学位論文

Critical-point-based Modeling for Smooth Surfaces

滑らかな曲面のための臨界点に基づくモデリング

平成8年12月博士(理学)申請

東京大学大学院理学系研究科
情報科学専攻

高橋成雄

学　位　論　文

Critical-point-based Modeling for Smooth Surfaces

滑らかな曲面のための臨界点に基づくモデリング

平成８年１２月博士（理学）申請

東京大学大学院理学系研究科
情報科学専攻

高橋 成雄

Critical-point-based Modeling for Smooth Surfaces

滑らかな曲面のための臨界点に基づくモデリング

by

Shigeo Takahashi

高橋 成雄

A Dissertation

Submitted to
the Graduate School of
the University of Tokyo
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Science
in Information Science

December, 1996

ABSTRACT

Recent advances in graphics hardware enable us to handle not only simple polyhedral shapes but complicated smooth shapes in CAD systems. Since the CAD systems represent such smooth surfaces by extending conventional simplicial representation, there are several problems in handling the smooth surface shapes that have no intuitive polyhedral approximations. The first problem is that the design of smooth surfaces requires a large amount of user interactions because of the complexity of the shapes. Secondly, none of the design operations characteristic of smooth surfaces are taken into account in contemporary CAD systems. Furthermore, they cannot provide the users with any efficient keys for shape databases due to the lack of information about smooth surface features. To remedy such problems, hierarchical representations of smooth surfaces based on the shape features are necessary.

This thesis presents a new feature-based modeling method for smooth surfaces. In particular, the aim of this thesis is to implement bidirectional operations between object shapes and shape features for smooth surfaces, i.e., design by features and feature extraction. As the shape features, critical points such as peaks, pits, and passes are used. The relations among the critical points are represented by the Reeb graph, which is one of the critical point graphs (CPGs). Within a theoretical framework, the smooth surfaces are assumed to be 2-dimensional C^2-differentiable manifolds. The features such as critical points and CPGs work at the upper levels in the hierarchical representations of smooth surfaces.

The shape design process begins with specifying the topological skeletons of an object shape using the Reeb graph. The Reeb graph is constructed by pasting the entities called cells that have one-to-one correspondences with the critical points of a height function. The iconic representation of the Reeb graph is used to visualize the embeddings of the object in 3-dimensional (3D) space. Macro operations are also provided for attaching a branch or a tube to an existing surface. The geometry of the smooth surface shape is outlined by flow curves that run on the object surface. From these flow curves, the system automatically creates a control network that encloses the object shape. Each vertex of the control network has its own local patch and the patches are then glued together using the manifold mappings in order to form the overall surface shape.

This thesis also introduces another hierarchical representation called multiresolution surface design that enables us to handle the detailed geometry of the local patches. In this design, the local patches are represented by endpoint-interpolating B-splines and their corresponding wavelets. The shape of the local patch is determined by minimizing the energy function subject to the deformation of the shape while preserving the given constraints. Constraints at a low resolution level are converted to those at a high resolution level using wavelet transforms in order to associate all the constraints with the common basis functions. The constraints of multiresolution levels are then solved recursively from low to high resolution levels.

The feature extraction from the polygonal representation of a surface shape, on the other hand, is implemented and used to change the height axis of the designed surface shape. Firstly, the critical points are extracted so that they satisfy the Euler formula which represents a topological invariant of smooth surfaces. The surface network, which is one of the CPGs, is then constructed by tracing ridge and ravine lines on the surface. An algorithm for converting the surface network to the Reeb graph is also presented. Using the obtained Reeb graph, the model of a control network is fit to the surface of the designed object in order to change its height axis.

This thesis also presents display examples generated in the system and describes the differences from the conventional shape modeling methods.

論文要旨

近年の計算機性能の向上に従い、形状設計システムで扱われる形状は簡単な多面体形状から複雑な曲面形状へ変化してきている。しかしながら、現在の形状設計システムでは、従来の多面体表現を拡張することにより曲面を表現しているため、滑らかな曲面を適当な多面体分割で表現することが難しく、いくつかの問題が生じてくる。まず第一に、曲面形状は複雑であるがため、設計の作業量が大きくなることがあげられる。第二には、曲面形状に固有の設計操作が全く考慮されていないことがある。さらには、システムが曲面形状データ・ベースのための効果的な検索キーを提供できない問題もある。これらは、すべてシステムが曲面の形状特徴を保持するレイヤを持たないために生じる問題である。この問題を解決するためには、形状特徴を上位階層として持つ曲面のための階層表現を作り上げる必要がある。

本論文では、滑らかな曲面のための形状特徴に基づくモデリング手法を示す。特に、本研究の目的として、物体形状と形状特徴の間の双方向の操作、つまり形状特徴による設計と形状からの特徴抽出を実現することを目指す。形状特徴としては、頂上、谷底、峠などの臨界点を用いる。それらの臨界点の関係を示すものとして、臨界点グラフのひとつであるレーブ・グラフを用いる。理論的側面として、滑らかな曲面は2次元 C^2 級可微分多様体であると仮定する。臨界点や臨界点グラフなどの形状特徴は、滑らかな曲面の階層表現における上位階層の役割を担う。

曲面形状の設計は、まず、レーブ・グラフを用いて形状の位相的な骨格を指定することから始まる。レーブ・グラフは、胞体と呼ばれる形状要素を貼り合わせることで指定され、その胞体は、高さ関数における曲面の臨界点に一対一に対応する。物体の3次元空間への埋め込みを表示するため、レーブ・グラフのアイコン表示を用いる。本研究の形状設計システムは、さらに分岐や筒を形状に追加するためのマクロ操作を提供する。滑らかな曲面の幾何形状は、曲面上を走る流れ曲線を指定することで設計がなされる。システムは、これらの流れ曲線から物体形状を覆う制御網を自動的に作り上げる。制御網の頂点はそれぞれ局所曲面を持っていて、それらの局所曲面が多様体写像により貼り合わされて全体の曲面を構成する。

本研究では、多重解像度曲面設計と呼ばれるさらなる階層表現を導入し、上記の局所曲面の細かい幾何形状の設計を支援する。この形状設計において、局所曲面は多重節点Bスプラインとそれに対応するウェーブレットによって表現される。局所曲面の形状は、与えられた幾何制約を満たしながら、形状の変形の度合を示すエネルギー関数を最小化することにより決定される。低解像度レベルの幾何制約は高解像度レベルの幾何制約にウェーブレット写像を用いることで変換され、すべての幾何制約が共通の基底関数によって表現されるようにする。多重解像度レベルに付加された幾何制約は低解像度レベルから高解像度レベルへ再帰的に解かれる。

今までの逆の操作として、本研究は曲面形状の多面体表現からの形状特徴抽出についても実現し、上記のシステムで設計された形状の高さ軸の変更操作に応用していく。まず、臨界点を滑らかな曲面の位相不変量を表す、オイラーの式を満たす形で抽出する。次に、臨界点グラフのひとつであるサーフェス・ネットワークを、曲面形状上の尾根線・谷線をたどることによって構築する。さらには、そのサーフェス・ネットワークをレーブ・グラフに変換するアルゴリズムについても示す。この抽出されたレーブ・グラフに基づき、制御網のモデルを設計された物体の曲面形状に合わせていき、高さ軸の変更を実現する。

本論文では、本システムで生成された例を示し、従来手法との違いについても言及する。

Acknowledgements

I wish to express my sincere gratitude to Prof. Tosiyasu L. Kunii of the president of the University of Aizu for his guidance in the early stage of this study. My sincere gratitude is also extended to Prof. Yoshihisa Shinagawa, my thesis advisor, for his continuous guidance and encouragement. The work in this thesis has its origin in his doctoral thesis [107] and has been done in collaboration with him.

I would like to thank Mr. Tetsuya Ikeda of Ricoh Co., Ltd. for his collaboration, Dr. Satoshi Nishimura of the University of Aizu for his constructive advice, Dr. Hitoshi Saji of Shizuoka University for fruitful discussions with him, and Mr. Masayuki Ohga, my research colleague, for his valuable comments on this study.

My special thanks are due to Prof. Anatoly T. Fomenko of the Moscow State University, Prof. Bianca Falcidieno and Dr. Michela Spagnuolo of C.N.R., Prof. Hiroshi Imai of the University of Tokyo, Prof. Issei Fujishiro of Ochanozimu University, Dr. Kenji Shimada and Dr. Hiroshi Masuda of IBM Tokyo Research Lab., Prof. Kenichi Kanatani and Dr. Naoya Ohta of Gunma University for their helpful comments and advice on this research. Mr. Yukio Sakagawa kindly read the early version of this thesis and gave suggestions for improving its expository style. My thanks also go to all members of the Shinagawa Laboratory for their help and encouragement.

Finally, I would like to thank my parents for their consideration, patience and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Recent developments in computer hardware enable us to handle a large amount of data within a short period of time. These developments have enabled us to handle not only simple polyhedral objects such as mechanical parts and manufactured objects, but complicated smooth objects such as terrains, human organs, and virtual objects (Figure 1.1). In particular, designing virtual objects such as humans, animals, plants, etc. has become important for computer graphics (CG) animation and virtual reality (VR) applications. In this way, the need to handle smooth curved objects by computers has been increasing.

Contemporary CAD systems handle the smooth object shapes, however, by extending conventional polyhedral representation. This leads to the polyhedral decomposition of smooth object shapes that has no relations with the geometric features of the smooth surfaces. In this situation, the contemporary CAD systems suffer from the following problems.

(1) Designing smooth object shapes requires a large amount of users' interactions because of inappropriate polyhedral approximations of complicated smooth shapes.

(2) Contemporary CAD systems do not take into account any of the design operations characteristics of the smooth surfaces.

(3) The CAD systems cannot provide the users with any efficient keys for shape databases due to the lack of information about the features of smooth surfaces.

In order to remedy these problems, it is necessary to construct a model for smooth surfaces based on shape features intrinsic to their smoothness.

1.2 Goal of This Thesis

The purpose of this study is to construct a hierarchical representation scheme for smooth surfaces where their shape features serve as the upper level of the hierarchical represen-

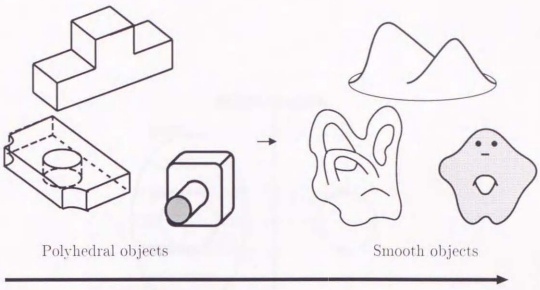


Figure 1.1: Object shapes handled in computers

tation. As the shape features, critical points such as peaks, pits, and passes are used in this study. In addition to the critical points, this study uses *critical point graphs (CPGs)*, which represents the critical points as its vertices and the relations among the critical points as its edges. Figure 1.2 illustrates the critical points and CPGs of a torus. In particular, the goal of this study is to provide bidirectional operations between object shapes and shape features, i.e., design by features and feature extraction as illustrated in Figure 1.3. These operations are helpful for avoiding the problems described in Section 1.1.

Let us see the relations between previous methods and the method of this thesis. Figure 1.4 illustrates the rough classification of previous modeling methods and this method based on the object shapes and representation schemes. The leftmost column corresponds to the modeling methods for polyhedral shapes and the rightmost column corresponds to those for smooth surfaces. The middle column indicates the modeling methods for the objects that contain both polyhedral and smooth surfaces. The bottom row corresponds to the hierarchical representation schemes while the top row corresponds to the representation schemes without explicit hierarchies. As described above, conventional modeling methods cover the smooth objects by extending the polyhedral representations. The CSG representation schemes are also extended to handle objects with free-form surfaces [130, 56, 78]. On the other hand, hierarchical representation schemes of polyhedral objects, such as constraint-based [37, 1, 99, 112] and feature-based [105, 63, 74] modeling methods, have been developed. Conversely, the method of this study directly handles the objects whose surfaces are smooth with hierarchical

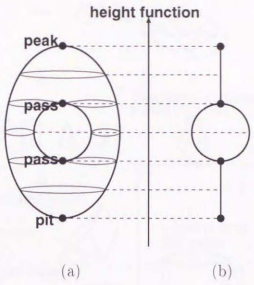


Figure 1.2: Features of a torus: (a) critical points and (b) the critical point graph (the Reeb graph)

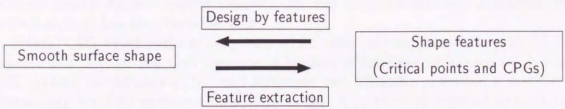


Figure 1.3: Bidirectional operations between object shapes and shape features

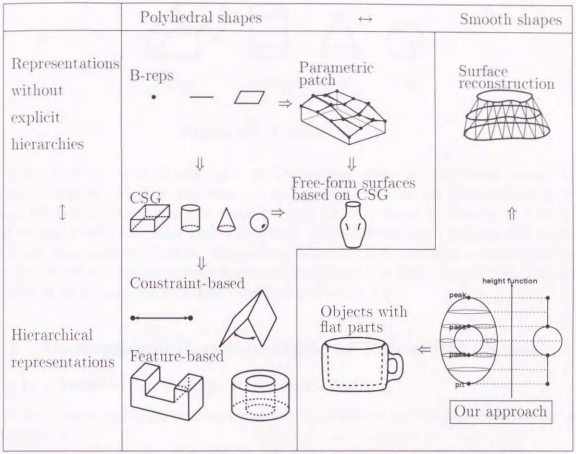


Figure 1.4: Classification of previous methods and the method of this thesis

representations based on features[1]. The method also covers the three-dimensional (3D) surface shapes reconstructed from cross-sectional data, and extends its target objects to smooth objects that contain flat surfaces partially as illustrated in Figure 1.4.

It must be noted that the objects handled in this modeling method are slightly different from those in the conventional modeling methods. While the conventional methods handle smooth objects indirectly by way of polyhedral approximation, this method aims at handling them directly.

Several CSG-based methods are presented for handling smooth surfaces [130, 56, 78]. In particular, Menon and Guo presented a method of handling sculptured solids using CSG boolean combinations [78], and Krishnan and Manocha presented a method of representing free-form surfaces by maintaining the connectivity of trimmed surfaces [56]. However, the methods cannot provide the operations based on the differential properties of smooth surfaces because they do not have the features of smooth surfaces.

Among the hierarchical representation schemes in Figure 1.4, the feature-based modeling methods have been extensively studied recently [104]. In the methods, object

[1] Cavendish proposed a method of designing and deforming the free-form surfaces with their features [14]. However, his method is limited to the surfaces represented by single-valued functions.

Figure 1.5: Primitives in CSG

features such as the slots and holes are used as the upper levels of their hierarchical representations. What is important to note is that they provide bidirectional operations between object shapes and shape features, i.e., the design by features and feature extraction. Users can design object shapes by their shape features, and can also extract features from existing objects. The goal of this study is to establish a smooth-surface version of the feature-based modeling method using critical points and CPGs. This will enable us to remedy the problems described in Section 1.1.

1.3 Conventional Representations of Polyhedral Shapes

1.3.1 Classification of Shape Representations

Let us review conventional representations of object shapes by following the classification of Figure 1.4.

- *Constructive solid geometry (CSG)*: CSG is a family of schemes for representing a solid object by boolean operations of simple primitive objects [98]. Figure 1.5 shows examples of such primitive objects. The CSG has various boolean operations such as union, difference, intersection, etc. As shown in Figure 1.6, the CAD system based on CSG has the shape data as a tree of boolean operations: its terminal nodes are either primitives or transformation data for rigid-body motions and its non-terminal nodes are either boolean operators or rigid-body motions that operate on their two subnodes. Hence, the CSG tree (based on boolean operations) is a candidate for an upper level of the hierarchical representation of an object.

- *Boundary representations (B-reps)*: B-reps hold the boundary data of a solid object and partition it into pieces called entities such as faces, edges, and vertices. In B-rep systems, boolean operations are used to design object shapes similarly to those in CSG systems. Figure 1.7 shows the entities of the B-reps. The representation is equivalent to the Hasse diagram [113], which represents the partial ordered set where the partial ordering is defined on the inclusion relations among the entities. Consider the Hasse diagram on the right of Figure 1.8, for example. Since the edge e_1 contains the vertex v_2 as shown in Figure 1.8, the partial order between e_1 and v_2 can be written as $v_2 \prec e_1$, where $\prec$ indicates the partial order. In this case, the nodes of e_1 and v_2 are connected by an edge as shown in the right side of Figure 1.8. The partial order between a face and an edge can be defined in

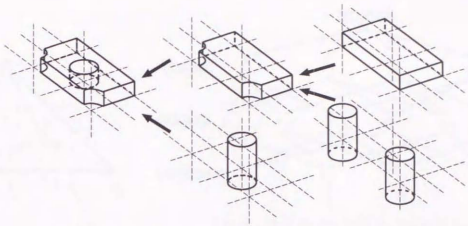


Figure 1.6: Boolean operations in CSG

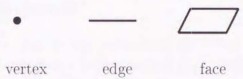


Figure 1.7: The entities of B-reps

the same way. As shown in Figure 1.8, the topology of the entities is represented by the Hasse diagram while the geometry is associated with the coordinates of the vertices.

The main advantage of the B-reps lies in easy access to the boundary data of a solid object, namely faces, edges, vertices, and the relations among them. This is important because it is necessary to generate line drawings and rendered images in graphic displays. Furthermore, the B-reps are convenient for approximating complex shapes such as smooth surfaces. Although the B-rep schemes require more data storage than CSG, they are used extensively in current systems because of the above advantages.

- *Hybrid representations*: Hybrid representations are the combination of CSG and B-reps [131, 11]. One of the advantages of the hybrid modeler is that the modeler tries to pick the most suitable representation for each task. In general, an object shape is first kept as a CSG representation and is then converted to a boundary representation when it is necessary.

Most contemporary CAD systems are based on either B-reps or hybrid representations, which means that the B-reps play a fundamental role in contemporary CAD systems. Such CAD systems generally support the representation schemes based on the

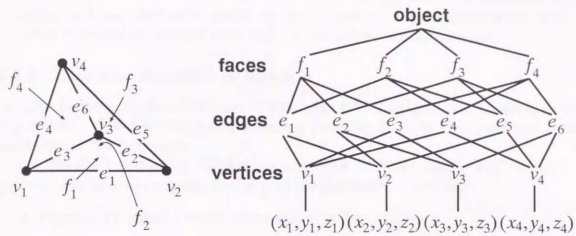


Figure 1.8: Data representation in B-rep systems

procedural operations called sweeping.

- *Sweep representations*: Sweep representations are based on the notion of moving a 1-dimensional (1D) curve or a 2-dimensional (2D) surface along a path called a trajectory [53, 22, 19, 119]. This representation scheme is simple to understand and used successfully for surface design in contemporary CAD systems. The domain of the representation scheme is, however, smaller than those of CSG and B-reps. In practice, sweeping is used as one of the design operations and the object shapes designed using the sweeping operations are usually converted to the corresponding boundary representations.

According to Requicha's survey [97], there are volume-based representation schemes for solid objects as follows.

- *Spatial occupancy enumeration*: The representation of spatial occupancy enumeration is a list of spatial cells occupied by a solid object. The cells are usually cubes of fixed size lying in a regular spatial grid and are called *voxels*. Although it is easy to handle the representations of spatial occupancy enumeration, it requires considerably large and redundant data storage. To avoid this inefficiency, the hierarchical representations of the voxels called octrees are developed [77, 34, 101].

- *Cell decomposition*: Cell decomposition is similar to spatial occupancy enumeration in that a solid object is decomposed into simpler primitives than the original object. The primitives are called *cells*. There are many ways of decomposing the solid object into cells; the selection of the way is dependent on the shape features of the object. In general, a polyhedral shape such as a tetrahedron is used as a cell (for example, as shown in [65]).

Although these volume-based representations are not popular in contemporary shape modeling systems, they were proven to be efficient for several applications such as scientific visualization, volume rendering, virtual surgery simulation, etc.

1.3.2 Data Representation in B-reps

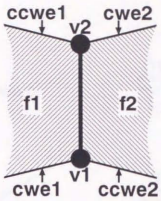
As seen in the previous subsection, B-reps play a fundamental role in shape modeling systems. The following two subsections describe the data structures and design operations in B-rep systems.

As described previously, the entities of B-reps are faces, edges, and vertices. In practical implementations, the B-reps hold the following six entities.

- *Objects*: An object consists of the following five primitives.
- *Shells*: A shell is an entity that confines a space.
- *Faces*: A face represents the boundary of an object. Faces that enclose a space constitute a shell.
- *Loops*: A loop is a boundary of a face. A face has at least one loop and additional loops if the face has holes in its interior.
- *Edges*: An edge is an intersection of the boundaries of two faces.
- *Vertices*: A vertex is a point at which several edges meet.

In B-rep systems, the combinatorial relations between these six entities are called *topology*, and the metric data such as coordinates of vertices is called *geometry* [4]. The topology is invariant under geometric transformations while the geometry is not. Since the B-rep system has the boundary data of object shapes directly, it is easy to generate graphic outputs on the display. However, some care should be taken in building the data structure of the B-reps because the number of edges that surround a face varies according to the shape of the face. Baumgart introduced the edge-oriented data structure called the *winged-edge structure* [5], where each component has a fixed size. The efficiency of this representation follows from the fact that an edge has two connected vertices and two touching faces without exceptions as shown in Figure 1.9. While the winged-edge structure is efficient, it still suffers from the following problem: If an object shape has some edge whose endpoints are identical with each other like a loop, the system cannot retrieve its correct topology from the winged-edge structure. The *half-edge structure* avoids this problem by decomposing an edge into left and right half edges [133, 71]. The efficiency in data access using this data structure is discussed in detail by Weiler [133] and Woo [138].

The above discussions on the B-reps are based on the assumption that an object shape is in the class of two-manifolds. A *two-manifold* is defined as an object bounded by a compact (closed) orientable 2-dimensional manifold surface, where the neighborhood of each point of the surface is topologically equivalent to an open disk. The objects



Edge

v1	v2
cwe1	cwe2
ccwe1	ccwe2
f1	f2

Figure 1.9: The winged-edge structure

assumed in this thesis are also contained in this class. An object included in the complementary set of two-manifolds is called a *non-manifold*. Unfortunately, the class of two-manifolds is not closed with respect to boolean operations such as union, difference, intersection, etc. Requicha introduced the class called *r-sets* [97, 126], which are closed with respect to the *regularized boolean operations* extended from the ordinary boolean operations. Extending the B-reps of two-manifolds to those of r-sets was studied in detail by Weiler [134].

1.3.3 Euler Operators

There exists the invariance theorem called the *Euler formula* that plays an important role in B-reps. The ordinary formula is expressed by

$$\#\{vertices\} - \#\{edges\} + \#\{faces\} = \chi, \tag{1.1}$$

where $\#\{vertices\}$, $\#\{edges\}$, and $\#\{faces\}$ represent the numbers of vertices, edges, and faces, respectively. In the above formula, χ is a topological invariant of polyhedral shapes and is called the *Euler characteristic*.

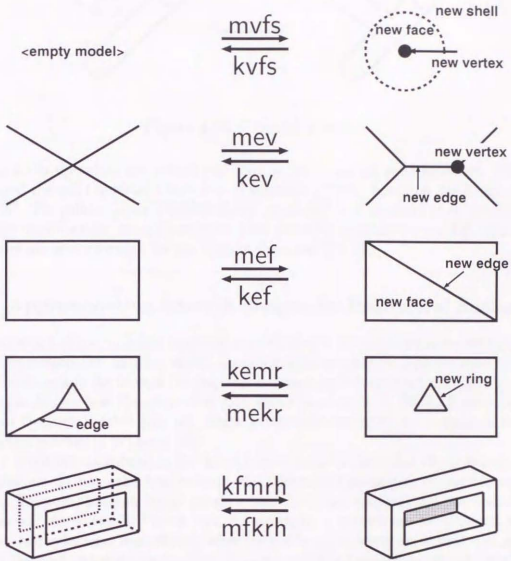
This formula is extended for the use in B-reps, which is written as

$$\#\{vertices\} - \#\{edges\} + \#\{faces\} = 2(\#\{shells\} - \#\{holes\}) + \#\{rings\}. \tag{1.2}$$

The way of extending (1.1) to (1.2) is described in [71].

The *Euler operators* were introduced by Baumgart [6] as the means of preserving the above formula while modifying the shape data in B-reps[2]. Figure 1.10 lists the basic Euler operators [72]. The Euler operators hide the low-level implementation of the data structure in B-reps and provide an efficient and also topologically verified way of handling the shape data.

[2] Wilson proved that the Euler operators can be applied not only to solid modelers but to the systems based on wireframe representations [136].



Euler Operator	Notation	Euler Operator	Notation
mvfs	make vertex face shell	kvfs	kill vertex face shell
mev	make edge vertex	kev	kill edge vertex
mef	make edge face	kef	kill edge face
kemr	kill edge make ring	mekr	make edge kill ring
kfmrh	kill face make ring hole	mfkrh	make face kill ring hole

Figure 1.10: The Euler operators

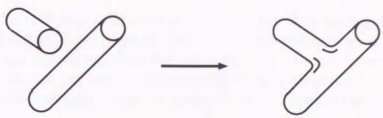


Figure 1.11: Creating a branch

The Euler operators are extensively used in contemporary solid modelers. Mäntylä developed the solid modeler *Geometric Workbench (GWB)* based on the Euler operators [72][3]. The solid modeler *DESIGNBASE* developed by Chiyokura et al. manipulates all shape modifications using a sequence of the extended Euler operators [17]. The Euler operators are also extended for the class of the r-sets [23, 28].

1.4 Approximating Smooth Shapes by Polyhedral Surfaces

Contemporary shape modelers represent smooth shapes by extending conventional polyhedral representation. In other words, smooth shapes are roughly approximated by polyhedral surfaces and the faces of the polyhedral shapes are interpolated smoothly by parametric patches such as Coons patches [21], Bézier patches [8, 9], NURBS patches [125, 91], and Gregory patches [18, 16]. Farin extensively surveyed the techniques for the parametric patches in his book [30].

The problems mentioned in Section 1.1 arise from the fact that the polyhedral approximations do not take into account the differential properties of smooth surface shapes. Suppose that you try to create a branch by the conventional polyhedral representation. As shown in Figure 1.11, for example, a branch can be obtained as the union of two cylinders. Immediately after the union operation is performed, the surface around the branch has discontinuities. In order to make the surface smooth, it is often necessary to reorganize the polyhedral decomposition of the shape. However, the system cannot provide the operations that automatically generate the smooth surface since it does not hold the information about the differential properties of smooth surfaces.

Another problem of the polyhedral approximations is the difficulties in connecting patches while preserving the continuity on the boundaries. It requires careful handling to avoid this problem because smooth surfaces are decomposed into polyhedral pieces that have no relationships with the features of smooth surfaces. In this study, manifold-based patch blending is employed in order to avoid this problem, which is described in Chapter 4.

[3] Mäntylä also proposed an inversion algorithm that reduces the boundary of an object shape to a sequence of the Euler operators [70].

1.5 Hierarchical Representations of Polyhedral Objects

Since the entities of B-reps such as vertices, edges, and faces are low-level representations, it is hard to find global shape features from such representations. As illustrated in Figure 1.4, the hierarchical representation schemes for polyhedral shapes have been studied recently. There are two important categories for such hierarchical representations of polyhedral shapes: constraint-based representations and feature-based representations.

- *Constraint-based representations*: Constraint-based representations [37, 54, 99, 112] are effective in that geometric constraints are used as the upper levels of the hierarchical representations. Here, the geometric constraints contain the distance between two vertices, the angle between faces, and so forth. A designer provides the system with geometric constraints as an input, and the system solves the given constraints to find an appropriate solution. This approach, however, has ambiguity in its representation: an object shape has no unique sequence of constraints and resultant object shapes are dependent on the order of the imposed constraints.

- *Feature-based representations*: Feature-based representations [27, 74, 128] use geometric features, such as slots and holes, as the upper levels of their hierarchical representations. As mentioned earlier, these representations provide bidirectional operations between object shapes and shape features. In other words, the design by features and feature extraction are integrated to provide users with efficient design methods. One of the problems of the feature-based approach is that the features such as slots and holes have no mathematical background while they are easy to use.

In this study, bidirectional operations similar to those in the feature-based representations are implemented especially for smooth surfaces. Furthermore, the features used in this study do not suffer from the above problems because they are derived from the mathematical properties of smooth surfaces.

1.6 New Feature-based Approach for Smooth Surfaces

This thesis presents new feature-based modeling techniques for smooth surfaces. In particular, the goal of this thesis is to implement bidirectional operations between object shapes and shape features, i.e., design by features and feature extraction. As the shape features, critical points such as peaks, pits, and passes are used. The relations among the critical points are represented by the Reeb graph [96], which is one of the critical point graphs (CPGs) (Figure 1.2). The smooth surfaces are supposed to be a 2-dimensional C^2-differentiable manifold within a theoretical framework. The features such as critical points and CPGs work at the upper levels in the hierarchical representations of smooth surfaces.

Firstly, this thesis presents a method of designing the topological skeletons of an object shape using the Reeb graph [108, 107]. The Reeb graph is constructed by pasting the primitives called cells that have one-to-one correspondences with the critical points

of a height function. The Reeb graph is edited by the *Morse operators* that specify the way of gluing cells. The iconic representation of the Reeb graph is used to visualize inclusion relations of cross sections at a height value. The system also provides macro operations for attaching a branch or a tube to the constructed surface using a pair of the Morse operators [123].

Secondly, this thesis proposes the geometric design method based on the above topological design [123]. The geometry of the smooth surface shape is outlined by flow curves that run on the object surface. From these flow curves, the system automatically creates a control network that encloses the object shape. Local patches are assigned to the vertices of the control network and are then glued together to form the whole surface shape similarly to the manifold construction. The local patches are mapped to the whole surface with overlaps where the patches are interpolated smoothly.

Thirdly, this thesis introduces a method of designing the local patches using multiresolution constraints [122]. The shapes are represented by endpoint-interpolating B-splines and their corresponding wavelets. At each resolution level, the shape is determined by minimizing the energy function subject to the deformation of the shape while preserving the given constraints. Constraints at a low resolution level are converted to those at a high resolution level using wavelet transforms in order to associate all the constraints with the common basis functions. The constraints at multiresolution levels are then solved recursively from low to high resolution levels.

This thesis also describes algorithms for extracting the shape features from the polygonal representation of a surface shape [120], which are used to change the height axis of the designed object. The first step is to extract the critical points that satisfy the Euler formula (cf. Section 1.7). The surface network, which is one of the CPGs, is then constructed by tracing ridge and ravine lines on the object surface. An algorithm for converting the surface network to the Reeb graph is also presented. Using the extracted Reeb graph, the system fits the model of a control network to the polygonal surface shape.

1.7 B-reps and the Proposed Representation

It is noted that critical points have close relations with the entities of B-reps. Figure 1.12 illustrates the relations. The entities of B-reps are *simplices*, i.e., faces, edges, and vertices. Mathematical theories tell us that the simplices have their surface-versions called the *cells*[4] that are the entities of a smooth object in this study[5]. Furthermore, according to the Morse theory [80], the cell has a one-to-one correspondence with the critical point of a height function of the object surface. Consequently, the entities of B-reps correspond to the features of smooth surfaces, i.e., the critical points.

These relations enable us to apply the framework of B-reps to this modeling method. The first and the most important example is the *Euler formula* that maintains the

[4] The cells are not new to current shape modeling techniques. Several papers propose methods of representing non-manifold objects using the topological properties of the cells [76, 142] Note that the word "cell" here is used in a different way from that of cell decomposition.

[5] The description of the theories can be found, for example, in [75].

	B-reps	the proposed representation
entities	simplices  faces edges, and vertices	cells $\xrightarrow{\text{Morse theory}}$ critical points  peaks, passes, and pits
invariants	#(faces) − #(edges)+ #(vertices) = χ	#(peaks) − #(passes)+ #(pits) = χ
operators	Euler operators	Morse operators

Figure 1.12: Relations between the entities of B-reps and of the proposed representation

validity of object shapes in B-reps. From the above relation, the Euler formula of a polyhedral shape (1.1) can be converted to that of a smooth surface[6]:

$$\#\{\text{peaks}\} - \#\{\text{passes}\} + \#\{\text{pits}\} = \chi. \tag{1.3}$$

To maintain the Euler formula in B-reps, the Euler operators are used. In the representation of this thesis, on the other hand, a set of operators called the *Morse operators* [108, 107] are used.

1.8 Organization of This Thesis

The overview of this study is illustrated in Figure 1.13. This thesis is organized along with the flow chart of this overview. While Chapters 2, 3, and 4 describe one of the bidirectional operations: design by features, Chapter 5 describes the other operation: feature extraction.

Chapter 2 describes how to design the topological skeletons of smooth objects using the iconic representation of the Reeb graph. Fundamental theories are also explained in this chapter. Chapter 3 presents the techniques for designing geometric shapes with control networks and manifold mappings. Chapter 4 introduces the method of designing surfaces using multiresolution constraints. Contrary to Chapters 2, 3, and 4, Chapter 5 proposes algorithms for extracting critical points and CPGs from the polygonal representation of a surface shape. Chapter 6 discusses the difference between the proposed modeling method and the conventional modeling methods. Chapter 7 concludes this thesis and refers to future work.

[6] This formula is also called the mountaineer's equation. The description of the mountaineer's equation can be found, for example, in [39].

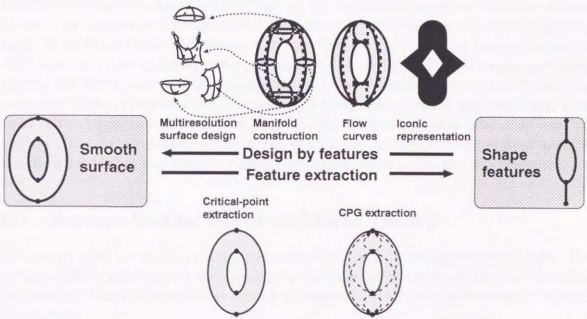


Figure 1.13: The system overview

Chapter 2

Designing the Topological Skeletons

This chapter explains a method of designing the topological skeletons of object shapes. In order to design the topological skeletons, one of the CPGs called the Reeb graph is used. In addition to the skeletons, the embeddings of the objects in three-dimensional (3D) space are also specified using an iconic representation called the embedded Reeb graph. The Reeb graph is edited using the Morse operators that correspond to the critical points of the object surface. The system provides macro operations that manipulate pairs of the Morse operators so that users can attach a branch or a tube to an existing surface. The data representations of the topological skeletons in the system and the interface for editing the iconic representation of the Reeb graph are also presented.

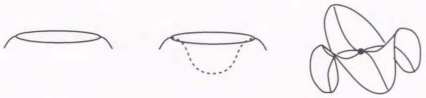
2.1 Surface Coding Based on Morse Theory

This study uses the surface coding techniques proposed by Shinagawa et al [108]. The coding techniques describe the topological features of an object shape by extending the classical Morse theory. This section explains the fundamental framework of these techniques.

2.1.1 Limits of the Theory

The classical Morse theory assumes that the object surface is a C^2 smooth compact closed manifold. With this assumption, the surface can be decomposed into pieces called cells by scanning the critical points of the object along a height direction. As described in Chapter 1, the cells and the critical points have one-to-one correspondences. In addition, they work as the entities in this representation scheme. For example, a torus can be decomposed into 4 pieces because it has four critical points as shown in Figure 1.2.

Morse theory is valid only if none of the critical points of the object surface are degenerate. Here, the degenerate critical points contain a flat peak, a flat rim, and a monkey saddle as illustrated in Figure 2.1. Note that the following descriptions are based on the assumption that none of the critical points are degenerate. Such degenerate critical points are handled as exceptional cases, which is described in detail in Section



(a) (b) (c)

Figure 2.1: Examples of degenerate critical points: (a) a flat peak, (b) a flat rim, and (c) a monkey saddle

3.4 for the shape design and Section 5.2 for the critical-point extraction. Appendix A provides the detailed descriptions of the Morse theory including the definitions of a critical point, its degeneracy, and so forth.

While the classical Morse theory provides us with the means of efficiently describing the object shapes, it cannot distinguish between their embeddings in several cases while they are different. For example, the theory does not code the difference of connectivity between the objects of Figure 2.2(a), knots between the objects of Figure 2.2(b), and links between the objects of Figure 2.2(c). For the difference of connectivity, the techniques of Shinagawa et al. can recognize it using the Reeb graph. However, the other differences require the mathematical theories of knots and links, which are still at the stage of rapid development.

2.1.2 The Reeb Graph

The Reeb graph serves as a tool for representing the topological skeletons of an object shape in this study. The *Reeb graph* [96] is defined as follows. Let f be a function of an object surface. Consider the cross-sectional contours of the object surface, i.e., $f^{-1}(h)$ (h = const). By identifying the connected component of the object surface with a point at each cross section, a set of points can be obtained. The Reeb graph is then constructed by tracing these points at each cross section from the top to the bottom of the object. Figure 1.2 illustrates the Reeb graph of a height function defined on a torus. Here, the height function represents the points (x, y, z) on the surface as the function $z = f(x, y)$. Appendix B describes the mathematical definition of the Reeb graph [107].

As mentioned earlier, the Reeb graph is one of the CPGs and its vertex corresponds to a critical point. From the above definition, the edge of the Reeb graph represents a tube that connects two critical sections. As can be seen in Figure 1.2, the Reeb graph represents the topological skeletons of the object shape. Since this skeletal representation is intuitive, the Reeb graph is used to design the topological skeletons of the object shape in this study.

In addition to the topological skeletons, the embeddings of the object in 3D space are also specified. For this purpose, this study employs the *embedded Reeb graph* proposed by Shinagawa et al [108]. The embedded Reeb graph is an iconic graph that represents

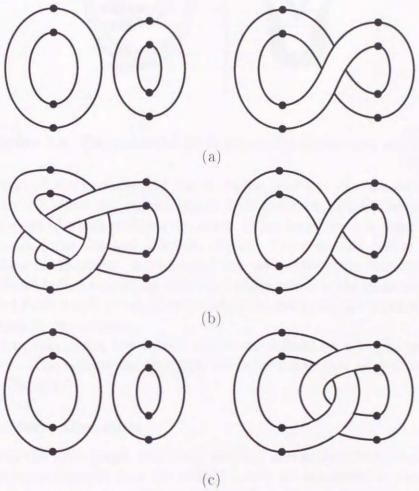


Figure 2.2: Equivalent objects in the classical Morse theory: differences in (a) connectivity, (b) knots, and (c) links [108]

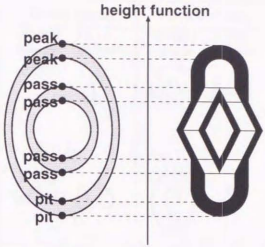


Figure 2.3: The embedded Reeb graph of a double-layered torus

the orientations of the surfaces and the inclusion relations of cross-sectional surface profiles. Figure 2.3 illustrates the embedded Reeb graph of a double-layered torus, i.e., two tori where one of which contains the other. If the outer torus is regarded as a solid object, the inner torus becomes a hollow object. The solid and hollow objects have different surface orientations. In the same way as in [108], the system distinguishes between solid and hollow objects by black and white colors in the iconic representation. The embedded Reeb graph serves as an interface for designing the topological skeletons of object shapes in the system.

Note that in this thesis, the critical points are defined on a height function of the object surface. The scheme for changing the height direction of designed objects is described in Chapter 5.

2.1.3 The Morse Operators

In order to edit the Reeb graph, this study uses the *Morse operators* proposed in [108]. The Morse operators specify how the critical points are connected in the Reeb graph, which is equivalent to how the cells are glued to form the whole object.

The Reeb graph of a torus is constructed with the Morse operators as follows. As illustrated in Figure 1.2, a torus contains, from top to bottom, four critical points: a peak, a pass, a pass, and a pit. This means that a torus is constructed from its top to bottom with the four Morse operators that correspond to the critical points of the torus as illustrated in Figure 2.4[1]. Firstly, the peak of the torus is created by putting a 2D cell e^2, i.e., a topological cap (Figure 2.4(a)). With this operation, a new circle appears at the cross section. Secondly, a 1D cell e^1, i.e., a topological curve, is attached in order to split the cross-sectional circle (Figure 2.4(b)). Thirdly, another 1D cell e^1 is attached

[1] This construction is depicted in detail in [80], and the algorithm animation of this construction is presented in [58].

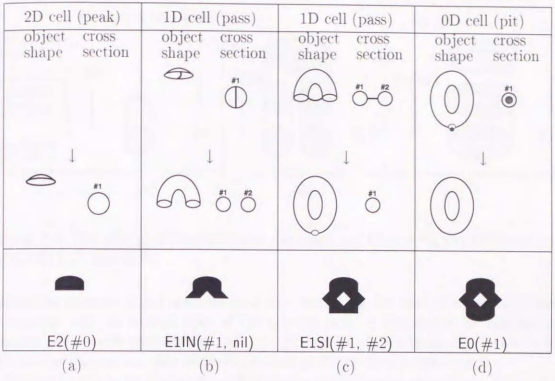


Figure 2.4: The Morse operators for constructing a torus: (a) putting a 2D cell e^2, (b) attaching a 1D cell e^1, (c) attaching a 1D cell e^1, and (d) attaching a 0D cell e^0 [108]

in order to merge the two divided cross-sectional circles into one, which results in the hole of the torus (Figure 2.4(c)). Finally, the surface of the torus is closed by attaching a 0D cell e^0, i.e., a point, to the existing surface (Figure 2.4(d)). The changes of the embedded Reeb graph (the iconic representation of the Reeb graph) are also illustrated at the bottom row of Figure 2.4.

In this way, the Morse operators describe the changes of cross-sectional contours at critical sections. In the implementation of this study, the system stores the inclusion

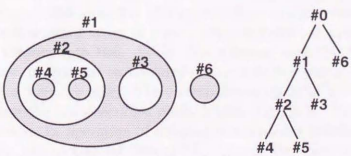


Figure 2.5: A contour tree based on inclusion relations [108]

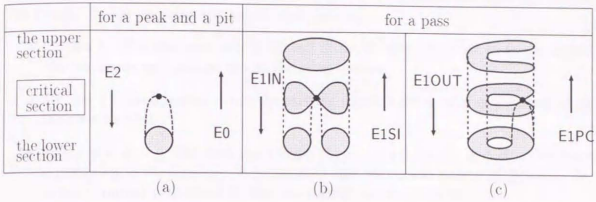


Figure 2.6: The effects of the six Morse operators: (a) E2 and E0, (b) E1IN and E1SI, and (c) E1OUT and E1PC

relations of cross-sectional contours as a tree, which can be seen in Figure 2.5. Here, the contour #0, the virtual root of the contour tree, is introduced for convenience. Consider the relation between the contours #2 and #4, for example. Since the contour #2 contains the contour #4, #2 is the parent of #4 in the contour tree.

Similar to the Euler operators in B-reps, the Morse operators maintain the validity of smooth object shapes. Here, the validity of object shapes means that they can be embedded in 3D space without self-intersections. To maintain the validity of the object shapes, six types of operators are required as described in [108]. The changes in cross-sectional contours by these six operators are shown in Figure 2.6.

In Figure 2.6(a), the downward arrow indicates the change of a peak, while the upward arrow indicates that of a pit. In this thesis, the corresponding operators are denoted by E2 (which is named after e^2) and E0 (which is named after e^0), respectively. In the case of a pass, there are four operators as illustrated in Figure 2.6(b) and (c). The downward arrow of Figure 2.6(b) indicates an operation of splitting one cross-sectional contour into two. This operator is denoted by E1IN (e^1-inside) because the cross-sectional circle is split inside it. The reverse arrow of Figure 2.6(b) indicates an operation of merging two different contours into one. Since the two contours are siblings in the contour tree (cf. Figure 2.5), the corresponding operation is denoted by E1SI (e^1-siblings). The situation of Figure 2.6(c) is more complicated than the previous one. The downward arrow of Figure 2.6(c) indicates an operation of splitting a cross-sectional contour into two. While this operator is similar to E1IN, it differs from E1IN, however, in that the two divided contours have a parent-child relationship, i.e., one contour contains the other. The corresponding operator is denoted by E1OUT (e^1-outside) because the cross-sectional circle is split outside it. The reverse arrow of Figure 2.6(c) indicates an operation of merging two contours that have a parent-child relationship in the contour tree (cf. Figure 2.5). Hence, the corresponding operator is denoted by E1PC (e^1-parent-child).

The Morse operators proposed by Shinagawa et al. also allow us to describe the embeddings of the cross-sectional contours. In order to describe such embeddings, the

Morse operators take contour numbers or lists of contour numbers as their arguments. The details of each operator are described as follows.

- E2($\#n$): This operator puts a 2D cell e^2 in 3D space in order to create a new contour inside the contour $\#n$ at the cross section.
- E0($\#n$): This operator closes the existing contour $\#n$ by gluing a 0D cell e^0 at the cross section.
- E1PC($\#n$, $\#m$): This operator merges the two contours $\#n$ and $\#m$ that have a parent-child relationship. It is assumed that $\#n$ is the parent of $\#m$ and the merged contour is denoted by $\#n$, the parent contour number.
- E1SI($\#n$, $\#m$): This operator merges the two contours $\#n$ and $\#m$ that are siblings in the contour tree. It is assumed that the merged contour is denoted by $\#n$, the first argument of the operator.
- E1IN($\#n$, $(\#a, \#b, \ldots)$): This operator splits the contour $\#n$ inside it in order to create a new contour at the cross section. The list of contour numbers $(\#a, \#b, \ldots)$ represents the contours contained in the newly created contour after this operation. If the list does not contain any contours, it is denoted by nil.
- E1OUT($\#n$, $(\#a, \#b, \ldots)$): This operator splits the contour $\#n$ outside it in order to create a new contour at the cross section. Recall that the newly created contour is contained in $\#n$ here. In the same way as for E1IN, the list of contour numbers $(\#a, \#b, \ldots)$ represents the contours contained in the newly created contour after this operation.

Although this notation is slightly different from the original ones proposed by Shinagawa et al., the operations are exactly the same as theirs. According to the above notation of the Morse operators, a torus is constructed by the sequence of the four Morse operators: E2(#0), E1IN(#1, nil), E1SI(#1, #2), and E0(#1) as shown at the bottom of Figure 2.4.

In the implementation of this thesis, these six operators are provided as fundamental tools for editing the Reeb graph. Since the system holds the contour trees as illustrated in Figure 2.5, it automatically rejects illegal Morse operators that result in generating invalid objects[2]. This will be described in Section 2.2 in detail.

2.1.4 Macro Operations

With the Morse operators, the user can edit the Reeb graph that represents the topological skeletons of a smooth object. The order of these operations, however, must follow the height order of their corresponding critical points. This means that it is necessary to describe the changes of cross-sectional contours from the top to the bottom of the

[2] This means that this model does not permit objects that cannot be embedded in 3D space. The Klein bottles are examples of such invalid objects. The model extended for such objects is described in [110].

object shape the Reeb graph object shape the Reeb graph

(a) (b)

Figure 2.7: Macro operations: (a) an operation for attaching a new branch, and (b) an operation for attaching a new tube

object. Since this limitation is not intuitive to design a whole object, it should be avoided.

This study provides macro operations for avoiding such limitations [123]. The *macro operation* inserts a pair of new critical points to an existing object shape while maintaining the topological validity of the object shape. Note that the topological validity of the object shape can be examined also in this case because the macro operation is equivalent to applying a pair of the Morse operators. Figure 2.7 illustrates examples of macro operations. Figure 2.7(a) shows a macro operation for attaching a new branch to a topological sphere. This macro operation is equivalent to applying the E2 and E1SI operations to the existing surface, which means that a peak and a pass are inserted. As illustrated in Figure 2.7(a), the Reeb graph is also modified with the macro operations. Another example of attaching a new tube to a topological sphere is shown in Figure 2.7(b). This macro operation amounts to applying the E1IN and E1SI operations to the existing surface. Note that the possible set of such macro operations is E2–E0, E2–E1SI, E2–E1PC, E1IN–E0, E1IN–E1SI, E1IN–E1PC, E1OUT–E0, E1OUT–E1SI, and E1OUT–E1PC. More complex operations can be implemented by combining these macro operations. These macro operations allow us to overcome the limitation due to the height order of the Morse operations.

2.2 Data Representation in the System

The reminder of this chapter describes the implementation of the system for designing the topological skeletons of smooth objects.

2.2.1 Representing the Reeb Graph and Its Embeddings

This subsection describes the representation of object shapes in the system. As described earlier, the system holds the Reeb graph for representing the critical points and their connections, and the contour trees for representing the inclusion relations among cross-sectional contours. In this thesis, cross sections that contain critical points are called *critical sections*, and other sections are called *regular sections*. When the cross sections of the object shapes are scanned from its top to bottom, the contour tree changes at

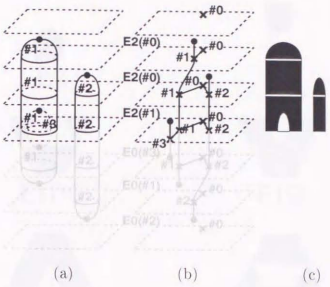


Figure 2.8: Data representation in the system: (a) the object shape, (b) graph data, and (c) iconic representation

the critical sections. Therefore, it is necessary and also sufficient to hold the contour trees at regular sections each of which lies between a pair of adjacent critical sections.

Let us see how the system holds the topological skeletons and its embeddings. Suppose there is an object shape as shown in Figure 2.8(a). The representations of its topological skeletons and embeddings are shown in Figure 2.8(b). As seen in this figure, the system holds the topological skeletons of the object shape as a graph. In addition to the topological skeletons, the system represents its embeddings as the contour trees at regular sections between adjacent critical sections. The planes of Figure 2.8 represent such regular sections. Note that the virtual contour #0 is introduced for representing the root of the contour tree in the system.

Furthermore, the system maintains the topological consistency of the object shape with this data structure. In order to implement this consistency, the system automatically constructs the closed surface by adding appropriate virtual pits to the object shape that is currently designed. Since the Morse theory is based on the assumption that the object consists of closed surfaces, this framework is convenient for verifying the topological consistency of the object shape. In Figure 2.8(b), the lower parts with the light black color represent the virtual parts automatically added by the system, while others represent the existing parts.

In the implementation, each vertex of the Reeb graph has its corresponding Morse operator. This allows us to check whether the embeddings of the designed shape are correct or not. To confirm the valid embeddings of the object shape, the system updates the contour trees at regular sections by scanning the Morse operators from the top to the bottom of the object. If the newly designed object is not topologically correct, the system rejects the latest operation. The contour trees at regular sections are updated

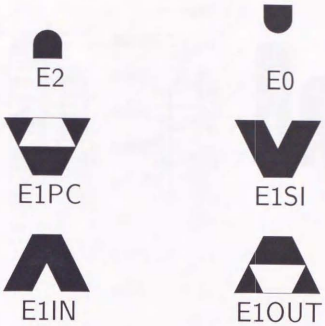


Figure 2.9: Iconic primitives used in the system

whenever the object shape is modified by users.

The graph data plays a fundamental role in representing object shapes. Additional information such as geometry of the object shapes will be also added to this graph data, which will be described in Chapter 3. As seen in Section 2.1, the system uses the iconic representation of the Reeb graph that visualizes the embeddings of object surfaces. In particular, it is helpful for visualizing the inclusion relations among cross-sectional contours at regular sections. Figure 2.9 shows the primitives of the iconic representation used in the system [108]. As described in Section 2.1.2, the *black icons* represent solid contours and the *white icons* represent hollow contours.

The iconic outputs are generated by converting the graph data using the mapping defined in the system. For example, as shown in Figure 2.8, the iconic representation (Figure 2.8(c)) of the object shape (Figure 2.8(a)) is converted from the graph data (Figure 2.8(b)).

2.2.2 Modifying the Object Data Using Morse Operators

The fundamental tools for modifying the graph data are the Morse operators described in Section 2.1.3. Let us consider how the Morse operators can be used to modify the graph data in the system. Since the macro operations can be reduced to the sequence of the Morse operations, it is sufficient to see how the six types of Morse operators modify the graph data in the system. In what follows, the six types of the Morse operators will be applied to the object shown in Figure 2.8.

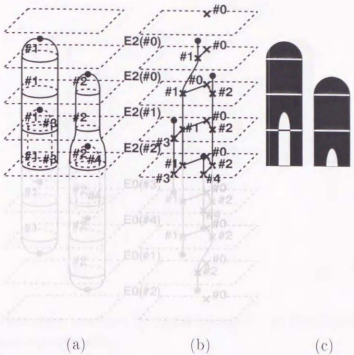


Figure 2.10: Object data modified by an E2 operator: (a) the object shape, (b) graph data, and (c) iconic representation

Applying an E2 operator

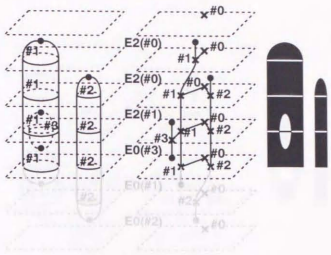
When a 2D cell e^2 is put inside the contour #2, the object shape, graph data, and iconic representation are modified as shown in Figure 2.10. This amounts to applying the operator E2(#2). An edge of the Reeb graph is inserted in order to represent the newly created contour #4. The system assigns the label of the Morse operator E2(#2) to the upper endpoint of the edge and the label of a virtual pit to its lower endpoint. The system also scans from top to bottom the sequence of the Morse operators and updates the contour trees at regular sections.

Applying an E0 operator

When a 0D cell e^0 is attached to the contour #3, the object shape, graph data, and iconic representation are modified as shown in Figure 2.11. This amounts to applying the operator E0(#3). In this case, the system only changes the pit of the contour #3 from a virtual pit to a real one.

Applying an E1PC operator

When a 1D cell e^1 is attached to the contours #1 and #3, the object shape, graph data, and iconic representation are modified as shown in Figure 2.12. This amounts to applying the operator E1PC(#1, #3). The system deletes the virtual pit of the contour



(a) (b) (c)

Figure 2.11: Object data modified by an E0 operator: (a) the object shape, (b) graph data, and (c) iconic representation

#3 and merges the edges of the contours #1 and #3 by inserting a new branch vertex that has the operator E1PC(#1, #3).

Applying an E1SI operator

When a 1D cell e^1 is attached to the contours #1 and #2, the object shape, graph data, and iconic representation are modified as shown in Figure 2.12. This amounts to applying the operator E1SI(#1, #2). The system deletes the virtual pit of the contour #2 and merges the edges of #1 and #2 by inserting a new branch vertex that has the operator E1SI(#1, #2).

Applying an E1IN operator

When a 1D cell e^1 that splits the contour #1 inside it is attached to #1, the object shape, graph data, and iconic representation are modified as shown in Figure 2.14. This amounts to applying the operator E1IN(#1, nil). The system inserts the vertex of the operator E1IN(#1, nil) into the edge of the contour #1, creates a new edge that corresponds to the contour #4, and assigns the label of a virtual pit to the lower endpoint of the edge of the contour #4 .

Applying an E1OUT operator

When a 1D cell e^1 that splits the contour #1 outside it is attached to #1, the object shape, graph data, and iconic representation are modified as shown in Figure 2.15. This amounts to applying the operator E1OUT(#1, nil). The system inserts the vertex

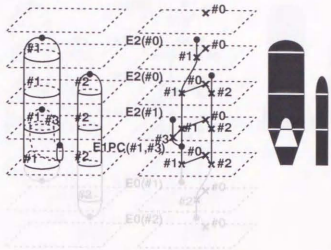


Figure 2.12: Object data modified by an E1PC operator: (a) the object shape, (b) graph data, and (c) iconic representation

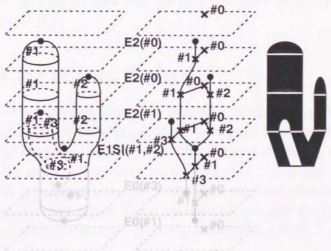


Figure 2.13: Object data modified by an E1SI operator: (a) the object shape, (b) graph data, and (c) iconic representation

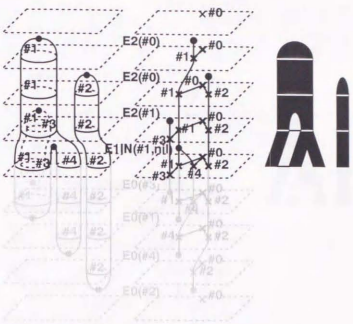


Figure 2.14: Object data modified by an E1IN operator: (a) the object shape, (b) graph data, and (c) iconic representation

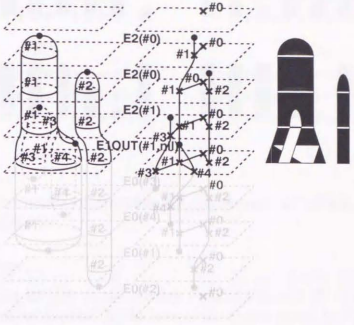
of the operator E1OUT(#1, nil) into the edge of the contour #1, creates a new edge that corresponds to the contour #4, and assigns the label of a virtual pit to the lower endpoint of the edge of #4.

2.3 Interface for Editing Icons

This section explains how to implement an interface for designing the topological skeletons of object shapes. As describe previously, the topological skeletons are edited with the iconic representation of the Reeb graph. First, the user selects the type of the next Morse operator in his design. The user then picks up an icon that corresponds to the desired operator and specifies how to paste the icon with the interface. As described in Section 2.2.2, the system can reject illegal operations because the system examines the validity of the object shape whenever the object shape is modified.

2.3.1 Pasting Primitive Icons

This subsection describes how to paste the primitive icons of Figure 2.9 with the interface. It is noted that pasting the primitive icons is equivalent to applying the fundamental Morse operators. In the following, the primitive icons are attached to the object shape shown in Figure 2.16.



(a) (b) (c)

Figure 2.15: Object data modified by an E1OUT operator: (a) the object shape, (b) graph data, and (c) iconic representation

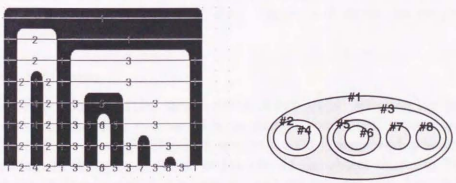


Figure 2.16: The iconic representation of an existing surface and its bottom cross section

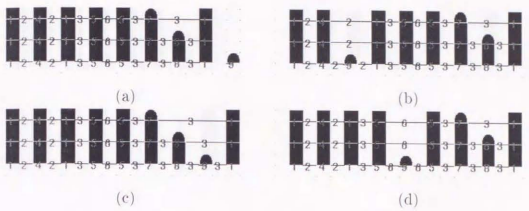


Figure 2.17: The results of candidate E2 operations: (a) E2(#0), (b) E2(#2), (c) E2(#3), and (d) E2(#6).

Pasting an E2 icon

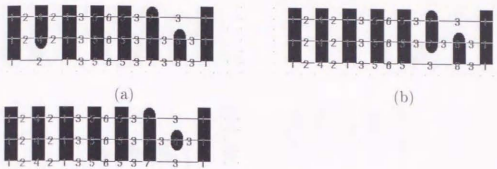
Let us paste a solid E2 icon to the existing icons of Figure 2.16. A solid E2 icon can be put inside a hollow contour. In the case of the object shown in Figure 2.16, the E2 icon can be put inside #0, #2, #3, and #6 because they are hollow contours. The corresponding Morse operators are E2(#0), E2(#2), E2(#3), and E2(#6). Figure 2.17 shows the results of these candidate operations. The system only permits acceptable operations that result in topologically valid objects.

Pasting an E0 icon

Let us paste a solid E0 icon to the existing icons of Figure 2.16. A solid E0 icon can be pasted to solid contours in order to close the contours; note that the contours where E0 icons are pasted do not have any child contours. In the case of the object shown in Figure 2.16, the E0 icon can be pasted to #4, #7, or #8. The corresponding Morse operators are E0(#4), E0(#7), and E0(#8). Figure 2.18 shows the results of these candidate operations.

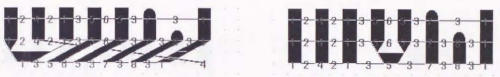
Pasting an E1PC icon

The process of pasting an E1PC icon consists of two steps: selecting the parent contour and selecting the child contour. Let us paste a solid E1PC icon to the existing icons of Figure 2.16. A solid E1PC icon merges a solid contour and a hollow contour that are siblings in the contour trees. In the case of the object shown in Figure 2.16, the candidates for the parent contour are #1 and #5 because they have hollow child contours. The corresponding Morse operators are E1PC(#1, #?) and E1PC(#5, #?), where #? represents an undecided child contour. Figure 2.19 shows the results of these candidate operations in the first step. As shown in this figure, the system automatically finds one of the candidates for the child contour by default, which can be modified in the second step. The candidates for the child contour in the second step are hollow



(a) (b) (c)

Figure 2.18: The results of candidate E0 operations: (a) E0(#4), (b) E0(#7), and (c) E0(#8)



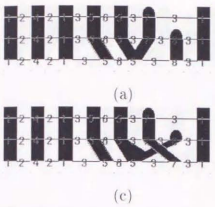
(a) (b)

Figure 2.19: The results of candidate E1PC operations in the first step: (a) E1PC(#1, #?) and (b) E1PC(#5, #?) (#? represents an undecided contour.)

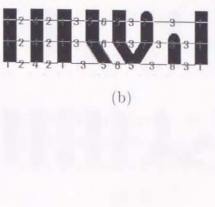
contours that are the children of the contour selected in the first step. If #1 is selected as the parent contour in the first step, #2 or #3 can be selected as the child contour. The corresponding Morse operators are E1PC(#1, #2) and E1PC(#1, #3). Figure 2.20 shows the results of these candidate operations in the second step.

(a) (b)

Figure 2.20: The results of candidate E1PC operators in the second step: (a) E1PC(#1, #2) and (b) E1PC(#1, #3).



(a)



(b)

(c)

Figure 2.21: The results of candidate E1SI operators in the first step: (a) E1SI(#5, #?), (b)E1SI(#7, #?), and (c)E1SI(#8, #?) (#? represents an undecided contour.)

(a)

(b)

Figure 2.22: The results of candidate E1SI operations in the second step: (a) E1SI(#5, #7) and (b) E1SI(#5, #8)

Pasting an E1SI icon

The process of pasting an E1SI icon consists of two steps: selecting the first and second contours that are siblings in the contour tree. Let us paste a solid E1SI icon to the existing icons of Figure 2.16. A solid E1SI icon merges two solid sibling contours into one. In the case of the object shown in Figure 2.16, the candidates for the first contour are #5, #7, and #8 because they have the common parent contour #3. The corresponding Morse operators are E1SI(#5, #?), E1SI(#7, #?), and E1SI(#8, #?), where #? represents an undecided second contour. Figure 2.21 shows the results of these candidate operations in the first step. As shown in this figure, the system automatically finds one of the candidates for the second contour by default, which can be modified in the second step. The candidates for the second contour are the rest of the candidate contours in the first step. If #5 is selected as the first contour, the candidates in the second step are #7 and #8. The corresponding Morse operators are E1SI(#5, #7) and E1SI(#5, #8). Figure 2.22 shows the results of these candidate operations in the second step.

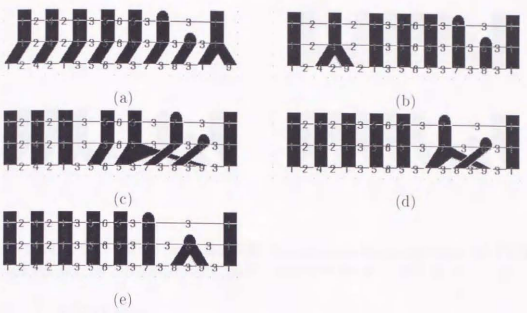


Figure 2.23: The results of candidate E1IN operations in the first step: (a)E1IN(#1, (#?)), (b)E1IN(#4, (#?)), (c)E1IN(#5, (#?)), (d)E1IN(#7, (#?)), and (e)E1IN(#8, (#?)) ((#?) represents an undecided list of contours to be contained in the newly created contour.)

Pasting an E1IN icon

The process of pasting an E1IN icon consists of two steps: selecting the contour to be split and assigning its child contours to the newly created contour. Let us paste a solid E1IN icon to the existing icons of Figure 2.16. A solid E1IN icon splits a solid contour to generate a new solid contour that becomes a sibling of the original one in the contour tree. In the case of the object shown in Figure 2.16, the candidate contours to be split are #1, #4, #5, #7, and #8. The corresponding Morse operators are E1IN(#1, (#?)), E1IN(#4, (#?)), E1IN(#5, (#?)), E1IN(#7, (#?)), and E1IN(#8, (#?)), where (#?) represents an undecided list of contours to be contained in the newly created contour. Figure 2.23 shows the results of these candidate operations in the first step. As shown in this figure, the system automatically assigns the child contours by default, which can be modified in the second step. The candidate lists of contours are equivalent to the possible subsets of the child contours, i.e., the power set of the child contours. If #1 is selected as the contour to be split in the first step, the candidate lists are nil, (#2), (#3), and (#2, #3) because #2 and #3 are the child contours of #1. The corresponding Morse operators are E1IN(#1, nil), E1IN(#1, (#2)), E1IN(#1, (#3)), and E1IN(#1, (#2, #3)). Figure 2.24 shows the results of these candidate operations in the second step.

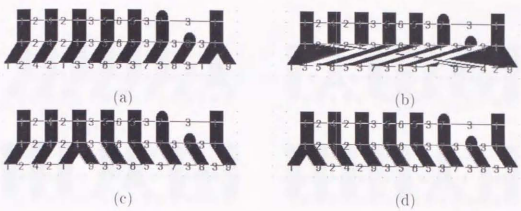


Figure 2.24: The results of candidate E1IN operations in the second step: (a) E1IN(#1, nil), (b)E1IN(#1, (#2)), (c)E1IN(#1, (#3)), and (d)E1IN(#1, (#2, #3))

Pasting an E1OUT icon

The process of pasting an E1OUT icon consists of two steps: selecting the contour to be split and assigning its sibling contours to the newly created contour. Let us paste a solid E1OUT icon to the existing icons of Figure 2.16. A solid E1OUT icon splits a solid contour to generate a new hollow contour that is a child of the original one in the contour tree. In the case of the object shown in Figure 2.16, the candidate contours to be split are #1, #4, #5, #7, and #8. The corresponding Morse operators are E1OUT(#1, (#?)), E1OUT(#4, (#?)), E1OUT(#5, (#?)), E1OUT(#7, (#?)), and E1OUT(#8, (#?)), where (#?) represents an undecided list of contours to be contained in the newly created contour. Figure 2.25 shows the results of these candidate operations in the first step. As shown in this figure, the system automatically assigns the sibling contours by default, which can be modified in the second step. The candidate lists of contours are equivalent to the possible subsets of the sibling contours, i.e., the power set of the sibling contours. If #7 is selected as the contour to be split in the first step, the candidate lists are nil, (#5), (#8), and (#5, #8) because #5 and #8 are the sibling contours of #7. The corresponding Morse operators are E1OUT(#7, nil), E1OUT(#7, (#5)), E1OUT(#7, (#8)), and E1OUT(#7, (#5, #8)). Figure 2.26 shows the results of these candidate operations in the second step.

2.3.2 Handling Macro Operations

Since the macro operation is equivalent to applying a pair of the Morse operators, the interface for pasting primitive icons is easily extended to that for handling the macro operations. In order to perform a macro operation, the user first specifies the position where a new critical point will be inserted (Figure 2.27(a)). The system hides the icons lower than the specified critical section and accepts the user's inputs through the interface (Figure 2.27(b)). The interface is almost the same as those described in Section 2.3.1 because this step amounts to applying one of the six Morse operations. The user

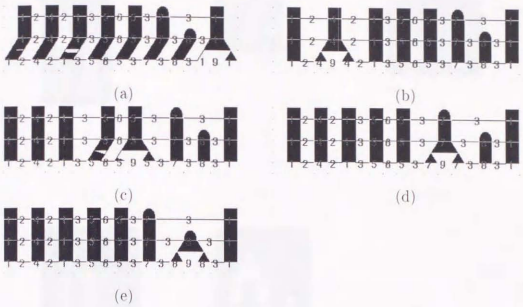


Figure 2.25: The results of candidate E1OUT operations in the first step: (a) E1OUT(#1, (#?)), (b) E1OUT(#4, (#?)), (c) E1OUT(#5, (#?)), (d) E1OUT(#7, (#?)), and (e) E1OUT(#8, (#?)) (#? represents an undecided list of contours to be contained in the newly created contour.)

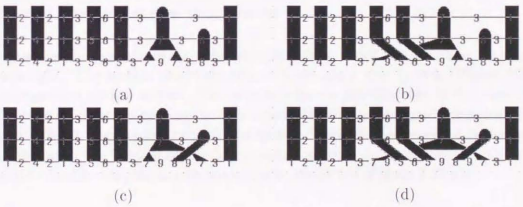


Figure 2.26: The results of the candidate E1OUT operations in the second step: (a) E1OUT(#7, nil), (b) E1OUT(#7, (#5)), (c) E1OUT(#7, (#8)), and (d) E1OUT(#7, (#5, #8))

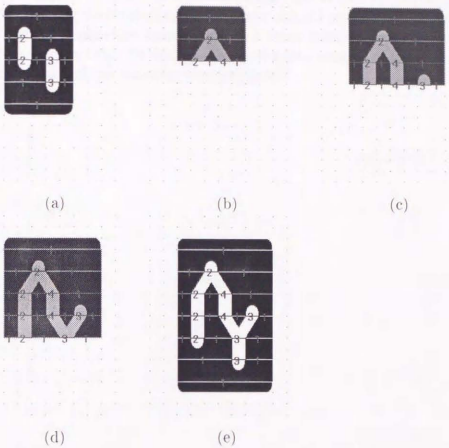


Figure 2.27: Editing icons for a macro operation: (a) an original object, (b) inserting the upper critical section, (c) going down step by step, (d) inserting the lower critical section, and (e) an object after the operation

then specifies the position of the other critical points, which is lower than the previous one in height. The system shows the hidden icons again step by step (Figure 2.27(c)) till this specified critical section. The same interface is provided also in this case except that the system does not destroy the topological skeletons of previously designed objects (Figure 2.27(d)). During this process, the system checks the sequences of critical points from the top to the bottom of the object shape in order to avoid any invalid parts of the shape. In this way, the macro operation is completed (Figure 2.27(e)).

2.4 Summary

This chapter has described the method of designing the topological skeletons of object shapes together with its embeddings in 3D space. To represent the topological skeletons and embeddings, the iconic representation of the Reeb graph, which is called the

embedded Reeb graph, is used. The Reeb graph is edited using the Morse operators that describe the way of connecting critical points. The macro operations are also introduced for avoiding the limitations of the height-ordered operations. This chapter has also presented the schemes for representing such topological skeletons and embeddings in the system. The interface for editing the iconic representation of the Reeb graph has been implemented and the examples were presented.

Chapter 3

Designing Geometry Using Manifold Mappings

The previous chapter described the method of designing the topological skeletons of smooth object using the Reeb graph. This chapter presents techniques for designing geometry of the object shape based on the designed topological skeletons [123].

3.1 Generating Surfaces Using Manifold Mappings

Generating smooth surfaces of arbitrary topological type is a topic of interest in the field of shape modeling. Since such topologically complicated surfaces require irregular decomposition of polyhedra, it is hard to form the whole object surface by pasting only rectangular patches such as tensor-product B-spline patches[1]. In order to overcome this difficulty, several techniques are proposed for generating smooth surfaces on multi-sided regions. Hosaka and Kimura [45] proposed a method of generating multi-sided patches by extending Bézier patches; however, their method has severe restrictions on the number of sides of the domain polygon. Loop and DeRose [68, 67] proposed a method that avoids these restrictions using multivariate barycentric coordinates, where the proposed patches are called S-patches. Whereas the above two methods use the networks of straight lines that do not lie on the object surface, the method proposed by Kuriyama [60, 61] uses a curve network whose curve segments are fit to the object surface. His method generates an n-sided smooth patch by blending the swept subsurfaces of the n-boundary curve segments.

Although the above methods of generating n-sided patches are efficient, they suffer from the restrictions on the underlying configuration of the network. This means that the methods cannot avoid the restrictions on the shapes of n-sided patches when it is necessary to connect the patches seamlessly with the continuity on the boundaries

[1] Recently, Eck and Hoppe [25] has presented the method of constructing surfaces of arbitrary topological type only from rectangular patches such as tensor-product B-spline patches. Although, their method automatically constructs the rectangular decomposition of the surface from discrete samples, it takes much time to get the decomposition and the resultant decomposition is not intuitive to use for further applications.

between adjacent patches. These restrictions often cause the non-intuitive design of the patches. Varady [129] avoided these restrictions by introducing overlapping patches each of which has its own local parametrization. In his technique, the overlapping patches are assigned to the vertices of the network that outlines the object shape. The advantage of his method is that it automatically guarantees C^1-continuity on the boundary curves without any special restrictions on the shapes of local patches. Note that his parametrization techniques can be regarded as a kind of manifold mapping techniques, and they have more flexibility in assembling local patches than the above techniques that connect the patches seamlessly. The manifold mapping techniques are also applied to several researches on shape modeling [40], 3D shape recovery [100], and image generation [59, 121][2].

This study uses the Varady's technique to construct such manifold mappings. To design the geometry of the object shapes, the user designs flow curves that run on the object surface. From the given flow curves, the system automatically constructs a network of curve segments called a control network in order to support the manifold mappings. A local patch is assigned to each vertex of the control network and is called a vertex patch. The shape of the vertex patch is designed using variational optimization techniques, which are described in Chapter 4. Finally, the system blends the shapes of vertex patches in their overlapping parametric domains to generate the whole surface of the smooth object.

3.2 Constructing a Control Network

This section describes how to design a control network that encloses an object shape.

3.2.1 Flow Curves

The first step of the geometric design is to specify the shapes of *flow curves* that run on the object surface. Figure 3.1 shows an example of the flow curves that run on the surface of a torus. As illustrated in Figure 3.1, the flow curves are assumed to go down monotonously with respect to the height value in the implementation. The flow curve used in the system is the curve that goes from one critical point to another in order to outline the rough shape of the object. In particular, the flow curves can be used to guide the cross-sectional shapes of the object surface. For later convenience, a pass is assumed to have four incident flow curves, two of which come to the pass from the upper and two of which go out of the pass to the lower. Note that since the configuration of flow curves is based on those of the critical points of an object surface, the differential properties of the surfaces around the critical points are reflected in the process of the surface generation.

The flow curve is represented by an endpoint-interpolating cubic B-spline curve in the system. This B-spline curve is defined on a knot sequence where the knots are

[2] Appendix G describes a method of generating multiple-viewpoint images using the manifold mappings.

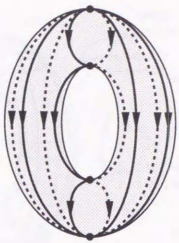


Figure 3.1: Flow curves of a torus

equally spaced except for its ends. The examples of the endpoint-interpolating B-spline functions are shown in Figure 4.3 (Chapter 4). As can be seen in this figure, the freedom of the B-spline curve is controlled by inserting and deleting the internal knots of the knot sequence. The shape of the flow curve is designed by imposing point-position constraints and tangent constraints of the curve. While preserving the imposed constraints, the system determines the curve shape by minimizing its deformation. This can be implemented by using the techniques of Welch and Witkin [135] that optimize the energy function subject to the deformation of the curve. In determining the shape using these techniques, the freedom of the curve can be adjusted in proportion to the degree of the constraints and is also specified manually by users when necessary. Note that the endpoint-interpolating B-splines are well suited to the multiresolution representations of shapes with spline wavelets [20, 94]. This scheme is also used for the design of local patches in the system and described in detail in Chapter 4.

3.2.2 Control Network

In order to generate an object surface from the given flow curves, the system automatically creates a *control network* that enclose the object shape. This control network is created by adding appropriate cross-sectional curves to the given flow curves. The control network of a torus is shown in Figure 3.2. By default, the cross-sectional curves are added to the cylindrical parts of the object that correspond to the edges of the Reeb graph. Of course, such cross-sectional curves can be modified by adding constraints to the curves or moving control points of the curves in the system. As can be seen in Figure 3.2, the control network encloses the object and will serve as a basic frame for constructing the manifold mappings.

A *peak vertex*, a *pit vertex*, and a *pass vertex* are defined to be the vertices of the control network that correspond to a peak, a pit, and a pass of the object surface, respectively. Other vertices of the control network are called *regular vertices*. As illustrated in

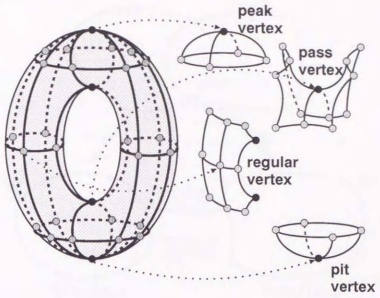


Figure 3.2: A control network of a torus and the regions around its vertices

Figure 3.2, the control network decomposes the object surface into regions surrounded by its curve segments. In the implementation of this study, the decomposed faces are topologically equivalent to three-sided regions (i.e., triangles), four-sided regions (i.e., quadrilaterals), and five-sided regions (i.e., pentagons). In particular, the faces around a peak vertex or a pit vertex are three-sided regions, the faces around a pass vertex are five-sided regions, and other faces are four-sided regions. The rule of the network construction also implies that the regular vertices and pass vertices have only four incident curve segments in the control network. Since this surface decomposition takes into account the configuration of the critical points on the object surface, the surfaces around the critical points are made smooth without any special manipulations.

3.3 Constructing Manifold Mappings

This section describes the manifold mappings based on Varady's parametrization assignment; the manifold mappings are used to form the overall surface of the object from local patches. In the implementation, the following procedures are carried out for designing the geometry of the object shape. First, local patches are assigned to the vertices of the control network. The local patches are then designed using the variational optimization techniques proposed by Welch and Witkin [135], which will be described in Chapter 4. In this variational optimization, the curve segments of the control network are used as geometric constraints to determine the shapes of the local patches. Finally, the local patches are glued together to form the overall surface of the smooth object so that the adjacent patches are blended in their overlapping parametric domains.

One of the advantages of this framework is the locality of the geometric design. In

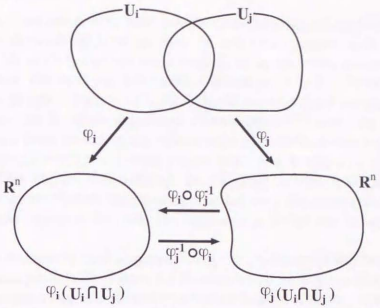


Figure 3.3: Charts of a manifold

other words, the users can design the local patches without modifying the overall shape of the object. Another advantage is the flexibility in assembling the local patches. This means that it is unnecessary to take care of the continuity in connecting the patches because the adjacent local patches have overlaps where their shapes are blended while preserving the continuity. In addition, this implementation enables us to introduce multiresolution design of object surfaces using wavelets. This will be described in detail in Chapter 4.

3.3.1 Definition of the Manifold

Let us see the definition of a manifold [81]. An n-dimensional *chart* is a pair $(U_i,\ \varphi_i)$, where U_i is an open set and φ_i is a mapping of U_i onto an open set of $\boldsymbol{R}^n$. A pair of charts $(U_i,\ \varphi_i)$ and $(U_j,\ \varphi_j)$ is said to be C^r-*compatible* if $\varphi_j \circ \varphi_i{}^{-1}$ and $\varphi_i \circ \varphi_j{}^{-1}$ are C^r on $U_i \cap U_j$ when $U_i \cap U_j \neq \emptyset$. The mappings $\varphi_j \circ \varphi_i{}^{-1}$ and $\varphi_i \circ \varphi_j{}^{-1}$ are called *coordinate transformations*. A C^r n-dimensional *manifold* is then defined to be an n-dimensional atlas $\{(U_i,\ \varphi_i)\}$ with the condition that every pair of charts in the atlas is C^r-compatible. Figure 3.3 illustrates the charts of a manifold. In the implementation, the system guarantees the C^1-continuity of the surface as described later in Section 3.3.4.

In this thesis, the dimension of the manifold n is set to 2. The open neighborhood is the region around a vertex of the control network shown in Figure 3.2, and the corresponding coordinate transformation is the mapping obtained from the Varady's parametrization techniques.

3.3.2 Overlapping Local Patches

This subsection describes how to establish the coordinate mappings between the object surface and the decomposed local patches. In order to construct such mappings, this study uses the Varady's parametrization techniques as described above. Following the Varady's notation, the local patch is called a *vertex patch* in this thesis. According to the Varady's techniques, each vertex patch has a vector-valued parametric form that maps the rectangular bivariate parametric domain onto 3D space. On the other hand, the n-sided region decomposed by the control network is defined on a regular n-gon, i.e., on a regular triangle when $n = 3$, on a square when $n = 4$, and on a regular pentagon when $n = 5$. This implies that defining the mapping between a surface region and a vertex patch amounts to defining the mapping between the corresponding parametric domains, i.e., the mapping between the regular n-gon and the bivariate parametric domain.

The system constructs such mapping using the parametrization based on a planar biquadratic Bézier patch [129]. Figure 3.4 illustrates how to map the bivariate parameter (u, v) onto an n-gon using this parametrization technique. Due to the cyclic symmetry, the same consideration can be applied for other vertex patches and polygon sides that are obtained with the rotation by $\frac{2\pi}{n}$. Let T_{ij} denote the control points of the Bézier patch where $i, j = 0, 1$, and 2. The bivariate coordinates (u, v) can be mapped onto the coordinates in the polygon (μ, ν) using the following equation:

$$\begin{pmatrix} \mu & \nu \end{pmatrix} = \begin{pmatrix} (1-u)^2 & 2u(1-u) & u^2 \end{pmatrix} \begin{pmatrix} T_{00} & T_{10} & T_{20} \\ T_{01} & T_{11} & T_{21} \\ T_{02} & T_{12} & T_{22} \end{pmatrix} \begin{pmatrix} (1-v)^2 \\ 2v(1-v) \\ v^2 \end{pmatrix}. \quad (3.1)$$

Figure 3.4(a) shows the bivariate parametrization in a regular triangle ($n = 3$), and Figure 3.4(b) shows that in a regular pentagon ($n = 5$). In this parametrization, T_{00} lies at the base vertex ($u = 0, v = 0$), T_{02} and T_{20} lie at the vertices adjacent to T_{00}, and T_{01} and T_{10} lie at the midpoints of the edges emanating from T_{00}. T_{11} lies at the center of the regular polygon. T_{21} and T_{12} lie at the midpoint of the edge next to the edge emanating from the base point T_{00}. T_{22} is generally put in the middle of the polygon boundary between T_{21} and T_{12}.

As can be seen in Figure 3.4, the control points of the above planer Bézier patch lie at the corner or the midpoint of the polygon side. Suppose that n denotes the number of polygon sides, ϕ_n denotes the corner angle, and ω_n denotes the angle between the first corner and the positive x-axis. The following expressions provide the coordinates of the corners (c_{uk}, c_{vk}) $(k = 1, \ldots, n)$ and the midpoints of the polygon sides (m_{uk}, m_{vk}) $(k = 1, \ldots, n)$:

$$(c_{uk},\ c_{vk}) = d_n\Big(\cos(\omega_n + (k-1)\frac{2\pi}{n}),\ \sin(\omega_n + (k-1)\frac{2\pi}{n})\Big) \quad (3.2)$$

$$(m_{uk},\ m_{vk}) = h_n\Big(\cos(\frac{3\pi}{2} + (k-1)\frac{2\pi}{n}),\ \sin(\frac{3\pi}{2} + (k-1)\frac{2\pi}{n})\Big), \quad (3.3)$$

where $d_n = \dfrac{1}{2\cos(\frac{1}{2}\phi_n)}$, and $h_n = \dfrac{1}{2\tan(\frac{1}{2}\phi_n)}$ ($\phi_3 = \dfrac{2\pi}{3}$; $\omega_3 = \dfrac{7\pi}{12}$; $\phi_5 = \dfrac{3\pi}{10}$; $\omega_5 = \dfrac{13\pi}{20}$).

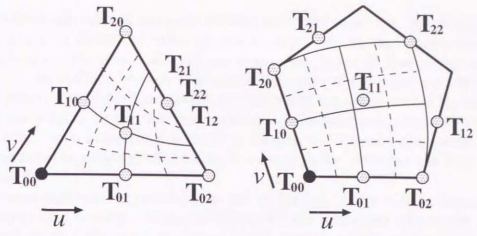


Figure 3.4: Varady's biquadratic parametrization of n-gons: (a) $n = 3$ and (b) $n = 5$ [129]

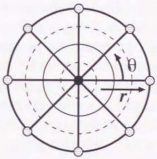


Figure 3.5: Polar parametrization for peaks and pits

The inversion of the given biquadratic map reveals another issue in this mapping because it requires numerically expensive calculations. One of the practical solutions is to approximate the inverse mapping by taking dense grid of the domain points with the corresponding parameter values of the vertex patches, and it would provide satisfactory results. The system takes this practical solution because the system can obtain the table of parameter mappings with only one calculation.

As described in Section 3.2.2, the peak and pit vertices of a control network can have more than four incident curve segments. For these cases, polar parametrization is used instead of the Varady's one for the peak and pit vertices as shown in Figure 3.5. This is made possible because the curve segments and adjacent faces around the peak (pit) constitute a spider's web in the control network.

3.3.3 Designing Local Patches

Having defined the manifold mappings between the object surface and the vertex patches, the next step is to design the vertex patches appropriately. In the implementation, the curve segments of the control network are used as the geometric constraints of the vertex patches. In addition to such constraints, the shape of the patch is determined so that the energy function subject to the deformation of the patch reaches the minimum. This is made possible by the variational optimization techniques proposed by Welch and Witkin [135], which is described in detail in Chapter 4. This subsection describes how to use the curve segments as the geometric constraints for designing the shapes of the vertex patches.

The curve segments are considered to run on the boundaries of the polygon regions in the parametric domains. Using the Welch-Witkin techniques, the vertex patch is determined so that the patch is aligned to the curve segments at the corresponding parameter values. Figure 3.6 illustrates the parametrization assignment of the vertex patches when they are fit to the curve segments of the control network. In this figure, the solid curves represent the parametric paths of the curve segments and the dotted curves represent the associated parametrization assignment of the vertex patch. The bottom left vertex patch is the simplest case where the standard bivariate parametrization is assigned to a rectangular region. The top right vertex patch has two triangle regions because it contains a peak on its boundary. In the triangles, bivariate parametrization is assigned using the biquadratic Bézier patch as described in Section 3.3.2. The bottom right vertex patch has a pass at its center and hence consists of four pentagons. Also in this case, the biquadratic Bézier parametrization is used to assign the bivariate parametrization to the pentagon regions. As an exceptional case, the vertex patch of a peak has a polar coordinate system as shown at the top left of the figure.

In the implementation, additional constraints such as point-position constraints can be attached to the faces of the control network. The constraints of a face are shared by the vertex patches that have effects on the shape of the face.

In this way, the system determines the shapes of the vertex patches by using the curve segments of the control network as geometric constraints.

3.3.4 Blending Local Patches

The final step of generating the overall surface of the object is to assemble the vertex patches using the manifold mappings. Since any point of an object surface is covered with more than one vertex patch, the object surface can be generated by blending the vertex patches in their overlapping parametric domains.

Let us calculate the coordinates of the point whose parameter vector is $\boldsymbol{x} = (u, v)$ as shown in Figure 3.7. Here, $\boldsymbol{x}$ is contained in the domains of five vertex patches because $\boldsymbol{x}$ lies inside a five-sided region. Note that Figure 3.7 illustrates only two of the five vertex patches for simplicity. The parameter vector $\boldsymbol{x} = (u, v)$ is mapped onto the two parametric domains by π_i and π_j, where the mapped vectors are denoted by $\boldsymbol{x}_i = (u_i, v_i)$ and $\boldsymbol{x}_j = (u_j, v_j)$ as illustrated in Figure 3.7. It means that $\pi_i(u, v) = (u_i, v_i)$ and $\pi_j(u, v) = (u_j, v_j)$. Since the shapes of the two vertex patches have been

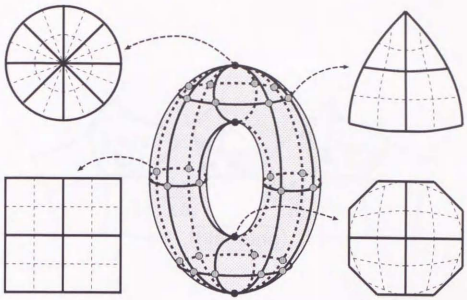


Figure 3.6: Mapping curve segments to the parametric domains of vertex patches

determined, the coordinates of the patches that correspond to $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ can be obtained as $\boldsymbol{s}_i(\boldsymbol{x}_i) = \boldsymbol{s}_i(u_i, v_i)$ and $\boldsymbol{s}_j(\boldsymbol{x}_j) = \boldsymbol{s}_j(u_j, v_j)$, respectively. For interpolating the vertex patches, the system uses a blending function $B(\boldsymbol{x})$. The coordinates of the blended surface $\boldsymbol{s}(\boldsymbol{x})$ are calculated from

$$\boldsymbol{s}(\boldsymbol{x}) = \frac{\sum_k B(\boldsymbol{x}_k) \cdot \boldsymbol{s}_k(\boldsymbol{x}_k)}{\sum_k B(\boldsymbol{x}_k)} \quad \Big(\boldsymbol{s}(u,v) = \frac{\sum_k B(u_k, v_k) \cdot \boldsymbol{s}_k(u_k, v_k)}{\sum_k B(u_k, v_k)}\Big). \tag{3.4}$$

Here, the blending function $B(\boldsymbol{x})$ is a tensor product of polynomial functions and is arbitrarily selected if it guarantees the smoothness of the blended surface. For example, the blending function can be set to the function $B(u, v) = b(u) \cdot b(v)$ ($(u, v) \in [-1, 1] \times [-1, 1]$) where

$$b(t) = \begin{cases} -t^3 - \dfrac{3}{2}t^2 + \dfrac{1}{2} & (-1 \le t \le 0) \\ t^3 - \dfrac{3}{2}t^2 + \dfrac{1}{2} & (0 \le t \le 1) \end{cases} \tag{3.5}$$

Figure 3.8 shows this polynomial function $b(t)$. Note that the coordinate map φ_k for the vertex patch $\boldsymbol{s}_k$ can be expressed by $\varphi_k = \boldsymbol{s}_k^{-1}$ (cf. Figure 3.3)

According to the Varady's paper [129], G^1-continuity of the surface is guaranteed on the curve segments of the control network. It means that in the context of the manifold mappings, C^1-continuity can be obtained by selecting the appropriate reparametrization around the curve segments. Let us examine the surface continuity on the curve segments in the following. Suppose that the boundary curve segments are covered by two vertex patches: the vertex patch of p_1 and the vertex patch of p_2 as shown in Figure 3.9. Because of the overlapping rule of the vertex patches, only two vertex patches will be

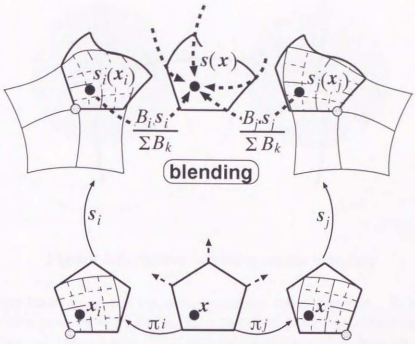


Figure 3.7: Blending local patches

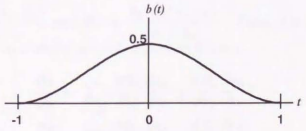


Figure 3.8: An example of the polynomial blending function

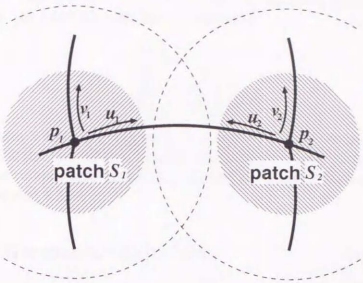


Figure 3.9: Surface continuity on the boundary

in effect at the limit along the common boundary curve segment. As shown in this figure, the vertex patches of p_1 and p_2 have the bivariate parametrization assignments (u_1, v_1) and (u_2, v_2), respectively. Note that this notation differs from the previous one in that u_1 and u_2 represent the quasi-parallel coordinates along the common boundary while v_1 and v_2 the quasi-orthogonal coordinates. Suppose that the vertex patch of p_1 is represented by $\boldsymbol{s}_1(u_1, v_1)$ and the vertex patch of p_2 is represented by $\boldsymbol{s}_2(u_2, v_2)$. Let us introduce another global parametrization (u, v) in such a way that $u = u_1 = 1 - u_2$ along the common boundary.

If the surface shape is represented by $\boldsymbol{s}(u, v)$, it will be the blended sum of the two vertex patches $\boldsymbol{s}_1(u_1, v_1)$ and $\boldsymbol{s}_2(u_2, v_2)$ (on the common boundary) as

$$\boldsymbol{s}(u_1, u_2) = \sum_{k=1,2} \boldsymbol{S}_k(u_k(u, v), v_k(u, v)), \tag{3.6}$$

where $\boldsymbol{S}_1 = \dfrac{b_1 \cdot \boldsymbol{s}_1(u_1, v_1)}{b_1 + b_2}$ and $\boldsymbol{S}_2 = \dfrac{b_2 \cdot \boldsymbol{s}_2(u_2, v_2)}{b_1 + b_2}$. Thus, the partial derivatives of $\boldsymbol{s}(u, v)$ with respect to u and v are calculated as follows.

$$\frac{\partial \boldsymbol{s}}{\partial u} = \sum_{k=1,2} \left(\frac{\partial \boldsymbol{S}_k}{\partial u_k} \frac{\partial u_k}{\partial u} + \frac{\partial \boldsymbol{S}_k}{\partial v_k} \frac{\partial v_k}{\partial u} \right) \tag{3.7}$$

$$\frac{\partial \boldsymbol{s}}{\partial v} = \sum_{k=1,2} \left(\frac{\partial \boldsymbol{S}_k}{\partial u_k} \frac{\partial u_k}{\partial v} + \frac{\partial \boldsymbol{S}_k}{\partial v_k} \frac{\partial v_k}{\partial v} \right) \tag{3.8}$$

Since $u_1 = u, u_2 = 1 - u$, and $v_1 = v_2 = 0$ along the common boundary curve,

$$\frac{\partial u_1}{\partial u} = 1; \qquad \frac{\partial u_2}{\partial u} = -1; \qquad \frac{\partial v_1}{\partial u} = \frac{\partial v_2}{\partial u} = 0. \tag{3.9}$$

According to the Varady's paper [129], the sufficient condition for obtaining the G^1-continuity is that the following equations are satisfied.

$$\frac{\partial u_1}{\partial v} = -\frac{\partial u_2}{\partial v}, \tag{3.10}$$

$$\frac{\partial v_1}{\partial v} = \frac{\partial v_2}{\partial v} = c(u), \tag{3.11}$$

where $c(u)$ denotes a function of u at $v = 0$ defined by the parametrization. This condition holds in the case of the Varady's biquadratic Bézier parametrization technique because the two patches in effect have continuous parametrization on the common boundary (cf. Section 3.3.2).

3.4 Other Geometric Operations

The prototype system implemented in this study also provides the following geometric operations.

Interference checking

The system provides operations that find the illegal interferences among surface layers in order to maintain the predefined topological skeletons of the object.

Flat-surface generation

In order to support flat surface generation perpendicular to the height axis, the system provides operations that set the height of the surface patches along the height axis to zero. This means that we first design a smooth surface and then press it by setting its height to zero as shown in Figure 3.10. The basic ideas of these operations are presented in [107].

Object embedding

The system also offers operations that embed objects in 3D space. With the operations, we can design multi-layered objects as shown in Figure 3.21 by designing the two objects separately and then embedding one inside the other.

3.5 Results

This section presents design examples generated in the prototype system. The system is implemented on IRIS workstations and its software is written in C++, using OpenGL as the graphics library and Motif for the user interface.

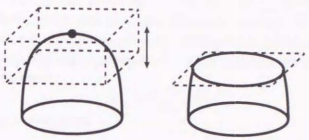


Figure 3.10: Generating a flat top

Design steps

In the system, the user specifies the topological skeletons of an object shape with the iconic representation of the Reeb graph, and then designs its geometry by modifying the shapes of the flow curves or the control network the system offers by default. Figure 3.11 shows the steps of designing a torus in the system. Firstly, the peak of the torus is created by pasting an E2 icon as shown in Figure 3.11(a). Secondly, the upper pass of the torus is created by attaching an E1IN icon as shown in Figure 3.11(b), where the initial shapes of the flow curves are also provided by the system. Thirdly, the lower pass of the torus is created by attaching an E1SI icon, which results in the hole of the torus as shown in Figure 3.11(c). Finally, the torus is completed by attaching an E0 icon as shown in Figure 3.11(d). After these steps of topological design, the user designs the geometry of the torus by modifying the flow curves. Figure 3.12 shows a flow curve designed in the system. In this figure, the shape of the flow curve is controlled by its control points or imposed constraints such as point-position and tangent constraints.

Designing objects with macro operations

Figure 3.13 shows the shape of the torus designed in the system. In addition to the above top-down surface construction, the system provides the means of attaching another surfaces to the existing torus, i.e., the macro operations described in Section 2.1.4. Figures 3.14 and 3.15 show such examples. Figure 3.14 shows a torus with two arms. The arms are pasted to the torus using macro operations. Figure 3.15 shows a monster-like object obtained by attaching an additional branch to the bottom pit of the object shown in Figure 3.14. In this way, the macro operations provide various means of designing the topological skeletons of the object shapes. Note that Figures 3.13, 3.14, and 3.15 show the flow curves, control networks, mesh samples, and rendered surfaces of the designed objects.

Design examples

Figures 3.16, 3.17, and 3.18 present the examples designed in the prototype system. Figure 3.16 shows a toy dog, Figure 3.17 shows the characters that are pasted together, and Figure 3.18 shows an inner ear organ that consists of a cochlea and three semicircular

canals. In the same way as the previous figures, each of these figures shows the flow curves, control network, mesh samples, and rendered surface of the designed object. Note that these objects have smooth shapes around their critical points. In this way, surfaces of arbitrary topological type can be designed systematically by using the critical points as the shape features.

Checking illegal interferences

As described in Section 3.4, the system provides the operations that find the illegal interferences among surface layers. Figure 3.19 illustrates two spheres where the inner sphere go through the outer one. In this figure, the illegal intersections are indicated by red spheres in the system. These operations provide the means of preserving the consistency of the predefined topological skeletons.

Generating flat surfaces

As described in Section 3.4, the system provides the operations for designing flat surfaces. Figure 3.20 illustrates such an example. Figure 3.20(b) shows a dog with flat top ears while Figure 3.20(a) shows its original non-degenerate surface.

Embedding objects

Figure 3.21 shows a multi-layered surface designed in the system. This double-layered spiral object is created by designing two spiral objects separately and then embedding one inside the other using the embedding operations described in Section 3.4.

3.6 Summary

This chapter has presented the techniques for designing the geometry of the object shape using the manifold mappings. The geometric design begins with specifying the shapes of the flow curves that run on the object surface between the critical points. From the given flow curves the system automatically generates the control network that encloses the object. Each vertex of the control network has its own local vertex patch that is mapped to the local bivariate parametric domain using a manifold mapping. The shapes of the vertex patches are determined by using the curve segments of the control network as the geometric constraints. Finally, the system generates the overall surface of the object by pasting the vertex patches with the overlaps where the vertex patches are interpolated smoothly.

One problem of this geometric design is that the object surface on the curve segments of the control network has only C^1-continuity, while Morse theory requires C^2-continuity. However, C^2-continuity (or G^2-continuity) imposes severe restrictions on the shape of the local patches and the way of gluing the patches. Besides this, since the surface generated in the system has visually appealing smoothness as shown in the above figures,

the C^1-continuity has no problems in the practical design of smooth surfaces. Consequently, this study takes the flexibility of the patch assembling while it discards the C^2-continuity the theory requires.

According to the rule of constructing a control network, the flow curves incident to the vertex of the control network constitute a crossing or a spider web in the implementation. To provide the operation for eliminating less important curve segments from the control network, it is desirable to include the T-connections of curve segments in the control network. This will reduce the users' interactions for designing the geometry such as the flow curves, and is one of the future extensions of this study.

As described in Section 3.4, illegal interferences among surface layers should be examined in order to preserve the topological validity of the object shape in the geometric design. Providing efficient operations for checking interferences based on the predefined topological skeletons is also a topic for future research.

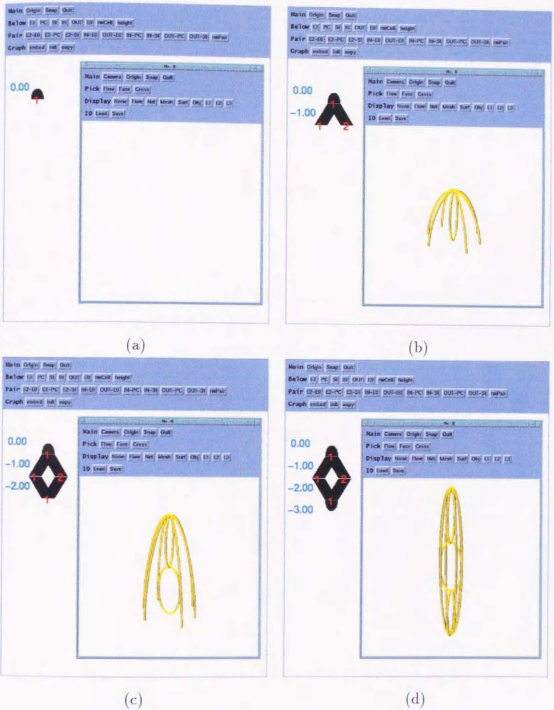


Figure 3.11: Designing a torus with the system: (a) pasting an E2 icon, (b) pasting an E1IN icon, (c) pasting an E1SI icon, and (d) pasting an E0 icon

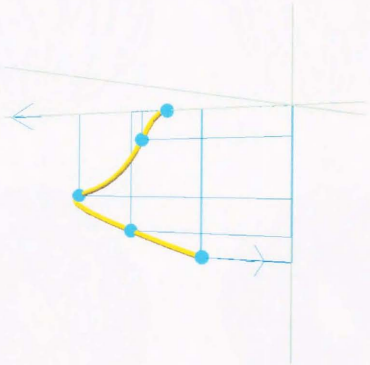


Figure 3.12: Designing a flow curve

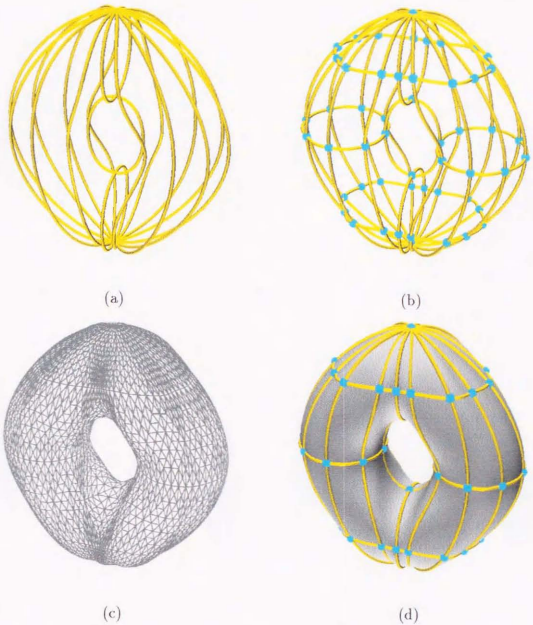


Figure 3.13: Designing a torus: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface

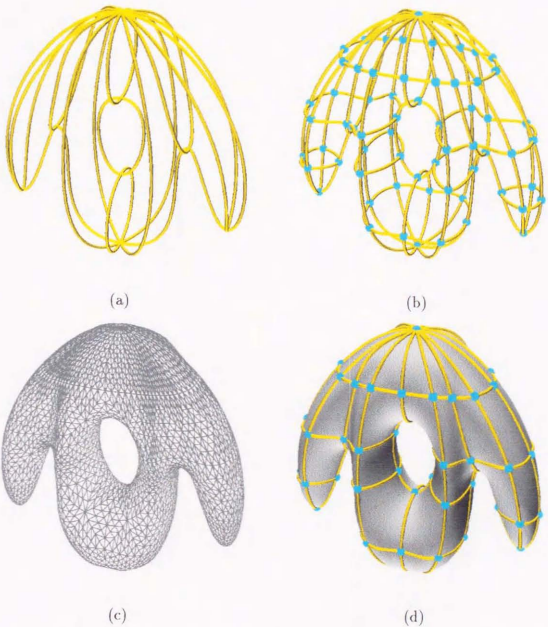


Figure 3.14: Designing a torus with arms: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface

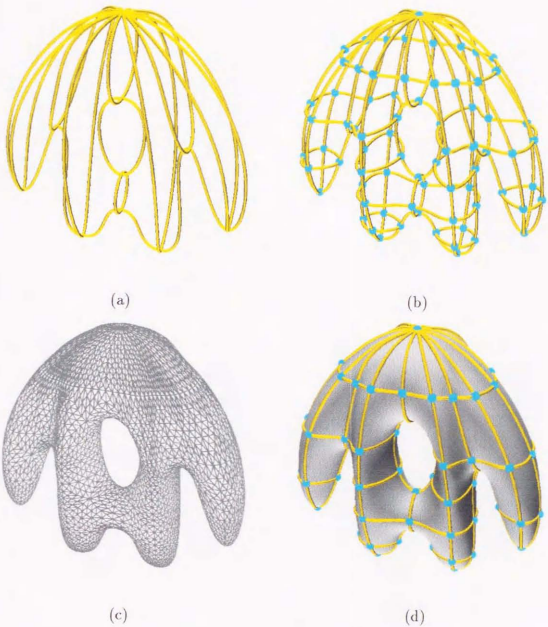


Figure 3.15: Designing a monster-like shape: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface

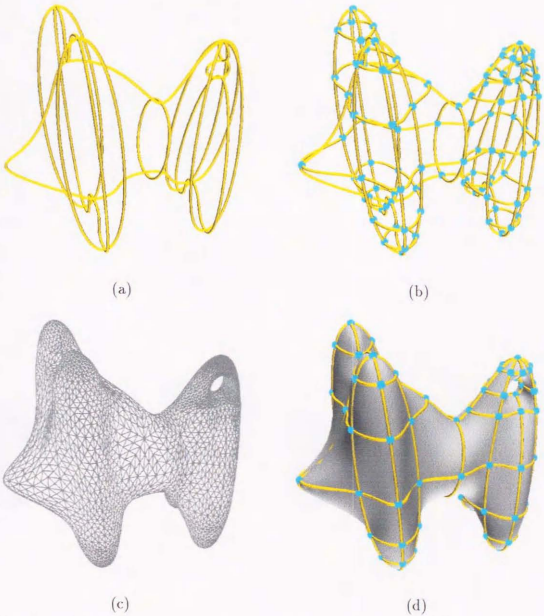
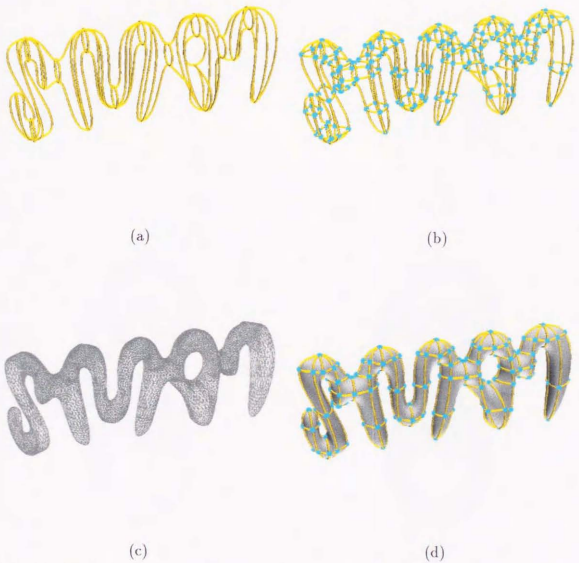


Figure 3.16: Designing a toy dog: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface



(a) (b)

(c) (d)

Figure 3.17: Designing characters: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface

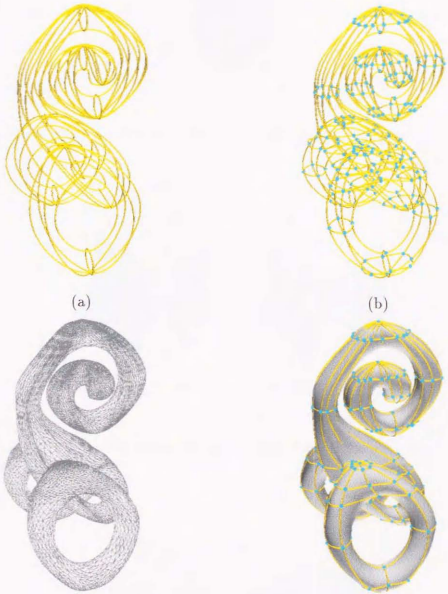
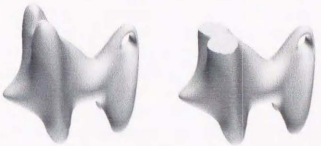


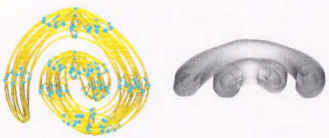
Figure 3.18: Designing an inner ear organ: (a) flow curves, (b) a control network, (c) mesh samples, and (d) an object surface

Figure 3.19: Checking illegal interferences: illegal intersections are indicated by red spheres



(a) (b)

Figure 3.20: Handling degenerate surfaces: (a) an original non-degenerate object and (b) an object with flat tops



(a) (b)

Figure 3.21: A double-layered spiral: (a) its control network and (b) its inner structure

Chapter 4

Designing Curves and Surfaces Using Multiresolution Constraints

Recent advances in wavelet theories enable hierarchical representations of curves and surfaces. Although the wavelet theories provide algorithms for transforming complicated shapes at fine and coarse levels of resolution, there is no research on designing shapes by controlling both fine and coarse levels at the same time. This chapter presents a new method of designing curves and surfaces by solving the constraints imposed on the shapes at multiresolution levels [122]. In this method, the curves and surfaces are represented by endpoint-interpolating B-splines and their corresponding wavelets. At each resolution level, the shape is determined by minimizing the energy function subject to the deformation of the shape while preserving the given constraints. Constraints at a low resolution level are converted to those at a high resolution level using wavelet transforms in order to associate all the constraints with the common basis functions. The constraints at multiresolution levels are then solved recursively from low to high resolution levels. Design examples are also presented in this chapter. The proposed method can be used to design the detailed geometry of the flow curves and the local patches described in Chapter 3.

4.1 Need for Solving Multiresolution Constraints

The advances in the wavelet theories allow us to represent curves and surfaces at multiple levels of details. In other words, wavelets serve as a mathematical tool for decomposing a shape into those at multiresolution levels. While wavelets come from signal processing and function approximation, they have been recently applied to computer graphics [114, 115, 116] including hierarchical editing of curves and surfaces [31, 36].

Finkelstein and Salesin [31] proposed a variety of editing operations such as smoothing a shape, editing an overall shape while preserving its details, attaching details to an overall shape, and so forth. These operations effectively control the shapes of curves and surfaces using their levels of details, i.e., the shape resolution. While the operations enable us to modify the overall shape while preserving its details, it is still difficult to

perform the reverse operations: modifying the details while preserving its overall shape. Suppose there is a curve as shown in Figure 4.1(a). The right-hand figure shows the curve at a fine level of resolution while the left-hand figure shows the corresponding curve at a low resolution level. Note that in Figure 4.1 the solid arrows indicate the modifications by users and the dotted arrows indicate the corresponding changes induced at the other resolution level. The Finkelstein-Salesin operations enable us to edit the overall shape without modifying its details as shown in Figure 4.1(b). However, as shown in Figure 4.1(c), it is difficult to modify the detailed shape without affecting the overall shape of the curve. For such operations, Finkelstein and Salesin introduced the *curve character library* which maintains the difference between the curves at adjacent resolution levels. Although this library is useful, it is not intuitive for users because it holds the shape data by the coefficients of wavelets. In addition, explicit operations that directly create the data for the library are not yet available.

Actually, it is impossible to fix the exact shape of the curve at the low resolution level because the changes of the details inevitably influence its coarse shape through the smoothing operations using wavelets. One of the best solutions to this is to impose constraints on the curve at multiple levels of resolution as shown in Figure 4.1(d) in order to fix several vertices of the curves at the coarse level. The goal of this chapter is to implement such design operations.

A further advantage of the constraints is that the shapes can be designed without explicitly handling their control points. Gortler and Cohen [36] presented a method for such shape design using the variational optimization techniques proposed by Welch and Witkin [135]. Their method, however, fits the shape to the imposed constraints from low to high resolution levels step by step so that the shape at the highest resolution level meets the given constraints within the specified tolerance. No operation that uses the constraints imposed at multiresolution levels has been proposed yet.

This chapter presents a new method of designing curves and surfaces by solving constraints imposed on the shapes at multiresolution levels. In this study, curves and surfaces are represented by endpoint-interpolating B-splines and the corresponding wavelets [20, 94] in order to design the shapes in a hierarchical fashion. For designing the shapes at multiresolution levels, constraints are given to the shape at each resolution level. From the given constraints, the shape is determined using the variational techniques proposed by Welch and Witkin [135]. To associate the multiresolution constraints with the common basis functions, the constraints at a low resolution level are converted to those at a high resolution level using wavelet transforms. The multiresolution constraints are then solved recursively from low to high resolution levels by taking into account the differences between the shapes at adjacent resolution levels. Several results are also presented to show the capability of this method.

This chapter is organized as follows: Section 4.2 reviews endpoint-interpolating B-splines and its corresponding wavelets used as basis functions for representing curves and surfaces. The variational techniques for designing curves and surfaces are explained in Section 4.3. Section 4.4 presents a method of solving multiresolution constraints by converting the constraints at a low resolution level to those at a high resolution level. Section 4.5 shows results of multiresolution curve and surface design. Finally, Section

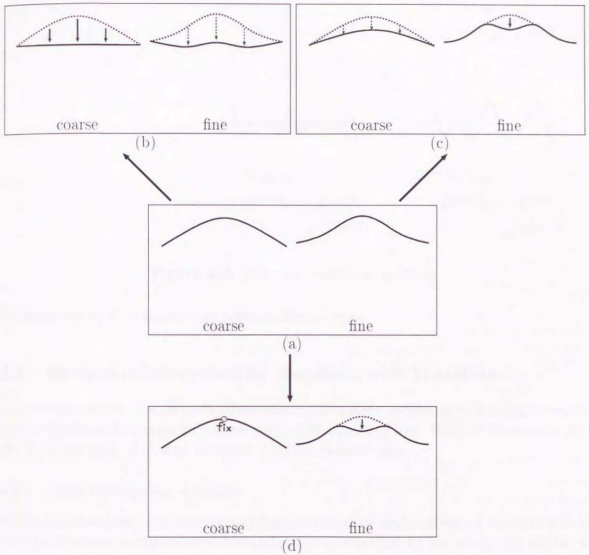


Figure 4.1: Editing a curve using multiresolution levels of details: (a) original curves at coarse and fine resolution levels, (b) editing the coarse curve (The overall shape of the corresponding fine curve is changed.) (c) editing the fine curve (The shape of the coarse curve cannot be fixed.) (d) editing both the coarse and fine curves (The shape of the fine curve can be changed while preserving the important vertices of the coarse curve.)

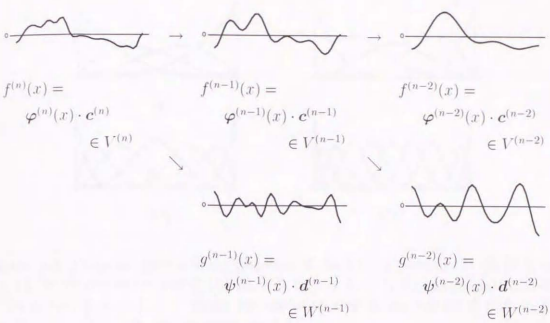


Figure 4.2: The multiresolution analysis

4.6 summarizes this chapter and refers to future work.

4.2 Endpoint-interpolating B-splines and Wavelets

This section reviews the endpoint-interpolating B-splines and its corresponding wavelets used as the basis functions for representing curves and surfaces. Refer to the papers [20, 94], tutorials [114, 115], and textbook [116] for more details.

4.2.1 Multiresolution Analysis

Mallat [69] developed a mathematical framework called *multiresolution analysis* where a shape function is represented hierarchically. According to his study, we obtain a vector space $V^{(n)}$ such that $V^{(n)}$ can be decomposed into a chain of nested vector spaces $V^{(0)} \subset V^{(1)} \subset V^{(2)} \subset \cdots \subset V^{(n)}$. The basis functions of $V^{(k)}$ are called *scaling functions* at the resolution level k and denoted by $\varphi_i^{(k)}$ $(i = 1, 2, \ldots)$ in this thesis. This means that the scaling function $\varphi_i^{(k)}$ represents a finer level of details as the resolution level k increases.

Let us also define $W^{(k)}$ as the space that fills the difference between $V^{(k+1)}$ and $V^{(k)}$, i.e., $V^{(k+1)} = V^{(k)} \dotplus W^{(k)}$ where $\dotplus$ indicates the direct sum. It follows from this definition that the vector space $V^{(n)}$ is decomposed as follows: $V^{(n)} = W^{(n-1)} \dotplus \cdots \dotplus W^{(n-m)} \dotplus V^{(n-m)}$ as shown in Figure 4.2. The basis functions of $W^{(k)}$ are called the *wavelets* at the resolution level k and denoted by $\psi_i^{(k)}$ $(i = 0, 1, \ldots)$ in this thesis.

Suppose that a function $f^{(n)}(x)$ is represented by the linear sum of the scaling

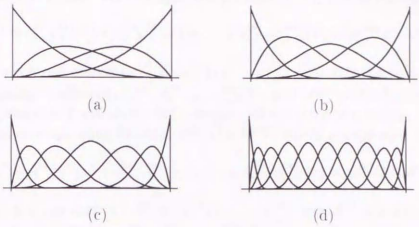


Figure 4.3: Endpoint-interpolating B-splines of the knot sequences (a) $\{0, 0, 0, 0, 1, 1, 1, 1\}$ for the resolution level 0, (b) $\{0, 0, 0, 0, \frac{1}{2}, 1, 1, 1, 1\}$ for the resolution level 1, (c) $\{0, 0, 0, 0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1, 1, 1, 1\}$ for the resolution level 2, and (d) $\{0, 0, 0, 0, \frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{5}{8}, \frac{6}{8}, \frac{7}{8}, 1, 1, 1, 1\}$ for the resolution level 3

functions $\varphi_i^{(n)}(x)$, i.e., $f^{(n)}(x) \in V^{(n)}$. Since $V^{(n)} = V^{(n-1)} \dot{+} W^{(n-1)}$, $f^{(n)}(x)$ can be decomposed into $f^{(n-1)}(\in V^{(n-1)})$ and $g^{(n-1)}(\in W^{(n-1)})$ as shown in Figure 4.2. $\psi^{(n-1)}(x)$ serves as a high-pass filter whereas $\varphi^{(n-1)}(x)$ serves as a low-pass filter. In this way, the decomposition of $f^{(n)}(x)$ at successive resolution levels produces $g^{(n-1)}(x), \ldots, g^{(n-m)}(x)$, and $f^{(n-m)}(x)$. Conversely, $f^{(n)}(x)$ can be reconstructed from $g^{(n-1)}(x), \ldots, g^{(n-m)}(x)$, and $f^{(n-m)}(x)$.

4.2.2 B-spline Wavelets

It is desirable that the space $W^{(k)}$ is orthogonal to $V^{(k)}$ in the hierarchical representations of curves and surfaces. This is because the changes of a function $f^{(k)} \in V^{(k)}$ have no influence on the corresponding function $g^{(k)} \in W^{(k)}$, which means that the functions $f^{(k)}$ and $g^{(k)}$ can be edited independently. The class of wavelets with this orthogonality is called *semi-orthogonal wavelets*.

Among the semi-orthogonal wavelets, this study uses the spline wavelets developed by Chui et al [20, 94]. The corresponding scaling functions are B-spline basis functions. In particular, this study employs the cubic endpoint-interpolating B-spline functions defined on a knot sequence that is uniformly spaced everywhere except for its ends, where its knots have the multiplicity 4. Such class of B-spline functions is shown in Figure 4.3. The reason for using this class of B-splines is that it provides the means of decomposing and reconstructing multiresolution shape functions using matrix calculations. This is explained in detail in Section 4.2.3.

Because the endpoint-interpolating B-spline functions and its corresponding wavelets are defined in bounded domains, the shape function is represented by a sum of a finite

number of basis functions. For example, the function $f^{(k)} \in V^{(k)}$ is represented by

$$f^{(k)}(x) = c_1^{(k)} \varphi_1^{(k)}(x) + c_2^{(k)} \varphi_2^{(k)}(x) + \cdots + c_{m^{(k)}}^{(k)} \varphi_{m^{(k)}}^{(k)}(x) = \boldsymbol{\varphi}^{(k)}(x) \cdot \boldsymbol{c}^{(k)}, \tag{4.1}$$

where $\boldsymbol{\varphi}^{(k)}(x)$ is a row matrix $(\varphi_1^{(k)}(x), \varphi_2^{(k)}(x), \ldots, \varphi_{m^{(k)}}^{(k)}(x))$, $\boldsymbol{c}^{(k)}$ is a column matrix of the corresponding coefficients $(c_1^{(k)}, c_2^{(k)}, \ldots, c_{m^{(k)}}^{(k)})^T$, and $m^{(k)}$ is the dimension of $V^{(k)}$. Here, the superscript T stands for the transpose of a matrix or a vector. The function $f^{(k)}(x)$ has its corresponding function $g^{(k)}(x) \in W^{(k)}$, which is expressed by

$$g^{(k)}(x) = d_1^{(k)} \psi_1^{(k)}(x) + d_2^{(k)} \psi_2^{(k)}(x) + \cdots + d_{n^{(k)}}^{(k)} \psi_{n^{(k)}}^{(k)}(x) = \boldsymbol{\psi}^{(k)}(x) \cdot \boldsymbol{d}^{(k)}, \tag{4.2}$$

where $\boldsymbol{\psi}^{(k)}(x)$ is a row matrix $(\psi_1^{(k)}(x), \psi_2^{(k)}(x), \ldots, \psi_{n^{(k)}}^{(k)}(x))$, $\boldsymbol{d}^{(k)}$ is a column matrix of the corresponding coefficients $(d_1^{(k)}, d_2^{(k)}, \ldots, d_{n^{(k)}}^{(k)})^T$, and $n^{(k)}$ is the dimension of $W^{(k)}$.

As described in Section 4.2.1, the subspaces $V^{(k)}$ are nested recursively. This leads to the fact that for all k there exists a matrix $\boldsymbol{P}^{(k)}$ such that

$$\boldsymbol{\varphi}^{(k-1)}(x) = \boldsymbol{\varphi}^{(k)}(x) \cdot \boldsymbol{P}^{(k)}. \tag{4.3}$$

It means that a scaling function at the resolution level $k-1$ can be represented by a linear sum of the scaling functions at the resolution level k. Note that the matrix $\boldsymbol{P}^{(k)}$ is an $m^{(k)} \times m^{(k-1)}$ matrix because $V^{(k)}$ and $V^{(k-1)}$ have dimensions $m^{(k)}$ and $m^{(k-1)}$ respectively. Since $W^{(k-1)}$ is also contained in $V^{(k)}$, there exists a matrix $\boldsymbol{Q}^{(k)}$ such that

$$\boldsymbol{\psi}^{(k-1)}(x) = \boldsymbol{\varphi}^{(k)}(x) \cdot \boldsymbol{Q}^{(k)}, \tag{4.4}$$

where $\boldsymbol{Q}^{(k)}$ is an $m^{(k)} \times n^{(k-1)}$ matrix. Note that $m^{(k-1)} + n^{(k-1)} = m^{(k)}$ because $V^{(k-1)}$ and $W^{(k-1)}$ are orthogonal in $V^{(k)}$. The equations (4.3) and (4.4) are said to be *two-scale relations* for $\boldsymbol{\varphi}^{(k)}(x)$ and $\boldsymbol{\psi}^{(k)}(x)$, and the matrices $\boldsymbol{P}^{(k)}$ and $\boldsymbol{Q}^{(k)}$ are called *synthesis filters*. The matrices $\boldsymbol{P}^{(k)}$ and $\boldsymbol{Q}^{(k)}$ are given in Appendix C.

4.2.3 Wavelet Decomposition and Reconstruction

As described above, the function $f^{(k)} \in V^{(k)}$ is represented by $\boldsymbol{\varphi}^{(k)}\boldsymbol{c}^{(k)}$. Note that the coefficient matrix $\boldsymbol{c}^{(k)}$ can be regarded as the control points of $f^{(k)}$. To create lower resolution coefficients $\boldsymbol{c}^{(k-1)}$ from $\boldsymbol{c}^{(k)}$ with a fewer number of coefficients $m^{(k-1)} (< m^{(k)})$, the following calculation is performed:

$$\boldsymbol{c}^{(k-1)} = \boldsymbol{A}^{(k)} \boldsymbol{c}^{(k)}, \tag{4.5}$$

where $\boldsymbol{A}^{(k)}$ is an $m^{(k-1)} \times m^{(k)}$ matrix. Since $m^{(k-1)} < m^{(k)}$, some details are lost in this process. The details are stored in $\boldsymbol{d}^{(k-1)}$ by the following calculation:

$$\boldsymbol{d}^{(k-1)} = \boldsymbol{B}^{(k)} \boldsymbol{c}^{(k)}, \tag{4.6}$$

where $\boldsymbol{B}^{(k)}$ is an $n^{(k-1)} \times m^{(k)}$ matrix. The process of obtaining the coefficients at a lower resolution level $\boldsymbol{c}^{(k-1)}$ and $\boldsymbol{d}^{(k-1)}$ from $\boldsymbol{c}^{(k)}$ is called the *wavelet decomposition*, and the

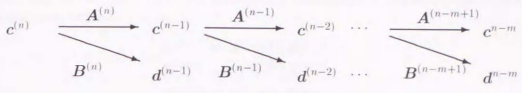


Figure 4.4: The filter bank

matrices $\boldsymbol{A}^{(k)}$ and $\boldsymbol{B}^{(k)}$ are called *analysis filters*. Conversely, $\boldsymbol{c}^{(k)}$ can be reconstructed from the lower resolution coefficients $\boldsymbol{c}^{(k-1)}$ and $\boldsymbol{d}^{(k-1)}$ as follows:

$$\boldsymbol{c}^{(k)} = \boldsymbol{P}^{(k)}\boldsymbol{c}^{(k-1)} + \boldsymbol{Q}^{(k)}\boldsymbol{d}^{(k-1)}. \tag{4.7}$$

Note that the matrices $\boldsymbol{P}^{(k)}$ and $\boldsymbol{Q}^{(k)}$ are the same as those in (4.3) and (4.4). This process of recovering the original function from the lower level of resolution is called the *wavelet reconstruction*. According to (4.5), (4.6), and (4.7), the following relation between $\boldsymbol{A}^{(k)}$, $\boldsymbol{B}^{(k)}$, $\boldsymbol{P}^{(k)}$, and $\boldsymbol{Q}^{(k)}$ is derived:

$$\left(\begin{array}{c} \boldsymbol{A}^{(k)} \\ \hline \boldsymbol{B}^{(k)} \end{array}\right) = \left(\boldsymbol{P}^{(k)} \,|\, \boldsymbol{Q}^{(k)} \right)^{-1}. \tag{4.8}$$

Here, $\left(\begin{array}{c} \boldsymbol{A}^{(k)} \\ \hline \boldsymbol{B}^{(k)} \end{array}\right)$ and $\left(\boldsymbol{P}^{(k)} \,|\, \boldsymbol{Q}^{(k)} \right)$ are both square matrices.

(4.5) and (4.6) allow us to recursively decompose $\boldsymbol{c}^{(n)}$ into $\boldsymbol{d}^{(n-1)}, \ldots, \boldsymbol{d}^{(n-m)}$, and $\boldsymbol{c}^{(n-m)}$ as shown in Figure 4.4. This recursive process is called a *filter bank*. Since $\boldsymbol{c}^{(n)}$ can be reconstructed from $\boldsymbol{d}^{(n-1)}, \ldots, \boldsymbol{d}^{(n-m)}$, and $\boldsymbol{c}^{(n-m)}$ using (4.7), this decomposed sequence can be considered as a transform of the original coefficients $\boldsymbol{c}^{(n)}$ and is known as a *wavelet transform*.

The above formulation can be directly applied to the multiresolution representations of curves because the scaling functions are B-spline basis functions. In the case of surfaces, the basis functions are usually tensor-products of B-spline functions. Let us define several matrices using scaling functions $\varphi_i^{(k)}$ and the wavelets $\psi_i^{(k)}$ as follows: $\boldsymbol{\Phi}^{(k)}(x,y) = (\varphi_i^{(k)}(x) \cdot \varphi_j^{(k)}(y))$, $\boldsymbol{\Psi}_x^{(k)}(x,y) = (\psi_i^{(k)}(x) \cdot \varphi_j^{(k)}(y))$, $\boldsymbol{\Psi}_y^{(k)}(x,y) = (\varphi_i^{(k)}(x) \cdot \psi_j^{(k)}(y))$, and $\boldsymbol{\Psi}_{xy}^{(k)}(x,y) = (\psi_i^{(k)}(x) \cdot \psi_j^{(k)}(y))$. Here, the element in the i-th row and j-th column of each matrix is represented. The corresponding coefficient matrices are also defined as follows: $\boldsymbol{C}^{(k)} = (c_i^{(k)} \cdot c_j^{(k)})$, $\boldsymbol{D}_x^{(k)} = (d_i^{(k)} \cdot c_j^{(k)})$, $\boldsymbol{D}_y^{(k)} = (c_i^{(k)} \cdot d_j^{(k)})$, and $\boldsymbol{D}_{xy}^{(k)} = (c_i^{(k)} \cdot d_j^{(k)})$. Suppose that $M_{\mathrm{row-major}}$ is the row-major representation of the matrix $M = (m_{ij})$, i.e., $M_{\mathrm{row-major}} = (m_{11}, m_{12}, \ldots, m_{21}, m_{22} \ldots)$. It is noted that the surface $\boldsymbol{S}(x,y)$ is represented by $\boldsymbol{S}(x,y) = \boldsymbol{\Phi}_{\mathrm{row-major}}^{(k)}(x,y)(\boldsymbol{C}_{\mathrm{row-major}}^{(k)})^{\mathrm{T}}$.

According to (4.5) and (4.6), the following equations for the wavelet decomposition of surfaces are derived.

$$\boldsymbol{C}^{(k-1)} = \boldsymbol{A}^{(k)}\boldsymbol{c}^{(k)}(\boldsymbol{A}^{(k)})^{\mathrm{T}} \tag{4.9}$$

$$\boldsymbol{D}_{x}^{(k-1)} = \boldsymbol{B}^{(k)}\boldsymbol{c}^{(k)}(\boldsymbol{A}^{(k)})^{\mathrm{T}} \tag{4.10}$$

$$\boldsymbol{D}_{y}^{(k-1)} = \boldsymbol{A}^{(k)}\boldsymbol{c}^{(k)}(\boldsymbol{B}^{(k)})^{\mathrm{T}} \tag{4.11}$$

$$\boldsymbol{D}_{xy}^{(k-1)} = \boldsymbol{B}^{(k)}\boldsymbol{c}^{(k)}(\boldsymbol{B}^{(k)})^{\mathrm{T}} \tag{4.12}$$

In the same way as for the curves, the following equation for the wavelet reconstruction of surfaces is derived.

$$\boldsymbol{C}^{(k)} = \boldsymbol{P}^{(k)}\boldsymbol{C}^{(k-1)}(\boldsymbol{P}^{(k)})^{\mathrm{T}} + \boldsymbol{Q}^{(k)}\boldsymbol{D}_{x}^{(k-1)}(\boldsymbol{P}^{(k)})^{\mathrm{T}} + \boldsymbol{P}^{(k)}\boldsymbol{D}_{y}^{(k-1)}(\boldsymbol{Q}^{(k)})^{\mathrm{T}} + \boldsymbol{Q}^{(k)}\boldsymbol{D}_{xy}^{(k-1)}(\boldsymbol{Q}^{(k)})^{\mathrm{T}} \tag{4.13}$$

4.3 Designing Shapes by Variational Optimization

Welch and Witkin [135] proposed variational optimization techniques that effectively control object shapes with the geometric constraints such as points, tangents, curves, areas, etc. In their method, an object shape is determined by minimizing the energy function subject to the deformation of the shape while satisfying the given constraints.

4.3.1 Energy Functions

The energy functions for designing curves and surfaces are defined so that they can measure how much the curves and surfaces are stretched and bent by looking at the differentials and curvatures at each point of the curves and surfaces.

Suppose a curve is represented by $\boldsymbol{w}_c(x) = \boldsymbol{\varphi}_c(x) \cdot \boldsymbol{c}_c$, where $\boldsymbol{\varphi}_c(x)$ represents a row matrix of B-spline basis functions and $\boldsymbol{c}_c$ represents a column matrix of its corresponding coefficients. Here, $\boldsymbol{c}_c$ indicates a vector of control points in two-dimensional (2D) or three-dimensional (3D) space. The energy function $E_c(\boldsymbol{w}_c)$ of the curve $\boldsymbol{w}_c$ is defined as

$$E_c(\boldsymbol{w}_c) = \int_{\boldsymbol{w}_c} \alpha_c(D\boldsymbol{w}_c)^2 + \beta_c(DD\boldsymbol{w}_c)^2 = \boldsymbol{c}_c{}^{\mathrm{T}}\boldsymbol{H}_c\boldsymbol{c}_c, \tag{4.14}$$

where $D\boldsymbol{w}_c$ indicates the derivative of the curve $\boldsymbol{w}_c(x)$ with respect to x. The first and second terms of (4.14) correspond to the stretching and bending of the curve, which are controlled by the values of α_c and β_c, respectively. Because the above energy function is the integration with respect to x, the control vector $\boldsymbol{c}_c$ can be brought outside the integration, and the integration is reduced to the matrix $\boldsymbol{H}_c$.

Suppose a surface is represented by $\boldsymbol{w}_s(x, y) = \boldsymbol{\varphi}_s(x, y) \cdot \boldsymbol{c}_s$, where $\boldsymbol{\varphi}_s(x, y)$ represents a row-major matrix of the tensor-products of B-spline functions and $\boldsymbol{c}_s$ represents a row-major matrix of the corresponding coefficients as described in Section 4.2.3. Here, $\boldsymbol{c}_s$ represents a vector of control points in 3D space. The energy function $E_s(\boldsymbol{w}_s)$ of the surface $\boldsymbol{w}_s$ is defined as

$$E_s(\boldsymbol{w}_s) = \int_{\boldsymbol{w}_s} \sum_{i,j=x,y} \alpha_s D_i\boldsymbol{w}_s D_j\boldsymbol{w}_s + \beta_s(D_iD_j\boldsymbol{w}_s)^2 = \boldsymbol{c}_s^{\mathrm{T}}\boldsymbol{H}_s\boldsymbol{c}_s, \tag{4.15}$$

where $D_k \boldsymbol{w}_s$ indicates the partial derivative of the surface $\boldsymbol{w}_s(x, y)$ with respect to k, and the values of α_s and β_s control the stretching and bending of the surface. In the same way as for the curves, the control vector $\boldsymbol{c}_s$ can be brought outside the integration, and the integration with respect to x and y is reduced to the matrix $\boldsymbol{H}_s$.

The above energy function is based on the thin-plate under tension model [124, 15]. Although this model is accurate only in the neighborhood of the minimum, it still behaves well away from the minimum, which is effective for designing curves and surfaces. Note that each of the coordinates such as x, y, and z is treated independently in the above formulations.

4.3.2 Attaching Geometric Constraints

In order to represent the curves and surfaces by the same notations, the subscripts c for curves and s for surfaces are omitted in the notations such as $\boldsymbol{\varphi}_c$, $\boldsymbol{\varphi}_s$, $\boldsymbol{c}_c$, $\boldsymbol{c}_s$, $\boldsymbol{H}_c$, and $\boldsymbol{H}_s$ in what follows. The geometric constraints are points, tangents, curves, areas, etc. According to the study of Welch and Witkin, these geometric constraints are grouped into two classes: the *finite-dimensional constraints* which control the shapes of curves and surfaces at discrete points, and the *transfinite constraints* which control the shapes along curves or subareas. Both classes of constraints can be formulated as a system of linear equations with respect to the control vector $\boldsymbol{c}$.

The finite-dimensional constraints fix the shapes of curves and surfaces at specified points. The position and tangent at a point are the examples of such constraints. For example, the point-position constraint that fixes the shape of the parameter $\boldsymbol{x}_0$ at a point $\boldsymbol{w}_0$ can be written as

$$\boldsymbol{w}_0 = \boldsymbol{w}(\boldsymbol{x}_0) = \boldsymbol{\varphi}(\boldsymbol{x}_0) \cdot \boldsymbol{c}. \tag{4.16}$$

It is a linear equation with respect to $\boldsymbol{c}$. Other constraints such as tangents at points can similarly be described.

The transfinite constraints need more complicated formulations. Since the curves and surfaces have only finite number of control points, they cannot exactly satisfy the transfinite constraints in general. Therefore, such constraints are formulated as integrals over the parametric domains of the constraints and the integrals are minimized in a least-square sense. For example, consider a constraint curve whose parameters are represented by $\boldsymbol{l}(t)$. This means that the constraint curve $\boldsymbol{L}$ can be represented by $\boldsymbol{L}(t) = \boldsymbol{L}(\boldsymbol{l}(t))$. The curve $\boldsymbol{N}(t)$ on the object shape that corresponds to $\boldsymbol{L}(t)$ is written as $\boldsymbol{N}(t) = \boldsymbol{w}(\boldsymbol{l}(t)) = \boldsymbol{\varphi}(\boldsymbol{l}(t)) \cdot \boldsymbol{c}$. The integral over the parametric range of t can be obtained as follows:

$$\int_{\boldsymbol{l}} (\boldsymbol{N} - \boldsymbol{L})^2. \tag{4.17}$$

Welch and Witkin formulate the transfinite constraint so that (4.17) becomes minimum by setting the derivative of (4.17) with respect to $\boldsymbol{c}$ to zero as

$$\begin{aligned} 0 &= \frac{\partial}{\partial \boldsymbol{c}} \int_{\boldsymbol{l}} (\boldsymbol{N} - \boldsymbol{L})^2 \\ &= \int_{\boldsymbol{l}} (\boldsymbol{N} - \boldsymbol{L}) \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{c}} \end{aligned}$$

$$
\begin{aligned}
&= \int_l (\boldsymbol{w}(l) - \boldsymbol{L}(l)) \frac{\partial(\boldsymbol{w}(l))}{\partial \boldsymbol{c}} \\
&= \int_l (\boldsymbol{\varphi}(l)\boldsymbol{c} - \boldsymbol{L}(l)) \frac{\partial(\boldsymbol{\varphi}(l)\boldsymbol{c})}{\partial \boldsymbol{c}} \\
&= \boldsymbol{c} \int_l \boldsymbol{\varphi}(l) \otimes \boldsymbol{\varphi}(l) - \int_l \boldsymbol{L}(l)\boldsymbol{\varphi}(l),
\end{aligned} \tag{4.18}
$$

where $\otimes$ denotes the tensor product. Note that the transfinite constraints are also reduced to a system of linear equations with respect to $\boldsymbol{c}$. Other constraints such as areas are similarly described.

In this way, both classes of the constraints are finally written in the following form:

$$
\boldsymbol{M}\boldsymbol{c} = \boldsymbol{q}, \tag{4.19}
$$

where each row of the matrix $\boldsymbol{M}$ represents a single linear constraint and the corresponding component of $\boldsymbol{q}$ represents its value.

4.3.3 Constrained Variational Optimization

As described in Sections 4.3.1 and 4.3.2, we have already obtained the energy function and the equations of constraints. The next step is to find the coefficient vector $\boldsymbol{c}$ by minimizing the energy function while satisfying the given constraints. This can be written as follows.

$$
\min \| \frac{1}{2}\boldsymbol{c}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{c} \| \quad \boldsymbol{c} \text{ is subject to } \boldsymbol{M}\boldsymbol{c} = \boldsymbol{q} \tag{4.20}
$$

One of the general solutions to this problem is to reformulate (4.20) by adding a term of Lagrange multipliers $\boldsymbol{y} = (y_1, y_2, \ldots)^{\mathrm{T}}$ and minimize

$$
\| \frac{1}{2}\boldsymbol{c}^{T}\boldsymbol{H}\boldsymbol{c} + (\boldsymbol{M}\boldsymbol{c} - \boldsymbol{q})^{T}\boldsymbol{y} \| . \tag{4.21}
$$

Considering the differentiation of (4.21) with respect to $\boldsymbol{c}$ and $\boldsymbol{y}$, the object shape is determined by solving the following system of equations:

$$
\begin{pmatrix} \boldsymbol{H} & \boldsymbol{M}^{T} \\ \boldsymbol{M} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} 0 \\ \boldsymbol{q} \end{pmatrix}. \tag{4.22}
$$

In this way, the coefficient vector $\boldsymbol{c}$ that minimizes the energy function while satisfying the given constraints is obtained. In calculating the solutions of (4.22), numerical errors arise when the matrix is close to singular. For such cases, the techniques called *singular value decomposition* [93], which are described in Appendix D in detail, is used. The techniques described in this section allow us to design the shapes of curves and surfaces at a single resolution level by attaching geometric constraints. Thus, the next step is to design the object shape by solving constraints at multiresolution levels.

4.4 Designing Shapes Using Multiresolution Constraints

This section describes the main contribution of this chapter: a method of solving constraints at multiresolution levels for the hierarchical curve and surface design.

4.4.1 Converting Constraints at Different Resolution Levels

The problem of handling multiresolution constraints is that the constraints at each resolution level are associated with the basis functions at its resolution level. In other words, constraints at different resolution levels cannot be handled directly because they have different basis functions. Therefore, it is necessary to associate the constraints at multiresolution levels with the common basis functions. Fortunately, this is possible in this framework since the constraints are reduced to a system of linear equations with respect to the control points. This is the main idea of the method to be presented.

Let $\boldsymbol{M}^{(k)}\boldsymbol{c}^{(k)} = \boldsymbol{q}^{(k)}$ be the equation of the constraints at the resolution level k. Recall that the analysis filter provides the relation between the vectors $\boldsymbol{c}^{(k-1)}$ and $\boldsymbol{c}^{(k)}$ as shown in (4.5) (for curves) and (4.9) (for surfaces), which is described as

$$\boldsymbol{c}^{(k-1)} = \boldsymbol{F}^{(k)}\boldsymbol{c}^{(k)}. \tag{4.23}$$

It means that $\boldsymbol{c}^{(k-1)}$ can be represented by the linear sum of the elements of $\boldsymbol{c}^{(k)}$. In particular, in the case of curves, $\boldsymbol{F}^{(k)}$ is equivalent to $\boldsymbol{A}^{(k)}$ as shown in (4.5). In the case of surfaces, (4.9) is converted into the form of (4.23) using the row-major representation of matrices because $\boldsymbol{c}^{(k-1)}$ is also linear with respect to the elements of $\boldsymbol{c}^{(k)}$. Let us confirm this. According to (4.9), the relation between $\boldsymbol{c}^{(k-1)}$ and $\boldsymbol{c}^{(k)}$ is written as

$$\boldsymbol{c}^{(k-1)} = \boldsymbol{A}^{(k)}\boldsymbol{c}^{(k)}(\boldsymbol{A}^{(k)})^{\mathrm{T}}. \tag{4.24}$$

This is equivalent to the following equation:

$$\begin{aligned} c_{ij}^{(k-1)} &= \sum_{p=1}^{m^{(k)}} a_{ip}\Big(\sum_{q=1}^{m^{(k)}} c_{pq}a_{jq}\Big) \\ &= \sum_{p=1}^{m^{(k)}}\sum_{q=1}^{m^{(k)}} a_{ip}a_{jq}c_{pq} \end{aligned}$$

The row-major representation of the above equation is represented by

$$\begin{pmatrix} c_{11}^{(k-1)} \\ \vdots \\ c_{1\nu}^{(k-1)} \\ c_{21}^{(k-1)} \\ \vdots \\ c_{\nu\nu}^{(k-1)} \end{pmatrix} = \begin{pmatrix} a_{11}a_{11} & \cdots & a_{11}a_{1\mu} & a_{12}a_{11} & \cdots & a_{1\mu}a_{1\mu} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{11}a_{\nu 1} & \cdots & a_{11}a_{\nu\mu} & a_{12}a_{\nu 1} & \cdots & a_{1\mu}a_{\nu\mu} \\ a_{21}a_{11} & \cdots & a_{21}a_{1\mu} & a_{22}a_{11} & \cdots & a_{2\mu}a_{1\mu} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{\nu 1}a_{\nu 1} & \cdots & a_{\nu 1}a_{\nu\mu} & a_{\nu 2}a_{\nu 1} & \cdots & a_{\nu\mu}a_{\nu\mu} \end{pmatrix} \begin{pmatrix} c_{11}^{(k)} \\ \vdots \\ c_{1\mu}^{(k)} \\ c_{21}^{(k)} \\ \vdots \\ c_{\mu\mu}^{(k)} \end{pmatrix}, \tag{4.25}$$

where $\mu = m^{(k)}$ and $\nu = m^{(k-1)}$. In this way, (4.24) can be reduced to the form of (4.23).

It is now possible to associate the constraints at different resolution levels with the basis functions at a single resolution level. Suppose that there are equations of constraints $\{\boldsymbol{M}^{(k)}\boldsymbol{c}^{(k)} = \boldsymbol{q}^{(k)}\}$ at the resolution levels $k = 0, 1, \ldots n$ as shown in Figure 4.5. The first step is to associate the equation of constraints $\boldsymbol{M}^{(0)}\boldsymbol{c}^{(0)} = \boldsymbol{q}^{(0)}$ with the basis functions at the resolution level 1. By substituting $\boldsymbol{c}^{(0)}$ for $\boldsymbol{F}^{(1)}\boldsymbol{c}^{(1)}$ using (4.23), the equation becomes $\boldsymbol{M}^{(0)}(\boldsymbol{F}^{(1)}\boldsymbol{c}^{(1)}) = (\boldsymbol{M}^{(0)}\boldsymbol{F}^{(1)})\boldsymbol{c}^{(1)} = \boldsymbol{q}^{(0)}$, which relates the constraints at the resolution level 0 to the basis functions at the resolution level 1. At the resolution level 1, there are two systems of equations: the original equation of constraints $\boldsymbol{M}^{(1)}\boldsymbol{c}^{(1)} = \boldsymbol{q}^{(1)}$, and the newly converted equation $(\boldsymbol{M}^{(0)}\boldsymbol{F}^{(1)})\boldsymbol{c}^{(1)} = \boldsymbol{q}^{(0)}$. The two systems of equations can be merged into $\widetilde{\boldsymbol{M}}^{(1)}\boldsymbol{c}^{(1)} = \widetilde{\boldsymbol{q}}^{(1)}$ as shown in Figure 4.5, because both systems are based on the same basis functions, i.e., the basis functions at the resolution level 1. The merged equation is converted to those at higher resolution levels in the same way. As shown in Figure 4.5, this process continues until all the constraints of multiresolution levels are merged.

4.4.2 Solving Multiresolution Constraints

As seen in Section 4.4.1, the framework of this study enables us to associate all the constraints at different resolution levels with the basis functions at the highest resolution level. Although the converted set of equations can be solved at the highest resolution level, an unexpected side effect will appear in this case. Figure 4.6 shows an example where the object shape is tangled. The cause of this unexpected result lies in the fact that the deformation is minimized only at the highest resolution level, although the constraints are specified at each level of resolution.

To avoid such an unexpected side effect, this study applies the Welch-Witkin techniques to the difference between object shapes at adjacent resolution levels. Here, the difference is equivalent to the 3D vector between the points that have the same parameter values of the object shape at each resolution level. Note that the difference vector is determined by handling x-, y-, and z-coordinates independently. This approach is similar to the Forsey-Wong method [33] for generating hierarchical B-spline surfaces in that both methods use least squares to control the hierarchical representations of the shapes.

The actual process is performed recursively from low to high resolution levels. First, we determine a temporary shape at the lowest resolution level directly using the Welch-Witkin techniques. This means that the shape is determined by solving the equation of constraints imposed only at this level. Note that the shape will be modified when we determine the shapes at higher resolution levels.

After determining a temporary shape at the lowest resolution level, we determine the shapes at higher resolution levels step by step. Let us now assume that the object shapes at resolution levels up to $n-1$ have been determined as shown in Figure 4.7. The next stage is to determine the object shape at the resolution level n. Suppose $\boldsymbol{w}_{\text{old}}^{(k)}$ $(0 \leq k \leq n-1)$ is the current shape of the object at the resolution level k and $\boldsymbol{w}_{\text{new}}^{(k)}$

resolution level n	 $\boldsymbol{M}^{(n)}\boldsymbol{c}^{(n)} = \boldsymbol{q}^{(n)}$	$\rightarrow$ $\widetilde{\boldsymbol{M}}^{(n)}\boldsymbol{c}^{(n)} = \widetilde{\boldsymbol{q}}^{(n)}$ $\uparrow$ $\begin{pmatrix} \boldsymbol{M}^{(n)} \\ \widetilde{\boldsymbol{M}}^{(n-1)}\boldsymbol{F}^{(n)} \end{pmatrix}\boldsymbol{c}^{(n)} = \begin{pmatrix} \boldsymbol{q}^{(n)} \\ \widetilde{\boldsymbol{q}}^{(n-1)} \end{pmatrix}$
	conversion	$(\widetilde{\boldsymbol{M}}^{(n-1)}\boldsymbol{F}^{(n)})\boldsymbol{c}^{(n)} = \widetilde{\boldsymbol{q}}^{(n-1)}$
	$\vdots$	$\vdots$
	conversion	$(\widetilde{\boldsymbol{M}}^{(1)}\boldsymbol{F}^{(2)})\boldsymbol{c}^{(2)} = \widetilde{\boldsymbol{q}}^{(1)}$
resolution level 1	 $\boldsymbol{M}^{(1)}\boldsymbol{c}^{(1)} = \boldsymbol{q}^{(1)}$	$\rightarrow$ $\widetilde{\boldsymbol{M}}^{(1)}\boldsymbol{c}^{(1)} = \widetilde{\boldsymbol{q}}^{(1)}$ $\uparrow$ $\begin{pmatrix} \boldsymbol{M}^{(1)} \\ \boldsymbol{M}^{(0)}\boldsymbol{F}^{(1)} \end{pmatrix}\boldsymbol{c}^{(1)} = \begin{pmatrix} \boldsymbol{q}^{(1)} \\ \boldsymbol{q}^{(0)} \end{pmatrix}$
	conversion	$(\boldsymbol{M}^{(0)}\boldsymbol{F}^{(1)})\boldsymbol{c}^{(1)} = \boldsymbol{q}^{(0)}$
resolution level 0	 $\boldsymbol{M}^{(0)}\boldsymbol{c}^{(0)} = \boldsymbol{q}^{(0)}$	$\rightarrow$ $\boldsymbol{M}^{(0)}\boldsymbol{c}^{(0)} = \boldsymbol{q}^{(0)}$

Figure 4.5: Converting constraints from low to high resolution levels: The arrows indicate the steps of converting constraints.

(a) (b)

Figure 4.6: The case where the multiresolution constraints are solved directly: The figure on the left shows a curve at a low resolution level and the figure on the right shows its corresponding curve at a high-resolution level. Point-position constraints are represented by gray disks.

$(0 \leq k \leq n)$ is the shape of the object to be obtained after this process at the resolution level k. Note that the current shape at the resolution level k $(0 \leq k \leq n-1)$ will be modified while satisfying the given constraints.

The equation of constraints at the resolution level k $(0 \leq k \leq n-1)$ is obtained by taking into account the difference between constraints and current object shape. For example, a point-position constraint that fixes the shape of the parameter $\boldsymbol{x}_0$ at the point $\boldsymbol{w}_0$ can be written as

$$\boldsymbol{w}_{\text{new}}^{(k)}(\boldsymbol{x}_0) = \boldsymbol{w}_0 - \boldsymbol{w}_{\text{old}}^{(k)}(\boldsymbol{x}_0). \tag{4.26}$$

Other constraints such as curves and subareas can similarly be obtained. The equation of constraints obtained here is denoted by $\widehat{\boldsymbol{M}}^{(k)}\boldsymbol{c}^{(k)} = \widehat{\boldsymbol{q}}^{(k)}$ as shown in Figure 4.7. At the resolution level n, on the other hand, $\boldsymbol{w}_{\text{old}}^{(n-1)}$ is used instead of $\boldsymbol{w}_{\text{old}}^{(n)}$ because we do not have $\boldsymbol{w}_{\text{old}}^{(n)}$ yet. This amounts to modifying the current shape at the resolution level $n-1$ to find the next shape at the resolution level n. Using the techniques described in Section 4.4.1, the equations of constraints $\{\widehat{\boldsymbol{M}}^{(k)}\boldsymbol{c}^{(k)} = \widehat{\boldsymbol{q}^{(k)}}\}$ are merged where k ranges from 0 to n. By solving the merged constraints, we now obtain the difference between the object shapes at the resolution levels n and $n-1$, i.e., $\boldsymbol{w}_{\text{new}}^{(n)} - \boldsymbol{w}_{\text{old}}^{(n-1)}$. $\boldsymbol{w}_{\text{new}}^{(n)}$ is now obtained by adding the resultant difference $\boldsymbol{w}_{\text{new}}^{(n)} - \boldsymbol{w}_{\text{old}}^{(n-1)}$ to $\boldsymbol{w}_{\text{old}}^{(n-1)}$, and then propagate the changes by decomposing $\boldsymbol{w}_{\text{new}}^{(n)}$ into those at the lower resolution levels as illustrated in Figure 4.7.

Figure 4.8 shows the examples where the same constraints as those of Figure 4.6 are solved using the method. In this way, the proposed method enables us to design curves and surfaces with constraints at resolution levels that range from coarse to fine.

4.5 Results

This section shows several design examples generated using the prototype system. In the implementation, users attach constraints at multiple resolution levels as shown in Figures 4.11 and 4.12.

resolution level n	? –  $\boldsymbol{w}_{\text{new}}^{(n)} - \boldsymbol{w}_{\text{old}}^{(n-1)}$	$\rightarrow$	$\widehat{\boldsymbol{M}}^{(n)}\boldsymbol{c}^{(n)} = \widehat{\boldsymbol{q}}^{(n)}$ $\longrightarrow$ find $\boldsymbol{w}_{\text{new}}^{(n)}$	$\Longrightarrow$	$\boldsymbol{w}_{\text{new}}^{(n)}$
			$\Uparrow$ merge		$\Downarrow$ decompose
resolution level $n-1$	? –  $\boldsymbol{w}_{\text{new}}^{(n-1)} - \boldsymbol{w}_{\text{old}}^{(n-1)}$	$\rightarrow$	$\widehat{\boldsymbol{M}}^{(n-1)}\boldsymbol{c}^{(n-1)} = \widehat{\boldsymbol{q}}^{(n-1)}$		$\boldsymbol{w}_{\text{new}}^{(n-1)}$
			$\Uparrow$ merge		$\Downarrow$ decompose
	$\vdots$		$\vdots$		$\vdots$
			$\Uparrow$ merge		$\Downarrow$ decompose
resolution level 1	? –  $\boldsymbol{w}_{\text{new}}^{(1)} - \boldsymbol{w}_{\text{old}}^{(1)}$	$\rightarrow$	$\widehat{\boldsymbol{M}}^{(1)}\boldsymbol{c}^{(1)} = \widehat{\boldsymbol{q}}^{(1)}$		$\boldsymbol{w}_{\text{new}}^{(1)}$
			$\Uparrow$ merge		$\Downarrow$ decompose
resolution level 0	? –  $\boldsymbol{w}_{\text{new}}^{(0)} - \boldsymbol{w}_{\text{old}}^{(0)}$	$\rightarrow$	$\widehat{\boldsymbol{M}}^{(0)}\boldsymbol{c}^{(0)} = \widehat{\boldsymbol{q}}^{(0)}$		$\boldsymbol{w}_{\text{new}}^{(0)}$

Figure 4.7: Solving multiresolution constraints: The arrows indicate the steps of solving multiresolution constraints.

(a) (b)

Figure 4.8: The case where the multiresolution constraints are solved using the proposed method: The figure on the left shows a curve at a low resolution and the figure on the right shows its corresponding curve at a high-resolution level. Point-position constraints are represented by gray disks.

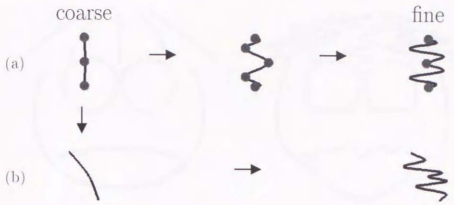


Figure 4.9: Editing a curve using multiresolution constraints: (a) designing curves using the proposed method and (b) attaching the details of (a) to another coarse shape

Designing curves

Figure 4.9 shows curves designed using multiresolution constraints. As seen in Figure 4.9(a), the system controls curves at multiresolution levels at the same time using the multiresolution constraints. The method can also be used to design the details at high resolution levels, which is equivalent to creating the data for the curve character library [31] as described in Section 4.1. Figure 4.9(b) presents such an example where the overall shape of the curve is designed and the details are then added.

Figure 4.10 shows line drawings of a young boy's face; Figure 4.10(a) shows the face at a low resolution level and Figure 4.10(b) shows its corresponding drawing at a high resolution level.

Designing surface patches

Figure 4.11 shows a display example of editing a mountain-like surface with a crater. The blue spheres represent point-position constraints and the yellow curves represent curve constraints. After creating a large mountain at a low resolution level, the small crater can be designed at a high resolution level. It can be seen in Figure 4.11 that designing shapes at multiresolution levels requires smaller number of constraints than designing shapes at a single resolution level. Figure 4.12 shows a display example of editing a surface with two peaks. As shown in Figure 4.12, a pair of steep mountains is created using multiresolution constraints. These two figures represent fundamental patterns of designing surface shapes using multiresolution constraints, which can be implemented as macro operations in the system. These fundamental operations allow us to edit more complicated shapes such as a facial shape as shown in Figure 4.13, where textures are mapped onto the surface. The figures at the top and bottom show the coarse and fine resolution surfaces, respectively.

(a) (b)

Figure 4.10: Line drawings: (a) a drawing at a low resolution level and (b) its corresponding figure at a high resolution level

Designing closed surfaces

Figure 4.14 shows a display example of editing a topological sphere using multiresolution constraints in the system of Chapter 3. As shown in Figure 4.14(a), the multiresolution constraints are imposed on the faces, which are shared by surrounding vertex patches. Figures 4.14(b), 4.14(c), 4.14(d) show the surfaces at a low resolution level, an intermediate resolution level, and a high resolution level. In this way, the proposed method can be applied to surfaces of arbitrary topological type using the manifold mappings described in Chapter 3.

4.6 Summary

This chapter has presented the method of designing curves and surfaces using multiresolution constraints. The curves and surfaces are represented in a hierarchical fashion by scaling functions and wavelets, i.e., endpoint-interpolating B-spline functions and its corresponding wavelets. The shapes of the curves and surfaces are determined by minimizing the function subject to the deformations of the shapes while preserving the given constraints. To associate the constraints at different resolution levels with the common basis functions, the constraints are converted to those at the high resolution level by the wavelet decomposition. The constraints at multiresolution levels are then solved recursively from low to high resolution levels with respect to the differences between the shapes at adjacent resolution levels.

As seen in Section 4.5, the proposed method can serve as a tool for developing the curve character library [31], which holds various kinds of fine details of the shape. Since the operations in the methods have close relations with the properties of scale-space theory [137, 66], users are expected to know some about such properties. Therefore,

macro operations based on such properties are useful for both well-trained and novice users. In this chapter, the method of designing curves and surfaces by geometric constraints is described. Implementing the reverse operations, i.e., operations for extracting constraints from the surface data, is also an important research theme. Algorithms for extracting significant constraints from the shape data are described in Chapter 5 [120], and an example of such extracted constraints is shown in Figure 4.15. Figure 4.15(a) shows the constraints of characteristic points and boundaries extracted from the surface data, and Figure 4.15(b) shows the surface determined at this resolution level from the extracted constraints.

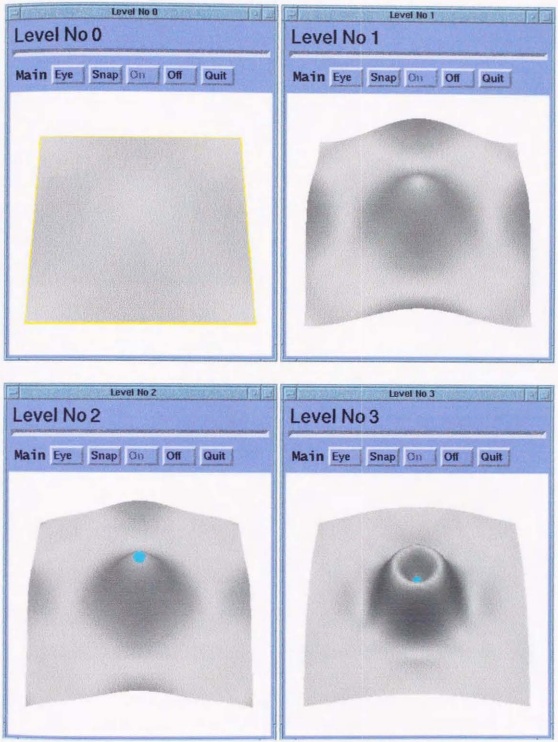


Figure 4.11: A display example of editing a mountain-like surface with a crater using multiresolution constraints; the blue spheres represent point-position constraints and the yellow curves represent curve constraints.

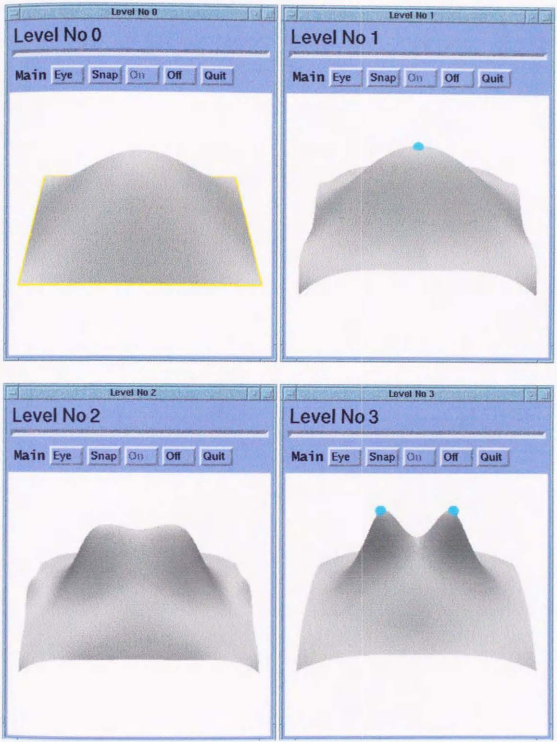


Figure 4.12: A display example of editing a surface with two peaks using multiresolution constraints; the blue spheres represent point-position constraints and the yellow curves represent curve constraints.

(a)

(b)

Figure 4.13: A display example of editing a facial shape: (a) the shape at a low resolution level and (b) its corresponding shape at a high resolution level.

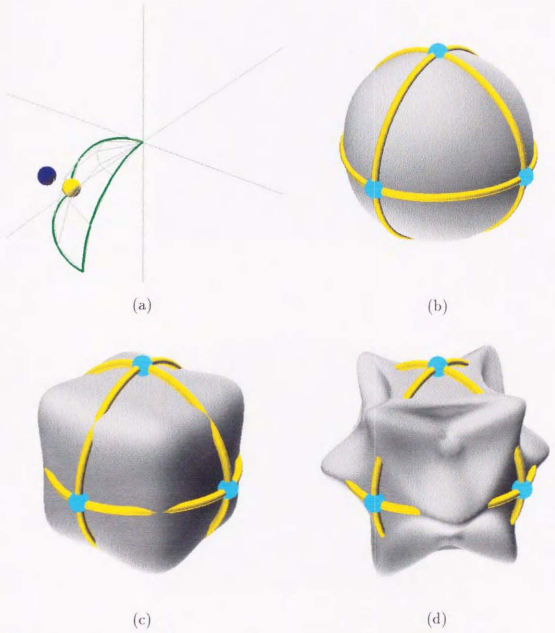
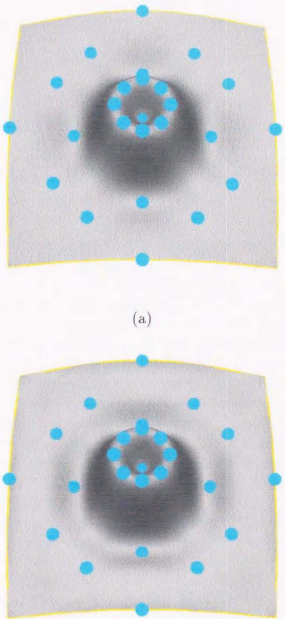


Figure 4.14: Designing a sphere using multiresolution constraints: (a) editing constraints, (b) a surface at a low resolution level, (c) a surface at an intermediate resolution level, and (d) a surface at a high resolution level



(a)

(b)

Figure 4.15: Extracting characteristic points and boundary curves from the designed surface: (a) extracted characteristic points (blue spheres) and boundary curves (yellow curves), and (b) a surface determined by the extracted characteristic points and boundary curves at this resolution level

Chapter 5

Robust Algorithms for Extracting Critical Points and Critical Point Graphs

Researchers in the fields of computer graphics, geographical information systems (GISs), and shape modeling have extensively studied the methods of extracting shape features such as peaks, pits, passes, ridges, and ravines from discrete samples. The existing techniques, however, do not guarantee the topological consistency of the extracted features because of their heuristic operations, which results in spurious features. Furthermore, there have been no robust algorithms for constructing critical point graphs (CPGs) such as the surface network and the Reeb graph from the extracted peaks, pits, and passes. This chapter presents new algorithms for extracting features and constructing CPGs from discrete samples. The algorithms enable us to extract correct shape features; i.e., the method extracts the critical points that satisfy the Euler formula, which represents the topological invariant of smooth surfaces. This chapter also provides an algorithm that converts the surface network to the Reeb graph for representing contour changes with respect to the height. The robustness of the proposed algorithms is also discussed. This chapter also describes the method of changing the height axis of object shapes using the algorithms. Display examples are presented to show that algorithms extract the features that appeal to our visual cognition.

5.1 Conventional Algorithms for Extracting Shape Features

The need to extract shape features from discrete samples has been increasing, for example, in contemporary GISs and shape modeling systems. In particular, several techniques are proposed for extracting the features of smooth surfaces such as the critical points (peaks, pits, and passes) [87, 132], ridge and ravine lines [102, 62], and surface curvatures [7, 29].

It follows from the theory of differential topology that critical points on a smooth surface satisfy the topological consistency, i.e., the Euler formula. The Euler formula represents a topological invariant of smooth surfaces. (Refer to Section 1.7 and Appendix A for more detailed definitions of these concepts.) While critical points are

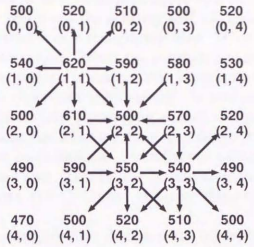
the features of differential topology, the critical points extracted using the conventional techniques do not satisfy the Euler formula. For example, let us see the critical points detected by the eight-neighbor method [87] from discrete elevation samples on a regular grid (Figure 5.1). The Euler formula states that the number of peaks $\#\{peaks\}$, the number of passes $\#\{passes\}$, and the number of pits $\#\{pits\}$ satisfy the relation $\#\{peaks\} - \#\{passes\} + \#\{pits\} = 1$ in this case. The eight-neighbor method extracts the point $(1,1)$ as a peak, the point $(2,2)$ as a pit, and the points $(3,2)$ and $(3,3)$ as passes. This result, however, does not satisfy the relation because $1 - 2 + 1 = 0 \neq 1$. This inconsistency arises from the gaps between the discrete samples and the continuous surface. The conventional techniques for extracting critical points are based on the assumption that the critical points can be extracted from the discrete samples in the same way as from the continuous one. If the sample data is dense enough, the techniques allow us to extract correct critical points. In practice, however, such data is not available because of restrictions such as the capacity of the data storage and the precision of the measurements. As a result, the techniques produce spurious critical points. Such spurious critical points also prevent us from tracing correct ridge and ravine lines for further analysis.

Topologically correct critical points enable us to analyze the geographical structures such as ridge and ravine lines among the critical points and contour changes with respect to the height. The critical point graphs (CPGs), which represent the relations among the critical points, show such structures efficiently. The surface networks [88, 89], the critical point configuration graphs [82, 83], and the Reeb graphs [96] are examples of the CPGs. The CPGs provide us with a means of handling the extracted features using an abstract data structure. So far, however, there have been no algorithms that construct the CPGs automatically from discrete sample data.

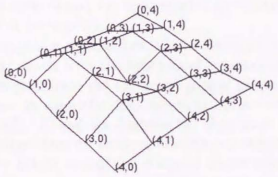
The robustness of the algorithms is also an important issue. If the algorithms are not robust enough, they will fail to finish their work when unexpected data is given. The examples of such unexpected data are degenerate critical points such as flat tops and monkey saddles, which is described in Section 2.1.1 and Appendix A. Previous robust algorithms for other problems have been implemented by transforming the continuous data to discrete one [38], avoiding numerical errors with finite-precision computation [117], and performing geometric computation while preserving the topological consistency [84]. In this study, a method similar to symbolic perturbation [26] is used to implement the robust algorithms. Note that the algorithms to be proposed consist of only the comparisons between the heights of vertices based on finite precision, manipulations of graph data, and boolean operations on the sets of vertices; they do not use divisions to avoid numerical errors[1].

This chapter presents the algorithms for extracting critical points and constructing CPGs from discrete surface data. The algorithms extract the critical point with the topological consistency; i.e., the critical points satisfy the Euler formula. The tools for understanding the surface features are the two CPGs: the surface network [88, 89] and the Reeb graph [96]. In addition, the algorithms are designed so that they are

[1] A method of extracting similar features numerically from the equation of an object surface is presented in [55].



(a)



(b)

(1, 1): $\Delta_+ = 690$,
$\Delta_- = 0$, $N_c = 0$

(2, 2): $\Delta_+ = 0$,
$\Delta_- = 650$, $N_c = 0$

(3, 2): $\Delta_+ = 120$,
$\Delta_- = 180$, $N_c = 4$

(3, 3): $\Delta_+ = 40$,
$\Delta_- = 200$, $N_c = 4$

(c)

Figure 5.1: An example of sample data: The eight-neighbor method fails to extract correct critical point from this example. (a) the height values of the sample data, (b) its three-dimensional image, and (c) the values in the criteria of the eight-neighbor method (explained in Section 5.2.2).

guaranteed to extract correct critical points and CPGs.

The surface network represents the ridge and ravine lines of the surface shape. It consists of the critical points (peaks, pits, and passes), and the ridge and ravine lines. From a pass there are two paths called the ridge lines along which we can go up to peaks and two paths called the ravine lines along which we can go down to pits. These ridge and ravine lines appear in turn when we go around the pass. In this thesis, the surface network also serves as an intermediate representation for constructing the Reeb graph. The Reeb graph represents the splitting and merging of equi-height contours, i.e., the abstract representation of a topographic map. When there is a bifurcation in the Reeb graph, a contour is split. When two contours are merged, edges of the Reeb graph are merged. It is important to extract the changes of the contours from discrete sample data, and several techniques are proposed. In particular, the Reeb graph is a mathematical generalization of the topographic change tree proposed in [62].

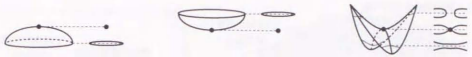
The algorithms perform the following three processes. Firstly, critical points are extracted that satisfy the Euler formula. Secondly, the surface network is constructed by tracing ridge and ravine lines. Thirdly, the surface network is converted to the Reeb graph. The critical points and CPGs are extracted from the given surface samples automatically.

In the following, this chapter first describes the algorithms for surfaces that are topologically equivalent to spheres in Sections 5.2, 5.3, and 5.4. In this case, the Reeb graph to be constructed will become a tree as described in Section 5.4. Algorithms for surfaces of other topological type, i.e., surfaces that are topologically equivalent to connected sums of tori, are implemented by extending the above algorithms and will be presented in Section 5.5.

This chapter is organized as follows: Section 5.2 describes an algorithm for extracting the critical points with the topological consistency. An algorithm for constructing the surface network from the extracted critical points is explained in Section 5.3. Section 5.4 presents an algorithm for converting the surface network to the Reeb graph. While Sections 5.2, 5.3, and 5.4 describe the algorithms for a topological sphere, Section 5.5 presents algorithms for surfaces of other topological type. Section 5.6 describes how to change the height axis of the designed object using the algorithms. Section 5.7 shows results to demonstrate the capabilities of the algorithms, and Section 5.8 summarizes this chapter and refers to future extensions.

5.2 Extracting Correct Critical Points

The surface handled in the following three sections is assumed to be a surface that is topologically equivalent to a sphere. Note that a terrain surface can be regarded as one of such examples when the virtual pit is attached to the surface (cf. Figure 5.3). Image intensities can also be considered as the elevation data of a rectangular area. The sample data to be used in the algorithms is a set of points on the surface.



(a) (b) (c)

Figure 5.2: Topological changes of cross-sectional contours at critical sections: (a) at a peak, (b) at a pit, and (c) at a pass

5.2.1 Critical Points and the Euler Formula

This section explains how to extract the critical points from the discrete sample data with the topological consistency. Before going into details, this subsection gives the mathematical definitions of the critical points [80] and the Euler formula [75, 39]. The detailed definitions of these concepts are provided in Appendix A.

Let $z = f(x, y)$ be a height function which gives the height of each point on a surface. A point p of the function f is called a *critical point* of f if $\mathrm{grad} f(p) = 0$, i.e., $\frac{\partial f}{\partial x} = 0$ and $\frac{\partial f}{\partial y} = 0$. Note that the topology of the cross-sectional contours changes at p while scanning the critical level $f^{-1}(f(p))$ from its upper side to its lower side. Let us consider the topological changes of the cross-sectional contours at the section of each critical point (Figure 5.2). If a new contour appears at p, the critical point p is called a peak (Figure 5.2(a)). If an existing contour disappears at p, the critical point p is called a pit (Figure 5.2(b)). (A peak (pit) is higher (lower) than all other points in its neighborhood.) If a contour is divided or two contours are merged at p, the critical point p is called a pass (Figure 5.2(c)). The critical point p is called non-degenerate if one and only one of the above topological changes occurs in the contour containing p. If χ represents the Euler characteristic of the surface to be handled, the Euler formula can be written as (cf. Section 1.7)

$$\#\{peaks\} - \#\{passes\} + \#\{pits\} = \chi. \tag{5.1}$$

If the surface is topologically equivalent to a patch such as a terrain surface, it can be regarded as a part of a sphere as illustrated in Figure 5.3. The local minimum at the bottom of the sphere is called a *virtual pit* in this chapter. When the surface is handled with the virtual pit, the corresponding Euler formula becomes

$$\#\{peaks\} - \#\{passes\} + \#\{pits\} = 2, \tag{5.2}$$

because $\chi = 2$ for a sphere. The above formula is also called the *mountaineer's equation* in the case of terrain surfaces [39]. Note that the formula is consistent with the relation shown in Section 5.1 when we consider the virtual pit. In the following, the virtual pit is allowed for to estimate the Euler formula if the surface is topologically equivalent to a patch.

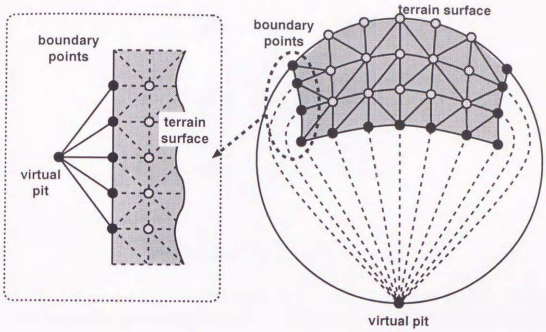


Figure 5.3: A surface patch and a virtual pit on a sphere.

5.2.2 The Eight-neighbor Method

Now let us see the eight-neighbor method [87] stated earlier. This method is proposed for extracting terrain features from discrete samples. In this method, the sample points are aligned on a regular grid of a terrain surface. The eight neighbors of the grid point p are the points in the 3×3 squares surrounding p as shown in Figure 5.4. Each neighbor $p_i (i = 1, 2, \ldots, 8)$ is scanned to see whether or not it satisfies the conditions of the critical points. The notations listed below are used in the following (Figure 5.4).

n the number of the neighbors of p

Δ_i the height difference between p_i $(i = 1, 2, \ldots, n)$ and p

Δ_+ the sum of all positive Δ_i $(i = 1, 2, \ldots, n)$

Δ_- the sum of all negative Δ_i $(i = 1, 2, \ldots, n)$

N_c the number of sign changes in the sequence $\Delta_1, \Delta_2, \ldots, \Delta_n, \Delta_1$

Notice that $n = 8$ in the eight-neighbor method[2]. The critical points are detected according to the following criteria:

peak $|\Delta_+| = 0, \quad |\Delta_-| > \mathrm{T}_{\mathrm{peak}}, \quad N_c = 0$

pit $|\Delta_-| = 0, \quad |\Delta_+| > \mathrm{T}_{\mathrm{pit}}, \quad N_c = 0$

pass $|\Delta_+| + |\Delta_-| > \mathrm{T}_{\mathrm{pass}}, \quad N_c = 4$

[2] The case where $n = 4$ is presented in [50].

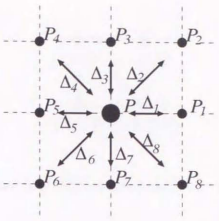


Figure 5.4: Eight neighbors in a grid

Here, T_{peak}, T_{pit}, and T_{pass} are the thresholds in each case. Note that this method is suitable for parallel processing because the result at a point is independent of those at other points.

As shown in Figure 5.1(c), when $T_{peak} = T_{pit} = T_{pass} = 0$, the critical points extracted using the eight-neighbor method do not satisfy the Euler formula. Furthermore, the eight-neighbor method has ambiguities in choosing the thresholds T_{peak}, T_{pit}, and T_{pass}. For example, when $T_{peak} = T_{pit} = T_{pass} = 250$, the Euler formula becomes $\#\{peaks\} - \#\{passes\} + \#\{pits\} = 1 - 1 + 2 = 2$, which keeps the topological consistency. When $T_{peak} = T_{pit} = T_{pass} = 500$, however, the Euler formula becomes $\#\{peaks\} - \#\{passes\} + \#\{pits\} = 1 - 0 + 2 = 3 (\neq 2)$, which violates the topological consistency.

5.2.3 New Criteria for Extracting Critical Points

To ensure that the extracted critical points satisfy the Euler formula, it is necessary to determine the contour changes according to the height. This means that the surfaces must be interpolated from the sample points. For this purpose, this study uses triangulation that is a natural choice for interpolating discrete samples. Note that the contour changes depend on the manner in which we triangulate the sample points. In order to avoid defective thin triangles, it is desirable to use a method similar to the Delaunay triangulation [41]. Suppose that the grid point is labeled with a pair of integer indices as shown in Figure 5.1. We generate a square by connecting four pairs of grid points: (i, j)—$(i + 1, j)$, $(i + 1, j)$—$(i + 1, j + 1)$, $(i + 1, j + 1)$—$(i + 1, j)$, and $(i + 1, j)$—(i, j) as shown in Figure 5.5. Now the grid points and the edges form a pattern like a chessboard. Each square is then divided into two triangles with either of the two diagonals. Here the diagonal is chosen so that the two divided triangles make a flatter surface. In other words, we choose the diagonal that makes the smallest absolute angle between the normals to the triangles divided by the diagonal. Note that it is not necessary to confine ourselves to grid sample points; random sample points can also be handled by

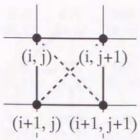


Figure 5.5: The two diagonals of a square

the ordinary Delaunay triangulation.

Data dependent triangulation is a powerful means of triangulating the sample points with smoothness, whatever the type of sample data is. Among several proposed methods, the edge-based method [24] that minimizes the total sum of angles of adjacent triangles at edges, and the vertex-based method [12] that minimizes the total sum of variances of surface normals at vertices are frequently used. Especially in the cases of terrain samples and image intensities, these methods provide smoothly triangulated surfaces.

Let us now see the new definition of the neighbor and the criteria of critical points. Suppose that all critical points are non-degenerate. The neighbors of the point p are the points that are adjacent to p in the triangulated sample points. In this implementation, each point p has a circular list of neighbors in counter-clockwise (CCW) order around p. When all the critical points are non-degenerate, the new criteria of the critical points are as follows:

peak	$\lvert\Delta_+\rvert = 0,$	$\lvert\Delta_-\rvert > 0,$	$N_c = 0$
pit	$\lvert\Delta_-\rvert = 0,$	$\lvert\Delta_+\rvert > 0,$	$N_c = 0$
pass	$\lvert\Delta_+\rvert + \lvert\Delta_-\rvert > 0,$		$N_c = 4.$

In the above criteria, manually specified thresholds are eliminated, and hence the criteria have no ambiguities. Note also that in the case of grid points, the influence of the previous eight neighbors is reduced by choosing either of the two diagonals in each square of the chessboard pattern. Let us return to the case of Figure 5.1. According to the above criteria, only the point $(3, 2)$ is judged as a pass, and the Euler formula is satisfied (Figure 5.6).

In the case of a topological patch, the boundary sample points and the corresponding virtual pit must be handled carefully. In this implementation, the virtual pit is assumed to be a point at the height $-\infty$. After triangulating the sample points, the virtual pit is inserted to the circular lists of the boundary points so that the virtual pit and two adjacent boundary points form a triangle. In this process, the patch and the virtual pit are located as shown in Figure 5.3.

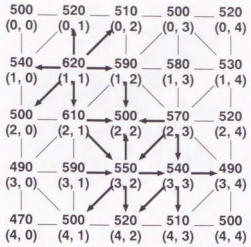


Figure 5.6: A triangulation of a grid

5.2.4 Handling Degenerate Critical Points

Let us now consider the cases containing degenerate critical points. The degenerate critical points are classified into two types: *degenerate passes* and *level regions*. A degenerate pass is a point at which three or more equi-height contours are merged. A degenerate pass and its neighbors including their height values are illustrated in Figure 5.7, where the shaded regions show the areas higher than the pass. In this case, it is necessary to decompose the degenerate pass into non-degenerate ones, because three contours are merged at the pass. While decomposing the degenerate pass, the number of the decomposed passes is counted. The criteria of critical points are now modified as follows to automatically count the decomposed passes.

peak $|\Delta_+| = 0, \quad |\Delta_-| > 0, \qquad N_c = 0$

pit $|\Delta_-| = 0, \quad |\Delta_+| > 0, \qquad N_c = 0$

pass $|\Delta_+| + |\Delta_-| > 0, \qquad N_c = 2 + 2m \ (m = 1, 2, \ldots)$

where m is the number of the decomposed passes

The number of decomposed non-degenerate passes m can be obtained by solving the equation $m = (N_c - 2)/2$.

The second case contains level regions, i.e., regions where two or more adjacent points are at the same height in the triangulated data. One remedy is to group the set of level points together and to regard the group as one point. However, it is not efficient when the region contains other critical points such as a pit or a peak in its interior as illustrated in Figure 5.8(a). The solution used in this study is to introduce another ordering of the sample points in addition to the height in order to discriminate between the points at the same height. This means that the two points are compared using the second ordering if they are at the same height. The algorithm uses the lexicographical ordering

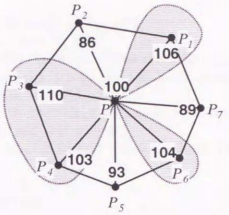


Figure 5.7: A degenerate pass

(a) (b)

Figure 5.8: A level region: (a) a level region containing a pit, and (b) introduction of the second order to the level region.

with respect to the (x, y)-coordinates as the second ordering. Introducing this ordering is equivalent to inclining the height function slightly as illustrated in Figure 5.8(b). Note that this ordering can be implemented using the indices assigned to the sample points if the ordering of the indices coincides with the lexicographical ordering with respect to the (x, y)-coordinates. This strategy is similar to the symbolic perturbation [26] in that it avoids the degenerate cases by introducing the additional comparisons based on the symbols. In this way, the degenerate critical points can be reduced to non-degenerate ones that can be handled within the unified framework.

5.2.5 Algorithm for Extracting Critical Points

The following is the algorithm for extracting critical points.

Algorithm 1 (For extracting critical points)

G_t : a graph representing the triangulation of sample points
L_c : a list of extracted critical points and their neighbors
begin

```
for each vertex p of G_I do
  begin
    generate the CCW neighbor list of p;
    compute Δ+, Δ−, and N_c of p;
    reduce the neighbor list of p;    (See the following explanation.)
    if ((|Δ+| = 0) and (|Δ−| > 0) and (N_c = 0)) then
      mark p as a peak and add p to L_c;
    else if ((|Δ−| = 0) and (|Δ+| > 0) and (N_c = 0)) then
      mark p as a pit and add p to L_c;
    else if ((|Δ+| + |Δ−| > 0) and (N_c = 2 + 2m)) then
      begin
        m := (N_c − 2)/2;
        while (m > 0) do
          begin
            retrieve the last four elements from the neighbor list of p;
            mark p as a pass and add p to L_c with its four representative neighbors:
                                  (See the following explanation.)
            eliminate the last two elements from the neighbor list of p;
            m := m − 1;
          end
      end
  end
end
```

Let us see how the algorithm handles the degenerate pass p shown in Figure 5.9(a). First, the algorithm generates $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ as the CCW neighbor list of p. After calculating Δ_+, Δ_-, and N_c of p, the algorithm reduces the neighbor list of p as follows. While scanning the elements of the neighbor list, the algorithm defines a sequence of neighbors higher than p as an *upper sequence*. In this example, $\{p_1\}$, $\{p_3, p_4\}$, and $\{p_6\}$ are the upper sequences. A *lower sequence* is defined similarly. The algorithm reduces the neighbor list by choosing the highest neighbor from each upper sequence and the lowest neighbor from each lower sequence, and by removing the rest of the neighbors from the list. The neighbor list of this example is reduced to the list $\{p_2, p_3, p_5, p_6, p_7, p_1\}$ by removing p_4, because p_3 is higher than p_4 in the upper sequence $\{p_3, p_4\}$ (Figure 5.9(b)). Here, the reduced list is assumed to begin with a lower neighbor if the list has more than one neighbor. The reason for doing this is to ensure that the four alternating upper and lower neighbors at the pass are selected correctly in what follows.

Since three contours are merged at p, the number of non-degenerate passes m is equal to 2. In the routine that handles a pass, the algorithm first selects the last four elements $\{p_5, p_6, p_7, p_1\}$ as the *representative neighbors* (Figure 5.9(c)). The same procedure is then carried out after the last two elements $\{p_7, p_1\}$ are eliminated from the list (Figure 5.9(d)). As can be seen in Figures 5.9(c) and (d), the four alternating upper and lower neighbors are selected correctly in each of the above steps. The list L_c holds the critical points extracted in the algorithm. The element of L_c also holds the four representative neighbors if its critical point is a pass. This list serves as the intermediate data storage for constructing the surface network.

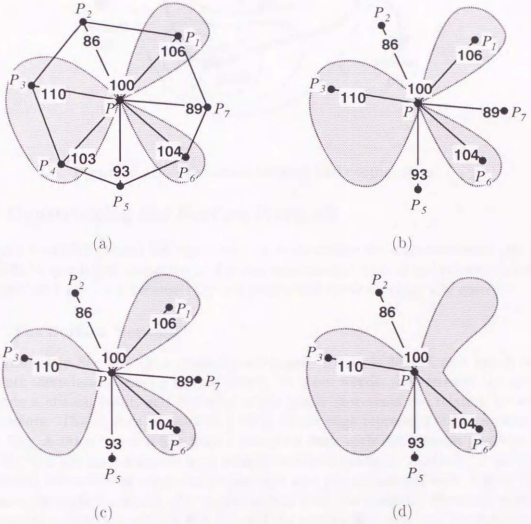


Figure 5.9: The neighbor list of a degenerate pass in extracting critical points: The height values of the points are indicated and the areas higher than the pass are shaded. (a) the original neighbor list, (b) the reduced neighbor list, (c) the list in the first turn of the loop in the algorithm, and (d) the list in the second turn of the loop in the algorithm

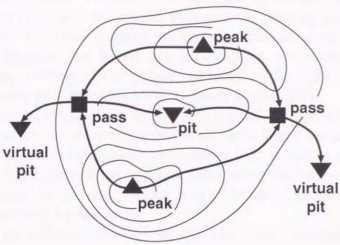


Figure 5.10: The surface network and contour lines

5.3 Constructing the Surface Network

Sections 5.3 and 5.4 present the algorithms for constructing the surface network [88, 89] and the Reeb graph [96], respectively. For this construction, the critical points extracted by Algorithm 1 are used because they are guaranteed to be topologically correct.

5.3.1 The Surface Network

As mentioned in Section 1.2, a *critical point graph (CPG)* is defined as a graph that represents the relations among critical points. In other words, the vertex of the graph represents a critical point, and the edge of the graph represents the relation between its endpoints. The *surface network* is a CPG whose edge represents either a ridge or ravine line. A ridge line is a line from a pass to a peak with the steepest gradient. A ravine line is a line from a pass to a pit with the steepest gradient. Appendix E provides the detailed definitions of ridge and ravine lines and the surface network. Figure 5.10 illustrates the surface network of a terrain surface with the contours. Here, the symbol ▲ represents a peak, the symbol ▼ a pit, and the symbol ■ a pass, and the bold line a contour. The direction of the thick arrow indicates a downslope. This notation is also used in the following explanations. In this way, the surface network represents the ridge and ravine lines of a surface shape efficiently.

5.3.2 Algorithm for Constructing the Surface Network

The algorithm for constructing the surface network is now presented. As described in the explanation of Algorithm 1, each of the extracted passes is stored with its four representative neighbors, two of which are higher than the pass and the other two of which are lower than the pass. For each of the extracted passes, ridge lines are traced from the two upper neighbors up to peaks, and the pass and the peaks are connected by

edges in the surface network. On the other hand, the ravine lines are traced from the two lower neighbors down to pits, and the pass and the pits are connected by edges in the surface network. The ridge (ravine) line of the surface network is traced as follows in the algorithm. Suppose that we are now at a point p. Since the ridge (ravine) line goes in the steepest direction of the surface, we move to the highest (lowest) neighbor of p. The tracing process is repeated until we reach a peak (pit).

This tracing process always ends by reaching a peak (pit) after finite steps. Let us prove this. Suppose we cannot reach a peak (pit) with finite tracing steps. Since the number of sample points is finite, there is a point p that is visited at least twice. This means that p is higher than p, which is a contradiction to the comparisons based on the symbolic perturbation described in Section 5.2.4. Therefore, the algorithm finishes the tracing steps by reaching a peak (pit).

Consequently, the algorithm for constructing the surface network is summarized as follows:

Algorithm 2 (For constructing the surface network)

G_t : a graph representing a triangulation of sample points
L_c : the list of the critical points in Algorithm 1
G_s : a graph representing the surface network to be constructed
(Initially, G_s is empty.)

```
begin
  for each critical point p of L_c do
    add p to G_s;
  for each pass p of L_c do
    begin
      retrieve the two upper neighbors of p from L_c;
      for each of the upper neighbors q do
        begin
          trace the ridge lines from q up to a peak r in G_t;
          connect p and r with an edge in G_s;
        end
    end
  for each pass p of L_c do
    begin
      retrieve the two lower neighbors of p from L_c;
      for each of the lower neighbors q do
        begin
          trace the ravine lines from q down to a pit r in G_t;
          connect p and r with an edge in G_s;
        end
    end
end
```

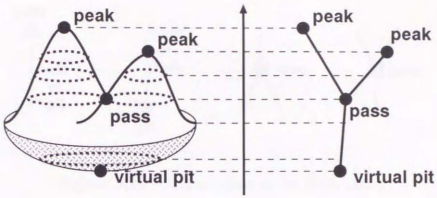


Figure 5.11: (a) A mountain shape with its critical points and (b) its Reeb graph

5.4 Converting the Surface Network to the Reeb Graph

In this thesis, the Reeb graph is not constructed directly; the surface network is first constructed and then it is converted to the Reeb graph. The reason for doing this is the robustness of the algorithm. Of course, the Reeb graph can be constructed directly by detecting the topological changes in the cross-sectional contours from the top to the bottom of the surface. However, this is prone to error because it requires divisions. Another advantage is that the locality of the construction. The surface network can be constructed using only local data access and without any divisions.

5.4.1 The Reeb Graph

The *Reeb graph* [96] represents the splitting and merging of equi-height contours, and is one of the CPGs. Let f be a height function of a surface, and let p and q be points on the surface. The Reeb graph of the height function f is obtained by identifying p and q if the two points are contained in the same connected component on the cross section at the height $f(p)(= f(q))$. This means that a cross-sectional contour corresponds to a point of the edge of the Reeb graph (Figure 5.11). In particular, the vertex of the Reeb graph represents the critical point of the height function f. Figure 5.11(a) shows a mountain shape and its critical points, and Figure 5.11(b) shows the corresponding Reeb graph [108, 109, 47]. The correspondence between the cross-sectional contour of the mountain shape and the point of the edge of the Reeb graph is also indicated by a dotted line in this figure. In this way, the Reeb graph can be used to represent the topological changes of equi-height contours with respect to the height. The formal definition of the Reeb graph is described in Appendix B.

According to the definition of the Reeb graph, the following two properties are derived.

Property 1 *If all the critical points of the height function f are non-degenerate, the*

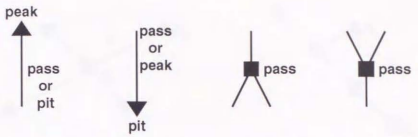


Figure 5.12: Critical points in the Reeb graph

vertices, i.e., the critical points of the Reeb graph of f have the following properties (Figure 5.12).

(1) A peak has only one edge whose opposite endpoint is a pass or a pit lower than the peak.

(2) A pit has only one edge whose opposite endpoint is a pass or a peak higher than the pit.

(3) A pass has either (a) *one (upward) edge whose opposite endpoint is an upper critical point and two (downward) edges whose opposite endpoints are lower critical points, or* (b) *one (downward) edge whose opposite endpoint is a lower critical point and two (upward) edges whose opposite endpoints are upper critical points.*

This is trivially derived by considering the topological changes in the cross-sectional contours at critical sections. □

Property 2 *If the object surface is topologically equivalent to a sphere, the Reeb graph of the object becomes a tree.*

Assume that the Reeb graph contains a cycle. According to the definition of the Reeb graph, the surface contains a torus. This contradicts with the fact that the surface is topologically equivalent to a sphere. □

5.4.2 Relations Between the Surface Network and the Reeb Graph

As preliminaries, this subsection presents the relations between the surface network and the Reeb graph.

Property 3 *An edge with its endpoints in the surface network corresponds uniquely to a path*[3] *between the corresponding critical points in the Reeb graph.*

[3] A path of a graph is an alternating sequence of vertices and edges which begins and ends with vertices, in which each edge is incident with the two vertices immediately proceeding and following it, and in which all the vertices are distinct [42].

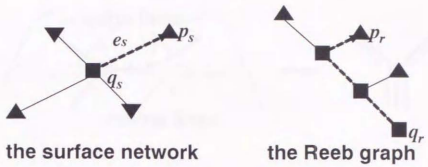


Figure 5.13: A correspondence between an edge of the surface network and a path in the Reeb graph

Let e_s denote an edge that corresponds to a ridge line, and let p_s and q_s denote the peak and pit at the endpoints of the edge e_s. Suppose the vertices of the Reeb graph p_r and q_r correspond to p_s and q_s, respectively (Figure 5.13). Since e_s represents a line that ascends monotonously on the surface from q_s to p_s, there exists a path between q_r and p_r that corresponds to e_s. Since there are no cycles in the Reeb graph according to Property 2, the corresponding path in the Reeb graph can be identified uniquely. Now the property has been derived for the case of ridge lines. The same procedure can be carried out for the case of ravine lines. □

In this thesis, a path in the Reeb graph is called a *monotonously ascending* (or *descending*) *path* if it corresponds to a ridge (or ravine) line.

Property 4 *For each edge incident to a pass in the Reeb graph, there exists either a monotonously ascending path or a monotonously descending path which contains the edge.*

Suppose that at the pass there is a "Y"-shaped branch in the Reeb graph as shown in Figure 5.14. As can be seen in this figure, the ridge lines that go out of the pass correspond to the upward edges of the pass in the Reeb graph. This is because there are two cross-sectional contours at the upper cross section of the pass and each of the two contours is traced by a ridge line. Therefore, there exists an edge in the surface network that corresponds to either of the two upward edges of the pass in the Reeb graph. The downward edge of the pass in the Reeb graph is also covered by the ravine lines that go down from the pass. This derives the property for the case of a "Y"-shaped branch. The same arguments are carried out for the case of a reversed "Y"-shaped branch. □

5.4.3 Algorithm that Converts the Surface Network to the Reeb Graph

This subsection provides an algorithm that converts the surface network to the Reeb graph. The correctness and robustness of this algorithm are presented by deriving several properties in the following subsection.

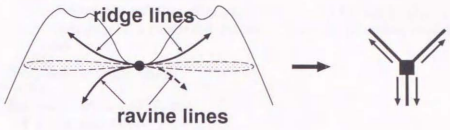


Figure 5.14: Ridge and ravine lines incident to a pass and its corresponding paths in the Reeb graph

The basic idea is to convert the ridge and ravine lines of the surface network to the edges of the Reeb graph. The conversion algorithm first determines the edges incident to peaks and pits, and then changes a pass into a peak or a pit after two of its three incident edges are determined in the Reeb graph. Here, note that a pass of the Reeb graph has three incident edges as described in Property 1. These processes are repeated until the Reeb graph is constructed completely.

Algorithm 3 (For constructing the Reeb graph)

G_s : a graph constructed through Algorithm 2
(Initially, G_s is the surface network.)
G_r : a graph representing the Reeb graph to be constructed
(Initially, G_r is empty.)

begin
for each vertex p of G_s **do**
add p to G_r;
while (TRUE) **do**
begin
if G_s has only one peak and only one pit **then**
begin
join the two vertices with an edge in G_r;
exit;
end
for each vertex p of G_s **do**
begin
if p is a peak in G_s **then**
begin
find the highest vertex p_0 from the lower adjacent vertices of p;
add the edge pp_0 to G_r;
remove the edge pp_0 from G_s;
change the existing edge $pp_i (i = 1, \ldots, n)$ to $p_0 p_i$ in G_s;
change p to a processed vertex; (See the following explanation.)
end
else if p is a pit in G_s **then**
begin
find the lowest vertex p_0 from the upper adjacent vertices of p;
add the edge pp_0 to G_r;

```
            remove the edge pp_0 from G_s;
            change the existing edge pp_i(i = 1, ..., n) to p_0p_i in G_s;
            change p to a processed vertex:   (See the following explanation.)
          end
        end
      if G_s has no edges then
        exit;
      for each vertex q of G_s do
        if q is a pass then
          begin
            if the vertex that corresponds to q in G_r has two edges in G_r then
              begin
                if q has only lower adjacent vertices then
                  change q to a peak;
                else if q has only upper adjacent vertices then
                  change q to a pit;
              end
          end
    end
  end
```

Figure 5.15 shows how the algorithm works with an example. Here, G_s and G_r represent the graphs of Algorithm 3. As mentioned earlier, the symbols ▲, ▼, and ■ represent a peak, a pit, and a pass respectively. The symbol ○ is also used to represent a vertex already processed in Algorithm 3. Figure 5.15(a) shows the initial states of G_s and G_r. The vertex 0 is the first peak to be handled in the routine that determines the edges of the peaks in the Reeb graph. The algorithm finds the vertex 2 as the highest vertex adjacent to the vertex 0, adds the edge 02 to G_r, removes the corresponding edge from G_s, and changes the edge 03 to the edge 23 in G_s (Figure 5.15(b)). Similar conversion process is applied to the vertex 1 representing a peak (Figure 5.15(c)). Symmetrical conversion processes are carried out for the pits, namely the vertices 4 and 5(virtual pit). After all the peaks and pits are processed, the algorithm finds the passes that have two edges already added to G_r. Let us consider the edges incident to the pass in G_s. If all the edges are downward edges, the algorithm changes the pass to a peak. If all the edges are upward edges, the algorithm changes the pass to a pit. In this example, the vertex 2 is changed to a peak and the vertex 3 is changed to a pit in G_s (Figure 5.15(d)). These conversion processes are repeated until all the edges of the Reeb graph are determined. Figure 5.15(e) is the final state of the conversion.

5.4.4 Correctness and Robustness of the Algorithm

The correctness of the conversion algorithm is obtained from the following property.

Property 5 *Algorithm 3 correctly converts the edges of the surface network to those of the Reeb graph.*

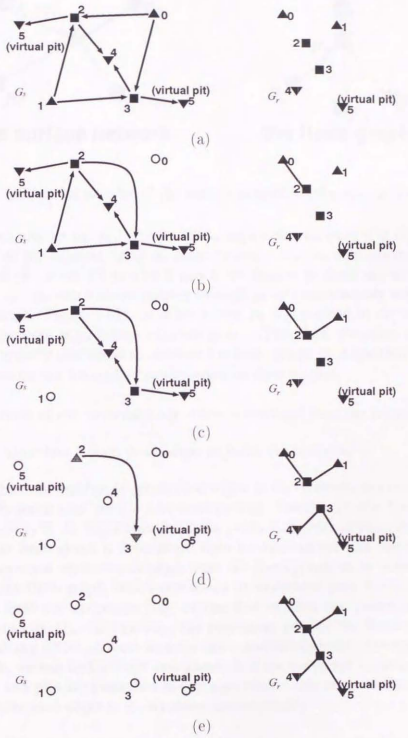


Figure 5.15: The steps of the conversion algorithm: The left-hand figure shows G_s (the surface network) and the right-hand figure shows G_r (the Reeb graph).

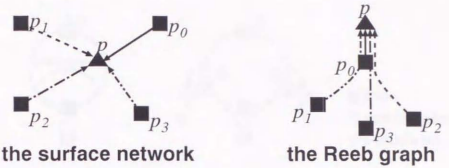


Figure 5.16: Converting an edge of the surface network to the edge of the Reeb graph

Let p be a peak and let p_0, p_1, ..., p_n be the adjacent vertices of p in G_s, where p_0 is the highest of all the adjacent vertices. From Property 1, p has only one downward edge in the Reeb graph. From Properties 3 and 4, we cannot go from any of the adjacent points p_1, p_2, ..., p_n to p without passing through p_0 by monotonously ascending paths in the Reeb graph (Figure 5.16). In other words, p_0 is contained in the monotonously ascending paths from any of these adjacent points. Therefore, the edges of the surface network are correctly converted to those of the Reeb graph in Algorithm 3. The symmetrical procedure can be applied to the edges incident to pits. □

The robustness of the conversion algorithm is obtained from the following property.

Property 6 *Algorithm 3 takes finite steps to finish the conversion.*

Let us show that the number of determined edges in G_r increases monotonously. Consider the Reeb graph that we are now constructing. Recall that the Reeb graph is a tree from Property 2. In Algorithm 3, all the peaks and pits are processed first. This means that the Reeb graph is determined from its extreme vertices and edges. Let us cut off these extreme vertices and edges from the Reeb graph to be constructed. The remainder of the Reeb graph, which represents its undecided part, is also a tree. Since a tree has at least two endpoints [42], we can find at least two passes that have two determined edges in G_r. In this way, the remaining part of the Reeb graph can be reduced by cutting off its extreme vertices and edges through the conversion processes. In other words, we can find at least two passes that are converted to peaks or pits after all the peaks and pits are processed in the algorithm. This leads to the fact that the number of determined edges in G_r increases monotonously. □

Note that Algorithms 1, 2, and 3 use only comparisons between the heights of vertices and graph manipulations. Therefore, the presented algorithms extract correct critical points and CPGs without fail.

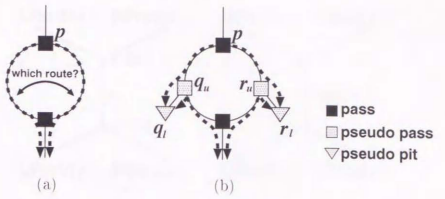


Figure 5.17: Cycles in the Reeb graph: (a) ravine lines in the Reeb graph and (b) tags inserted to the cycle

5.5 Extracting Features from a Surface of Arbitrary Topological Type

The previous sections have presented the algorithms for extracting critical points and constructing CPGs from sample data of a topological sphere. The next step is to extend the proposed algorithms for a surface of arbitrary topological type, i.e., a surface that is topologically equivalent to a connected sum of tori.

5.5.1 Idea of the Extended Algorithm

As described earlier, the proposed algorithms fail to extract the shape features if the object surface is topologically equivalent to a connected sum of tori. This limitation arises from the fact that the algorithms are based on the assumption that the Reeb graph is a tree. If the Reeb graph is not a tree, Property 3 does not hold due to the lack of Property 2. Let us see this with an example. Suppose there is a cycle in the Reeb graph of an object as shown in Figure 5.17(a). The two ravine lines incident from the pass p correspond to the paths indicated by the thick dotted arrows in the Reeb graph. The problem is that the route of each path cannot be identified because of the cycle. In order to overcome this limitation, it is necessary to add some information about the routes of the ravine lines in the cycle. This is also sufficient because the proposed algorithms can determine the Reeb graph except for its cycles.

In the implementation of this study, tags are inserted to the cycle as shown in Figure 5.17(b). Here, the inserted tag is an edge bounded by artificial critical points: a pass and a pit. In Figure 5.17(b), the edges $q_u q_l$ and $r_u r_l$ are the tags. In the following, the artificial critical points are called *pseudo critical points*. Furthermore, the artificial passes such as q_u and r_u are called *pseudo passes* and the artificial pits such as q_l and r_l are called *pseudo pits*. Note that as can be seen in Figure 5.17(b) the left ravine line changes its route at the pseudo pass q_u so that it goes down to the pseudo pit q_l. In addition, new two ravine lines emanate from the pseudo pass q_u; one ravine line goes

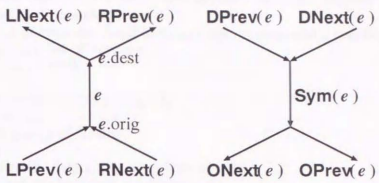


Figure 5.18: The edge functions

to the pseudo pit q_l, and the other takes over the ravine line that comes to q_u from the upper side. The same can be applied to a pair of pseudo critical points: r_u and r_l. This is the main idea of the extended version of the algorithms. Note that the inserted artificial critical points will be eliminated after the Reeb graph is constructed.

In what follows, the samples of the object surface are already triangulated[4]. In particular, it is assumed that the *edge functions* [41] of the triangulated surface are available in the following processes. The edge functions provide us with algebraic operations that retrieve the topological structures of edges in the triangulation. Figure 5.18 illustrates such operations when we look at the closed surface from its outside. Assume that e.orig represents the origin of the oriented edge e, and e.dest represents the destination of the edge e. As shown in Figure 5.18, $\mathsf{Sym}(e)$ represents the same edge as e but with its orientation reversed. $\mathsf{ONext}(e)$ represents the edge next to e in the CCW circular list of edges incident to e.orig. $\mathsf{OPrev}(e)$ represents the edge next to e in the CW circular list of edges incident to e.orig, and so forth.

The reminder of this section describes modifications to Algorithm 2 and Algorithm 3, which are needed for surfaces of arbitrary topological type.

5.5.2 Modifications to Algorithm 2

To extract the Reeb graph from the mesh samples of the surface of arbitrary topological type, it is necessary to add supplemental information to the surface network when it is extracted. For this purpose, two additional processes are inserted to Algorithm 2. One is to insert the pseudo passes and pseudo pits and to extract their corresponding cross-sectional belts, and the other is to modify the routes of ravine lines that pass through the cross-sectional belts. The extended version of Algorithm 2 is listed below.

Algorithm 4 (For constructing the surface network (Extended version))

G_t : a graph representing a triangulation of sample points
L_c : the list of the critical points in Algorithm 1

[4] There are several techniques for constructing visually appealing triangulations of the object surface [43, 106, 52].

```
G_s : a graph representing the surface network to be constructed
      (Initially, G_s is empty.)
L_b : the list of cross-sectional belts and the corresponding pseudo passes and pits
L_v : temporary list of vertices
u : temporary vertex
begin
  for each critical point p of L_c do
    add p to G_s;
  for each pass p of L_c do
    begin
      retrieve the two upper neighbors of p from L_c;
      for each of the upper neighbors q do
        begin
          trace the ridge lines from q up to a peak r in G_t;
          obtain a cross-sectional belt from the edge pq;
          insert a pseudo pass s and a pseudo pit t to G_s;
          insert the cross-sectional belt together with s and t to L_b;
          connect s and r with an edge in G_s;
          connect s and t with an edge in G_s;
        end
    end
  for each pass p of L_c do
    begin
      retrieve the two lower neighbors of p from L_c;
      for each of the lower neighbors q do
        begin
          trace the ravine lines from q down to a pit r in G_t;
          set L_v to be empty;
          for each of the belts of L_b do
            begin
              find the union of the ridge line and the belt;
              if the union is not empty then
                insert the pseudo pass and pit associated with the belts into L_v ;
            end
          sort L_v by the heights of the pseudo passes in an descending order;
          u = p;
          for each pair of a pass and a pit (s, t) of L_v do
            begin
              connect u and t with an edge in G_s;
              u = s;
            end
          connect u and q with an edge in G_s;
        end
    end
end
```

In Algorithm 4, the part marked with ↑'s corresponds to the former process, and the part marked with ↓'s corresponds to the latter process.

In the part marked with ↑'s, the algorithm inserts a pseudo pass s and a pseudo pit t after tracing the ridge line from a pass p. The pseudo pass and pseudo pit are set so that they satisfy $s > t > p$ in the height order using the technique of the symbolic perturbation (cf. Section 5.2). The *cross-sectional belt* is introduced to represent the cross-sectional contour at the height of p. As shown in Figure 5.19(a), the cross-sectional belt is bounded by the upper and lower cyclic lists of vertices. The reason why a belt is used instead of a cross-sectional contour itself is that the algorithm can obtain the belt by consulting only the height order of vertices (via the technique of the symbolic perturbation) and the topological structures of edges (via the edge functions) although it is impossible to obtain a real cross-sectional contour without divisions.

The algorithm uses the cross-sectional belt to represent the cross-sectional contour of the cylindrical part that corresponds to the upward edge of the pass p in the Reeb graph. As shown in Algorithm 4, the algorithm first inserts a pseudo pass s and a pseudo pit t, and then finds its corresponding cross-sectional belt from the edge pq, where q is one of the two upper representative neighbors of the pass p (cf. Algorithm 1). Algorithm 5 detects the cross-sectional belt from the edge pq.

Algorithm 5 (For extracting a cross-sectional belt)

```
pq : the input edge of the pass p and its upper neighbor q
L_u : the upper cyclic list of vertices
L_l : the lower cyclic list of vertices
flag : either "lower" or "upper"
e, e_u, e_l : edges
begin
  insert q to L_u;
  insert p to L_l;
  e = pq
  while ( TRUE ) do
    begin
      e_u = OPrev( e );
      e_l = DNext( e );
      if e_u.dest is higher than p then
        begin
          e = e_u;
          flag = "upper";
        end
      else if e_l.orig is equal to or lower than p then
        begin
          e = e_l;
          flag = "lower";
        end
      if e is equivalent to pq then
        exit;
      if ( flag = "upper" ) then
        insert e_u.dest to L_u;
      else if ( flag = "lower" ) then
        insert e_l.orig to L_l;
```

(a) (b)

Figure 5.19: The cross-sectional belt extracted from the edge pq: (a) its side view and (b) its top view

```
    end
end
```

Algorithm 5 extracts the upper and lower cyclic lists of the cross-sectional belt by moving the edge e zigzag on the surface. The process is illustrated in Figure 5.19(b), where the cross section at the height of p is indicated by the dotted line. Note that during this process the edge e is always intersected with the cross section. Therefore, the edge e goes along the cross-sectional contour and extracts the upper and lower cyclic lists of vertices.

The following property follows from the fact that the edge e has an intersecting point with the cross section at the height of the pass p.

Property 7 *The lower cyclic list of the cross sectional belt does not contain other critical points that are higher than the pass p. The upper cyclic list of the cross sectional belt does not contain other critical points that are lower than the pass p.*

This means that there are no illegal cases of cross-sectional belts as shown in Figure 5.20. Property 7 guarantees the cross-sectional belt corresponds to the edge of the Reeb graph uniquely unless the surface samples are sparsely taken; such illegal cases do not occur in general[5]. In the experiments of this study, no such cases are encountered.

The part marked with ♭'s modifies the routes of ravine lines that pass through the extracted cross-sectional belts. In other words, if a ravine line passes through a cross-sectional belt, it is divided into two at the pseudo pass associated with the belt. Here, one of the two divided ravine lines goes to the corresponding pseudo pit and the other takes over the original ravine line. Suppose that as shown in Figure 5.21 there is a ravine line that goes from the pass p to the pit r and there are also cross-sectional belts whose corresponding pseudo passes and pseudo pits are $(s_k, t_k)(k = 1, 2, \ldots)$. If the ravine line passes through the belt of (s_1, t_1) first, the edge of the surface network pr is changed to pt_1, and an edge $s_1 r$ is newly inserted to take over the original ravine line. Note that

[5] Such illegal cases occur, for example, when there exists a ravine line that do not corresponds to the edge while intersected with both the corresponding upper and lower cyclic lists, for example, in the cycle of a torus. Refer also to the explanation of the part marked with ♭'s.

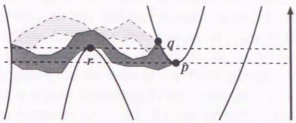


Figure 5.20: An illegal example of a cross-sectional belt

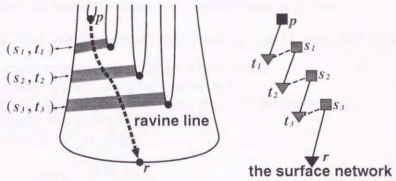


Figure 5.21: Modifying the route of a ravine line

the edge s_1t_1 is also inserted in this process. If the ravine line passes through the belt of (s_2, t_2) next, the edge s_1r is change to s_1t_2, and new edges s_2r and s_2t_2 are inserted. The same procedure is carried out if the ravine line goes through other cross-sectional belts.

This process is implemented in the part marked with ›'s as follows. After tracing the ravine line that goes from a pass p to a pit r, the algorithm finds the intersections of the ravine line with the cross-sectional belts that are extracted previously. These intersections can be obtained by calculating the union of the ravine vertices and the cross-sectional vertices. If the ravine line has intersections with both of the upper and lower cyclic lists of a belt, the algorithm inserts the corresponding pseudo pass and pseudo pit into the list L_v. After examining intersections with all the cross-sectional belts, the algorithm sorts the elements of L_v in a descending order by the heights of the pseudo passes. For each of the pair of L_v, the algorithm creates a branch to change the route of the ravine line as shown in Figure 5.21.

5.5.3 Modifications to Algorithm 3

Since Algorithm 3 can determine the tree parts of the Reeb graph whatever the object surface is, it is necessary to extend the algorithm so that it can also determine the cycle parts of the Reeb graph. If Algorithm 3 is used to extract the Reeb graph for

surfaces of arbitrary topological type, the parts that contain cycles remain undecided. In this situation, it is easy to see that the lowest vertex of the undecided parts is a pass. According to Section 5.5.2, the first and second lowest adjacent vertices of the pass are its corresponding pseudo passes. Therefore, the edges between the pass and the corresponding pseudo passes can be fixed in the Reeb graph, and one of the undecided cycles is reduced to a tree where Algorithm 3 can be applied. The process can be applied until the remaining parts contain no cycles. This is the main idea of the extended version of the algorithm. In this way, the pseudo critical points effectively determine the cycle parts of the Reeb graph.

Consequently, only simple modifications are needed in Algorithm 3 for converting the surface network to the Reeb graph in the case of surfaces of arbitrary topological type. The extended version of the algorithm is listed as follows.

Algorithm 6 (For constructing the Reeb graph (Extended version))

```
    G_s : a graph constructed through Algorithm 2
          (Initially, G_s is the surface network.)
    G_r : a graph representing the Reeb graph to be constructed
          (Initially, G_r is empty.)
    flag : either "true" or "false"
    begin
      for each vertex p of G_s do
        add p to G_r;
      while (TRUE) do
        begin
          if G_s has only one peak and only one pit then
            begin
              join the two vertices with an edge in G_r;
              exit;
            end
          for each vertex p of G_s do
            begin
*             if p is a (pseudo) peak in G_s then
                begin
                  find the highest vertex p_0 from the lower adjacent vertices of p;
                  add the edge pp_0 to G_r;
                  remove the edge pp_0 from G_s;
                  change the existing edge pp_i(i = 1, ..., n) to p_0p_i in G_s;
                  change p to a processed vertex;
                end
*             else if p is a (pseudo) pit in G_s then
                begin
                  find the lowest vertex p_0 from the upper adjacent vertices of p;
                  add the edge pp_0 to G_r;
                  remove the edge pp_0 from G_s;
                  change the existing edge pp_i(i = 1, ..., n) to p_0p_i in G_s;
                  change p to a processed vertex;
                end
            end
          if G_s has no edges then
            exit;
```

```
†       flag = "false";
        for each vertex q of G_s do
*         if q is a (pseudo) pass then
            begin
              if the vertex that corresponds to q in G_r has two edges in G_r then
                begin
                  if q has only lower adjacent vertices then
                    begin
                      change q to a peak;
†                     flag = "true";
                    end
                  else if q has only upper adjacent vertices then
                    begin
                      change q to a pit;
†                     flag = "true";
                    end
                end
            end
†       if ( flag = "false" ) then
†         begin
†           find the lowest pass p of all the vertices in G_s;
†           find the lowest vertex p_A from the upper adjacent vertices of p;
†           add the edge pp_A to G_r;
†           remove the edge pp_A from G_s;
†           find the lowest vertex p_B from the upper adjacent vertices of p;
†           add the edge pp_B to G_r;
†           remove the edge pp_B from G_s;
†           remove the existing edge pp_i (i = 1, ..., n) from G_s;
†         end
      end
    end
```

In Algorithm 6, the parts marked with $*$'s show that it is not necessary to distinguish between pseudo critical points and ordinary critical points. The parts marked with †'s perform the newly inserted process described above.

The correctness of Algorithm 6 can be obtained from the following property.

Property 8 *The pseudo critical points inserted in Algorithm 4 do not prevent Algorithm 6 from determining the tree parts of the Reeb graph.*

It is sufficient to confirm the two types of passes: a "Y"-shaped pass and a reversed "Y"-shaped pass (Figure 5.22). Recall that the pseudo passes are set to the vertices next higher than the corresponding ordinary pass using the technique of the symbolic perturbation.

Figure 5.22(a) illustrates the case of a "Y"-shaped pass: the pass has two upward edges and one downward edge. While the left side shows the Reeb graph to be determined using only ordinary critical points, the right side shows the Reeb graph also using pseudo critical points. The ridge and ravine lines are indicated by the arrows in the figure. In this case, only a pair of pseudo critical points is inserted into each of the two upward edges. It follows from Figure 5.22(a) that this part can be determined using

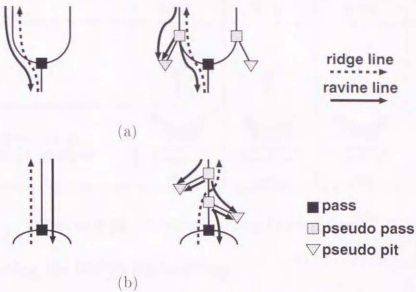


Figure 5.22: The effects of inserting pseudo critical points in the Reeb graph: (a) a "Y"-shaped pass and (b) a reversed "Y"-shaped pass

Algorithm 3 if the part is not contained in a cycle of the Reeb graph. Of course, if the part is contained in a cycle of the Reeb graph, it will work effectively in Algorithm 6 as described above.

Figure 5.22(b) illustrates the case of a reversed "Y"-shaped pass; the pass has two downward edges and one upward edge. While the left side shows the Reeb graph to be determined using only ordinary critical points, the right side shows the Reeb graph also using pseudo critical points. In this case, two pairs of pseudo critical points are inserted to the upward edge. As shown in the right side of Figure 5.22(b), the pseudo passes take the first and second lowest vertices in the upper adjacent vertices of the corresponding pass. This is made possible by using the technique of the symbolic perturbation. Thus, this part can be also determined using Algorithm 3. □

The pseudo critical points of the Reeb graph that is constructed by Algorithm 6 can be eliminated by cutting off the edges bounded by the pseudo critical points.

5.6 Changing the Height Axis

The extended algorithms allow us to extract the Reeb graph from the discrete triangulated samples of a surface of arbitrary topological type. This section describes how to change the height axis of the object. For this purpose, the model of the control network is constructed, and the model is then fit to the object surface.

Operator	E1PC	E1SI	E1IN	E1OUT
Branch type				
Orientation of neighbors projected onto (x, y)-plane	 CW	 CCW	 CW	 CCW

Figure 5.23: Classification criteria for passes

5.6.1 Extracting the Object Embeddings

The extended algorithms enable us to obtain the topological skeletons of the object surface. To construct the model of the control network, the embeddings of the object in 3D space should be extracted in addition to the topological skeletons. This amounts to extracting the sequence of the Morse operators together with their arguments (cf. Section 2.1.3).

Among the Morse operators, E2 (for a peak) and E0 (for a pit) can easily be identified. For other types of the Morse operators (i.e., for passes), one can determine the types by considering the surface orientations around the corresponding critical points[6]. Suppose that the surface samples are dense enough to represent the neighborhoods around the critical points as a single-valued function $z = f(x, y)$; there are no overhangs in the neighborhoods. In this case, the surface orientation at a critical point can be retrieved by examining the orientation of the circular list of its corresponding neighbors projected onto the (x, y)-plane. If the circular list of neighbors is arranged in CCW order when looking at them from the outside of the surface, the types of the Morse operators for passes are identified using the criteria as shown in Figure 5.23.

In addition to the type of the Morse operators, their corresponding arguments should be retrieved. This is achieved by examining the inclusion relations among the cross-sectional contours at regular sections.

In this way, the sequence of the Morse operators are extracted from the triangulated samples. This sequence makes it possible to construct the model of the control network.

5.6.2 Fitting the Control Network

The process of fitting the constructed model of the control network to the object surface consists of the following steps.

[6] A method of determining the types of branches in the case of terrain surfaces is presented in [46]

(1) The first step is to obtain the representative contours at regular sections from the sample data[7]. Here, each of the cross-sectional contours corresponds to the edge of the Reeb graph of the object surface.

(2) The second step is to extract characteristic points, i.e., points of maximum and minimum curvature on the extracted cross-sectional contour. Note that the points of maximum and minimum curvature have close relationships with the planar skeletons of the contour obtained by the *2D medial axis transform (MAT)* [10, 57].

(3) The third step is to assign the curve segments of the control network to the object surface so that the curve segments are fit to the characteristic points of the cross-sectional contours. This means that the model of the control network will try to capture the cross-sectional planar skeletons of the object surface.

(4) The fourth step is to fit the vertical curve segments of the control network to the object by tracing the object surface. This fitting is accomplished using the variational optimization techniques described in Chapter 4.

(5) The final step is to fit the horizontal, i.e., cross-sectional curve segments of the control network to the object. This fitting process is also performed using the variational optimization techniques described in Chapter 4.

In this way, the height axis of the object shape is changed by fitting the model to the object surface. The results are shown in Section 5.7.

5.7 Results

This section provides results obtained using the proposed algorithms.

Terrain samples

Since it is possible that small undulations hide large undulations in the case of steep mountain regions, the wavelet transforms (cf. Chapter 4) are used to eliminate such detailed features. While grid sample data is used in these examples, random sample data can be handled using the proposed algorithm as explained in Section 5.2.

Features are extracted from the terrain samples around Mt. Fuji, which is the highest mountain in Japan (Figure 5.24). Figure 5.24(a) shows a rendered image of the terrain surface around Mt. Fuji. The top view of the critical points with contour lines is shown in Figure 5.24(b). The critical points are extracted using Algorithm 1. The red point represents a peak, the green point a pass, and the light blue point a pit. Figure 5.24(c) shows the ridge and ravine lines obtained using Algorithm 2. The yellow line indicates a ridge line and the purple line indicates a ravine line. Figure 5.24(d) is the surface network constructed using Algorithm 2. The side view of the Reeb graph obtained using Algorithm 3 is shown in Figure 5.24(e) and its enlarged image is shown in Figure

[7] Of course, this requires divisions.

5.24(f). The edges incident to the virtual pit are omitted in Figures 5.24(d), (e), and (f). These results show that the proposed algorithms efficiently extract shape features such as ridge and ravine lines from the terrain samples. Both the surface network and the Reeb graph are constructed correctly.

Another example to be presented is the region around Lake Ashinoko, which is a famous crater lake in Japan (Figure 5.25). Figure 5.25(a) shows a rendered image of the terrain surface around Lake Ashinoko. The top view of the critical points with contour lines is shown in Figure 5.25(b). The critical points are extracted using Algorithm 1. Figure 5.25(c) shows the ridge and ravine lines obtained using Algorithm 2. Figure 5.25(d) is the surface network constructed using Algorithm 2. The side view of the Reeb graph extracted from the surface network using Algorithm 3 and its enlarged image are shown in Figures 5.25(e) and (f). The outer rim of the crater is efficiently detected as ridge lines, and Route No. 1, which is famous for the course of the Hakone Ekiden race, is detected as a ravine line. The geographical structures of the terrain surface are extracted by constructing the surface network and the Reeb graph.

These obtained results demonstrate the capabilities of the proposed algorithms.

Samples of the surface of arbitrary topological type

The extended algorithms can extract critical points and CPGs from the sample data of a surface even when the surface is topologically equivalent to a connected sum of tori. Figure 5.26 shows the results extracted from a monster-like object. The rendered surface of the object is shown in Figure 5.26(a). The extracted critical points together with the contour lines are shown in Figure 5.26(b). Note that the contour lines were not used in this extraction; they are shown only for clarity of the results. Figure 5.26(c) shows the surface network of the object, which is constructed using Algorithm 4. Figure 5.26(d) shows the Reeb graph of the object, which is extracted from the surface network using Algorithm 6. The torus part of the object is successfully extracted. The extended algorithms will serve as the fundamental tools for changing the height axis of the object.

Changing the height axis

The model of the control network is fit to the object surface in order to change the height axis of the designed object. Figure 5.27 shows the difference between the object designed in the system and the object reconstructed from the polygonal surface by fitting the model of a control network. Figures 5.27(a) and (b) show the control network and the object surface designed in the system, respectively. From the designed surface, the system obtains the polygonal representations of the object shape. Figures 5.27(c) and (d) show the control network and the object surface reconstructed by fitting the model of the control network based on the extracted Reeb graph. While there are differences between the two objects, the reconstructed shape effectively simulates the original object shape.

Figure 5.28 shows a cup designed using this operation. First, the handle of a cup is designed (Figure 5.28(a)). The height axis of the handle is then changed (Figure

5.28(b)) in order to combine the handle with a cup (Figure 5.28(c)) that is designed separately. The resultant cup with the handle is shown in Figure 5.28(d).

5.8 Summary

This chapter has presented the algorithms for extracting critical points and CPGs from the surface samples with the correctness and robustness. Critical points are extracted so that they satisfy the Euler formula. The surface network is constructed by tracing the ridge and ravine lines from the extracted critical points. This chapter also presented the algorithm for converting the surface network to the Reeb graph. The correctness and robustness of the proposed algorithms were described by deriving several properties. The issue of changing the height axis of the object was addressed. Examples were presented to demonstrate the capabilities of the algorithms.

The proposed algorithms play a fundamental role for further applications. It is important to integrate the smoothing operations such as wavelet transforms [69] with the algorithms because the smoothing operations provide us a powerful means of extracting large undulations from the noisy data. The issues of taking sample points and generating a triangle mesh from a smooth surface representation also remain to be addressed. The algorithms are also useful for designing smooth surfaces using geometric constraints because they provide the means of extracting characteristic points for the constraints (cf. Section 4.6). Coding the time-varying surfaces using the proposed algorithms is also an important research theme. Furthermore, there are other definitions of ridge and ravine lines [3, 2] that have a close relationship with the definition in this study, and it is interesting to use these definitions for the problems discussed in this chapter.

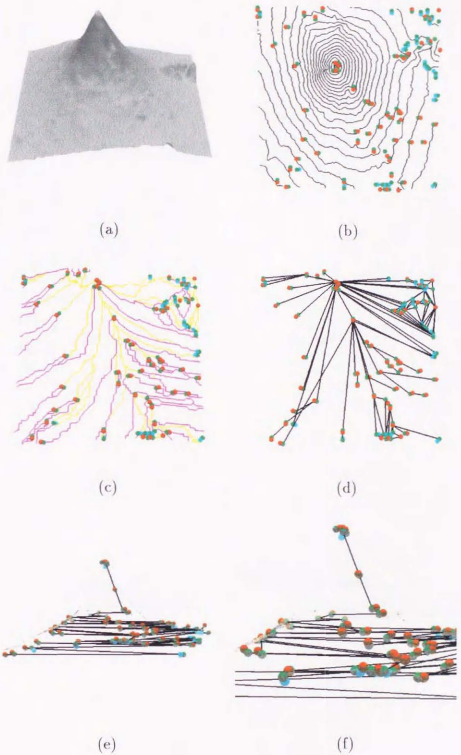


Figure 5.24: Mt. Fuji: (a) a surface, (b) critical points and contour lines, (c) ridge and ravine lines, (d) the surface network, (e) the Reeb graph, and (f) its enlarged image

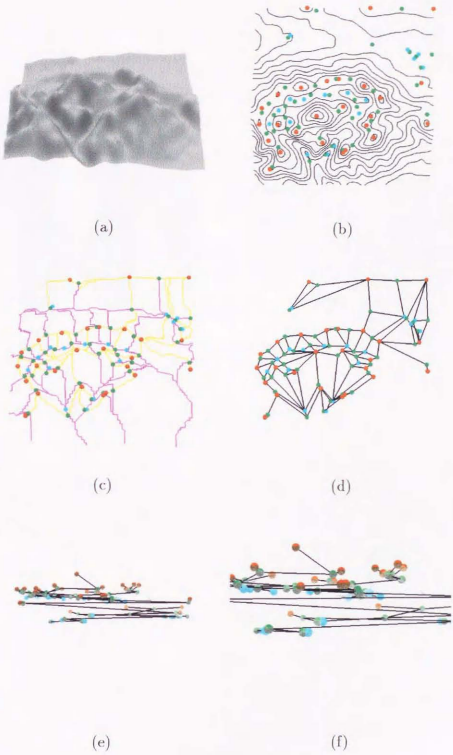


Figure 5.25: Lake Ashinoko: (a) a surface, (b) critical points and contour lines, (c) ridge and ravine lines, (d) the surface network, (e) the Reeb graph, and (f) its enlarged image

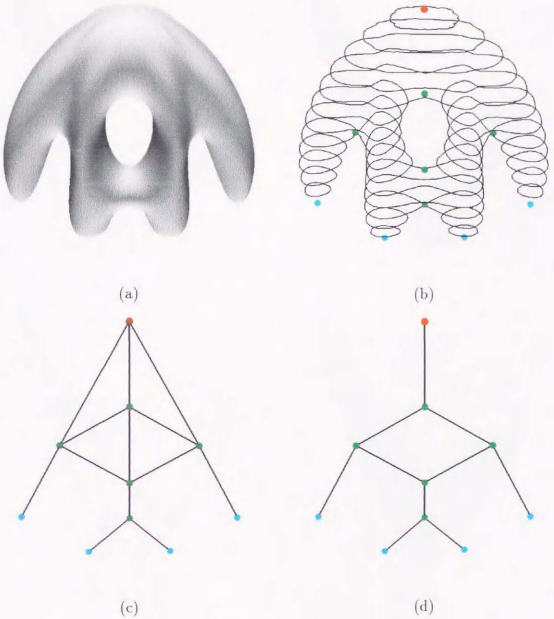


Figure 5.26: A monster-like shape: (a) a surface, (b) critical points and contour lines, (c) the surface network, and (d) the Reeb graph

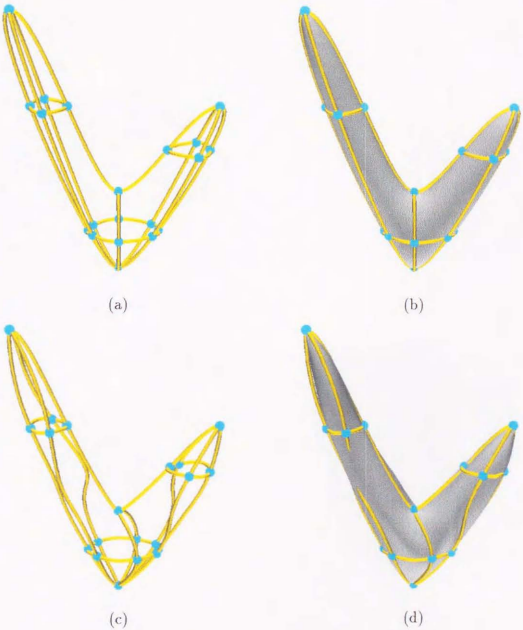


Figure 5.27: Fitting the model of a control network to a surface: (a) the control network and (b) the surface of the object designed in the system; (c) the control network and (d) the surface of the object reconstructed by fitting the model of a control network

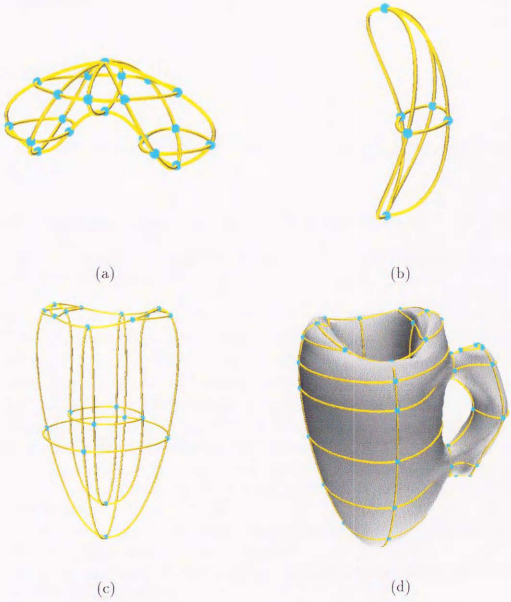


Figure 5.28: Designing a cup: (a) designing a handle, (b) changing the height axis of the handle, (c) designing a container, (d) combining a container and a handle to form a cup

Chapter 6

Discussions

This chapter discusses the differences between the proposed method and the conventional methods.

6.1 Differences from the Major Representations

This section compares the proposed representation schemes with the conventional major schemes with respect to the following properties:

(1) Expressive power: What objects can be covered by the representation scheme? How is the precision of the representation scheme?
(2) Validity: Do all admissible representations correspond to valid objects?
(3) Unambiguity: Do all valid representations designate one object?
(4) Uniqueness: Do some objects have more than one valid representation?
(5) Description language: What kinds of description languages can be supported in the system based on the representation scheme?
(6) Conciseness: How large do representations of practically interesting smooth objects become? (This property is often in contradiction with the precision of the representation.)
(7) Closure of operations: Do object description and manipulation operations preserve the validity of object representations?
(8) Computational ease: What kinds of computational complexity are involved in the representation scheme?

These properties of representation schemes are derived from [71]. The major representation schemes to be compared with the proposed one are the octree, cell decomposition, CSG, and B-reps. Note that the first three representations are contained in the category of volumetric enumeration models while the proposed model and B-reps try to represent an object through the representation of its bounding surface. Therefore, the proposed model and B-reps share advantages and disadvantages for some of the above properties. Tables 6.1–6.8 show the evaluations of the representation schemes based on

Table 6.1: The evaluations of the expressive power

Representation	Evaluation
Octree	Approximate representation Controls the depths of octrees for expressive precision
Cell decomposition	Represents general shapes using the cells with curved surfaces
CSG	Difficult to represent complex smooth surfaces
B-reps	Holds a general domain (in particular for polyhedral objects)
The proposed model	Holds a general domain (in particular for smooth objects)

the above described properties. Let us consider each of the properties in more detail in what follows.

Expressive power

The expressive power of the representation scheme means the domain of objects covered by the scheme. This property also represents the precision of the object representations. The properties of the major conventional representation schemes and the proposed one are described as follows.

- Octree: The octrees are approximate representations. While the arbitrary precision can be achieved by controlling the depths of the octrees, it requires the cost of high storage use.
- Cell decomposition: The expressive domain of the cell decomposition is larger than that of the octree, because the cell decomposition can represent more general shape using the cells with curved surfaces.
- CSG: The expressive power of the CSG depends on the number of available primitives. The CSG cannot cover complex smooth surfaces because the expressive domain is limited to combinations of simple primitives in general.
- B-reps: The scheme of the B-reps has a more general domain than that of the CSG. While the scheme of the B-reps is suited for polyhedral objects, its expressive power is large enough to represent the shapes of arbitrary objects if many kinds of surfaces are supported.
- The proposed model: The proposed scheme also has a general domain of object shapes. Although this scheme is suitable for smooth objects, its expressive power is as much as that of the B-reps.

Validity

The validity of the representation scheme indicates whether or not an acceptable representation in the scheme certainly designates a valid object.

Table 6.2: The evaluations of the validity

Representation	Evaluation
Octree	Valid if connectivity is not required
Cell decomposition	Hard to establish because of the lack of the structural information of cells
CSG	Valid if the CSG tree is correct
B-reps	Difficult to establish the geometric correctness
The proposed model	Easy to establish the geometric correctness by checking illegal interferences among surface layers

- Octree: All the octrees are valid if no special connectivity requirements are imposed.
- Cell decomposition: While the validity of the octree is assured by its structural properties, that of the cell decomposition is hard to establish because a general cell decomposition is just an unordered set of cells.
- CSG: Every object represented by the CSG tree is guaranteed to be valid.
- B-reps: The validity of the B-reps is quite difficult to establish. While it is easy to preserve the topological validity (i.e., topological relations among the entities), it is hard to enforce geometric correctness because the boundaries are not guaranteed to be closed.
- The proposed model: Since the proposed model holds the object data as closed surfaces, it is easier to enforce the geometric correctness than the B-reps. It is still necessary to detect the illegal interferences among the surface layers, the geometric correctness can be achieved because the system holds the embeddings of the object in 3D space together with its topological skeletons.

Unambiguity

The unambiguity of the representation scheme guarantees that the valid representation determines exactly one object.

- Octree: Up to the limits of resolution, all the octrees define exactly one object.
- Cell decomposition: A valid cell decomposition represents a valid object.
- CSG: Every CSG tree completely determines an object.
- B-reps: Valid boundary representations are unambiguous.
- The proposed model: The valid representation of the proposed model unambiguously determines an object.

Table 6.3: The evaluations of the unambiguity

Representation	Evaluation
Octree	Unambiguous up to the limits of resolution
Cell decomposition	Unambiguous
CSG	Unambiguous
B-reps	Unambiguous
The proposed model	Unambiguous

Table 6.4: The evaluations of the uniqueness

Representation	Evaluation
Octree	Unique if the resolution is fixed
Cell decomposition	Not unique
CSG	Not unique
B-reps	Not unique
The proposed model	Not unique also with different height axes

Uniqueness

The uniqueness of the representation scheme guarantees that an object has only one corresponding valid representation.

- Octree: The representation of the octree is unique if the resolution is fixed.
- Cell decomposition: The representation of the cell decomposition is not unique.
- CSG: The representation of the CSG is not unique.
- B-reps: The representation of the B-reps is not unique.
- The proposed model: The representation of the proposed model is not unique. Since the proposed model is dependent on the height direction, it is also not unique under the changes of the height direction.

Description languages

The description languages are the languages supported in the system based on the representation scheme. It is important to see whether the description languages are directly based on the representation scheme or the object representations are converted from other representation schemes.

Table 6.5: The evaluations of the description language

Representation	Evaluation
Octree	Converted from other representation schemes
Cell decomposition	Converted from other representation schemes
CSG	Textural languages and graphical interface
B-reps	Tedious to manipulate directly Converted from other representation schemes
The proposed model	Textural languages for the topological skeletons Graphical interface for the geometric details

- Octree: The octrees are usually formed by the conversion from other representation schemes.
- Cell decomposition: It is very hard to directly manipulate the cell decompositions of interesting objects. The cell decomposition is generally converted from other representation schemes.
- CSG: The objects in the CSG can be described by textual languages. It is also possible to include a graphical interface into the system.
- B-reps: The B-reps are tedious to describe directly. While the Euler operators are provided as the means of manipulating the boundary data, they are still low level operators that are tedious for users. In B-rep systems, the objects are usually designed by sweeping operations or CSG-like boolean operations and then converted to boundary representations.
- The proposed model: The topological skeletons of an object can be described by textual languages such as the Morse operators. The geometric details of the object are controlled with a graphical interface.

Conciseness

The conciseness shows how large the object representation becomes in terms of storage. Note that this property is often in contradiction with the precision of the representation.

- Octree: The number of the nodes in the octree is proportional to the surface area of the object. The octree representation takes a large amount of storage for complex smooth surfaces in general.
- Cell decomposition: Although the cell decomposition is relatively concise for simple objects, it requires a fair amount of storage for complex smooth surfaces.
- CSG: Although the CSG trees are in principle concise for simple objects, its expressive power has limitations when complex smooth surfaces are handled.
- B-reps: The boundary representation will achieve precise representation of the smooth objects with a small amount of storage, because it holds only the boundary data of the object.

Table 6.6: The evaluations of the conciseness

Representations	Evaluation
Octree	Takes a large amount of storage
Cell decomposition	Takes a fair amount of storage
CSG	Takes a small amount of storage but limited expressive power
B-reps	Takes a small amount of storage
The proposed model	Takes a small amount of storage

- The proposed model: While this model holds the data of the smooth surface in a hierarchical fashion, the final representation of the smooth surface becomes almost the same as that of B-reps. Note that the overhead of the upper levels in the hierarchical representation is relatively small.

Closure of operations

The closure of operations indicates whether the supported operations preserve the validity of object representations. It is necessary to take into account the generality of the supported operations.

- Octree: The octree scheme supports closed operations for problems such as translation, rotation, and boolean operations.
- Cell decomposition: No closed operations are supported in the cell decomposition.
- CSG: Boolean operations are algebraically closed for the CSG trees.
- B-reps: B-reps are usually not closed under boolean operations.
- The proposed model: The proposed model is closed under the Morse operations and their extensions. However, it will not be closed if boolean operations are introduced.

Computational ease

The computational ease means the computational complexity involved in the representation scheme.

- Octree: Many algorithms for the octrees consist of relatively simple operations.
- Cell decomposition: There are no explicit operations of manipulating the cell decompositions.
- CSG: The computational power of important CSG algorithms (such as boundary evaluation) is poor. However, many of their basic steps are very simple.

Table 6.7: The evaluations of the closure of operations

Representation	Evaluation
Octree	Closed under translation, rotation, and boolean operations
Cell decomposition	Has no closed operations
CSG	Algebraically closed under boolean operations
B-reps	Not closed under boolean operations
The proposed model	Closed under the Morse operations Not closed under boolean operations

Table 6.8: The evaluations of the computational ease

Representation	Evaluation
Octree	Simple algorithms
Cell decomposition	Has no explicit operations
CSG	Has a poor set of algorithms while they are simple
B-reps	Requires computational expensive operations for complex smooth objects
The proposed model	Requires computational expensive operations for complex smooth objects

- B-reps: The B-reps are useful for generating graphical output, because they readily include the data needed for driving a graphical display. Algorithms based on the B-reps become quite difficult if the objects become more complex.
- The proposed model: In addition to the problems in B-reps, this model contains some computationally expensive calculations such as variational optimizations.

6.2 Other Representations

This section describes other representations: sweeping and 3D MAT.

Sweep representations

Sweeping techniques are widely used to generate cylindrical objects. Contemporary CAD systems contain the sweeping operations as the fundamental techniques of manipulating boundary data of the object shape. Since the Reeb graph is well suited for the notion of sweeping, the model that associates the sweeping representation with the Reeb graph is a potential technique for designing complex smooth surfaces.

This study uses the techniques of manifold-like patch assembling instead of the sweeping because of the following two reasons. The first reason is that the sweeping is not convenient for representing the smooth surfaces around the branches as described in Appendix F. The second reason comes from the fact that this study tries to implement multiresolution surface design based on the patch gluing. Tai succeeded in incorporating the sweeping techniques with the Reeb graph in her study [118].

3D MAT representations

Another potential representation scheme for smooth surfaces is 3D medial axis transform (3D MAT) representation. Recently, Gelston and Dutta have presented an effective implementation of a surface modeler based on 3D MAT [35]. On the other hand, Reddy and Turkiyyah have presented a technique for extracting the 3D medial axis from the object surface using a generalized Delaunay triangulation [95]. Although the 3D MAT is independent of the coordinates of the space where the objects are embedded, the shape modelers based on 3D MAT are still inconvenient for describing surfaces of complex topological type. On the other hand, the proposed modeling method suffers from the fact that the Reeb graph is variable under the rotational transformations of the objects. Therefore, it is interesting to find the connection between the 3D medial axis and the Reeb graph in order to overcome the limitations of both representation schemes.

Chapter 7

Conclusions

7.1 Contributions

This thesis has presented a new feature-based modeling method for smooth surfaces. As the shape features, critical points such as peaks, pits, and passes are used. In particular, the Reeb graph, which is one of the CPGs, is used to design the topological skeletons of the object shape. The critical points and CPGs work at the upper levels in the hierarchical representations of the smooth objects.

The implementations of the bidirectional operations between the object shapes and shape features, i.e., design by features and feature extraction were presented.

The design by features begins with specifying the topological skeletons of the object shape using the Reeb graph. As a design interface, the iconic representation of the Reeb graph called the embedded Reeb graph is used, which represents the embeddings of the object shape in 3D space. The Reeb graph is manipulated by the Morse operators that describe the way of connecting the critical points. The macro operations were also introduced to avoid the limitations due to the height order of the Morse operators. The scheme for representing the topological skeletons and the implementation of the iconic interface were also presented.

The geometry of the object shape is designed by specifying the flow curves that run on the object surface. From the flow curves, the system automatically constructs the control network that encloses the object. Each vertex of the control network has its own local patch that is designed by using the curve segments of the control network as the geometric constraints. The local patches are then glued together using the manifold mappings in order to form the overall surface of the object.

This thesis has introduced the multiresolution surface design method for describing the detailed geometry of the local patches. The local patches are represented by endpoint-interpolating B-splines and their corresponding wavelets. The shape of the local patch is determined by optimizing the energy function subject to its deformation while preserving the imposed constraints. The multiresolution constraints are solved by converting the constraints at a low resolution level to those at a high resolution level.

The feature extraction, one of the bidirectional operations, has also been implemented in this study. The critical points are extracted so that they satisfy the Euler

formula. The surface network, one of the CPGs, is constructed by tracing ridge and ravine lines on the surface. This thesis has presented an algorithm that converts the surface network to the Reeb graph. Using the extracted Reeb graph, the model of the control network is fit to the object surface, which allows us to change the height axis of the object shape.

7.2 Future Work

The future extensions of the proposed modeling method can be grouped into four categories: (1) introducing more detailed features to the hierarchical representations, (2) providing operations that make better use of the features of smooth surfaces, (3) allowing more flexibility and ease in the surface design, and (4) extending the proposed model to other applications.

Introducing more detailed features to the hierarchical representations

- Distinguishing between the differences of knots and links

  Integrating the theory of spatial graphs [51, 141] with this model is an interesting research theme.

- Extending the model for non-orientable surfaces such as Klein bottles

  The framework based on the atom-molecule graphs [110] is a candidate for such an extension.

- Creating more sophisticated model of degeneracy

  While the presented model is well suited for designing smooth surfaces, it is preferable to extend the model to handle flat surfaces. This needs a more sophisticated model of degeneracy.

- Extracting the critical point configuration graph (CPCG) [82, 83] from discrete samples

  There is a close relationship between the CPCG and the surface network because both of them represent ridge and ravine lines by their edges. While the edge of the surface network indicates a set of connected ridge or ravine lines, that of the CPCG indicates exactly one ridge or ravine line (See Appendix E). Constructing the CPCG from its corresponding surface network is a topic for future research.

Operations that make better use of the features

- Taking visually-appealing mesh samples based on the shape features

  Since the shape features are already obtained in this implementation, a mesh based on such features should be generated. Hierarchical mesh generation [127, 44] based on these features is also a potential application of this study.

- Assigning texture mappings based on the shape features

  The shape features can also be used for defining mappings for textures. Interactive texture assignment [85] based on these features are also an interesting topic for future research.

- Finding relationships between the Reeb graph and the 3D medial axis

  Since the Reeb graph is defined under the specific height function, it is variable under rotational transformations. Finding relationships with the 3D medial axis [10] will overcome this limitation because the 3D medial axis is invariant under rotational transformations.

Allowing more flexibility and ease in the surface design

- Allowing conventional operations in this model

  Allowing the conventional operations such as boolean operations should be considered. Since this thesis has presented the algorithms for extracting shape features from polygonal surfaces, it is possible to perform the boolean operations if the polygonal representation of the resultant surface is available in the system.

- Allowing additional flexibilities in the configuration of the control network

  The configuration of the control network used in the system is still restricted to a regular one that contains only intersections like crossings. It is desirable to allow more flexible configurations such as those containing "T"-shaped intersections.

- Providing intuitive operations in multiresolution shape design

  Although the proposed method of designing multiresolution shapes is efficient, the users require prior knowledge on scale-space theory [137, 66] in order to make full use of the method. The operations that take into account the properties of the scale-space theory are useful for both well-trained and novice users.

- Providing operations for controlling the tolerance of features

  Specifying constraints such as dimensions and tolerances is an important issue in feature-based modeling techniques. In this implementation, geometric constraints can be attached in order to control the shape of the features to meet the specified dimensions. Allowing variations in size such as tolerances is left as future work.

Extending the proposed model to other applications

- Applying the feature extraction algorithms in order to describe time-varying surfaces

  The feature extraction algorithms will be useful also for coding time-varying surface shapes. In particular, the appearance and disappearance of critical points can be effectively described using the algorithms. Describing time-varying shapes will be important for animation techniques.

- Extending the model for artistic design

  The proposed model is a kind of functional approaches for smooth surfaces because it tries to implement systematic design schemes based on shape features. In order to extend the model for artistic design of smooth surfaces, it is important to incorporate the model with surface curvatures. For example, a network of lines of curvature [73] can be used to control the surface curvatures.

Appendix A

Morse Theory

This appendix provides mathematical prerequisites for Morse theory. Refer to the textbooks [80, 32] for more details.

First, let us define critical points of a smooth surface. Let f denote a smooth real-valued function on a smooth manifold M of dimension n. Note that in this thesis, $n = 2$.

Definition A.1 (Critical Point) *The point $p \in M$ is called a critical point of the function f if $\mathrm{grad} f(p) = 0$. The value $f(p)$ is called a critical value of f.*

The matrix of the second partial derivatives of f is defined as

$$H(i,\ j) = \frac{\partial f}{\partial x_i \partial x_j}, \tag{A.1}$$

which is called the Hessian matrix of f.

Definition A.2 (Non-degenerate Critical Point) *The critical point p of f is called non-degenerate if the Hessian matrix is regular at p.*

Definition A.3 (Index) *The index of the critical point p of f is the number of negative eigenvalues of the Hessian matrix at p.*

Since the dimension of the manifold is 2 in this thesis, the indices of the critical points range from 0 to 2. The critical point of the index 2 is called a *peak*, the critical point of the index 1 is called a *pass*, and the critical point of the index 0 is called a *pit*.

Definition A.4 (Morse Function) *The smooth function f on M is called a Morse function if none of its critical points are degenerate.*

A Morse function exists on any smooth compact manifold [80]. Note that in this thesis a height function is used as the Morse function.

Lemma A.5 (Morse Lemma) *Let f be a smooth function on M and let p be a non-degenerate critical point. In the open neighborhood of p, there exist local regular coordinates $x_1, \ldots, x_n$ such that the function f is expressed by*

$$f = -{x_1}^2 - {x_2}^2 - \ldots - {x_\lambda}^2 + {x_{\lambda+1}}^2 + \ldots + {x_n}^2, \tag{A.2}$$

where λ is the index of the critical point p.

Morse lemma is used for investigating the surface behavior around the critical points. When $n = 2$, the lemma can be written as follows.

$$f(x, y) = \begin{cases} -x^2 - y^2 & \text{for a peak} \\ -x^2 + y^2 & \text{for a pass} \\ x^2 + y^2 & \text{for a pit} \end{cases} \tag{A.3}$$

Before going into the details of Morse Theory, let us define cells and CW-complexes as follows [49].

Definition A.6 (n-Cell) *A topological space e^n is called an n-cell if it is homeomorphic to an n-dimensional disk $\boldsymbol{D}^n$.*

Definition A.7 (CW-Complex) *Let X be a Hausdorff space. We call the partition of X cell decomposition $\mathcal{E}$ if X is a set of pairwise disjoint cells. Suppose that the pair $(X, \mathcal{E})$ consists of a Hausdorff space X and a cell decomposition $\mathcal{E}$ of X. $(X, \mathcal{E})$ is called a CW-complex if the following three conditions are satisfied.*

(1) **(Characteristic Maps)** *For each n-cell $e \in E$ there is a continuous map $\Phi_e : D^n \to X$ taking $\mathring{D}^n$ (interior of D^n) homeomorphically onto the cell e and S^{n-1} into the union of the cells of dimension at most $n - 1$.*

(2) **(Closure Finiteness)** *The closure $\overline{e}$ of each cell $e \in \mathcal{E}$ intersects only a finite number of other cells.*

(3) **(Weak Topology)** *$A \subset X$ is closed if and only if every $A \cap \overline{e}$ is closed.*

The following theorem is proven by applying the *homology theory* to CW-complexes [75].

Theorem A.8 (Euler Characteristic) *Let K be a finite CW-complex. The Euler characteristic satisfies the following equation:*

$$\chi(K) = \sum_n (-1)^n \#(e^n) \tag{A.4}$$

where $\#(e^n)$ represents the number of n-cells.

In the case of 2-dimensional manifolds, the above equation is reduced to the following equation.

$$\chi(K) = \#(e^0) - \#(e^1) + \#(e^2) \tag{A.5}$$

(A.5) is called the *Euler formula* in this thesis. The Euler characteristic is a topological invariant and is used for verifying the consistency of object data.

Morse theory is explained as follows.

Theorem A.9 (Morse Theory) *Let f be a Morse function on the smooth compact closed connected manifold M. M is then homotopy equivalent to a finite CW-complex which is a set of cells whose dimensions correspond to the indices of its critical point respectively. In other words,*

$$M \simeq e^{r_1} \cap e^{r_2} \cap \ldots \cap e^{r_k}, \tag{A.6}$$

where r_1, r_2, ..., and r_k are the indices of the critical points of M.

From the theories of algebraic topology, compact 2-dimensional manifolds can be classified into any compact surfaces that are homeomorphic to either [103, 75]

- spheres,
- connected sums of tori, or
- connected sums of projective planes.

Morse theory adds the information about surface embeddings to the above topological classification.

Appendix B

Definition of the Reeb Graph

The mathematical definition of the *Reeb graph* is described as follows. Note that this definition is derived from [107]. Let $f : M \to \boldsymbol{R}$ be a real valued function on a compact manifold M. The Reeb graph of f is the quotient space of the graph of f in $M \times \boldsymbol{R}$ by the equivalence relation given below:

$$(X_1, f(X_1)) \sim (X_2, f(X_2)) \tag{B.1}$$

holds if and only if $f(X_1) = f(X_2)$ and X_1 and X_2 are in the same connected component of $f^{-1}(f(X_1))$. That is, the two points on the graph $(X_1, f(X_1))$ and $(X_2, f(X_2))$ are represented as the same connected component of the inverse image of $f(X_1)$ (or $f(X_2)$). All points that belong to the same equivalent class of the original space are represented as a node in the quotient space such as the Reeb graph. Figure 1.2 illustrates the Reeb graph of a torus.

Appendix C

Endpoint-interpolating Cubic B-spline Wavelets

The synthesis matrices $\boldsymbol{P}^{(k)}$ and $\boldsymbol{Q}^{(k)}$ for endpoint-interpolating cubic B-spline wavelets are given below [94, 115, 116].

$$\boldsymbol{P}^{(1)} = \frac{1}{2}\begin{pmatrix} 2 & & & \\ 1 & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \\ & & & 2 \end{pmatrix} \qquad \boldsymbol{Q}^{(1)} = \sqrt{7}\begin{pmatrix} 1 \\ -2 \\ 3 \\ -2 \\ 1 \end{pmatrix} \tag{C.1}$$

$$\boldsymbol{P}^{(2)} = \frac{1}{16}\begin{pmatrix} 16 & & & & & \\ 8 & 8 & & & & \\ & 12 & 4 & & & \\ & 3 & 10 & 3 & & \\ & & 4 & 12 & & \\ & & & 8 & 8 & \\ & & & & 16 & \end{pmatrix} \qquad \boldsymbol{Q}^{(2)} = \sqrt{\frac{315}{31196288}}\begin{pmatrix} 1368 & \\ -2064 & -240 \\ 1793 & 691 \\ -1053 & -1053 \\ 691 & 1793 \\ -240 & -2064 \\ & 1368 \end{pmatrix} \tag{C.2}$$

$$
\boldsymbol{P}^{(k\geq 3)} = \frac{1}{16}
\begin{pmatrix}
16 & & & & & & & & & & \\
8 & 8 & & & & & & & & & \\
 & 12 & 4 & & & & & & & & \\
 & 3 & 11 & 2 & & & & & & & \\
 & & 8 & 8 & & & & & & & \\
 & & 2 & 12 & 2 & & & & & & \\
 & & & 8 & 8 & & & & & & \\
 & & & 2 & 12 & \cdot & & & & & \\
 & & & & 8 & \cdot & 2 & & & & \\
 & & & & 2 & \cdot & 8 & 2 & & & \\
 & & & & & & 12 & 8 & & & \\
 & & & & & & 8 & 11 & 3 & & \\
 & & & & & & 2 & 4 & 12 & & \\
 & & & & & & & & 8 & 8 & \\
 & & & & & & & & & 16 &
\end{pmatrix}
\tag{C.3}
$$

$$
\boldsymbol{Q}^{(3)} =
\begin{pmatrix}
6.311454 & & & & \\
-9.189342 & -1.543996 & & & \\
7.334627 & 4.226722 & 0.087556 & & \\
-3.514553 & -5.585477 & -0.473604 & -0.000155 & \\
1.271268 & 6.059557 & 1.903267 & 0.019190 & \\
-0.259914 & -4.367454 & -4.367454 & -0.259914 & \\
0.019190 & 1.903267 & 6.059557 & 1.271268 & \\
-0.000155 & -0.473604 & -5.585477 & -3.514553 & \\
 & 0.087556 & 4.226722 & 7.334627 & \\
 & & -1.543996 & -9.189342 & \\
 & & & 6.311454 &
\end{pmatrix}
\tag{C.4}
$$

$$Q^{(k\geq 4)} = \sqrt{\frac{5 \cdot 2^k}{675221664}}$$

$$\begin{pmatrix}
25931.200710 & & & \\
-37755.271723 & -6369.305433 & & \\
30135.003012 & 17429.266054 & 385.797044 & \\
-14439.869635 & -23004.252368 & -2086.545605 & -1 \\
5223.125428 & 24848.487871 & 8349.373420 & 124 \\
-1067.879425 & -17678.884301 & -18743.473059 & -1677 \\
78.842887 & 7394.685374 & 24291.795239 & 7904 \\
-0.635830 & -1561.868558 & -18420.997597 & -18482 \\
 & 115.466347 & 7866.732006 & 24264 \\
 & -0.931180 & -1668.615872 & -18482 \\
 & & 123.378671 & 7904 \\
 & & -0.994989 & -1677 \\
 & & & 124 \\
 & & & -1
\end{pmatrix}$$

$$
\begin{array}{rrrrrr}
-1 & & & & & \\
124 & & & & & \\
-1677 & & & & & \\
7904 & & & & & \\
-18482 & -1 & & & & \\
24264 & 124 & & & & \\
-18482 & -1677 & -0.994989 & & & \\
7904 & 7904 & 123.378671 & & & \\
-1677 & -18482 & -1668.615872 & -0.931180 & & \\
124 & 24264 & 7866.732009 & 115.466347 & & \\
-1 & -18482 & -18420.997597 & -1561.868558 & -0.635830 & \\
 & 7904 & 24291.795239 & 7394.685374 & 78.842887 & \\
 & -1677 & -18743.473059 & -17678.884301 & -1067.879425 & \\
 & 124 & 8349.373420 & 24848.487871 & 5223.125428 & \\
 & -1 & -2086.545605 & -23004.252368 & -14439.869635 & \\
 & & 385.797044 & 17429.266054 & 30135.003012 & \\
 & & & -6369.305453 & -37755.271723 & \\
 & & & & 25931.200710 &
\end{array}
\qquad \text{(C.5)}
$$