INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Automated Generation of Control Skeletons for Use in Animation

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Lawson Wade, B.A., M.S.

* * * * *

The Ohio State University

2000

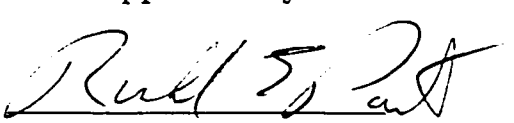
Dissertation Committee:

Dr. Richard E. Parent, Adviser

Dr. Wayne E. Carlson

Dr. Rephael Wenger

Approved by



Adviser

Department of Computer and Information Science

UMI Number: 9994950

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

The animation of an articulated figure is typically accomplished through the use of a corresponding control skeleton. Although the control skeleton is an effective tool, the manual construction of the skeleton can be a laborious process often requiring several hours of work and a fair degree of proficiency with the animation software used.

The focus of the research described here is the automatic generation of such control skeletons. To this end, two solutions to the problem are presented, one general and one specific. In both cases, the input is required to be a set of polygonal data that defines the figure, and the output is a description of a control skeleton to be used in animating that figure.

The general solution is widely applicable; it makes very few assumptions about the figure given as input or about the type of control skeleton that should be generated. A system is described that divides the problem into a series of steps, each of which is performed automatically. The basic process involves discretizing the figure, approximating its medial surface, and using that surface to construct a control skeleton. The system can produce a reasonably good control skeleton for any of a variety of figures in as little as one or two minutes on a low-end PC.

The specific solution builds upon the general one but is geared toward producing more desirable skeletons for the very common case involving human-like and animal-like figures. Certain assumptions are made about the figure and about the type of control skeleton desired. In addition, heuristics based upon human and animal anatomy are invoked to adjust the control skeleton so that it is more anatomically appropriate. The motivation for this solution is the belief that a more anatomically appropriate control skeleton allows for more natural looking movement of a human or animal-like figure.

Partly to support that claim, the system can produce geometry for individual bones that might function as the anatomical skeleton of the figure. This skeletal geometry can form the foundation for additional anatomical modeling that might add more realism to the animation of the figure.

To Kathy

ACKNOWLEDGMENTS

So many people have helped me in so many ways. I will always be grateful for their insight, enthusiasm, support, and wisdom.

I would like to thank my adviser, Rick Parent, for his assistance with this research and for his support throughout my graduate studies. I have especially appreciated his willingness to listen to my ideas and to share his vast knowledge and experience in computer graphics. I also wish to thank the other members of my committee – Rafe Wenger and Wayne Carlson. Rafe has been an excellent teacher and has been quite helpful and supportive of my research efforts. Wayne introduced me to the ACCAD environment and has provided me with a number of research assistantships; I have greatly appreciated his encouragement and advice.

I especially want to thank Meg Geroch, Matt Lewis, and Pete Carswell for countless lengthy discussions about the research. Meg has been tremendously helpful with her careful review of the dissertation, and Matt generously provided several humanoid data models for my use. I also want to say thanks to Meg and Matt for encouraging me to continue the research when everything seemed so bleak. Pete has offered me various research assistantships; it has been a joy to work with him and to join him in so many intriguing conversations on mathematics.

I would like to thank all of the teachers I have had. Of special mention are Kerm Almos, Kurt Anderson, David Block, Larry Booth, Harold Brockman, George

Carver, Joanne Dawson, John Detrick, Tom Gearhart, Tim Hildreth, Jerry Izzo, Dennis Kapenga, Carol Krell, Larry McElwee, Kevin Michael, Kelly Moody, Scott Neal, Larry O'Flynn, Jordan Pollack, Peggy Rinehart, Gary Ross, Lou Schultz, Steve Shaffer, Edie Sidwell, Neelam Soundarajan, Ken Supowit, David Trowbridge, and Howard Wilson.

I wish to thank the staff of the Computer and Information Science Department: Tom Fletcher, Sandy Hill, Marty Marlatt, Elizabeth O'Neill, and Eleanor Quinlan. Ellie was especially helpful during my time as a graduate teaching associate.

I would like to thank the staff and faculty at ACCAD, both current and previous: Chuck Csuri, Aline Davis, Viki Dennis, Barb Helfer, Ruedy Leeman, Steve May, Mike Miller, Phil Ritzenthaler, Elaine Hamilton Smith, Steve Spencer, and Traci Temple. Steve May offered valuable comments and suggestions and created the AL programming language. Many of the figures in this document and numerous supporting animations were created with AL and rendered with RenderMan®.[1]

I would also like to thank the other graduate students I have known: from CIS, Debashis Basak, Paolo Bucci, Matt Camuto, Al Fencl, Mark Fontana, Neeraj Gupta, Julie Hartigan, Scott King, Nathan Loofbourrow, Raghu Machiraju, Neal McDonald, Torsten Moeller, Klaus Mueller, Saty Raghavachary, David Reed, Doug Roble, Kevin Rodgers, Ferdi Scheepers, Carl Schuyler, Naeem Shareef, Ed Sindelar, Karan Singh, Sara Susskind, Ed Swan, Suba Varadarajan, and Pete Ware; from Biomechanics, Kinda (Khalaf) Abdullah; and from ACCAD, Julie Apley, Ian Butterfield, Wooksang Chang, Pete Gerstmann, John Gladden, Heath Hanlin, Wobbe Koning, Heesung Koo, Melissa Kupper, Zil Lilas, Janet Lucroy, Brandon Morse, Todd Sines,

[1]RenderMan is a registered trademark of Pixar Animation Studios.

Clarke Stallworth, Scott Swearingen, Nathania Vishnevsky, and John Warren. Being with such friendly, helpful, funny, intelligent, and inspiring people has been one of the best things about graduate school. I particularly want to thank Pete Gerstmann for creating a number of data models for the research and Ian Butterfield for helping to make the last few months more enjoyable with his humor and conversation. A special thanks goes to John Warren for the informative and enlightening brain-storming session from which this research was spawned.

Finally, I would like to acknowledge my family and friends – I greatly appreciate their love and support. To Mike and Julie Reid and John and Janene Metzger, thank you for being such good friends and for cheering me on. To Larry and Charlotte, Mark and Angie, and Jen and Bob, thank you for caring. To my parents, Robert and Ann Wade, thank you for your guidance and encouragement and for sharing the wisdom that only one's parents seem to have. To Sam and Tina and to John and Chrissy, thank you for your unquestioning acceptance and unwavering confidence. To Michael, thanks for allowing me the joy of being a parent. And to my wife, Kathy, thank you for your patience and understanding, your cheers and celebration, the occasional kick in the butt, and for making the world a much happier place.

VITA

May 12, 1969	Born - Dayton, Ohio
1991	B.A. Computer Science/Mathematics *Summa Cum Laude* Capital University, Columbus, Ohio
1993	M.S. Computer and Information Science The Ohio State University, Columbus, Ohio

PUBLICATIONS

Research Publications

Lawson Wade and Richard E. Parent. "Fast, fully-automated generation of control skeletons for use in animation". In *Proceedings of Computer Animation 2000*. 2000.

Lawson Wade and Richard E. Parent. "Fast, fully-automated generation of control skeletons for use in animation". *Technical Report* OSU-ACCAD-9/99-TR3. The Ohio State University, Advanced Computing Center for the Arts and Design, 1999.

Lawson Wade and Richard E. Parent. "Practical issues for implementation of a DDT algorithm for polyhedra". *Technical Report* OSU-ACCAD-5/99-TR1, The Ohio State University, Advanced Computing Center for the Arts and Design, 1999.

K.A. Khalaf, M. Parnianpour, L. Wade, and S.R. Simon. "Feature extraction and modeling of the variability of performance in terms of biomechanical motion patterns during MMH tasks". In *Proc. Rocky Mountain Bioengineering Symposium*, pages 35–40, 1997.

K.A. Khalaf, M. Parnianpour, L. Wade, P.J. Sparto, and S. Simon. "The importance of dynamic strength models for proper ergonomic task analysis". In *Proc. International Society for the Study of the Lumbar Spine*", pages 166-168, 1996.

K.A. Khalaf, M. Parnianpour, L. Wade, and P.J. Sparto. "Biomechanical simulation of manual multi-link coordinated lifting". In *Proc. The Fifteenth Southern Biomedical Engineering Conference*, pages 197–198, 1996.

David M. Reed, Lawson Wade, Peter G. Carswell, and Wayne E. Carlson. "Particle tracing in curvilinear grids". In *Visual Data Exploration and Analysis II (Proc. SPIE 2410)*, Georges G. Grinstein and Robert F. Erbacher, editors, pages 120–128, 1995.

David M. Reed, Lawson Wade, Peter G. Carswell, and Wayne E. Carlson. "Particle tracing in curvilinear grids". *Technical Report* OSU-ACCAD-6/94/TR1, The Ohio State University, Advanced Computing Center for the Arts and Design, 1994.

Peter Carswell, Wayne Carlson, David Reed, W. Seun, and Lawson Wade. "kitchenVIEW: an interactive interface to heat flow solutions in commercial kitchens". *Technical Report* OSU-ACCAD-1/93/TR4, The Ohio State University, Advanced Computing Center for the Arts and Design, 1993.

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in:

Computer Graphics	Dr. Richard E. Parent
Algorithms	Dr. Rephael Wenger
Artificial Intelligence	Dr. Jordan Pollack

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Articulated figures abound in computer graphics. They appear most frequently in the areas of character animation [CHP89, Mae96, VSC00] and human figure modeling and animation [BPW93, SPCM97, WG97], and their study typically garners at least a chapter or two in most books on computer animation in general [MTT90, WW92]. Indeed, there is at least one book devoted entirely to the topic of articulated figures [BBZ91].

The animation of such a figure is generally accomplished through the use of a *control skeleton* - an articulated structure of segments and joints combined with information detailing how the surface geometry of the figure is anchored to that structure. The control skeleton is sometimes referred to as the *skeleton rig* or the *IK skeleton*. The latter term is derived from the frequent use of inverse kinematics when posing or animating the structure.

When an animator is faced with the problem of animating a complex model, he or she can create a control skeleton that corresponds to the model, specifying each individual segment and joint and attaching parts or regions of the model to nearby segments. Next, the animator can specify a set of joint values in order to pose the control skeleton, and the attachment information is then used to reposit on

the original figure in a corresponding manner. Animation of the figure can thus be performed by animating the figure's skeleton and updating the geometry of the figure in turn.

1.1 Motivation

An *articulated figure* (or *articulated model*) consists of two main components: the figure to be animated and a control skeleton to be used to animate it. The figure to be animated is usually some geometric model which, by itself, is a motionless object. It is the control skeleton which imparts articulation and flexibility to that object.

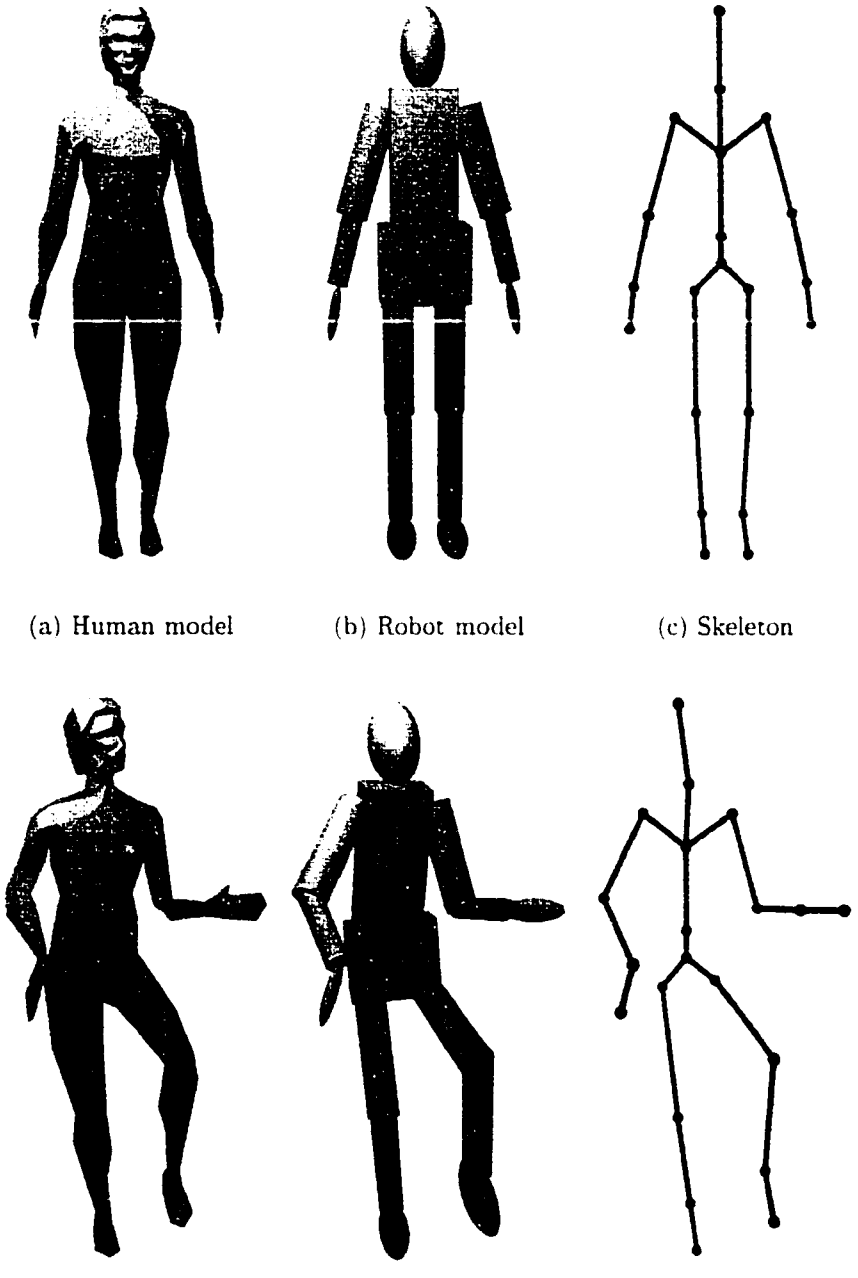
Articulated figures can be divided into two basic classes according to how the geometry of the figure is defined. In the first class, the geometry is a single model that is usually a surface or shell considered to be the skin of the figure. When the figure is animated, this skin is deformed to match the pose of the skeleton. A figure of this kind will be termed a *continuous model*. In the second class, the geometry is a collection of component models. Normally each component is intended to be a separate movable part of the articulated structure. During animation, each individual component is typically transformed in rigid fashion to align with a related part of the control skeleton, though occasionally some minor deformation of the components is performed near the joints. Even though the separate components suggest the structure of an articulated figure, a control skeleton still must be paired with the geometry in order to form an articulated figure capable of hierarchical movement of those components. A figure belonging to this second class will be referred to as a *segmented model*.

Figure 1.1 shows examples of the two classes. The human figure is modeled with a single polygonal mesh. The robot model is a composition of primitive shapes: two boxes for its trunk, a couple of cylinders for each of its arms and legs, and a few ellipsoids to represent its head, hands, and feet. Note that these shapes are simply transformed when the robot is posed, while in contrast the polygonal mesh for the human figure is deformed when it is posed.

The control skeleton defines the movement capabilities of an articulated figure. The control skeleton is composed of an interconnected structure of segments and joints as well as a collection of attachment information that defines how the geometry of the figure is anchored to that structure.

A *segment* is usually a rigid form that serves to provide locations for joints and anchors. Segments are connected to each other by *joints*, which are the points of articulation for the control skeleton. Typically, a joint is used to connect a pair of segments. The exception is the *root joint*, which is the foundational joint for the structure, providing a pivot point between the control skeleton and the world in which it exists. A joint has a fixed position relative to each the segments it adjoins, and it provides a set of degrees of freedom (DOFs) that constrains the relative motion between its adjacent segments. To obtain a pose for the interconnected structure, a user may specify a set of values for the DOFs at each joint; these values are then used to compute the relative configuration for each connected pair of segments and thus the overall configuration of the structure.

The attachment information is the means of connecting the surface of the figure (that is, its geometry) to the structure of segments and joints. Although methods of attachment vary, a common method uses special points which will be called *anchors*.



(a) Human model (b) Robot model (c) Skeleton

(d) Human (posed) (e) Robot (posed) (f) Skeleton (posed)

Figure 1.1: The two classes of articulated figures. The human (a), a continuous model, is a single polygonal mesh, while the robot (b), a segmented model, is composed of geometric primitives. Both models are anchored to a control skeleton structure (c). When the skeleton is posed (f), the models are deformed accordingly (d, e).

An anchor is a point, defined in the local coordinate frame of a segment, that relates to the position of a point on the surface of the figure. The local position of an anchor is thus fixed, but the global (or world-space) position of an anchor will vary with the movement of the segment containing that anchor. A set of one or more anchors on one or more segments may be used to attach a single surface point to the structure. Such a set can then be used to recompute the position of its corresponding surface point: when only one anchor is used, its global position is copied as the updated location of the surface point; when two or more anchors are used, their global positions are combined in a weighted average to obtain the updated location of the surface point. Usually, if the surface geometry of the figure is defined as a set of polygons, then each vertex will have a dedicated set of anchors; if the geometry is defined using spline surfaces, then each control point will have a dedicated set of anchors. Whenever the structure of segments and joints is posed in a new configuration, each set of anchors is used to reposition its corresponding surface point, and thus the entire surface of the figure is remolded accordingly.

Undoubtedly the control skeleton is a useful tool integral to the animation of articulated figures. Its use simplifies the problem of posing or animating a complex geometric object into the problem of specifying values for the joints of an easily understood structure which has an obvious and inherent correspondence to the more complex object.

It is unfortunate, then, that the construction of a suitable control skeleton for use with a given geometric model can be such a tedious and time-consuming task. Many sophisticated modeling and animation packages include support for working with articulated figures, usually providing users with an interface that enables them

to construct a control skeleton for use in animating a model. Nonetheless, the creation of a control skeleton can be a laborious undertaking sometimes requiring several hours of work, and a user typically must possess a fair degree of proficiency with a package to obtain even rudimentary motion via a control skeleton.

Equally unfortunate is the fact that so much of the monotonous work is often needlessly replicated when constructing similar control skeletons for different objects. Such is often the case when producing control skeletons for such common figures as humans and animals, which make up a significantly large portion of the articulated models created.

Examination of an articulated figure usually reveals a special relationship between the figure's geometry and the underlying control skeleton. Notable protrusions of the geometry are typically reflected in the control skeleton by segments that extend into those protrusions, and junctions of geometric parts are generally marked by specific joints within the skeleton. In addition, areas of branching in the geometry often correspond to branching configurations of segments and joints in the control skeleton. This is no coincidence. The effective manipulation of the geometry through use of the control skeleton requires a similarity in the structure of both.

Such a close relationship between the geometry and the control skeleton suggests that some type of automated process can be used to generate the control skeleton. Furthermore, automation can alleviate the monotony and tedium and speed up the creation process.

1.2 Problem Description

Depending on the ordering of construction tasks, there are two basic approaches to creating articulated figures. If an animator knows the articulation he or she desires of the figure, then the animator can first build its skeleton - or rather the articulated structure of segments and joints. The actual geometry of the figure is added later, usually in conjunction with information as to its attachment to the structure. This completes the construction of the control skeleton and the articulated figure as a whole.

The other method, likely the more common of the two, involves creating the geometry first. Most people are probably anxious to model a figure they have envisioned, wanting quickly to realize the intricate details of the shape and quality of its surface, and delaying the possibly more mechanical task of building segments and joints. Or perhaps the model already exists, having been created by someone else at an earlier time, possibly through some digitizing process, and now a person wishes to animate the model. Presumably the animator already has a rough idea of how the figure should move and a reasonable mental sketch of the placement of most of its segments. If the shape of the figure is human-like or animal-like, then that mental image of its movement capabilities is likely very clear, as the animator no doubt has observed countless humans and animals in action.

The research presented here deals with the latter method and assumes that the geometric model has already been created. Specifically, it addresses the problem of automatically generating a control skeleton for a given model. It is applicable regardless of whether the model was created with the intention of articulated movement or whether it was originally developed as only a static model.

The problem is confronted from two perspectives. In the general view, few restrictions are placed on the form of the given model. The objective is simply to produce, in an automated fashion, a corresponding control skeleton for use in animating that model. The control skeleton generated is expected to resemble the model both in structure and complexity.

The specific view of the problem assumes that the model is human-like or animal-like, as is often the case when creating articulated figures. Here, correspondence between the model and the generated skeleton should be stronger, because certain structural assumptions can be made. The resulting articulated figure (that is, the model in conjuction with the control skeleton) should be capable of more natural looking motion - motion that one might expect of a human or animal.

1.3 Overview of the Solution

In response to the two views of the problem, both a general and a specific solution are presented. Both solutions have been implemented within the context of a single system which is described in the text. The input to the system is a set of polygonal data that defines the geometry of the figure. The output is a set of files that functions as a description of the control skeleton for the figure. Various methods for visualization of the control skeleton are available within the system.

The general solution is widely applicable, because it makes very few assumptions about the form of the figure given as input or about the type of control skeleton that should be generated. The solution consists of a series of steps, each of which is performed automatically. First, the figure is converted to a voxel representation, and an approximation to its discrete medial surface is constructed. Next the medial surface

is simplified into a tree structure, and that tree is divided into an interconnected structure of segments and connecting joints. Finally, the voxel representation and the medial surface are used to generate anchors for attaching the vertices of the polygonal data to the segments.

The specific solution builds upon the general one; however, it is geared toward producing a more desirable skeleton for the case when a human-like or animal-like model is provided as input. In this case, certain assumptions are made about the model and its form - assumptions such as how it is posed and about the relative sizes of its features. Assumptions are also made about the structure of the control skeleton that should be generated, leading the system to produce tree-structured skeletons with some degree of bilateral symmetry. Many of the same steps involved in the general solution are executed; but heuristics based upon human and animal anatomy are also invoked to adjust the control skeleton so that its segments and joints correspond more closely to the bones and joints of the anatomical skeleton that might be expected in such a being. In short, the heuristics are used in an attempt to make the control skeleton more anatomically appropriate and thus capable of more natural looking motion.

Although the system is not without its shortcomings, it is shown to produce a reasonably good control skeleton for any of a variety of figures in as little as one or two minutes on a low-end personal computer (specifically, a PC with a 133 MHz Intel® Pentium® processor).[2] It is capable of producing a slightly better and more representative control skeleton when allowed to run for a longer period, but this is mainly

[2]Pentium is a registered trademark of Intel Corporation.

the result of a trade-off between the execution time and the level of discretization of the model.

When the input provided is a human-like or animal-like figure, then the system is shown to produce a control skeleton with an observable anatomically justified quality. In further validation of the anatomical basis, the system can generate geometric models of bones for visual realization of an anatomical skeleton that corresponds to the anatomically based control skeleton. This skeletal geometry can form the foundation for additional anatomical modeling, providing attachment points for layers of muscles, fatty tissue, and skin. Such anatomically based modeling has been shown to add more realism to the animation of the figure. Chadwick et al. appear to have acted as the pioneers of the layered construction approach for articulated figures [CHP89]. More recent works by Scheepers et al. and by Wilhelms and Van Gelder have demonstrated the realism achievable through closer adherence to principles of anatomically based modeling [SPCM97, WG97]. For further discussion of anatomically based modeling in the literature, see Section 2.4.

The attempt has been made to keep the implementation of the two solutions as general as possible. Nevertheless, due to the complexity of the task at hand and to further complications involved in creating a useful but general system, various assumptions and simplifications have been made and are documented throughout the text.

1.4 Overview of the Document

This chapter is a brief introduction to the material presented in this dissertation. Section 1.1 defines the concept of the articulated figure, provides general information

about its key component, the control skeleton, and discusses factors which have motivated this research. Section 1.2 describes the problem addressed by this research and how it fits in to the methods by which articulated figures are created. Section 1.3 then presents an overview of the general and specific solutions to the problem, introducing the concept of anatomically based modeling and its relationship to the research. Finally, the current section gives a short summary of the contents of each chapter of the document.

Chapter 2 provides relevant background information. The geometric concepts of distance maps and the medial surface are described, along with their relationship to the general problem of automated control skeleton generation. Other work in the area of control skeleton generation and anatomical modeling is also presented.

Chapters 3 and 4 discuss the discrete geometry algorithms developed for the research. The first algorithm computes a close approximation to the Euclidean distance map, and the second algorithm finds the discrete medial surface for a voxelized object. These two algorithms are the main underlying components of the general solution.

Chapter 5 details the various steps involved in the general solution to the problem. It describes the discretization of the model and how that discretization is used both in the creation of the articulated structure of segments and joints as well as in the appropriate anchoring of the geometric model to that structure.

Chapter 6 introduces the anatomical knowledge upon which the specific solution to skeleton generation is based. Comparative anatomy of vertebrates is discussed, specifically with respect to the structure of the skeleton and musculature.

In Chapter 7, the assumptions behind the specific solution are discussed. These assumptions form the foundation for the heuristics developed to help identify the

anatomical features of the model. Implementation issues related to the application of the heuristics are mentioned.

Chapter 8 describes further use of anatomical knowledge in the context of control skeleton generation for human-like and animal-like figures. More heuristics are developed and applied in order to create more anatomically appropriate control skeletons. Again, implementation issues are presented.

Chapter 9 discusses the creation of geometric models for anatomical components. It describes a generalized model for component models of bones and proposes a generalized model for the musculature as well. In doing so, it demonstrates the visual realization of the general anatomical skeleton as an enhancement to the specific solution.

Chapter 10 summarizes the research and lists the main contributions. Possibilities for extending or enhancing the research are suggested.

CHAPTER 2

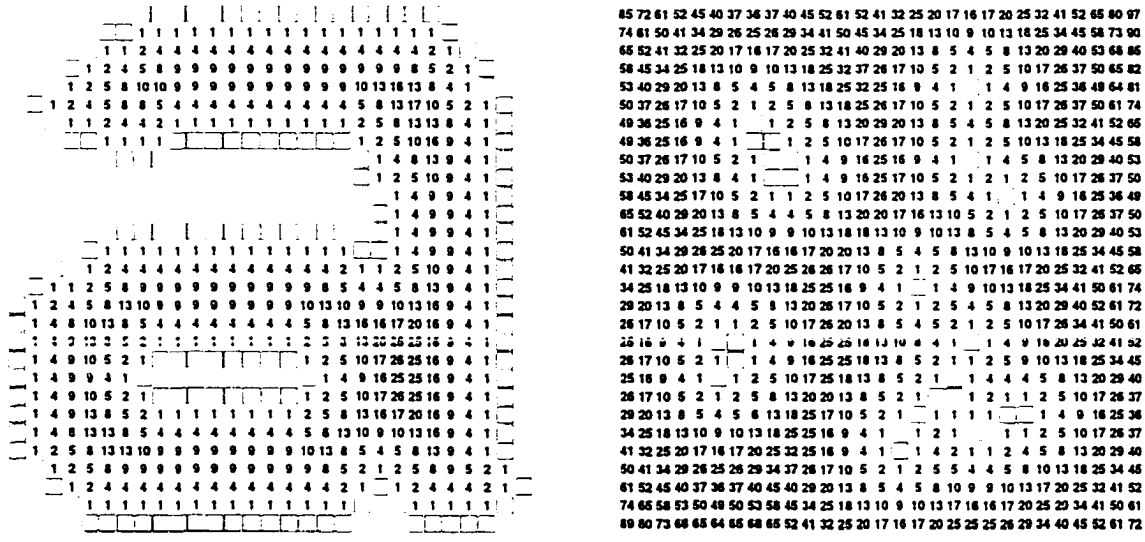
BACKGROUND

This chapter describes various concepts related to the research, citing relevant publications in each area. Section 2.1 introduces the distance map, and Section 2.2 presents the medial axis and medial surface. Together these two sections provide the background for the discrete geometry forming the foundation of the research. In Section 2.3, other work in the area of control skeleton generation is discussed. Finally, Section 2.4 presents a brief look at the ever-expanding area of anatomically based modeling.

2.1 Distance Maps

The *distance map* is a frequently used tool in computer graphics. It is especially useful in areas such as image processing, image analysis, computer vision, and pattern recognition, and it often arises wherever discretization is employed. The distance map is sometimes referred to as the *distance transform* or the *digital distance transform*, although it is more accurate to use these "transform" terms to refer to the process used to generate the distance map.

The input to the distance transform is a grid of discrete points with each point marked as being either a *feature point* or a *background point*. The output of the

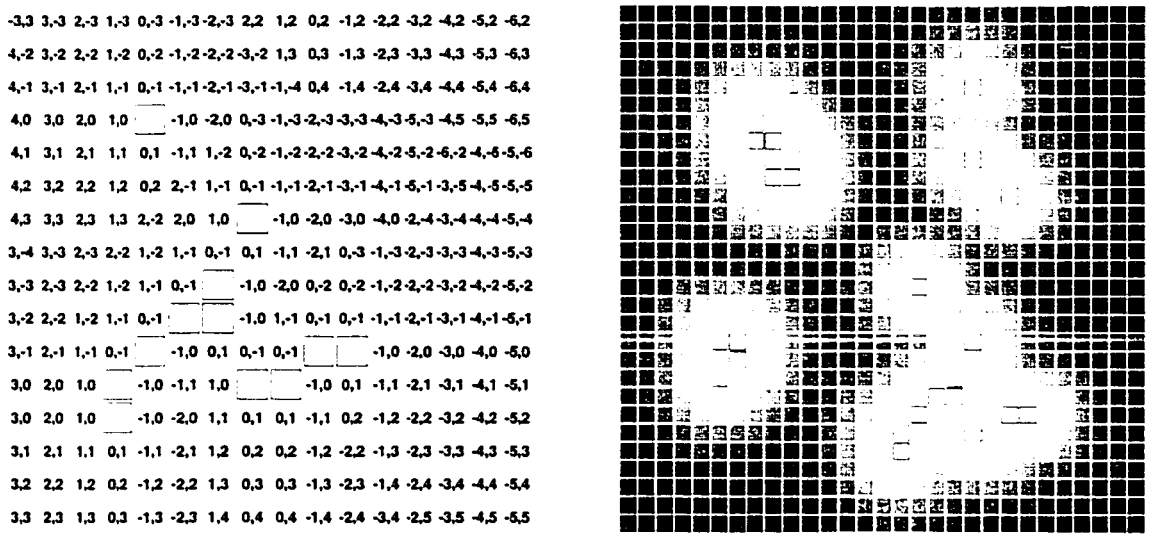


(a) (b)

Figure 2.1: The two forms of the 2D Euclidean distance map. The empty squares represent the feature points, and the shaded squares represent the background points, with each value being the square of the Euclidean distance to the nearest feature point. In (a), the feature points form a boundary surrounding the background points, and the distance map provides information about the internal area of a discretized letter *a*. In (b), the feature points are contained within an array of background points; such a distance map might provide useful information concerning the locations of oil deposits buried beneath some terrain. Note that the two maps are unrelated; they merely serve to illustrate the different manners in which the distance map typically appears.

distance transform is the distance map, which is a corresponding grid with a label for each background point reflecting its relative distance to the nearest feature point. Note that this "relative distance" may or may not be the Euclidean distance between the points; if it is, then the map is called a *Euclidean distance map* (EDM).

Distance maps appear in either of two forms according to whether the feature points form a simple boundary around the set of background points or whether the feature points are contained amidst a field of background points. The formulations



(a) (b)

Figure 2.2: Vector and grayscale displays of the distance map in Figure 2.1(b). In (a), a portion of the map (roughly the lower right quadrant) has been relabeled using vectors pointing to the nearest feature point: in (b), the entire map is redisplayed using grayscale values.

have different uses, and the selection of which form to use is entirely dependent on the application: the first form focuses on the internal structure of an object: the second form permits the examination of an object or objects in relationship to the surrounding environment or to each other. The two formulations are illustrated in Figure 2.1 along with the concept of the Euclidean distance map.[3]

As is the case with Figure 2.1, the distance map is often displayed as a grid of values. An alternative is to show the coordinates of vectors that refer to the nearest

[3]The design for Figure 2.1 was borrowed from one of the many excellent diagrams appearing in a paper by Ogniewicz and Kübler [OK95]; for a summary of the paper, see page 31.

background points, as is done in Figure 2.2(a). The distance map can also be viewed as a grayscale image (see Figure 2.2(b)).

How a distance map is displayed does not necessarily correspond to how it was computed. The actual computation may involve the use of either scalar values or vectors. Scalars suffice when only the magnitude of the distance is needed. If it is also necessary to know the direction to the nearest feature point, then vectors are typically used, in which case the computation is sometimes referred to as a *vector distance transform* or a *nearest-neighbor transform*.

Integer operations are usually preferable in distance map construction. For maps involving vectors, this comes naturally. For maps with scalars, however, simple tricks are sometimes used: in the case of the EDM, for example, instead of working directly with the Euclidean distance, it can be just as convenient to use the square of the Euclidean distance (and this permits storage of the distance values as integers).

For many applications that utilize distance maps, working with the Euclidean distance map is ideal. There are several potential pitfalls involved in the construction of an exact Euclidean distance map, however, and this has caused many people instead to use fast or easily-implemented algorithms that compute useful approximations to the EDM. Some of these approximations utilize non-Euclidean metrics such as manhattan distances, chessboard distances, octagonal expansions, or chamfer metrics (for a discussion of these, see Paglieroni [Pag92]). Nevertheless, efficient algorithms for correct computation of the EDM in two dimensions do exist, some even having linear time complexity with respect to the number of grid points [BGKW95]. The problem has also led people to use non-rectangular grids; for instance, Vincent offers an effective solution for computing the EDM for a hexagonal grid [Vin91]. For an

extensive list of papers offering approximation techniques and a shorter list of papers presenting exact construction methods. the reader is referred to Rosenfeld's massive bibliography on digital geometry [Ros98].

Most of the algorithms for computing the distance map use one of four approaches, depending primarily on how distance values or vectors are propagated through the grid. The following paragraphs describe each approach.

The first approach, one that can often take advantage of computer hardware, uses raster scanning [Dan80, Bor86, Mul92]. This technique is sometimes referred to as a *sequential local transformation*. In this method, the grid for the distance map is initialized with either scalars or vectors – zeros for the feature points and sufficiently large values or vectors for the background points. The grid is scanned two or more times, usually in alternating directions (horizontally or vertically for 2D grids). As each grid point is processed, a neighborhood of points around it is examined, and the results can be used to update the value or vector of the grid point or those of its neighbors in order to reflect shorter distances. For each scan, a different neighborhood mask is used according to the directions in which the scan traverses the grid. After a sufficient number of scans (typically two or four), the resulting grid is a distance map. This technique is generally applied only to rectangular grids, such as in the formulation in Figure 2.1(b). For a discussion of the method as applied to more complex domains, see Piper and Granum [PG87]. The main drawback of the raster scanning approach is its inefficiency of computation – much of the propagation that occurs in the grid is effectively wasted when the propagation from one scan overwrites values or vectors from previous scans.

The second approach uses ordered propagation. It can be applied equally well to either formulation for the distance map, and it involves propagating distance values or vectors simultaneously in an outward direction from each feature point (initialization of the grid is performed as in the raster scanning method). Two styles of ordered propagation exist. Piper and Granum [PG87], Ragnemalm [Rag92b], and Vincent [Vin91] provide algorithms that exhibit the first style. This consists of propagating values/vectors throughout the grid in a slightly independent fashion somewhat reminiscent of the particles in the simulation of a particle system. Each feature point spawns an initial generation of values/vectors in the direction of neighboring background points. If a value/vector can be used to improve the value/vector stored in the corresponding neighbor to which it is directed, then that neighbor point is updated and it spawns values/vectors during the next generation of propagation. If a value/vector cannot improve what is stored in the neighbor, then it effectively dies. Successive generations of propagation are computed until no more values/vectors exist to be propagated, signaling that no further improvements can be made to the distance map. Because a point potentially can be updated numerous times, there is an inherent inefficiency associated with this style of propagation. Piper and Granum [PG87] as well as Vincent [Vin91] note that the number of points that must be updated more than once is typically low in practice; however, this does not rule out the possibility of pathological cases. Neither group claims their algorithm to have linear time complexity. Ragnemalm does claim linear time complexity of his algorithm, though no formal proof is given.

The second style of the ordered propagation approach ensures that each point of the grid is processed only one time through the use of bucket sorting. Algorithms

by Verwer, Verbeek, and Dekker [VVD89] and by Ragnemalm [Rag92a] fall into this category. Each grid point, when its distance value is computed, is placed into a bucket with other points sharing that value. The buckets are processed in increasing order, and as a bucket is processed, the points it contains can propagate values to neighboring points. The propagation develops as a single contour from each feature point, and contours from adjacent or distant feature points can merge into a single contour as the propagation unfolds.

The third approach to distance map computation involves a two-step process. First, each row of the grid is scanned independently to computing the 1D distance map within that row. Next, each column of the grid is scanned independently in order to transform those 1D distance maps into a 2D distance map for the entire grid. The technique allows for the efficient use of memory during the computation, since only one row or column is being processed at any time. Examples of this method include the works of Paglieroni [Pag92] and Saito and Toriwaki [ST94].

For each of the three approaches already described, the given method can operate in linear time when computing non-Euclidean distance maps or approximations to the EDM. When an exact EDM is required, however, none of the above methods has yielded a linear time algorithm (a possible exception to this is Ragnemalm's algorithm [Rag92b], though nothing other than experimental verification of certain test cases is given to prove that the algorithm computes the exact EDM in all cases).

The fourth approach to computing the distance map is apparently the first provable method to operate in linear time complexity when computing the exact EDM. The approach is due to Breu et al. [BGKW95], and it achieves linear time by computing portions of the Voronoi diagram as it processes each row of the grid.

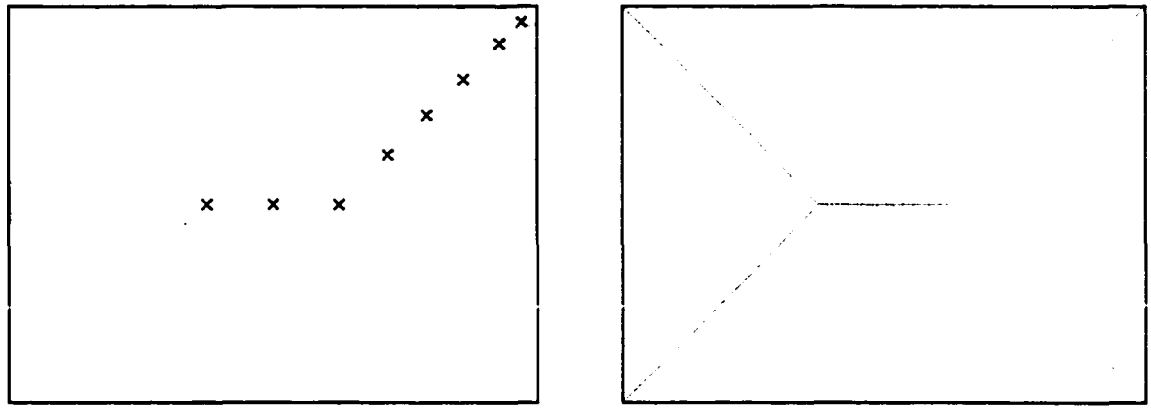
The task of distance map computation lends itself to parallel implementation. Yamada presents a massively parallel algorithm for computing the EDM that uses one processor for each background or feature point [Yam84]. He shows that parallel propagation of distance map values can resolve the approximation errors that plague sequential algorithms and that make correct computation of the EDM so difficult.

Just as the concept of distance extends to higher dimensions, the concept of the distance map is easily extended to three or more dimensions.[4] The first three approaches to computing the distance map also generalize, though the particulars of creating efficient algorithms are more intricate. It is unclear whether the fourth approach can be extended to three dimensions. Linear time computation of exact 3D Euclidean distance maps is apparently still unresolved.

2.2 Medial Axis/Medial Surface

The *medial axis* (also called the *symmetric axis*) can be thought of as a sort of branching geometric centerline of a 2D object. When the concept is applied to a 3D object, the centerline can become a centralized surface, so the term *medial surface* is used instead. More precisely, the medial axis (MA) of a 2D object is defined as the locus of the centers of all maximal disks interior to the object that touch the boundary of the object at two or more points. Figure 2.3 demonstrates this definition as applied to the MA of a rectangle. In like manner, the medial surface (MS) of a 3D object is defined as the locus of the centers of all maximal spheres interior to the object that touch the surface of the object at two or more points. Figure 2.4 shows

[4] Again, see [Ros98] for a listing of relevant papers.



(a) (b)

Figure 2.3: The medial axis of a rectangle. In (a), several maximal disks interior to the rectangle are shown, along with their centers (marked with "×"). The centers are points on the medial axis, which is shown in gray in (b).

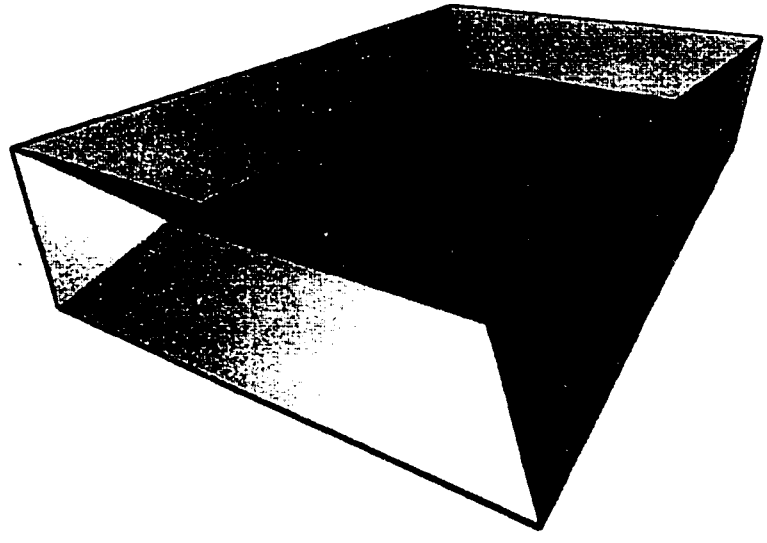


Figure 2.4: The medial surface of a box. It consists of 13 sheets: four triangles and eight trapezoids – each of which extends to a particular edge of the box – and one centrally located rectangle. The visible edges of the box are drawn in black.

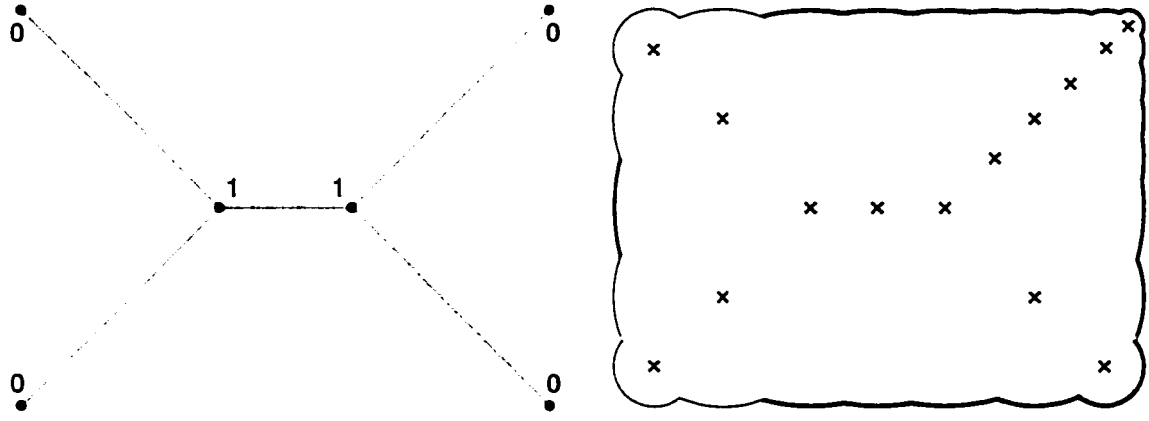
the MS of an elongated box. The concept of the medial axis and medial surface can also be extended to higher dimensions.

Interestingly enough in the context of this research, the medial axis or medial surface is often referred to in the literature as the *geometric skeleton*, or more simply, as the *skeleton*. In order to avoid confusion with the control skeleton, though, only the terms medial axis and medial surface will be used here.[5]

Data structures for storing the MA/MS are often equipped to hold additional information - in particular, the radii of the maximal disks or spheres. The motivation is simple: in conjuction with such radial information, the MA/MS can be used to reconstruct the original object. This operation is termed the *inverse transform*, the *inverse distance transform*, or the *inverse medial axis/surface transform*: an example is shown in Figure 2.5. The power of the inverse transform and the simplistic but representative structure of the MA/MS has prompted many to argue for the use of the MA/MS as an alternative shape representation (see the paper by Blanding et al. [BBGS99] for an example of a solid model editing system based on altering the MS of an object). Other arenas for application of the MA/MS include pattern recognition, robot navigation, and offset surface construction.

The MA/MS arises from the generalization of the *Voronoi diagram*. Whereas the Voronoi diagram is usually defined for a set of points in a domain, dividing the domain into regions according to the closest point of the set, the *generalized Voronoi diagram* can be defined for sets of points, line segments, curves, polygons, surfaces, shapes, or any combination thereof. The domain is thus divided into regions according to the closest point, line segment, curve, and so forth. The MA/MS is a subset of the

[5]The acronyms MA and MS will also be used, and when the discussion holds regardless of the dimensionality, the acronym MA/MS will be used.



(a) (b)

Figure 2.5: The inverse medial axis transform. In (a), the medial axis from Figure 2.3(b) is shown with radial information at endpoints and junction points (to be linearly interpolated). From this representation, the original rectangle can be reconstructed. In (b), several maximal disks have been reconstructed using the information in (a), and their images have been merged (the centers of the disks are marked with "×"). Reconstructing more disks will improve the approximation to the original rectangle: in the limit, the original rectangle will be obtained.

boundaries between the Voronoi regions. In the event that the defining components of the generalized Voronoi diagram form a boundary for some 2D or 3D object, then usually only the Voronoi regions interior to that boundary are considered. As an example, the MA in Figure 2.3(b) can be seen as dividing the interior of the rectangle into four regions according to which edge of the rectangle is closest to each interior point. Similarly, in Figure 2.4, the MS can be viewed as dividing the interior of the box into six regions corresponding to the six faces of the box: points in each region have the same closest face.[6]

[6]Note that in both examples, the objects are convex, so the medial axis/surface uses all boundary edges/surfaces between adjacent regions of the generalized Voronoi diagram. When concavities are involved, some of the Voronoi boundaries may not appear in the medial axis/surface.

The *Delaunay triangulation* (DT)[7] is the dual of the Voronoi diagram. For a Voronoi diagram and a DT defined on a common set of points, the structural components correspond in a one-to-one fashion such that, for instance, each edge of the 2D Voronoi diagram (or each face in the 3D Voronoi diagram) has an associated perpendicular edge in the DT.

The triangles of the 2D DT (and the tetrahedra of the 3D DT) have a special property: for any particular triangle of the 2D DT (or any particular tetrahedron of the 3D DT), the circumscribing disk (or sphere) does not contain any point of the set in its interior. This property is typically the basis for algorithms for the construction of the DT.

Because of the duality mentioned earlier, the construction of the DT is often used as a stepping stone in the construction of the Voronoi diagram. This is especially true for the 3D case. Also, methods used to construct the Voronoi diagram can sometimes be modified to produce the MA/MS. Thus, it is not surprising to see algorithms for the construction of the MA/MS based upon triangulation.

Kirkpatrick presents a medial axis algorithm based directly on the construction of the 2D generalized Voronoi diagram [Kir79]. Gold discusses a method for MA construction using both the Voronoi diagram and the DT [Gol99]. Some other 2D constructions use algebraic techniques instead of triangulation [Boo79, YR91]. For the 3D case, Goldak et al. describe a method for medial surface approximation based on constructing the DT of a discrete set of points scattered on the surface of an object [GYKD91]. Sheehy et al. present a similar but more thorough construction technique that uses a special type of DT known as the domain Delaunay triangulation [SAR95].

[7] In three dimensions, the DT is often referred to as the Delaunay tetrahedralization. For simplicity, the term Delaunay triangulation (or DT) will be used for either the 2D or 3D case.

Other, more algebraic methods for generating the medial surface are due to Dutta and Hoffmann [DH90] and to Sherbrooke et al. [SPB95] In a paper describing an efficient way to represent a 3D object as a union of spheres, Amenta and Kolluri [AK00] show that the set of sphere centers of their approximation converges to the MS of the object as the number of spheres used in the approximation increases.

2.2.1 Continuous versus Discrete Geometry

The previous discussion of the MA/MS and the construction techniques just mentioned are given in the context of *continuous geometry*. In continuous geometry, the object is defined using a continuous representation, usually as a polygon, polyhedron, or some closed curve or surface; and the MA/MS is defined using curves, surfaces, or continuous approximations to either (for example, polylines or polygonal meshes).

When the same ideas are applied in the realm of *discrete geometry* (also called *digital geometry*), there is more approximation involved. (For the purposes of this research, discrete geometry will refer to applications on a regular, rectilinear grid such as is formed by the integer points in a Cartesian coordinate system.) An object defined in a continuous space must be approximated as a set of discrete grid points (in two dimensions, the object is said to have been pixelized, whereas in three dimensions, the object is said to have been voxelized). The medial axis or medial surface of the object must be approximated as well; typically it takes the form of a subset of the pixels or voxels that comprise the discretized object. In this formulation, the MA or MS is sometimes referred to as the discrete medial axis (DMA) or the discrete medial surface (DMS), respectively.

Whereas the mathematical definition of the continuous MA or MS results in the existence of a unique MA or MS for any given 2D or 3D object (respectively), approximation methods vary for constructing the DMA or DMS. Since there is no precise mathematical definition, there can be very noticeable differences between the DMAs or DMSs of the same object as constructed by different algorithms. Furthermore, while the continuous MA/MS has the same topological structure as the continuous object, the topological structure of a DMA/DMS as constructed may be radically different from that of the discrete object. Nevertheless, certain desirable properties for DMAs or DMSs are suggested in the literature. The list that follows is based on properties mentioned by Ge and Fitzpatrick [GF96] and by Staunton [Sta96]:

- **Similar Topology:** The DMA/DMS should have the same basic connectivity, or topology, as the object.
- **Centering:** The DMA/DMS should be centered with respect to the boundary of the object.
- **Exact Reconstruction:** The set of points generated by the inverse distance transform – that is, by using the distance values at the points of the DMA (or DMS) and plotting discrete disks (or spheres) with corresponding radii (see Figure 2.5) – should be identical to the set of points of the original discretized object.
- **Rotational Invariance:** The general appearance of the DMA/DMS should be the same regardless of how the object might be rotated before being discretized.
- **Immunity to Noise:** Even in the presence of surface noise (defined as the presence or absence of individual pixels or voxels near the boundary of the

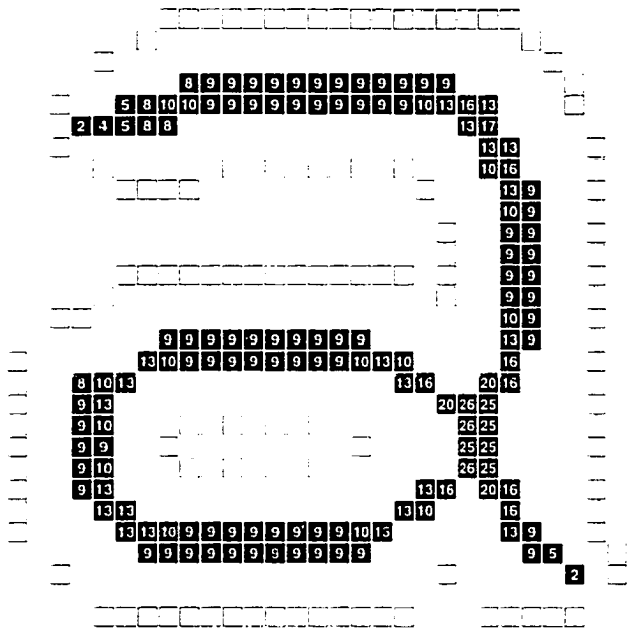


Figure 2.6: The discrete medial axis (DMA) of the object from Figure 2.1(a). The DMA cells are drawn in black with white distance map values. This DMA exhibits three desirable properties from the list on the previous page of the text: it has the same topology as the object, it is well-centered, and the presence of radial information (in this case, the distance map value for each DMA cell) allows for an exact reconstruction of the object via an inverse distance transform.

object), the DMA/DMS for an object should be very close in appearance to the DMA/DMS of the object without the surface noise.

Note that the last two items are actually desired properties of an algorithm for producing a DMA/DMS, rather than desired properties of a specific DMA/DMS; of course, the first three items would be desired in the output of such an algorithm. Figure 2.6 shows a specific example of a DMA that possesses the first three characteristics.

Note that there is another characteristic that is sometimes sought: **thinness**. Sometimes it is desirable to have a DMA/DMS that is as thin as possible, having

only the bare bones required to be in topological agreement with the original object and thus providing the most concise description of the object. The goal of thinness, however, almost always conflicts with the goal of exact reconstruction, and it can occasionally undermine the goals of centering and rotational invariance in subtle ways. In order to present a coherent list of desirable qualities, thinness has been left out of the list.

Closely related to the DMA and DMS and to the goal of thinness is the topic known as *thinning*. Thinning refers to the process of removing pixels or voxels from a discretized object in an attempt to whittle the object down in topological fashion to a more simple representation consisting of connected, unit-width pathways of pixels or voxels. In three dimensions, the simplified representation may also include unit-width surfaces of voxels. The pathways and surfaces of the simplified structure typically have a centralized location with respect to the corresponding part of the object. The main focus of thinning algorithms is the preservation of topology, with the primary purpose being to aid in the identification of basic structure. In application, thinning algorithms are frequently used in fields such as medical imaging in order to visualize networks of blood vessels or branching patterns of air passageways in the lungs.

Thinning algorithms often make use of information from a distance map of the object to be thinned. Such a distance map can aid in the gradual, even thinning of the object; however, the use of the distance map is not absolutely necessary. Depending on how well distance information is used, the result of the thinning process can be a fairly close approximation to a DMA or DMS of the object. Not surprisingly, thinning is a fairly common technique for computing the DMA or DMS. Lee, Kashyap, and Chu, for example, present a parallel algorithm for constructing the DMS of a discretized

object, employing several concepts from digital topology [LKC94]. Their algorithm uses specially constructed tables for preserving the Euler characteristic of a discrete 3D object as it is thinned. After finding the DMS, the algorithm can be reapplied to the DMS in order to find the DMA of the DMS itself. Lee et al. call this DMA the medial axis of the 3D object, though the use of the term medial axis (or discrete medial axis) to refer to the simplification of the medial surface (or discrete medial surface) of an object does not appear to be standard terminology.

For a presentation of thinning algorithms as applied to hexagonal grids, see Staunton [Sta96]. Along with their work in designing a solid model editing system around modification of the MS of an object, Blanding et al. [BBGS99] provide a comparison between Delaunay-based methods and thinning methods for medial surface construction, weighing such issues as ease of implementation, execution speed, and memory usage. Rosenfeld [Ros98] provides a list of over 160 papers dealing with thinning of 2D and 3D objects, and he also lists numerous papers specifically geared toward DMA or DMS construction; note, however, that the vast majority of these papers deal with the problem in two dimensions.

Another approach frequently used to construct the DMA or DMS is the direct extraction of the DMA or DMS from the (Euclidean) distance map of the object. The 2D distance map can be interpreted as a height field and viewed as a 3D landscape; the ridges of the landscape represent branches of the DMA. Thus, extracting the DMA (or DMS) from a 2D (or 3D) distance map amounts to finding and following the ridges implied within the map; the main difficulty comes in handling saddle points along the ridges. Figure 2.7 shows the distance map from Figure 2.1(a) viewed as a landscape.

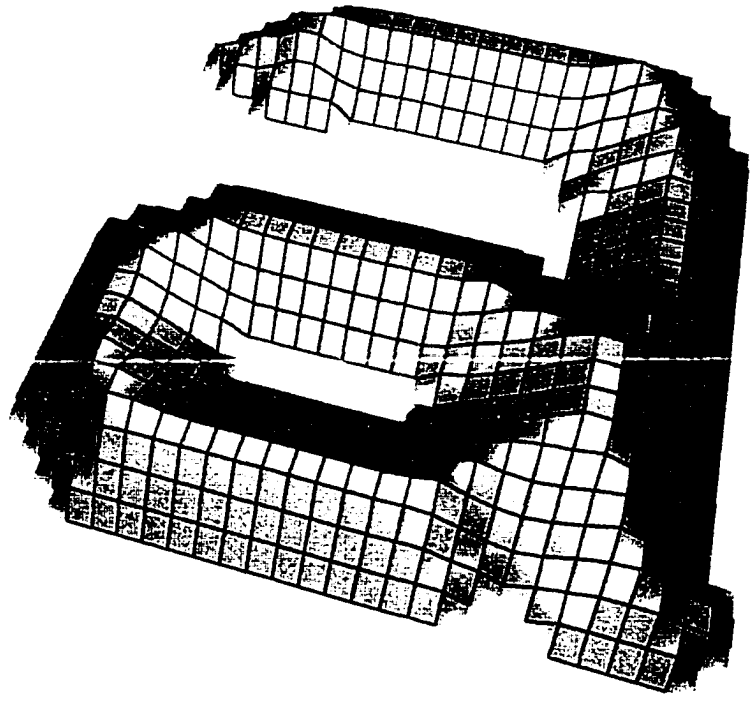


Figure 2.7: The distance map from Figure 2.1(a) viewed as a height field in three dimensions. The center point of each cell in Figure 2.1(a) has been raised to a height equal to its (unsquared) Euclidean distance: black lines connect the raised center points. Compare the ridges of the resulting landscape with the discrete medial axis shown in Figure 2.6.

In implementing the extraction approach, some sort of filtering is performed, either on a local or a global scale, in order to identify points that are the centers of maximal disks (in two dimensions) or spheres (in three dimensions). In effect, these maximal center points dot the ridges of the distance map. Additional, intermediate points are typically added in order to link the maximal center points, completing the ridges and forming a connected DMA/DMS for the object. In this context, Danielsson proposes an easy extension to a raster scanning implementation of a 2D distance map algorithm that utilizes a simple 3×3 filter for identifying points of the DMA [Dan80].

Ge and Fitzpatrick take the approach a step further [GF96]. After constructing the 2D Euclidean distance map, they take each locally maximal center as detected by Danielsson's algorithm and perform a gradient-based search to weed out center points according to whether the corresponding discrete disk is contained in any other discrete disk implied by the distance map. In the process, saddle points are identified. The remaining centers, which are the true centers of maximal disks (CMDs) as would be found by global filtering, are then connected by paths of points generated from steepest ascent searches from the CMDs and saddle points. Their results are very good, and their algorithm and the DMAs it generates exhibit the first four of the desirable properties mentioned on page 26. The extension of their algorithm to three dimensions, however, is problematic due to the difficulties in defining and identifying saddle points in the 3D Euclidean distance map.

Perhaps the most interesting work in the context of 2D DMAs is that of Ogniewicz and Kübler [OK95]. Their approach is a hybrid combining ideas from both continuous and digital geometry. From the boundary points of a discrete 2D object, a Voronoi diagram is constructed. The boundary edges of the Voronoi diagram are then intersected with the discrete object to form the DMA. Ogniewicz and Kübler define various measures of importance of DMA points, depending on the size of the point's corresponding disk and how important the point is for the connection of the DMA. Measurements are computed automatically and assigned to the points of the DMA; these same measurements are used to decompose the DMA into a layered structure – a hierarchical DMA. A range of threshold values can be applied to the hierarchical DMA in order to realize a specific DMA at a particular level of detail. A high threshold will result in a very simple DMA that relates to the basic overall structure of the

object; a low threshold will result in a more detailed DMA that corresponds better to all of the protrusions and intrusions of the object's boundary. In a further paper by Ogniewicz, an algorithm is demonstrated whereby a good, representative threshold can be calculated automatically, thus allowing for automatic pruning of the DMA [Ogn95]. In the paper, several examples are shown which illustrate the effectiveness of the technique in automatically generating a specific DMA at a level of detail appropriate to the object as a whole. It is certainly possible that a similar approach could be taken for 3D objects and that the same benefits would be seen; however, none of the importance measures defined in the first paper has a clear and obvious extension when applied to a 3D object.

2.3 Control Skeleton Generation

As noted at the beginning of Chapter 1, a control skeleton is a fundamental component of an articulated figure and includes a hierarchical structure of segments and bones together with information detailing how the surface geometry, or skin, of the figure is connected to that structure. This section will describe various commercial and non-commercial systems or methods that can aid a user in the steps involved in constructing a control skeleton for a given model.

2.3.1 Commercial Products

Several commercially available modeling and animation packages include support for working with articulated figures and their control skeletons. For the most part, control skeleton creation within these packages is a manual task performed via the user interface, though certain features are often provided as convenience routines to help speed up the process. A complete discussion of the control skeleton related

aspects of commercial software is beyond the scope of this dissertation: the following paragraphs merely serve to present the reader with a basic understanding of the skeletal apparatus in each of a few well-known packages and to highlight some of the special convenience tools they contain.

Maya®

Maya is a very popular modeling and animation package, and character animation is only a small arena of its possible applications.[8] Facilities in Maya allow a user to build a control skeleton for an object by modeling individual segments (termed "bones" within Maya) and joints as part of a connected hierarchy [Tea98, Tea99]. Each joint possesses three rotational degrees of freedom about a set of orthogonal axes, though joint parameters can be set to limit movement about any of the three axes. A segment acts as a spacer between the two joints it connects and indicates which of the two joints is the parent (that is, which joint is the closer of the two to the root joint). The interface also provides numerous means for creating skeletons, such as for creating chains of joints and segments by specifying a set of points, for inserting or removing joints within a skeleton, for splitting a skeleton into two by disconnecting it at a particular joint, for merging two skeletons into one, and for changing the direction of the hierarchy by specifying a different joint to be the new root joint. A feature called "mirroring" allows a user to duplicate a portion of the skeleton, possibly to make a reflected copy – this is especially useful for creating symmetric skeletons, for example, allowing a user to build a skeletal subhierarchy for the left arm of a character which is then automatically mirrored and duplicated to

[8]Maya is a registered trademark of Silicon Graphics, Inc., and exclusively used by Alias|Wavefront, a division of Silicon Graphics Limited.

create a skeletal subhierarchy for the right arm. Other features allow for the automatic setting of certain joint parameters, such as having the program guess joint limits or orient joint axes either to align with the world axes or instead to align relative to each joint's first child joint.

Attaching a figure's surface geometry to the skeleton is known as "skinning" in Maya. A user may invoke Maya's myriad selection tools to choose the points defining a portion of the object and then bind those points to specific bones/joints of the skeleton. Alternatively, each point can be bound automatically to the closest bone/joint, though a user may have to re-bind points that are grouped incorrectly. Maya offers two basic methods of skin binding: rigid skinning and smooth skinning. Under rigid skinning, each point is bound to only one bone/joint; under smooth skinning, each point may be bound to multiple bones/joints, with the influence of each bone/joint determined by sets of weights. Whereas the weighted influencing of bones/joints in smooth skinning allows for automatic flexing and deforming of the skin around joints, such effects are not possible under rigid skinning without the use of additional tools such as "flexors" or "deformers". Flexors, which are used only with rigid skinning, are free-form deformation (FFD) tools that allow for smoothing, rounding, and creasing of the skin surface around a bending joint. Deformers, of which there are several varieties, can be used with either skinning method and provide more options for skin deformation. Wrinkle-formation and muscle bulging effects, for example, are often performed through the use of deformers. Posing or animating the skeleton can be accomplished through the use of forward or inverse kinematics toolkits. For more information regarding Maya, see the references [Tea98, Tea99, HKGL00, VSC00].

3D Studio MAX®

Another popular modeling and animation package is 3D Studio MAX.[9] In addition to having several other graphics tools, this package presents a reasonable interface for creating control skeletons and attaching surface points of the object to the control skeleton. Many people who use 3D Studio MAX for character animation, however, choose to use a special plug-in for the package called Character Studio®.

Character Studio provides a more advanced interface for creating control skeletons [Dis00]. It consists primarily of two components: Biped® and Physique®. Biped is specially geared towards modeling and animating two-legged characters. It contains an interface to allow quick creation of a skeleton structure consisting of segments and joints. The interface acts as a template of sorts for producing bipedal skeletons, containing boxes that a user can check to generate various additional parts of the structure (such as a skeletal chain for a tail) or to specify how many joints and segments should be used in a particular limb. Biped will automatically produce a generically-posed skeleton structure conforming to the user's requests. The structure has the additional advantages that values for various segment and joint parameters have been defined in meaningful ways for a humanoid skeleton, and miscellaneous additional aids such as inverse kinematic chains have been produced to help in the animation of the skeleton. Not everything is done automatically, however. The user must still reposition the generically-posed skeleton so that the joints and segments align with the figure the user is trying to animate. Also, the user must use other parts

[9]3D Studio MAX is a registered trademark of AutoDesk, Inc. Character Studio is made by Discreet, a division of AutoDesk, Inc. Character Studio, Biped, and Physique are all registered trademarks of the company.

of the interface to anchor the geometry of the figure to the skeleton structure. Nevertheless, Biped does streamline the process of generating articulated, bipedal figures. In addition, Biped provides a sophisticated system for animating the locomotion of the figure by allowing a user to specify footprints for the movement and also for handling dynamically realistic motion of the figure, not to mention facilities for importing motion capture data. Another feature allows animation sequences designed for one figure created in Biped to be mapped to another figure created in Biped, effectively separating the animation from the figure and making it reusable.

Physique is the other main part of Character Studio. It contains the tools that allow a user to attach the surface geometry to the skeletal structure. In addition, it provides means for producing muscle bulging effects, including the apparent action of tendons, based on the bending of the skeleton. A user may even define the profile at various points along a muscle for more control of the details. Tools for other skin altering effects such as creasing and vein deformations are also present in Physique. More information on 3D Studio MAX and Character Studio is available at the company's web site [Dis00] or in books such as [JBD+00].

Poser

A product specifically designed for the purposes of modeling, posing, and rendering articulated characters is Poser.[10] Poser provides libraries of predesigned characters, complete with surface geometry, shading information, and control skeletons [FS99]. It also contains libraries for various props, shading models, and lighting models that can be used during scene design and construction. If a user is satisfied with one of

[10]MetaCreations Poser™ had been a trademark of MetaCreations Corporation, the former owner of Poser. In April 2000, Poser was purchased from MetaCreations by a company named egi.sys and is now a product of Curious Labs, an egi.sys company [Egi00, Cur00].

the predesigned characters, he or she may proceed straight to the task of posing the character or creating animation sequences for the character. Poser even possesses a library of some common animation sequences (walk cycles, for instance) that a user can apply to a character. Custom design of a character is more cumbersome. A user can either modify one of the predesigned characters or can import some geometry for a figure from an external source. If the geometry is imported, then the user must work systematically with Poser's Hierarchy Editor in order to set up a control skeleton for the figure, specifying which parts of the geometry are parents of which other parts, specifying how the geometry should be deformed when the skeleton moves, and setting various parameters to ensure that the joints are placed and oriented appropriately. If various parts of the geometry are named according to a standard used by Poser, then part of the process can be performed automatically during importation; nonetheless, much is still required on the part of the user. In fact, if a user wants to take advantage of certain animation tools provided by Poser, then the geometry must adhere to Poser's standard naming convention. Poser can propel a possibly novice user into the world of articulated character manipulation, but it seems best suited for users who are willing to work with libraries of predesigned characters. For more information on Poser, see [Mor00].

2.3.2 Control Skeleton Research

As exemplified during the discussion of commercially available software, several steps have been taken to automate a few of the more mechanical tasks involved in creating a control skeleton for a given object. Still, some research has been done which has yet to be incorporated into commercial packages.

Perhaps the earliest work on automatic skeleton generation is that of Tsao and Fu [TF84]. Their method operates entirely in the domain of discrete geometry. It begins with a 2D or 3D bitmap representing the object, upon which they perform a distance transformation that yields an approximation to the Euclidean distance map. The distance map is then processed using a local filtering method to identify individual points of the discrete medial axis or surface. These scattered DMA or DMS points are preserved in a subsequent thinning operation which results in a connected DMA/DMS. The DMA/DMS is then converted into a graph whose vertices are the DMA/DMS points and whose edges indicate adjacent pairs of those DMA/DMS points. Once formed, the graph can be randomly manipulated through vertex modification, insertion, and deletion. During modification, the vertices of the graph may move to other grid points so long as the adjacency relationships are maintained. The vertex modification routines thus permit the bending and repositioning of the DMA/DMS within the confines of a Cartesian grid. Since distance map values are stored for each vertex of the graph, an inverse distance transform can then be used to construct a new bitmapped representation of the object that corresponds to the modified DMA/DMS.

In effect, the graph in Tsao and Fu's program functions as the control skeleton for the original bitmapped object, with the graph vertices being the joints and the graph edges being the segments (though Tsao and Fu themselves never use the terms *joint* or *segment*). In fact, it is not clear that Tsao and Fu ever think of their modeling technique in the context of creating an animatable articulated figure: nowhere do they mention animating the graph or the object. Apparently, they only intend for their system to be used to create random but similar objects based on simple repositioning

of the skeleton graph. Note also that their graph has two basic differences from the control skeleton as presented in Chapter 1. First, the control skeleton segments (the graph edges) are not rigid. Their lengths are the distances between neighboring voxels, and these lengths can change, such as when two diagonally adjacent vertices are repositioned to be orthogonally adjacent. This means that the skeleton joints (the graph vertices) are not necessarily fixed in the local coordinate space of the skeleton segments. Second, the boundary of the object is not preserved. Instead of being attached to the control skeleton in some fashion, it is completely re-created during the inverse distance transformation. This fact, combined with the simplicity of the graph modifications, causes many of the resulting reconstructions of their example objects to have a somewhat blobby appearance. Hard edges and sharp convex or concave corners of the original boundary are almost always rounded over in the randomly posed instances. Perhaps this is why Tsao and Fu comment that their method might work best for stochastic modeling of natural objects such as clouds and trees.

Tsao and Fu's research dates back to 1984. Recently there have been other efforts to provide tools that automate parts of the control skeleton creation process.

A method somewhat similar to that of Tsao and Fu is one by Gagvani, Kenchammana-Hosekote, and Silver [GKHS98]. It also operates entirely in the discrete domain and contains steps to compute a distance map and a discrete medial surface for an object. The distance map is constructed using a quasi-Euclidean 3-4-5 distance metric (this is a specific type of chamfer metric which is named according to the initial distance values assigned to boundary voxels: use of this metric results in a distance map that has properties similar to those of the Euclidean distance map, hence the term "quasi-Euclidean"). After the distance map is constructed, a local filter is applied to

each of its voxels to identify DMS points. A "thinness" parameter may be supplied by a user to influence the thickness and connectedness of the DMS – a lower value results in better connectivity but a thicker DMS; a higher value results in a thinner DMS with poorer connectivity. The DMS is converted into a fully-connected graph with one vertex for each DMS point. Each edge is assigned a weight according to its length and to the difference between the distance map values of the two DMS points it connects. Based upon these edge weights, a minimum spanning tree is constructed. Here again, a user may specify a "connectivity" parameter that indirectly influences the resulting spanning tree by changing the calculation of the edge weights. The authors of the paper suggest that the spanning tree can be converted into a control skeleton by marking certain vertices as joints and using those joints to divide the tree into sections; each section of unmarked vertices and edges would form a rigid segment of the control skeleton. No details are provided as to how joint vertices might be marked as such, and only very simple examples are provided showing any articulation of the spanning tree control skeleton, with each example demonstrating only a single joint. As with Tsao and Fu's method, the inverse distance transform is used to generate new voxelized instances of the object for various poses of the control skeleton.

Gagvani and Silver have also implemented their method as a plug-in for Maya [GS99]. This plug-in can be used to convert the spanning tree into a control skeleton in Maya's internal format; however, simple, straightforward conversion with each DMS point being used to form a joint usually results in a control skeleton that is entirely too complex. Instead, Gagvani and Silver suggest that a user merely view the discrete medial surface while manually constructing a control skeleton whose

segments run along stretches of medial surface voxels. They show how the animation of the control skeleton within Maya can be exported and used to drive the animation of the voxelized figure.

Teichmann and Teller have presented a system for assisting in the generation of control skeletons [TT98]. Given a closed polyhedral model, their algorithm first computes a Voronoi diagram for sample points on the surface of the polyhedron. This set of sample points must be sufficiently dense in order to ensure that the Voronoi vertices interior to the polyhedron lie approximately on its medial surface. The user then selects Voronoi vertices that should be endpoints of branches of the control skeleton, and the Voronoi graph (that is, the graph made from the vertices and edges of the Voronoi diagram) is simplified in order to produce a spanning tree whose leaf nodes are those Voronoi vertices the user has selected. The Voronoi graph is not necessarily connected, and only the largest connected component of the Voronoi graph is simplified; any other components are ignored. Next, the user specifies nodes of the spanning tree as points of articulation; these nodes become the joints of the control skeleton. The user is also provided with tools that allow manual reorientation of the coordinate frame for each joint. Segments of the control skeleton are constructed by simplifying portions of the spanning tree lying between joints and/or endpoints; thus, each segment corresponds to a chain of spanning tree vertices.

A sophisticated network of springs is created for the purpose of attaching the polyhedral model to the control skeleton structure. To achieve this, a 3D Delaunay triangulation is performed on the combined set of polyhedron vertices and spanning tree vertices. Edges of the tetrahedra formed are examined, and any edge connecting a polyhedron vertex to a spanning tree vertex is converted into a spring. If any

polyhedron vertex is not adjacent to any such edge, then a spring is created to attach it to the closest spanning tree vertex. Each edge of the polyhedron is also converted into a spring. Thus, the polyhedral edges form a network of springs that is connected to the control skeleton using additional springs. When the control skeleton is posed, the vertices of the spring network are first repositioned along with their corresponding control segments, a simulation is then performed to allow the network of springs to reach a stable configuration. At that point, the vertex positions can be used to redraw the surface polygons. Teichmann and Teller include a table of results in their paper indicating that the time required to create a control skeleton using their system ranges from about 13 minutes to 6 hours, depending on the complexity of the polyhedral model. They also mention that models consisting of large numbers of polygons should probably be simplified beforehand. The original model may be used for animation once the control skeleton has been generated, but Teichmann and Teller do not state how the spring network, as constructed for the simplified model, should be extended to work with the original, complex model.

In a method proposed by Bloomenthal and Lim, a control skeleton for an object is automatically produced from the medial surface of an object [BL99]. First, the medial surface itself is automatically produced using an implicit method Bloomenthal and Lim have developed based upon examining how the direction to the nearest surface point changes as a point of examination is moved within the object – large or obvious changes in the direction signal the presence of the medial surface. For each point in a grid of sample points, the direction to the nearest surface point is computed. Adaptive subdivision is employed along grid edges whose endpoints have substantially different directions (as determined using a threshold parameter).

The medial surface is constructed as a polygonal mesh of the points resulting from the subdivision processes. Additional information is stored with the mesh as to the distance from the mesh points to the surface of the object. The control skeleton structure can be derived automatically from the medial surface mesh, though no details are provided as to how this is accomplished. The points of the medial surface mesh are then anchored to the control skeleton structure. Using the control skeleton to reform the surface of the object involves a two-step process: first, the control skeleton is used to modify the position of the medial surface mesh; then, an inverse distance transformation is applied to the mesh in order to reconstruct the surface. The reconstructed surface is thus defined implicitly but approximated using a polygonal mesh. Note that here, as with Tsao and Fu's method (see page 38), the original surface is not preserved; rather, the surface is completely reconstructed for each new pose – such is typical of the use of the inverse distance transform, whether in the discrete case or the continuous case.

Few details are provided by Bloomenthal and Lim with respect to the construction of the control skeleton, the quality of the results, or the time required to execute the algorithm. Their method appears to restrict the input object to be a single, closed surface. Also, implicit methods typically require more computation time than non-implicit methods. Bloomenthal and Lim apparently plan to release a commercial version of their algorithm in a product called *Actionizer*.

Stalpers and van Overveld also use an underlying polygonal mesh in connection with the control skeleton [SvO97]. Their method focuses on the problem of attaching the surface of the object to the control skeleton. It starts with two pieces of input: a closed polygonal surface model for the object, and a polygonal mesh representing the

structure of the control skeleton - in reality, the polygonal mesh is itself anchored to an articulated structure consisting of a hierarchy of hinge joints. A dual-connectivity search is performed, examining vertices of the surface mesh in conjuction with those of the skeleton mesh and determining a mapping of the one set to the other. The mapping is constructed according to the vertex-polygon adjacencies of the surface mesh and the skeleton mesh. Two surface vertices adjacent to the same surface polygon are mapped either to a single skeleton mesh vertex or to two skeleton mesh vertices that themselves are adjacent to a shared skeleton mesh polygon. In order for there to be an effective mapping, Stalpers and van Overveld mention that the skeleton mesh should resemble the surface mesh in its general structure. Once the mapping is completed, it is used to construct a weighted anchoring of surface mesh vertices to skeleton mesh vertices. In order to prevent undesired surface creases from forming during deformation of the skeleton mesh, additional hinge normal vectors can be computed and integrated into the weight averages. Unlike Bloomenthal and Lim's approach, Stalpers and van Overveld's method preserves the surface mesh of the object but deforms it based on the positioning of the skeleton mesh, which itself is deformed and posed via the specification of the hinge joint angles for the control skeleton. The authors of the paper recommend that the hinge joint axes of the control skeleton should be aligned with the shared edges of the skeleton mesh, noting that otherwise, non-planar skeleton mesh polygons can adversely affect the shape of the deformed surface. They also recommend the alternative use of free-form deformation (FFD) methods for relatively spherical objects, where a polygonal skeleton mesh may not adequately correspond to the basic structure of the surface mesh, or the use

of other skeleton-skin attachment schemes in the event that hinge joints alone are insufficient for the motion of the control skeleton.

A concept potentially useful for automated control skeleton generation is that of level set diagrams as applied to polyhedral objects, such as in the work of Lazarus and Verroust [LV99]. In their work, the construction of the level set diagram (LSD) begins by selecting a source vertex of the object and, for each other vertex, computing the shortest distance along surface edges to the source vertex. The field of distance values at surface vertices becomes the domain for the generation of isocontours of particular distance values. Through analysis of the isocontours on either side of vertices with local maximum values (at which point the topology of the isocontour set may change), a tree-shaped structure can be generated whose root is the source vertex and whose branching points and leaves are local maximum vertices. This structure approximates the branching shape of the object's interior by viewing only the boundary of the object. As such, it is a fairly rough approximation that may or may not correspond well with the medial surface (although contour centers can be used to centralize the limbs of the tree, the branching points of the tree lie on the surface of the object and thus not on the medial surface). Nevertheless, the construction does offer some possibilities for control skeleton generation: it provides a one-dimensional branching structure with possible articulation points (that is, the branching points); and the surface polygons, by virtue of the distance values at their vertices and their use in the generation of isocontours, can be readily divided into sets corresponding to the limbs of the tree, leading to a fairly straightforward anchoring of surface to skeleton.

2.4 Anatomically Based Modeling and Animation

Anatomically based modeling and animation can be described as modeling and animation whose goal is a close and apparent similarity to anatomical shape and movement of that shape, especially with regard to outward appearance of the skin, and possibly with regard to the simulation of underlying anatomical forms. Work in anatomically based modeling and animation is driven by people's desire for increased levels of realism in computer graphics. This realism with respect to the way characters should look or behave may be motivated by such goals as the demand for more immersive virtual worlds similar to our own, the quest for seamless integration of digitally created characters into real-life photography and cinema, or the need for better modeling and simulation for purposes of medical research such as in the areas of biomechanics and ergonomics or visualization of surgical planning. Whatever the motivation, anatomically based modeling and animation appears in various forms, and the topic has been a growing focus of graphics research for some time.

Because human and animal anatomy are subject to the laws of physics, physically based modeling as applied to character animation can be viewed as an extension of anatomically based modeling and animation within a simulated physical world. Physically based modeling contributes in two basic manners to anatomical modeling and animation. The first is the application of simulated dynamics to articulated figures, as exemplified by the works of Armstrong and Green, Wilhelms, Forsey and Wilhelms, and the Gascuels [AG85, Wil87, FW88, GG94]. The second is the role of simulated dynamics in the modeling and animation of anatomical components, such as with spring-mass systems used to attach the surface to the skeleton for the purpose

of modeling fatty tissue or the integumentary system. Some examples of this role will be noted in the works described in the remainder of this section.

Early research in articulated figure modeling involves two basic layers: skeleton and skin. The skeleton is the articulated structure of segments and joints and is the layer controlled by the animator. The skin represents the object to be animated. It is typically a geometric surface wrapped around the articulated skeleton, bending with the skeleton as the structure is moving. Occasionally a third layer is mentioned, consisting of a behavioral model used to drive the animation of the skeleton. This third layer, which might also involve dynamic simulation or inverse kinematics for positioning the figure, has the effect of further distancing the animator from the shaping of the skin as the figure moves.

Chadwick, Haumann, and Parent were apparently the first to incorporate an additional anatomical layer into articulated figure creation and animation [CHP89]. Pioneering a more advanced notion of layered construction, they insert a muscle and fatty tissue layer between the layers of skeleton and skin. This muscle and fatty tissue layer allows for such interesting deformations as muscle bulging and secondary motion effects on the skin, like the swinging of fatty deposits. The foundation for this middle layer is provided by FFD lattices in which the points of the skin are embedded. For the muscle model, the control points of the FFD lattice are repositioned based upon the angles of corresponding joints so that the skin appears to be affected by an underlying muscle which can be flexed or extended. For the fatty tissue, the control points of the FFD lattice become mass points of a spring-mass system, some of which are rigidly attached to segments of the skeleton. The motion of the spring-mass system

is dynamically simulated based on the kinematic movement of the skeleton and the dynamic movement of the mobile mass points from frame to frame.

Chen and Zeltzer have created a finite element model of a muscle [CZ92]. In their implementation, a muscle is modeled as a polyhedral mesh comprised of multi-node finite elements together with spring-like generators imparting both active and passive forces on the model. Although they do not attend to the problem of skin attachment to the muscle, they do provide a model that is both biomechanically accurate and well suited for realistic graphical display.

Laser scanning of humans, which typically has the purpose of creating more realistic skin geometry for human data models, also falls in the realm of anatomically based modeling. Related research that takes this scanning concept even further is due to Kakadiaris and Metaxas [KM95]. In their work, which employs image processing techniques using multiple camera views, the goal is to construct an articulated data model for the human subject. The human is taken through a scripted set of poses that allow for automatic identification of body parts from the appearance and disappearance of limb silhouettes in the images. In addition, automatic segmentation of flexed limb silhouettes is performed and checked for frame-to-frame coherence in order to identify joint locations for the figure. In this way, a segmented model of a human subject is constructed. The surface constructed for each limb section appears to be rigidly attached to the corresponding segment of the control skeleton: special skin deformation around flexed joints is not performed, though that appears not to be a goal of their research.

Much research in modeling and animating virtual humans has been done under the direction of Norman Badler in the Center for Human Modeling and Simulation at the

University of Pennsylvania [BPW93]. The hub of this research is a software package known as Jack.[11] Jack provides an interface for human modeling and animation suited for such diverse purposes as ergonomics research, applications involving inverse kinematics, and goal-directed behavioral simulation of virtual humans. As input for dynamics simulations on human models, Jack's Spreadsheet Anthropomorphic Scaling System permits the specification of anthropometric parameters such as segment mass, segment dimensions, joint type, and joint limits. Animation of a human model in Jack involves scripting or simulating movement of its control skeleton [ABH+94]. The surface of the model is deformed based on FFD meshes whose control points are anchored to the skeleton, though it appears the FFD meshes are used primarily for the continuity of skin across multiple skeleton segments and not for the simulated appearance of muscle effects.

Various work in the modeling and animation of anatomical components has been spearheaded by Jane Wilhelms at the University of California, Santa Cruz. Starting with a tree-structured skeleton, Wilhelms shows how various anatomical layers can be constructed for modeling animals [Wil94, Wil97]. Bones and muscles are modeled as combinations of ellipsoids. "Stuffing" - meant to represent soft tissue and useful for adding features such as the nose and ears - is also modeled using ellipsoids. Bones (which for the most part are elongated ellipsoids with spherical knobs at each end) are rigidly anchored to corresponding segments of the skeleton. The same is true of the ellipsoids used for stuffing. Each muscle is a scalable combination of three linearly arranged ellipsoids, with the outer two representing tendons. The linear arrangement

[11] Jack is a registered trademark. Originally developed at the University of Pennsylvania, Jack was acquired in 1996 by Transom Technologies, Inc., which was itself acquired by Engineering Animation, Inc. (EAI) in 1998 and is currently part of the Digital Human Group of EAI. The software is now available commercially under the trade name Transom Jack [Tra00, Eng00].

spans from a point of origin on one segment to a point of insertion on a distal segment. After the control skeleton is has been posed, each muscle model is rescaled to fit between its transformed points of origin and insertion. Since the transformation is designed to preserve the volume occupied by the model, the model will appear to bulge or stretch appropriately. For these intermediate layers, a user has the option of starting with a default configuration before making modifications or specifying additional bones, muscles, and stuffing. A polygonal mesh skin is automatically generated to cover the anatomical components. First, the entire set of ellipsoidal component models is voxelized; then, repeated filtering is applied to blur the voxelization; finally, a marching cubes algorithm is employed to generate a polygonal mesh for an isosurface of the voxel grid. This resulting mesh contains the set of ellipsoids while also allowing a small gap between itself and the components. Each point of the skin mesh is anchored to the closest underlying ellipsoid. When animated, vertices of the skin mesh are initially situated after their corresponding ellipsoids are positioned; then the vertices of the mesh are repositioned during a spring based simulation that allows the mesh to approach an equilibrium. A user may change the anchoring of the skin for different portions of the figure, producing a larger or smaller gap between anatomy and skin; in addition, a user may change spring constants for the skin mesh to change the apparent flexibility of the skin model.

In work with Van Gelder, Wilhelms has improved the muscle model [WG97]. Instead of three linearly arranged ellipsoids, a muscle and its tendons are modeled as a single, generalized, deformable cylindrical mesh with an elliptical cross section. With two origin points and two insertion points, the mesh allows for a wider spectrum of muscles, including broad muscles as in the chest or the back; furthermore, a pivot

point allows the mesh to be bent, such as when a tendon of one of the quadriceps bends over the front of the knee. During animation, the length of the mesh is computed based upon the kinematic positioning of the two bones to which it is attached, and this length is used to scale the thickness and width of the mesh for approximate volume preservation. In an interesting twist to the skin surface creation, parts of an animal model such as the hands and feet can be temporarily scaled up before skin generation in order to generate larger number of polygons in highly flexible regions. As an example of the complexity of modeling an entire animal this way, a monkey is modeled from a hierarchy of 85 skeletal segments, with layers of 156 bones, 52 muscles, and 52 generalized tissue components. The skin generated for the monkey model has about 75,000 vertices and 150,000 triangles; nevertheless, the system is capable of interactive speeds when a user is working with the model.

Other work at the University of California, Santa Cruz, has been geared toward hybrid modeling. Instead of the automatic generation of a skin mesh for an anatomically modeled animal, the work of Schneider and Wilhelms involves starting with a skin mesh and constructing the underlying anatomical components to fill in the volume of the figure [SW98]. This approach provides the benefit of working with existing polygon mesh models (which typically have fewer polygons than the automatically generated skin meshes); however, the authors note that manual placement of underlying component models is a tedious process, even though they can start with an existing set of component models for a similar animal figure and make modifications. Lapierre and Wilhelms have taken several steps to speed up the process, adding various features to the interface for helping to match an underlying animal anatomy model with a predefined polygonal skin mesh [LW99, Lap99]. Lapierre has

demonstrated the effective use of super segments, which are groupings of the anatomy hierarchy into connected chains (such as a hind leg) that can be conveniently rescaled or repositioned using such techniques as the application of inverse kinematics to the super segment applied after the user has moved the end-effector for the chain.

Turner and Gobbetti have developed an interactive system for constructing and animating layered deformable characters [TG98]. A user may create a character within the system by building successive layers representing the control skeleton, bones and muscles, and fatty tissue and skin. The system provides a virtual environment with interactive tools, and users don head-mounted stereo displays and operate 3D input devices while creating layered figures and producing keyframed animation of those figures.

With the goal of providing more visual realism, Scheepers et al. have modeled musculature in more detail [Sch96, SPCM97]. From thorough analysis of muscles from an artistic perspective, Scheepers has developed several muscle models according to the different kinds of muscles present in the human body. The set of available models includes fusiform muscles, which have a simple, ellipsoidal shape and are attached using one or two tendon models; multi-belly muscles, which approximate wide muscles through convenient spacing of a lateral sequence of simple ellipsoidal muscles; and general muscles, which can bend or twist around other anatomical structures. Each muscle model is designed with animation in mind and allows for approximate volume preservation during animation. The muscle models are implemented as classes in a procedural modeling language known as AL.[12] Although the models are generalized enough to be used for a human figure or any animal with similarities in musculature,

[12]AL, short for Animation Language, was developed by Steve May at ACCAD [May95].

Scheepers has chosen the upper. right limb of the human as a testbed for the models. He has also shown how implicit forms of the component models can be used to help offset the skin surface over a portion of the testbed so that the skin reacts appropriately to deformations of the underlying bone and tissue during animation. The results of both the musculature modeling and the skin modeling are impressive and demonstrate a remarkable level of realism.

CHAPTER 3

APPROXIMATING THE EUCLIDEAN DISTANCE MAP

This chapter and the one that follows describe the discrete geometry algorithms developed for use in this research, specifically, algorithms for computing the distance map and the discrete medial axis/surface for a discretized object. The concepts of the distance map, the medial axis (MA), and the medial surface (MS) were introduced in Sections 2.1 and 2.2 in the previous chapter.

This chapter focuses on the algorithm designed for computing an approximation to the Euclidean distance map (EDM). Section 3.1 presents a general overview of the algorithm. Sections 3.2 and 3.3 then detail the data structures and various steps involved in propagating distance values through the map, which is basically how the algorithm works. The algorithm is then analyzed in various ways in Section 3.4.

Chapter 4 presents a similar discussion with regard to the algorithm for constructing the discrete medial axis (DMA) or discrete medial surface (DMS) of a discretized object. In both chapters, numerous figures and tables are provided to help explain the data structures and to demonstrate execution of the algorithms in step-by-step fashion. Although the descriptions for both algorithms are illustrated using the 2D case, implementation for the 3D case is very similar (some particulars will be mentioned), and indeed, the algorithms are easily extended to higher dimensions.

Exactly how these algorithms are used in the rest of the research will be discussed in Chapter 5. For now, simply note that for the purposes of this research, it is not necessary to compute the exact EDM, nor is it necessary to compute a precise DMA/DMS. The algorithms presented here and in Chapter 4 are designed to provide close approximations and are optimized for efficiency.

3.1 Overview of the Algorithm

As alluded to in the introduction above, the algorithm works by propagating distance information through the grid. The propagation is performed in a layered fashion moving outward, away from the feature points of the grid. During processing, each grid point receives distance information from its processed neighbors; later, it can pass distance information along to its unprocessed neighbors as they are processed. Because distance values are assigned to grid points in increasing order of distance value, the grid points are essentially processed as a series of broken contours corresponding to each assigned value. A clearer understanding of these contour layers can be obtained by jumping ahead temporarily to examine Figures 3.2 through 3.5 on pages 65 through 68.

As with any distance map algorithm, the input to the one described here is a grid where each cell is clearly marked as being either a feature point or a background point. The output of the algorithm is a labeling of each background point with a distance map value corresponding to the square of the Euclidean distance from its center to that of the closest feature point. The vast majority of the values assigned will be equal to values for an exact EDM; in a few cases, however, the value computed is slightly greater than the square of the shortest Euclidean distance to a feature point.

though typically the error is negligible. Reasons for the errors are mentioned in the analysis and discussion which follows the presentation of the algorithm.

Because this research deals with voxelized objects and because it involves computing distance map values for interior voxels, instead of being presented in terms of feature points and background points, the algorithm will be presented in terms of exterior voxels and interior voxels, respectively. Furthermore, for the sake of consistency, the term *voxel* will also be used when the discussion involves 2D grids, even though the term *pixel* is perhaps more appropriate. Note too that although the algorithm is described for the set-up where the feature points surround the background points (as in Figure 2.1(a) on page 14), the algorithm can be applied - without modification - in order to compute the distance map for the alternate set-up where the feature points are amidst a field of background points (as in Figure 2.1(b)).

3.2 The Reference Table

The initial step of the process is the construction of a look-up table for use in propagating distance values through layers of interior voxels. Actually, instead of propagating distance values directly, the algorithm works by propagating reference numbers, each of which has an associated distance value. These reference numbers and their corresponding distance values are available in a look-up table known as the *reference table*. To aid in the creation of the reference table, an array called the *reference grid* is constructed which will contain all possible reference numbers and distance map values that might appear in a particular EDM.

The reference grid is an array with each cell containing two items: a distance map value and an associated reference number. Figure 3.1 on page 58 shows the first eight

rows of the reference grid for the 2D implementation. The distance map value for a cell $(\Delta x, \Delta y)$ is calculated using the expression $(\Delta x)^2+(\Delta y)^2$, where Δx and Δy represent the relative distances along each of two orthogonal directions from a particular interior voxel to a particular exterior voxel. Note that when direction is important, as in the construction of a vector distance map such as shown in Figure 2.2(a) on page 15, there are four cases to consider, as given by $(\pm\Delta x, \pm\Delta y)$. For the purposes of this research, however, where the magnitude of the distance is the main item of interest, it suffices to ignore direction to some degree. The symmetric nature of the squared distance calculation collapses the four cases into one; furthermore, the diagonal symmetry that results from transposing Δx and Δy makes it necessary to use only the diagonal cells and the cells of either the upper or lower triangle of the reference grid. In the case of Figure 3.1, for instance, this can be stipulated by applying three constraints: $\Delta x \geq 0$, $\Delta y \geq 0$, and $\Delta x \leq \Delta y$. For the reference grid in the 3D implementation, the squared distance for a cell $(\Delta x, \Delta y, \Delta z)$ is given by $(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2$, and only a tetrahedral portion of the 3D array is used, with the constraints being as follows: $\Delta x \geq 0$, $\Delta y \geq 0$, $\Delta z \geq 0$, and $\Delta x \leq \Delta y \leq \Delta z$.

Reference numbers are assigned to the cells of the reference grid in increasing order according to the squared distance values. In the event of a tie (for example, cells (0, 5) and (3, 4) both have the distance value 25), reference numbers for the tying cells are awarded in order of increasing maximum coordinate (continuing the example, since $4 < 5$, cell (3, 4) is labeled as *r13* and cell (0, 5) is labeled as *r14*). The motivation for favoring the cell with the smaller maximum coordinate is that fewer propagation steps are required to reach that cell from the representative exterior cell, cell (0, 0) of the reference grid.

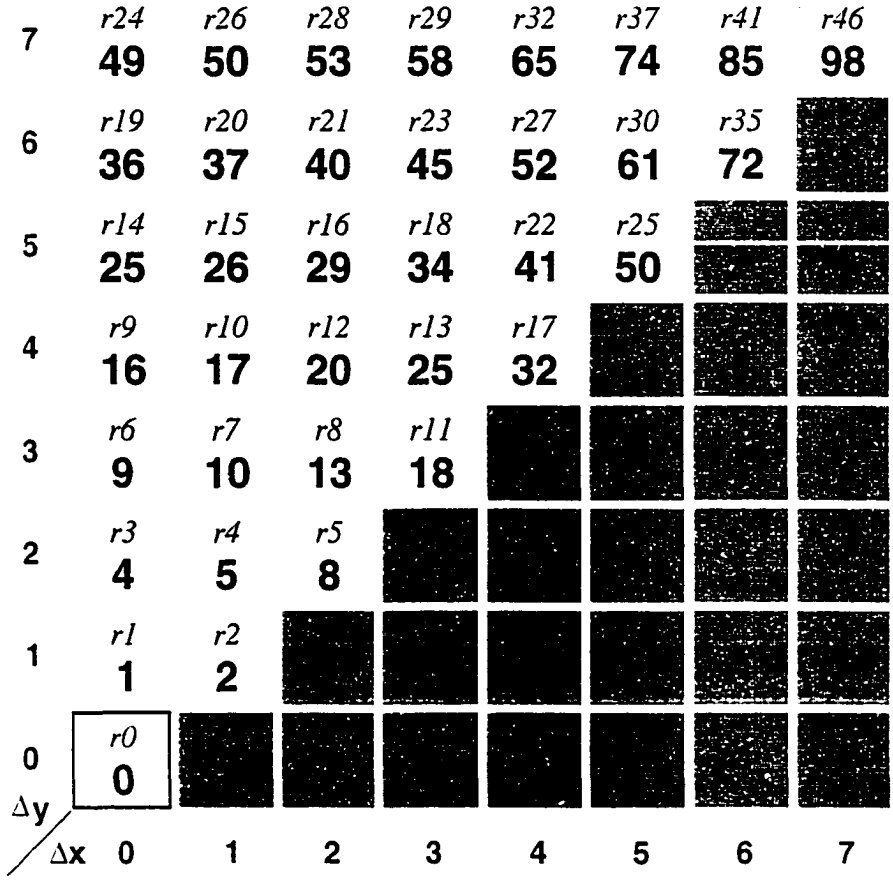


Figure 3.1: The reference grid for use in approximating the 2D Euclidean distance map (only the first eight rows are displayed). Although the reference grid may resemble a distance map, note that it is not a distance map. It is merely an array of the potential relationships between an interior and an exterior voxel that may occur in a distance map, and its sole purpose is to aid in the creation of the the reference table (Table 3.1). In the reference grid above, each cell's distance map value (shown in boldface type) is computed as $(\Delta x)^2 + (\Delta y)^2$, which is simply the square of the Euclidean distance from the center of that cell to the center of the lower left cell labeled *r0* (which represents the nearest exterior cell). Reference numbers (shown in italics directly above each distance map value) are assigned in increasing order based on distance map values, with ties broken as described in the text.

Reference Number	Squared Distance	4-adjacent Generating Reference	8-adjacent Generating Reference
$r0$	0	–	–
$r1$	1	$r0$ (0)	–
$r2$	2	–	$r0$ (0)
$r3$	4	$r1$ (1)	–
$r4$	5	$r2$ (2)	$r1$ (1)
$r5$	8	–	$r2$ (2)
$r6$	9	$r3$ (4)	–
$r7$	10	$r4$ (5)	$r3$ (4)
$r8$	13	$r5$ (8)	$r4$ (5)
$r9$	16	$r6$ (9)	–
$r10$	17	$r7$ (10)	$r6$ (9)
$r11$	18	–	$r5$ (8)
$r12$	20	$r8$ (13)	$r7$ (10)
$r13$	25	$r11$ (18)	$r8$ (13)
$r14$	25	$r9$ (16)	–
$r15$	26	$r10$ (17)	$r9$ (16)
$r16$	29	$r12$ (20)	$r10$ (17)
$r17$	32	–	$r11$ (18)
$r18$	34	$r13$ (25)	$r12$ (20)
$r19$	36	$r14$ (25)	–
$r20$	37	$r15$ (26)	$r14$ (25)
$r21$	40	$r16$ (29)	$r15$ (26)
$r22$	41	$r17$ (32)	$r13$ (25)
$r23$	45	$r18$ (34)	$r16$ (29)

Table 3.1: The reference table for construction of the 2D distance map (the first 24 references are displayed in increasing order). Each row corresponds to a labeled cell in Figure 3.1. The first two columns show the reference number and distance map value from the corresponding cell. The third column lists the 4-adjacent reference (and, for convenience, its associated distance map value) that can generate the row's reference number; likewise, the fourth column lists the 8-adjacent reference that can generate the row's reference number. These generating references correspond respectively to the cells directly below and diagonally below and to the left of the row's associated grid cell. As an example, for the eighth row (for $r7$), the corresponding cell in Figure 3.1, labeled $r7$, has 10 as its distance map value. The cell labeled $r4$ is directly below it, and the cell labeled $r3$ is below and to the left.

To jump ahead for just a moment, note that as the distance map is computed, many different voxels may be labeled with the same reference number. As an example, any interior voxel whose closest exterior voxel is precisely a knight's move away (using chess terminology) will be labeled with an *r4* reference. This illustrates another important point mentioned two paragraphs earlier, namely, that each reference number represents a class of similar directional distance relationships; for instance, even though it appears in cell (1, 2) of the reference grid, the *r4* reference corresponds to any of the following eight directional relationships between an interior voxel and its closest exterior voxel, as measured from the exterior voxel: (2, 1), (1, 2), (−1, 2), (−2, 1), (−2, −1), (−1, −2), (1, −2), and (2, −1).

As reference numbers are assigned within the reference grid, the reference table is created. Each separate assignment causes a single, corresponding row to be appended to the reference table. Table 3.1 shows the first 24 rows of the table for the 2D implementation; these rows can be created using the grid in Figure 3.1.

Each row of the reference table consists of a reference number, the distance map value associated with that reference number, and references to the cells of the reference grid that can generate that row's reference number during the propagation process. These latter items are termed *generating references*. If a reference r_i is a generating reference for reference r_j, then r_j is called a *descendant reference* of r_i. Reference *r4*, for instance, is a descendant reference of *r1* and *r2*; conversely, *r1* and *r2* are the generating references for *r4*.

Fundamentally, in the 2D case, propagation of the reference from a given voxel can occur in two basic directions heading away from the nearest exterior voxel: orthogonally or diagonally. In the context of the reference grid in Figure 3.1, orthogonal

propagation occurs when the reference in cell $(\Delta x, \Delta y - 1)$ is used to generate the reference in cell $(\Delta x, \Delta y)$, and diagonal propagation occurs when the reference in cell $(\Delta x - 1, \Delta y - 1)$ is used to generate the reference in cell $(\Delta x, \Delta y)$. In the former case, the two voxels involved in the propagation must be 4-adjacent, sharing a common edge; in the latter case, the two voxels must be 8-adjacent, sharing a common vertex. The generating references for the reference in grid cell $(\Delta x, \Delta y)$ are stored in the appropriate columns of the corresponding row of the reference table. Obviously, the cells along the diagonal of the reference grid will not have 4-adjacent generating references; likewise, the cells along the left edge of the reference grid will not have 8-adjacent generating references.

Construction of the reference table for the 3D implementation is similar, though since there are three fundamental directions of propagation in the 3D case, there are three columns for generating references. These correspond to 6-adjacency (when the two voxels involved in the propagation share a common face), 18-adjacency (when the two voxels share a common edge), and 26-adjacency (when they share a common vertex). The respective generating references corresponding to the reference number in cell $(\Delta x, \Delta y, \Delta z)$ of the 3D reference grid are contained in cells $(\Delta x, \Delta y, \Delta z - 1)$, $(\Delta x, \Delta y - 1, \Delta z - 1)$, and $(\Delta x - 1, \Delta y - 1, \Delta z - 1)$. Table 3.2 shows the first 32 rows of the reference table for the 3D implementation.

Through the use of dynamic data structures, the reference table and the grid used to generate it can be incrementally computed to whatever size is necessary in order to complete the reference number and distance value propagation. In the execution shown in Figures 3.2 through 3.5 in the next section, for example, the largest distance

Reference Number	Squared Distance	6-adjacent Generating Reference	18-adjacent Generating Reference	26-adjacent Generating Reference
$r0$	0	–	–	–
$r1$	1	$r0$ (0)	–	–
$r2$	2	–	$r0$ (0)	–
$r3$	3	–	–	$r0$ (0)
$r4$	4	$r1$ (1)	–	–
$r5$	5	$r2$ (2)	$r1$ (1)	–
$r6$	6	$r3$ (3)	–	$r1$ (1)
$r7$	8	–	$r2$ (2)	–
$r8$	9	–	$r3$ (3)	$r2$ (2)
$r9$	9	$r4$ (4)	–	–
$r10$	10	$r5$ (5)	$r4$ (4)	–
$r11$	11	$r6$ (6)	–	$r4$ (4)
$r12$	12	–	–	$r3$ (3)
$r13$	13	$r7$ (8)	$r5$ (5)	–
$r14$	14	$r8$ (9)	$r6$ (6)	$r5$ (5)
$r15$	16	$r9$ (9)	–	–
$r16$	17	$r12$ (12)	–	$r6$ (6)
$r17$	17	$r10$ (10)	$r9$ (9)	–
$r18$	18	$r11$ (11)	–	$r9$ (9)
$r19$	18	–	$r7$ (8)	–
$r20$	19	–	$r8$ (9)	$r7$ (8)
$r21$	20	$r13$ (13)	$r10$ (10)	–
$r22$	21	$r14$ (14)	$r11$ (11)	$r10$ (10)
$r23$	22	–	$r12$ (12)	$r8$ (9)
$r24$	24	$r16$ (17)	–	$r11$ (11)
$r25$	25	$r19$ (18)	$r13$ (13)	–
$r26$	25	$r15$ (16)	–	–
$r27$	26	$r20$ (19)	$r14$ (14)	$r13$ (13)
$r28$	26	$r17$ (17)	$r15$ (16)	–
$r29$	27	–	–	$r12$ (12)
$r30$	27	$r18$ (18)	–	$r15$ (16)
$r31$	29	$r23$ (22)	$r16$ (17)	$r14$ (14)

Table 3.2: The reference table for construction of the 3D distance map (only the first 32 rows are shown). The first two columns show the reference number and distance map value for each reference, and the final three columns list the corresponding 6-adjacent, 18-adjacent, and 26-adjacent generating references.

map value assigned is 26 (corresponding to *r15*); hence, in that case, growth of the reference table stopped after 16 rows had been computed.

Note that if the sizes of a reference table and its reference grid are sufficient, the same reference table may be used in the construction of multiple distance maps. In theory, there is a single, infinitely large 2D reference table that suffices for all 2D applications of the algorithm, and there is a single, infinitely large 3D reference table that suffices for all 3D applications of the algorithm; Tables 3.1 and 3.2 simply show the initial portion of these tables. For applications involving the construction of numerous distance maps, rather than recomputing the reference grid and reference table for each map, it may be useful to precompute a sufficiently large reference grid and reference table and to store the reference table in a file for quick recall prior to distance map construction.

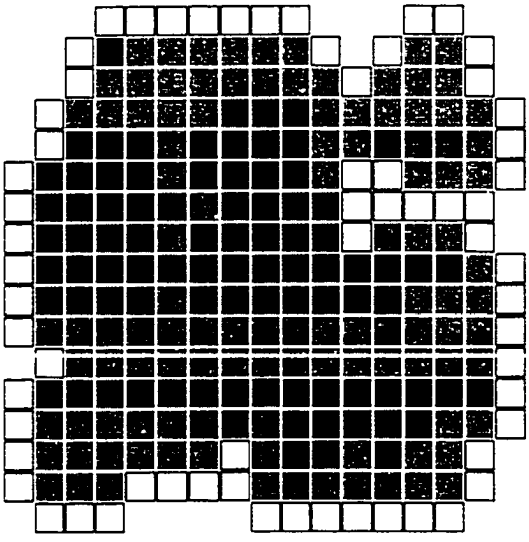
3.3 Propagation of References

Reference numbers are propagated to voxels in increasing order based on information stored in the reference table. During propagation, all reference numbers play two roles: first as a descendant reference (when voxels are labeled with that reference), then later as a generating reference (in order to propagate their own descendant references). The exception to this is *r0*, which is not a descendant reference of any other reference but is simply a label assigned to all exterior voxels during initialization.

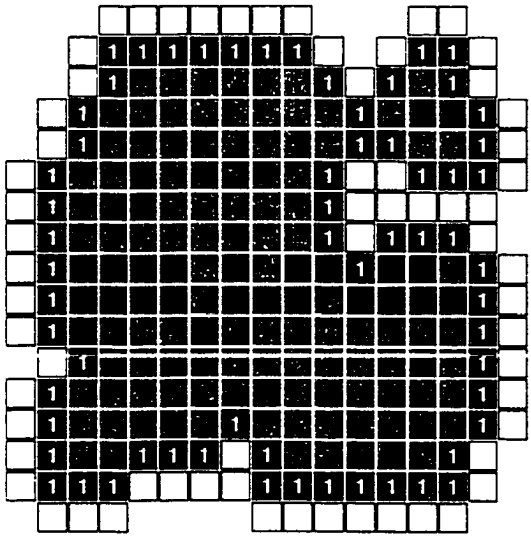
In order to propagate references efficiently, a dynamic array of linked lists is maintained. Each linked list of the array is used for storing pointers to voxels with a particular reference number. The linked list for the array index "0" contains pointers to the exterior voxels; the list with index "1" contains pointers to voxels labeled with

r1 references: the list for "2" has pointers to the *r2* voxels, and so forth. When a voxel is labeled with a particular reference, a pointer to that voxel is inserted into the matching linked list. When voxels with a particular generating reference need to be examined for potential propagation of a descendant reference, the program can simply access the linked list corresponding to the generating reference and examine each voxel in the list.

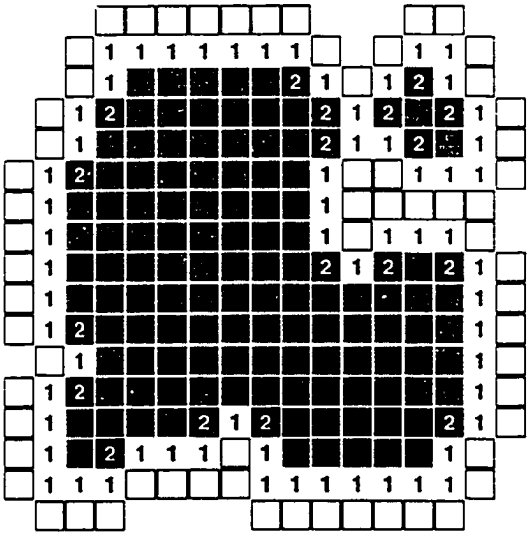
The first row of the reference table, for *r0*, corresponds to the initialization process for the voxel grid. As mentioned earlier, each exterior voxel is labeled as an *r0* voxel, and each interior voxel is initialized as unlabeled. As each successive row of the reference table is computed, the set of unlabeled interior voxels is examined to find the set that should be labeled with the reference number for that row. This amounts to examining voxels labeled with the associated generating reference to see whether they are adjacent to unlabeled interior voxels in the appropriate direction. When the row for *r1* is computed, for instance, all unlabeled interior voxels that are horizontally or vertically adjacent to an *r0* voxel are labeled as *r1* voxels (this is done by stepping through each voxel in the linked list of *r0* voxels and examining its orthogonally adjacent voxels). When the next row (for *r2*) is computed, all unlabeled interior voxels diagonally adjacent to an *r0* voxel are labeled as *r2* voxels. This process continues as the reference table is created and stops when all interior voxels have been labeled. Figures 3.2 through 3.5 on the next four pages demonstrate each propagation step involved in the execution of the distance map approximation algorithm on a collection of voxels designed for illustration purposes. Pseudocode is provided in Figures 3.6 and 3.7.



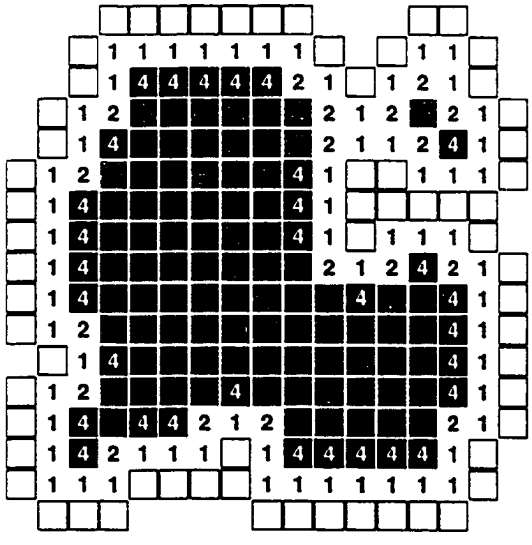
(a) Initial boundary (*r0* references).



(b) Finding *r1* references ($d = 1$).

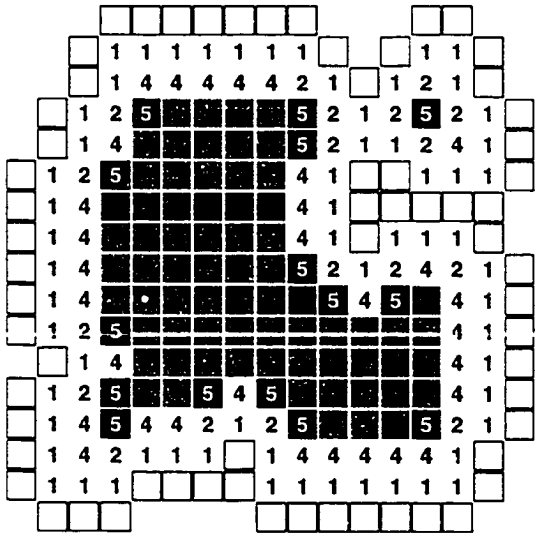


(c) Finding *r2* references ($d = 2$).

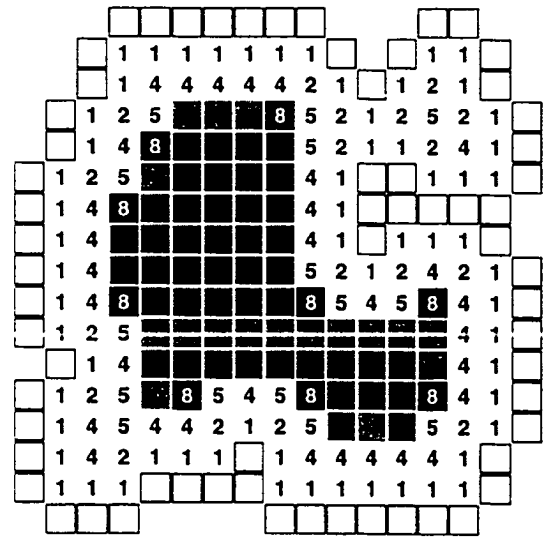


(d) Finding *r3* references ($d = 4$).

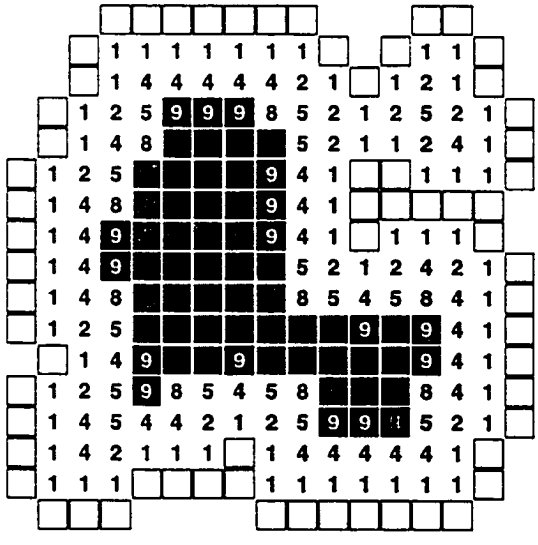
Figure 3.2: Distance map computation (propagation steps 0–3). The figures above and on the following three pages show the results of each propagation step during an execution of the distance map algorithm. In figure (a) above, the initial step involves assigning the reference *r0* to each exterior voxel. Each successive diagram shows the result of assigning the next consecutive reference number from Table 3.1 to appropriate voxels, which are drawn in black and labeled with their distance map values (d). In the actual implementation, the reference numbers are stored as well.



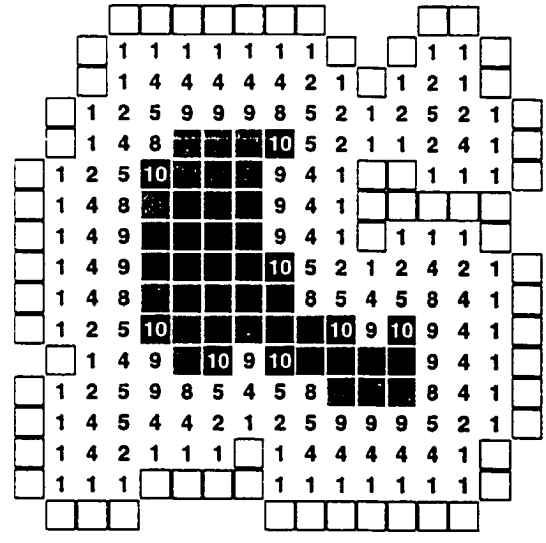
(a) Finding *r4* references ($d = 5$).



(b) Finding *r5* references ($d = 8$).

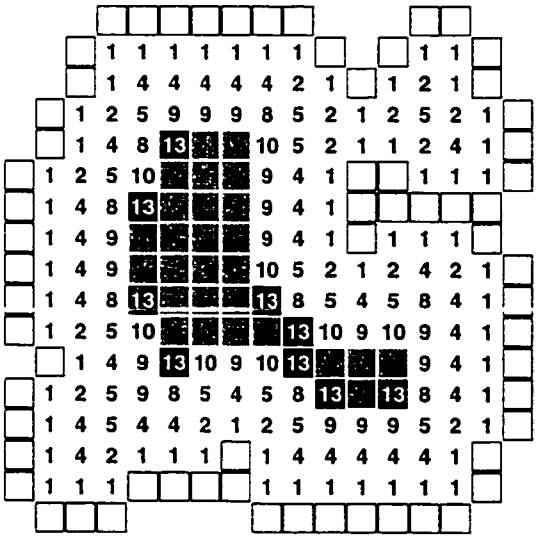


(c) Finding *r6* references ($d = 9$).

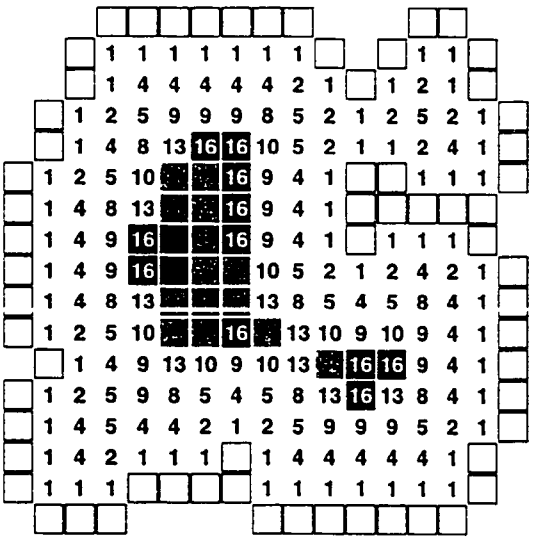


(d) Finding *r7* references ($d = 10$).

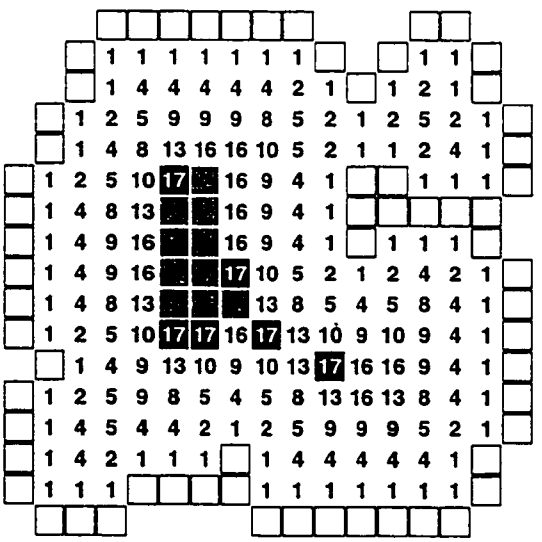
Figure 3.3: Distance map computation (propagation steps 4–7). For the step in figure (a) above, all previously unlabeled voxels that are diagonally adjacent to a "1" (or rather, to an *r1* voxel that in the diagram just happens to be labeled with distance map value of 1) or that are horizontally or vertically adjacent to a "2" (or rather, to an *r2* voxel) are assigned to be *r4* voxels with distance value 5. This propagation step corresponds to the row for *r4* in the reference table (Table 3.1 on page 59).



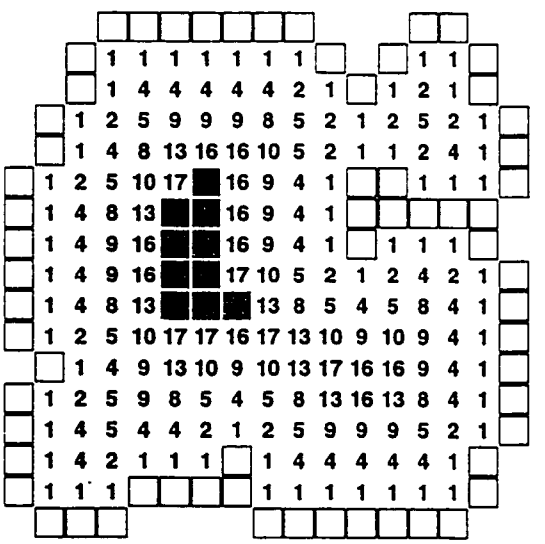
(a) Finding *r8* references ($d = 13$).



(b) Finding *r9* references ($d = 16$).

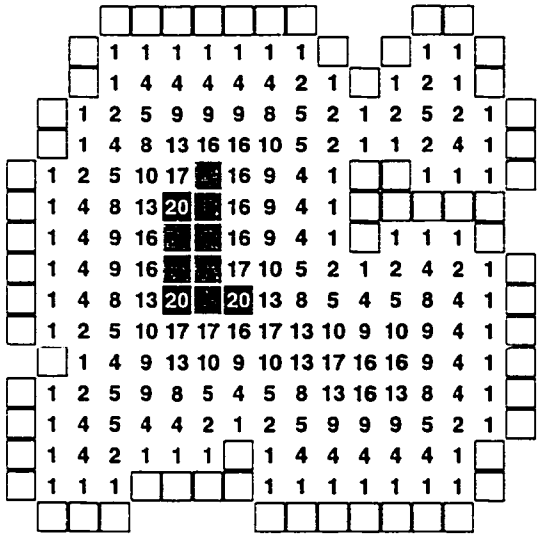


(c) Finding *r10* references ($d = 17$).

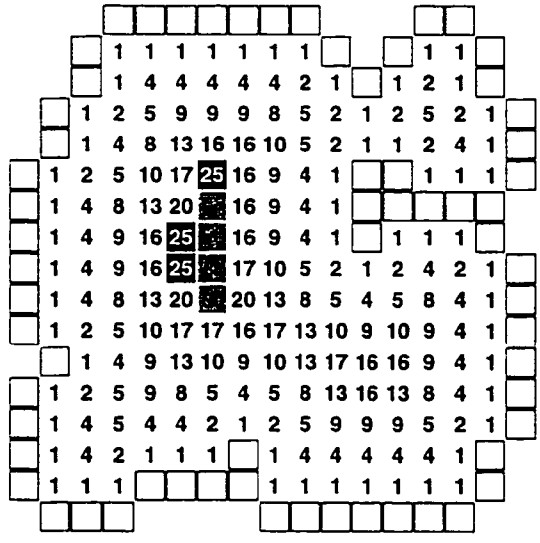


(d) Finding *r11* references ($d = 18$).

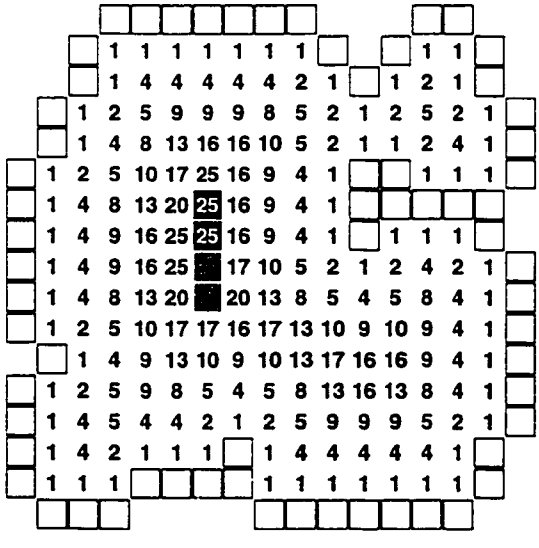
Figure 3.4: Distance map computation (propagation steps 8–11). Note how in the propagation step in figure (d) above, no *r11* references are generated, as *r11* does not have a 4-adjacent generating reference and there are no applicable instances of its 8-adjacent generating reference (that is, there are no unlabeled voxels diagonally adjacent to an "8", which in these diagrams represents an *r5* voxel).



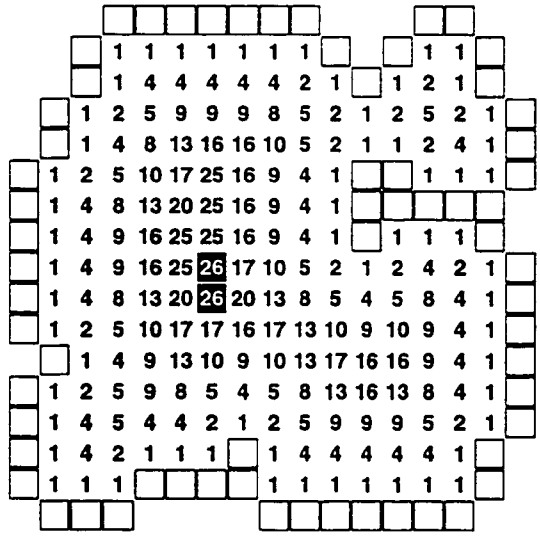
(a) Finding *r12* references ($d = 20$).



(b) Finding *r13* references ($d = 25$).



(c) Finding *r14* references ($d = 25$).



(d) Finding *r15* references ($d = 26$).

Figure 3.5: Distance map computation (propagation steps 12–15). Note in figures (b) and (c) above how the propagation of different reference numbers may result in the same distance map value being produced during separate steps. The accuracy of the approximation algorithm depends upon being able to distinguish between such cases, which is why it is important to propagate the reference numbers and not just the distance map values. After the step in (d), there are no more unlabeled interior voxels, so execution terminates, having produced the distance map values as shown.

```
// handling the reference table

generate the reference grid RG and reference table RT to a
sufficient size (alternatively, the rows of the reference grid and
reference table may be dynamically generated on an as-needed
basis during the propagation computations shown on the next page)

// initialization: flag exterior voxels with reference number zero
// and count the number of interior voxels

numinterior ← 0
for each voxel V
    if V is an exterior voxel
        ref[V] ← 0
        insert V into list[0]
    else
        ref[V] ← UNKNOWN
        numinterior ← numinterior + 1
    end-if
end-for
```

Figure 3.6: Pseudocode for the initialization phase of the distance map algorithm. This figure and the one that follows provide pseudocode for the primary part of the distance map algorithm; they should be read consecutively. Note that *list* is an array of linked lists; $list[N]$ is the linked list designated to hold the voxels to which the reference number N is assigned. The two arrays *ref* and *distance* are designed to store the reference number and the distance map values for each voxel. RG and RT refer to the reference grid and the reference table, respectively. Comments are preceded by a double slash.

```
// propagation: in each iteration. the objective is to find and
// label the voxels that should have reference number N

numlabeled ← 0
N ← 1
repeat until (numlabeled = numinterior)
    // handle the 4-adjacent references
    if there is a 4-adjacent generating reference in row N of RT
        G ← 4-adjacent generating reference from row N of RT
        for each voxel V in list[G]
            for each 4-adjacent neighbor X of V
                if (ref[X] = UNKNOWN)
                    ref[X] ← N
                    insert X into list[N]
                    numlabeled ← numlabeled + 1
                end-if
            end-for
        end-for
    end-if
    // handle the 8-adjacent references
    if there is an 8-adjacent generating reference in row N of RT
        G ← 8-adjacent generating reference from row N of RT
        for each voxel V in list[G]
            for each strictly 8-adjacent neighbor X of V
                if (ref[X] = UNKNOWN)
                    ref[X] ← N
                    insert X into list[N]
                    numlabeled ← numlabeled + 1
                end-if
            end-for
        end-for
    end-if
    N ← N + 1
end-repeat

// assigning distance map values

for each voxel V
    N ← ref[V]
    distance[V] ← squared distance value from row N of RT
end-for
```

Figure 3.7: Pseudocode for the propagation phase of the distance map algorithm.

3.4 Analysis and Discussion

During execution, each voxel is examined a constant number of times (once for initialization, then one time from each adjacent voxel). Construction of the reference grid and reference table is linear with respect to the number of references computed, and because the grid and table are incrementally computed only to whatever size is necessary, the number of distinct references is bounded by the number of interior voxels. The overall time complexity of the algorithm is thus linear with respect to the sum of the number of interior voxels and the number of feature point voxels. In the case when the feature point voxels consist only of the exterior voxels in the layer immediately surrounding the interior voxels, the number of feature points is bounded by a constant multiple of the number of interior voxels, and the time complexity is more simply stated as linear with respect to the number of interior voxels. Although the constant multiplier increases as the dimensionality increases (the number of voxels potentially adjacent to a particular voxel increases with the dimension), the linearity of the time complexity holds regardless of the dimension. See Table 3.3 for typical execution times of the 2D and 3D implementations.

Note that if a vector distance map is desired, a post-processing step can be applied. In order to do this, the reference table must be extended to include the corresponding row and column (with respect to the reference grid) for each reference number. After each reference number has been computed in the final grid, the reference number for a given voxel v can be used to index into the reference table to find the corresponding row and column of that reference in the reference grid. This provides an offset vector $(\Delta x, \Delta y)$ that suggests where to look for the relative location of the exterior voxel causing that reference. Due to the formulation of the reference grid, the offset vector

2D Distance Map	
Voxel Grid Dimensions	Approximate Execution Time
256×256	0.6 sec
512×512	2.7 sec
1024×1024	11 sec
2048×2048	57 sec

3D Distance Map	
Voxel Grid Dimensions	Approximate Execution Time
$32 \times 32 \times 32$	0.3 sec
$64 \times 64 \times 64$	2.6 sec
$128 \times 128 \times 128$	26 sec

Table 3.3: Approximate execution times for the 2D and 3D implementations of the Euclidean distance map approximation algorithm. The times shown are the average computation times over multiple test runs on grids in which the boundary voxels and a few other randomly chosen voxels were marked as feature points. Execution was performed on a Silicon Graphics® O2® (R5000 Processor Chip).

must be combined with the coordinates of the particular voxel v in each of eight ways to test for the presence of the exterior voxel. If the coordinates of v are (a, b), then the eight voxels to test will have the coordinates $(a + \Delta x, a + \Delta y)$, $(a + \Delta x, a - \Delta y)$, $(a - \Delta x, a + \Delta y)$, $(a - \Delta x, a - \Delta y)$, $(a + \Delta y, a + \Delta x)$, $(a + \Delta y, a - \Delta x)$, $(a - \Delta y, a + \Delta x)$, and $(a - \Delta y, a - \Delta x)$. The test is used to determine the appropriate signs with which the offset vector should be amended before being assigned as the distance map vector for v.

The algorithm described in this chapter is a contour style ordered propagation algorithm along the lines of those of Verwer, Verbeek, and Dekker [VVD89] and Ragnemalm [Rag92a] (see page 18). In contrast to the algorithm of Verwer et al., which computes a non-Euclidean distance map with scalar values, the algorithm given here uses Euclidean distance values and provides a very close approximation to the EDM. Due to the presence of the reference grid and the fact that reference labels are propagated as opposed to distance map values, the algorithm given here also

allows easy conversion of the reference labels to create either a (scalar) distance map or a vector distance map as mentioned earlier. Ragnemalm's algorithm [Rag92a] uses Euclidean distances, however, the method of bucketing is based on the squared Euclidean distances and is thus not as economical as the bucketing method used here, which is based on the reference numbers.

There is an additional difference between the algorithms that has to do with the way the bucketing is performed. The algorithms of Verwer et al. [VVD89] and Ragnemalm [Rag92a] take a voxel from the current bucket and use it to assign values/vectors to voxels yet to be processed: then, after processing those voxels, those algorithms insert the voxels into buckets to be processed later. This will be called *forward processing.* The algorithm in this chapter inverts that process, examining voxels in earlier buckets to determine which voxels should be assigned the current value/reference and inserted into the current bucket. This will be referred to as *inverted processing.* Although forward processing may not affect the accuracy of the resulting distance map when a non-Euclidean metric is employed, it can have an adverse effect on the accuracy when a Euclidean metric is used. As an example, examine the reference grid from Figure 3.1 and observe that the reference *r6* ($d = 9$) can generate the reference *r10* ($d = 17$) along a diagonal, and that the reference *r5* ($d = 8$) can generate the reference *r11* ($d = 18$) along a diagonal. Now consider the case of an unprocessed voxel v being diagonally adjacent to both an *r5* and an *r6* voxel during propagation. Forward processing would result in the *r5* voxel being processed first, thus assigning v the value 18, which is one greater than necessary. If each voxel is processed only once, as is the case with these contour style ordered propagation algorithms, then the inaccurate assignment for v would not be corrected. Such an inaccuracy could

then be further propagated, causing more errors in the resulting distance map. In the situation just described, inverted processing would have the result that v is not assigned a value until the *r10* references are assigned, and so v would be assigned the more accurate value 17. Closer inspection of the reference grid will reveal that many more situations like this one can occur during either diagonal or orthogonal propagation, and it should also be clear that this type of problem can result when forward processing is used regardless of whether values, vectors, or reference numbers are the actual elements being propagated. Thus, with respect to approximating the Euclidean distance map with contour style ordered propagation algorithms, using inverted processing will produce distance maps that are at least as accurate if not more accurate than the distance maps produced when using forward processing.

The algorithm in this chapter produces an approximation to the Euclidean distance map – errors may be introduced by the method in which references and distance map values are propagated. These errors, although small, can cause the distance map not to be an exact Euclidean distance map. Although no formal analysis of all possible errors of the approximation algorithm has been performed, detailed observations of errors have been made over hundreds of executions of the algorithm for actual applications as well as for contrived test data. These observations are summarized in Table 3.4. The 2D approximation algorithm has been observed to produce a maximum error (as compared to the exact Euclidean distance) of about 0.068288 units, and the 3D approximation algorithm has been observed to produce a maximum error of about 0.071068 units (in both cases, the error is less than $\frac{1}{14}$ of the edge length of a voxel). Note that the vast majority of distance map values computed by the algorithm are the same as those that would be computed by an exact EDM algorithm.

Computed Value	Exact EDM Value	Difference	Absolute Error	Relative Error
d_{comp}	d_{exact}	$d_{\text{comp}} - d_{\text{exact}}$	$\sqrt{d_{\text{comp}}} - \sqrt{d_{\text{exact}}}$	$\frac{\sqrt{d_{\text{comp}}} - \sqrt{d_{\text{exact}}}}{\sqrt{d_{\text{exact}}}}$
2D Distance Map				
170	169	1	0.038405	0.00295422
676	671	2	0.038190	0.00118258
484	481	3	0.068288	0.00311366
2045	2041	4	0.044248	0.000979432
5625	5620	5	0.033341	0.000444741
17530	17524	6	0.022660	0.000171179
15129	15122	7	0.028459	0.000231424
16945	16937	8	0.030732	0.000236141
9049	9040	9	0.047317	0.000497664
none observed		10	-	-
25405	25394	11	0.034510	0.000216563
58865	58853	12	0.024731	0.000101944
59189	59176	13	0.026719	0.000109836
none observed		14	-	-
60736	60721	15	0.030434	0.000123508
3D Distance Map				
50	49	1	0.071068	0.0101525
205	203	2	0.070014	0.00491403

Table 3.4: Observed errors for the 2D and 3D implementations of the Euclidean distance map approximation algorithm. Although the distance map values are given as squared distances, the absolute and relative errors are computed according to the actual (unsquared) distances – see the formulas in the second row, and note that d_{comp} is the squared distance as computed and d_{exact} is the exact squared Euclidean distance. The maximum possible error for each observed deviation from an exact EDM value is listed; for example, in one instance for the 2D algorithm, an observed value of 361 was computed when the exact value was 360, but the error for this deviation by 1 unit is less than the error for the 1 unit deviation for the case of computing 170 instead of 169; thus, it is the 170 versus 169 case that appears in the table. For the 2D approximation algorithm, the largest observed error is about 0.068288 units; for the 3D algorithm, the largest observed error is about 0.071068 units.

As observed for both the 2D and 3D implementations, less than one percent of voxels receive values differing from the exact EDM (the figure is usually somewhere between 0.01% and 0.1% for most objects used during applications of this research).

For an exact EDM, each voxel has a value or reference reflecting the closest exterior voxel. This means that the set of voxels whose values or references correspond to a particular exterior voxel v_e must be the same set of voxels whose center points lie within the Voronoi region defined by v_e when viewing the Voronoi diagram induced by the center points of the exterior voxels. The propagation method used by the algorithm described in this chapter fails because a sliver section of a Voronoi region may cause discontinuities between the set of voxels considered to belong to that Voronoi region, that is, the set may not be connected. Figures 3.8 and 3.9 illustrate an error caused by propagation around such a sliver region. The voxels of the Voronoi region for exterior voxel (or feature point) B contain a voxel disconnected from the rest, and this problem voxel is assigned a reference associated with exterior voxel A instead, with the result being that the problem voxel receives a distance map value that is one unit greater than what it should have in an exact EDM.

The root of the problem just described lies in the fact that the use of a local neighborhood for the simple propagation that takes place in this algorithm is insufficient for exact EDM computation in the general case. The local neighborhood works well enough to compute an exact EDM for some cases of input (the example in Figures 3.2 through 3.5, for instance). As is clearly shown in the example in Figures 3.8 and 3.9, however, in other cases there might be sliver sections of Voronoi regions that undermine simple propagation based on examination of only a local region of nearby

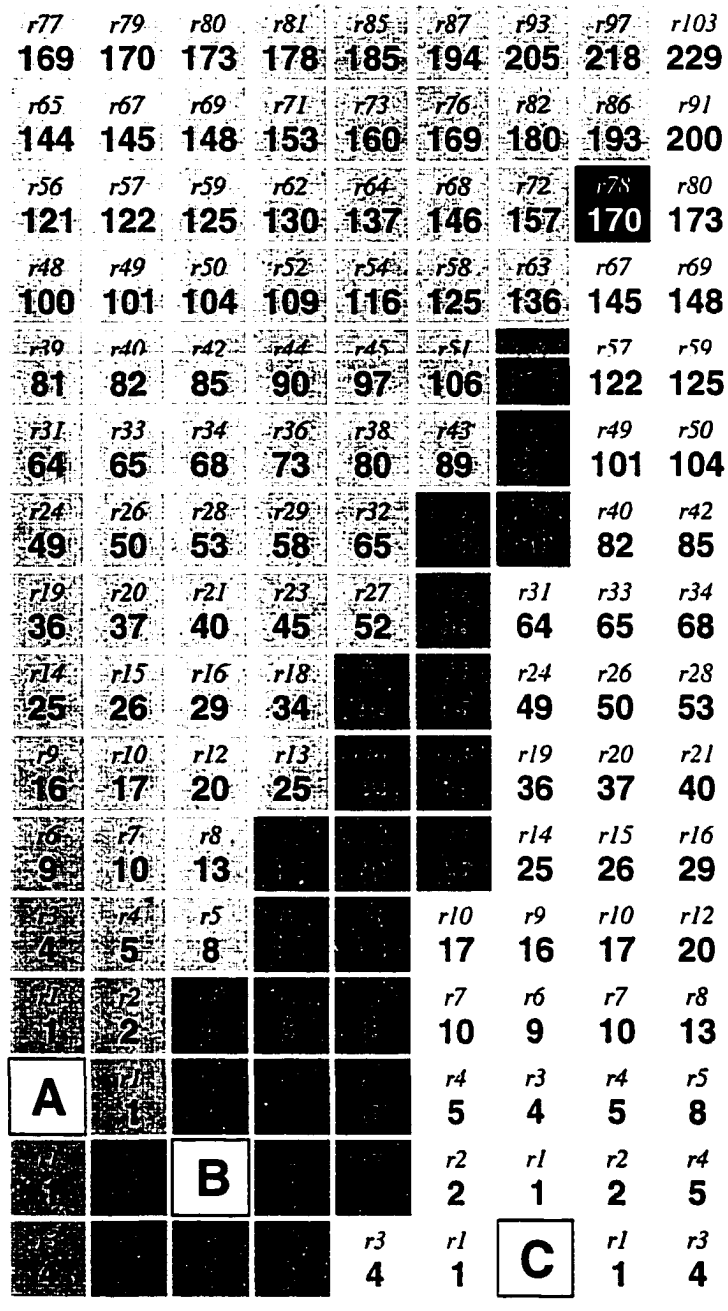


Figure 3.8: An example of the type of error introduced by using local propagation when computing the Euclidean distance map. The black voxel in the upper right, with reference *r78* and squared distance 170, has been labeled incorrectly. Although it is $\sqrt{170}$ units away from the feature points A and C, it is only $\sqrt{169} = 13$ units away from feature point B. The other voxels have been shaded according to which feature point has caused them to be labeled as they are. Compare this with Figure 3.9.

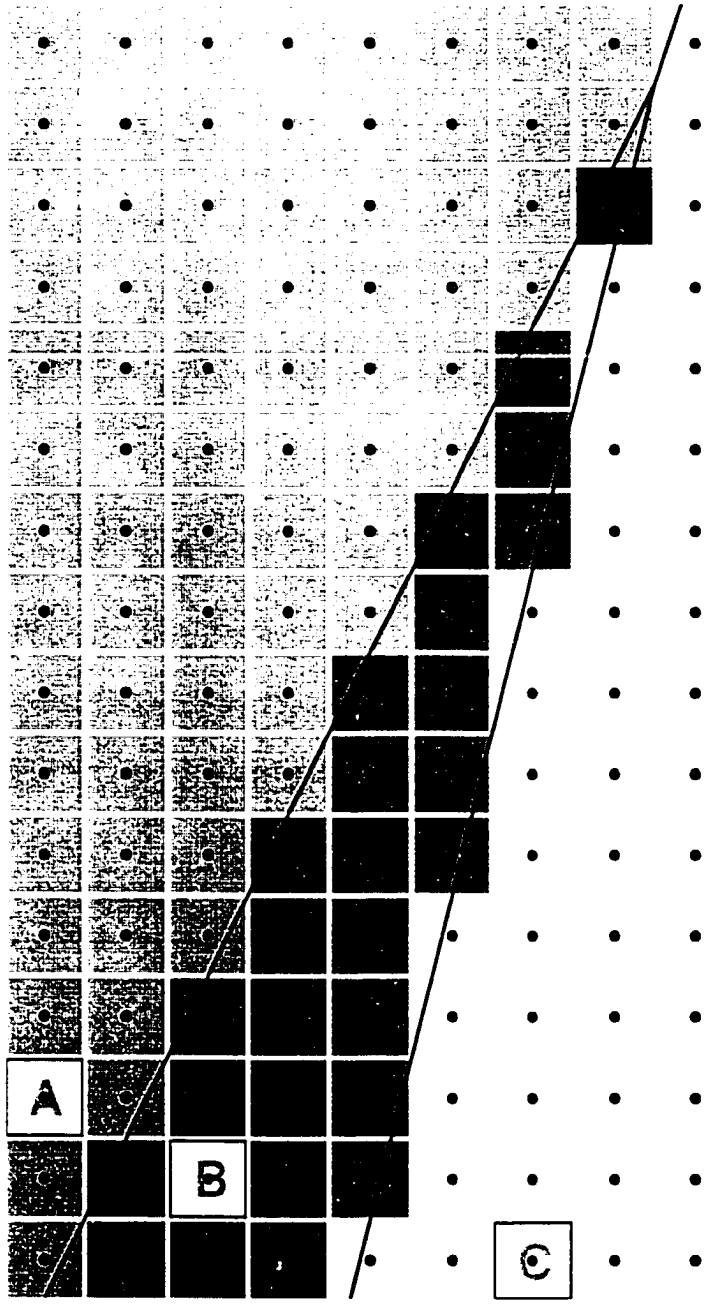


Figure 3.9: The Voronoi diagram for the error example of Figure 3.8. The lines are the boundaries between the Voronoi regions for feature points **A**, **B**, and **C**, and voxels are shaded according to the Voronoi region containing their centers (note the exact correspondence with the shading from Figure 3.8). The problem voxel, shaded dark gray in the upper right, lies in the Voronoi region for **B**; however, it has no adjacent voxels that will propagate references corresponding to **B**.

voxels. Nevertheless, there are a few ways to circumvent the problem and thus arrive at a robust algorithm that consistently computes an exact EDM:

1. Change the information that is propagated. For each voxel, save and propagate references to all exterior voxels to which the Euclidean distance is within one unit of that to the closest exterior voxel. After processing is finished, simply report the minimum distance value and reference at each voxel. A raster scanning type algorithm based on this idea is proposed by Mullikin [Mul92].

2. Increase the size of the local neighborhood so that propagation does not have to occur between adjacent voxels. Theoretically, though, a global neighborhood is the only neighborhood sufficient to overcome all possible slivers, so by itself, increasing the local neighborhood will just decrease the error inherent in the approximation. Unfortunately, use of a global neighborhood results in a quadratic time algorithm.

3. Compute part or all of the Voronoi diagram either to assist with propagation or to make propagation unnecessary. For the case of 2D distance maps, partial computation of the Voronoi diagram has led to an exact EDM that operates in linear time with respect to the number of voxels [BGKW95]. For higher dimensions, due to the increased time complexity required for computing the Voronoi diagram, it is unclear whether its partial computation could lead to efficient construction of the exact EDM, much less a linear time algorithm.

When considering efficient computation of the exact EDM under any dimension, the first option offers the most promising approach. Indeed, the 2D and 3D versions of the approximation algorithm described in this section have been used as the basis for

exact EDM implementations, with modifications along the lines suggested by the first option. Precise analysis of the time complexity of these exact algorithms is difficult, however, and it may be the case that their time complexities are not linear.

CHAPTER 4

CONSTRUCTING THE DISCRETE MEDIAL SURFACE

This chapter describes the second of the two main discrete geometry algorithms fundamental to this research. The first algorithm, designed to approximate the Euclidean distance map, is the focus of the previous chapter. This chapter discusses the algorithm developed for computing the discrete medial axis (DMA) of a 2D discretized object or the discrete medial surface (DMS) of a 3D discretized object. The concepts of the distance map, the medial axis (MA) and the medial surface (MS) were introduced in Sections 2.1 and 2.2 in Chapter 2.

An overview of the algorithm is given in Section 4.1. Section 4.2 then describes the elements of the exposure calculation and how the concept of exposure is used to help identify DMA/DMS voxels. The heart of the algorithm is presented in Section 4.3, in which the stepwise processing of voxels with common distance map values is explained. Results appear in Section 4.4, and analysis and discussion of the algorithm follow in Section 4.5.

As with the distance map algorithm in the last chapter, the DMA/DMS algorithm detailed in this chapter generalizes to higher dimensions. The algorithm will be explained and illustrated for the 2D case, and both 2D and 3D results will be presented. Extensions to three dimensions or higher that are not obvious will be discussed.

4.1 Overview of the Algorithm

The algorithm extracts the DMA/DMS for an object from its Euclidean distance map (EDM) by attempting to identify and track the ridges of the EDM (recall the basic discussion of this process from page 29 in Chapter 2). Whereas in the EDM approximation algorithm, voxels are basically processed in increasing order of distance map value, in the DMA/DMS algorithm presented here, voxels are processed in decreasing order of distance map value.

During the processing of a voxel, an *exposure*[13] calculation is utilized to help determine whether a voxel belongs to the DMA/DMS. Two exposure measures are used: *relative exposure*, which is the amount by which the disk/sphere for one voxel protrudes from the disk/sphere for a specific neighboring voxel (this is diagrammed in Figure 4.1), and *local exposure*, which is the minimum relative exposure a voxel has when considering all of its neighbors (the "local" qualifier will frequently be omitted).

Other processing is performed in order to ensure that each separate region of processed voxels contains a single, connected portion of the DMA/DMS. The DMA/DMS algorithm thus uses two basic classes of DMA/DMS voxels: true DMA/DMS voxels (whose exposure values meet or exceed a given threshold), and bridging DMA/DMS voxels (whose exposures values themselves are insufficient, but that act as the connecting voxels between clusters of true DMA/DMS voxels).

The input to the algorithm is an EDM (or a close approximation) for a discretized object – distance map values are assumed to be squared Euclidean distances. Interior and exterior voxels must be clearly marked. In addition to the distance map, one input

[13]The term "exposure" was coined to denote the amount by which a disk/sphere for one voxel is exposed in relationship to one or more disks/spheres for adjacent voxels.

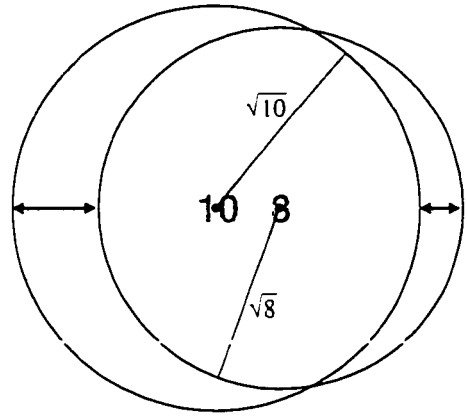


Figure 4.1: The relative exposure of neighboring disks. The voxel on the left has a distance map value of 10 and thus an associated disk of radius $\sqrt{10}$ as shown: the voxel on the right has a distance map value of 8 and thus an associated disk of radius $\sqrt{8}$. The relative exposure of the "10" voxel with respect to the "8" voxel is the extent to which the $\sqrt{10}$ disk protrudes from the $\sqrt{8}$ disk and is the length of the double arrow on the left. Given that the centers of the two voxels are separated by a distance of one unit, this length is given by the expression $\sqrt{10}+1-\sqrt{8}$, which is approximately 1.334 units. Similarly, the relative exposure of the "8" voxel with respect to the "10" voxel is the length of the double arrow on the right, or $\sqrt{8}+1-\sqrt{10} \approx 0.666$ units.

parameter can be specified by the user - the exposure threshold, briefly mentioned in the previous paragraph but described in more detail in the next section. The output of the algorithm is simply a labeling of each interior voxel as to whether it is a DMA/DMS voxel.

4.2 Local Exposure Calculation

The algorithm extracts the DMA/DMS from the distance map in part through a local analysis of the exposure for each voxel, or rather, the exposure of the associated disk or sphere for each voxel. Recall that the distance map value for a voxel is the square of the radius of a disk or sphere that is centered at the center of the voxel and

that just touches the boundary of the object. The exposure, then, provides a measure useful for determining whether the voxel is the center of a maximal disk or sphere, which is the pivotal component in the definition of the medial axis or surface (see the first paragraph of Section 2.2 for the definition). In other words, the exposure of a voxel is a rough measure of its relative importance to the DMA/DMS in comparison to its neighbors. It corresponds somewhat with whether a voxel is considered to lie on a ridge or plateau of the landscape created when the distance map is viewed as a height field for an object in the immediately higher dimension (see Figure 2.7 and the related discussion in the text on page 29).

To compute the exposure for a voxel, it is necessary to compare the distance value for the voxel with that for each of its neighbors in a way that corresponds to the adjacency relationship between the two voxels. Let d_i be the distance map value for voxel v_i, and let v_n represent a neighboring voxel (with distance value d_n). The relative exposure of v_i with respect to v_n, denoted $e(v_i : v_n)$, is the amount by which the disk/sphere for v_i protrudes from the disk/sphere for v_n. This amount can be computed as follows:

$$e(v_i : v_n) = \sqrt{d_i} + \text{distance}(v_i, v_n) - \sqrt{d_n}$$

where "distance(v_i, v_n)" is the distance between the centers of v_i and v_n.

The local exposure e_i for voxel v_i is the simply the minimum of the relative exposures of v_i with respect to each of its neighbors. For purposes of computation, it is handy to partition the neighbors into groups according to their adjacency to a voxel. For the 2D case, let N_4^i be the set of voxels that share an edge with v_i and let N_8^i be the set of voxels that share a vertex (but not an edge) with v_i. The local exposure is then computed as the minimum of the exposures for each adjacency type

(e_{i_4} is the exposure for v_i with respect to its 4-adjacent neighbors, e_{i_8} is the exposure with respect to its strictly 8-adjacent neighbors):

$$e_{i_4} = \min_{v_n \in N_4^i} e(v_i : v_n) = \min_{v_n \in N_4^i} (\sqrt{d_i} + 1 - \sqrt{d_n})$$

$$e_{i_8} = \min_{v_n \in N_8^i} e(v_i : v_n) = \min_{v_n \in N_8^i} (\sqrt{d_i} + \sqrt{2} - \sqrt{d_n})$$

$$e_i = \min\{e_{i_4}, e_{i_8}\}$$

For the 3D case, the neighbors for a voxel v_i are partitioned into three sets: 6-adjacent neighbors (N_6^i), strictly 18-adjacent neighbors (N_{18}^i), and strictly 26-adjacent neighbors (N_{26}^i). The local exposure is computed in a fashion similar to the 2D case:

$$e_{i_6} = \min_{v_n \in N_6^i} e(v_i : v_n) = \min_{v_n \in N_6^i} (\sqrt{d_i} + 1 - \sqrt{d_n})$$

$$e_{i_{18}} = \min_{v_n \in N_{18}^i} e(v_i : v_n) = \min_{v_n \in N_{18}^i} (\sqrt{d_i} + \sqrt{2} - \sqrt{d_n})$$

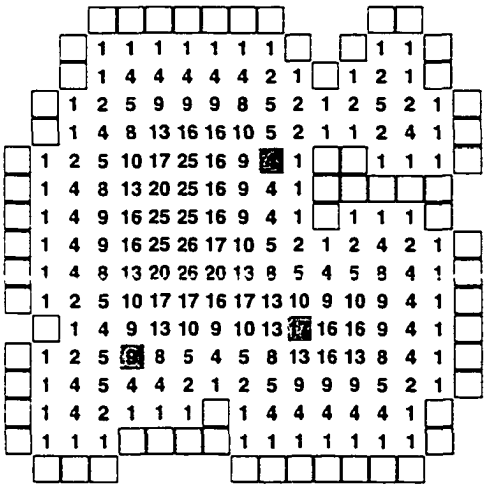
$$e_{i_{26}} = \min_{v_n \in N_{26}^i} e(v_i : v_n) = \min_{v_n \in N_{26}^i} (\sqrt{d_i} + \sqrt{3} - \sqrt{d_n})$$

$$e_i = \min\{e_{i_6}, e_{i_{18}}, e_{i_{26}}\}$$

Extending the exposure computation to higher dimensions is straightforward.

Note that for the actual implementation, a few CPU cycles can be saved by first finding the maximum distance value amongst the voxels in each partition, since that distance value will result in the minimum exposure with respect to the corresponding partition. It then suffices to apply the exposure calculation one time for each partition and to report the minimum of those results. Figure 4.2 on the next page illustrates this method of computing the exposure.

Potentially, exposure values can range from 0 to 2. The vast majority of voxels have exposure values less than 0.5, but most voxels that should belong to the



EXPOSURE FOR THE "4"

Adj	Max Nbr	Calculation	Exp
4-adj	9	$\sqrt{4}+1-\sqrt{9}$	0.000
8-adj	10	$\sqrt{4}+\sqrt{2}-\sqrt{10}$	0.252
		Local Exposure	0.000

EXPOSURE FOR THE "9"

Adj	Max Nbr	Calculation	Exp
4-adj	9	$\sqrt{9}+1-\sqrt{9}$	1.000
8-adj	13	$\sqrt{9}+\sqrt{2}-\sqrt{13}$	0.809
		Local Exposure	0.809

EXPOSURE FOR THE "17"

Adj	Max Nbr	Calculation	Exp
4-adj	16	$\sqrt{17}+1-\sqrt{16}$	1.123
8-adj	16	$\sqrt{17}+\sqrt{2}-\sqrt{16}$	1.537
		Local Exposure	1.123

Figure 4.2: Examples of calculating the exposure for the three shaded voxels in the distance map on the left. The tables on the right show the maximum distance values for the 4-adjacent and strictly 8-adjacent neighbors of the shaded voxels (the column heading abbreviations are for adjacency, maximum neighbor, and exposure). These values are plugged into the exposure equation, and the minimum of the results is the exposure for the voxel. The local exposures for the shaded "4", "9", and "17" voxels are thus computed to be 0.0, 0.809, and 1.123, respectively. Figure 4.3 shows the exposure values for all of the voxels.

DMA/DMS have exposure values of 0.4 to 0.5 or greater. Specifying an exposure threshold somewhere in the range 0.4 to 0.5 and examining the voxels meeting or exceeding that threshold gives a good indication of the DMA/DMS. These voxels are usually not completely connected; instead, they form connected clusters that dot the ridges and plateaus implied within the distance map. Such clusters can be seen in Figure 4.3, in which the exposure threshold is 0.5.

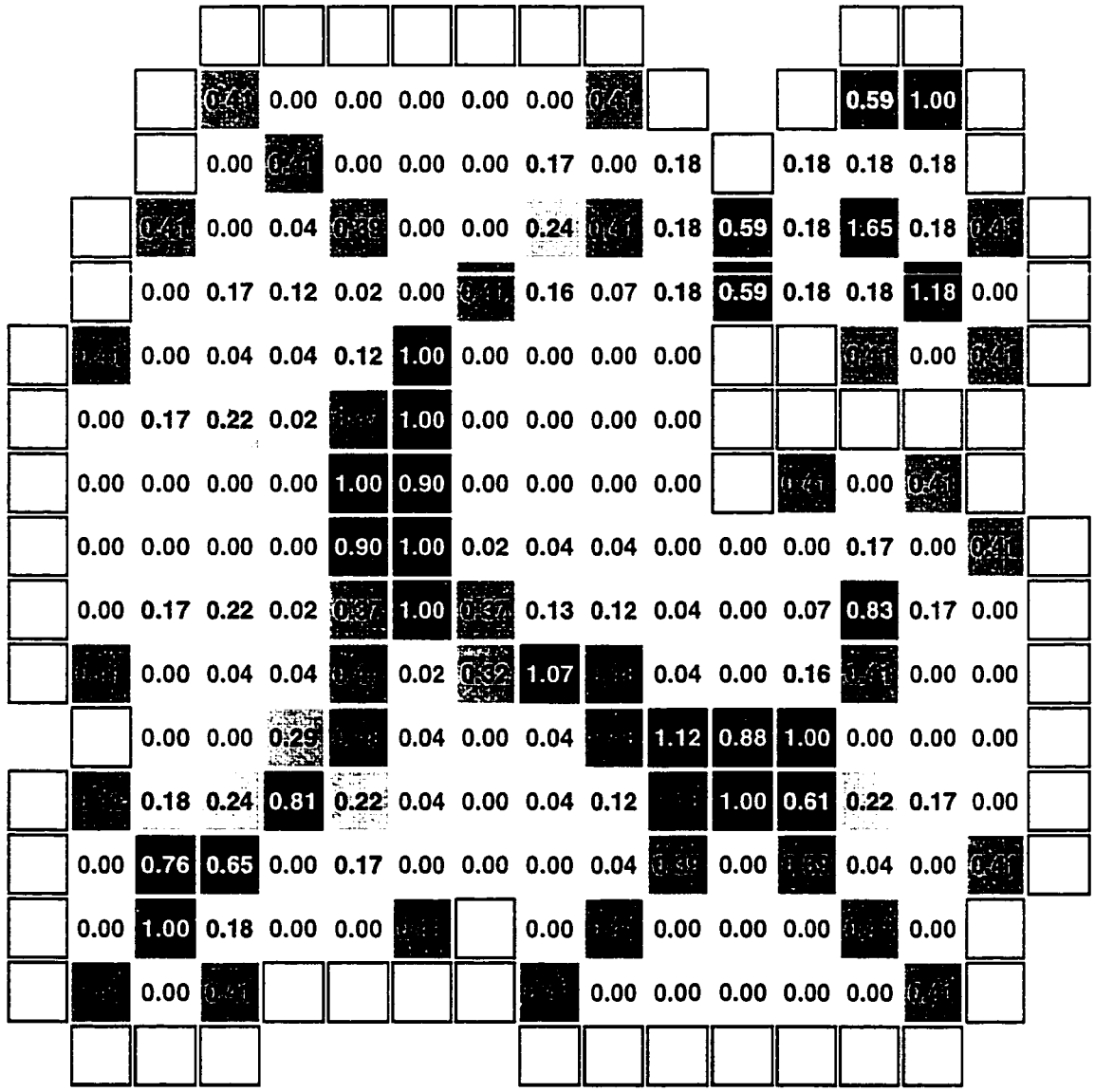


Figure 4.3: The complete grid of exposure values computed for the distance map shown in Figure 4.2. Exposure values have been rounded to the nearest hundredth: values of 0.5 or greater are shown in white lettering to illustrate the clustering that results from simple application of an exposure threshold of 0.5. Voxels are shaded with graylevels according to where their exposure values fall in the range from 0 (white) to 1 (black) – voxels with exposures greater than or equal to 1.0 are colored black.

In most applications, and particularly in the context of this research, it is useful to have a fully connected DMA/DMS; for this reason, further processing is performed to find suitable voxels that will work to connect the clusters. Note that the processing described in the following section does not directly correspond to the concept of connecting clusters as was just mentioned; nevertheless, it should be clear after reading that forming such connections is the main by-product of the processing.

4.3 Extracting the DMA/DMS

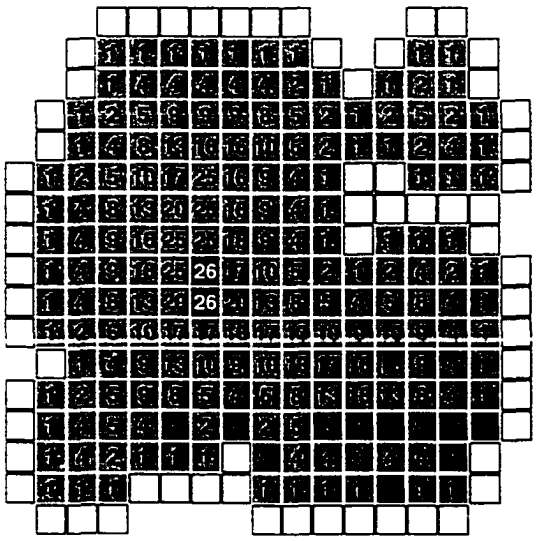
As mentioned above, the exposure value of a voxel is the first indicator of whether a voxel is a DMA/DMS voxel, but examination of exposure values alone is insufficient to guarantee a connected DMA/DMS. Some voxels must be used as bridging voxels to connect the DMA/DMS even though their exposure values are below the threshold. In order to help identify these bridging DMA/DMS voxels, it is important that voxels be processed in a certain order. Therefore, before the actual processing of a voxel is discussed, a few paragraphs will be spent discussing the ordering of voxels to be processed.

After each voxel has been assigned an exposure value, the voxels are organized into a sorted array of lists corresponding to each distinct distance map value. Voxels with a common distance value are inserted into a single, matching list, corresponding to a particular contour level of the distance map. This array of contour lists is sorted in decreasing order according to distance value, and the contour lists are then processed in sequence beginning with the list for the largest distance value. Figures 4.4 through 4.7 on pages 90 through 93 illustrate the execution of the algorithm as it processes each

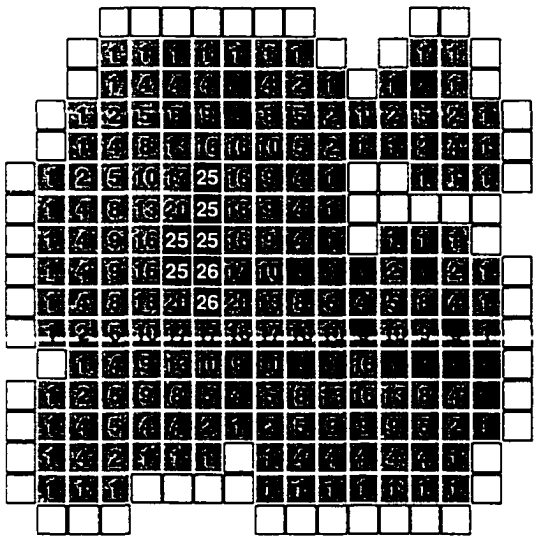
contour level of a distance map (the same distance map computed in the illustration for the EDM approximation algorithm - Figures 3.2 through 3.5).

The distance map value of a voxel thus determines the contour level in which the voxel will be processed. With regard to the processing of a particular contour, though, the voxels having the largest number of processed neighbors are given priority. Processing the voxels in this order helps to reduce the number of connected clusters of processed voxels at any stage of the execution and also simplifies the case-wise analysis used to determine how a voxel should be processed. To help achieve the prioritization, just before a particular contour list is processed, its voxels are partitioned into arrays (called contour arrays) corresponding to the number of processed neighbors each voxel has. The processing of the contour, then, consists of stepping through this array of arrays beginning with the one corresponding to the largest number of processed neighbors.

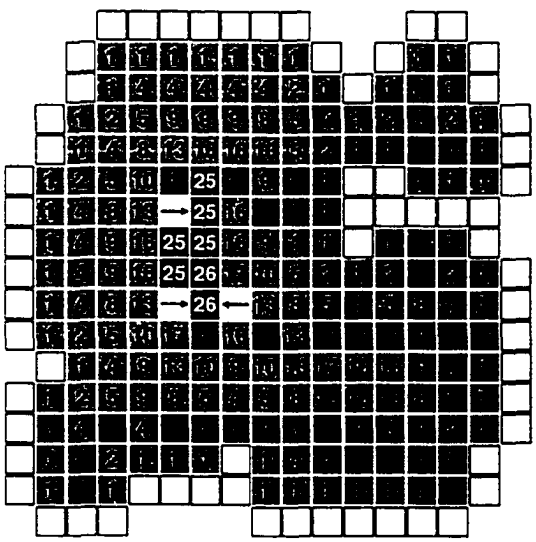
Each unprocessed voxel keeps a record of how many of its neighbors have been processed. To help maintain these records, whenever a voxel is processed, it increments the counter for each of its unprocessed neighbors. If the neighbor has the same distance value as the current voxel being processed, then reference to that neighbor must be transferred from its containing contour array to the array corresponding to having one more processed neighbor. As an example, given that the current voxel and a particular neighbor both have the same distance value (that of the contour list), if the neighbor voxel itself has 2 processed neighbors before the current voxel is processed, then it resides in contour array #2. After the current voxel has been processed, though, that neighbor will have 3 processed neighbors, and so it will need to be moved from contour array #2 to contour array #3. In addition to having a



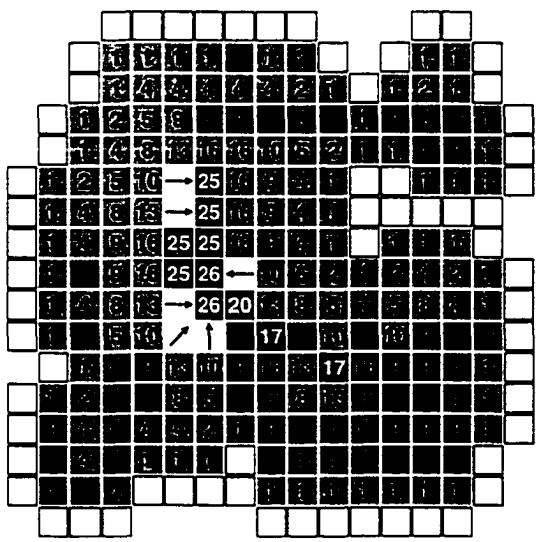
(a) Step 0: ($d = 26$).



(b) Step 1: ($d = 25$).

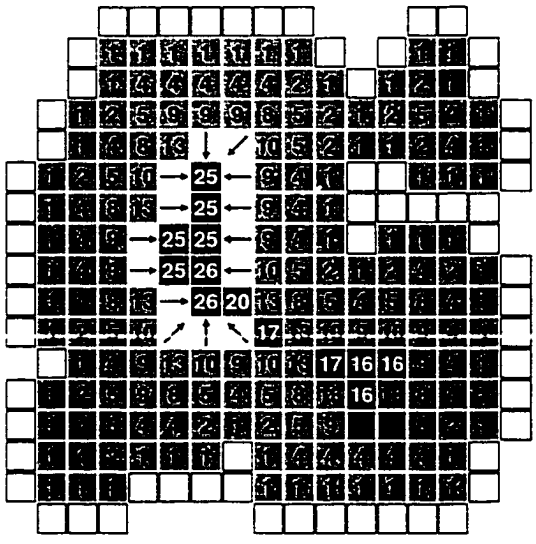


(c) Step 2: ($d = 20$).

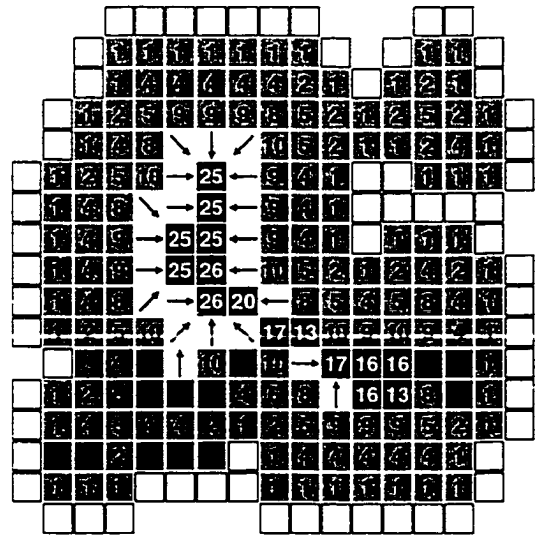


(d) Step 3: ($d = 17$).

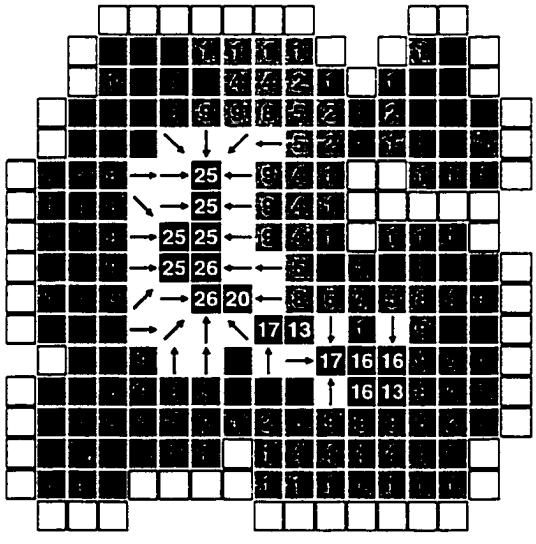
Figure 4.4: Discrete medial axis computation (steps 0–3). This figure and the three that follow illustrate the DMA/DMS algorithm. Voxels are processed in decreasing order of distance map value (denoted by d in the individual captions). In (a), the first DMA voxels are processed (black with white lettering). More are identified in (b). In (c), the first three voxels whose exposures are below the threshold are processed (light gray with arrows denoting direction of steepest ascent); note in (d) how one of these becomes a bridge (dark gray with white lettering) to help connect the DMA.



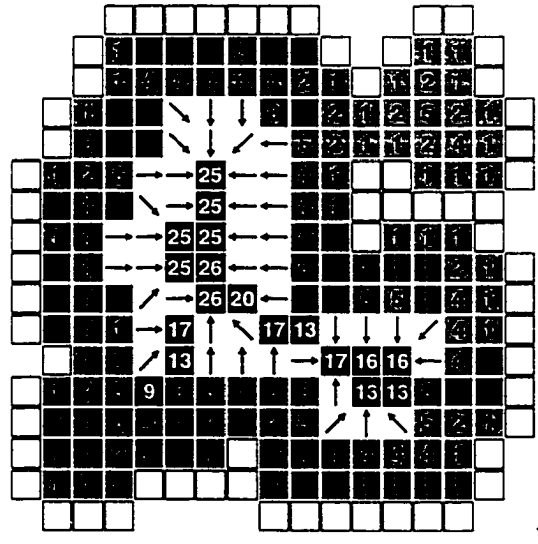
(a) Step 4: ($d = 16$).



(b) Step 5: ($d = 13$).

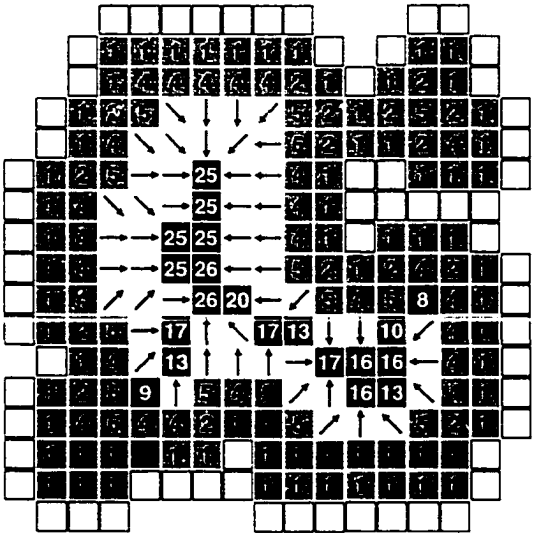


(c) Step 6: ($d = 10$).

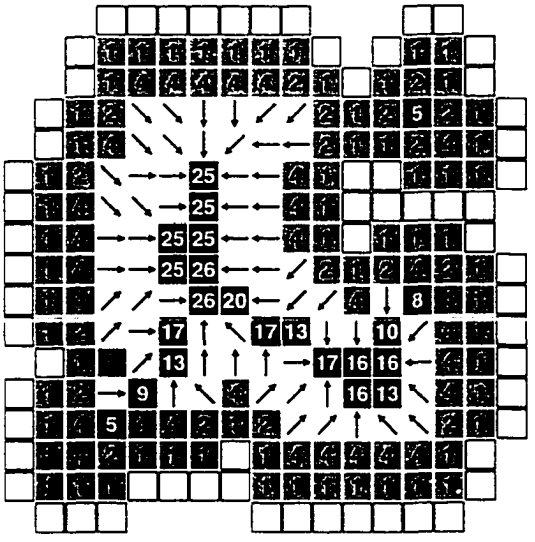


(d) Step 7: ($d = 9$).

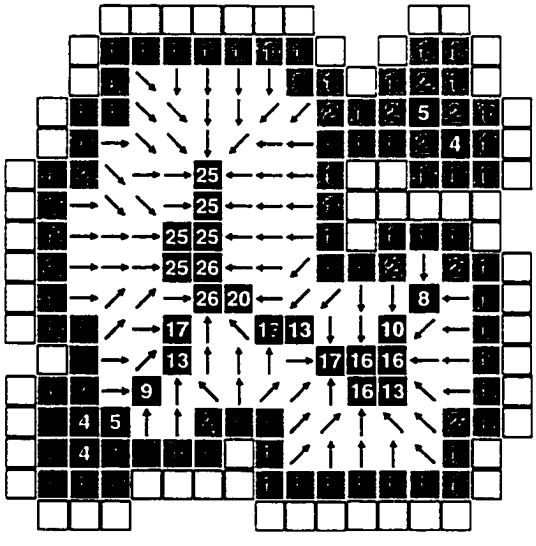
Figure 4.5: Discrete medial axis computation (steps 4–7). Voxels that have not yet been processed are shaded medium gray with distance map values drawn in black. Two separated regions of processed voxels are apparent in diagram (a) above – the genesis of the smaller region occurred in Figure 4.4(d). In (b), a "13" voxel becomes a special type of bridge, known as a saddle point, that effectively merges the two regions. No DMA voxels are added in (c). In (d), a bridging path composed of two voxels is formed to connect the "9" to the rest of the DMA.



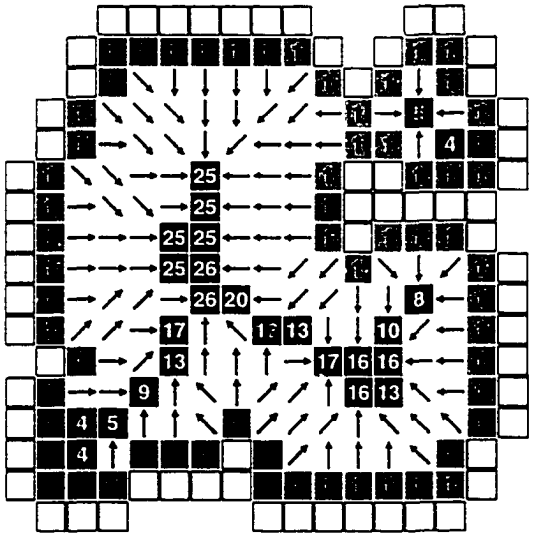
(a) Step 8: ($d = 8$).



(b) Step 9: ($d = 5$).

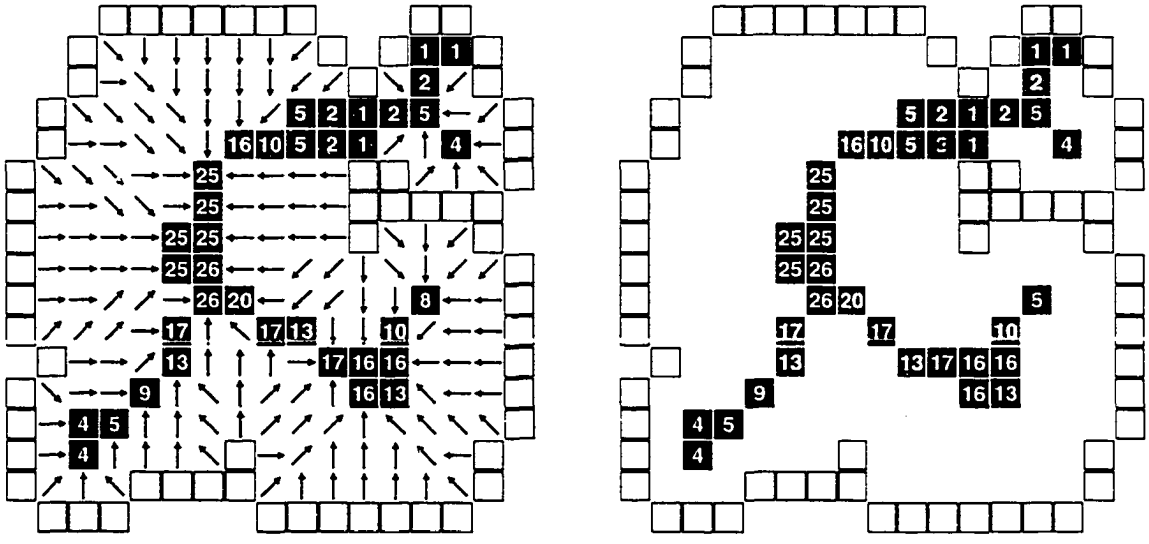


(c) Step 10: ($d = 4$).



(d) Step 11: ($d = 2$).

Figure 4.6: Discrete medial axis computation (steps 8–11). In the upper right of (b), another separated region of processed voxels forms. Both regions continue to grow in (c) and (d) and are finally merged in Figure 4.7(a) though the use of a saddle point and several bridging DMA voxels - note how each bridging path traverses its voxels along the path of steepest ascent. Actually, either "1" between the groups could act as the saddle point for the merge, but really, both should be considered saddle points.



(a) Step 12: ($d = 1$).

(b) Final DMA.

Figure 4.7: Discrete medial axis computation (step 12 and output). In (a), the final level of voxels has been processed. DMA voxels are marked as either black or dark gray with white lettering; the black voxels have exposures equal to or above the threshold (0.5), the dark gray voxels have insufficient exposures but act as bridges to help connect the DMA. Non-DMA voxels are drawn in light gray and shown with arrows representing the direction of steepest ascent. The output of the algorithm, shown in (b), is a grid in which the DMA voxels are marked as either true DMA voxels (black) or bridging DMA voxels (dark gray) and are labeled with their distance map values.

field for the number of processed neighbors, then, a voxel must also have a field that acts as a subindex into the appropriate contour array, noting the position of the voxel within that array, so that the transfer can be performed quickly. Note that in the example, if contour array #2 happened to be the current array being processed, then processing of the contour arrays would have to step back to process contour array #3 (since that array would contain the neighbor voxel) before returning to finish any processing of contour array #2.

As alluded to in the overview of the algorithm, voxels are processed differently depending on whether their exposure values are greater than or equal to the exposure threshold. When a voxel having a sufficient exposure value is processed, it is automatically marked as a DMA/DMS voxel. Following this, a bridging operation may occur to ensure that this new DMA/DMS voxel is connected to the other DMA/DMS voxels in the same region of processed voxels. When a voxel with an insufficient exposure value is processed, a check is performed to determine whether the voxel connects two or more regions of processed voxels, in which case the voxel is a saddle point. If it is a saddle point, then a bridging operation is performed for each adjacent region in order to connect the DMA/DMS for the conglomeration. Note that these bridging operations for saddle points can also occur when processing a voxel with sufficient exposure if that voxel is adjacent to more than one region of processed voxels.

To aid in the bridging process, each voxel maintains two additional pieces of information: a group identification number and a potential bridging direction. The group ID is used to keep track of connected regions of processed voxels; all processed voxels (whether DMA/DMS voxels or not) that form an 8-connected cluster (26-connected cluster for the 3D case) will have the same group ID. The bridging direction is used to determine bridging paths that will keep each portion of the DMA/DMS connected during formation; each separate group of processed voxels will have its own connected portion of the DMA/DMS.

Initially, all voxels are assigned a unique group ID and thus reside in their own one-voxel size group. During processing, groups are merged by applying a union-find algorithm to the set of group IDs (for details on the union-find algorithm, see Cormen, Leiserson, and Rivest [CLR90]).

Bridging directions are assigned to voxels as they are processed. In the current implementation, the bridging direction is the direction of steepest ascent (what is actually assigned to each voxel is a pointer to the neighboring voxel in the direction of steepest ascent). For the steepest ascent computation, neighboring voxels are handled in separate groups according to their adjacency to the current voxel being processed. In the 2D case, the 4-adjacent voxels are all one unit away from the current voxel, and the strictly 8-adjacent neighbors are all $\sqrt{2}$ units away, and this relative distance must be taken into account in the gradient calculation. Using the same symbols as in the formulation of the exposure calculation (see page 84), for the current voxel v_i, the amount of ascent, or slope, to a neighboring voxel v_n is computed as follows:

$$\text{ascent}(v_i : v_n) = \frac{\sqrt{d_n} - \sqrt{d_i}}{\text{distance}(v_i, v_n)}$$

The bridging direction corresponds to the neighboring voxel with maximum ascent value. As in the exposure computation, though, instead of calculating the ascent to each neighboring voxel, the 4-adjacent and 8-adjacent neighbors with the maximum distance map values can be found first to lessen the amount of computation involved.

When a voxel is processed, the neighboring voxels that have already been processed are examined to determine which groups are represented. Action is taken according to how many groups are represented, with the possibilities divided into the three cases from the following list:

- **No adjacent groups:** No adjacent groups translates into no adjacent voxels having been processed, so the current voxel simply keeps its own group ID. Unless the exposure threshold has been set artificially high (that is, greater

than one, in which case the algorithm is not guaranteed to work), the voxel will have sufficient exposure to be labeled as a DMA/DMS voxel.

- **One adjacent group:** If the processed neighbors all belong to the same group, then the current voxel joins that group (this is done by merging the group IDs of the current voxel and the adjacent group using the union-find algorithm). If the exposure value of the current voxel is below the threshold, then nothing else is done (this is the situation for each of the "20" voxels processed in Figure 4.4(c)). If the exposure is sufficient for the current voxel to be labeled as a DMA/DMS voxel, though, then an additional step is performed to ensure that the DMA/DMS for the group is connected in light of the fact that the current voxel is a new member of the DMA/DMS. This amounts to finding a bridging path from the current voxel to another DMA/DMS voxel for the group: any non-DMA/DMS voxels discovered along the bridging path are relabeled to be DMA/DMS voxels. The bridging path is composed of voxels found by starting at the current voxel and repeatedly moving to the steepest ascent neighbor until a DMA/DMS voxel is reached; often, the steepest ascent neighbor is already one of the DMA/DMS voxels, so no further action is required. In Figure 4.4(d), the processing of one of the "17" voxels requires a bridging path to be formed which causes a "20" voxel to be relabeled as part of DMA/DMS. In Figure 4.5(d), a longer bridging path results from the processing of a "9" voxel.

- **Multiple adjacent groups:** When the processed neighbors belong to more than one adjacent group, several things happen. First, regardless of its exposure value, the current voxel is labeled as a DMA/DMS voxel (it is a special type

of bridging voxel that corresponds to a saddle point). Next, bridging paths are found connecting the current voxel to the DMA/DMS of each adjacent group, effectively connecting the DMA/DMS for the conglomeration (note that to do this, it is necessary to compute the steepest ascent direction into each adjacent group). The final step is to merge each adjacent group with the current voxel's group through multiple applications of the union-find merging operation on the associated group IDs. In the execution illustrated in Figures 4.4 through 4.7, there are two instances of this merging via saddle points: the first occurs as a "13" voxel is processed in Figure 4.5(b), and the second occurs in the final step in Figure 4.7(a) when the large, central group is merged with the smaller group in the upper right.

The actions in the previous list focused on the bridging process, but remember that as any voxel is processed, additional steps are required to notify its unprocessed neighbors that they have yet another processed neighbor. Through careful coding, accessing the neighbors for the currently processed voxel can be minimized to total one access per neighbor plus one access per adjacent group (the latter access results from initiating any bridging operations that may need to be performed from the current voxel). Path traversals during bridging operations require accessing each voxel along the path in order to find its steepest ascent neighbor and to label it as a bridging voxel of the DMA/DMS.

There is a fair amount of bookkeeping that must be performed for the algorithm to work efficiently. For convenience, the following list provides a summary of the information stored with each voxel:

- **Distance Map Value:** Obviously, it is necessary to know the distance value for each voxel.

- **Exposure Value:** The exposure value of a voxel, the computation of which is described in Section 4.2, is compared against the exposure threshold to determine whether the voxel is a DMA/DMS voxel outright.

- **Array of Neighbors:** Each voxel possesses an array of pointers to voxels immediately adjacent to it. This array is organized according to adjacency relationships to facilitate the exposure and ascent calculations; for the 2D case, for instance, the four 4-adjacent neighbors precede the four 8-adjacent neighbors.

- **Steepest Ascent Neighbor:** This is a pointer to the neighboring voxel in the direction of steepest ascent, to be used during potential bridging operations.

- **Group ID Number:** This represents the current region of processed voxels in which the voxel resides. It is actually an index into the array that serves to maintain the group affiliations for the union-find algorithm.

- **Number of Processed Neighbors:** For an unprocessed voxel, this is the number of neighboring voxels that have already been processed. It also serves as an index indicating which contour array contains the voxel when the contour list containing this voxel is being processed. Once the voxel has been processed, this field is no longer updated.

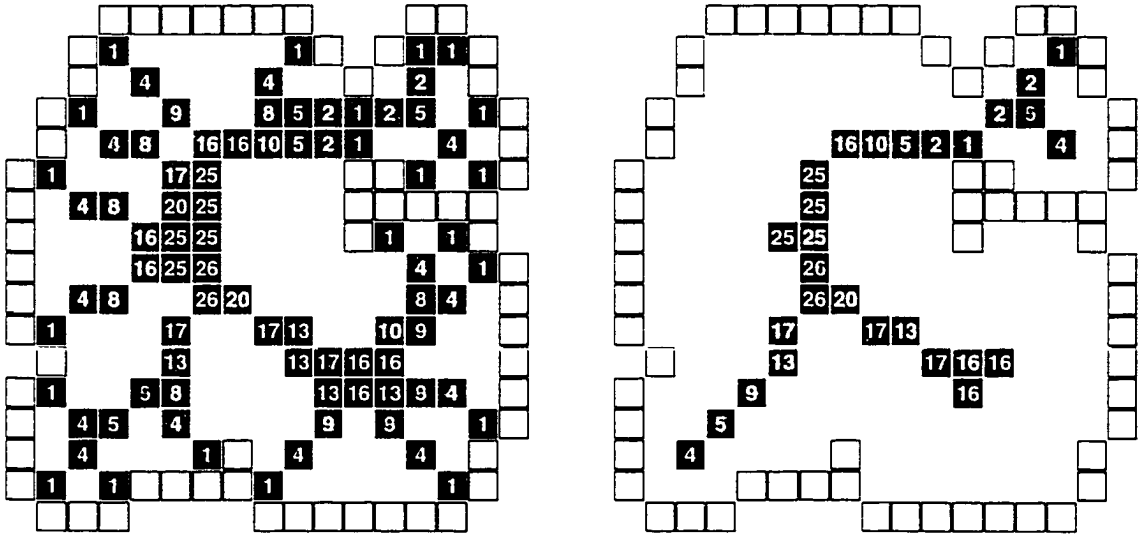
- **Contour Array Sub-index:** Whereas the number of processed neighbors tells which contour array the voxel is in during processing of its contour list, this field tells which position within that array is held by the voxel. Together, these two

fields enable the algorithm to shift the voxel from one contour array to another in constant time as the need arises. This particular field is only used during the processing of the contour list containing the voxel; it is initialized when the contour list is partitioned.

- **DMA/DMS Classification:** This field is used for two purposes: labeling the voxel as to whether it has been processed, and labeling it as to whether it belongs to the DMA/DMS. Before the voxel is processed, this field contains the value "unprocessed"; later, after processing, it may contain one of three values: "true DMA/DMS voxel", "bridging DMA/DMS voxel", "non-DMA/DMS voxel".

4.4 Results

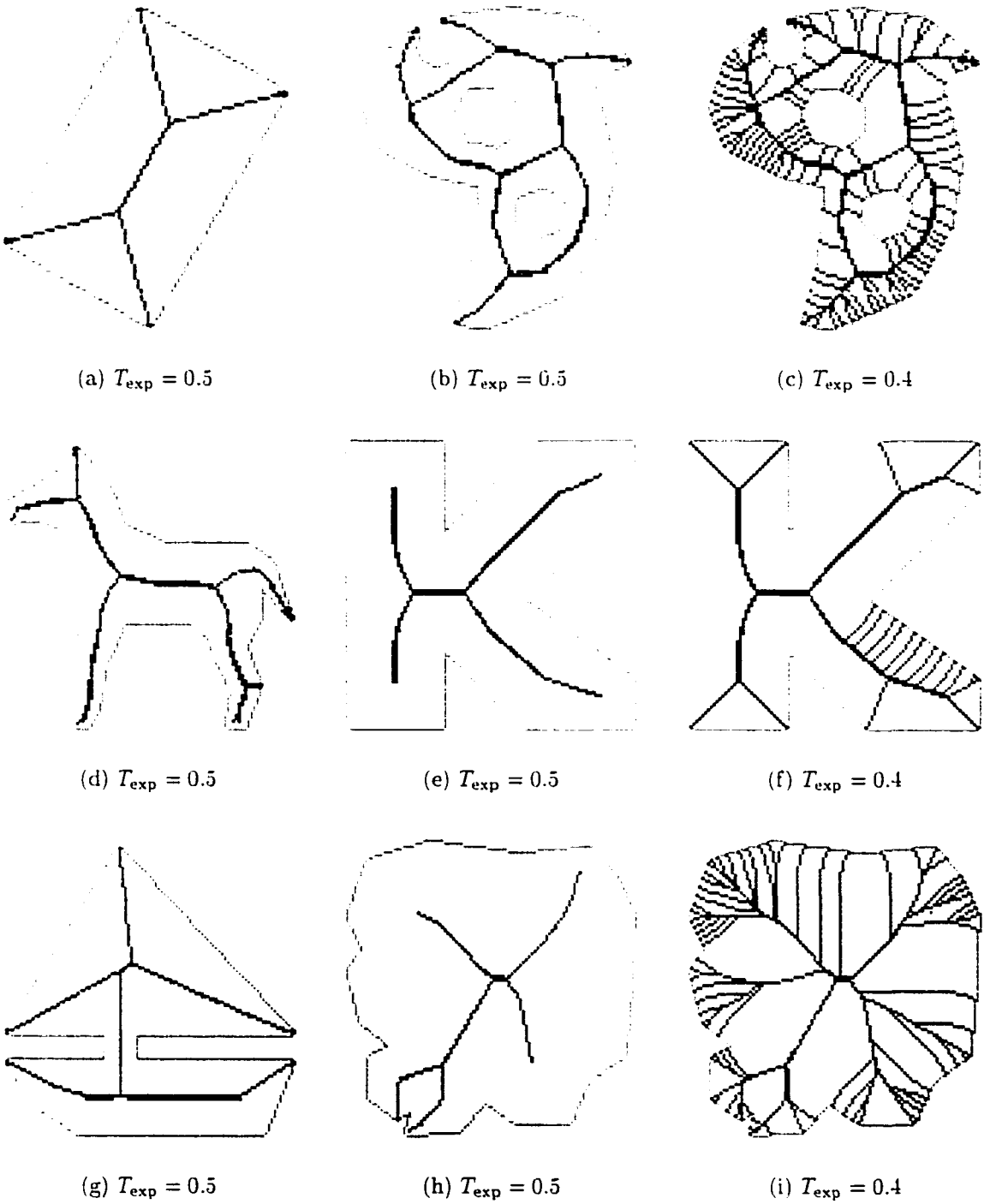
By specifying different values for the exposure threshold, various DMAs and DMSs can be produced, the difference being primarily in the level of detail the DMA/DMS shows. A low threshold such as 0.4 can extend the DMA into the finer protrusions of the boundary; a high threshold such as 1.0 is useful for generating a lean DMA for a concise analysis of the structure of the object's interior. Threshold values above 1.0 or below roughly 0.25 typically do not result in useful DMA/DMSs. Figure 4.8 shows the DMAs produced by the algorithm by using thresholds of 0.4 and 1.0 for the distance map used in Figures 4.4 through 4.7. Compare the results with Figure 4.7(b). For the purposes of this research, fine detail relating to each bump on the surface of an object is unnecessary, so the exposure threshold is usually set at either 0.5 or 1.0, depending on the object. More examples of DMAs produced by the algorithm are given in Figure 4.9.



(a) $T_{exp} = 0.4$ (b) $T_{exp} = 1.0$

Figure 4.8: The discrete medial axis that results from using other exposure thresholds (denoted by T_{exp}). Whereas Figure 4.7(b) shows the DMA computed for an exposure threshold of 0.5, the two grids above show the DMA computed for exposure thresholds of 0.4 and 1.0, respectively.

It is sometimes the case that the voxels with a distance value of one (also referred to as "1" voxels in the text) have relatively high exposure values when compared to voxels with other distance values. This can adversely affect the results of the algorithm by causing numerous spurious extensions of the DMA/DMS. For the 2D case, these extensions can often provide additional information for analysis of the object (for examples, see Figures 4.9(c) and 4.9(i)); nevertheless, they can often be filtered out by increasing the exposure threshold (contrast the examples just mentioned with Figures 4.9(b) and 4.9(h)). For the 3D case, the extra extensions can provide additional information, though in the vast majority of cases they simply clutter the DMS without providing anything of real use or significance.



(a) $T_{exp} = 0.5$ (b) $T_{exp} = 0.5$ (c) $T_{exp} = 0.4$

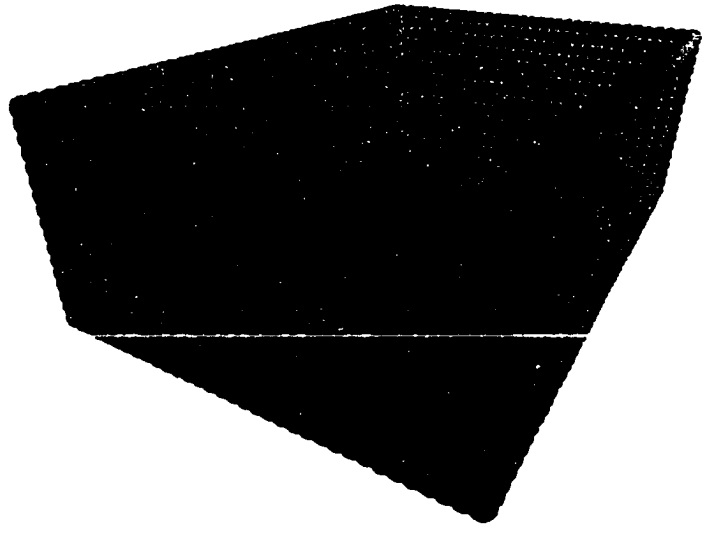
(d) $T_{exp} = 0.5$ (e) $T_{exp} = 0.5$ (f) $T_{exp} = 0.4$

(g) $T_{exp} = 0.5$ (h) $T_{exp} = 0.5$ (i) $T_{exp} = 0.4$

Figure 4.9: Several DMAs produced by the algorithm. The grid dimensions for each example are roughly 100×100, and exposure thresholds (denoted by T_{exp}) are shown. The shape in (b) and (c) is loosely based on one used by Ge and Fitzpatrick [GF96].

Note that these extensions may be technically accurate in the discrete realm, where the voxels functioning as feature points are perceived as individual elements devoid of any coherent relationships with other feature point voxels. Contrast this with the alternative case, where the voxelization is an approximation to a continuous object. Here, an outside observer would likely group the feature point voxels (which comprise the first layer of exterior voxels) into coherent sets based on a visual partitioning of the object's boundary; the observer would then expect the DMA/DMS for the object to respect this partitioning. Unfortunately, the discretization process usually conceals or discards any inherent partitioning of an object's boundary elements - and this problem manifests itself through an undesirable side effect: spurious extensions of the DMA/DMS that typically have no correspondence to the continuous medial surface of the object.

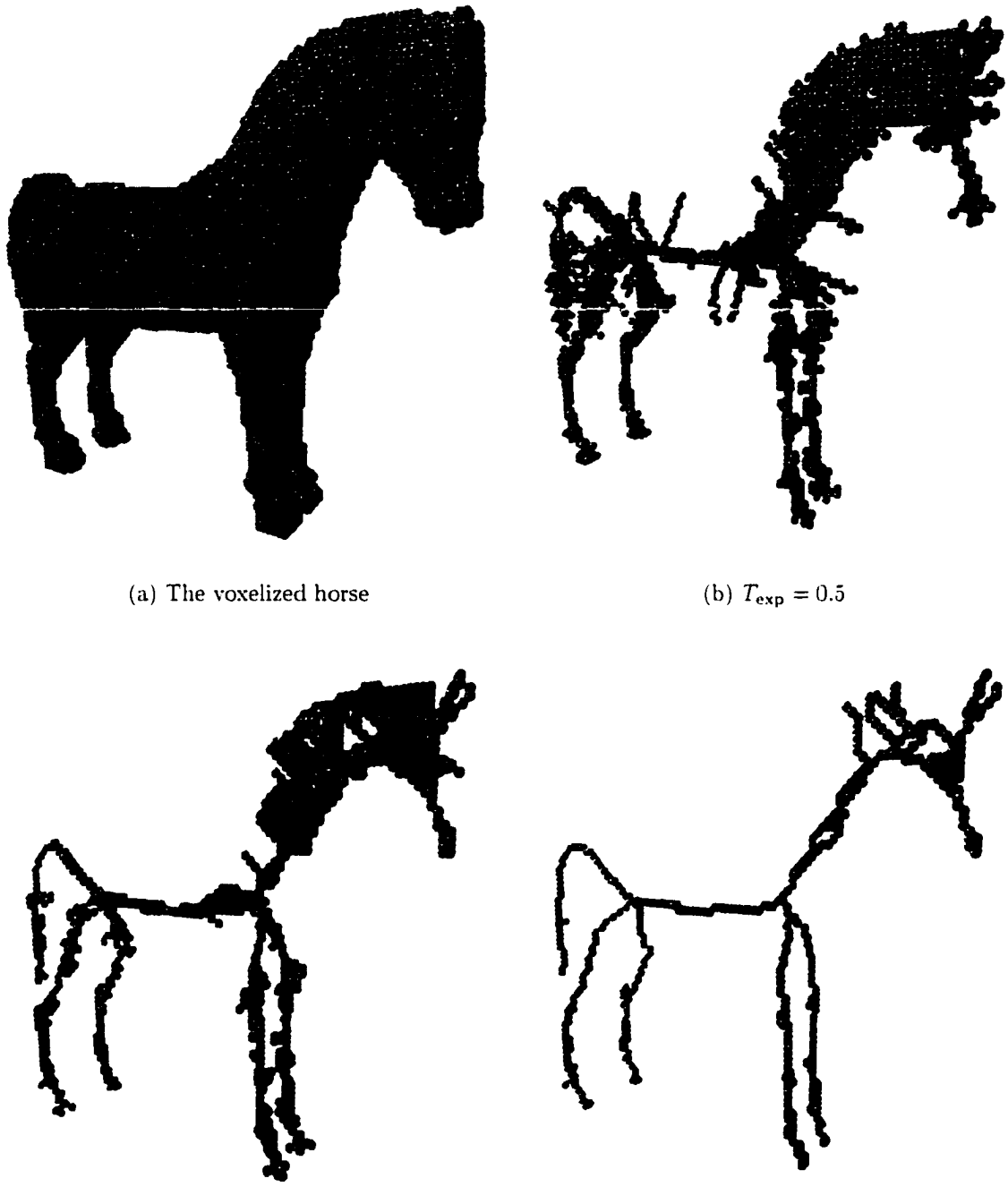
To help limit the growth of spurious branches of the DMA/DMS, it is often useful to ignore the "1" voxels to a certain degree. To do this, all "1" voxels are set to be non-DMA/DMS voxels, with the only exceptions being the "1" voxels that function as saddle points, which are labeled as DMA/DMS voxels. This helps to keep the DMA/DMS more clear and concise so that it better corresponds to the continuous medial axis/surface of the object. Figures 4.10 and 4.11 show examples of DMSs produced by the 3D implementation of the algorithm, in all cases handling the "1" voxels as a special case in the manner just described. In each figure, voxels are drawn as spheres with a radius of $\frac{\sqrt{3}}{2}$ times the width of a voxel so that a 26-connected "surface" of voxels, when rendered, will completely occlude anything on the side farther from the camera. For the 2D implementation, such special handling of "1" voxels is typically unnecessary in light of the influence had by simply raising the



(a) $T_{exp} = 0.4$

(b) $T_{exp} = 0.4$

Figure 4.10: Two DMSs of a box as produced by the algorithm. The box has the relative dimensions $2 \times 1 \times 3$. In (a), the box is aligned with the axes, resulting in a more regular voxelization and DMS. In (b), before being voxelized, the box was first rotated by 25 degrees about the z-axis and then by 20 degrees about the x-axis (for comparison with (a), the resulting DMS is shown with a similar viewpoint and lighting). In both instances, the voxelized box consisted of approximately 35,000 interior voxels, and the exposure threshold was set at 0.4.



(a) The voxelized horse

(b) $T_{\text{exp}} = 0.5$

(c) $T_{\text{exp}} = 0.7$

(d) $T_{\text{exp}} = 1.0$

Figure 4.11: DMSs of a voxelized horse as produced by the algorithm. The discretized horse model in (a) contains roughly 50,000 interior voxels. The other images show the DMS produced for the voxelized horse using various exposure thresholds (T_{exp}).

exposure threshold; nevertheless, keen observation of the shading used for DMA voxels in Figure 4.9 (black for true DMA voxels versus dark gray for bridging DMA voxels) will reveal which extensions of the DMAs would disappear if "1" voxels were basically ignored.

4.5 Analysis and Discussion

The time complexity analysis that follows relies on knowledge of the union-find algorithm and the counting sort algorithm. For more details on these algorithms, as well as for a discussion of $O(n \lg^* n)$ time complexity, see Cormen, Leiserson, and Rivest [CLR90].

If n is the number of interior voxels, then the time complexity of the DMA/DMS algorithm presented in this chapter is $O(n \lg^* n)$. Note that $\lg^* n$ is a function that grows extremely slowly: in fact, $\lg^* n \leq 5$ when $n \leq 2^{65536}$. Thus, for all practical purposes, the time complexity for the algorithm is as good as linear time complexity. The time complexity arises due to the use of the union-find algorithm, which is implemented using path compression and union-by-rank, and to the fact that the total number of disjoint-set operations in the algorithm is a constant multiple of the number of union operations. As for creating the sorted array of contour lists, it should be observed that the number of distance map values can be no larger than the number of interior voxels, so a linear time counting sort can be used to help order the array. Likewise, a counting sort can be used to partition the voxels of the contour list according to the number of processed neighbors each voxel has. As for the possible movement of a voxel from one contour array to another as its contour list is processed, observe that this cannot happen more than k times for any particular voxel, where k

2D (DMA) Implementation	
Interior Voxels	Execution Time
10,000	0.2 sec
100,000	3 sec
1,000,000	36 sec

3D (DMS) Implementation	
Interior Voxels	Execution Time
10,000	0.5 sec
100,000	6 sec
1,000,000	78 sec

Table 4.1: Approximate execution times for the 2D and 3D implementations of the DMA/DMS algorithm. The times shown are the average computation times over multiple test runs on various grids which had approximately the number of interior voxels listed in the table. These times were compared to the EDM approximation algorithm from Chapter 3, which was executed prior to the DMA/DMS algorithm in order to generate the distance map for input to the DMA/DMS algorithm. In general, the DMA implementation required roughly three times as much execution time as the EDM approximation algorithm, and the DMS implementation required about five or six times the execution time of the EDM approximation algorithm. Execution was performed on a Silicon Graphics® O2® (R5000 Processor Chip).

is the total number of adjacent voxels a voxel may have. For a particular dimension, this is a constant number ($k = 8$ for two dimensions, $k = 26$ for three dimensions, and so forth); thus, the number of transfers also has a linear time bound. Finally, note that the number of voxels traversed during all of the bridging operations combined cannot exceed the number of interior voxels; therefore, the extra processing required for bridging has a cumulative bound that is of linear time complexity.

Actual execution times for the 2D and 3D implementations are given in Table 4.1. The timings for the 3D version are roughly twice that of the timings for the 2D version. The only difference in the two implementations is in the number of possible adjacencies for a voxel, and this seems quite reasonable for explaining the relationship between the timings.

For the purposes of the research behind this dissertation, the DMA/DMS algorithm works quite well (its use will be made clear in the next chapter); nevertheless, there are a few problems with the algorithm as it pertains to producing quality DMAs or DMSs. As for judging the algorithm in terms of the preferable characteristics presented beginning on page 26 of Chapter 2, the algorithm performs nearly acceptably with a few noted shortcomings. The characteristics are reprinted in the list that follows along with a discussion of the performance of the algorithm with respect to each one.

- **Similar Topology:** For the 2D version, the DMA produced often has the same topology as the original object, such as in Figures 4.9(b) and 4.9(c). Such agreement is definitely not guaranteed, however, and this is especially evident in the extreme case of grids generated by randomly dropping feature points into a plane of background points. Even for the more usual case of working with discretized objects, it is not difficult to design objects where the genus of the DMA for the interior differs from that of the object. As for the 3D version of the algorithm, it is more often the case that the genera differ. In Figure 4.11, each DMS shown has a genus greater than zero (the genus of the voxelized horse), and in Figure 4.10, although the first DMS shown has genus zero like the box itself, pin-size holes are visible in the second DMS, indicating a non-zero genus. Precise determination of the genus of discrete objects turns out to be a fairly confusing undertaking, and the subject of discrete topology is beyond the scope of this text.

- **Centering:** For the most part, the DMAs and DMSs produced by the algorithm are well centered. Occasionally, however, bridging paths may be created that

are not as well centered as they could be, since they appear to diverge from where the continuous MA or MS might pass through the voxelization. This would seem to indicate that having the bridging paths follow in the direction of steepest ascent is not the optimal solution to the centering issue.

- **Exact Reconstruction:** If the exposure threshold is set low enough (0.4, for instance), and if "1" voxels are processed just like any other voxel, then the DMA or DMS produced seems like it could be used to reconstruct the original voxelization exactly via the inverse distance transform. No formal testing of this claim has been performed, however.

- **Rotational Invariance:** Here the algorithm often performs fairly well, though having a sufficiently large number of interior voxels helps to minimize any observable variance under rotation. Figure 4.10 is somewhat typical of the rotational results possible with the algorithm.

- **Immunity to Noise:** Due to the dependence of the algorithm on the exposure calculation, noise can affect the local structure of the DMA or DMS produced. Special handling of the "1" voxels as described earlier can aid the algorithm in better handling of surface noise.

- **Thinness:** Recall that thinness was a characteristic omitted from the original list but described afterwards. As for the performance of the algorithm with regard to this characteristic, note that the application of the exposure threshold is blind to the thickness of the DMA or DMS at any point. Nonetheless, the algorithm can produce reasonably thin results (Figure 4.10, for example), and using an exposure threshold of 1.0 can help (see Figures 4.8(b) and 4.11(d)).

The basic problem with the 3D implementation (and one that might worsen in implementations for higher dimensions) is that no special processing is performed in order to ensure that "surfaces" of voxels are being generated for the DMS. The bridging paths are all essentially 1D, but perhaps the voxels along the bridging paths could be made aware of neighboring bridging paths in some attempt to weave a bridging surface for parts of the DMS.

With all of its problems, then, one may ask what the main selling points for the algorithm are. The simple answers to the question are the algorithm's relative ease of implementation for any dimension and its efficient execution. If all that is needed in an application is a rough approximation to the medial axis or surface, or a reasonably well connected DMA or DMS, then this algorithm works quite well. As will be demonstrated in the chapters that follow, the DMA/DMS algorithm is entirely satisfactory for the research described in this document.

CHAPTER 5

AUTOMATED GENERATION OF CONTROL SKELETONS

This chapter details the steps involved in the general solution to the problem of automatic control skeleton generation for a given polygonal data model. The basic goals for generalized skeleton production are outlined in Section 5.1. Section 5.2 then steps through the discussion of each stage of the algorithm, from the voxelization of the model through construction of the discrete medial surface and on through to the creation of the control skeleton. Finally, Section 5.3 provides illustration and analysis of the results of applying this algorithm to several example models, and Section 5.4 concludes the discussion of the general solution.

Note that many of the steps of the algorithm are nearly identical to those in an earlier report on this research [WP00]. The main differences between the algorithm described in this chapter and the one reported on previously are that the one described here uses the discrete medial surface of the voxelized object, that it includes a step to smooth the path tree before creating the skeleton structure, and that it provides the user with slightly more control over the skeletonization process through the use of several input parameters. The results of this algorithm are slightly better than the results from the previous method.

5.1 Goals

The primary goal of the algorithm is the automatic construction of a skeleton for use in controlling the animation of a given set of polygonal data. To be effective, the control skeleton produced by the algorithm must correspond well with the input object. This correspondence should be present in the three basic respects: structure, articulation, and attachment. These three areas are further described in the respective sections of the outline of objectives below:

1. The skeleton and the object should agree in their basic shape and structure.
    - The skeleton should be centrally located with respect to the object's surface.
    - Major branches of the skeleton should match the major protrusions of the object, and minor branches of the skeleton should match the minor protrusions of the object.
2. The flexibility, or articulation ability, of the skeleton should be appropriate for the object. This means that the joints for the skeleton should reside at locations such that the skeleton exhibits the following qualities:
    - Skeletal segments should have meaningful lengths in relation to the nearby surface elements of the object.
    - The articulation of the skeleton in a particular region must seem appropriate with respect to the local topology of the object: in other words, joints should be placed at points where the object intuitively should be able to bend.

- The articulation at a particular joint must seem appropriate with respect to the local geometry of the object, meaning that the joint's axes should be aligned with the proximal and distal segments so as to allow easier specification of joint angles for animation.
- The skeleton should provide a sufficient but manageable level of control. There should be enough of a skeleton to provide some desired degree of control, yet there should not be so much of a skeleton that its manipulation would seem unwieldy.

3. The object should be attached to the skeleton in a sensible, straightforward fashion.
   - Each point of the object should be attached to one or more nearby segments of the skeleton.
   - The attachment should make the surface of the object appear flexible so that the surface is seen to bend gradually but in direct agreement with the bending of the skeletal segments.

Closely associated with the primary goal is the aim of requiring very little user input. Besides the polygonal data, the user can specify seven input parameters, though generally the algorithm performs fairly well using the default values of the parameters. The most influential parameters are the voxel-size parameter, the exposure threshold, and the closeness-of-fit parameter. The voxel-size parameter is simply the desired edge length of a voxel, the exposure threshold is the input parameter for the DMS calculation, and the closeness-of-fit parameter relates loosely to the extent of

skeletal branches. The various parameters will be discussed in more detail as they arise in the presentation of the algorithm.

5.2 The Algorithm

This section describes the various steps of the algorithm. Section 5.2.1 discusses the manner in which the given model is discretized, and Sections 5.2.2 and 5.2.3 tell how the distance map and discrete medial surface are computed for the discretized model. Section 5.2.4 describes how the medial surface approximation is used to generate a tree-like structure of voxel paths, which, as detailed in Section 5.2.5, is used to generate the segments and joints of the control skeleton. Also in that section, the method of attaching the original polygon model to the control skeleton is presented.

The geometric input to the algorithm is currently restricted to sets of polygonal data. The polygons are not required to form a single, closed surface, or really even to be connected at all. What is required is that after voxelization of the polygonal data and classification of each voxel as being either interior or exterior to the object, the interior voxels form a single, connected set. A closed polyhedron works quite well as input, but a figure consisting of overlapping closed polyhedra works equally well. The voxelization and classification process is often rather forgiving of aberrant polygons or of polygonal surfaces that are not closed.

5.2.1 Volumetric Discretization

For purposes of uniformity, the first step of the algorithm consists of transforming the polygonal data model so that its bounding box lies just inside the unit cube. After the transformation, the user is prompted to enter the edge length of a voxel (this is

the voxel-size parameter mentioned earlier), and the bounding box is then diced into a regular grid of small cubes (voxels).

After the voxel grid has been generated, the polygonal data is examined, and any voxels that are intersected by a polygon are marked as containing faces of the object. A filling routine is then applied to 6-connected regions of unmarked voxels in order to label each region as interior or exterior; a region is labeled as exterior if and only if that region includes voxels on the edge of the grid. Note that voxels containing faces are also considered to be interior voxels. After applying the filling routine, each voxel is labeled as either interior or exterior, and the grid is essentially a volumetric bitmap of the object.

For simplicity, it is required that the interior voxels form a single 26-connected group, though additional steps could be implemented to process disconnected groups and generate a separate control skeleton for each one. Note that the algorithm will work if the group of interior voxels contains holes; however, the control skeleton that is generated has the basic structure of a tree, and a tree-structured skeleton may not work well for animating an object such as a doughnut or any other shape that is not of genus zero.

The objective is to have a sufficient number of interior voxels. Having more interior voxels allows for a finer approximation to the shape of the model and thus equates to a better control skeleton, specifically with respect to the centralization of the segments and joints as well as to their relationship with the polygonal data. The trade-off, of course, is that more interior voxels require more memory and more processing time. Experiments have shown edge lengths of 0.005, 0.01, or 0.02 units to work fairly well, depending on the manner in which the transformed object fills the unit cube. For

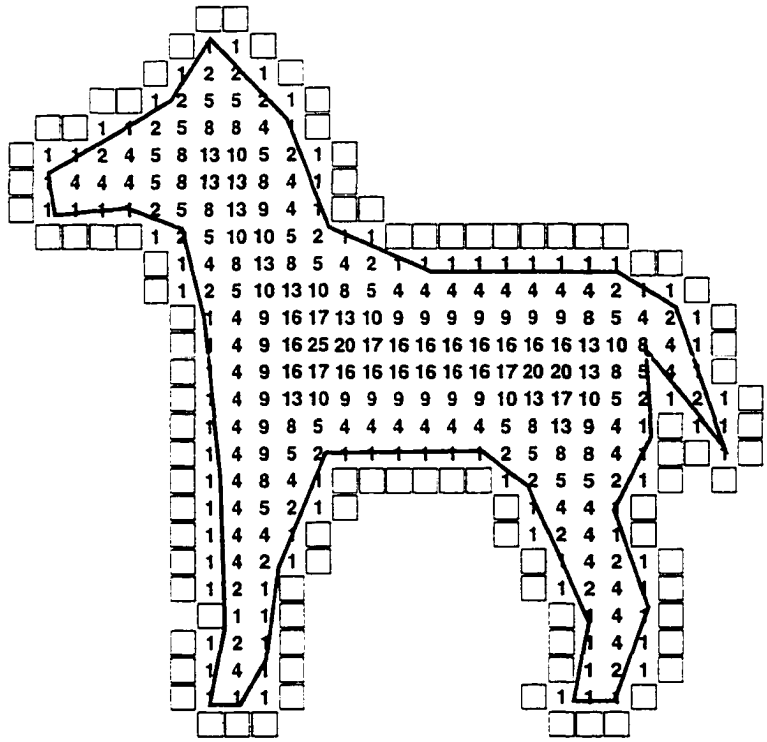


Figure 5.1: The 2D Euclidean distance map for a discretized, animal-shaped polygon. Cells intersected by the polygon or contained therein are the interior cells, shown as shaded squares; the first surrounding layer of exterior cells is shown using empty squares. The value in each interior cell is the square of the Euclidean distance to the nearest exterior cell.

most of the models used, the interior voxels account for about 10% to 40% of the total, and anywhere between 20,000 and 200,000 interior voxels are usually sufficient to produce a reasonable control skeleton.

5.2.2 Distance Map Computation

The next step of the algorithm is the generation of a Euclidean distance map (EDM) for the interior voxels. For each interior voxel, the square of the Euclidean distance from its center to that of the closest exterior voxel is computed. Figure 5.1

provides a 2D example of the EDM as computed for a discretized, animal-shaped polygon. The extension to three dimensions should be clear. Note that for the purposes of this research, the exact EDM is not necessary, so in actuality, a very close approximation is computed instead. For background on the distance map, see Section 2.1 of Chapter 2; for a detailed discussion of the algorithm used to compute the distance map, see Chapter 3.

5.2.3 Medial Surface Extraction

After the distance map has been computed, it is fed into the algorithm described in Chapter 4 for computing the discrete medial surface (DMS) of the object. The DMS algorithm flags those interior voxels that belong to the DMS of the object. It accepts one input parameter, the exposure threshold, which influences roughly how thick the DMS appears as well as to what degree it extends into each individual surface protrusion of the discretized object.

Generally, the skeleton generation algorithm works best if the DMS is relatively clean and simple, that is, if it has relatively few extensions other than those corresponding to major protrusions of the voxelization. For this reason, it is suggested that the exposure threshold be set somewhere in the range $[0.5, 1.0]$. The DMS algorithm is also set to ignore voxels whose distance map value is "1" unless they are needed to keep the DMS connected (see Section 4.4 of the previous chapter for a description of this special handling of the "1" voxels).

5.2.4 Path Tree Generation

After the DMS voxels have been identified, a path tree is generated that effectively simplifies the DMS to a tree structure of 1D pathways (referred to hereafter

as *chains*). The path tree is developed so as to maintain a tree structure regardless of the genus of the DMS or the object. The formation of the path tree begins by identifying a centrally located voxel referred to as the *heart*. A breadth-first search of the DMS is performed beginning at the heart in order to identify extreme points in the DMS – these extreme points are potential end-effectors of the control skeleton. The process of growing the path tree then begins, and each new branch of the path tree is created to extend to a previously unreached extreme point. During this process, the corresponding spheres for the path tree voxels are examined to see which DMS voxels are contained within them – any voxels contained within the spheres are said to be "covered" by the path tree. This coverage is used to help weed out insignificant extreme points resulting from spurious extensions of the DMS. When no more path tree branches can be added that are at least a certain length, path tree growth stops. Chains of the path tree are then identified, and the chain vertices are filtered to help smooth the otherwise jagged pathways resulting from stepwise movement between consecutive voxels along the chain. The following subsections describe these processes in more detail.

Identifying Extreme Points

Before extreme points are identified, the algorithm needs a point from which to label points as being extreme. For this reason, the concept of the heart was developed. As applied to the DMS, the heart is a DMS voxel that is centrally located with respect to the connectivity of the DMS as a whole.

One way to compute the heart is to perform repeated depth labelings of the DMS voxels, each time using a different DMS voxel as the origin of the depth labeling (all other DMS voxels are then labeled with their depth, or distance from the origin). After

each depth labeling, each DMS voxel adds its assigned depth value to an individual accumulator. Over the course of multiple depth labelings, DMS voxels that are more centralized overall will accumulate lower depth sums that those DMS voxels that are on the periphery of the DMS. After every DMS voxel has been the origin of a depth labeling, the DMS voxel with the minimum depth sum is the heart. Depending on the connectivity of the DMS, there may be multiple heart voxels (all having the minimum depth sum); for simplicity, however, the first voxel discovered to have the minimum depth sum is considered to be the one and only heart voxel for use in identifying extreme points.

The heart computation just mentioned requires a quadratic number of computation steps with respect to the number of DMS voxels. To avoid such a computational cost, a constant number of DMS voxels (say, 100 or so) can be randomly selected to be origins for depth labelings. Searching for the DMS voxel with the minimum depth sum then provides a reasonably close approximation to the heart.

After the heart voxel has been found, another depth labeling is performed on the DMS using the heart as an origin. The length of the path between the heart voxel and the deepest DMS voxel is saved for future use; this length is called the *heart radius*. Any DMS voxels whose depth values are local maxima are then tested to see whether they are still local maxima with respect to all DMS voxels within a slightly larger neighborhood (such as by comparing the depth of a local maximum against the depths of all DMS voxels within five adjacent voxels of the local maximum and discarding the local maximum if its depth is less than that of any DMS voxels within the neighborhood). The remaining local maxima are the extreme points of the DMS, and these are partitioned into groups according to their proximity within the

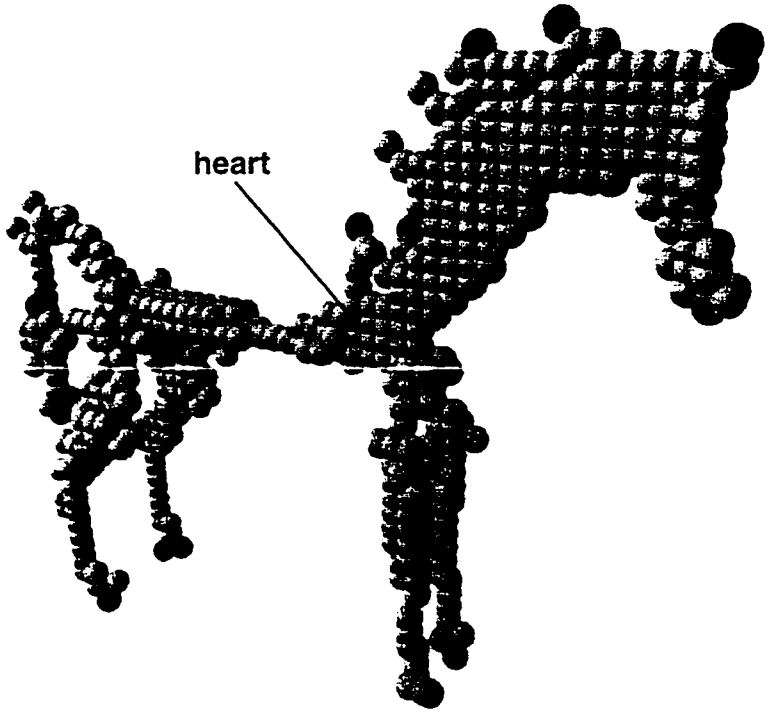


Figure 5.2: The heart and extreme points for a DMS of a horse. The heart voxel is labeled, and there are thirteen groups of extreme points (shown as black spheres) as seen from the heart - the hooves (4 groups), the tip of the tail (1), the haunches (2), the nose (1), the ears (1), and along the mane (4).

voxelization (another application of neighborhood searches). When the path tree is extended to an extreme point, all other extreme points of the same group are then ignored for future path tree extensions. Figure 5.2 shows the heart voxel and extreme points for a DMS of a horse.

Forming Path Tree Extensions

During the formation of the path tree, the algorithm examines connected paths of DMS voxels. Two measures of voxel paths are used during this process: the path length and a special weighted measure. The path length is simply the sum of

the distances between the centers of consecutive voxels along the path: the distance between the centers of two adjacent voxels is l, $l\sqrt{2}$, or $l\sqrt{3}$, where l is the edge length of a voxel. The weighted measure W_P of a voxel path P is based on the Euclidean distance map:

$$W_P = \sum_{v_i \in P} \frac{1}{d_i^3}$$

where d_i is the squared value for voxel v_i as stored in the distance map (the use of 3 as the exponent was arrived at empirically).

The purpose of the weighted measure is to provide a means for favoring centralized paths through the figure that follow along the deepest portions of the DMS. Using a modified version of Dijkstra's shortest paths algorithm (see Cormen, Leiserson, and Rivest [CLR90] for the standard version), the algorithm can find the voxel path through the DMS connecting any given pair of voxels and minimizing the weighted measure of all such connecting paths. Although minimizing the weighted measure does not guarantee that the path will follow along the deepest region of the DMS, experimental results have shown that it appears to do so.

Another concept crucial to the formation of the path tree is that of "coverage," which is related to the inverse distance transform. Each value in the Euclidean distance map defines a sphere, centered at the corresponding voxel, that just touches the boundary of the object. The radius of the sphere for a voxel v_i is $\sqrt{d_i}$, where d_i is the (squared) distance map value for v_i. Each sphere may contain the center points of other voxels; if so, a sphere is said to "cover" those voxels. When a new path is added to the path tree, any DMS voxels that lie within any of the corresponding spheres of the new path are marked as being covered by the path tree. Figure 5.3 shows a simple example of coverage.

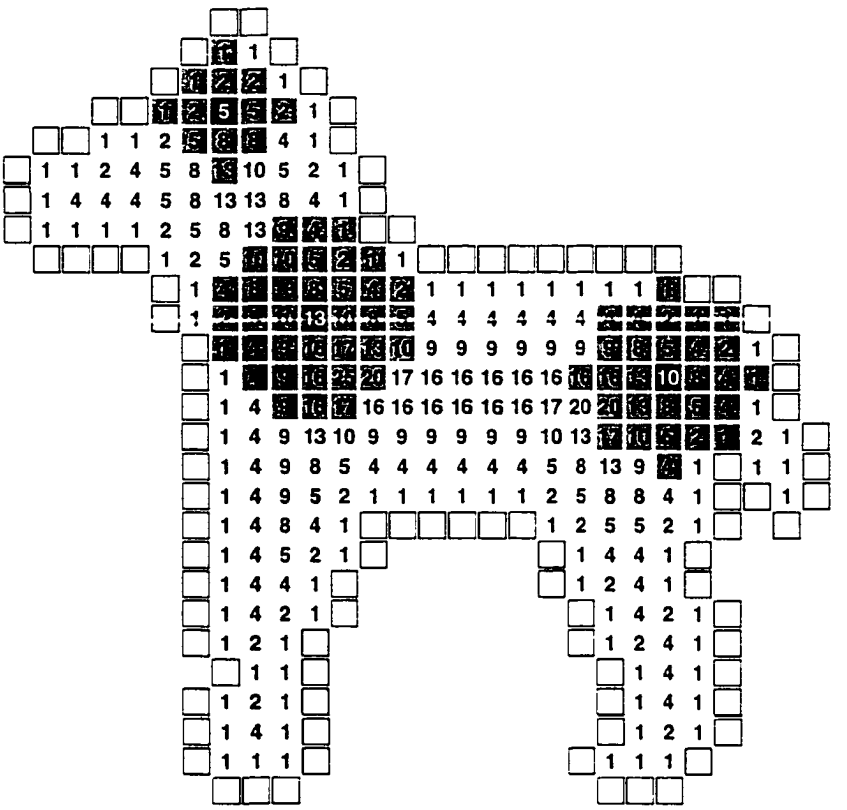


Figure 5.3: Examples of the coverage of three disks. The disks are centered at three black voxels and have radii equal to the square root of the respective distance map values in those voxels. Voxels shaded dark gray are contained in the disks and are considered to be "covered" by the black voxels (which are also considered to be covered themselves).

The extreme point with the largest depth value (relative to the heart voxel) is the starting point for the first branch of the path tree. The path tree is then grown by creating and appending extensions to it until further extensions to the path tree will unnecessarily complicate the structure. Each extension to the path tree is formed by executing the following steps:

1. Mark (or update) the DMS voxels covered by the path tree.

2. Find the extreme point DMS voxel v_f farthest from the covered region (note that the group of extreme points containing v_f must not already have had a branch of the path tree extended to one of its members, and also note that any covered extreme points are simply ignored).

3. Find the minimum weight path of DMS voxels connecting v_f to the path tree.

4. Append that minimum weight voxel path to the path tree.

In Step 1, note that the coverage of the DMS does not need to be recomputed each time a new branch is added to the path tree; instead, the coverage can simply be updated in the area surrounding the new extension. Figure 5.4 shows how coverage changes during the formation of path tree extensions within the DMS of the horse from Figure 5.2.

In step 2, the algorithm searches for the non-covered extreme point voxel v_f that is farthest from the set of covered DMS voxels. If the shortest path length from v_f to a covered DMS voxel is greater than or equal to a certain threshold, then the algorithm proceeds with steps 3 and 4 to extend the path tree to v_f and then repeats the process beginning with step 1. If the shortest path length from v_f is less than the threshold, then steps 3 and 4 are skipped, and no more branches are added to the path tree.

The threshold used in this process is the product of the user-supplied closeness-of-fit parameter (mentioned in Section 5.1) and the heart radius computed at the beginning of the path tree generation process. Observation has shown that a closeness-of-fit value between 0.05 and 0.1 works fairly well for producing a good, simple control skeleton - this means that a new branch will be added if it extends at least $\frac{1}{20}$ to $\frac{1}{10}$

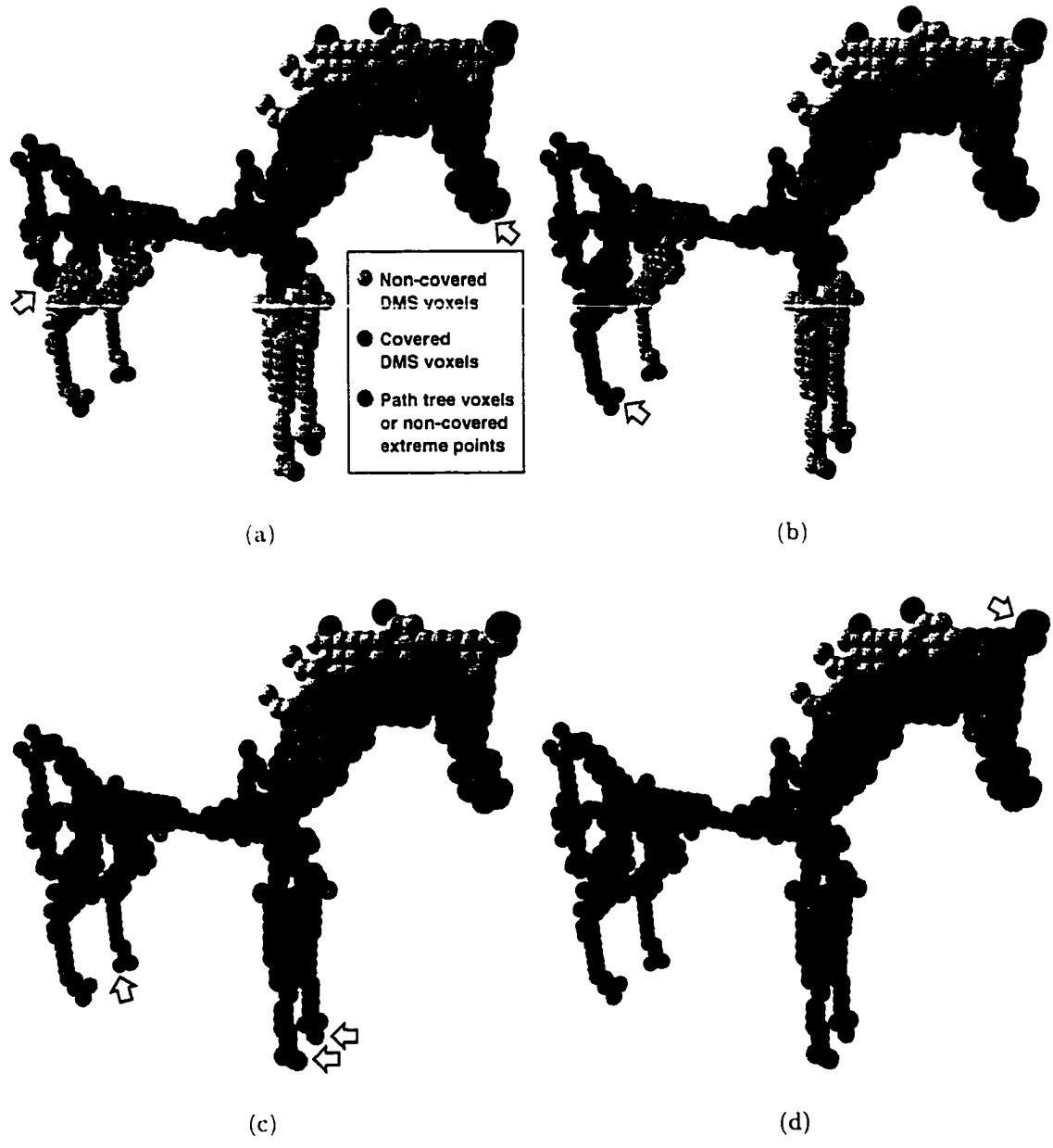


Figure 5.4: Forming path tree extensions. The extreme point at the tip of the tail is the first point of the path tree. In (a), the first extension to the path tree reaches from the tail to the nose. In (b), the second extension reaches to the right hind hoof. The next three extensions branch out to the other hooves as shown in (c). In (d), the final branch extends to the ears. The remaining extreme points are not far enough from the covered region to warrant further extensions of the path tree. The completed path tree is shown in Figure 5.5 without the other DMS voxels.

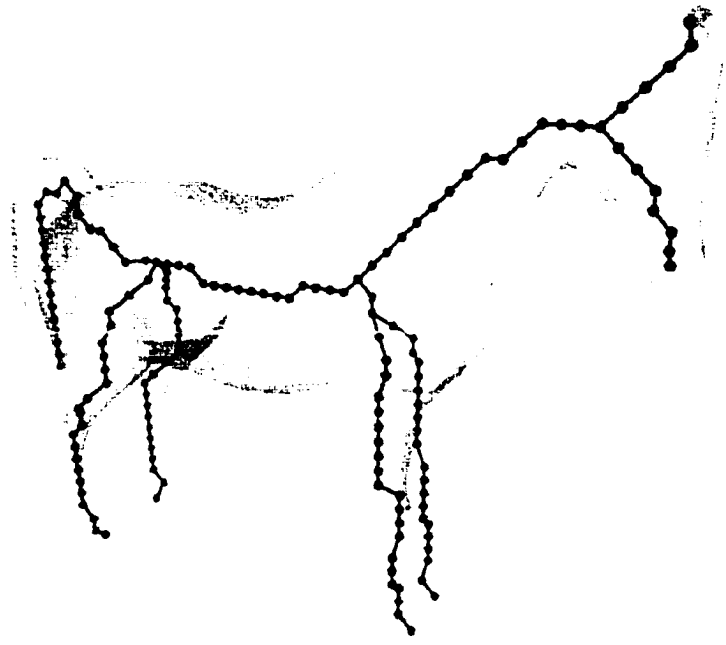


Figure 5.5: The completed path tree for the horse. The path tree voxels are drawn as small spheres to allow the edges of the path tree to be seen.

of the length of the heart radius beyond the current coverage of the path tree. Using finer values will usually allow the extension of the path tree into smaller protrusions of the object, such as the fingers of a hand; however, it can also result in the formation of other seemingly spurious branches.

Step 3 makes use of the modified version of Dijkstra's shortest paths algorithm mentioned previously. Each path tree voxel is assigned a weight of zero and becomes a source point for the shortest paths. The weighted measure is applied as the shortest paths search spreads through the DMS. When the search reaches v_f, it is a simple matter to backtrack to find the actual minimum weight path from v_f to the path tree. This minimum weight path is then added to the path tree. Its coverage of the DMS

voxels is then computed as the process of extending the path tree is repeated from step 1.

Smoothing the Path Tree

The path tree is basically a collection of vertices (the centers of the path tree voxels) connected by a set of edges (based on the adjacency of consecutive voxels of the path tree extensions). The path tree for the horse has been redrawn as a collection of vertices and edges in Figure 5.5. After the path tree has been formed, its vertices can be sorted into three classes. Endpoint vertices have only one adjacent edge - these correspond to the extreme points used during the growth of the path tree. Junction vertices have three or more adjacent edges - this is where the path tree forks or branches. The remaining vertices, termed intermediate vertices, have exactly two adjacent edges and come in connected sequences between endpoint and/or junction vertices. The endpoint and junction vertices split the path tree into a set of connected path segments termed chains.

Due to the regularity of the voxelization, the chains of the path tree can be fairly jagged. The jaggedness may be especially noticeable in parts of the figure where the main direction of a chain section does not align reasonably well with any of the axes of the voxelization. To lessen any peculiar effects the orientation of the voxelization can have on the path tree, and also to diminish the influence of the jaggedness on the later creation of segments and joints, the path tree is subjected to a smoothing operation.

As each chain of the path tree is identified, it is smoothed by applying a filtering process to average positions of consecutive voxels along the chain. A filter radius of three edges usually works well to smooth out any jaggedness of the original chain.

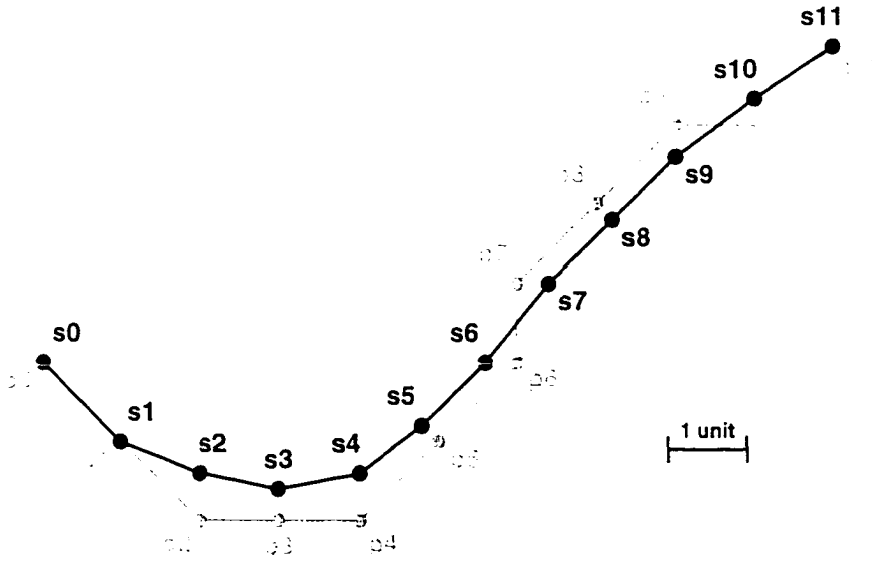


Figure 5.6: Smoothing of a path tree chain. A path tree chain is smoothed by applying a filter around each vertex of the chain. For the chain **p0..p11** above (shown in light gray), the smoothed chain **s0..s11** (shown in black) was computed using essentially a box filter with a two edge radius. As an example, the position of **s5** is the average of all chain vertices within two edges of **p5**; thus, $s5 = \frac{p3+p4+p5+p6+p7}{5}$. The radius is limited at the ends of the chain: for instance, $s1 = \frac{p0+p1+p2}{3}$, and $s0 = p0$.

The smoothing process is illustrated in Figure 5.6 using a filter radius of two edges. Figure 5.7 shows the result of smoothing the path tree for the horse.

5.2.5 Control Skeleton Construction

The path tree itself is usually too complicated to use directly as the structure of the control skeleton: instead, an approximation to the path tree is formed in what is called the skeletal graph (which is really a tree, since it approximates the path tree). The skeletal graph is the precursor to the final control skeleton structure. It is created so as to approximate the path tree using appropriately sized edges, each of which will become a segment of the control skeleton. The discussion that follows

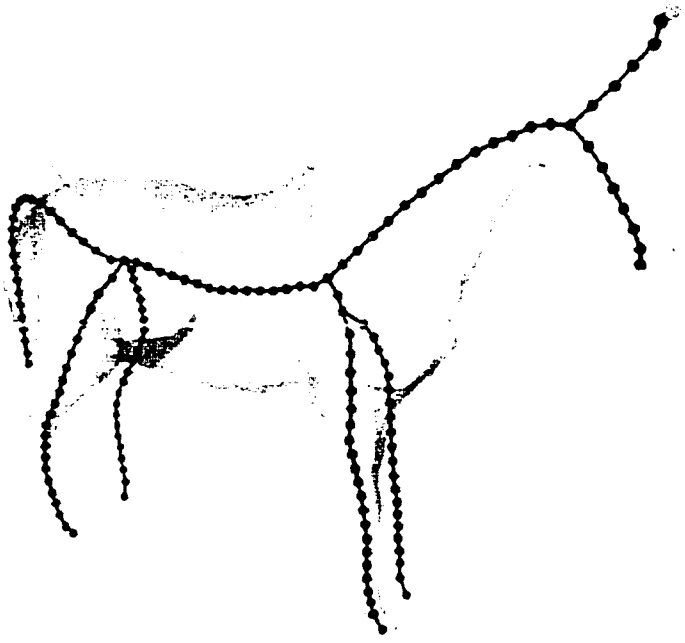


Figure 5.7: The smoothed path tree for the horse. This is the result of applying the smoothing operation to the chains of the path tree from Figure 5.5.

explains how the skeletal graph is constructed and how it is then used in the creation of the segments and joints of the control skeleton. The final part of the discussion reveals how the coverage of the path tree can be used in determining how vertices of the polygonal model are to be attached to the skeleton structure.

Creating the Skeletal Graph

The initialization of the skeletal graph results from a simple conversion of path tree chains. The endpoint and junction voxels of the path tree are used to create the initial vertices of the skeletal graph. Each chain of the path tree is used to create an initial edge of the skeletal graph.

After the initial edges and vertices of the skeletal graph are formed, tests are performed to determine which edges should be split. Splitting of a skeletal edge is accomplished by inserting an intermediate vertex into the skeletal graph (at the location of a specially selected chain vertex from the corresponding path tree chain) and replacing the edge with two new edges. Each new edge then corresponds to a subsection of the original chain. The edges for any particular chain may be split repeatedly in order to form closer approximations to the chain or in order to have more appropriate lengths.

Two input parameters are used to specify a range of desired edge lengths (the specified range is actually applied not to the skeletal edges but to their corresponding section of a path tree chain). The parameters, called min-fraction and max-fraction, are entered as values between zero and one (default values are 0.1 and 0.3, respectively). The lower limit of the range is the product of min-fraction and the heart radius; the upper limit of the range is the product of max-fraction and the heart radius. Any skeletal edges whose chains are already shorter that the lower limit will not be split. Any skeletal edges whose chains are longer than the upper limit will definitely be split. Edges whose chain lengths are within the range may be split based upon how closely they approximate the corresponding chain section.

Skeletal edges are assigned error values according to how closely they approximate the corresponding chain section of the smoothed path tree. This error is simply the maximum distance between the skeletal edge and one of the vertices of its related chain section. Figure 5.8 provides an illustration of the error computation and the splitting of a skeletal edge.

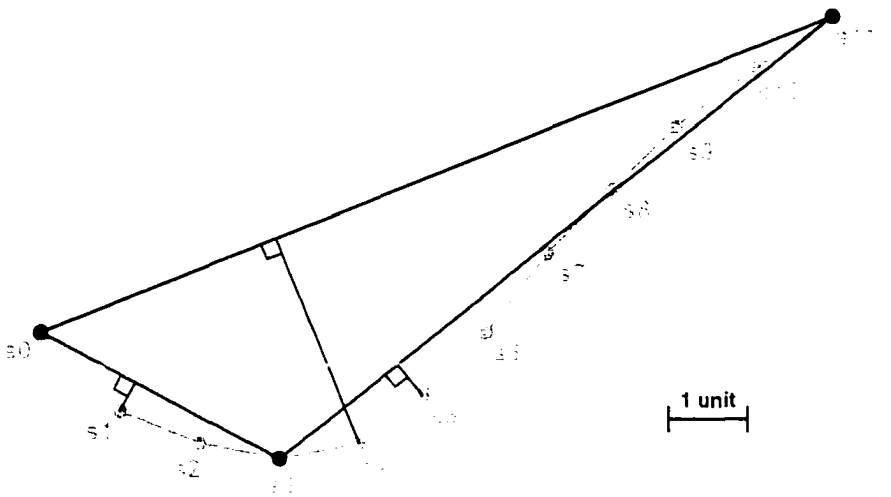


Figure 5.8: Error and splitting of a skeletal graph edge. The error for a skeletal graph edge is the maximum distance between the edge and any of the vertices of its corresponding smoothed chain. The error for the skeletal edge **s0–s11** is the length of the perpendicular segment to **s4**, which is 2.79 units. The best split for the skeletal edge is obtained by inserting a skeletal vertex at **s3**, as this results in the smallest maximum error (0.50) for the two replacement edges. The replacement edges for this example are **s0–s3** (error = 0.41 units) and **s3–s11** (error = 0.50 units).

The splitting of skeletal edges is performed incrementally: at each step, the entire skeletal graph is compared to the entire path tree to determine which edge should be split next. In this way, the skeletal graph gradually becomes more complex while providing an acceptable approximation to the path tree at any stage of the splitting process. The reason for this global approach is to provide the best approximation given the constraint imposed by the number-of-segments parameter (each skeletal edge corresponds to one segment of the control skeleton). The processing is accomplished using a heap whose node weights are the error values of the skeletal edges. Any edges whose chains are longer than the lower limit are placed into the heap. The edge with the largest error is removed from the heap and processed. If it can

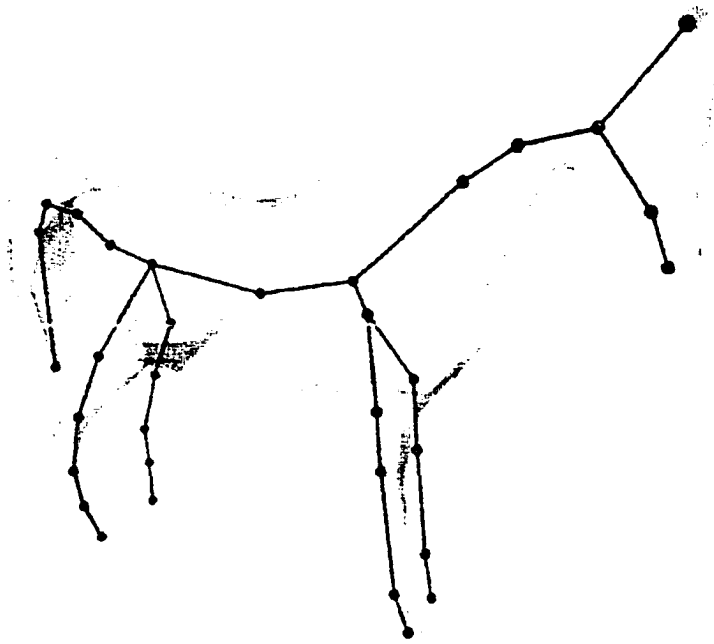


Figure 5.9: The skeletal graph for the horse. Each edge of the skeletal graph is used to create a segment for the control skeleton. Joints of the control skeleton are created at interior vertices of the skeletal graph, but note that more than one joint may be created at a vertex depending on how many edges are incident to that vertex.

be split and if its error is larger than what is acceptable (that is, if its error is larger than the approximation-error parameter), then errors for the two replacement edges are computed, and those edges are inserted into the heap. If the number-of-segments parameter has been set, then the splitting process is repeated until the skeletal graph contains an equivalent number of edges (or until the heap is empty and there are no more edges to be split). If the number-of-segments parameter is left unspecified by the user, then splitting stops when the heap is empty. Figure 5.9 shows the skeletal graph computed by allowing the heap to empty as the smoothed path tree for the horse is processed.

Creating Segments and Joints

After the skeletal graph has been formed, creating the segments of the control skeleton is rather simple. Each edge of the skeletal graph essentially becomes a segment of the control skeleton. A deep segment is then selected to host the root joint for the control skeleton, or rather, to be the only segment connected to the root joint. The root joint itself is positioned at the midpoint of the skeletal edge for that segment (note that it does not divide the segment).

The location of the root joint imposes proximity relationships on the skeletal graph edges and thus on the control segments. Each pair of adjacent segments has either a proximal-distal relationship or a sibling relationship with respect to their proximity to the root joint. For each proximal-distal pair, a joint is created at the shared joint-voxel (note that this results in coincident joints at the branching points of the tree structure, with the coincident joints numbering one less than the number of segments meeting at the branching point). Each joint other than the root joint thus has one proximal and one distal segment; the root joint has only a distal segment. Vertices of the skeletal graph that are not used for joint creation become end-effector points of the control skeleton.

Each joint has three rotational degrees of freedom. Joint axes are determined automatically to align with the proximal and distal segments. They form an orthonormal set of vectors defined as follows: the z-axis points outward along the distal segment, the x-axis is formed to be perpendicular to the plane defined by the proximal and distal segments, and the y-axis is then formed to complete a right-handed coordinate system. In the event that the plane used to create the x-axis is not uniquely defined, the algorithm searches proximally to find the closest ancestral segment that can be

used to help uniquely define a plane (a distal search is performed if the proximal search fails). The first rotational degree of freedom is about the joint's x-axis, the second is about the joint's y-axis, and the third is about the joint's z-axis.

Anchoring Skin Vertices to the Control Skeleton

Once the control segments and joints have been assembled, the algorithm turns to the process of anchoring the vertices of the polygonal data to the control segments. To do this, the algorithm returns to examine the spheres of the path tree voxels. Each control segment corresponds to a section of a chain of the path tree, and each voxel along that section of the path tree has a sphere which covers voxels of the figure (see Section 5.2.4 for an explanation of coverage). The collected coverage for a particular section of the path tree essentially defines a volume within which the corresponding control segment exerts influence. In the actual implementation, each voxel gathers and maintains a set of pointers to those control segments that cover (or influence) it.

Not all voxels interior to the figure are necessarily covered by spheres of path tree voxels. A voxel that is not covered by some path tree voxel will not at first have an influencing set of control segments; instead, such a set must be created. The sets for such voxels are constructed by propagating sets from covered voxels into non-covered regions of the voxelization. This propagation is performed in a breadth-first manner moving away from the covered region.

When each voxel has a list of those control segments exerting influence over it, it becomes a simple matter to anchor the vertices. The voxel that contains a specific vertex provides the list of control segments that influence that vertex. The coordinates of that vertex can then be expressed using the local coordinate frame for each influencing segment. When the control skeleton is moved, the (fixed) local definitions

of that vertex are converted into global positions and used in a weighted sum formula that computes a new global position for the vertex. The following paragraph describes this process in detail.

Let S be the set of control segments that influence a specific vertex v, and let s_i be the ith control segment influencing that vertex. Let $origin_i$ be the origin of the local frame of s_i, and let $RtoWorld_i$ be a 3×3 rotation matrix used to help transform a point from the local basis of s_i to the global basis of the figure. If the local position of v in the frame of s_i is denoted by p_i, then the global coordinates of v, according to s_i, is given by the expression $origin_i + p_i \times RtoWorld_i$. If only one segment influences v, then that expression suffices; however, if S contains multiple segments, then the expressions are combined using a weighted sum:

$$p_v = \sum_{s_i \in S} w_i \times (origin_i + p_i \times RtoWorld_i)$$

Here, p_v is the global position of v resulting from the combination, and w_i is the weight (or amount of influence) that segment s_i exerts on v. Note that the symbol "$\times$" used above denotes either simple multiplication or matrix multiplication and not the vector cross product. If only one segment influences v, then the weight w_i in the formula above is set to one. When multiple segments influence v, then the weights w_i used in the formula must sum to one. In this case, the weights are computed as follows:

$$w_i = \frac{totaldist - dist_i}{(n-1) \times totaldist}$$

where n is the number of segments in S, $dist_i$ is the shortest distance between v and segment s_i, and $totaldist = \sum_{s_i \in S} dist_i$. Closer segments have larger weights. Note that w_i, p_i, $dist_i$, and $totaldist$ as mentioned above are constants – these values

are computed once using the original positions of the polygonal data and the control skeleton structure. Values that are updated each time the control skeleton is repositioned include $origin_i$, $RtoWorld_i$, and, of course, p_v.

5.3 Results

In general, the algorithm is quite effective in producing a useful control skeleton in a short period of time. The quality of individual results is highly dependent on the object and the input parameters. In some cases, the algorithm performs extremely well, but in some other cases, the algorithm does only a mediocre job. For most objects that an animator might wish to animate by using a control skeleton, the skeleton produced by the algorithm is at least a reasonable start worthwhile for finer hand-editing.

Table 5.1 shows the results of several executions of the algorithm on various polygonal models (the models themselves can be seen in Figures 5.10, 5.12, and 5.13). Note that each model has been scaled to fit inside the unit cube; thus, a grid size of 0.01 will allow approximately 100 voxels along the edge of the unit cube. The graph in Figure 5.11 illustrates the unproven but apparently superlinear time complexity of the algorithm (superlinear with respect to the number of interior voxels). The data in the figure can be approximated fairly well by the function

$$f(x) = \frac{x^{1.2}}{54,000}$$

where x is the number of interior voxels and $f(x)$ is the number of seconds required to create the control skeleton.

With respect to the goals described at the beginning of Section 5.1, the algorithm performs reasonably well. As can be seen in the figures, the control skeletons are

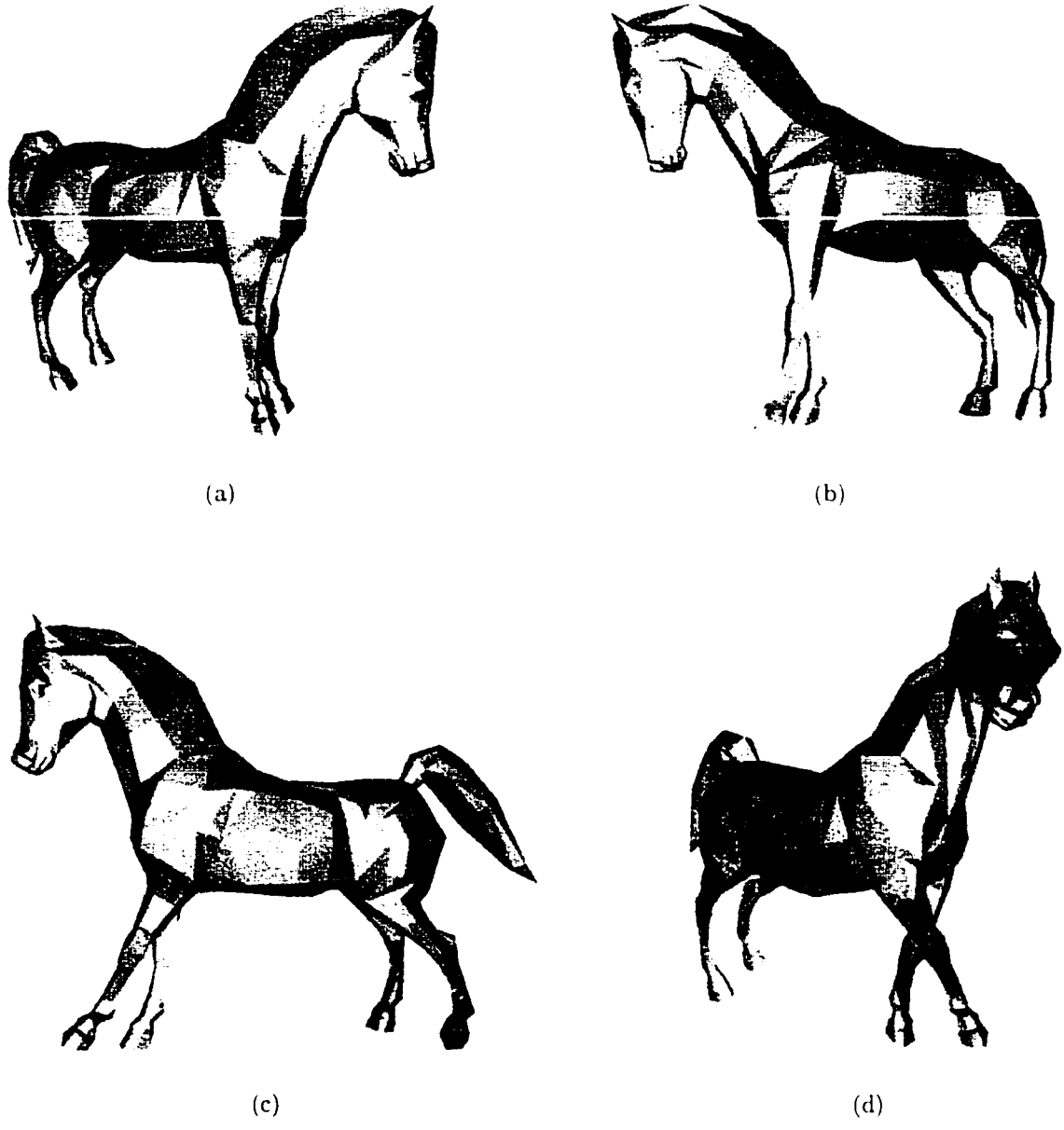


Figure 5.10: The horse and a few random poses. In (a), the horse is shown in its default pose (as input for the algorithm). The other three images are selected random poses of the horse using the skeleton from Figure 5.9.

Voxel Size	Number of Segments	Grid Dimensions	Total Voxels	Interior Voxels	Time (min:sec)
HORSE (681 vertices, 1,354 polygons)					
0.02	**32**	**51 × 42 × 15**	**32,130**	**7,324**	**0:02**
0.01	36	101 × 84 × 29	246,036	48,626	0:11
0.005	31	201 × 168 × 58	1,958,544	352,971	1:29
HUMAN (349 vertices, 694 polygons)					
0.01	29	37 × 101 × 17	63,529	14,044	0:04
0.005	**27**	**73 × 201 × 33**	**484,209**	**92,934**	**0:24**
0.0025	31	145 × 401 × 65	3,779,425	675,120	2:58
OCTOPUS (2,347 vertices, 4,690 polygons)					
0.01	52	101 × 59 × 78	464,802	42,772	0:11
0.008	**57**	**126 × 74 × 97**	**904,428**	**79,926**	**0:21**
0.00675	51	149 × 88 × 115	1,507,880	129,474	0:34
JELLYFISH (2,526 vertices, 5,048 polygons)					
0.008	**102**	**119 × 126 × 110**	**1,649,340**	**157,150**	**1:04**
0.007	103	136 × 143 × 125	2,431,000	227,784	1:36

Table 5.1: Execution results for the skeletonization algorithm on a horse, a human, an octopus, and a jellyfish. Each row corresponds to a single execution of the algorithm for which the voxel-size parameter was specified as in the first column. The number of control segments was determined automatically by the program, and all other input parameters used default values (see Table 5.2). The boldface rows correspond to the set-ups used for Figures 5.9, 5.12, and 5.13 (with a minor change in the case of the skeletonization of the human in Figure 5.12, where an exposure threshold of 1.0 was used instead of the default value of 0.5 used in the table). The column at the far right contains the amount of time required (minutes and seconds) to execute all stages of the algorithm (from figure voxelization through anchoring the vertices) on a Silicon Graphics® O2® (R5000 Processor Chip). Many of the test runs were also performed on a PC with a 133 MHz Intel® Pentium® processor running under Linux®, and those execution times were quite similar, typically 100% to 105% of the execution time on the O2.

Input Parameter	Valid Range	Default Value	Description
voxel-size	$(0..\infty)$	0.02	The edge length of a voxel, generally specified relative to the unit-cube.
exposure threshold	$[0..1]$	0.5	The parameter for the DMS calculation that relates somewhat to the thickness of the DMS.
closeness-of-fit	$[0..1]$	0.1	When multiplied by the heart radius, this provides a minimum acceptable length for new extensions to the path tree.
approximation-error	$[0..\infty]$	0.4	The error tolerance for approximation of path tree chains by skeletal graph edges.
min-fraction	$[0..1]$	0.1	When multiplied by the heart radius, this provides the minimum desired length of a path tree chain for a corresponding skeletal edge.
max-fraction	$[0..1]$	0.3	When multiplied by the heart radius, this provides the maximum desired length of a path tree chain for a corresponding skeletal edge.
number-of-segments	$[1, 2, ..\infty)$	*unspecified*	The maximum number of control segments the skeleton may have. If left unspecified, the program determines the number of segments automatically.

Table 5.2: Input parameters for the skeletonization algorithm.

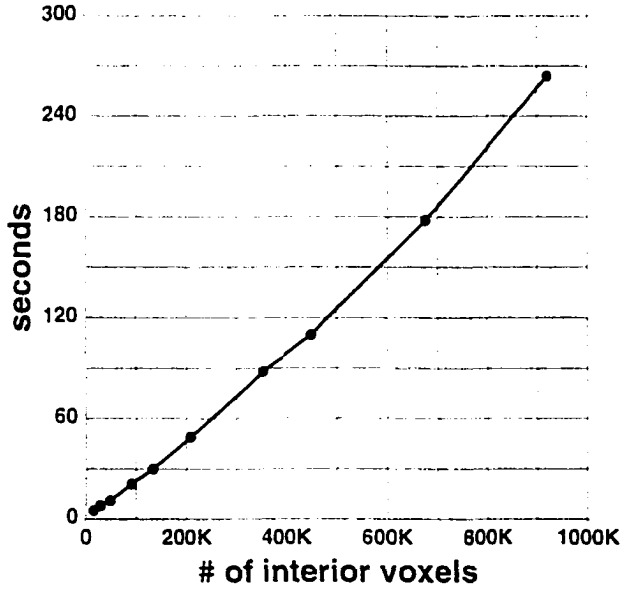
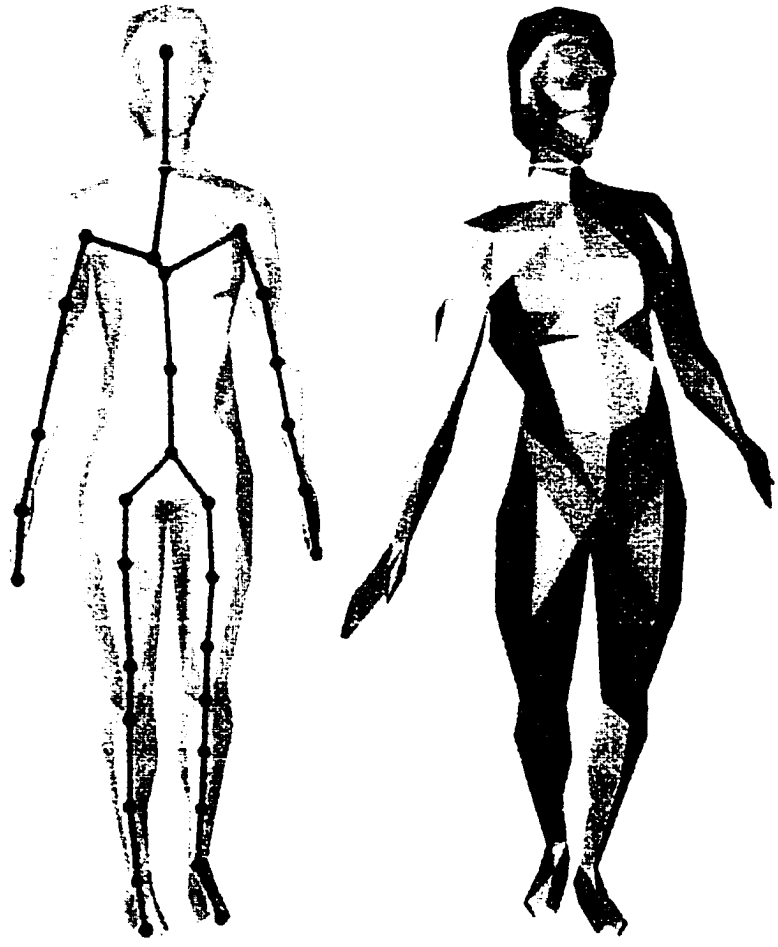


Figure 5.11: An analysis of execution times for the algorithm. Execution time (in seconds) is graphed against the number of interior voxels for several executions of the algorithm on the horse (each dot corresponds to one execution). Except for the voxel-size, all input parameters used default values. The time complexity appears to be superlinear with respect to the number of interior voxels (see page 134 for an approximating function). Graphs created for other objects were very similar.

centralized, and they have branches that reach to the ends of the major protrusions of the objects. The segments and joints relate fairly well to the surface features, although there is some room for improvement here, notably in the control features produced for some of the limbs posed in a straight fashion (observe the apparently arbitrary segmentation in the arms and legs of the human in Figure 5.12, for instance). With respect to the attachment problem, the scheme used is relatively simple and yet still quite effective under moderate repositioning of the control skeleton.

As for the idea of having "just enough but not too much" of a control skeleton, results are rather highly dependent on the objects given as input. For the most part,



(a) (b)

Figure 5.12: The control skeleton for a human figure and one pose. The skeleton in (a) was generated using a voxel-size parameter of 0.005 and an exposure threshold of 1.0. All other parameters used default values. The figure in (b) is the result of a selected random pose of the skeleton.

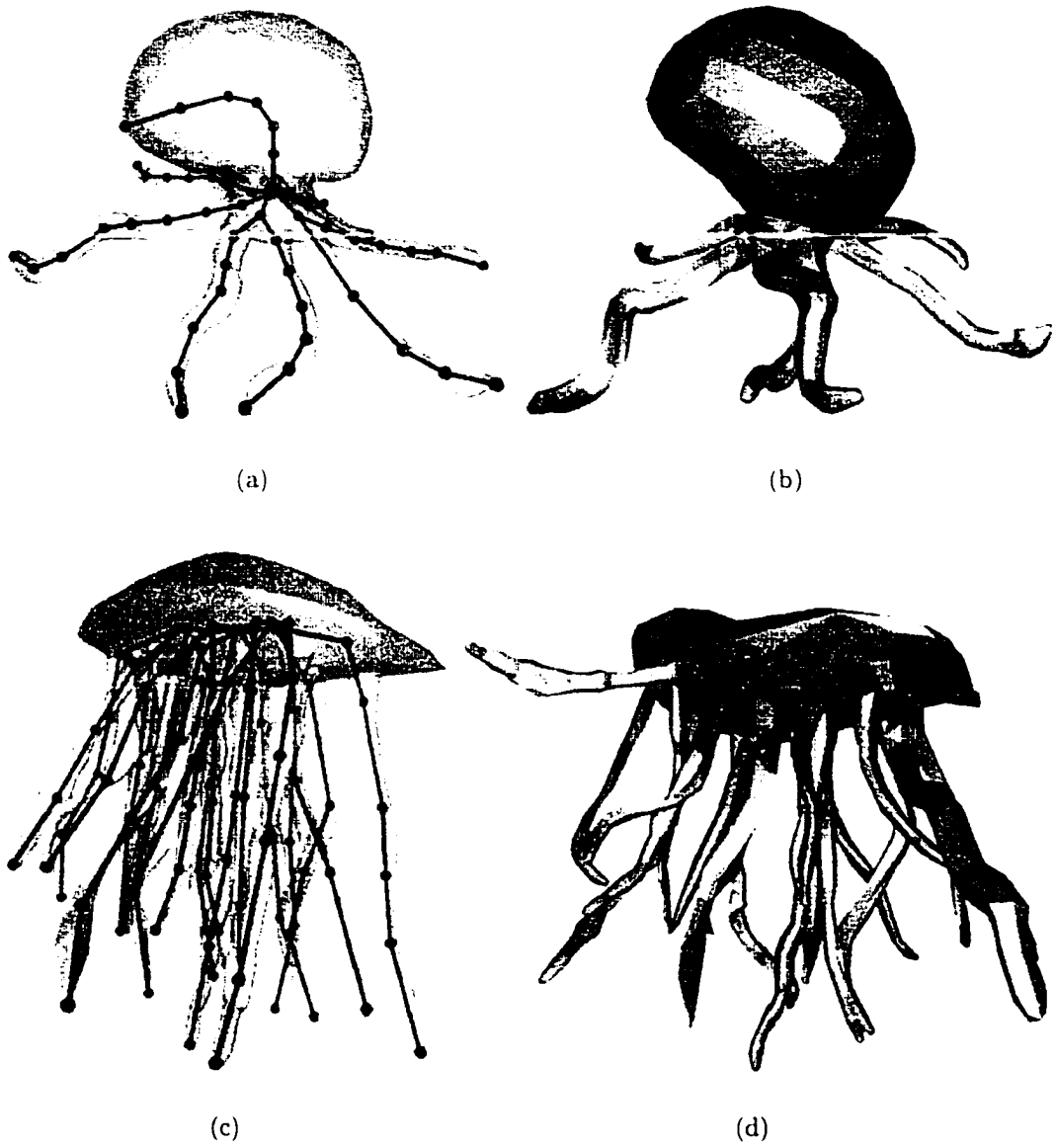


Figure 5.13: Control skeletons and poses for an octopus and a jellyfish. The skeletons in (a) and (c) were generated using a voxel-size parameter of 0.008. All other parameters used default values. The figures in (b) and (d) are the results of selected random poses of the skeletons.

skeletons produced by specifying the voxel-size and using default values for the other input parameters are reasonably succinct. (A list of the various input parameters and their default values is given in Table 5.2. With a closeness-of-fit value below 0.1, the algorithm can extend the skeleton into shorter surface protrusions such as the fingers of a hand; when doing so, however, it also usually produces at least a few spurious branches in other parts of the figure.

One area where the algorithm can have noticeable difficulty is with multi-junction points, such as where two "arm" sections of the path tree might joint a "spine" section of the path tree (often the arm sections join the spine section at different points). It can sometimes be useful to increase the exposure threshold to 0.9 or 1.0 so that the DMS used for path tree generation is rather lean. This usually collapses the area involved in the multi-junction point and sometimes results in better joining of path tree extensions in the area of the junction.

As can be expected, the quality of the skeleton is dependent on the quality of the voxelization of the object. The use of finer grids allows better approximations of the surface details of the object, but not without a cost – simply halving the voxel-size parameter will produce eight times the number of interior voxels and result in a related increase in the running time. As long as the topology of the grid is not compromised, a coarse grid can still produce reasonable results: the main benefits of a finer grid are better centralization of the control skeleton and better determination of surface protrusions (the latter allows better application of the closeness-of-fit parameter).

Several shortcomings of the algorithm have been identified:

- Because the algorithm produces a tree-structured control skeleton, it does not work very well for objects with holes; for example, when given an object such

as a doughnut, it will create a C-shaped skeleton. With some additional programming, the algorithm might be extended to produce a kinematic constraint that effectively closes the "C" during animation.

- The step-wise greedy approach to splitting the control segments in order to produce a desired number of them is probably not the optimal method, especially since it only considers bifurcations of the segments.

- Long, straight sections of the path tree are sometimes segmented in a seemingly arbitrary fashion. This can be the case when a figure's arm is posed without a bend at the elbow – sometimes no elbow joint is generated, at other times, numerous joints are generated along the straight-away. In contrast, when the input figure has bent limbs, the algorithm does very well at producing joints at the expected locations.

- For many objects, a tree being one example, it is probably not desirable to have the root joint centrally located with respect to the articulation points of the control skeleton. An input flag could be provided to request that the root joint be placed at the lowest end-voxel of the path tree, or better yet, a user could simply select the root once the control segments have been generated.

- The surface attachment scheme is rather simplistic, which is an advantage in terms of easy understanding and implementation. The repositioned surface, however, can sometimes suffer from interpenetration problems, especially if the joints are bent beyond a small amount (say 20 or 30 degrees). In the vicinity of the joints, sufficient numbers of vertices are necessary to minimize the penetrations, and the algorithm could be extended to produce extra vertices near the

joints; regardless, a better and probably more complicated attachment scheme is necessary to avoid the penetration problem.

- Often the algorithm produces a control skeleton that overall is quite good but that could use some tweaking. Since the focus of this research has been the automation of the control skeleton construction, there is currently no interface for tweaking it; nevertheless, such an interface would definitely be useful and indeed would be required for widespread use of the algorithm. A better idea would be to convert the implementation into a plug-in for a software package designed for modeling and animation and to allow tweaking of an automatically generated skeleton via the skeleton-control interface of that package.

5.4 Conclusion

This chapter has detailed an approach to automating the process of generating control skeletons. The method described achieves a higher degree of automation than previous approaches; furthermore, the algorithm is very fast, quite general, and fairly robust. With very little user input, the algorithm produces control skeletons of relatively good quality, sometimes good enough for immediate use in animation. At the very least, the algorithm is generally useful for providing an initial skeleton that an animator could hand-tune. It is especially useful for producing skeletons for more complex objects like trees or jellyfish, where creating a skeleton by hand would be a tedious and time-consuming process.

The algorithm is intended as a general solution to the problem of automatic generation of control skeletons. It must be emphasized that the algorithm constructs a control skeleton based solely upon a geometric analysis of the object. The algorithm

has no knowledge of what kind of an object it is dealing with, nor of any semantic relationships between the parts of an object. Of course, this does not prevent a user from having definite ideas about what kind of skeleton should be produced based on what type of object was provided as input. Nonetheless, even in the face of possibly unrealistic assumptions on the part of the user, the algorithm can often produce acceptable results.

The remainder of this dissertation is an investigation of how knowledge of certain types of objects can facilitate the creation of control skeletons that might be deemed as more appropriate for those particular objects. Specifically, the research examines skeletonization of animal-like and human-like models, easily the most common classes of objects whose members are typically animated in an articulated fashion using a control skeleton.

The research to come suggests various assumptions that might assist the algorithm from this chapter in producing a more desirable skeleton for objects from these classes. In nature, for example, it is often the case that a large, interior region of an animal is populated by several short bones (witness the vertebrae) and a long, narrow region is populated by a few long bones (arm and leg bones, for instance). It is also the case that the vast majority of creatures have an anatomical skeleton exhibiting some type of symmetry. Observations such as these can be reformulated as heuristics that can be incorporated into automatic control skeleton generation. Such ideas are the topics of discussion in the chapters that follow.

CHAPTER 6

COMPARATIVE ANATOMY OF VERTEBRATES

This chapter describes the anatomical knowledge upon which the remainder of this dissertation is based. Because the majority of articulated figures are human-like or animal-like, and because it is the anatomy of the human or animal that determines its movement capabilities, any attempt at the automatic generation of anatomically appropriate control skeletons would seem remiss without such an investigation into human and animal anatomy.

The discussion in this chapter draws from sources specific to human anatomy [Mad94] and animal anatomy [EBD56], as well as from the slightly more general areas of artistic anatomy [AS79, Par90] and comparative anatomy [Ken87, Hil95, Har99, Par88, Joh94, Ale94]. The wealth of information from Kent [Ken87] has been especially enlightening; indeed, many of the details of the following discussion come directly from his book.

Artistic anatomy can be loosely summarized as an examination of anatomy where the focus is on those features which influence surface form. In computer graphics, artistic anatomy texts are often sought when attempting to "flesh out" a figure; that is, when an animator employs individual models of bones, muscles, and other anatomical components to create a layered model for a figure. Typically the goal

is to arrive at a final model which, when posed or animated, will have a surface that will deform in accordance with the repositioning of the underlying (deformable) component models.

Comparative anatomy is the study of the anatomical similarity of different species of the animal kingdom, with a popular focus being the comparison of human and animal anatomy. This generally involves an exploration of the entire subphylum *Vertebrata* or its encompassing phylum, *Chordata*.

The similarities in vertebrate anatomy provide the foundation for the research that follows: the motivation consists of two basic goals. The first objective is to improve the automated generation of control skeletons so as to produce a more anatomically appropriate skeleton, meaning that the control skeleton created for a figure should exhibit the expected anatomical flexibility of the figure. The realization of this objective is the topic of Chapters 7 and 8. The second objective is to develop generalized component models for use in the automated generation of a layered model to flesh out a figure. Steps toward this goal are the subject of Chapter 9.

The presentation that follows looks at vertebrate anatomy with an eye toward the two goals just mentioned. Section 6.1 begins by laying out some simple principles concerning the general structure of vertebrates. Sections 6.2 and 6.3 then focus on two specific anatomical systems within vertebrates: the skeleton and the musculature. Note that Appendix A provides a glossary of the anatomical terms used in this dissertation.

6.1 Vertebrate Structure in General

Vertebrates are animals that have a spine or backbone. This includes amphibians, birds, fishes, mammals, and reptiles. Due in part to their common evolutionary ancestry, vertebrates exhibit a remarkable level of similarity. Comparisons between any two species of vertebrates will reveal similarities in their overall structure as well as similarities in the structure and function of their various anatomical systems. Although there will be noticeable differences between the species, especially as the age of the nearest common evolutionary ancestor becomes further and further distant, large numbers of anatomical similarities will still exist. What follow are some general principles of or relating to vertebrate structure.

6.1.1 Bilateral Symmetry

When discussing vertebrate structure, anatomists often refer to three principal axes. The longitudinal axis runs from the anterior end of the body to the posterior, or from the head to the tail, and the dorsoventral axis runs from the dorsal (back) side of the body to the ventral (belly) side.[14] The left-right axis simply runs from the left side of the body to the right side.

Structurally, the left and right sides of a vertebrate are mirror images of each other; hence, vertebrates exhibit bilateral symmetry. The most obvious example of this is the pairing of the limbs, but bilateral symmetry also manifests itself in countless

[14]It should be noted that for human anatomy, the terms anterior and posterior are typically used to refer to the front and the back of the human body which is usually presented in standing position. In such a stance, anterior becomes synonomous with ventral, and posterior becomes synonomous with dorsal. In the discussions within this dissertation, however, the terms anterior and posterior are used in a more general sense - anterior to mean toward the head, and posterior to mean toward the hind quarters or tail.

ways with respect to the anatomical systems within an individual and seems to be a persistent feature in the skeleton and musculature.

6.1.2 Two Sets of Paired Limbs

Vertebrates can be roughly divided into two groups: fishes and tetrapods. Amphibians, birds, mammals, and reptiles are all tetrapods, which literally means "four-footed." Tetrapods have two sets of paired limbs, though in some tetrapods (dolphins, whales, and snakes, for instance) one or both pairs may be vestigial. Some fishes can also be seen as having two pairs of appendages, though, so some anatomists prefer to define tetrapods as "vertebrates that dwell on land (or that had land-dwelling ancestors)." [Hil95] The research here leans primarily toward the structure of tetrapods.

6.1.3 Cylindrical Shape

An intriguing characteristic of the form of vertebrates, and one that permeates the natural world, is the generally cylindrical shape of so many things. Fingers, toes, limbs, tails, trunks, and necks all have a nearly cylindrical shape. So do blood vessels and several other conduits within the body. And countless items of anatomy (numerous bones, muscles, and parts of the digestive tract, for instance) that could not really be said to be cylindrical often have a nearly circular cross section. The reasons for this are beyond the scope of this dissertation[15] (see [Wai88]), but the idea is worth noting when contemplating an abstraction of the skeleton or musculature.

[15] In the simplest of summaries, it relates to the economy exhibited by rounded shapes with respect to the ratio of perimeter to enclosed area or the ratio of surface area to enclosed volume.

6.1.4 Metamerism

Another principle, at least with regard to the vertebrates, is usually not as obvious as the ones previously described. It usually becomes apparent only on an examination of certain internal anatomical structures. That principle is metamerism. With respect to primitive species (worms, for example), metamerism refers to a segmentation of the body into nearly identical, or homologous, segments. Metamerism is easily visible in embryonic vertebrates, but the specialization processes that occur during development distort and obscure the once metameric structures. Typically only tiny remnants of metamerism or its results remain in adult vertebrates; examples in the skeleton include the vertebrae and ribs, and examples in the musculature consist primarily of some oblique and longitudinal muscle groups running along the spinal column. [Ken87]

6.1.5 Form Follows Function

A final point to note is the resounding message so often expressed in biology classes: form follows function. If an anatomical entity exhibits a particular shape, then it does so because that is the shape it needs to have in order to function effectively. Stated another way, if a particular function is necessary for evolutionary survival, then better forms will evolve to serve that function. Although this principle is probably too general to be of any specific use with respect to creating a generalized control skeleton or fleshing one out, it is nevertheless such an important tenet that such a task should probably not be undertaken without at least being aware of the principle.

6.2 Skeletal Anatomy of Vertebrates

The skeletal system of most vertebrates follows a fairly typical pattern. It can be divided into two main parts: the axial skeleton and the appendicular skeleton. The axial skeleton includes the skull, rib cage, spine, and tail bones (which are basically an extension of the spine); the appendicular skeleton consists of the bones of the girdles and limbs. Table 6.1 provides a simple breakdown of these parts.

The Axial Skeleton

Numerous bones make up the skull. For purposes of this research, in which articulation is the primary focus, the skull can be seen as consisting of just two bones. The first of these is the cranium, whose primary features include the brain case, the eye sockets, any nasal openings, and the upper teeth. The other bone is the mandible, or jawbone, which houses the lower teeth.

The spine and tail are composed of vertebrae. The vertebrae are divided into groups based upon regional specialization. From anterior to posterior, there are cervical, thoracic, lumbar, sacral, and caudal vertebrae. Cervical vertebrae function to give an animal a flexible neck so that it can turn or nod its head. Thoracic vertebrae provide anchor points for ribs, with one pair of ribs for each thoracic vertebra. Lumbar vertebrae allow for a flexible lower back. Sacral vertebrae generally fuse with each other and with the pelvic bones to help brace the body against the movement of the hind limbs. Caudal vertebrae run along part or all of a flexible tail (in humans, the caudal vertebrae fuse together into the coccyx).

As mentioned, the ribs are joined to the thoracic vertebrae in the back. In the front, the ribs either join to the sternum or terminate without a joint, in which case

Axial Skeleton	
Spine and tail (vertebrae)	
Rib cage (ribs and sternum)	
Skull (cranium and mandible)	
Appendicular Skeleton	
Anterior	*Posterior*
Pectoral girdle (scapula, clavicle)	Pelvic girdle (pelvis)
Forelimb	Hind limb
Upper arm (humerus)	Thigh (femur)
Forearm (radius and ulna)	Shank (tibia and fibula)
Wrist (carpals)	Ankle (tarsals)
Palm (metacarpals)	Instep (metatarsals)
Digits (phalanges)	Digits (phalanges)

Table 6.1: A simplified view of the skeletal components of vertebrates (based on tables from Kent [Ken87]).

they are referred to as floating ribs. As a whole, the rib cage protects the vital organs of the thoracic cavity (the heart, lungs, liver, and kidneys).

The Appendicular Skeleton

Discussion of the appendicular skeleton typically follows the parallelism of its anterior and posterior components, comparing the forelimbs and the hind limbs and contrasting the girdles connecting the limbs to the axial skeleton. The pectoral girdle consists of the clavicles (if present) and the scapulae; it connects the forelimbs to the trunk via the rib cage. The pelvic girdle, which is essentially the pelvis, connects the hind limbs to the spine. Whereas the bones of the pectoral girdle can operate independently for the left and right halves, the bones of the pelvic girdle are generally

not independent, being immobilized through symphysis with each other and fusing with the sacral vertebrae.

The basic limb structure for vertebrates can be viewed as consisting of five segments [Ken87]. For the forelimb, those segments (and their associated bones) are the upper arm (humerus), forearm (radius and ulna), wrist (carpals), palm (metacarpals), and digits/fingers (phalanges). For the hind limb, the five segments are the thigh (femur), shank (tibia and fibula), ankle (tarsals), instep (metatarsals), and digits/toes (phalanges). The hind limb commonly has one other bone at the knee: the patella.

In comparative anatomy, the term *manus* is often used as a generalization of the term *hand*. Specifically, it refers to the wrist, palm, and digits of the forelimb. Likewise, the term *pes* is used as a generalization of *foot* to refer to the ankle, instep, and digits of the hind limb. For quadrupeds, the structures of the manus and pes are often very similar (such as with the horse or dog); for bipeds, however, there is usually a greater difference due to the differing functions of the forelimb and hind limb.

Like the bones, the joints for forelimbs and hind limbs have a similar pattern. The shoulder and hip joints are ball-and-socket joints, having three degrees of freedom (DOF). The elbow and the knee are both hinge joints, having a single degree of freedom, though the elbow generally points toward the rear of the animal while the knee generally points toward the head of the animal. As with the tibia and fibula with respect to the foot, the radius and ulna allow for the pronation (rolling inward) or supination (rolling outward) of the hand. Note that for many species, the ulna is either non-existent or is fused to the radius, so no such rotation of the hand is possible. The same holds in the hind limb, where the fibula disappears or fuses with

the tibia. The wrist joint and the ankle joint both allow two rotational DOFs, and the joints proximal to the metacarpals and metatarsals allow two rather heavily limited DOFs. Joints of the phalanges can be considered as hinge joints with one DOF whose axis is perpendicular to the digit.

6.2.1 Differences

With a few exceptions, the bones of one species are often fairly similar to the homologous bones of another species of vertebrates (in this context, the word *homologous* actually refers to the correspondence of bones between species). Nevertheless, there are some noticeable differences. Although the basic form of homologous bones is usually the same between species, the bones may differ (perhaps vastly) in length, girth, and the sizes and shapes of distinct features.

The number of bones within functional groupings may also differ from species to species: common examples are the number of ribs, vertebrae, carpals, tarsals, and digits. Many birds have seven pairs of ribs, mammals commonly have twelve pairs (though some have as few as nine or as many as twenty-four pairs), and snakes (notably an exceptional vertebrate) can have hundreds of pairs [Ken87]. Note that difference in quantity is sometimes obscured as a result of bones ankylosing, or fusing together; for instance, the human sacrum consists of five fused vertebrae. As another example, the synsacrum of birds is a massive fusing of thoracic, lumbar, sacral, and caudal vertebrae and a couple of ribs, which itself then fuses with the pelvic girdle [Ken87].

The orientation of adjacent bone groups can also vary. In many animals, the spinal column meets the back of the skull. This usually means that the back-to-front

line of the skull is nearly colinear with the spine. In many other animals (humans being one example), the spinal column meets the base of the skull, so the alignment of the head and trunk might be viewed as perpendicular.

Perhaps the most important difference across vertebrate skeletons is that homologous bone groups do not necessarily have the same function. On a local scale, for example, the opposable thumb of humans serves a different function than the non-opposable first digit of some other primates. On a more global scale, the forelegs of a quadruped have a vastly different function than the arms of a biped or the wings of a bird. Finally, as alluded to earlier, some bones or groups of bones may be vestigial in some species.

6.3 Muscular Anatomy of Vertebrates

6.3.1 Muscle Basics

The musculature of vertebrates can be discussed from several vantage points. One view divides the muscles into two groups: somatic muscles and visceral muscles. Somatic muscles are primarily responsible for interacting with the external environment and are mostly voluntary muscles, meaning they can be flexed at will. These include the muscles of the body wall, the appendages, and the tail. Visceral muscles, which are mostly involuntary, are responsible for internal body functions. These are usually muscle sheets around hollow organs, tubes, and ducts, such as those muscles responsible for peristalsis along the digestive tract. For the purposes of this research, the only concern is with somatic muscles.

A single muscle consists of a belly (the contractile portion) and possibly one or two tendons connecting the belly to different bones. The points where the muscle or

tendon attaches to the bones are known as the origin and the insertion. In general, the origin is on the bone that remains fixed when the muscle is contracted, and the insertion is on the bone that is moved when the muscle is contracted.

Muscles come in a variety of shapes depending on the sizes and locations of their origin and insertion points. The simplest type of muscle is the fusiform muscle. It has one origin and one insertion and is spindle shaped. The biceps brachii and several other limb muscles are spindle shaped. More complicated muscle forms arise if a muscle has several points of origin and/or insertion. Fan-shaped muscles are common in the chest and shoulders, and sheet-like muscles are common in the abdominal wall. [Hil95]

Since the active function of a muscle is to shorten, muscles are arranged in antagonistic pairs throughout the body. With regard to the bone to which they attach, the muscles of an antagonistic pair usually act to flex or extend it, to adduct or abduct it, to protract or retract it, to lift or depress it, or to rotate it one way or the other.

6.3.2 The Musculature

Like the skeletal system, the musculature for vertebrates can also be divided into axial and appendicular components [Ken87]. The muscles of the trunk and tail are the primary axial muscles. The appendicular muscles are themselves usually divided into two groups: extrinsic appendicular muscles, which connect the limb or girdle to the trunk, and intrinsic appendicular muscles, which connect one section of a limb or girdle to another section of the limb.

The musculature typically change more quickly over an evolutionary time line than does the skeleton. Therefore, it is generally more difficult to determine homologous

muscle groups amongst different species than it is to determine homologous bone groups. Fortunately, the homologies between major muscle groups are fairly clear. An exhaustive discussion of the numerous muscles comprising the musculature of vertebrates and detailing the differences between the musculatures of the various classes is beyond the scope of this dissertation; however, some generalizations are worth noting.

The axial muscles consist primarily of two groups: epaxial muscles and hypaxial muscles. The epaxial muscles originate on one vertebra and insert on one or more vertebra or possibly on the base of the cranium. These are the oblique and longitudinal muscles along the spine that function to bend or to stabilize the spine. The hypaxial muscles are the sheet-like muscles of the body wall. They serve less to move the skeleton than they do to contain the innards of the trunk. Other axial muscles of significance are the muscles of the jaw.

For the extrinsic appendicular muscles, a pattern emerges: the dorsal or posterior muscles are responsible for extending the appendages, while the ventral or anterior muscles are responsible for flexing the appendages, or bringing them closer to the body [Ken87]. The latissimus dorsi and the trapezius are examples of the dorsal muscles; the pectoralis muscles are examples of the ventral muscle. Note that these examples are muscles affecting the forelimbs. Since the pelvic girdle is fused to the spine, it cannot move independently, so their are essentially no extrinsic appendicular muscles that affect the hind limbs.

For the intrinsic appendicular muscles, other patterns emerge, depending in part on whether the limb is a forelimb or a hind limb. There are also parallels between the fore and hind limbs. The deltoideus and other shoulder muscles stretch from the

scapula to the humerus over the shoulder joint. These muscles function to adduct and to rotate the humerus. In a similar fashion, a group of muscles stretching from the pelvis to the femur across the hip joint (the gluteus being one of them) functions to abduct and to rotate the femur. The triceps, two heads of which originate on the humerus and one of which originates on the scapula, inserts on the ulna and functions to extend the forearm. In comparison, the quadriceps femoris originates either on the pelvis or an upper portion of the femur and inserts more or less on the patella. Three of its four muscles function is to extend the lower leg, while the other functions to adduct the thigh. The primary antagonists for the triceps are the biceps brachii and the brachialis, while the primary antagonists for the quadriceps femoris are the biceps femoris (commonly referred to as the hamstring in humans). These muscles function to flex the forearm and the lower leg, respectively. Various muscles of the forearm act to pronate or supinate the manus or act either directly or through long tendons to flex or extend the manus or its digits. Various muscles of the lower leg affect the pes and its digits in a similar manner (the gastrocnemius is a one of these). [Ken87]

As one final note on the musculature of vertebrates in general, the footprints of the major muscles upon their bones of origin and insertion have marked similarities. Homologous bones typically have similar features (protrusions, processes, and so forth), and each of those features typically serve to provide places of origin or insertion for one or more homologous muscles. This usually holds true even if the functions of those homologous muscles differ somewhat between various species.

CHAPTER 7

AUTOMATED IDENTIFICATION OF ANATOMICAL FEATURES

This chapter and the one that follows discuss how knowledge of human and animal anatomy as presented in the previous chapter can be incorporated into the skeletonization algorithm described in Chapter 5. The use of this knowledge is intended to assist the algorithm in producing more appropriate control skeletons for human-like and animal-like figures. This chapter will describe various assumptions and heuristics for automatically identifying gross anatomical features of the figures. The next chapter will then show how those classifications can be employed to generate a control skeleton that might mimic the expected anatomical flexibility of the figure.

Section 7.1 lists a number of basic constraints imposed on the input object in order to simplify the task of identifying its parts. Based upon those constraints, Section 7.2 sketches some simple heuristics for identifying gross anatomical regions. Section 7.3 describes the implementation of these heuristics within the skeletonization system, and Section 7.4 briefly discusses some typical results of applying the method to various objects.

The research described in this chapter takes a fairly simple approach to the problem of automated identification of anatomical features. This basic method for the

anatomical breakdown of an object works reasonably well within the larger scope of this dissertation research. Nevertheless, a reader interested in a more thorough treatment of such automation might enjoy delving into the broad area of artificial intelligence, and more specifically, the topic of pattern recognition.

7.1 Constraints

The fundamental goal of this chapter is to divide the object into regions corresponding to basic anatomical features found in humans and animals. In short, this consists of sectioning the figure into a trunk and various appendages, and labeling those appendages as arms, legs, wings, and so forth. To achieve a reasonable degree of success in this task, the algorithm relies on various assumptions about what type of figure it is dealing with and how that figure is posed. These assumptions are essentially constraints that a user must ensure are satisfied before the algorithm may be expected to perform its share of the automation process.

7.1.1 Structural Constraints

The primary assumption is that the object provided as input is some human-like or animal-like figure. What that means and what sorts of constraints that assumption imposes on the basic skeletal structure the figure is expected to possess are discussed below.

Vertebrate

The figure is assumed to be of a vertebrate creature; that is, it is expected that the creature would have a backbone or spinal column in real life. Living vertebrates consist of amphibians, birds, fishes mammals, and reptiles; however, the figure need

not be that of a living creature. It is simply expected to have a vertebrate-like anatomical structure so that it is well-suited for animation via a control skeleton composed essentially of spinal segments and appendicular segments. This expected structure is most easily evident in the tetrapods, or the non-fish vertebrates. Note, however, that there is no constraint that the figure must have four limbs, though it will be assumed that the figure has at most one head and at most one tail.

Tree-like Structure

The overall skeletal structure for vertebrates in general is tree-like. Some might argue that the rib cage imparts cyclical structural elements on the skeleton or that the human shoulder complex, since it exhibits inherent dependencies of the combined human skeletal and muscular structure, is more accurately modeled in a cyclical fashion. Nevertheless, for the purposes of this research, the gross description of all vertebrate skeletons as tree-like is acceptable; thus, it is also assumed that the control skeleton for the figure should possess such a hierarchical structure.

Bilateral Symmetry

As discussed in the previous chapter, another overwhelming characteristic of vertebrate skeletons is their bilateral symmetry. This is most obvious with regard to their limbs, which form in pairs. In this research, limb-like appendages of the figure will be expected in pairs so that the expected control skeleton will exhibit bilateral symmetry.

Proportions

The final structural constraint assumes that the figure is proportioned in a reasonable manner. This means that any limb of the figure is expected to have approximately

the same dimensions as its pair. It also means that there is a credible relationship between the sizes of different elements of the figure; for instance, if one element is too large, other elements may be overlooked by the algorithm as being insignificant.

7.1.2 Postural Constraints

The second set of constraints require that the input object be presented in a way that does not obfuscate the identification process.

Orientation

When given as input, the figure is assumed to be oriented such that the y-axis points in the approximate "up" direction with respect to the figure. The user must then specify an approximate "forward" direction for the figure. For human-like figures, the forward direction should be the ventral direction; for animal-like figures, it should be some combination of the ventral and anterior directions depending on the resting pose of the figure. For a quadruped, the forward direction is almost exclusively the anterior direction, but for a bird in a standing position, it might be a more equal combination.

Pose

Although there is a structural constraint that the skeleton generated should be symmetric, it is not necessary that the figure itself be posed in a symmetric fashion. The figure is assumed to be in a stable, self-supporting pose. This requirement allows for easier identification of the figure's legs. A further expectation of the pose is that there is sufficient space between the limbs of the figure as well as between the limbs and the trunk. This allows for a voxelization of the figure to have a separate

protrusion for each limb and helps form a more clear delineation of where the trunk might be seen to end and a limb might be seen to begin.

7.2 Heuristics for Identification

Based on the constraints just discussed, the figure is to be divided into sections corresponding to its major body parts. The constraints also provide the basis for various expectations regarding each type of body part. The following list shows the assumptions made with regard to each basic type of body part:

- Trunk - The trunk is expected to be central to the figure, to contain the heart (that is, the geometric concept of the heart defined in Chapter 5), and to be bounded by major junctions of limbs or other protrusions (any region of the figure that is adjacent to only one major junction is assumed to be a protrusion: arm, leg, wing, head, or tail).

- Leg - Legs are expected to support the figure and run from the ground to the trunk. In other words, the legs are expected to define the extreme part of the figure in the downward direction. Legs are also likely to have a vertical or mostly vertical orientation between the ground and the trunk, and they are expected to be found in pairs.

- Arm - Arms are expected to join the anterior part of the trunk and to come in pairs. They are not necessarily expected to reach the extreme of any direction, though it would not be surprising to find them defining the lateral extremes of the figure.

- Wing - Whereas arms and legs are expected to have circular, elliptic, or at least somewhat regular cross sections, the wings are expected to have fairly linear cross sections. Like arms and legs, wings are expected in pairs.

- Head - The head is expected to define the forward extreme, the upward extreme, or both the forward and upward extremes for the figure. It is expected to join the most anterior part of the trunk.

- Tail - If present, a tail is expected to extend in the rearward direction and to join the trunk at roughly the same point as the most posterior pair of limbs.

7.3 Implementation Issues

The heuristics above are realized in three phases. In the first phase, a special graph is created that effectively simplifies the DMS. The second phase involves marking vertices of the graph according to certain characteristics: whether they are major junctions for the figure, whether they are near the extreme top or bottom of the figure, what the general shape of the cross section is for that part of the figure, and so forth. In the final phase, the sections of the graph corresponding to various body parts are identified and labeled.

7.3.1 Creating the Level Graph

For the purpose of simplifying the DMS, the algorithm relies on the heart computation from Section 5.2.4 in the Chapter 5. Recall that the heart is a voxel that is well centered with respect to the DMS voxels. During the computation of the heart, each DMS voxel accumulates a series of the shortest distances between itself and other (source) voxels. After a sufficient number of voxels have acted as sources for

spreading distances, accumulation stops. The minimum and maximum accumulator values are found, and each DMS voxel is then assigned a normalized heart value by determining where its accumulator value lies between the minimum and maximum.

In a manner not unlike that of forming level set diagrams (see the last paragraph of Section 2.3.2 for a brief review of the work of Lazarus and Verroust [LV99]), a graph termed the level graph is formed using the heart values. The level graph functions to divide the interior of the figure into interconnected strata. First, the DMS voxels are partitioned into sets according to the percentile in which their heart values reside. Experimentally it has been shown that using about twenty sets works fine for most figures. In this case, the first set would consists of all DMS voxels whose heart values were within 5% of the minimum, the second set would include voxels with heart values within 5% to 10% of the minimum, and so forth. Next, each set is divided into connected components based on the adjacency of its DMS voxels. For each connected component, a corresponding vertex is created in the level graph. The position of the level graph vertex is the centroid of its connected component. Edges are added between level graph vertices if the corresponding sets of DMS voxels represented by the vertices are adjacent to each other in the DMS.

In the actual implementation, the level graph is formed in a bottom up fashion by analyzing sets of DMS points in decreasing order of heart value. Throughout the formation, the program maintains a record of the disconnected portions of the level graph. Each connected portion has a unique group number, and it is the group numbers that are compared when testing for adjacency during edge creation. In this way, the level graph formed will always be a tree, and the root of that tree will be the group of DMS points containing the heart voxel.

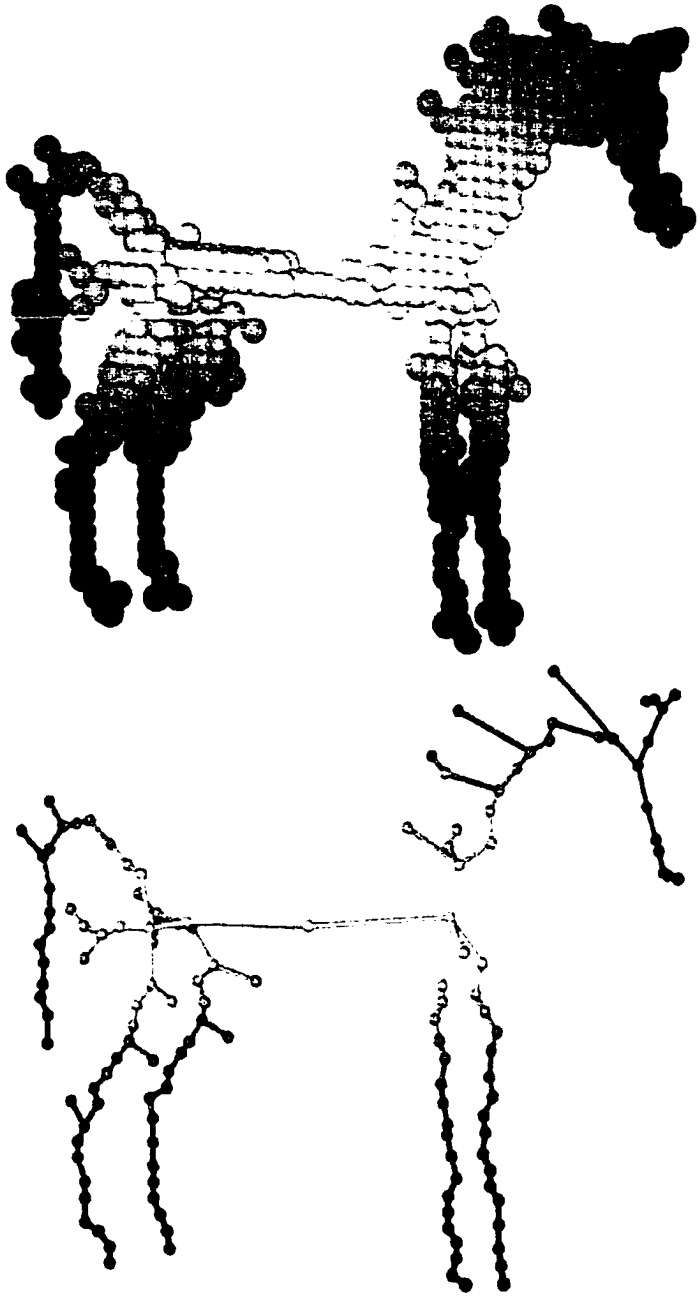


Figure 7.1: The DMS and the level graph of the horse shaded according to heart values. For the DMS of the horse, the voxels are shaded from white to black as the normalized heart values run from zero to one. The level graph for the horse is shaded to correspond with the DMS, from which the level graph was created.

The top of Figure 7.1 shows the DMS voxels of the horse shaded according to their heart values, and the bottom of the figure shows the level graph created. As should be apparent from the figures, the level graph is a simplification of the DMS. Whereas the DMS might have a "surface" of voxels that extends down the center of some protrusion of the figure, the level graph typically will have only a single chain of edges and vertices corresponding to that same portion of the protrusion. Due to the nature of how heart values will lie within a volume, the level graph effectively provides an approximately longitudinal view of each protrusion of the object. In addition, it offers a good indication of where each protrusion joins the trunk of the figure in relationship to other protrusions.

7.3.2 Marking the Level Graph Vertices

Because the level graph is a tree, and because each of its vertices is associated with a group of DMS points, it is a fairly simple matter to assign weights to the vertices according to how many DMS points are dependent on a particular vertex for connecting to the heart. The weight assigned to a vertex is a fraction ranging from zero to one. The numerator of the fraction is the sum of the number of DMS points that are contained in the group corresponding to the vertex and the number of DMS points contained in groups corresponding to other level graph vertices that are distal to the vertex in question. The denominator of the fraction is the total number of DMS points. The assignments of these fractions to the vertices can be computed fairly easily during the bottom up creation of the level graph. It is done by keeping track of the number of DMS points represented by each connected component of the level graph as it is formed.

There are two main reasons for keeping track of these fractions. First, observe that a very small fraction indicates that a level graph vertex and its distal group can probably be ignored when identifying gross pieces of the anatomy via the level graph. For this reason, vertices with a very small fraction are marked as being insignificant. Second, when a vertex is created that merges two or more sufficiently large regions of the DMS (corresponding to previously disconnected components of the level graph), the new vertex is marked as being a major junction for the figure. It is these major junctions that serve to carve the DMS into regions corresponding to the body parts of the figure.

Once the level graph has been created, each of its vertices is tested for a few simple criteria, or rather, the corresponding DMS points are tested and the result of the test is assigned to the vertex. Four of the tests deal with directions. If any DMS point for a vertex is within the top 10% of the voxelization (with respect to the y-coordinate), then that vertex is marked as an extreme vertex in the upward direction. Similarly, if any DMS point for a vertex is within the bottom 10% of the voxelization, the vertex is marked as extreme with respect to the downward direction. Other tests are used to mark vertices that are seen as extreme with respect to the forward or backward direction (recall that the user must specify which direction is considered to be forward). Note that the forward and backward tests are slightly more complex since the forward direction may not align with an axis of the voxelization. In preparation for the test, the most forward and the most backward interior voxels are found and then projected onto a line that is parallel to the forward direction vector. Each DMS point can then be projected onto the same line and tested to see where it falls between those two extreme projections.

The final test for a vertex is an attempt to determine the approximate shape of the cross section of the figure at that vertex, specifically, whether is should be considered rounded or linear. For a given vertex, the corresponding DMS point group is examined to find the point farthest from the group's centroid (which is the location of the levelgraph vertex). Another pass through the group points finds the point farthest from the first farthest point. The distances from these farthest points to the centroid is compared to the radius of the distance map sphere whose center is closest to the group's centroid. If either distance is greater than a certain multiple of the radius (a multiple of 1.5 seems to work fairly well), then the level graph vertex is marked as having a linear cross section; otherwise, the vertex is marked as having a rounded cross section.

7.3.3 Labeling the Level Graph

The level graph is considered to be divided into a number of sections by the locations of the major junction vertices. If the major junction vertices were to be removed from the graph, then the connected components that would remain would correspond to the sections.

The first part of the level graph that is labeled is the trunk. The trunk consists of the major junction vertices and any edges connecting them to each other. The slope of the trunk is examined to determine whether the trunk is mostly horizontal (such as that of an animal), mostly vertical (such as a human figure), or somewhere in between. After the trunk is labeled, a simple examination helps to weed out unimportant branches of the level graph: if any non-trunk section consists of only insignificant vertices, then the section itself is also labeled as insignificant.

Any unlabeled section that has an extreme vertex with respect to the downward direction is labeled as a leg. As legs are expected to be paired, there should be an even number of leg sections identified. In the event that an odd number is identified, then a tail section may have been mislabeled as a leg. This is sometimes remedied during the pairing procedure. Legs are paired based upon where they join the trunk, and each leg is paired with the closest unpaired leg. Trunk vertices or edges that aid in the pairing (via forming connections between paired legs) are marked as being pelvic pieces of the trunk.

The next body parts identified are wings. If at least 40% of the significant vertices in an unlabeled section have linear cross sections, then that section is marked as a wing. Any wings discovered undergo a pairing operation similar to that of the leg sections: portions of the trunk that aid in the pairing are marked as being girdle pieces for wings.

After the wings comes the tail. The section that extends farthest to the rear of the figure is examined. If it is unlabeled or if it is an unpaired leg, then it is (re)labeled as being a tail.

Head identification begins with unlabeled sections at the anterior portion of the trunk. If the trunk of the figure is found to be mostly in a vertical pose, then the assumption is that the figure is human-like, so whichever of those sections extends farthest upwards is labeled as the head. If the trunk is mostly horizontal, then the focus is on the section that extends farthest forward. If the trunk is slanted or if there is no clear direction for the trunk, then the topmost and/or forward most sections are compared – whichever is found either to align best with any trunk direction or most likely not to be paired with another unlabeled section is marked as the head

section. Note that the head section may or may not include a part that should really be considered as a neck.

The remaining unlabeled sections are analyzed to see if there are any pairs that join the trunk at approximately the same vertex. If there are, and if each of the sections has at least a moderate length, then those sections are labeled as arms. Once again, a pairing operation is used to mark the connecting trunk pieces as elements of a pectoral girdle.

Any insignificant sections and any unlabeled sections that are less than a sufficient length are now (re)labeled as portions of the trunk. Any unlabeled sections of at least a sufficient length are at this point brought to the attention of the user, who may then assign labels to them. Also at this point, the user may modify the labeling of any section.

After the user has made any corrections, each arm or leg section of the level graph is processed to determine if it might have separated digits (fingers or toes, respectively) as evidenced by notable branches toward the end of that section of the level graph. If a limb is discovered to have separated digits, then the limb is marked for specialized processing so that an appropriate hand or foot section of a control skeleton can be generated; otherwise, the limb is marked as needing either a manus with a single digit or a one-segment manus. This is discussed further in the next chapter.

7.4 Results

The identification algorithm works reasonably well, though it can have problems with certain objects. It seems to work quite well for quadruped figures (a horse or a

dog, for instance) where the various types of protrusions (head, legs, and tail) do not compete with each other for prominence in a common direction. For such figures, it rarely makes an incorrect classification.

It is less robust for human-like figures, sometimes confusing a human arm for a tail if the arm is posed in a slightly rearward direction. The confusion is the result of using a rather simplistic identification scheme that labels the tail section before identifying any arm sections. On a model of a dragon, which has two arms, two wings, and two legs, the algorithm performs fairly well, though it occasionally mislabels a wing as an arm in coarser voxelizations of the figure. A more common occurrence is the mislabeling of spurious branches of the level graph as arm sections (the reason for these actions is discussed below). All in all, the algorithm does fairly well, and when it does make an incorrect classification, it does not take long for a user to make the necessary corrections.

One problem with the algorithm is its dependence on the accuracy of the heart computation. Since the sources for the heart computation are chosen randomly, the position of the heart can vary from one execution to the next. On occasion, a slight variation in the heart position and the distribution of heart values can alter the level graph just enough to impede proper classification of its branches.

The main problem with the algorithm is that the level graph is computed on the set of DMS voxels. The DMS often has some spurious branches that have an adverse effect on the computations for the heart and level graph. Such spurious branches would likely not appear if the heart and level graph were computed on the set of all interior voxels. Unfortunately, that computation over the whole of the interior is prohibitively expensive, typically increasing the execution time by at least one

order of magnitude. Considering the trade-off between requiring more execution time versus requiring more user interaction to assist the algorithm, in this particular case, requesting a little help from the user seems well worth while.

CHAPTER 8

AUTOMATED GENERATION OF ANATOMICALLY APPROPRIATE CONTROL SKELETONS

This chapter describes how anatomical knowledge can be used with the feature classification algorithm from the previous chapter in order to generate a control skeleton that seems anatomically appropriate for a given figure. Section 8.1 discusses the application of the knowledge in the creation of the axial portions of the control skeleton, and Section 8.2 describes the application with respect to the appendicular sections of the control skeleton. Section 8.3 then discusses how the surface data is attached to the control skeleton. Section 8.4 concludes with some preliminary results of an implementation of the process.

8.1 The Axial Skeleton

The axial parts of the control skeleton are comprised of the segments and joints relating to the head, trunk, and tail. With respect to the axial portion of the figure, the rib cage has been ignored to some degree – greater consideration of the rib cage comes into play later during the formation of segments and joints of the pectoral girdle. The articulation of the rib cage is rather limited, and it is considered to add

redundancy to the control skeleton – that is, it adds cycles of dependency when modeled more realistically. For simplicity, the rib cage has been conceptually trivialized to have individual ribs positioned statically with respect to the coordinate frames of the vertebral segments to which they attach, and the rotational motion between these vertebral segments is assumed to be quite limited.

Recall from the previous chapter that the level graph has been partitioned so that its edges and vertices correspond to the various parts of the figure (head, tail, trunk, arms, legs, and wings). The level graph vertices are also associated with a partitioning of the DMS voxels; thus, the DMS voxels are assigned body part labels in accordance with their respective level graph vertices. The limbs of the figure have been paired, and the figure as a whole has been identified as being either human-like or animal-like depending on the orientation of the trunk portion of the level graph (a nearly vertical trunk implies a human-like figure; a horizontal, slanted, or essentially non-vertical trunk implies an animal-like figure).

The first step in realizing the axial control skeleton is finding a centralized path from the tip of the head to the tip of the tail. The tip of the head is found by examining the DMS voxels labeled as being part of the head and selecting the voxel that is farthest in the anterior direction. For animal-like figures, this typically corresponds to the figure's nose; for human-like figures, it is the top of the head. The tip of the tail is the tail DMS voxel with the highest heart value. If the figure does not have a tail, then another point is substituted as the posterior goal for the path. This replacement point is the midpoint of the centralized path connecting the extreme points of the hindmost pair of limbs.

Each centralized path (head-to-tail or limb-to-limb) is computed in the same manner as the first extension of the path tree in Chapter 5 (see Section 5.2.4 for more details). This is based on a modified version of Dijkstra's shortest paths algorithm where the weights are based on the reciprocals of the cubes of the distance map values. After each centralized path is computed, weights for the entire DMS are reinitialized so that the calculation of the next centralized path will not be affected by any previously computed paths.

After the head-to-tail path has been computed, the next step involves dividing the path into sections, with one section for the head, one for the neck, one for the trunk, and one for the tail. To help accomplish this sectioning, a path is computed for each pair of limbs connecting the most extreme DMS point of each limb with that of its pair. Each limb-to-limb path is then processed to find the middlemost voxel of the path - this is the voxel that most evenly divides the path into two roughly equivalent sections; for convenience, it will be referred to as the midpoint of the path. The head-to-tail path is then searched to determine which of its voxels are closest to each of the midpoint voxels of the limb-to-limb paths. With respect to the limbs, this serves two purposes: to determine the front-to-back ordering for the limb pairs and to find an approximate location for creating the girdle for each limb pair. The closest head-to-tail voxel to the hindmost limb pair's midpoint effectively marks the end of the trunk section of the head-to-tail path and the beginning of the tail section (provided, of course, that the figure has a tail). The closest head-to-tail voxel to the foremost limb pair's midpoint is considered the forward-most voxel of the trunk section. The rest of the head-to-tail chain (the forward-most part) is divided in half, with the anterior half representing the section for the head and the posterior half

representing the section for the neck. Although somewhat arbitrary, the head/neck division seems to work reasonably well for many figures.

In a more flexible system, a user would be provided with interface options to override arbitrary implementation decisions such as the placement of the head/neck division just mentioned. Nevertheless, providing too many such controls could overwhelm a user and undermine the potential benefits that automation offers. Since this research has focused mainly on automation of the processes involved, it has neglected the issue of user control to some degree. In Chapter 10, a workable interface is proposed that offers a compromise between user interaction and automation.

With the head-to-tail chain divided into sections, adjustments can now be made to the chain so as to make it more anatomically appropriate. Since the spinal column for most vertebrates runs along the center of the dorsal side of an animal's trunk, the trunk portion of the head-to-tail chain is adjusted dorsally. This is accomplished by examining the distance map spheres for sample trunk voxels of the chain and creating a replacement trunk section that is offset dorsally by 70% of the radius of each respective distance map sphere.

The cervical vertebrae for many vertebrates (especially those whose heads typically overhang the front portion of their bodies) form a mild S shape. For these animals, the joint between the first cervical vertebra and the cranium (at the top end of the S) is fairly high in the neck region. In order to imitate this characteristic in animal-like figures, a replacement section for the portion of the head-to-tail path near the junction of the head and neck sections is computed. The replacement section is also offset in the dorsal direction. Instead of using distance map spheres, however, rays

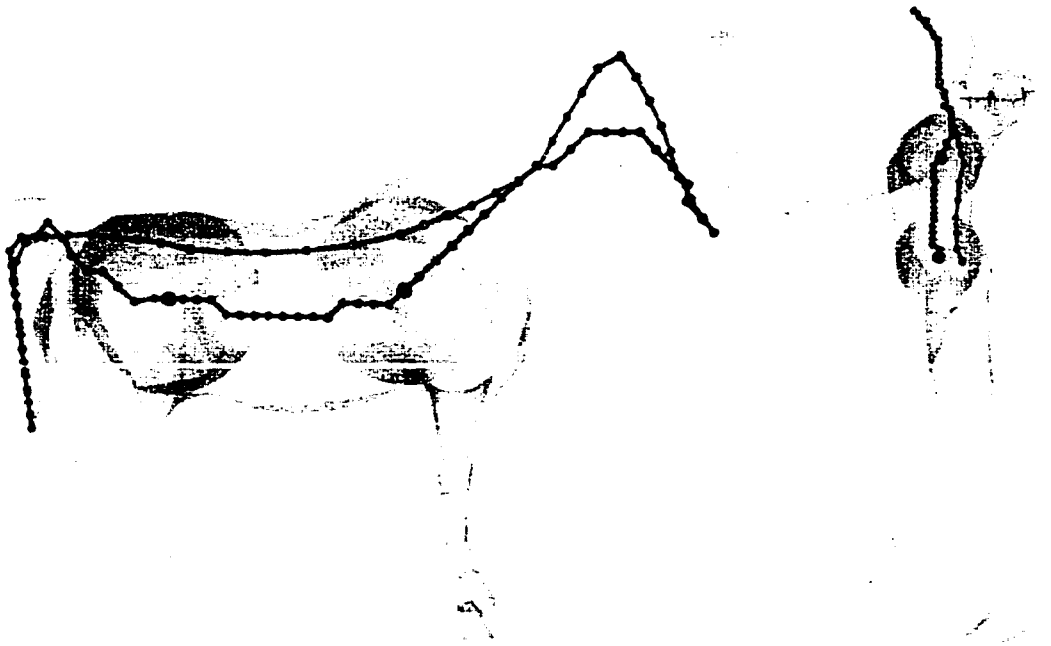


Figure 8.1: Head-to-tail chains and the adjusted head-to-tail chains of a horse and a human. The location of the adjusted chain is a better approximation to where the spine of the creature would be. The original head-to-tail chain is shown in black, and the adjusted chain is shown in dark gray. Silhouettes of the girdle spheres for each figure are also shown.

are cast dorsally from the corresponding voxels of the head-to-tail path, and points along these rays are used to create the offset path.

The head-to-tail path, having been modified with appropriate replacement sections, is then smoothed in an operation similar to that described in Section 5.2.4. Figure 8.1 show the results of making these adjustments to the head-to-tail paths for a horse and a human figure.

The head section of the path is fixed as a rigid segment of the axial control skeleton, and the rest of the modified head-to-tail chain is then divided into a series of vertebral segments, each having the same length. Experimentally it has been

determined that having segments that are approximately one third the length of the head segment provides a reasonable number. This seems to allow good flexibility of the spinal column while not overwhelming a user with the otherwise large number of joints that a real spinal column would possess. A joint is created between each consecutive pair of vertebral segments as well as between the head segment and the most anterior vertebral segment. An additional joint is created at the midpoint of the vertebral segment that best corresponds to the location of the midpoint of the hindmost pair of limbs: this joint serves as the root joint for the control skeleton. The root joint and the vertebral joints each have have three rotational degrees of freedom (DOFs) by default. The axes for these DOFs form an orthonormal basis. The z-axis points tangentially along the smoothed head-to-tail path in the direction away from the root joint, the y-axis points in a distal direction, normal to the head-to-tail path. The x-axis points laterally to complete a right-handed coordinate system. Movement about the three axes can be constrained by setting fairly restrictive joint limits. The segments and joints created for the horse are visible in Figure 8.2 in the next section.

Within the head of the figure, a jaw segment is created if the head portion of the level graph reveals the presence of a jaw. In animal-like figures, this means that the head section of the level graph has a branch that extends downward and outward from the main line running to the nose region; for human-like figures, any jaw branch is expected to extend forward from the main line running to the tip of the head. The jaw segment is attached to the head segment by a single joint, the parameters of which are set so as to permit hinge-like motion about one axis. This completes the formation of the axial portion of the control skeleton.

8.2 The Appendicular Skeleton

The appendicular portions of the control skeleton are formed in a template-based manner depending on a few characteristics: whether the figure is human-like or animal-like; whether a particular pair of limbs has been classified as arms, legs, or wings; and whether a particular pair of limbs has separated digits. The basic approach is to create a partial control skeleton in independent fashion for each pair of limbs and then to attach it to the axial control skeleton.

The first step in the processing of a pair of limbs is to determine the type of girdle it needs for attachment to the axial skeleton. Based on the discussion in Chapter 6, two types of girdles are possible: a pectoral girdle and a pelvic girdle. The pectoral girdle is modeled as two independent segments corresponding to the two independently mobile scapulae. The pelvic girdle is modeled as a rigid extension to the vertebral segment to which it is attached. Arms, wings, and the foremost pair of legs (provided there are at least two pairs of legs) are provided with pectoral-style girdles, single pairs of legs or all leg pairs but the first are modeled with pelvic girdles.

The method for creating the girdle segments differs for human-like and animal-like figures. Both methods involve examining the sphere for the voxel on the original (centralized) head-to-tail path closest to the midpoint of the limb-to-limb path. The limb-to-limb path is assumed to intersect this 'girdle sphere', and for most figures with fairly circular cross-sections, the assumption holds. For figures with more oblong cross-sections, the limb-to-limb path may not intersect the girdle sphere. For simplicity in this particular implementation, however, the single girdle sphere is assumed to suffice, and the limb-to-limb path is assumed to intersect that sphere.

For human-like figures, the shoulder joints and the hip joints are modeled as being at the voxels of the limb-to-limb paths just outside the reach of the girdle sphere. Any limb segments created are thus exterior to the girdle sphere.

For animal-like figures, the process is more involved. Instead of becoming places for shoulder or hip joints, limb-to-limb path voxels just outside the girdle sphere are used as positions for the knee joints or the elbow joints of the limb pair.[16] The shoulder joint locations or the hip joint locations are then computed to lie inside the girdle sphere. These interior joints are positioned such that they roughly preserve the width between the outside joints (that is, the elbow or knee joints), so that they allow for the scapular/coxal segment and the humerus/femur segment to be of roughly the same length, and so that they are offset in the anterior direction for pectoral girdles or in the posterior direction for pelvic girdles (see Figure 8.2). This method of positioning the shoulder and hip joints was determined experimentally, and it appears to work reasonably well in most cases.

The portions of the limb-to-limb paths outside the girdle sphere are then smoothed and divided into segments. The segmentation depends on whether the figure has been identified as human-like or animal-like and also on whether the limbs stem from a pectoral-style girdle or a pelvic-style girdle. Each separate limb path exterior to the girdle sphere is divided according to the ratios presented in Table 8.1. The ratios are meant to correspond to the relative lengths of the various bones from a limb of that type. Note that in the segmentation of the limb paths for animals, the humerus/femur measurement is not used, since that segment is already accounted for

[16]The motivation for this comes from a heuristic in artistic anatomy: the elbow and knee joints of quadruped animals are located approximately at the same height as the line of the creature's belly.

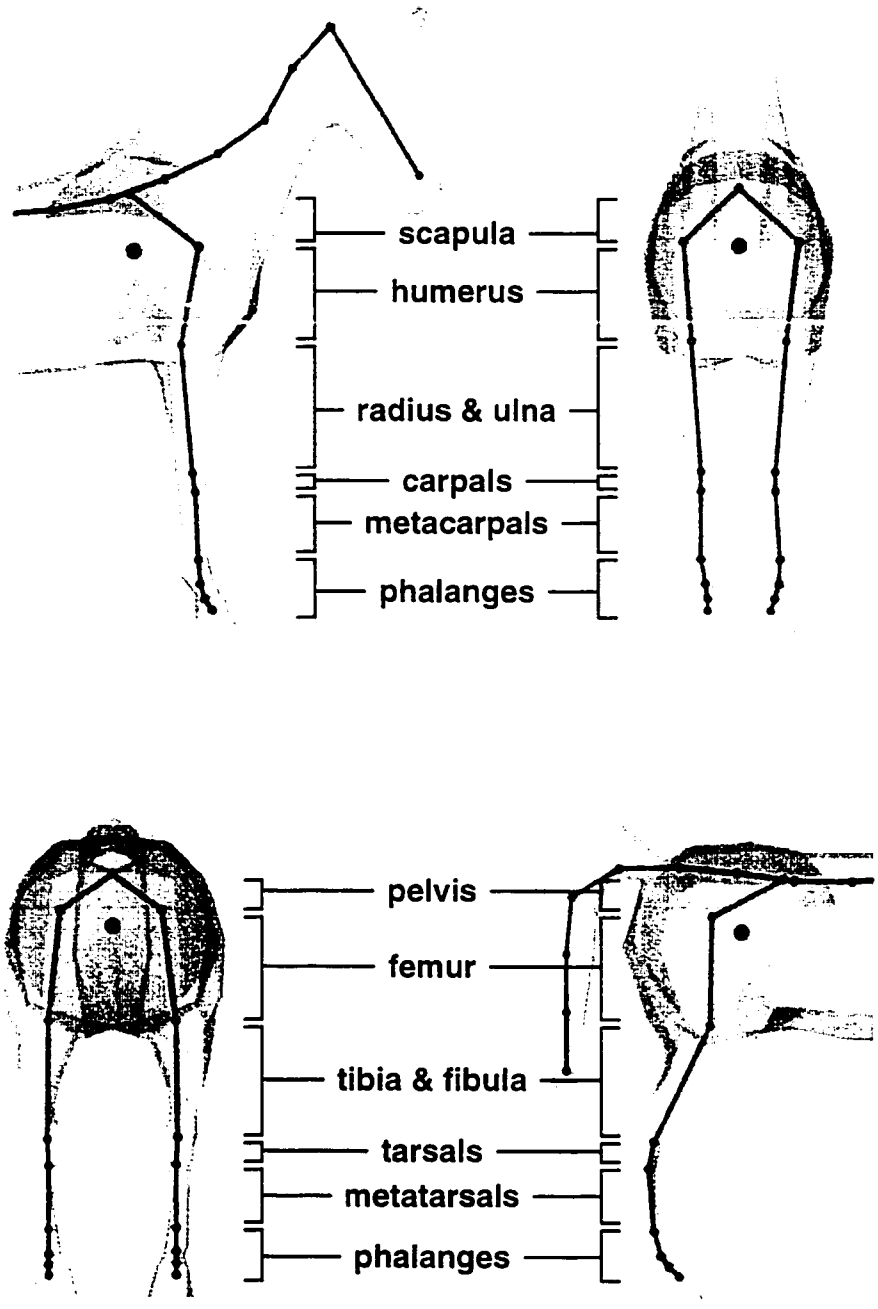


Figure 8.2: The girdle spheres and the control skeleton for the horse. The pectoral girdle sphere gives rise to the segments corresponding to the scapulae and the humerus for each forelimb; the pelvic girdle sphere is used to generate the segments corresponding to the pelvis and the femur of each hind limb. The remainder of each limb is segmented based on the data in Table 8.1.

Bones within the Segment	Animal Forelimb	Animal Hind Limb	Human Arm	Human Leg
humerus/femur	149	175	190	180
radius-ulna/tibia-fibula	154	174	150	160
carpals/tarsals	22	35	20	30
metacarpals/metatarsals	78	89	36	30
proximal phalanx	29	29	25	12
middle phalanx	19	19	15	5
distal phalanx	16	17	10	5

Table 8.1: The ratios used in the segmentation of the limbs. Each column shows the relative lengths of the segments for that type of limb. Note that the data are relative only within a column and not between columns. The data were derived from analyses of images in various references on anatomy [EBD56, Par88, Mad94].

from the processing of the girdle sphere. Figure 8.2 shows the results obtained for a figure of a horse.

Recall that an analysis of appropriate regions of the level graph (described near the end of the previous chapter) is used to determine whether a limb has a single digit or multiple digits. If a limb is identified as having multiple digits, then further examination of the branching structure of the region is performed to label one of the level graph vertices as a wrist or ankle vertex.

For a limb with only a single digit, the segmentation is fairly straightforward, as the whole limb path can be partitioned according to the ratios in Table 8.1. For limbs with multiple digits, however, the segmentation occurs in phases - one phase for the the portion of the limb proximal to the wrist or ankle, and an additional phase for each digit the limb has. In the phase for the proximal portion of the limb, that part of the path is divided into two sections for animal-like figures (the carpus/tarsus and the lower arm/leg) or into three sections for human-like figures (the carpus/tarsus,

the lower arm/leg, and the upper arm/leg). The relative sizes or these sections are again based on the data in Table 8.1. Before the segmentation can occur for the digits, corresponding voxel paths must be generated. For each limb, the DMS voxel set corresponding to the wrist/ankle vertex is examined to determine the deepest DMS voxel (the one with the greatest distance map value). Similarly, the DMS voxel set corresponding to the level graph vertex at the end of each digit is examined, and the DMS voxel with the greatest heart value is chosen as the end-effector point for the digit. For each digit, a centralized path is computed between its end-effector point and the wrist/ankle voxel. The digit closest to the centerline of the body is assumed to be the first digit (corresponding to the thumb or the big toe). Each digit path other than that of the first digit is divided into four segments whose relative lengths, ordered distally to proximally, conform to the ratio 10 : 15 : 25 : 36 (see Table 8.1). The first digit is divided into only three segments according to the ratio 15 : 25 : 36. Examples of the skeletonization for multiple digits is seen in Figure 8.9 on page 193.

In vertebrates with wings, the bones of the wing generally lie along the anterior edge of the wing. In the implementation, when a limb has been identified as a wing, a post-processing step is performed to shift the joint locations for the wing to the anterior edge of the wing. Results of this forward shifting are apparent in Figures 8.7 and 8.10.

As alluded to in Chapter 6, the joints for the limbs can be constrained to rotate according to typical patterns. Shoulder and hip joints are given three DOFs with axes aligned appropriate to the adjacent segments, and elbow and knee joints are effectively made into hinge joints by allowing no range of motion about two of the three axes of the three DOF joints. Wrist/ankle joints are given three DOFs, a simplification

that combines the roll of the radius and ulna (or tibia and fibula) with the yaw and pitch of the wrist/ankle joint of humans and animals. Joints between digital segments are constrained as hinge joints, and joints between the digital segments and the carpus/tarsus are constrained as hinge joints except for the joint for the first digit, which is constrained to have two rotational DOFs (no roll).

8.3 Attachment

After the segments and joints of the control skeleton have been constructed, the polygonal data for the given model must be anchored to that structure. The basic process is similar to that for the more general algorithm (described in Section 5.2.5): the first step consists of determining which parts of the voxelization will be influenced by each control segment, and the second step involves setting up a weighted summation of a set of locally defined anchor points to be used to recompute the global position of each vertex of the model.

Each segment has a corresponding list of voxels forming the core of its region of influence within the voxelization. Applying the inverse distance transform to these voxels generates a set of spheres, and any voxels interior to any of those spheres are marked as being under the influence of the segment. In the general algorithm described in Chapter 5, the core voxels for a particular segment's region of influence come directly from the chain of the path tree from which the segment had been derived. Since the path tree is constructed from the DMS, the core voxels are DMS voxels.

For the algorithm described in this chapter, the core voxels are selected in a different fashion. Recall that a head-to-tail chain of DMS voxels is computed during

the construction of the axial portion of the control skeleton. After the vertebral segments and the cranium segment are constructed, the voxels of the head-to-tail chain are partitioned into sets corresponding to which of those segments is closest to each voxel. These sets are used as the core voxels for each segment's region of influence. For the jaw segment (if it exists) and for each segment of the appendicular portion of the control skeleton, the core voxels are found by intersecting each segment with the voxel grid. Using the jaw as an example, the line segment from the mandibular joint to the tip of the jaw passes through the set of voxels that form the core of the region of influence for the jaw. As in the general algorithm, the core voxels are then expanded into spheres according to their distance values to help form the region of influence for a segment.

The formulation of the weighted sum is similar to the formulation from Section 5.2.5. Each vertex of the model lies within a voxel, and that voxel is contained in one or more regions of influence. The containing voxel thus dictates which segments will affect a particular vertex when the control skeleton is animated. Each influencing segment lends a term to the weighted sum for the position of the vertex, and that term is the product of an anchor point fixed within the local frame of the segment and a weight. The weight is derived from the relative proximity of the vertex to that particular segment.

8.4 Results

Overall, the results of the anatomically based method of generating control skeletons are promising. For many human and animal figures, the assumptions built into

the implementation work quite well. The control skeletons have a noticeable anatomical quality to them, and this closer adherence to anatomy serves to allow more natural looking motion of the figures.

The algorithm depends on a generalized model of the anatomy of humans and animals. Because of this, the control skeletons produced by the algorithm for a specific model may not be as anatomically accurate as a user might desire. Nevertheless, the control skeletons produced are generally of sufficient quality to function as an initial skeleton worthy of manual tweaking.

Because the algorithm uses heuristics based on human and animal anatomy, a comparison of the results with the actual anatomy of such creatures seems in order where possible. A few comparisons pertain to many of the figures. Anatomically speaking, each of the figures would have a jaw. If the jaw is not discernable as a separate protrusion of the voxelization, however, then no jaw segment is generated by the algorithm. The same comment holds true for the digits of creatures, and this explains why some of the hands, feet, and paws seen in the diagrams have only one digital extension of the control skeleton when the true anatomy would reflect several. As mentioned previously, a compromise has been made with respect to the number of spinal segments in the control skeleton. The anatomy would argue for many more spinal segments (and that they be of varying length according to body region), but any benefit of having as many as the anatomy would dictate would probably be overshadowed by the extra time consumed by an animator to control them all. Each of the following paragraphs will make specific comparisons with regard to one of the anatomically based control skeletons generated by the algorithm (Figures 8.3, 8.5, and 8.7 through 8.10).

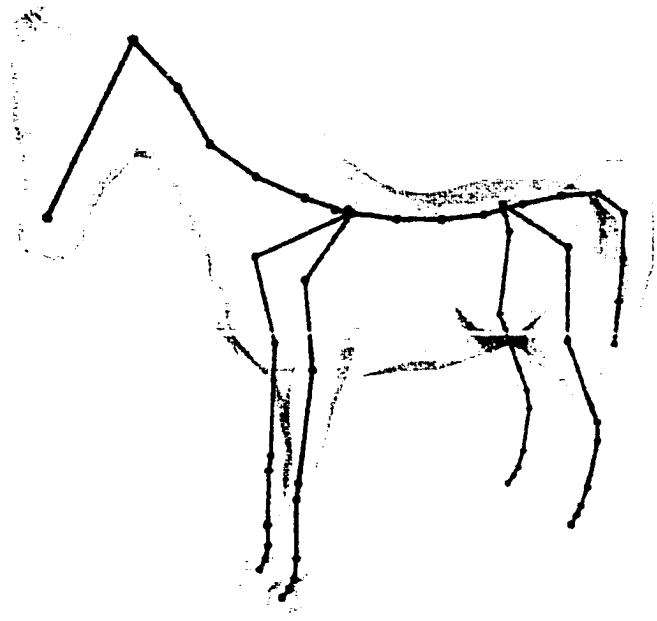


Figure 8.3: Anatomically based control skeleton for a horse. For comparison with actual horse anatomy, examine Figure 9.2 on page 202.

The control skeleton structure generated for the horse is shown in Figure 8.3. Figure 8.4 presents a few poses of the horse using this control skeleton. To agree more with the real anatomy of a horse (see Figure 9.2), a few changes would be necessary. The joint proximal to the head segment should be farther forward, slightly behind the ears. Also, the vertebral segments in the thorax should extend farther forward before the S shape of the cervical vertebral segments is realized. The scapular segments should be farther forward as well so that the shoulder joint is at the front of the figure. Finally, the segments in the legs should extend farther into the hoof regions of the feet, and the segments in the tail extend too far to be anatomically accurate, since the bulk of a horse's tail is hair. For control purposes, however, an animator would probably want the tail segments to extend as far as they do in Figure 8.3.

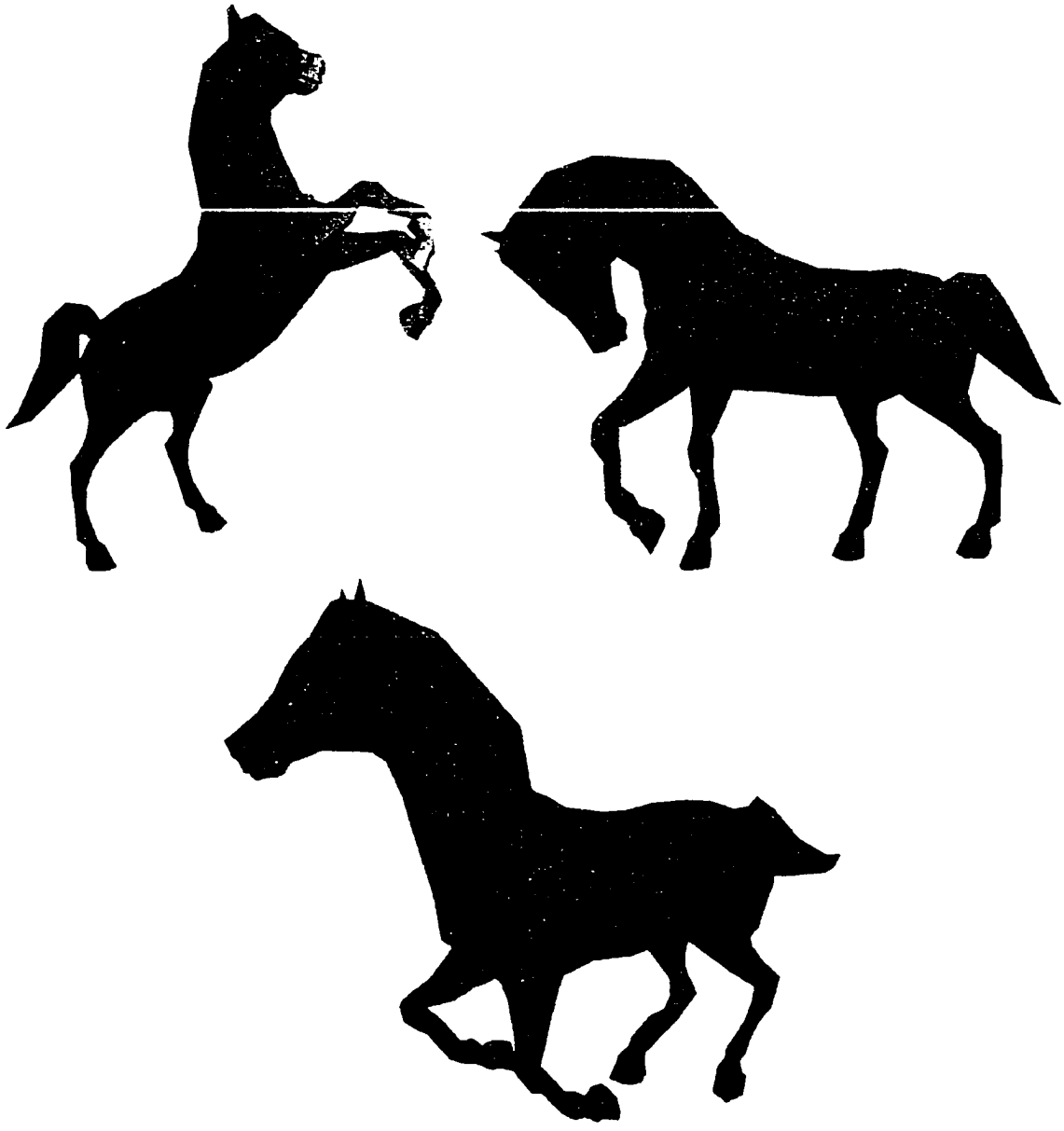


Figure 8.4: The horse in various poses. Each pose is the result of manually assigning values to the joint angle parameters of the control skeleton in Figure 8.3.

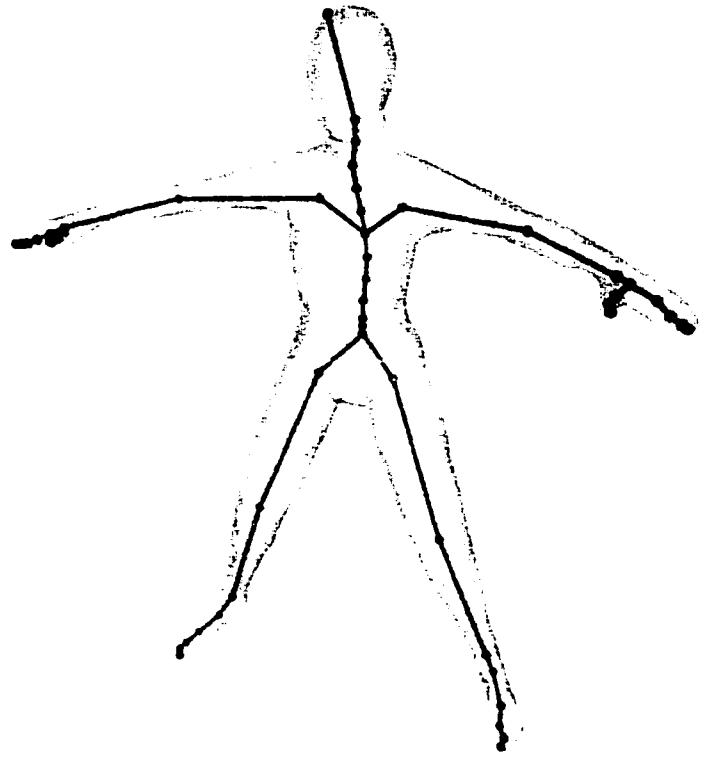


Figure 8.5: Anatomically based control skeleton for a human figure.

Figure 8.5 shows the skeletal structure generated for a human figure, and a few related poses of the figure are seen in Figure 8.6. In comparison to human anatomy, the shoulder joints in the figure should be farther away from the spine and higher up in the body as well. Note that the control structure has been built using segments corresponding to the scapulae instead of segments corresponding to the clavicles. Using clavicular segments is probably the more common method when a control skeleton is manually designed. The decision to use scapular segments in the implementation was based on the generalization that all tetrapods have scapulae, whereas not all tetrapods have clavicles.

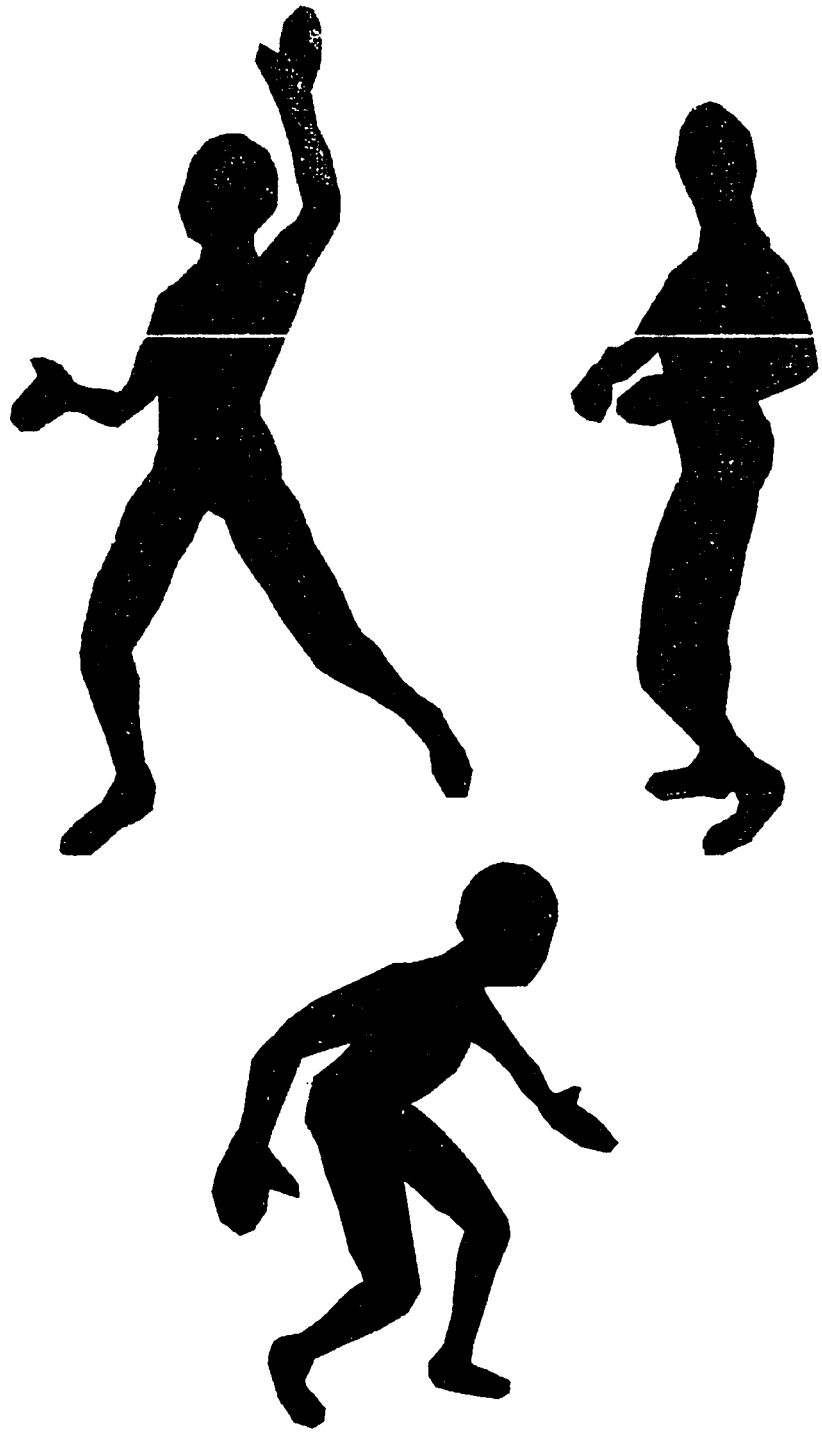


Figure 8.6: A human figure in various poses. Each pose is the result of manually assigning values to the joint angle parameters of the control skeleton in Figure 8.5.

Figure 8.7: Anatomically based control skeleton for a bird.

The anatomically based control skeleton for a bird can be seen in Figure 8.7. Even though the model is not an anatomically accurate model of a bird, a few things are worth noting. The real anatomy of a bird has much less flexibility than the control skeleton shown. This is because the thoracic, lumbar, and sacral vertebrae of a bird, as well as its sternum, ribs, and pelvis, are typically fused into two large, rigid bony masses. Discussion of the control skeleton in the tail region of the bird parallels that of the horse's tail. The hindmost portion of a bird's tail is populated by feathers, not by bones and tissue. Lastly, the knee joints in the figure should be farther forward, and the ankle joints should be higher up in the legs and farther toward the rear of the legs as well.

Figure 8.8 shows the skeleton generated for an asymmetrically posed dog. The joint proximal to the head segment is obviously incorrect – it should be at the rear

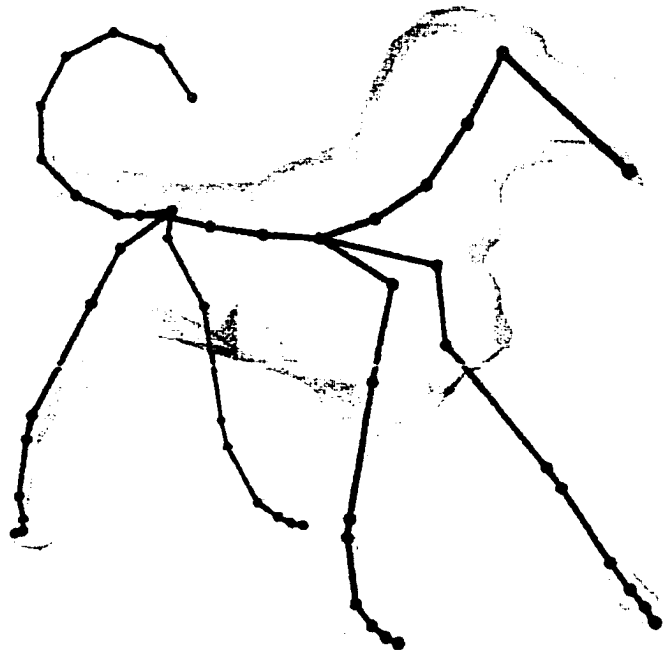


Figure 8.8: Anatomically based control skeleton for a dog. Note that the asymmetric pose was present in the given model before it was processed by the algorithm.

of the head and not quite as high as it is. To better agree with a real dog's anatomy, the segments corresponding to the femur, tibia and fibula, humerus, and radius and ulna should be longer, and the metatarsals should be shorter. These changes would result in the ankles being lower and thus more accurately placed. A dog also has multiple digits, so the earlier comment about having a single group of digital segments also pertains here. The scapular segments should be farther forward, and the pelvis (represented as the two segments joining together just above the spine) should be farther back – this includes the point at which it joins the spine as well as the hip joints. The knee joints should be lower. Note that the caudal vertebrae of a dog do typically reach to the tip of its tail.

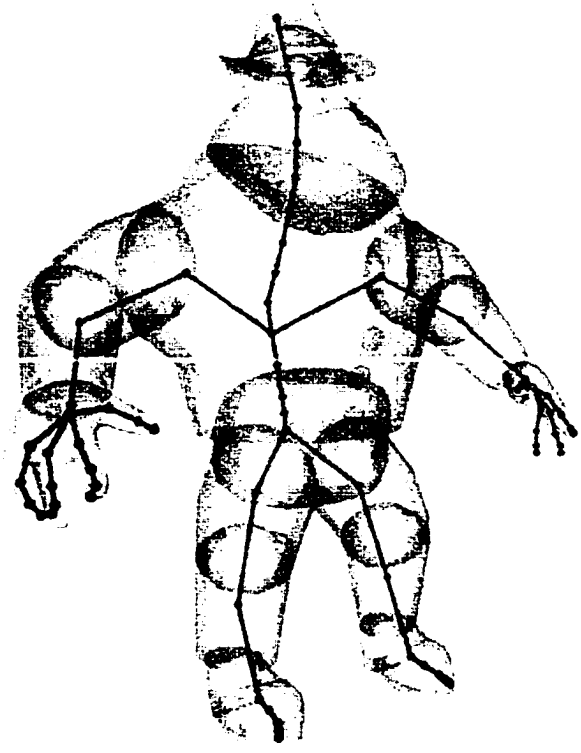


Figure 8.9: Anatomically based control skeleton for a cartoon-style human figure. Note that the model is segmented: the darker regions show how the different parts of the segmented model overlap.

The control skeleton generated for a cartoon-style human figure is shown in Figure 8.9. With the model having cartoon proportions (and non-anatomical extras such as the hat), the anatomical heuristics are likely not to work as well. Given the skeletal structure shown, an animator would likely desire a few changes. The shoulder joints, the hip joints, and the knee joints should be raised, and the elbow joints should be pushed farther back toward the elbow's typically bony protrusion. The ankle joints should be moved into better position at the bottom of the shanks, thus lengthening the foot segments as well. The head segment should be longer, and this would perhaps be best accomplished by folding the top two vertebral segments into that of the head.

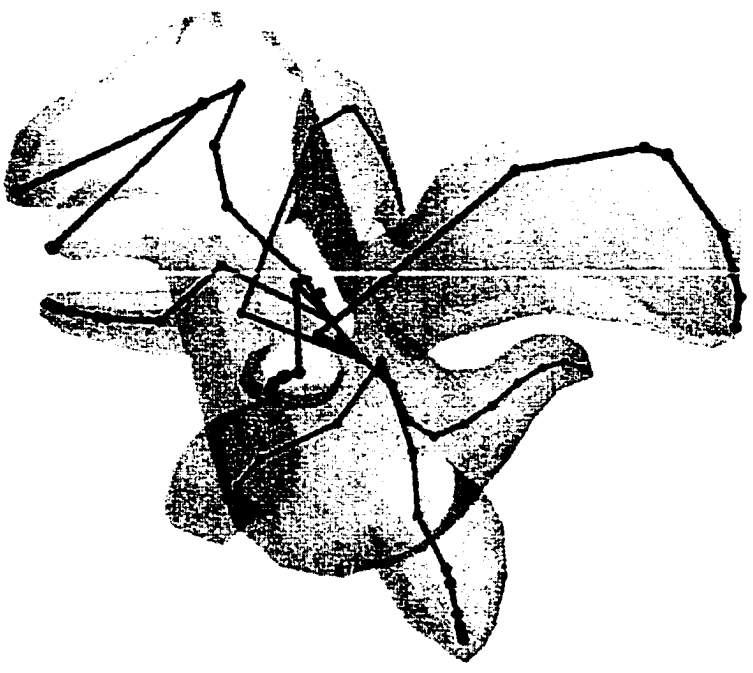


Figure 8.10: Anatomically based control skeleton for a dragon.

Of final note, the joints proximal to the metacarpal segments are coincident (the same holds for the hands in Figure 8.5). Whereas this is probably seen as acceptable for animation purposes, it is not anatomically accurate, since anatomical joints are never coincident.

The final control skeleton shown is that of a dragon (Figure 8.10).[17] Since it is a mythical creature (and modeled in a cartoon style as well), its anatomy is unknown. From an animation stance, the vertebral segments in the neck region of the figure should probably be farther toward its dorsal side, and the shoulder joints should probably be closer to the wings. Finally, the number of segments in the arms and

[17]The dragon model is available with the Teddy package [IMT99, Iga99].

legs could probably be decreased, as the digital segments are perhaps unnecessary for the model.

The processing time required for the models ranges from about five seconds (in the case of the horse) to about two and a half minutes (in the case of the human figure). The computation was performed on a Silicon Graphics O2 (R5000 Processor Chip). Note that these times include the generation of the component models for bones, which is discussed in the next chapter. As with the algorithm in Chapter 5, the computation time is primarily dependent on the number of interior voxels used to approximate the models.

A few more general points should also be noted:

- If the limb sections of a model (in the eyes of a user) have strange proportions (perhaps incredibly long forearms), then the segmentation resulting from the application of the ratios in Table 8.1 may not be appropriate. An example of this is seen in the legs of the cartoon character in Figure 8.9. Also, for a model of some mythical creature whose hypothetical anatomy does not closely resemble that of most vertebrates, the control skeleton generated may or may not be similar to what the user had expected. In general, the more closely a model conforms to the constraints listed in Chapter 7, the better the results of the algorithm are.

- With respect to assigning both joint frames and joint limits, the algorithm produces mixed results. Correct or appropriate assignments of joint frames and limits requires having the given figure in a default pose. The constraints listed in Chapter 7 are not stringent enough in this regard; however, making them more strict would limit the applicability of the algorithm.

For animal-like figures, the implementation has assumed a default pose similar to that of the horse in Figure 8.3; for human-like figures, the default pose is similar to that of the model in Figure 8.5. When the pose of a given model deviates from the default, the joint frames for the generated skeleton may not align in a convenient fashion (for instance, the hinge axis for a knee or elbow joint may not be aligned with one of the three axes for the knee or elbow joint). Of course, if the joint frames are not set up properly, assigning joint limits makes little sense.

Note that if the default poses were required of input models, then the algorithm could be redesigned to take advantage of the symmetric nature of the default poses. Instead of computing the axial skeleton as described in Section 8.1, for instance, the algorithm could examine the cross section of the model in its plane of symmetry. Thus, 2D techniques could be used to generate the axial skeleton, leading to easier computation with more desirable and predictable results. The process of pairing limbs would also be made easier.

- As mentioned previously, some assumptions have been hard-coded into the implementation. An example of where this has a negative effect is seen in Figure 8.8: the neck region for the dog is fairly short in comparison to its head, so arbitrarily marking the head/neck division as described in Section 8.1 produces poor results for that region of the control skeleton. Again, a more flexible system could offer the user choices that could alleviate such problems, but the intent here has been to gather experience on how much can be done without requiring such assistance from the user.

- One could obviously argue the need to expand the girdle sphere into a shape that more closely matches the cross-section of the trunk at the location of the girdle. This might allow for a more robust treatment of the girdle sections of the control skeleton. It would probably not be too difficult to extend the implementation in such a manner, perhaps modeling the cross-section region as a collection of spheres centered on the medial axis of the cross-section.

Clearly anatomically based techniques are not always appropriate. When more realism is called for, the algorithm provides a useful starting point. When less realism is desired, such as with a cartoon-style figure, anatomically based techniques may be of only limited usefulness. The next chapter delves more deeply into anatomically based modeling, describing how component models of bones can be produced in automated fashion to enhance the anatomically based control skeleton.

CHAPTER 9

CREATING ANATOMICAL COMPONENT MODELS

This chapter discusses how component models of anatomical systems such as the skeleton and musculature can be used to help flesh out a control skeleton into a layered deformable model for a figure. Section 9.1 presents a generalized skeleton model and describes how it can be grafted onto an anatomically appropriate control skeleton as computed in the previous chapter. Results and possible improvements are discussed. Section 9.2 proposes how a similar idea might be applied in generating a musculature for an arbitrary human-like or animal-like figure. Finally, Section 9.3 briefly describes the role of modeling other supporting tissue and a skin surface.

9.1 A General Skeleton Model

The general model that has been implemented is fairly simple. It consists of a set of normalized bone models derived from data models of human skeletons. Models of human skeletons are more readily available than models of animal skeletons, and except for a few special cases, most animal bones can be approximated fairly well by using modified versions of human bones.

The model contains both axial and appendicular sections. The components of the axial skeleton include a cranium and mandible for the head, a generic vertebra, and a

rib cage. The components of the appendicular skeleton consist of two parallel groups of bones, a pectoral group and a pelvic group.

The cranium and mandible are created and scaled based on the length of corresponding segments of the control skeleton; their widths and heights are computed using distance map values in an attempt to occupy as much interior space in the head as possible. For the vertebral column, the approach is similar. The vertebra is instanced once for each vertebral segment of the control skeleton and scaled in an appropriate manner. The length of the instanced model is determined by the length of the vertebral segment. For neck and trunk vertebrae, the width of a vertebra instance is one twelfth of the diameter of the largest sphere within the trunk of the figure (computed by examining distance map values). For tail vertebrae, the width begins with the same measure as the trunk vertebrae but tapers off in a geometric progression with a decrease of 10% between each successive pair of vertebrae. The specific implementation does not necessarily correspond in any anatomical or biological sense, but it provides visually reasonable results; the intention is merely to mimic the tapering evident in the tail bones of real skeletons.

For the rib cage, instantiation is more involved. The girdles of limb pairs that are arms, wings, or forelegs are assumed to be pectoral girdles (all other leg girdles are treated as pelvic girdles). The girdle spheres for the limb pairs with pectoral girdle are examined to determine how many there are and whether any of them overlap. The spheres are partitioned into groups according to which ones overlap, and a rib cage is created for each group. Each rib cage is set up to be in agreement with the vertebrae to which the ribs would be attached; the sizing of the rib cage is determined by the by the size of the girdle sphere or group of spheres.

For limbs with a pectoral girdle, instantiation of numerous bone models is performed. Each pectoral limb is provided with a scapula, a humerus, a radius and an ulna, a group of carpals, and, for each digit, a metacarpal and a set of phalangeal bones. These bones are scaled to lengths to correspond to their respective segments of the control skeleton, and for all but the scapula, to widths according to distance map values. The width of the scapula is calculated as half its length, and its orientation is computed so that the scapula has an approximately tangential relationship to the rib cage filling out its corresponding girdle sphere.

Limbs with pelvic girdles are instantiated in a similar manner as the pectoral limbs. They are provided with scaled versions of the femur for the thigh segment and the tibia and fibula as well as the patella for the shank segment, and a group of tarsals for the ankle segment. Each digit is provided with a metatarsal and two or three phalangeal bones. The pelvis is instantiated as a single bone fixed within the coordinate space of the nearest vertebral segment, positioned and scaled so as to place the sockets of the pelvis roughly at the locations of the hip joints of the control skeleton.

9.1.1 Results and Discussion

Figures 9.1 and 9.3 through 9.5 show the results of instantiating bone models for various control skeletons from the previous chapter. Scaling the components of a human skeleton and placing them within the segmented hierarchy of an anatomically based control skeleton produces acceptable results, but there is room for improvement. Earlier results confirmed that simply scaling a human skull to appear as the skull of an animal, for instance, produced little more than a strangely scaled version of a human

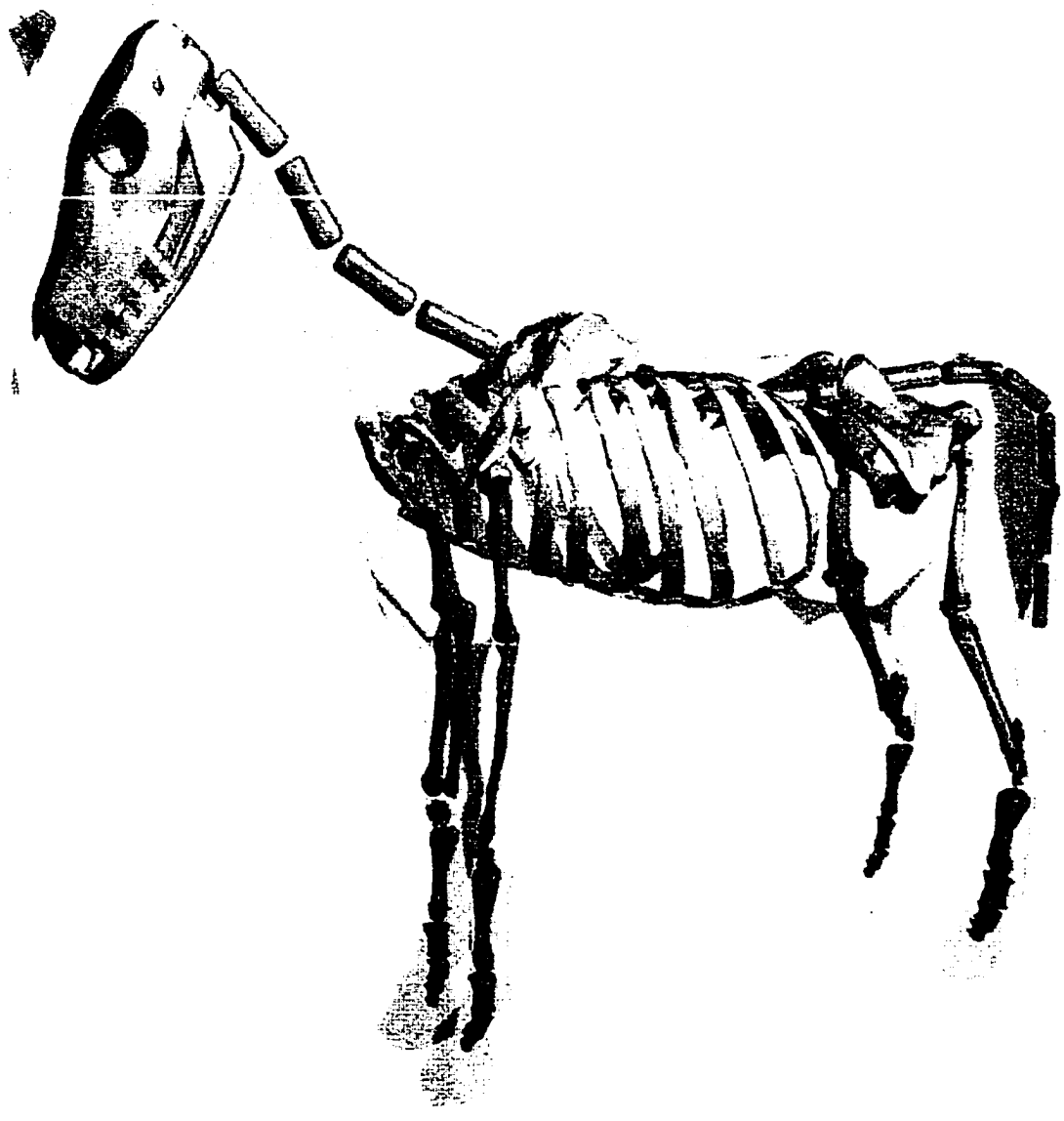


Figure 9.1: Bone models generated for the skeleton of a horse. See Figure 8.3 for the corresponding control skeleton. Compare the image above with the anatomical illustration in Figure 9.2.

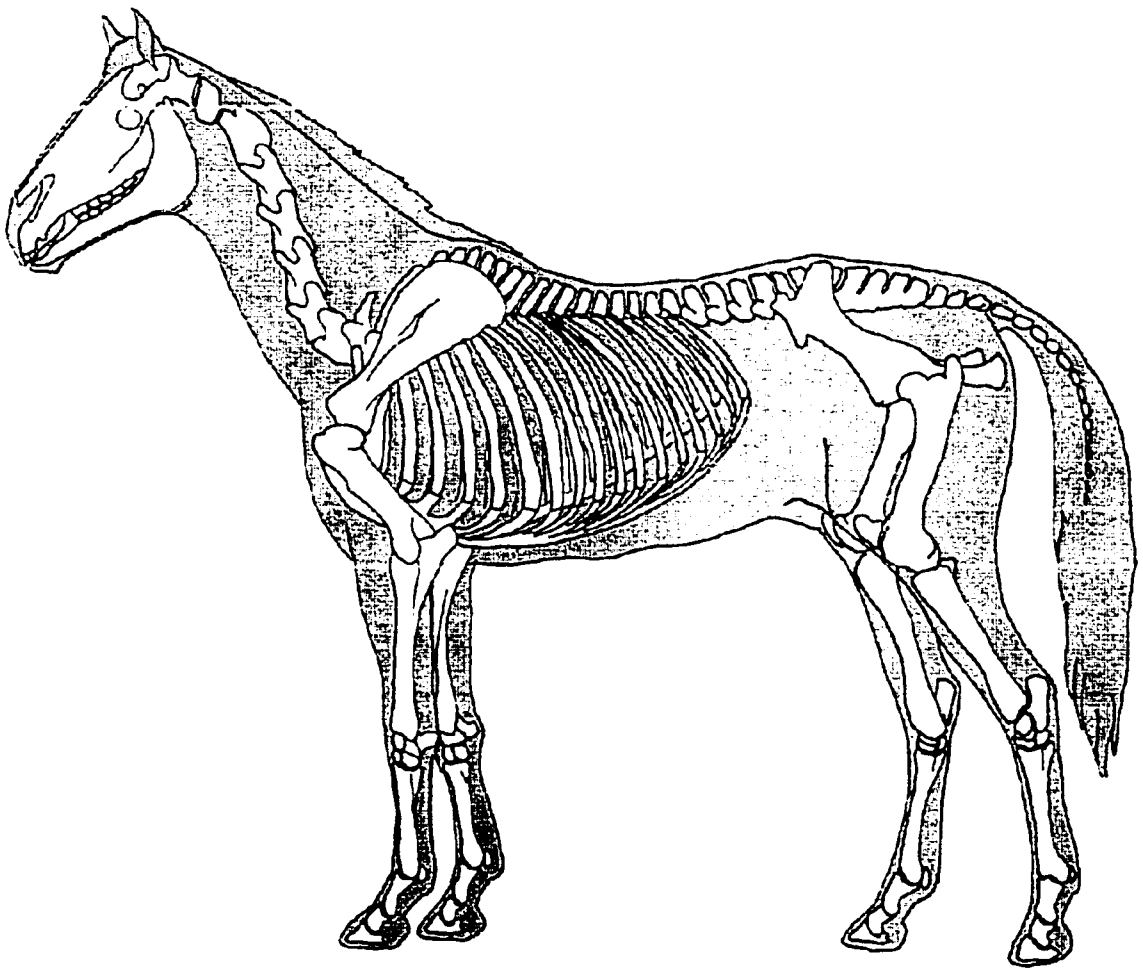


Figure 9.2: Anatomical illustration of the skeleton of a horse (based on a figure in [EBD56]).

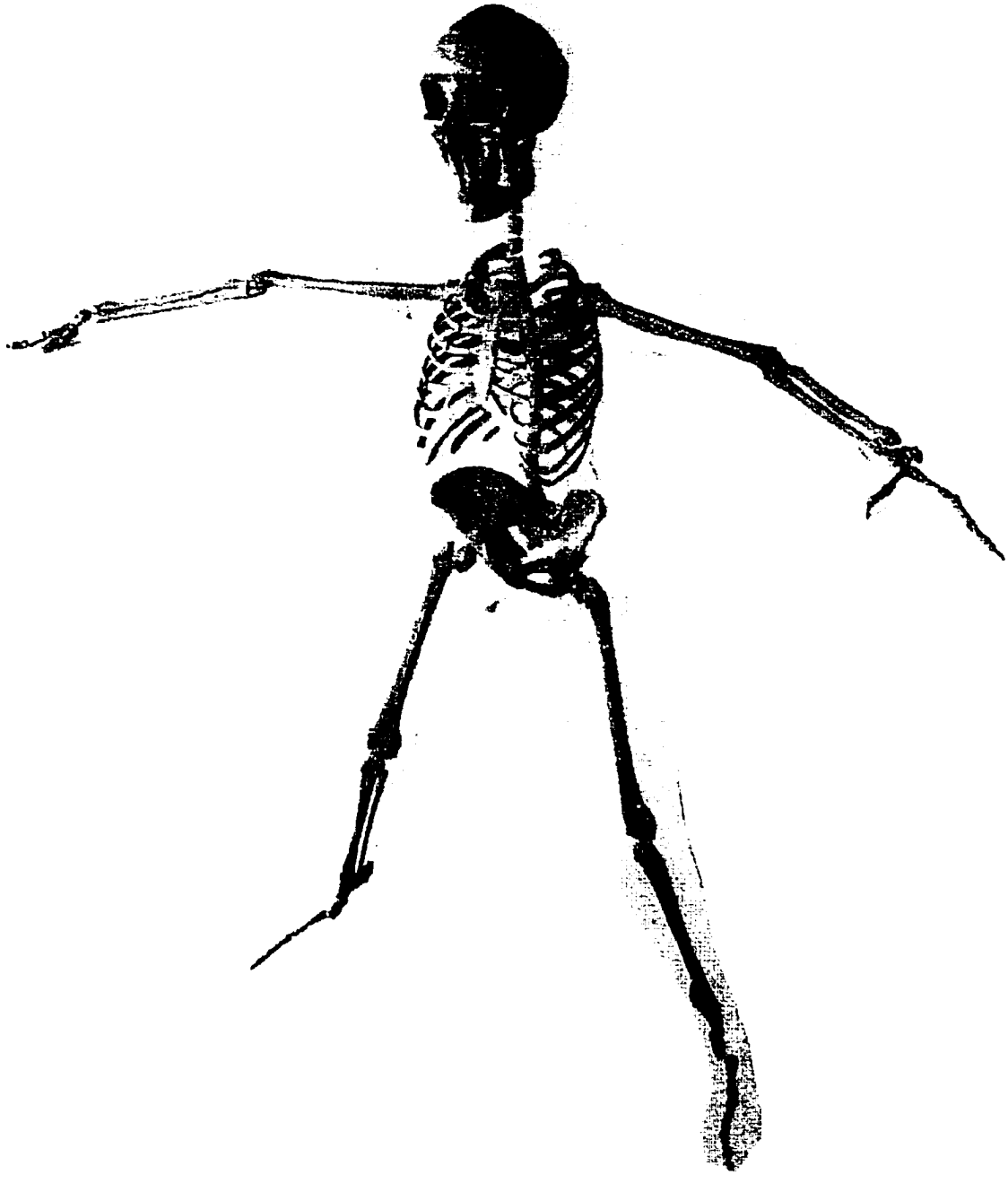


Figure 9.3: Bone models generated for the skeleton of a human figure. The corresponding control skeleton can be seen in Figure 8.5.

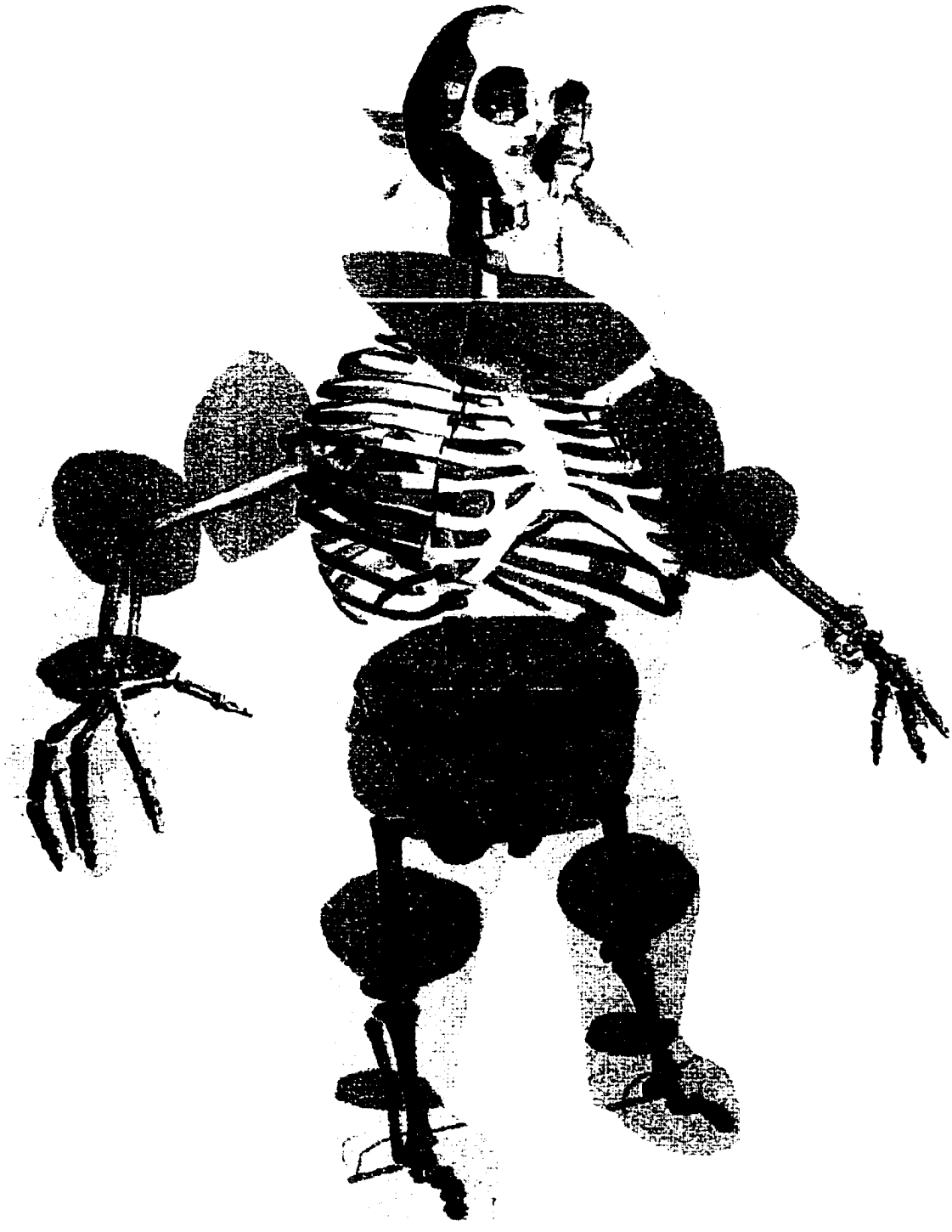


Figure 9.4: Bone models generated for the skeleton of a cartoon-style human figure. Figure 8.9 shows the underlying control skeleton.

Figure 9.5: Bone models generated for the skeleton of a dragon. The control skeleton itself is shown in Figure 8.10.

skull with very little resemblance to the expected animal skull. For that reason, a generalized animal skull has been modeled and used for the animal-like figures shown here. Creating additional bones for the hands and feet of the human in Figure 9.3 and for the feet in Figure 9.4 could generate pleasing results, but such extras might not animate well when fixed to the single digit control skeleton for that portion of the hand or foot.

Typically, any particular bone of a creature (a skull or a femur, for example) is recognizable as an instance of that particular type of bone even though it may come in a variety of distorted forms specific to the kind of animals from which it came. This recognition is due primarily to the features that the bone possesses (the eye sockets and nasal cavity of the cranium, or the rounded head of the femur that helps form the ball-and-socket joint of the hip). Standard scaling of bone models along three independent axes often causes too much distortion of these features (in the case of the femur, for instance, the rounded head might take on too much of an ellipsoidal shape).

For more realistic skeletal models, several things could be changed. First, individual bones of the system might best be described in a procedural fashion. Local parameters could then be specified that would determine not only the length and girth of the bone's basic shape but also the positions and sizes of the defining features of the bone. Such feature-based rescaling would allow better reproduction of a bone group like the cranium or of single bones like the femur.

The entire skeleton could be set up in a modular way with procedurally defined groups of bones. The axial skeleton could be instantiated through the use of various global parameters to have more or fewer cervical, thoracic, lumbar, sacral, and caudal

vertebrae, as well as to have more or fewer ribs. Other global parameters could specify the sizes of the chest cavity (to determine the overall size of the rib cage and each rib in it) and the head (to determine the overall and relative sizes of the cranium and mandible), and still others could specify which individual bones might be fused together. Such global parameters would provide for recommended sizes of bone groups and the bones within them, local parameters could then be specified for individual bones if it were necessary to override or adjust the global defaults.

Such parameterization could also be used for the appendicular skeleton, the implementation of which would be capable of generating individual limbs. For each limb generated, a half-girdle and an attached five-segment limb could be formed. Parameters might allow the specification of the relative sizes of the girdle bones and of each segment, as well as their default girths. Other parameters could dictate how many digits would be present on each limb to enable the production of anything from a horse's limb skeleton (one digit) to a human's pentadactyl limb skeleton. Boolean parameters could be set as to whether a limb was to take a forelimb or hind limb orientation (note that the elbow of vertebrates generally points caudally and the knee cephalically) and as to whether a limb was on the left or right side of the body. General functionality of the limb might also be specified as a parameter. For planted "hands" or feet, this could enable separate instantiation of plantigrade (flat-foot, like a human), digitigrade (walking on curved digits, like a cat), and unguligrade (walking on fingertips, like a horse) feet. Such a parameter could alternatively specify that that limb was to be a wing or non-supporting arm.

There are a few problems with such an approach, however. If the goal is to have an automated algorithm, then the large number of parameters that would need

to be incorporated would likely make an implementation infeasible. Automatically assigning values to such parameters might be possible for a very constrained set of figures (a horse, for example), but that would limit the usefulness of the system for fleshing out other figures.

Another problem concerns the amount of realness necessary for a given figure. After all, how appropriate is it to create realistic sets of bones, muscles, or other internal tissues for a cartoon-style person or animal?

9.2 A General Musculature Model

Though no modeling of the musculature has been implemented in the course of this research, it is not hard to imagine how a generalized musculature could automatically be generated. The following discussion draws from the analysis in Chapter 6 and some of the research described in Section 2.4.

A full set of muscles for an arbitrary vertebrate would probably be too difficult and time-consuming to model, not to mention too much of a computational burden: simplifications and generalizations seem called for. The musculature model would depend upon the underlying model of the skeletal system, so it might work best to have muscle groups defined in a procedural fashion relative to the locations of the individual bones of that system. Each generic bone model could have default locations for the origins or insertions of the muscle models that would attach to that bone. These default locations would relate to the patterns of muscle footprints on the bones as identified in the comparative anatomy literature.

As for the actual modeling of any individual muscles, an approach such as that of Scheepers [Sch96, SPCM97] would likely be sufficient. This would allow for instantiation of relatively simple fusiform muscles or for the more complex arrangements of muscle groups in the chest and back.

For the axial muscles, a standard pattern of both oblique and longitudinal muscles could be created along the spine (the epaxial muscles) that would respond to arching or to lateral bending of the vertebral column. Each muscle would originate on one vertebra and insert on one or more other vertebrae or possibly on the base of the skull. Simplified jaw muscles could also be modeled. A pattern of body wall muscles (the hypaxial muscles) could also be generated, though it should be noted that since these muscles function mainly to contain the innards of the trunk, it may be necessary to have a simple model of the innards as well.

As for the appendicular muscles, there could be two standard templates of limb muscles, one for the pectoral girdle and anterior limbs and another for the pelvic girdle and posterior limbs. For a forelimb, the extrinsic appendicular muscles could be simplified to consist of the trapezius and the latissimus dorsi on the dorsal side of the figure and the pectoralis on the ventral side. Models of the intrinsic muscles might include the deltoid, the biceps and triceps, the brachioradialis, and perhaps one or two forearm muscles that flex or extend the "hand". For a hind limb, the models could include the gluteus maximus, the quadriceps, biceps femoris, gastrocnemius, and a flexor for the foot.

9.3 Fatty Tissue and Skin

The main reason for creating component models of bones and muscles is to have the skin surface of the animated figure deform in a manner one would expect for a human or animal. Muscles and bones alone, however, are not enough to support a believable skin model for a complete figure. For this reason, additional models for shape-holding tissue (such as the cartilage in a figure's ears) or for fatty deposits on a figure would be useful. A simple model such as that of Wilhelms' "stuffing" [Wil94, Wil97] would probably work well.

As for the modeling of the skin surface itself, one of two approaches could be used. The first possibility would be to generate a completely new skin as a surface offset from the underlying bone, muscle, and stuffing models, as is done by Wilhelms [Wil94, Wil97]. The original polygonal data defining the figure would simply be discarded. An alternative approach would be to keep the exterior polygons from the original data and to anchor their vertices to the underlying anatomical components. This is the method of Schneider and Wilhelms [SW98]. Both approaches to skin modeling have been set up in an automated fashion.

CHAPTER 10

CONCLUSION

10.1 Summary

This research began from a conversation with an artist who was describing the time-consuming task of building control skeletons for figures he wanted to animate. The author made a suggestion postulating the automation of the process, and the idea was met with such excitement and enthusiasm from the artist that the author became driven to pursue its realization. After several early attempts that produced somewhat mixed results, the author embarked on the approach that is described in this document. Although the main thrust of the research has always been the automated generation of control skeletons for arbitrary figures, the research has expanded to include the development of supportive algorithms as well as various extensions to the original concept.

Two basic solutions to the problem of automatically generating control skeletons have been designed and implemented. The first solution maintains the generality of the original idea for the research by allowing the skeletonization of almost any polygonal model. It is based on a purely geometric analysis of the given model. The other solution is more restrictive about the models with which it can work, assuming

them to be human-like or animal-like figures. It relies not only on a geometric analysis of the model but also on an anatomical assessment of the model and on anatomically based heuristics. For most human-like or animal-like figures, it is capable of producing a more anatomically accurate control skeleton than the general solution.

The algorithms for each solution begin by constructing a voxelization of the polygonal data. The voxelized model is fed into an algorithm that computes a close approximation to a Euclidean distance map (EDM). This map represents the depth of each voxel from the surface of the figure. The distance map is then processed by an algorithm that extracts the discrete medial surface (DMS) for the object by tracking the ridges implied within the map. Information from the EDM and the DMS is used at various points within both control skeleton generation algorithms.

For the more general algorithm, the DMS is used as the domain for the generation of a tree-like structure of voxel paths. The set of paths is then smoothed and divided into a series of interconnected segments. These segments and the intervening joints form the base structure for the control skeleton. Each segment corresponds to a chain of DMS voxels, and applying the inverse distance transform to each voxel of a chain helps reveal the region of the model's interior over which a particular segment should exert influence. These regions of influence are used to determine the set of control segments to which a vertex of the original model should be anchored. After the polygonal data has been attached, it will be deformed appropriately whenever the pose of the control skeleton is changed.

The anatomically based algorithm uses some additional tools. The voxels of the DMS are partitioned into sets according to an analysis of the shortest paths between

its voxels, and a graph called the level graph is constructed based upon the connectivity of these sets. The level graph is processed using a few simple heuristics in order to determine the anatomical features present in the model. Parts of the level graph and DMS are classified as particular body parts of the figure, and voxel paths within the DMS are generated for each of those body parts. These paths are modified according to certain heuristics so that they follow more closely along the main lines of what might be expected of an anatomical skeleton for the figure. Other heuristics are invoked to produce a segmentation of the paths that is meant to correspond to expected joint locations for that anatomical skeleton. As with the general algorithm, the inverse distance transform is used to help anchor the model's vertices to the segments of the control skeleton. The articulation capabilities of the control skeletons produced by the anatomically based algorithm are believed to be reasonable accurate with regard to the anatomy that the model might be expected to possess.

As an extension to the anatomically based algorithm, the system can automatically produce individual models of bones. Thus, polygonal models of the bone structure of a human-like or animal-like figure can be generated. Since the individual bone models are constructed within the coordinate spaces of the appropriate segments of the control skeleton, the bones will move in accordance with the control skeleton. These models provide a foundation for further anatomically based modeling.

10.2 Contributions

The following list presents the main contributions of the research:

- *A fast algorithm for fully automatic generation of control skeletons for 3D models.* With the aim of alleviating tedium and shortening the time required for

a user to create a control skeleton for a given model, an algorithm has been designed and implemented that automates the entire process. In relatively little time and with very little user input, the algorithm produces a reasonable control skeleton for a wide variety of polygonal models.

- *Use of anatomical knowledge to improve automatic skeleton generation for human-like and animal-like figures.* Drawing from sources on comparative anatomy, an algorithm has been developed and implemented to generate more anatomically appropriate control skeletons for certain common classes of figures. The articulation abilities of these control skeletons are designed to mimic the expected flexibility of the figure. For the most part, the algorithm operates quickly and produces control skeletons that are quite reasonable for animating the given figures in a realistic fashion.

- *A variant of a contour propagation algorithm for approximating the Euclidean distance map in 2D or 3D.* The details of the algorithm are explained clearly, and the algorithm operates in an intuitive manner. Its implementation is straightforward, and it offers a very close approximation of the EDM. The algorithm may be extended in a straightforward fashion to compute close approximations to the EDM in any dimension. Furthermore, regardless of the dimension, the algorithm maintains a linear time complexity.

- *An approach for computation of the discrete medial axis for a 2D object and the discrete medial surface for a 3D object.* Using a contour-based approach, the algorithm offers incremental computation of the DMA or DMS that, at any stage of the execution, is accurate for the set of voxels that have already been

processed. It operates under nearly linear time complexity and can be readily extended to higher dimensional spaces.

- *Use of the inverse distance transform in automatically anchoring surface points of a model to control skeleton segments.* The Euclidean distance map, through the application of the inverse distance transform, offers a convenient means to help determine the skeletal segments that should exert influence over the various regions of a figure. Combining this process with a simple weighted average of anchor points fixed in the frames of those influencing segments provides an effective technique for attaching the surface points of the model to the skeletal segments in a flexible manner.

- *Automatic generation of bone and joint anatomy for a 3D model.* Having constructed an anatomically appropriate control skeleton, the system can generate individual component models for the bones of a skeletal system. These bone models could be used as a base layer for further anatomically based modeling.

10.3 Future Research

This research offers useful approaches to the problem of automatically generating a control skeleton for use in animating a given figure. There are several avenues for improving or extending the research.

One way of improving this research would be to allow for an adaptive voxelization of the figure. This could involve the adaptive subdivision of an initially coarse voxelization, possibly through the use an octree. Various regions of the figure could thus be partitioned with differently sized voxels so that there would always be an appropriate number of voxels representing those regions. It may be possible to automate

the adaptive subdivision based on a distance map computation to ensure that the centralized portions of the DMS are computed at a sufficient discrete depth from the surface. This would allow the DMS to approximate the continuous medial surface at an appropriate level for any region of the figure. This could help to centralize the control skeleton better in various parts of the figure as well as to ensure that the control skeleton effectively represents both the main protrusions of the figure (arms and legs, for instance) as well as its finer protrusions (such as the fingers).

Related to the idea of adaptively computing the DMS is the hierarchical DMA concept of Ogniewicz and Kübler [OK95, Ogn95], discussed on page 2.2.1 of this dissertation. It may be that similar ideas could be developed for computing a hierarchical DMS and for automatically pruning that DMS to a representative level appropriate for an object. Fruits from such research could benefit automatic generation of control skeletons, not just in regard to having a control skeleton whose complexity is appropriate with respect to a given model, but also with respect to modifying that control skeleton to have more or less detail in appropriate regions.

Perhaps a sort of hierarchical control skeleton could be generated automatically for a given object. Such a structure might offer various levels of detail (LOD) with respect to the articulation of a particular model by allowing the instantiation of a specific control skeleton at any of a number of levels of complexity for that model. These levels of articulation could be defined either with respect to the one original surface of the model or with respect to several different LOD representations of that original surface. In the latter case, as an animation switched between different LOD representations, so would it switch between different level of articulation representations of the control skeleton.

It might even be possible to generate appropriate LOD representations by developing a specialized mesh compression algorithm. The compression algorithm, when given a polygonal data model and its hierarchical control skeleton, would produce specific LOD representations of the model (for specific instances of the control skeleton) with the goal of preserving surface details in correspondence with the flexibility of the surface implied by the underlying control skeleton for that LOD.

Another area for extending the research involves the anatomical models. Procedurally defined models of bones could significantly improve the results of automated generation of a skeletal system for a given figure. Automated generation of a figure's musculature is another possibility for future work.

10.4 Final Thoughts

This research has focused on the idea of automation. Although automation is a very powerful tool, it does have limits. With this research, the automation approaches a breaking point when applied to anatomically appropriate control skeleton generation. A type of "software bloat" starts to creep in as the implementation grows ever larger, becoming more and more like an expert system whose goal is to handle all the special cases that the diversity of the natural world can bring.

Instead of trying to automate everything, it might be useful to have a healthy combination of automation and user interaction. The more mathematically or geometrically tedious affairs lend themselves well to programmatical solutions, but the elements that require a higher level of intelligence are probably best left in the hands of a user.

The best platform for constructing control skeletons would probably be a combination of three main elements. First, it would have a group of relatively standard but well-designed user interface tools for manually constructing or modifying control skeletons. The second element would be a set of templates for commonly animated figures. The templates would be designed to produce prefabricated control skeletons whose joints a user would then drag into proper alignment and orientation with respect to the user's model. Templates would be provided not just for generic bipeds and quadrupeds, but also for more specific creatures: different types of birds, reptiles, mammals, and so forth. There would be realistic as well as cartoon versions of the skeletons. Furthermore, the templates would have parameters for dictating such characteristics as the number of fingers on a hand or the number of segments to use in the trunk of the figure. Given the continual push towards more anatomically based modeling, the templates might include complete bone and muscle models for the creatures. The final element would be a few well-conceived, robustly implemented routines for automatically generating control skeletons. The control skeletons generated by these routines would already be positioned inside a given model, so perhaps only minor tweaking of the results would be necessary. One routine might automatically take one of the prefabricated template-based skeletons as selected by the user and fit it into a given model. Another routine might create a customized skeleton for more general figures. It would be designed to handle models that do not correspond to the standard templates, especially those with numerous appendages or with more complex branching structures. As was described in Section 2.3, commercially available animation software already contains the first two elements (though the variety

of templates is still somewhat limited). It will probably not be long before the fully automated methods of control skeleton generation are included as well.

APPENDIX A

GLOSSARY OF ANATOMICAL TERMS

The following is a list of the anatomical terms used in the text. Many of the definitions are taken directly (either word for word or with minor adaptations) from one of the following sources: [Ken87, Web84, Mad94, Sch96, Mad85].

abdomen the part of the body between the thorax and the pelvis.

abduct to move a part away from the main axis: the opposite of adduct.

adduct to draw or pull a part toward the main axis: the opposite of abduct.

amphibian any of various cold-blooded, smoothed-skinned vertebrate organisms of the class *Amphibia*, such as a frog, that typically hatch as aquatic larvae that breathe by means of gills, and metamorphose to an adult form with air-breathing lungs.

ankle the joint that connects the foot with the leg; the proximal segment of the pes.

ankylose to fuse in an immovable articulation.

anterior toward the front; the opposite of posterior.

appendicular skeleton the portion of the skeleton pertaining to the girdles and limbs.

articulation method or manner of jointing.

axial skeleton the portion of the skeleton pertaining to the trunk, head, and tail.

ball-and-socket joint a joint allowing three rotational degrees of freedom, such as the shoulder or hip joint.

biceps brachii the large muscle at the front of the upper arm that flexes and supinates the forearm.

biceps femoris the large muscle at the back of the thigh that flexes the lower leg.

biped a two-footed animal.

bone the dense, semirigid, porous, calcified connective tissue of the skeleton of most vertebrates.

brachialis a muscle of the upper arm that flexes the forearm.

brachioradialis a forearm muscle that flexes the forearm at the elbow.

brain case the part of the skull containing the brain.

carpals the bones of the carpus.

carpus the wrist; the proximal segment of the manus.

cartilage a tough white fibrous connective tissue attached to the articular surfaces of bones.

caudal of, at, or near the tail or hind parts; posterior.

caudal vertebrae the vertebrae in the portion of the spine pertaining to the tail.

cephalic located on, in, or near the head.

cervical vertebrae the vertebrae in the portion of the spine pertaining to the neck.

chordate any of the numerous animals of the phylum *Chordata*, including all vertebrates and certain marine animals having a notochord.

clavicle a bone linking the sternum and the scapula; the collarbone.

coccyx a small bone at the base of the spinal column in humans, composed of several fused vertebrae.

cranium the skull.

deltoideus a thick, triangular muscle covering the shoulder joint, used to raise the arm from the side.

depress to lower; the opposite of lift.

digestive tract the system of organs and tissues responsible for digestion.

digit a finger or toe.

digitigrade walking on curved digits with wrist and ankle elevated, as cats and dogs do.

distal located far from the origin or line of attachment: the opposite of proximal.

dorsal of, toward, or near the back: the opposite of ventral.

dorsoventral axis the line of the body running from the belly to the back.

elbow the joint between the upper arm and the forearm.

embryonic of or relating to an organism in its early developmental stages.

epaxial muscles the dorsal muscles of the trunk and tail: the oblique and longitudinal muscles along the spine, collectively functioning in straightening the vertebral column and in lateral flexion of the body.

evolution the historical development of a related group of organisms: the theory that groups of organisms, as species, may change over time so that descendants differ morphologically and physiologically from their ancestors.

extend to straighten: the opposite of flex.

extrinsic appendicular muscles the appendicular muscles arising on the axial skeleton or fascia of the trunk and inserting on a girdle or limb.

femur the thighbone; the proximal bone in the hind limb.

fibula the outer and smaller of the two bones in the shank.

flex to bend: the opposite of extend.

forearm the portion of the forelimb between the elbow and the wrist, containing the radius and ulna.

fusiform tapering at each end; spindle shaped.

gastrocnemius a muscle in the back of the shank that extends the foot.

girdle the pelvis or pectoral arch; the portion of the skeleton between a pair of limbs and the axial skeleton.

gluteus the large muscles of the buttocks.

hinge joint a joint allowing a single rotational degree of freedom, such as the knee or elbow.

hip the joint between the femur and the pelvis; alternatively, the lateral projecting prominence of the pelvis.

homologous corresponding in structure and evolutionary origin, such as the flippers of a seal and the arms of a human.

humerus the long bone of the upper arm.

hypaxial muscles the ventral muscles of the trunk; the sheet-like muscles of the body wall, functioning to flex the spine and to contain the innards of the trunk.

innards the internal bodily organs.

insertion the distal site of attachment for a muscle, as opposed to the origin.

instep the arched middle part of the human foot or the analogous part in animals; the middle segment of the pes, containing the metatarsals.

intrinsic appendicular muscles the appendicular muscles that arise on a girdle or limb and insert more distally on the limb.

kingdom the largest taxonomic category into which organisms are placed: Monera, Protista, Fungi, Plants, and Animals.

knee the joint distal to the femur that provides the articulation for the tibia, fibula, and patella.

latissimus dorsi a wide muscle of the back that originates on the spine and inserts on the humerus, functioning to extend and adduct the arm or forelimb.

lift to raise; the opposite of depress.

limb an animal's jointed appendage, used for locomotion or grasping, as an arm, leg, wing, or flipper.

longitudinal axis the line of the body running from the head to the tail.

lumbar vertebrae the vertebrae in the portion of the spine between the ribs and the pelvis.

mammal a member of the vertebrate class *Mammalia*, such as a human or a dog, distinguished by self-regulating body temperature, hair, and, in the females, mammae.

mandible the lower jaw: the jawbone.

manus the end of the vertebrate forelimb, consisting of the wrist, palm, and digits.

metacarpals the bones of the palm.

metamerism the condition of having the body divided into a series of homologous segments.

metatarsals the bones of the instep.

muscle a tissue made up of fibers that can contract and relax to effect bodily movement.

muscle belly the bulging part of a muscle.

musculature the system of muscles of an animal or body part.

notochord a cordlike skeleton of the back: the primitive backbone.

origin the proximal site of attachment for a muscle, as opposed to the insertion.

palm the middle segment of the manus, containing the metacarpals.

patella a flat, triangular bone at the front of the knee.

pectoral girdle the skeletal structure attached to and supporting the forelimbs and consisting of the scapulae and, if present, the clavicles.

pectoralis a muscle mass on the ventral side of the thorax that functions to adduct the humerus.

pelvic girdle the skeletal structure of bone or cartilage by which the hind limbs or analogous parts are supported and joined to the vertebral column.

pelvis a basin-shaped skeletal structure that connects the lower limbs to the spine.

pentadactyl Having five digits.

peristalsis Wavelike muscular contraction that push contained matter along tubular organs.

pes the end of the vertebrate hind limb, consisting of the ankle, instep, and digits.

phalanx a bone of a finger or toe.

phylum a taxonomic category applied to animals that follows kingdom and lies above class.

plantigrade walking with the entire lower surface of the foot on the ground, as humans and bears do.

posterior toward the rear; the opposite of anterior.

process a part extending or projecting from an organ or organism.

pronation rotation of the forearm to turn the palm of the hand to face downward or backward; the opposite of supination.

protract to extend or protrude; the opposite of retract.

proximal near the central part of the body or a point of attachment or origin; the opposite of distal.

quadriceps femoris the muscles on the front and sides of the thigh that act to to extend the shank or adduct the thigh.

quadruped a four-footed animal.

radius the shorter and thicker of the two forearm bones.

reptile a cold-blooded, usually egg-laying vertebrate of the class *Reptilia*, such as a snake, lizard, crocodile, turtle, or dinosaur, having an outer covering of scales or horny plates and breathing with lungs.

retract to draw back; the opposite of protract.

rib one of a series of long, curved bones extending from the spine to the sternum.

rib cage the enclosing structure formed by the ribs and the bones to which they are attached.

sacral vertebrae the vertebrae to which the pelvic girdle is attached.

sacrum a bony complex consisting of a number of sacral vertebrae that have fused together, located at the dorsal side of the pelvis.

scapula either of a pair of large, flat, triangular bones that form the back part of the shoulder.

shank the portion of the hind limb between the knee and the ankle, containing the tibia and the fibula.

shoulder the joint proximal to the humerus; alternatively, the region of the body between the upper arm and the neck.

skeleton the internal vertebrate structure composed of bone and cartilage that protects and supports the soft organs, tissues, and parts.

skull the framework of the head of vertebrates, made up of the bones of the brain case and face.

socket the hollow part of a joint that receives the end of a bone.

somatic muscles the muscles primarily responsible for interacting with the external environment.

species a fundamental taxonomic classification category consisting of organisms capable of interbreeding.

spinal column the assemblage of articulated vertebrae extending from the cranium to the coccyx or the end of the tail, encasing the spinal cord and forming the supporting axis of the body.

spine the spinal column; backbone.

sternum a long flat bone forming the midventral support of most of the ribs and, if present, the clavicles.

subphylum a taxonomic category ranking between a phylum and a class.

supination rotation of the forearm to turn the palm of the hand to face forward or upward; the opposite of pronation.

symphysis a growing together.

synsacrum a bony complex in birds resulting from the ankylosing of the last thoracic vertebra, all the lumbar and sacral vertebrae, the first few caudal vertebrae, and associated ribs.

tarsals the bones of the tarsus.

tarsus the ankle; the proximal segment of the pes.

tendon a band of tough inelastic fibrous tissue connecting a muscle with its bony attachment.

tetrapod a vertebrate having two sets of paired limbs; a vertebrate that dwells on land or had land-dwelling ancestors.

thigh the proximal segment of the hind limb, containing the femur.

thorax the part of the body between the neck and the abdomen, partially encased by the ribs.

thoracic vertebrae the vertebrae to which the ribs are attached.

tibia the inner and larger of the two bones in the shank.

trapezius a superficial muscle of the shoulder region which acts to raise the shoulders.

triceps a large three-headed muscle running along the back of the arm and functioning to extend the forearm.

ulna the larger of the two forearm bones.

unguligrade walking on the fingertips, such as the hoofed mammals: horses, cattle, deer, and so forth.

upper arm the proximal segment of the forelimb, containing the humerus.

ventral relating to or located on or near the belly; the opposite of dorsal.

vertebra any of the bones or cartilaginous segments making up the spinal column.

vertebrate having a backbone or spinal column; a member of the subphylum *Vertebrata* that includes the fishes, amphibians, reptiles, birds, and mammals, all of which have a segmented bony or cartilaginous spinal column.

vestigial existing or persisting as a rudimentary or degenerate structure.

visceral muscles the muscles primarily responsible for internal body functions.

wrist the junction between the hand and the forearm; the system of bones forming this junction.

BIBLIOGRAPHY

[ABH+94] Francisco Azuola, Norman I. Badler, Pei-Hwa Ho, Ioannis Kakadiaris, Dimitri Metaxas, and Bond-Jay Ting. Building anthropometry-based virtual human models. In *Proceedings IMAGE VII Conference*, June 1994.

[AG85] William W. Armstrong and Mark W. Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1(4):231–240, December 1985.

[AK00] Nina Amenta and Ravi Krishna Kolluri. Accurate and efficient unions of balls. In *Proceedings of the 16th Annual ACM Symposium on Computational Geometry*, pages 119–128, 2000.

[Ale94] R. McNeill Alexander. *Bones: The Unity of Form and Function*. Macmillan, New York, 1994.

[AS79] Norman Adams and Joe Singer. *Drawing Animals*. Watson-Guptill Publications, New York, 1979.

[BBGS99] Robert Blanding, Cole Brooking, Mark Ganter, and Duane Storti. A skeletal-based solid editor. In *Solid Modeling '99 (Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications)*, pages 141–150, June 1999.

[BBZ91] Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. Morgan Kaufmann, 1991.

[BGKW95] Heinz Breu, Joseph Gil, David Kirkpatrick, and Michael Werman. Linear time Euclidean distance algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-17:529–533, May 1995.

[BL99] Jules Bloomenthal and Chek Lim. Skeletal methods of shape manipulation. http://www.unchainedgeometry.com, 1999.

[Boo79] Fred L. Bookstein. The line-skeleton. *Computer Graphics and Image Processing*, 11:123–137, 1979.

[Bor86] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing.*, 34:344–371, 1986.

[BPW93] Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1993. ISBN 0-19-507359-2.

[CHP89] John E. Chadwick, David R. Haumann, and Richard E. Parent. Layered construction for deformable animated characters. In Jeffrey Lane, editor. *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 243–252, July 1989.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[Cur00] Curious Labs web site. http://www.curiouslabs.com, 2000.

[CZ92] David T. Chen and David Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. In Edwin E. Catmull, editor, *Computer Graphics (Proceedings of SIGGRAPH '92)*, volume 26, pages 89–98, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois.

[Dan80] Per-Erik Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3):227–248, November 1980.

[DH90] Debasish Dutta and Christoph M. Hoffmann. A geometric investigation of the skeleton of CSG objects. In B. Ravani, editor, *Proceedings of the 16th ASME Design Automation Conference: Advances in Design Automation. Computer Aided and Computational Design*, volume I, pages 67–75, 1990.

[Dis00] Discreet web site. http://www.discreet.com, 2000.

[EBD56] W. Ellenberger, H. Baum, and H. Dittrich. *An Atlas of Animal Anatomy for Artists.* Dover Publications, Inc., New York, second revised and expanded edition, 1956.

[Egi00] egi.sys web site. http://www.egisys.com, 2000.

[Eng00] Engineering Animation, Inc. web site. http://www.eai.com, 2000.

[FS99] Carol Franger and Linda Stevens. *MetaCreations Poser 4 User Guide.* MetaCreations Corporation, Carpinteria, California, 1999.

[FW88] David R. Forsey and Jane Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. In *Proceedings of Graphics Interface '88*, pages 8–15, June 1988.

[GF96] Yaorong Ge and J. Michael Fitzpatrick. On the generation of skeletons from discrete Euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(11), November 1996.

[GG94] Jean-Dominique Gascuel and Marie-Paule Gascuel. Displacement constraints for interactive modeling and animation of articulated structures. *The Visual Computer*, 10(4):191–204, March 1994. ISSN 0178-2789.

[GKHS98] Nikhil Gagvani, Dilip Kenchammana-Hosekote, and Deborah Silver. Volume animation using the skeleton tree. In *IEEE Symposium on Volume Visualization*, pages 47–54, October 1998. ISBN 0-8186-9180-8.

[Gol99] Christopher Gold. Crust and anti-crust: A one-step boundary and skeleton extraction algorithm. In *Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, pages 189–196, 1999.

[GS99] Nikhil Gagvani and Deborah Silver. Realistic volume animation with alias. In *Volume Graphics*, chapter 15. Springer-Verlag, October 1999.

[GYKD91] John A. Goldak, Xinhua Yu, Alan Knight, and Lingxian Dong. Constructing discrete medial axis of 3-D objects. *International Journal of Computational Geometry and Applications.*, 1(3):327–339, 1991.

[Har99] Dr. Tony Hare. *Animal Fact File: Head-to-tail Profiles of More than 90 Mammals*. Facts On File, Inc., New York, 1999.

[Hil95] Milton Hildebrand. *Analysis of Vertebrate Structure*. John Wiley & Sons, Inc., New York, fourth edition, 1995.

[HKGL00] Perry Harovas, John Kundert-Gibbs, and Peter Lee. *Mastering Maya Complete 2*. SYBEX, Inc., San Francisco, 2000.

[Iga99] Takeo Igarashi's Home Page. http://www.mtl.t.u-tokyo.ac.jp/~takeo/teddy/teddy.htm, 1999.

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3D freeform design. *Proceedings of SIGGRAPH 99*, pages 409–416, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[JBD+00] Angie Jones, Sean Bonney, Brandon Davis, Sean Miller, and Shane Olsen. *3D Studio Max 3 - Professional Animation.* New Riders Publishing, Indianapolis, Indiana, 2000.

[Joh94] Jinny Johnson. *Skeletons: An inside look at animals.* The Reader's Digest Association, Inc., Pleasantville, NY, 1994.

[Ken87] George C. Kent. *Comparative Anatomy of the Vertebrates.* Wm. C. Brown Publishers, Dubuque, Iowa, seventh edition, 1987.

[Kir79] David G. Kirkpatrick. Efficient computation of continuous skeletons. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 18–27, 1979.

[KM95] Ioannis A. Kakadiaris and Dimitri Metaxas. 3D human body model acquisition from multiple views. In *Proceedings of the Fifth International Conference on Computer Vision*, pages 618–623, 1995.

[Lap99] Jeffrey Lapierre. Matching anatomy to model for articulated body animation. Master's thesis, University of California, Santa Cruz, December 1999.

[LKC94] Ta-Chih Lee, Rangasami L. Kashyap, and Chong-Nam Chu. Building skeleton models via 3-D medial surface/axis thinning algorithms. *CVGIP: Graphical Models and Image Processing*, 56(6):462–478, November 1994.

[LV99] Francis Lazarus and Anne Verroust. Level set diagrams of polyhedral objects. In *Solid Modeling '99 (Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications)*, pages 130–140, June 1999.

[LW99] Jeffrey Lapierre and Jane Wilhelms. Matching anatomy to model for articulated body animation. In M. H. Hamza, editor, *Proceedings of the IASTED International Conference on Computer Graphics and Imaging.* The International Association of Science and Technology for Development (IASTED), ACTA Press, 1999.

[Mad85] Sylvia S. Mader. *Inquiry into Life.* Wm. C. Brown Publishers, Dubuque, Iowa, fourth edition, 1985.

[Mad94] Sylvia Mader. *Understanding Human Anatomy & Physiology.* Wm. C. Brown Publishers, Dubuque, Iowa, second edition, 1994.

[Mae96] George Maestri. *Digital Character Animation.* New Riders Publishing, Indianapolis, Indiana, 1996.

[May95] Stephen F. May. AL: Animation language reference manual. Technical Report ACCAD-11/94-TR3, ACCAD, The Ohio State University, July 1995.

[Mor00] R. Shamms Mortier. *The Poser 4 Handbook*. Charles River Media, Inc., Rockland, Massachusetts, 2000.

[MTT90] Nadia Magnenat-Thalmann and Daniel Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, Tokyo, second revised edition, 1990.

[Mul92] James C. Mullikin. The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535, 1992.

[Ogn95] Robert L. Ogniewicz. Automatic medial axis pruning by mapping characteristics of boundaries evolving under the Euclidean geometric heat flow onto voronoi skeletons. Technical Report 95-4, Harvard Robotics Laboratory, 1995.

[OK95] Robert L. Ogniewicz and O. Kübler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[Pag92] David W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing*, 54(1):56–74, 1992.

[Par88] Steve Parker. *Eyewitness Books: Skeleton*. Alfred A. Knopf, New York, 1988.

[Par90] José M. Parramón. *Human Anatomy*. Watson-Guptill Publications, New York, 1990.

[PG87] Jim Piper and Erik Granum. Computing distance transformations in convex and non-convex domains. *Pattern Recognition*, 20(6):599–615, 1987.

[Rag92a] Ingemar Ragnemalm. Fast erosion and dilation by contour processing and thresholding of distance maps. *Pattern Recognition Letters*, 13:161–166, 1992.

[Rag92b] Ingemar Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics, and Image Processing. Image Understanding*, 56(3):399–409, November 1992.

[Ros98] A. Rosenfeld. *Digital geometry: introduction and bibliography*, chapter 1. Springer, 1998. ISBN 981-3083-94-8, edited by R. Klette and A. Rosenfeld and F. Sloboda.

[SAR95] D. J. Sheehy, C. G. Armstrong, , and D. J. Robinson. Computing the medial surface of a solid from a domain delaunay triangulation. In Chris Hoffman and Jarek Rossignac, editors, *Solid Modeling '95 (Proceedings of the Third ACM Solid Modeling Conference)*, pages 201–212, May 1995.

[Sch96] Coenraad Frederik Scheepers. *Anatomy-Based Surface Generation for Articulated Models of Human Figures*. Ph.D. thesis, The Ohio State University, 1996.

[SPB95] Evan C. Sherbrooke, Nicholas M. Patrikalakis, and Erik Brisson. Computation of the medial axis transform of 3-D polyhedra. In Chris Hoffman and Jarek Rossignac, editors, *Solid Modeling '95 (Proceedings of the Third ACM Solid Modeling Conference)*, pages 187–200, May 1995.

[SPCM97] Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. *Proceedings of SIGGRAPH 97*, pages 163–172, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[ST94] Toyofumi Saito and Jun-Ichiro Toriwaki. New algorithms for Euclidean distance tranformations of an n-dimensional digitised picture with applications. *Pattern Recognition*, 27(11):1551–1565, 1994.

[Sta96] Richard C. Staunton. An analysis of hexagonal thinning algorithms and skeletal shape representation. *Pattern Recognition*, 29(7):1131–1146, 1996.

[SvO97] M. G. J. R. Stalpers and C. W. A. M. van Overveld. Deforming geometric models based on a polygonal skeleton mesh. *Journal of Graphics Tools*, 2(3):1–14, 1997. ISSN 1086-7651.

[SW98] Philip J. Schneider and Jane Wilhelms. Hybrid anatomically based modeling of animals. *Computer Animation '98*, June 1998. Held in Philadelphia, Pennsylvania, USA.

[Tea98] The Maya Documentation Team. *Using Maya: Animation*. Alias|Wavefront, Inc., Toronto, Canada, 1998.

[Tea99] The Maya 2 Documentation Team. *Using Maya: Character Setup*. Alias|Wavefront, Inc., Toronto, Canada, 1999.

[TF84] Yea-Fu Tsao and King-Sun Fu. Stochastic skeleton modeling of objects. *Computer Vision, Graphics and Image Processing. (USA)*, 25:348–370. March 1984.

[TG98] Russell Turner and Enrico Gobbetti. Interactive construction and animation of layered elastically deformable characters. *Computer Graphics Forum*, 17(2):135–152, 1998. ISSN 1067-7055.

[Tra00] Transom Technologies, Inc. web site. http://www.transom.com, 2000.

[TT98] Marek Teichmann and Seth Teller. Assisted articulation of closed polygonal models. http://graphics.lcs.mit.edu/~marekt, 1998.

[Vin91] Luc Vincent. Exact Euclidean distance function by chain propagations. In *Proceedings CVPR '91 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition)*, pages 520–525. IEEE Computer Society Press, 1991.

[VSC00] Nathan Vogel, Sherri Sheridan, and Tim Coleman. *Maya 2 Character Animation*. New Riders Publishing, Indianapolis, Indiana, 2000.

[VVD89] Ben J. H. Verwer, Piet W. Verbeek, and Simon T. Dekker. An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):425–429, 1989.

[Wai88] Stephen Wainwright. *Axis and Circumference: the Cylindrical Shape of Plants and Animals*. Harvard University Press, Cambridge, 1988.

[Web84] *Webster's II New Riverside University Dictionary*. Houghton Mifflin Company, Boston, 1984.

[WG97] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. *Proceedings of SIGGRAPH 97*, pages 173–180, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.

[Wil87] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[Wil94] Jane Wilhelms. Modeling animals with bones, muscles, and skin. Technical Report UCSC-CRL-95-01, Baskin Center for Computer Engineering and Information Sciences, University of California, Santa Cruz, 1994.

[Wil97] Jane Wilhelms. Animals with anatomy. *IEEE Computer Graphics & Applications*, 17(3):22–30, May - June 1997. ISSN 0272-1716.

[WP00] Lawson Wade and Richard E. Parent. Fast, fully-automated generation of control skeletons for use in animation. In *Proceedings of Computer Animation 2000*, 2000. A lengthier version is available as Technical Report OSU-ACCAD-9/99-TR3, The Ohio State University, Advanced Computing Center for the Arts and Design, 1999.

[WW92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Publishing Company, Wokingham, England, 1992.

[Yam84] H. Yamada. Complete Euclidean distance transformation by parallel operation. In *Proceedings of the Seventh International Conference on Pattern Recognition*, pages 69–71, 1984.

[YR91] C. Yao and J. G. Rokne. A straightforward algorithm for computing the medial axis of a simple polygon. *International Journal of Computer Mathematics*, 39:51–60, 1991.