2D Object Abstraction: The Automatic Generation of Hierarchical Geometric Representations

Maryann Simmons

Master's Project
under the direction of Carlo H. Séquin

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley

September 17, 1997

Abstract

This thesis presents an exploratory framework for automatically generating hierarchical object representations specialized for geometric tasks. The approach consists of two subprocesses: the first extracts geometric information from the input object, and the second generates a specialized representation based on this information.

The first subprocess produces a multi-resolution representation that encapsulates the salient geometric features of an object, as well as its topological decomposition into parts. The representation is generated in two steps. First, a multi-resolution Geometry Representation (G-Rep) is built. Its main components are a Cell-Based Spatial Representation (CSR) that provides spatial filtering at the desired feature resolution, and an Axial Shape Graph (ASG) that captures local shape information as well as global information about the overall geometric structure of the object. The CSR and ASG components are calculated at multiple resolutions and linked together to form the G-Rep hierarchy. In the second step, the Axial Shape Graph is decomposed into a tree representing the overall shape structure of the object as a hierarchy of subcomponents. Using the Axial Shape Graph, the task of shape decomposition is reduced to a graph partitioning problem whose solution results in a well-balanced part hierarchy.

The second subprocess constructs a hierarchy of task-specific representations utilizing the geometric information provided by the G-Rep. The entire representation generation framework is driven by metrics that quantify the desirable characteristics of representations for the particular task. We show that this structure can be utilized to generate representations specialized for the task of Collision Detection in 2D environments.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Problem Statement . . . 2

1.3 Overview of Approach . . . 2

2 Multi-Resolution Geometry Representation **5**

2.1 Cell-Based Spatial Representation . . . 6

2.2 Axial Shape Graph . . . 7

2.2.1 Review of Axial Representations . . . 8

2.2.2 Axial Shape Graph Structure . . . 10

2.2.3 Axial Shape Graph Weighting . . . 11

2.2.4 Axial Shape Graph Multi-Resolution Information . . . 13

2.3 Hierarchical Decomposition of the Geometry Representation . . . 13

3 Construction of the Cell-Based Spatial Representation **16**

3.1 Polygon Insertion . . . 16

3.2 Image Insertion . . . 19

4 Construction of the Axial Shape Graph **20**

4.1 Axis Segment Creation . . . 21

4.1.1 Calculating the Distance Map . . . 21

4.1.2 Identifying Local Maxima in the Distance Map . . . 22

4.1.3 Creating the Axis Segments from Axial Points . . . 25

4.2 Creating the Axial Shape Graph . . . 28

4.2.1 Constructing the Graph Structure . . . 28

4.2.2 Assigning Weights to the Graph Nodes . . . 31

4.2.3 Linking the Axial Shape Graph across Resolutions . . . 32

5 Hierarchical Decomposition of the Geometry Representation **33**

5.1 Selecting a Partitioning Point . . . 34

5.1.1 Choosing a Graph Partition Node . . . 34

5.1.2 Choosing a Axis Segment Partition Point . . . 35

5.2 Partitioning the Geometry Representation . . . 36

5.3 Utilizing the Hierarchy . . . 38

5.4 Managing the Part Hierarchy . . . 38

5.5 Comparison to Previous Work . . . 39

6 Generating Representations for Collision Detection **41**

6.1 Representations for Collision Detection 41
6.2 Overview of the Representation Generation Framework 43
6.3 Quality Metrics 45
6.4 Construction of Geometry Representation 47
6.5 Generation of Hierarchical Representations 47
6.5.1 Decomposition 48
6.5.2 Representation Generation 49
6.5.3 Evaluation 50
6.5.4 Output Representation 51
6.6 Results 51
6.6.1 Effects of Object Rotation 53
6.6.2 Application Performance 54
6.7 Comparison to Previous Work 56

7 Conclusion/Future Work **58**

8 Appendix **59**

Acknowledgements **68**

References **69**

List of Figures

1 Two-stage process of Object Abstraction 2
2 Representation Generation Framework 4
3 Components of the Geometry Representation (G-Rep) 5
4 G-Rep structures in the hierarchy 6
5 Maximal Discs: a) Object with local maxima highlighted b) Maximal Discs c) Discs of strong local maxima d) Discs of weak local maxima 9
6 Axial Shape Graph: a) Axis Segments and connections b) Associated Components c) Graph Representation with Edge Labels 11
7 Axial Shape Graph: a) Axis Size Complexity Weights b) Axis, Left and Right Complexity Weights 12
8 Axial Feature Weights: Axes for four levels of the hierarchy are shown. The axes of the first level are labeled with their Feature Weights. The axis corresponding to the base is linked across all four levels. 13
9 Two possible object decompositions based on shape and Complexity Weights 14
10 Cell terminology: a) cell (x,y) and its 8 nearest neighbors b) Neighbor labeling: Direct Neighbors $N_D = \{0,2,4,6\}$, Indirect Neighbors $N_I = \{1,3,5,7\}$ 16
11 a) Original object b) Grid Cell Representation c) Adaptive Background d) Fully compressed quadtree representation 18
12 a) Original object b) Grid Cell Representation c) Fill cells 19
13 Distance Metrics: a)1-1 unweighted (r=3) b) 1-2 (r=3) c) 2-3 (r=6) d) 3-4 (r=12) 21
14 Setting Direction Vectors a) Distance values after first pass b) Highlighted cell is missing NE direction vector 22
15 Distance Map: a) greyscale by distance with direction vectors. b) 3D representation of DM 23
16 Axial Points: a) Rectangle Local Maxima cells b) Rectangle Strong Local Maxima cells c) Cactus Local Maxima cells d) Cactus Strong Local Maxima cells e) Dino Local Maxima cells f) Dino Strong Local Maxima cells 24
17 Axial Normal Direction Pairs a) E-W b) N-S c) NW-SE d) SW-NE e) N-S/SW-NE 25
18 Axial Normal Directions (AND) and Axial Directions: a) N-S Axial Direction b) E-W Axial Direction b) E-W/NW-SE Axial Direction 26
19 Fitting linear segments to axial cells: a,b,c) The top figure shows the axial cells with the left points labeled by circles and the right points labeled by triangles. The bottom figure shows the resulting axis line segments. 27
20 Cell Connectivity Types a) 1x1 saddle b) 2x2 saddle c) 1x1 saddles, local maxima and a strict local maximum 29
21 Connecting the Graph: The Axis Segments 1,2, and 3 are connected by following the highlighted climbing paths. The resulting graph representation is shown below: the letters L and R (for Left and Right) at the ends of the edge designate which end of the axis the edge connects to. 31
22 G-Rep Decomposition process and the Representation Generation Pipeline 33
23 Calculation of Axis Partition Ratio 35
24 Selection of Partition Point 36
25 G-Rep Decomposition: a) Original b) Paths to shore from axis end cell c) Resulting Decomposition d) Resulting expanded cut area 37
26 Part Tree Structure: a-d show the progressive decomposition of an object into parts. The upper figure shows the cut and resulting object parts and the lower figure shows the tree representation. 38
27 Hierarchical Representation for Collision Detection 43
28 Representation Generation Framework for Collision Detection 44
29 Hausdorff points for dinosaur parts 46

30 Size balanced Decomposition a) Input Object b) Axial Shape Graph c) ASG after single decomposition step d) Resulting Conservative Hull Hierarchy e) Resulting Bounding Box Hierarchy . . . 49
31 Step-by-step decomposition of dinosaur . . . 50
32 Dinosaur: $\mathcal{H}_{MAX} = 5.1, \mathcal{H}_{AVG} = 1.8, \mathcal{Q}_{CH} = 0.9, \mathcal{Q}_{BB} = 0.5, \mathcal{Q}_{A} = 0.33, \mathcal{B}_{CH} = 0.8, \mathcal{B}_{BB} = 1.02, \mathcal{B}_{A} = 1.02$ . . . 52
33 Effects of Rotating Object on Axial Shape Graph . . . 54
34 Effects of Rotating Object on Output Representation . . . 55
35 Application running times: a) Running times for a rectangle at different cell sizes undergoing a single partition. b)Running times for a size of 9600 cells with 1,2,4,8 and 16 partitions. c)Running times for each size and partition count. . . . 57
36 Duck: $\mathcal{H}_{MAX} = 7.3, \mathcal{H}_{AVG} = 2.9$ . . . 59
37 Skeleton:$\mathcal{H}_{MAX} = 5.2, \mathcal{H}_{AVG} = 2.3$ . . . 60
38 T-Rex: $\mathcal{H}_{MAX} = 7.3, \mathcal{H}_{AVG} = 3.0$ . . . 61
39 Snowflake:$\mathcal{H}_{MAX} = 9.1, \mathcal{H}_{AVG} = 2.4$ . . . 62
40 Marlin: $\mathcal{H}_{MAX} = 31.9, \mathcal{H}_{AVG} = 3.4$ . . . 63
41 Frog: $\mathcal{H}_{MAX} = 8.5, \mathcal{H}_{AVG} = 2.6$ . . . 64
42 Muledeer:$\mathcal{H}_{MAX} = 9.14, \mathcal{H}_{AVG} = 2.7$ . . . 65
43 Praying mantis: $\mathcal{H}_{MAX} = 14.4, \mathcal{H}_{AVG} = 3.5$ . . . 66
44 Dancing skeleton: $\mathcal{H}_{MAX} = 7.9, \mathcal{H}_{AVG} = 2.7$ . . . 67

1 Introduction

1.1 Motivation

There are many ways to represent the objects comprising a two or three dimensional environment suitable for visualization, simulation, and experimentation. Objects in the real world have descriptions of infinite complexity, but most applications need only a small amount of information to perform a task adequately. The key to success in an interactive, complicated virtual environment is to choose models for each task that best encapsulate the information necessary to perform the task in an efficient manner.

As an example, consider a model of a cactus plant in a virtual environment for real-time visualization and exploration. In this application, the goal is a realistic rendering of the cactus from any direction and distance which can be chosen arbitrarily by the user. The model must contain reasonably accurate geometric data, as well as surface reflectance and texture information that define its appearance in an illuminated environment. If the same model is used in a mobile robot's world, solely as an obstacle to be avoided, a much cruder geometric representation may suffice, possibly as simple as a 2D area in the ground plane into which the robot cannot pass.

The intelligent use of specialized representations has played an important role in the success of work done to date in interactive visualization environments and will continue to make a key contribution as the frontiers in this area are pushed even further. The Berkeley architectural WALKTHRU program [17] is an illustrative example of a system that achieves significant performance gains by specializing its representations for different tasks. Representations are also specialized within a task to optimize the task when performed in different contexts.

The WALKTHRU program makes it possible for a user to move through a complex model of a building at interactive speeds. There are two major representations contributing to the ability of the system to achieve interactive frame rates: visibility cells and level of detail (LOD) representations.

The WALKTHRU system performs visibility preprocessing [34, 35], a technique exploiting the fact that in architectural environments only a small portion of the world is visible at any one time. The representation used in this approach models the world as a collection of cells and portals, and computes the cell-to-cell visibility in a preprocessing stage. This information is used at runtime to cull out large portions of the model that are not visible, but that would still be rendered if only traditional view frustum culling and z-buffering are used.

Visual level of detail (LOD) representations are also utilized by the WALKTHRU system to optimize the rendering phase [15, 16]. This technique exploits the fact that if an object contributes very little to the final image, it does not need to be represented in full detail to achieve the same visual effect when rendered to the display. Each object has several representations, at various levels of complexity. The display manager chooses which representation to render in each frame based on its visual contribution in the current context. In this method, multiple specialized representations are utilized within the *same* task.

The WALKTHRU system illustrates the benefits of utilizing specialized representations. For the use of these specialized representations to be practical in complex environments, however, it is essential to be able to generate them automatically, or with only limited user intervention.

1.2 Problem Statement

Many of the tasks that are performed with objects in a virtual environment, such as collision detection, rendering, and visibility culling, are based on the geometric structure of the objects. Efficient implementations of these tasks first remove objects from processing that are not relevant to the current computation. This culling involves a test: Is the object visible? (visibility culling), Is the object in contact with any other object? (collision detection), Is the object visually prominent enough in the scene to be rendered at full detail? (LOD rendering). The test must be conservative, i.e. an object may be included in the computation even if it is not relevant, but must not be excluded if it is. These operations are most efficient when the objects are represented with well-balanced trees of hierarchical subcomponents that reflect the geometric structure of the object at resolution levels appropriate for the particular task. A hierarchical representation allows the elision of multiple objects with a single test, and a balanced hierarchy provides optimal worst case performance behavior when processing representations at the leaf levels. While many systems utilize such specialized models, the process of generating representations automatically for a wide range of objects and geometric tasks remains an open problem.

1.3 Overview of Approach

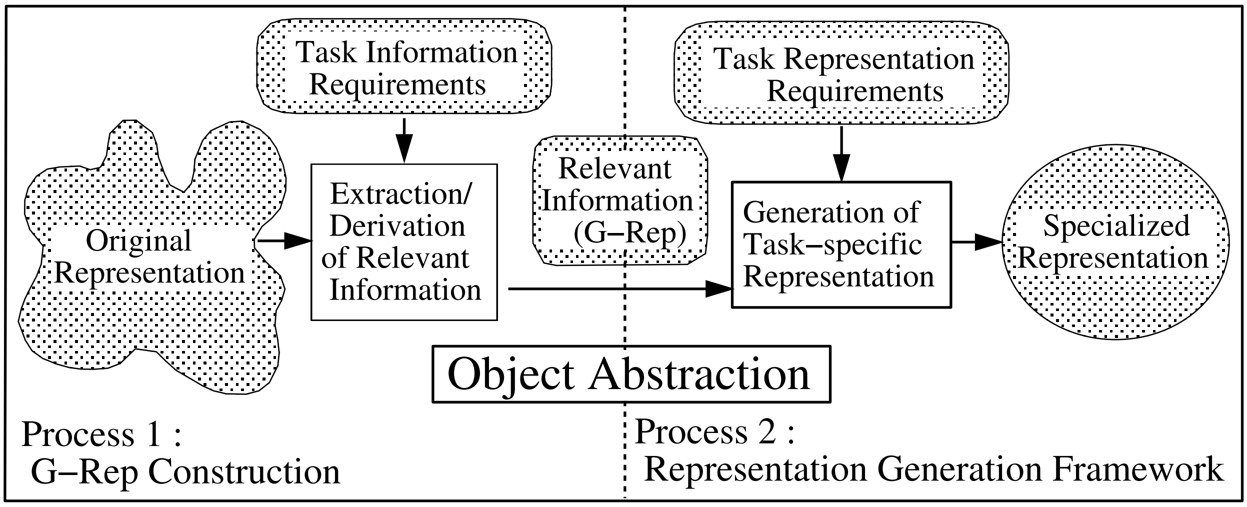


Figure 1: Two-stage process of Object Abstraction

This thesis presents a framework for automated Object Abstraction and Decomposition. Object Abstraction is defined in the context of a particular task, and refers to the process of taking a complicated object description and distilling out the information relevant to the task. Once the important information is isolated, a representation is created that embodies this information in a simplified form and allows the task to be performed efficiently. Object Decomposition is utilized as part of the Abstraction process to produce a well-structured output representation. Our approach considers only geometric tasks, i.e. those that are dependent solely on the geometric properties of an object such as shape, spatial occupancy and area. We assume that the desired output representation is a well-balanced tree of hierarchical, conservative representations that satisfy metrics defined by the particular task. We discuss the general framework and the implementation of an application based on this framework that generates representations specialized for the task of Collision Detection in two dimensional environments.

The problem of generating task-specific representations is broken down into two subprocesses: the first extracts the information relevant to the task, and the second generates a specialized representation based on this information. Figure 1 illustrates the general structure of the abstraction process.

The first subprocess produces a multi-resolution representation that encapsulates the salient geometric features of an object, as well as its topological decomposition into parts. The representation is generated in two steps. First, a multi-resolution Geometry Representation (G-Rep) is built. Its main components are a Cell-Based Spatial Representation (CSR) that provides spatial filtering at the desired feature resolution, and an Axial Shape Graph (ASG) that captures local shape information as well as global information about the overall geometric structure of the object. The CSR and ASG components are calculated at multiple resolutions and linked together to form the G-Rep hierarchy. In the second step, the Axial Shape Graph is decomposed into a tree representing the overall shape structure of the object as a hierarchy of subcomponents. Using the Axial Shape Graph, the task of shape decomposition is reduced to a graph partitioning problem whose solution results in a well-balanced part hierarchy.

The second subprocess constructs a hierarchy of task-specific representations utilizing the geometric information provided by the G-Rep. The entire representation generation framework is driven by metrics that quantify the desirable characteristics of representations for the particular task.

We have tested this framework with the implementation of an application that generates representations specialized for the task of Collision Detection in 2D environments. For this task, the desired output representation is a balanced hierarchy of convex approximations that conservatively yet efficiently bound the input object. In addition, the axis-aligned bounding boxes of the approximations should be nearly invariant under object rotation. The metric used to guide the process is based on a quality/cost heuristic that attempts to generate simplified subcomponents that both minimize the area difference between the object and its composite convex hulls and produce bounding boxes of unit aspect ratio.

The application is based on a general *Representation Generation Framework* structured as a feedback loop that recursively generates potential representations for subcomponents of the object based upon its geometric structure. The resulting representations are evaluated at each step, and this information is used to guide the process. Figure 2 shows the basic steps of the Representation Generation Framework schematically. The first step *Decomposes* the object into two subcomponents. The next step *Generates* a representation for each subcomponent. These representations are then *Evaluated* in the next step according to quality/cost metrics. The values of the new representations are *Compared* to the value of the parent component, and if the improvement in value exceeds specified Thresholds, the representation is accepted. Finally, the subcomponent is removed from further processing if the representation meets the overall task quality requirements. If the quality is still below threshold, the subcomponent is sent back to the first step for further processing.

The remainder of this thesis is organized as follows. Section 2 presents the multi-resolution Geometry

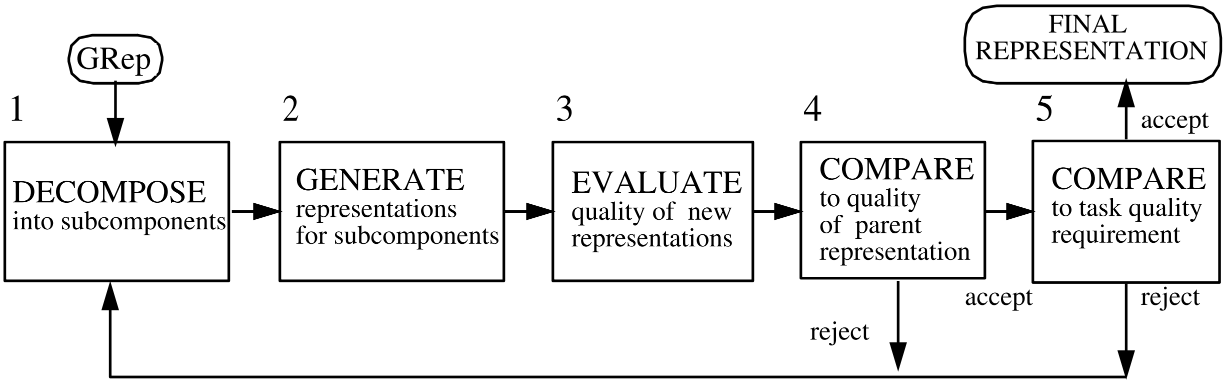


Figure 2: Representation Generation Framework

Representation and the process of G-Rep decomposition based on the Axial Shape Graph. Section 3 describes the construction of the G-Rep Cell-Based Spatial Representation and Section 4 describes the construction of the Axial Shape Graph. Section 5 describes an implementation of ASG-based G-Rep decomposition. Section 6 discusses the use of this framework in the generation of representations for Collision Detection. The final section presents conclusions and discusses future directions (Section 7) of this work.

2 Multi-Resolution Geometry Representation

The first process of Object Abstraction requires the extraction of relevant information from the input model. For geometric tasks, the solid properties of an object are important: the space that the object occupies, and its local and overall shape at resolution levels appropriate for the particular task . A boundary representation only implicitly contains this information, and is therefore not suitable for our purposes. Our approach constructs the multi-resolution Geometry Representation (G-Rep) hierarchy from the input object, which is designed to represent geometric features of the object explicitly.

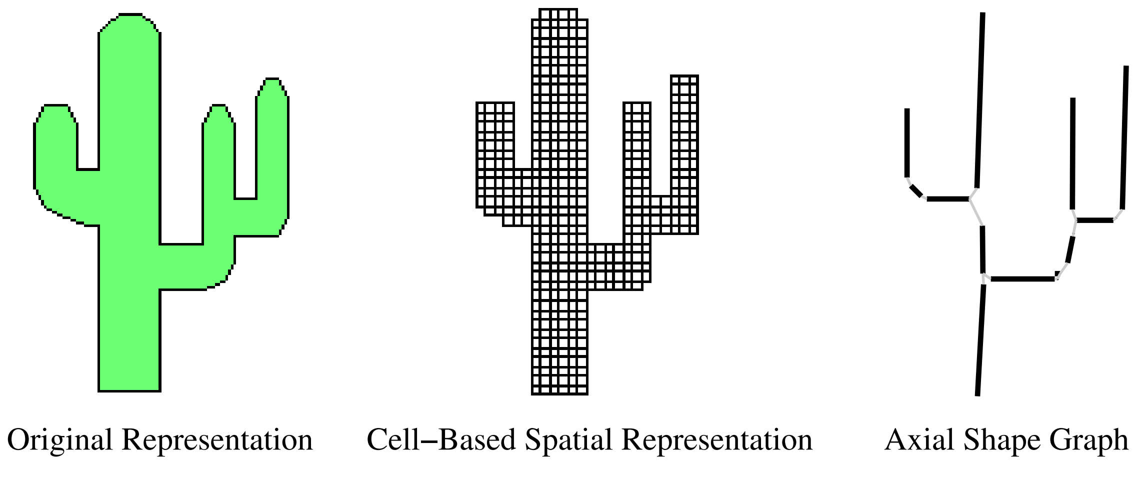


Figure 3: Components of the Geometry Representation (G-Rep)

The Geometry Representation utilizes a combination of simplification and decomposition to characterize the hierarchical geometric structure of an object. A G-Rep consists of the following structures:

1. Original Representation

2. Cell-Based Spatial Representation (CSR)

3. Axial Shape Graph (ASG)

The first component of the representation is the input representation itself. This model is maintained as a reference throughout the various processes. The second G-Rep component is the Cell-Based Spatial Representation. This structure represents an object explicitly in terms of the cells that it intersects in a uniform grid. The third component, the Axial Shape Graph, is designed to encode the geometric structure of the object. It describes an object in terms of its fundamental geometric subcomponents and their spatial relationships. Figure 3 illustrates the different structures of the G-Rep for a simple input object.

The CSR and ASG components are calculated at multiple grid resolutions and linked together to form the G-Rep hierarchy. Figure 4 shows five levels selected from the hierarchy, with the CSR shown in the top row and the ASG structure shown in the bottom row for each level. The following sections discuss the CSR and ASG representations as well as related work in spatial and axial shape representations. Sections 3 and 4 describe the algorithms for the construction of the G-Rep from an input object.

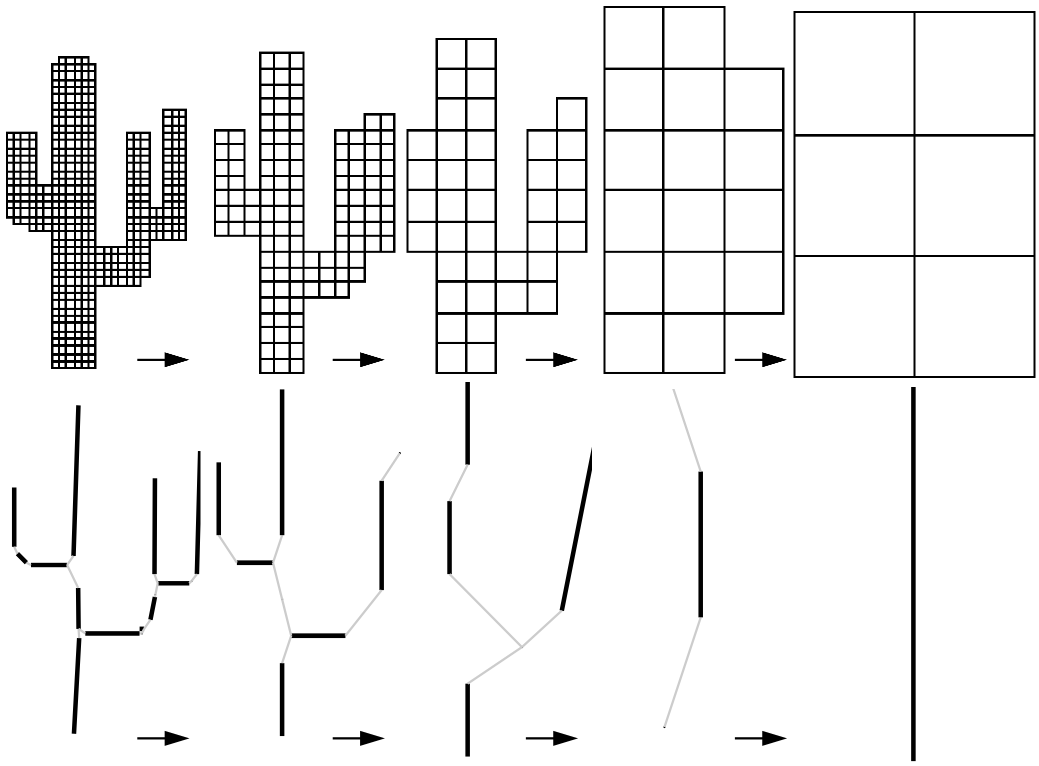


Figure 4: G-Rep structures in the hierarchy

The geometric structure of an object is best represented hierarchically. The Axial Shape Graph is used to decompose the G-Rep into a well-balanced tree of subcomponents. Section 2.3 discusses the decomposition process, and Section 5 describes implementation details.

2.1 Cell-Based Spatial Representation

The first derived component of the Geometry Representation is a Cell-Based Spatial Representation (CSR). The original object description is inserted into a uniform grid at a specified resolution. The spatial filtering inherent to the insertion process performs object simplification; any details smaller than the grid cell resolution are filtered out. If the cell size is chosen based on the resolution of the task, only those features relevant to the current task are preserved.

Consider the problem of planning a path through a 2D environment for a robot represented as a disc of radius r. In this application, the disc is tested against other objects in the environment to determine collisions as it moves about. If an object has narrow concavities of initial width $< 2r$, it is not possible for the disc to collide with any portion of the object along the interior of the cavity. All such features can therefore be removed when creating the representation to be used for this task.

A similar approach based on a spatial representation is utilized by He et al [19, 20] as a first step in the generation of simplified representations of 3D objects for rendering. The original object representation is sampled into a uniform 3D voxel grid and the result is low-pass filtered. This provides two levels of simplification. The discrete sampling removes details smaller than the voxel size, and the filtering process removes high frequency features, filling in cavities and holes and smoothing. Once the simplification has been performed, a boundary representation is constructed from the contents of the resulting buffer. This approach removes small features and may perform genus simplification. However, while the overall geometric structure is simplified, the final boundary representation may be more complicated than the original. Our Cell-Based Spatial Representation has one fundamental advantage over the voxel grid of He et al: our grid insertion process is not sample-based. We can therefore guarantee that the resulting representation is conservative, an essential property for the tasks that we are interested in.

There are additional advantages to performing this conversion process. The spatial representation models objects as solids with area, not just empty hulls, making it a scheme well-suited for extracting geometric information. Assuming the model has planar faces, the spatial representation is unique for a given input model at a given orientation, regardless of the particular choice of polygonization of the original boundary representation. In addition, many different representation types can be accepted as input (e.g. B-rep, mesh, pixel, CSG), if the proper insertion algorithm is provided. Groups of objects can be handled in the same manner as single objects.

There are also potential disadvantages to working with the spatial representation instead of the original. Any approach that discretizes the input is subject to certain artifacts: the same object at different orientations or locations may produce a different representation. Because we choose the cell size to be smaller than the error bound specified by the task, and because our output representations are conservative approximations, discretization artifacts do not cause problems in our approach.

In the G-Rep hierarchy, the cells of the grid are combined recursively to form the spatial representation for the next level. For efficiency, the grid is stored as an adaptive quadtree structure. During processing, the cells corresponding to the object boundary or its interior are expanded to the finest resolution leaf level, but the cells exterior to the object are kept in quadtree form.

2.2 Axial Shape Graph

The spatial representation provides explicit information about what area an object occupies, but for geometric tasks, we are also interested in the overall geometric structure of the object and in the relationships between its substructures. The second derived component of the G-Rep, the Axial Shape Graph, is designed to encapsulate this information in an easily accessible form. We want to expose the intrinsic hierarchical, geometric structure so that we can decompose the object into a balanced tree of subcomponents, where each subcomponent corresponds to a *fundamental part* of the object based on its shape.

For our purposes, a *fundamental part* is defined to be a spatially coherent area of the object whose boundary at a specified resolution contains no concavities. The parts are fundamental in the sense that their union covers the object with minimal overlap between parts. Such subcomponents are convex, and can be represented up to some resolution by a linear axis segment with an associated width defined at each segment point (or a point and a radius in the limit). We call these parts generalized *bars*.

2.2.1 Review of Axial Representations

The Axial Shape Graph is based on an axial object description composed of linear axis segments that correspond to the fundamental parts of the object. There is a vast amount of literature on axial representations (often called skeletons) of 2D images, originating with the work of Blum [3]. In this section we will therefore not attempt to give a complete review, but instead describe the work most relevant to our approach.

Many of the skeleton construction methods are based on one of several general approaches: iterative or parallel thinning [23], analytical calculation of the medial axis (Delaunay triangulation and calculation of Voronoi regions [26, 21]), and calculation of a Distance Map and its Medial Axis Transform (MAT) [3, 13, 1, 25, 36, 30, 22, 33, 12, 27]. Our implementation is of the last type, and thus this section focuses on MAT-based methods.

MAT-based methods belong to a class of approaches that represent a shape by a spine and a geometric primitive [31] that moves along the spine sweeping out the shape, possibly changing its size as it moves. The methods differ in the definition of spines and the generator primitive. In MAT-based methods, the generator is a disc.

The MAT approach to skeletonization first calculates the Distance Map for the object; each pixel is labeled with the closest distance to the exterior of the object. The set of local maxima in this Distance Map is then identified. The final skeleton consists of these pixels with the possible addition of the set of pixels necessary to form a connected structure.

Niblack et al [25] present a method for generating skeletons for binary images based on a distance transform. In this approach, skeletal points are defined to be the local maxima in the distance map, where a local maximum is any pixel with distance value greater than or equal to the distance value of any of its eight nearest neighbors. The skeleton is connected by climbing ridges. Each pixel is assigned a *climbing neighbor*, which is the neighbor representing the next step along an ascending ridge. If a pixel is a local maximum, there are no ascending neighbors. In this case, the climb starts by first taking a step to any equal valued neighbors. The local maxima are connected by climbing paths between local maxima and out of saddle points. (Section 4.2.1 discusses this process in more detail). The skeletons generated have the following desirable properties: they are guaranteed to have the same connectivity as the object, they allow reconstruction of the object, and they are minimal (under the constraint of the previous property).

The skeletonization approaches of Dorst [13], di Baja and Thiel [12], and Shih and Pu [33], are all based on the same basic algorithm, with slight variations in the following three aspects:

1. the choice of distance metric,
2. the definition of local maxima,
3. the method for assembling the points into a connected skeleton.

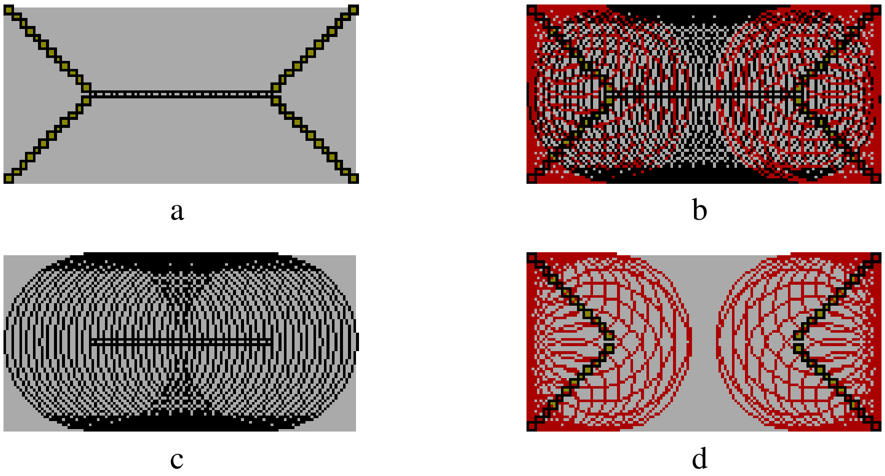


Figure 5: Maximal Discs: a) Object with local maxima highlighted b) Maximal Discs c) Discs of strong local maxima d) Discs of weak local maxima

In the skeletonization method of di Baja and Thiel [11], and Arcelli and di Baja [1], the local maxima are defined to be loci of the centers of *Maximal Discs* of the object. Maximal Discs are defined relative to the distance map. A disc can be associated with each pixel; the disc is centered on the pixel and has radius equal to the distance value of the pixel. The shape of the disc is dependent on the distance metric used (see Figure 13 for examples of the discs defined by different metrics). A disc is maximal if no other disc completely covers it. Arcelli and di Baja prove that the union of the set of Maximal Discs is equivalent to the object. The set of local maxima , along with the corresponding distance values, therefore provides sufficient information to exactly reconstruct the object. Figure 5a shows a simple object with the local maxima pixels highlighted. Figure 5b shows a representation of the Maximal Discs superimposed on the object. Figures 5c,d show the discs associated with subsets of the maxima. The skeleton is treated as a curve in 3D (x,y, and width are the three components for each pixel) which is divided into linear segments. Each linear segment represents an *elementary region* which is a component of the object with constant orientation of the skeletal spine, and monotonically changing width values. An approximate representation of each elementary region can be generated by taking the convex hull around the two discs associated with the segment endpoints. Merging and deletion operations are performed on the set of segments to simplify and remove redundancy in the resulting region representation. Deletion occurs in two forms: branch segments that correspond to non-significant protrusions are pruned and internal segments that correspond to regions that are mostly overlapped by other regions are marked as *linking* segments. *Linking* segments are not used to represent regions, only to represent spatial relationships between regions.

Other axis-based schemes are the Smoothed Local Symmetries (SLS) of Brady and Asada [5], and Brooks' generalized ribbons [31]. The generator for Brooks' ribbons is a line segment constrained to stay at a fixed angle to the spine. The generator for SLS is a line segment that makes equal angles with the surface normals at the two points where it intersects the shape boundary. In this case, the spine/generator characterization is a useful way of describing the shape, and not a means of generating it.

Kimmel et al [22] present an alternate skeletonization method based on the MAT. The object boundary is first segmented at points of maximal positive curvature. A distance map is calculated for *each* of the resulting boundary segments. The skeleton is calculated by taking distance map differences of maps generated by different boundary segments and marking the zero values as skeletal points (i.e. the zero values correspond to points that are equidistant from two or more boundary points).

Pizer et al [6] present a related method of representing an object by axes with associated widths called *cores*. *Cores* are computed by assigning pixels *medialness* values at multiple scales, and then locating ridges in the medialness function. The medialness function is invariant to translation, rotation and zoom, and therefore the resulting *cores* do not have the sensitivity to object and noise perturbations that is present in many other axial representations.

2.2.2 Axial Shape Graph Structure

The Axial Shape Graph is derived from the spatial representation and is designed to model the object as a connected set of bar subcomponents. Each bar is represented by a linear axis segment with the associated object width stored at each point along the segment.

We follow the work of Arcelli and di Baja [1] and define an *Axis Segment* to be the loci of the centers of *Maximal Discs*. A Maximal Disc is an inscribed disc that touches the boundary of the object in at least two places. Each axial point making up an Axis Segment is the center of a disc whose radius is the distance to the closest boundary points. Each axial point is therefore associated with two or more boundary points. Bars and their associated axes are similar to the *elementary regions* and corresponding segments of di Baja and Thiel [11].

The graph structure is composed of the following components:

- **Nodes**: represent Axis Segments and the associated bar components. When referring to the ASG, the terms *node* and *axis* are used interchangeably.
- **Edges**: connect each pair of nodes that represent the Axis Segments of adjacent object features, or *bars* that meet at a concave boundary region. With respect to a particular Axis Segment, there are three types of edges:
  - **Left Edges**: The corresponding adjacent object component is linked to the left/bottom end of the axis.
  - **Right Edges**: The adjacent component is linked to the right/top end of the axis.
  - **T Edges**: If two bar components meet at *two* concave regions (e.g. a "T" junction), one end of the edge is linked to a point interior to one of the two Axis Segments. These edges are labeled as a *T* edges. T edges are stored with the axial point that they are linked to. A node may therefore have several groups of T edges, where each group corresponds to a single axial point.

Figure 6a shows an example object with three Axis Segments. Dotted lines connect axes that represent adjacent object components. Figure 6b is a representation of the object components associated with each axis. The apex points of the major concavities are indicated on the figure. Figure 6c shows a representation of the graph structure. There are three nodes corresponding to the three axes. There are two major concave points between component 1 and component 2, and the connecting edge is of type T where it joins node 1 and type Right relative to node 2. There is a single concave point between component 2 and component 3. The edge connecting the two nodes is of type Left where it joins node 2 and type Right relative to node 3.

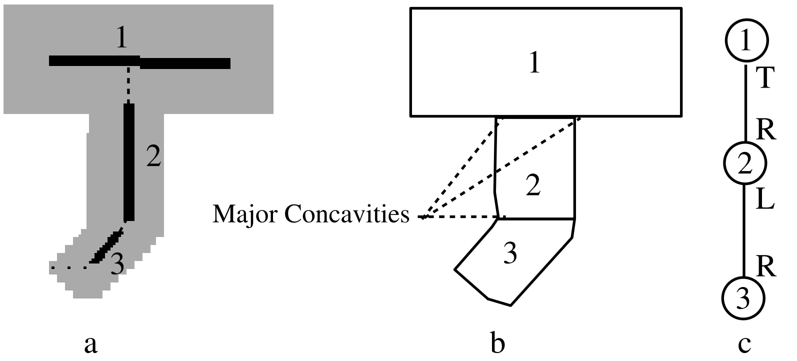


Figure 6: Axial Shape Graph: a) Axis Segments and connections b) Associated Components c) Graph Representation with Edge Labels

For each node n we define the *Left Graph* of n to be the set of all nodes reachable from the Left edges of the node. The *Right Graph* contains all nodes reachable from the Right edges. The Left, and Right Graphs may consist of a set of disjoint graphs if there are multiple edges adjacent to the node of type Left, or Right respectively. A *T Graph* is defined for each group of T edges associated with a particular axial point. Each T Graph contains the set of nodes reachable from that set of T edges.

The Axial Shape Graph has the following properties:

1. It has the same connectivity as the object at the specified level of simplification.
2. Nodes correspond to fundamental parts of the object.
3. Edges capture the spatial relationships between fundamental parts.

Many of the previous approaches we have discussed [25, 13, 12, 33] generate connected skeletal structures. The skeletons produced consist of a collection of *branches*. Skeletal points are categorized according to their connectivity to other skeletal points: *end* points have a single skeletal neighbor, *normal* points are interior to a skeletal branch and thus have exactly two skeletal neighbors, and *branch* points have three or more skeletal neighbors.

In these approaches, the points are not further organized into higher level components. The work of di Baja and Thiel [11] most closely resembles our axial structure. Once the skeletal points are identified, they are collected into linear segments. The resulting segments are divided into two classes: those that have "representational power" (i.e. correspond to an *elementary region*) and those that are *linking* segments only used for showing spatial relationships. A segment is classified as *linking* if the region it represents is largely covered by other regions (i.e. it is not *elementary*). The region associated with a segment is the union of the Maximal Discs of the segment skeletal points. The regions have significant overlap and are thus not fundamental according to our definition.

2.2.3 Axial Shape Graph Weighting

The Axial Shape Graph is a weighted graph. Each node stores four types of weights: an *Axis Complexity Weight*, a *Left Complexity Weight* associated with the left end of the Axis Segment, a *Right Complexity*

Weight associated with the right end of the segment, and a *T Complexity Weight* for each axial point that has one or more T edges connected to it .

The *complexity* of an axis is measured in three different ways: Size Complexity (C_s), Count Complexity (C_c), and Spatial Complexity (C_{bb}).

1. **Size Complexity**: The estimated size of the subcomponent *bar* corresponding to the Axis Segment. This value is directly proportional to the axis length and distance to the closest boundary points.
2. **Count Complexity**: The number of axes (this is trivially 1 for a single axis).
3. **Spatial Complexity**: The area of the axis-aligned bounding box around the associated subcomponent.

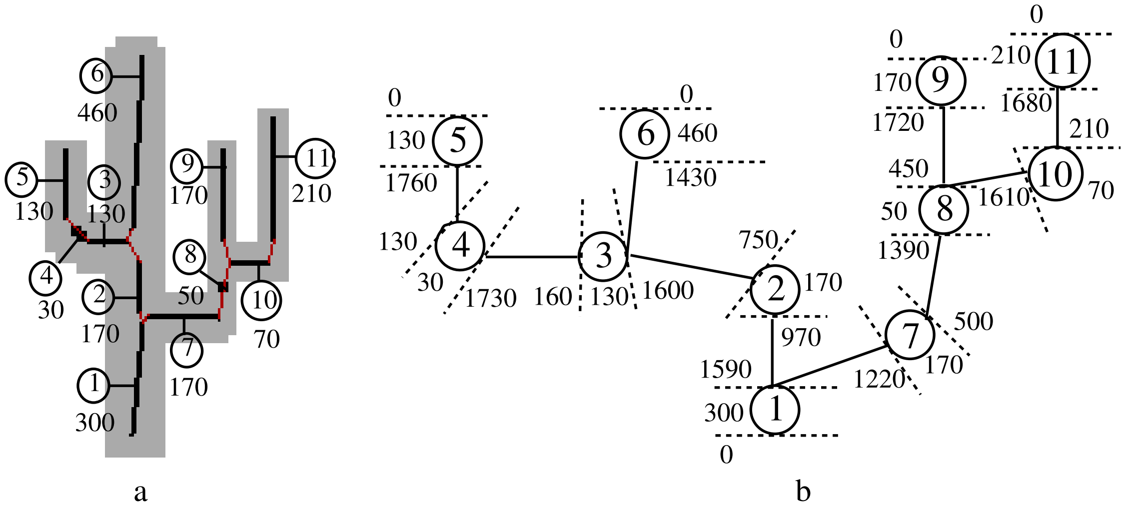


Figure 7: Axial Shape Graph: a) Axis Size Complexity Weights b) Axis, Left and Right Complexity Weights

The Axis Complexity Weights are calculated for each axis using one of the above measures. The Complexity Weight of an ASG subgraph is the sum of complexities of all axes that are part of the subgraph. Any reference to ASG *complexity* in general is assumed to refer to Size Complexity.

Three additional types of complexity are calculated for each node, corresponding to the three edge types.

- **Left Complexity Weight**: The sum of the complexity weights of all nodes in the Left Graph.
- **Right Complexity Weight**: The sum of complexity weights of all nodes in the Right Graph.
- **T Complexity Weight**: The sum of complexity weights of all nodes in the T Graph.

These values are utilized by the graph decomposition process.

Figure 7a illustrates the Axial Shape Graph for the cactus. The axes are numbered for reference (in the circles). The number below the reference number is the Axis Complexity, measured according to

Size Complexity. Figure 7b shows the graph representation for the same example. Three values are shown with each node, corresponding to the Left Complexity, Axis Complexity, and Right Complexity, respectively. The Axial Shape Graph contains no T edges in this example.

At each node, the Axis Complexity Weight is a measure of the shape of the object in its neighborhood, providing local geometric information. The Left,Right and T Complexity Weights provide global information about the geometric structure of the object as a whole.

2.2.4 Axial Shape Graph Multi-Resolution Information

An Axial Shape Graph is constructed for each resolution of the spatial hierarchy. These structures are linked across resolutions in two ways. First, nodes at one level in the hierarchy are linked to nodes of the ASG one level up in the hierarchy if the corresponding axes represent the same bar component. Second, multi-resolution information is captured locally at each level by assigning each Axis Segment a *Feature Weight* corresponding to the number of levels in the hierarchy that it appears in. Figure 8 shows the cactus with the Feature Weights indicated next to each axis. The axis corresponding to the base of the cactus is shown linked across four hierarchy levels.

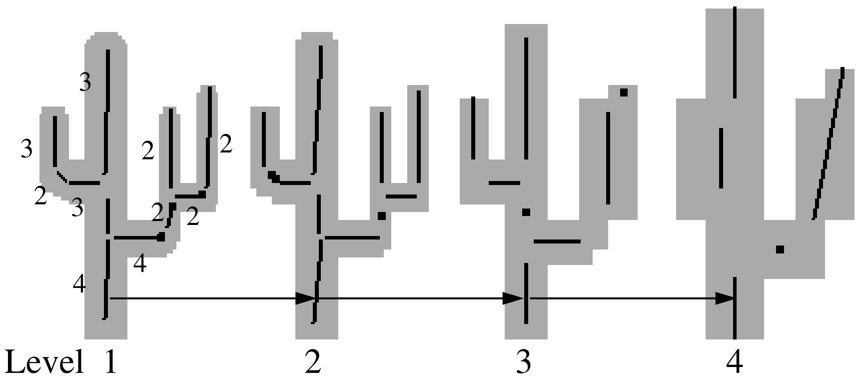


Figure 8: Axial Feature Weights: Axes for four levels of the hierarchy are shown. The axes of the first level are labeled with their Feature Weights. The axis corresponding to the base is linked across all four levels.

2.3 Hierarchical Decomposition of the Geometry Representation

Most objects can naturally be represented as a hierarchy of subcomponents. The ideal decomposition will depend on the particular task that we are interested in. There are two main qualities that seem intuitively desirable: Shape and Balance. The relative importance of each will be weighted according to the needs of the task.

1. **Shape**: The subcomponents should correspond to fundamental parts of the object.
2. **Balance**: The hierarchy should be balanced relative to the following measures
   (a) **Size**: total size of the subcomponents
   (b) **Count**: number of subcomponents

(c) **Spatial** : total area of axis-aligned bounding boxes of the subcomponents

A partitioning based on the first property corresponds to the process of shape decomposition. In our environment, this is equivalent to partitioning an object into its component *bars*. This is the most important property. The second property favors balanced trees. Property 2a requires that the sum of the areas of the subcomponents of each branch be roughly equal. Property 2b refers to the number of subcomponents in the tree, and implies that, on average, the same number of levels will need to be traversed to reach any leaf component. This is a desirable property for many applications, but often will not lead to the most natural decomposition relative to the connectivity of object components. Property 2c refers to the spatial characteristics of the partitioning and favors decompositions where the children of each subbranch are spatially clustered. It is clear that it is not necessarily possible to meet all of the above properties, and the decomposition process will need to make tradeoffs according to the relative importance of each to the task.

The G-Rep is well suited for decomposition based on the stated properties. The information contained in the G-Rep structure is approximate, but provides very strong hints to the local and global geometric structure of the object. The decomposition process utilizes both local and global geometric characteristics and connectivity information to determine how to make the partitions to produce a well-balanced part hierarchy. The global information provided by the G-Rep structure can be used to infer in general where partitioning should occur. The local information is used to decide how to make the actual partitioning cut.

The information contained in the G-Rep can be utilized to meet the requirements specified by the above properties. The axes of the Axial Shape Graph approximately identify the fundamental parts of the object. Property 1 can therefore be satisfied by partitioning the graph at axis endpoints. The weighting of the ASG provides information that can be used to achieve Property 2. Each type of balance measure (i.e. Size, Count, and Spatial) corresponds to one of the three Complexity Weights: Size Complexity, Count Complexity and Spatial Complexity, respectively. We utilize the information suggested by the Complexity Weights to achieve a balanced hierarchy by casting the decomposition process into a graph partitioning problem based on these weights. At each step in the decomposition, a partitioning point is chosen in the graph such that the resulting two sets of subgraphs have roughly equal weight.

We are not guaranteed to find an optimal decomposition, both because the weights themselves are approximations, and because the best partition may not even belong to the set presented by the graph. In practice, however, this information provides valuable hints, which when used in conjunction with evaluation metrics, produce well-balanced hierarchies. Section 6 discusses the decomposition process in more detail, as it is applied to a specific task and shows examples of representation hierarchies generated by an application based on the G-Rep and the general decomposition process we have described here.

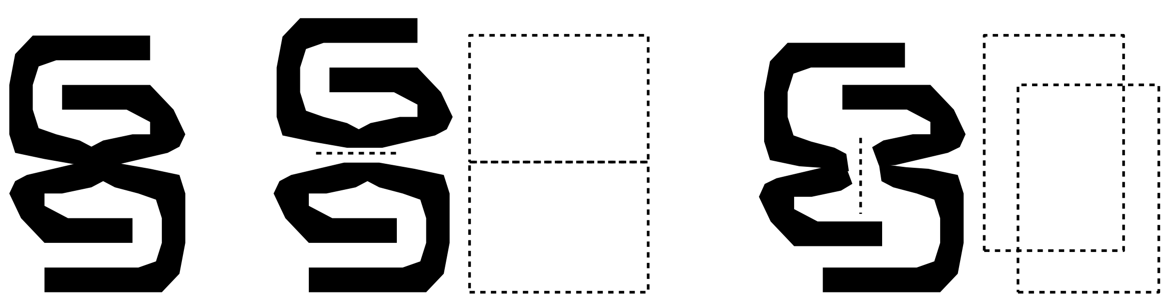


Figure 9: Two possible object decompositions based on shape and Complexity Weights

Figure 9 illustrates two possible decompositions of an object. It is assumed that the desired result is a balanced binary tree, and that partitions are made only on shape boundaries. The original object is shown at the left. The decomposition cut is indicated in the middle and right figures as a dotted line, and the resulting axis-aligned bounding boxes are shown to the right of each decomposed figure. In both cases, the cut produces subcomponents of equal size. The cut on the right, however, produces subcomponents with relatively poor spatial locality, as evidenced by the overlapping bounding boxes. The middle figure meets all of the balance criteria. In practice, it will not always be possible to produce a decomposition tree whose subcomponents respect the shape structure of the object and are balanced by Size, Count, and Spatial Complexity,

The G-Rep is also used to determine where to make the actual partitioning cut. It is generally accepted that the partitioning of an object into subcomponents based on shape should occur at concave points. It is not sufficient, however to simply connect such points. In the ASG structure, the axis endpoints, and points with local width minima along the axes, imply the presence of concave points. The concave points correspond to the closest boundary points that were used to define the axis points. Pairs of such boundary points are chosen as candidates for partitioning.

The decomposition process is recursive; at each iteration, the global graph structure is utilized to determine the general location of a partition, and then the local axial information is utilized to make the actual cut. Once the partition is made, the ASG is regenerated for each subcomponent, and the process continues on the subcomponents. In this section, we have outlined the general process of G-Rep decomposition. In practice, each phase of the decomposition is guided by task-specific metrics. Section 5 discusses the implementation of general G-Rep decomposition and Section 6 discusses the details of the process when utilized for the task of Collision Detection.

3 Construction of the Cell-Based Spatial Representation

This section discusses the construction of the G-Rep Cell-Based Spatial Representation from a two-dimensional input representation. The work for this thesis was primarily based on input files in the UniGrafix (UG3) [9] format, and on images in the Silicon Graphics RGB image format. We discuss the construction of the spatial representation from these two types of input.

3.1 Polygon Insertion

In the case of polygonal input, the B-Rep based object description is inserted into a grid at a specified resolution. The insertion process we have implemented is a scan-line approach designed to be conservative: a grid cell is considered part of the representation if and only if it is touched by some part of the object. This differs from traditional conversion algorithms, which are based upon discrete sample points [14].

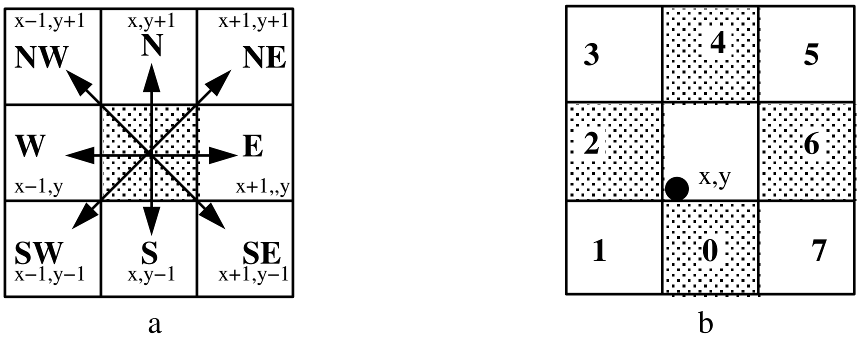


Figure 10: Cell terminology: a) cell (x,y) and its 8 nearest neighbors b) Neighbor labeling: Direct Neighbors $N_D = \{0, 2, 4, 6\}$, Indirect Neighbors $N_I = \{1, 3, 5, 7\}$

We first define some relevant terminology as it is used in the remainder of this thesis. The neighbors directly adjacent to a cell (in the South, West, North and East directions) are referred to as *direct* neighbors, N_D. The diagonally adjacent neighbors (in the South West, North West, North East, and South East directions) are *indirect* neighbors, N_I. The neighbors are labeled 0-7, corresponding to a clockwise order from S-SE, for reference. With this numbering, the direct neighbors are $N_D = \{0, 2, 4, 6\}$, and the indirect neighbors are $N_I = \{1, 3, 5, 7\}$. The term neighbors is always meant to refer to the eight closest neighbors as defined above. For neighbor direction i, the direction opposite to i is denoted $\bar{i}$ and is calculated as $((i + 4) \mod 8)$. Figure 10a illustrates the spatial relationships between a cell and its nearest neighbors. Figure 10b shows the numerical labeling of the neighbors, with the direct neighbors indicated by the shaded cell regions.

A cell that is not part of the object of interest is an *exterior* cell. Cells that are part of the object, and totally contained within the object are called *interior* cells. Object cells with one or more neighbors that are *exterior* are *boundary* cells. Grid cells are addressed by the (x, y) value of the lower left-hand corner.

The insertion process accepts a list of polygon vertices, assumed to be in counter clockwise order. The basic algorithm follows the standard polygon rasterization approach exploiting edge coherence: the outer loop processes one scanline at a time in the y direction, while the inner loop fills in spans between edges in the x direction. The algorithm is outlined below.

1. Sort edges on minimum y coordinate into cell-sized buckets, with each bucket Edge List (EL) sorted on minimum x coordinate
2. Initialize Active Edge List (AEL) to NULL, and set $y = y_{min}$
3. While there are edges in AEL and unprocessed buckets:
   (a) Perform incremental x coordinate update for each edge in AEL for new y value
   (b) Add all edges starting at y into AEL sorted on x coordinate
   (c) For $x = minimum$ to $x = maximum$:
      i. If x in polygon, mark cell (x,y) as IN
   (d) Remove edges ending at y
   (e) Increment y

The only portion of the algorithm that differs from a traditional approach is the *in polygon* test in the innermost loop. Filling is done with the odd-even rule (i.e. subsequent pairs of edges are assumed to border an **IN** region). Slight modifications are necessary to ensure that the fill is conservative. The traditional algorithm computes the x value of the edge where it crosses the current grid line. The floor of this floating point value is taken as the current x value, and thus the test for inclusion is based on a single sample point at the lower left corner of the 1x1 grid cell. To ensure a conservative answer, we take the x value at both the current y value and at $y + 1$. This gives the range of grid cell(s) that the edge passes through between the two scanlines. An x grid cell is considered **IN** if

- An edge E passes *through* the grid cell addressed by (x, y), This is true if

  $$\lfloor (E_x(y)) \rfloor <= x < E_x(y+1) \text{ or } E_x(y) = E_x(y+1)$$

  where $E_x(y), E_x(y+1)$ are the calculated x intersection of the edge E with the grid lines y and $y + 1$, respectively. All floating point comparison operations are performed within an ϵ value to combat round-off error.

- The x value lies interior to a fill region. **FILL** is turned on between subsequent pairs of edges on the scanline according to the following paradigm:

- Each edge passing through a single cell, as they are processed in left-right order, causes a toggle of the **FILL** value.
- Horizontal edges are not used to determine **FILL**.
- If the initial edge or edge point on a **FILL** coincides with a grid line, the cell will be marked **IN**. If the final edge or edge point falls on a gridline, the final cell will not be included.
- For the initial scanline, the **FILL** value is calculated at $y + 1$, for all subsequent scans it is calculated at y.

The final output of the insertion process is an adaptive quadtree structure. When the insertion process begins, the initial representation is the root of a quadtree with dimensions sufficient to enclose the bounding box of the input polygon. Each time the insertion routine makes a call to *mark (x,y)*, the quadtree is traversed to find the 1x1 grid cell addressed by (x, y). As the traversal proceeds down the quadtree levels, the quadrants are expanded if necessary until the desired leaf cell is reached. In this way, the quadtree is expanded on demand, only creating a cell at the finest resolution if it, or one of its three neighbors in the same parent, is touched.

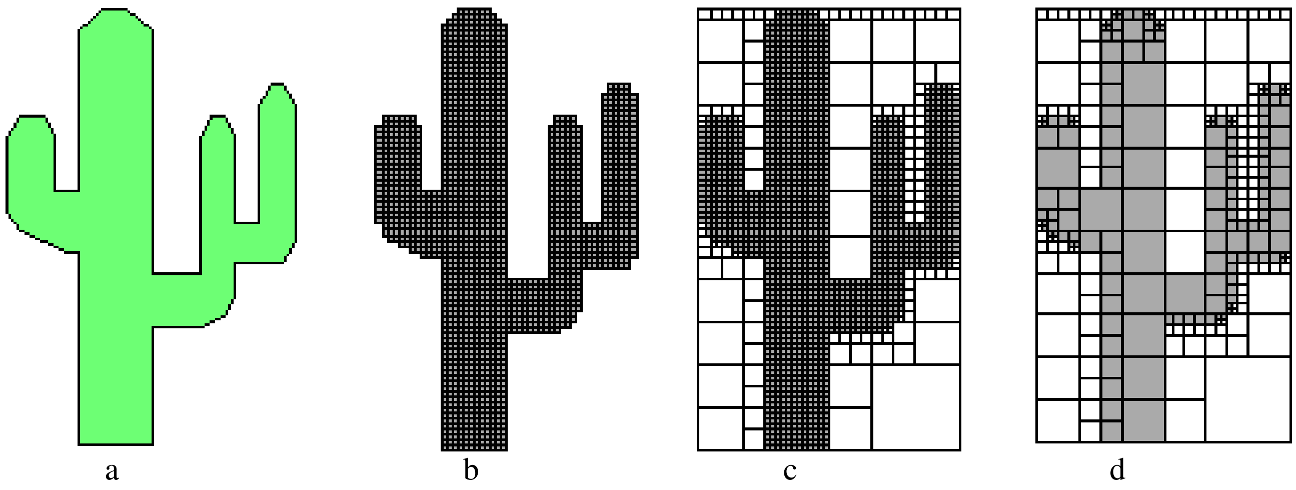


Figure 11: a) Original object b) Grid Cell Representation c) Adaptive Background d) Fully compressed quadtree representation

Once the object is inserted into the grid, a fill process is run to remove any holes interior to the object. For the task that we are considering (collision detection), we assume such areas are unreachable, and therefore need not be represented.

In the G-Rep hierarchy, the cells of the grid are combined recursively to form the spatial representation for the next highest level.

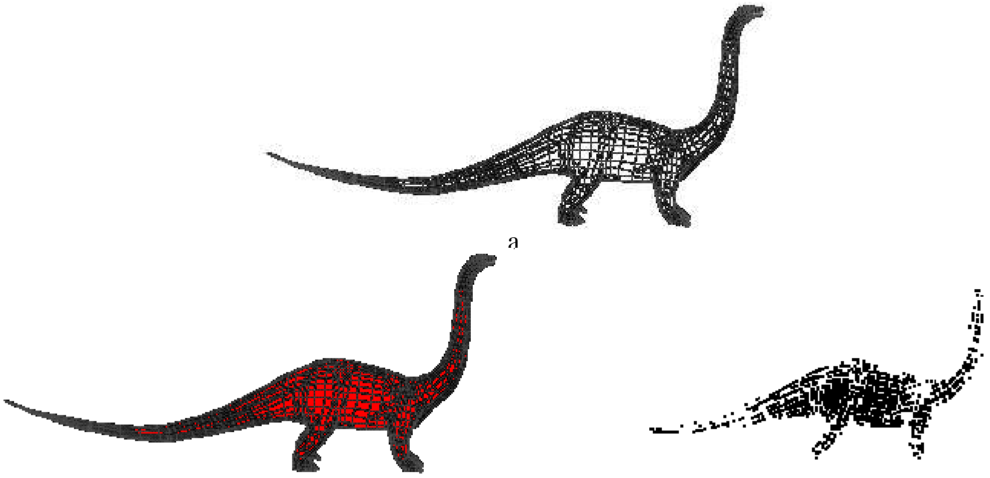
The grid cells belonging to the object are left in expanded, non-adaptive form while being used for processing. To make input and output operations and storage more efficient, the grid cells are compressed adaptively and the object quadtree structure is stored in linear form in an ASCII file format. Figure 11a shows an input polygon, Figure 11b is an image of the resulting **IN** grid cells. Figure 11c shows the background in adaptive form, and Figure 11d shows a representation of the fully adaptive quadtree form used for I/O and storage purposes.

Each cell stores pointers to its eight nearest spatial neighbors to facilitate traversals. Due to the adaptive grouping of cells, a cell's neighbor may be at a different resolution. Neighbor pointers always point to a cell of the same size or larger. If pointers were allowed to point down to higher resolution cells, a cell would have multiple neighbors in a particular direction. If a cell's neighbors are smaller, the pointer is set to the first parent cell encountered in an upward traversal of the hierarchy that is of the same resolution.

3.2 Image Insertion

In addition to the polygonal boundary representation insertion approach described above, we have also implemented an insertion routine that accepts images of objects in the Silicon Graphics RGB format. In the images used for this work, the objects of interest could be trivially separated from the background (e.g. colored objects on a white background), so image segmentation is not an issue.

The initial image is used as the occupied grid cells corresponding to the most detailed level of the quadtree. The cell resolution is assumed to be the original input resolution of the image. A fill process is run on the grid; this not only fills in internal holes, but also correctly relabels as *interior* any cells internal to the object that happened to have the background color and therefore were labeled as *exterior* initially. Background cells are combined adaptively to compress the representation. Finally, the cell blocks are combined recursively to form the G-Rep hierarchy. Figure 12a shows an example input image file of a dinosaur. Figure 12b is the resulting spatial representation. Figure 12c shows the cells added in the fill operation.


a

b c

Figure 12: a) Original object b) Grid Cell Representation c) Fill cells

4 Construction of the Axial Shape Graph

The Axial Shape Graph is the second derived component of the G-Rep and is constructed from the Cell-Based Spatial Representation in two stages: Axis Segment Creation and Graph Construction.

In the first stage, *Axial Points* are identified in the spatial representation using a method based on the Medial Axis Transform (MAT) [32], and connected to form *Axis Segments*. The identification of Axial Points proceeds by first calculating the Distance Map (DM) of the spatial representation. Local Maxima are identified in the DM by applying the Medial Axis Transform. Axial Points are derived from the resulting local maxima and then collected into Axis Segments.

In the second stage, the Axial Shape Graph is constructed by mapping the Axis Segments to graph *nodes*, and connecting with *edges* those nodes that correspond to adjacent geometric subcomponents by following ridges in the Distance Map. Once the graph is formed, the graph is weighted by assigning Axis Complexity Weights representing the size of the geometric subcomponents and Left, Right and T Complexity Weights based on graph connectivity.

The algorithm can be broken down into the following steps:

1. Axis Segment Creation
    - Identify Axial Points
        - Calculate Distance Map DM
        - Find Local Maxima M in DM
    - Connect Axial Points into Segments
2. Graph Construction
    - Form Connected Graph
    - Assign Weights to Graph Nodes
    - Link Graph Across Resolutions

The following sections discuss each of these steps in detail.

4.1 Axis Segment Creation

4.1.1 Calculating the Distance Map

The Distance Map (DM) is calculated for the object from the spatial representation. The DM labels each cell with the distance to the closest boundary cell on the object. An integer distance value is used as an approximation to the true Euclidean distance for efficiency. The DM calculation utilizes a *weighted* distance metric: a cell's direct neighbors are a distance d_D away, and its indirect neighbors a distance d_I away, where $d_D < d_I$. It has been shown [4] that if $d_D = 1$, setting $d_I = 1.351$ will produce the best approximation to the Euclidean distance. The closer the pseudo-Euclidean metric is to the true distance, the more robust the measure will be under object rotation. We have used both the 2-3 ($d_D = 2, d_I = 3$), and 3-4 ($d_D = 3, d_I = 4$) metrics for our study. Other integer distance metrics more closely approximate the Euclidean measure but include a larger neighborhood (e.g. 5x5 if the cell's *knight* neighbors (one move horizontal(vertical), followed by two vertical(horizontal) moves) are included). The 2-3 and 3-4 metrics are sufficiently accurate for our purposes and easier to calculate.

Figure 13 illustrates the different metrics by marking the set of cells that would approximate a circle around the center cell (i.e. the set of cells such that the distance from the center cell under the specified metric is less than or equal to the circle radius).

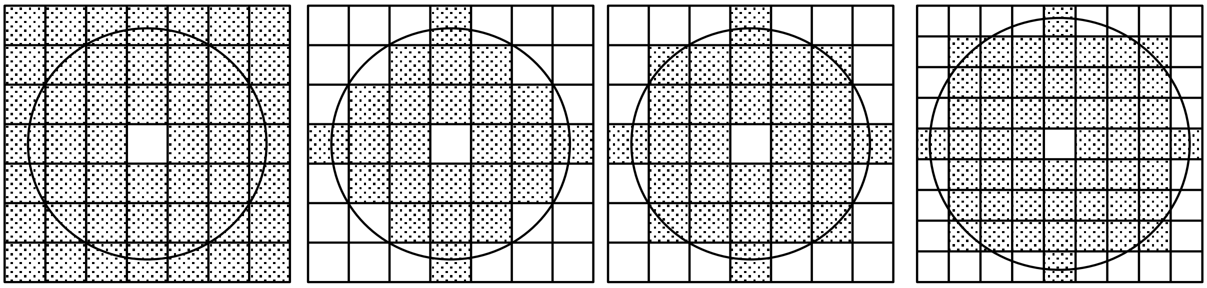


Figure 13: Distance Metrics: a)1-1 unweighted (r=3) b) 1-2 (r=3) c) 2-3 (r=6) d) 3-4 (r=12)

All processing occurs on object cells that are at the finest resolution grid level in the quadtree. If the input is not coming directly from the insertion process (which leaves the marked object cells at the finest level), the branches of the quadtree containing portions of the object are expanded down to the finest level in an initial pass. The external areas are left in adaptive form for efficiency.

We have implemented a two-pass algorithm to calculate the DM [10]. The distance value of each cell in the object is initialized to some large integer value, and exterior cells are initialized to zero. For each cell c, the distance value $d(c)$ is calculated by taking the distance value of each of the cell's eight neighbors and adding d_D if the neighbor is directly adjacent (N,S,E, or W) or d_I if it is a diagonal neighbor (SW,NW,NE, or SE). The cell's new distance is set to the minimum of these calculated distances. This operation is efficient because each cell stores pointers to its eight nearest neighbors. Two passes are made over the cells within the bounding box of the object: the first processes cells from top to bottom and left to right, the second processes cells from bottom to top and right to left. In each pass, the minimum distance to the boundary is updated for each cell.

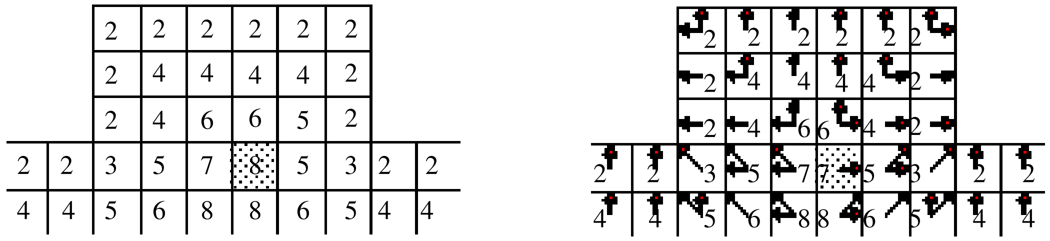
1. Pass 1:

$$d(c) = min(d(N_2) + d_D, d(N_3) + d_I, d(N_4) + d_D, d(N_5) + d_I).$$

2. Pass 2:

$$d(c) = min(d(c), d(N_6) + d_D, d(N_7) + d_I, d(N_0) + d_D, d(N_1) + d_I).$$

The quadtree structuring of the cells exterior to the object makes the sequential scanning process more efficient. When processing areas of the bounding box not covered by the object, the scan is able to quickly step over the larger scale exterior cells to reach the fully expanded cells of the object.



a b

Figure 14: Setting Direction Vectors a) Distance values after first pass b) Highlighted cell is missing NE direction vector

Each cell is annotated with the direction(s) from which it received its lowest distance value (N, NE, E, SE, S, SW, W, or NW). Following a path along the vectors representing these directions produces a shortest path to the object's boundary. The directions are stored with the cell in an 8-bit direction bit-vector and are referred to as the cell's *DM Direction(s) (DMD).* An additional pass is performed to set the Distance Map Directions for each cell. It is not possible to overlap the two-pass setting of the distance values with setting the DMD values, because not all of the final neighbor information is available until the completion of the second pass. Figure 14a shows an example object with the distance values after the first pass. Figure 14b shows the DMD vectors if they are set during the two passes. The highlighted cell is missing the NE direction vector because when the cell received its distance value, the neighbor cell in the NE direction had not yet achieved its minimum value.

Figure 15a shows the Distance Map for the cell representation of the cactus example with the increasing height values shown in greyscale. A close up of a portion of the figure is shown on the right with the DMD vectors to the closest boundary point indicated as arrows. Figure 15b shows the Distance Map represented as a 3D surface with the cell heights corresponding to distance values.

4.1.2 Identifying Local Maxima in the Distance Map

The set of *Axial Points* is derived from the Distance Map by applying the Medial Axis Transform (MAT). If the Distance Map is represented as a 3D surface, with the height corresponding to the distance values, the MAT is the set of *local maxima* M. Many skeletonization approaches [12, 1, 25], originating with the work of Blum on the SAT [3], utilize a distance map/MAT approach, but with varying definitions of the local maxima set M. We follow the definition of Arcelli and di Baja [1] where M is defined to be the centers of the set of *Maximal Discs* of the object. Any subsequent mention of a "maximum" in this report will be assumed to refer to such a local maximum.

Arcelli and di Baja show that the centers of the Maximal Discs are those pixels through which information does not "flow" during the computation of the distance function. In our environment, this corresponds to cells with no DMD vectors pointing into the cell. This is expressed computationally as follows: for each neighbor N_i of the cell, the $\bar{i}$th bit in the DMD vector of N_i is not set:

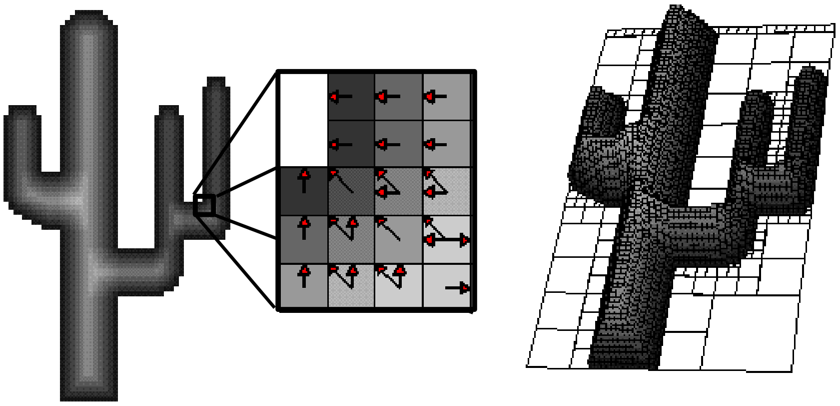


Figure 15: Distance Map: a) greyscale by distance with direction vectors. b) 3D representation of DM

$$\forall i \in [0,7] \quad B(DMD(N_i), \bar{i}) = 0$$

where $DMD(n)$ is the Distance Map Direction bit vector of cell n and $B(b,i)$ is the value of the ith bit of bit vector b.

This definition is also utilized by Niblack [25]. Niblack defines a *witness* to a cell c (their work is in terms of pixels, but the definition is equivalent in our cell-based environment) as any direct neighbor N_D with distance value $d(c) - d_D$ or any indirect neighbor N_I with distance value $d(c) - d_I$, i.e. those cells that defined c's distance value. Under this definition, the set of local maxima as defined above is composed of all *non*-witness cells.

We define three types of local maxima: *strong*, *weak*, and *strict*. A strong local maximum has one or more pairs of DMD vectors that are at 180°. Each pair of directions present is referred to as an *Axial Normal Direction* (AND) of the cell. The remaining set of maxima are referred to as weak local maxima. A strict local maximum has distance value d greater than the distance value of any of its eight closest neighbors. Strict maxima may be weak or strong.

The set of *Axial Points* is defined to be all cells that are strict or strong local maxima (they may be both). Our approach does not consider the non-strict, weak local maxima when forming Axial Segments. The strong maxima and strict maxima are indicative of the bar-like components of the object that we wish to represent. The weak local maxima do not necessarily correspond to geometric regions of interest and increase the complexity of the axial structure. We filter these cells out, resulting in a cleaner, more efficient axial representation that is sufficient to represent the geometric structure that we are interested in.

Figures 16a,c and e show both the weak and strong maxima cells for three example objects. Figures 16b,d, and f show only the strong maxima cells from the same examples. For the rectangle object, it is clear that the weak maxima cells do not correspond to any geometric subcomponents as we have defined them. In the case of the cactus and the dinosaur, the strong maxima capture the geometric structure well. The removal of the weak maxima cells in these two cases simplifies the representation.

Due to the discrete nature of the cell environment, a pair of DMD vectors defining a strong local maximum may meet at a single cell or at a pair of adjacent cells. In the latter case, the pair of adjacent

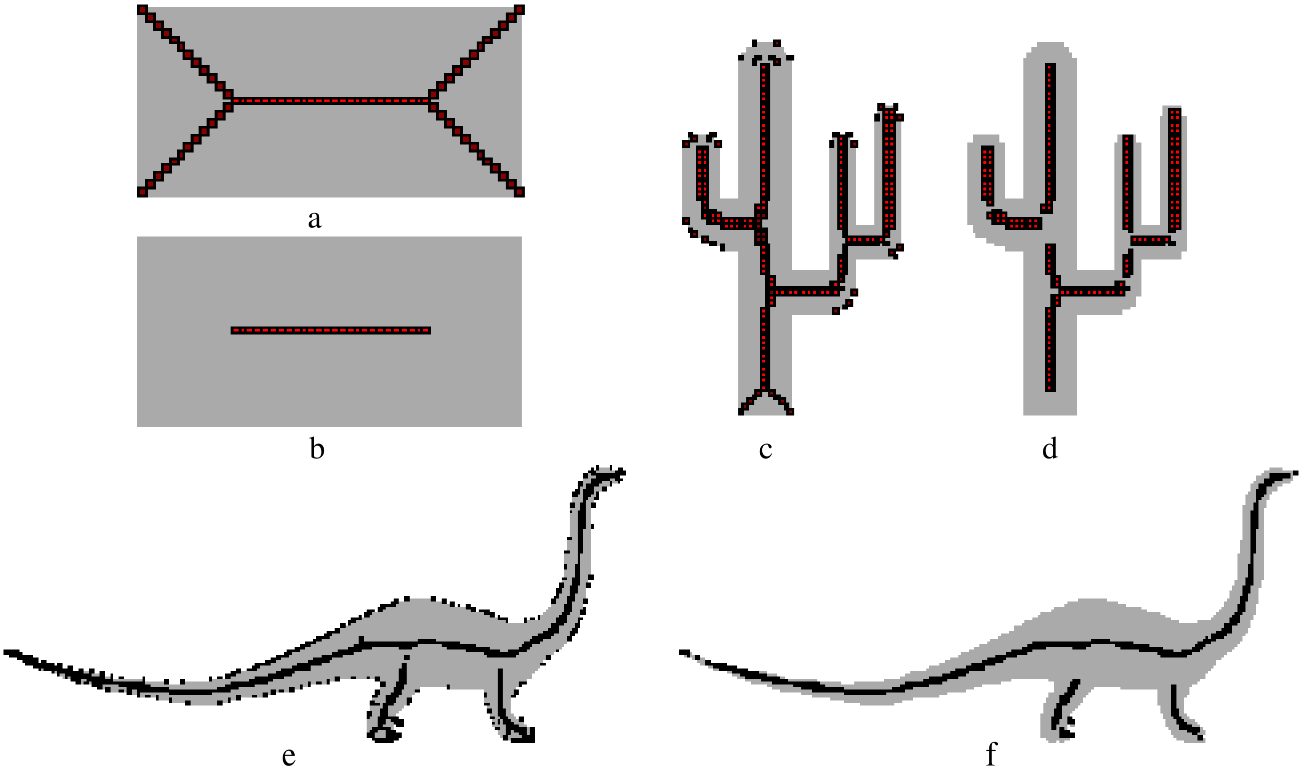


Figure 16: Axial Points: a) Rectangle Local Maxima cells b) Rectangle Strong Local Maxima cells c) Cactus Local Maxima cells d) Cactus Strong Local Maxima cells e) Dino Local Maxima cells f) Dino Strong Local Maxima cells

cells is treated as a single composite cell. Each cell is linked to its 180° partner. Figure 17a-d illustrates the possible DMD pairs that produce a strong local maximum. For each case, the upper figure shows the Distance Map Directions for a single-cell strong local maximum, and the lower figure shows the DMD when they meet at a pair of cells.

An axial cell may have any combination of 180° DMD pairs set. In this case, the composite cell may be formed from several adjacent cells, and the Axial Normal Direction will consist of multiple direction pairs. Figure 17e shows a composite cell with the SW-NE and N-S direction pairs leading to the closest boundary points. If *all* direction pairs are set, the object is "circular" within the resolution of the discrete approximation. In this labeling scheme, a cell that is a strong local maximum belongs to exactly one composite cell, and thus has a unique Axial Normal Direction that is the combination of the Axial Normal Directions of all the component cells. This property is used to collect the Axial Points into Axial Segments (see Section 4.1.3).

Axial Points are identified utilizing the Distance Map distance values and direction vectors. Each cell is first tested to see if it is a strong local maximum. The Distance Map Directions are stored as a bit vector, with the directions numbered in counter-clockwise order, starting with South (i.e. South is represented by bit 0, South-West by bit 1, etc.). A cell is a single-cell strong local maximum if it has one or more pairs of Distance Map Direction pairs (E-W, N-S, SW-NE, or NW-SE) set in the DMD bit vector. The test for a single-cell strong maximum is very efficient in our implementation and can be performed with a single bitwise `SHIFT` and `AND` operation. If any bit is set in the result, the cell is marked as an Axial

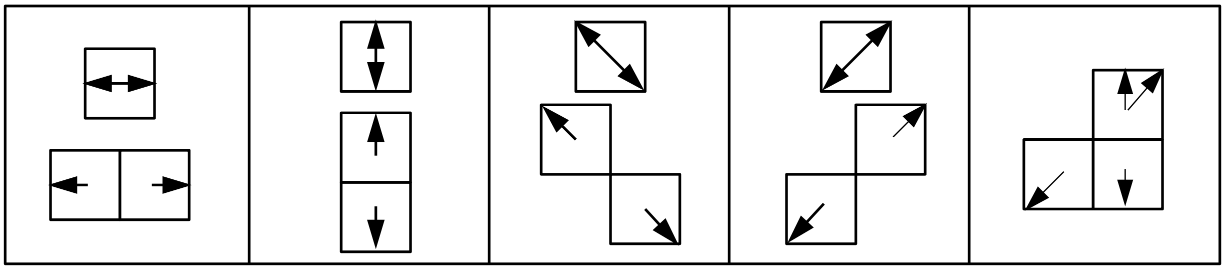


Figure 17: Axial Normal Direction Pairs a) E-W b) N-S c) NW-SE d) SW-NE e) N-S/SW-NE

Point and the DMD pairs are recorded.

In the case when DMD vectors meet at a pair of cells, the test requires two comparisons. If a cell has bit i set, and its neighbor $N_{\bar{i}}$ has DMD bit $\bar{i}$ set and does *not* have DMD bit i set, then the two cells form a composite pair with Axial Normal Direction $(i - \bar{i})$.

The identification of composite cells with multiple Axial Normal Directions is done by following transitive relationships after all single and pair composite cells are formed. We will use Figure 17e as an example. The lower left cell has DMD SW set, and it will be labeled in the first pass along with its neighbor in the NE direction as part of a composite pair of strong maxima cells with Axial Normal Direction SW-NE. Similarly, the top and bottom cells on the right will be labeled as a N-S composite pair. In a second pass, each strong maximal cell is examined to determine if any of its Axial Normal Direction partners have other Axial Normal Directions set. If so, the two sets are merged into a composite cell and the Axial Normal Direction vectors are updated accordingly. In this example, it would be discovered when processing the lower left hand cell that its partner, the neighbor in the NE direction, had Axial Normal Direction N-S set. The two pairs are therefore merged into a single composite voxel with Axial Normal Directions N-S/SW-NE.

If the cell is not a strong local maximum, it may still be an Axial Point if it is a strict local maximum. The test for a strict local maximum is also a local operation. A cell's distance value d is compared to the distance value of each of its eight closest neighbors. Strict local maximum are always composed of a single cell.

4.1.3 Creating the Axis Segments from Axial Points

Once the set of Axial Points has been identified, the corresponding cells are organized into linear Axis Segments. Axis Segments correspond to connected ridges in the 3D distance map and are composed of connected axial cells with the same set of Axial Normal Directions.

If a cell has Axial Normal Direction d, any neighbors it has in the direction approximately perpendicular to d, called the *Axial Direction*, that are also axial with Axial Normal Direction d, are potential nodes on the same axis. If an axial cell has a E-W DM direction pair, for example, we would look to neighbors on the North and South as potential axial neighbors. Figure 18a shows a set of axial cells (shaded grey) with E-W Axial Normal Direction. The Axial Direction is N-S in this case. In Figure 18b, the Axial Normal Direction is N-S and the Axial Direction is E-W. In Figure 18c, there are two Axial Normal Directions (N-S/SW-NE), and two corresponding Axial Direction pairs (E-W/NW-SE).

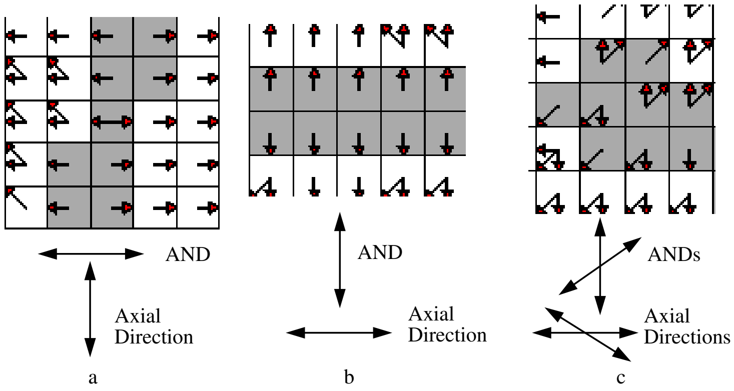


Figure 18: Axial Normal Directions (AND) and Axial Directions: a) N-S Axial Direction b) E-W Axial Direction b) E-W/NW-SE Axial Direction

We will now describe our implementation of Axis Segment Creation. In the following discussion, any reference to an axial cell is meant to refer to the entire composite cell where applicable. The axial cells are put in a list when they are identified in the previous step that performs the Medial Axis Transform. Axial cells are processed one at a time from the list. A new axis is formed with Axial Normal Direction equal to the Axial Normal Direction of the axial cell. Axes are built up and stored in left-right (bottom-top) order. The *initial end* cell for an axis is the cell that defines the left/bottom end of the axis. An initial axis end cell for a particular AND is an axial cell that does not have any adjacent neighbors to the left/bottom with the same set of Axial Normal Directions.

If the current cell is an initial axis end cell, a new axis is started from that cell. Cells are visited in the right/top Axial Direction to grow the axis (i.e. perpendicular to the AND). If these neighbors are axial cells with the same AND set, they are added to the axis.

If the current cell is not an initial axis end cell, cells are visited to the left/bottom Axial Direction until an initial axis end cell is found.

There is one additional caveat to growing the axis: it must remain linear. The axis is tested for linearity after the addition of each new cell. If the test fails, the new cell is not added to the axis, but instead the current axis is ended, and a new one started with the last cell as the initial axis end cell.

To test whether an axis is linear, we run a procedure to determine if a line segment exists that stays within the bounds of all of the cells on the axis. This test is implemented by casting the problem as a test for linear separability. For each cell that is added to the axis, we add a set of intervals to a list. The intervals correspond to the bounds of the cell along each of the Axial Normal Directions. For example, if the AND is E-W, and the axis is growing to the right/top, then the bottom and top x intervals of the cell are saved in the list. The left side of each of these intervals is grouped with a set of left points and the right side with a set of right points. A linear programming solution to the 2D linear separability problem [29] is then run on the two sets. This procedure determines if it is possible to fit a line that strictly passes through each cell, and if it is possible, it returns one such line. This line is used to define the Axis Segment for visualization purposes only, and thus no attempt is made to find a line that optimally fits the axial cells.

Figure 19 shows the groups of axial cells from Figure 18. The DMD vectors define the intervals that the Axis Segment must pass through. The top figure of 19a,b,c shows the points representing the interval endpoints. The circles correspond to left points and triangles to the right points passed to the linear separability code. The bottom figure in each case shows a possible resulting line that is returned from the linear separability code.

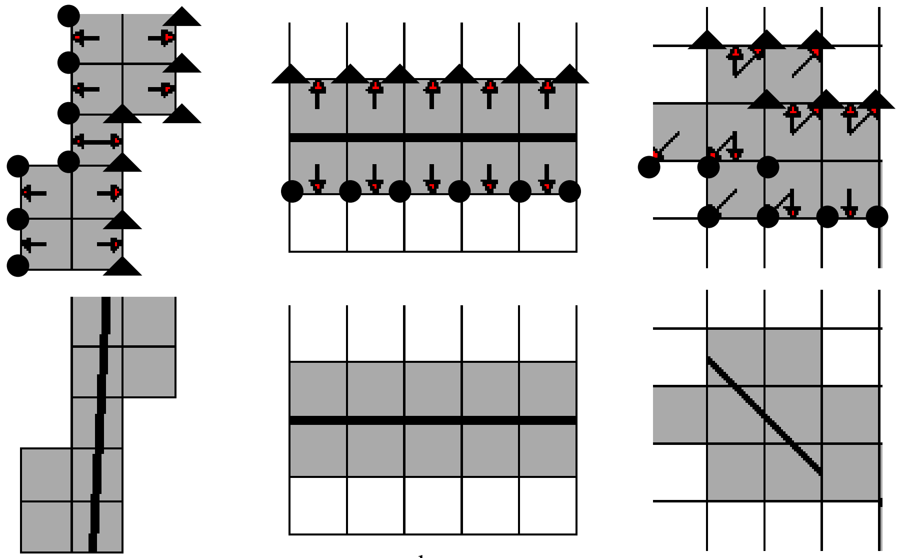


Figure 19: Fitting linear segments to axial cells: a,b,c) The top figure shows the axial cells with the left points labeled by circles and the right points labeled by triangles. The bottom figure shows the resulting axis line segments.

The skeletonization process of di Baja and Thiel [11] also forms linear segments from skeletal points. The endpoints of a skeletal branch are set as the end vertices of a linear segment. This segment is then recursively split if the pixels along the branch fall too far outside of the segment. The split occurs at the pixel with the greatest distance from the line segment. This process has the advantage over our approach of being independent of the order in which pixels are processed. A final segment splitting is performed by di Baja and Thiel where non-monotonic changes occur in the width. Our axes maintain this width information internally, but are not split at these locations. We view these width changes as secondary indications of subregions within a parent region, and thus do not split the axes. The decomposition process will discover these features once the parent region has been isolated.

Our approach is unique in that it does a preliminary high-level segmentation; axial points are not considered for grouping into a segment unless they have the same Axial Normal Directions. This approach has the advantage of utilizing local shape information, and results in Axis Segments whose endpoints correspond directly to concave boundary regions.

Di Baja and Thiel only consider the spatial relationships between skeletal points when forming segments. They split segments at branches and points where non-linear changes occur in the branch. These locations correspond to concave regions in the boundary. This generates segments whose endpoints only roughly correspond to the concave boundary points. Our approach more naturally assigns segments to fundamental regions. Our axes are therefore better suited for the balanced, hierarchical type of decomposition into fundamental parts that we are interested in.

Each axial cell contains a pointer to the axis it belongs to. The Axis Segment stores its Axial Normal Direction set, length, cells corresponding to its initial and final endpoints, and the maximum DM distance value of all its component axial cells. The interior cells lying along the axis between the endpoints are derived if needed by retraversing the axis.

4.2 Creating the Axial Shape Graph

The Axis Segments are connected into a graph structure reflecting the spatial relationships between the fundamental parts of the object.

4.2.1 Constructing the Graph Structure

The graph structure is formed by associating a *node* with each Axis Segment. An *edge* is added between each pair of nodes that represent Axis Segments of adjacent fundamental parts. The Axis Segments may be directly connected, or indirectly connected via ridges in the Distance Map.

The graph structure has the same connectivity as the object: if the object consists of a single, connected component, the ASG is composed of a single tree structure. The Axis Segments lie along ridges in the Distance Map. We connect each axis endpoint to all other axes that are directly reachable by walking out from the axis endpoint along DM ridges.

In order to describe the ridge-walking implementation, we will first establish terminology for characterizing axial cells in the DM. This follows directly from the work of Niblack [25].

- **Strict Maximum**: A cell with DM distance value d strictly greater than any of its eight nearest neighbors.
- **Maximum**: A cell with DM distance value d greater than or equal to any of its eight nearest neighbors.
- **Saddle Point**: A cell with DM distance value d whose nearest neighbors represent alternating *hump*s and *valley*s relative to d. A hump is a series of cells with value greater than d, bounded on either end by cells with distance values less than or equal to d (the valleys), or a series of cells with distance equal to d with adjacent valleys with value strictly less than d. A saddle point is a cell with two or more humps.

  Saddles can be either formed from a single cell (a 1x1 saddle) or a cluster of four cells (a 2x2 saddle):

  - **1x1 Saddle**: A single cell whose eight closest neighbors form two or more humps.
  - **2x2 Saddle**: A 2x2 cluster of cells all having the same distance value, and none of which is a 1x1 saddle. The twelve closest neighbors adjacent to the 2x2 block are defined as the neighborhood in this case and must contain two or more humps.
- **Climbing Neighbor**: For a non-maximum cell, the neighbor with the maximum DM distance value. If there is more than one such equal valued cell in a series, the climbing neighbor is chosen according to the following rules:

  - If the series contains an odd number of cells, the middle cell is chosen.
  - If the series contains an even number of cells, one of the cells that is a direct neighbor is chosen.

  If there is more than one series, one is chosen arbitrarily. Each non-maximum, non-saddle cell is therefore assigned a single climbing neighbor. Saddle cells have a climbing neighbor for the maximum valued cell on each of their humps.

Figure 20 shows examples of cells fitting these definitions. In each case the upper figure shows the distance values and cell type labels, and the bottom figure illustrates the climbing neighbors. The center cell in Figure 20a is a 1x1 saddle point,designated by the `S` label. Its humps and valleys are labeled `H` and `V`, respectively. The maximum cell for each hump is shown highlighted. Figure 20b illustrates a 2x2 saddle cell. Figure 20c shows two 1x1 saddles, two local maximum (labeled as `L`) and a strict local maximum (labeled as `M`).

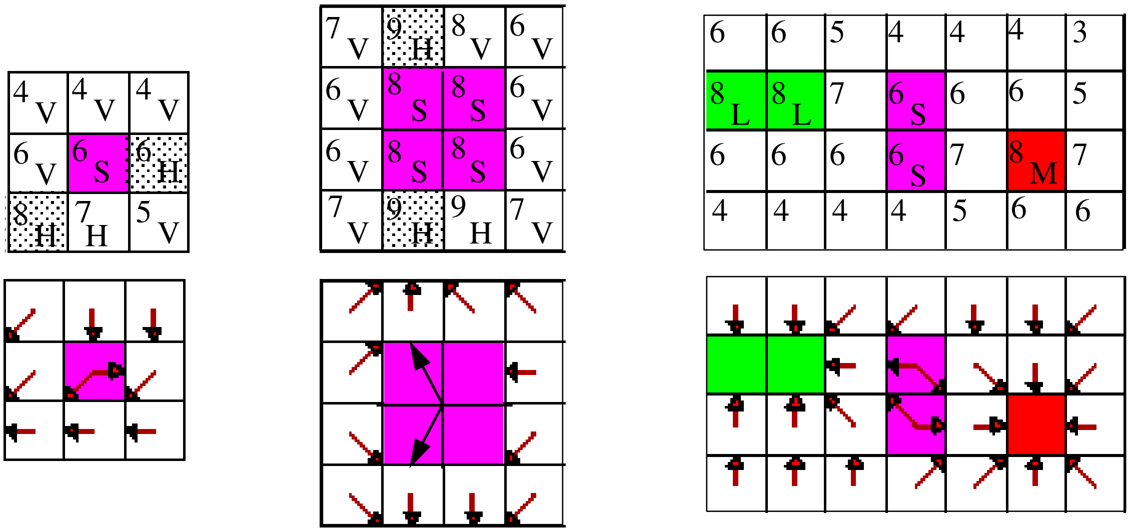


Figure 20: Cell Connectivity Types a) 1x1 saddle b) 2x2 saddle c) 1x1 saddles, local maxima and a strict local maximum

Niblack et al construct a skeleton composed of local maxima in a distance map that are connected along climbing ridges. They prove that the resulting skeletal structure will have the same connectivity as the original object. For any two maxima, with no local maximum in between the two, the path from the minimum of the two will either be strictly increasing after one step to an equal valued neighbor, or it will pass through a saddle point. In their approach, the skeletal structure is connected by climbing uphill from equal valued neighbors of the local maxima, and from all paths out of saddle point humps, until a maximum is reached.

We have utilized this basic approach to implement a ridge walking procedure from axis endpoints. The axis endpoints may or may not be local maxima. If they are, then they can be connected as outlined above. If they are not local maxima, the approach still holds. Axis Segments coincide with Distance Map ridges, and therefore lie along a path between pair(s) of local maxima.

The path between axis endpoints may pass through multiple saddles and local maxima. We have implemented an approach based on the path climbing of Niblack et al to make connections directly from axis endpoints. This is similar to the methods employed by Dorst [13] and Arcelli and di Baja [1].

An initial pass is made that labels all special cells with a connectivity type (strict maximum, maximum, saddle). Each cell is then assigned a climbing neighbor and saddle cells are assigned multiple climbing neighbors, one for each hump. This information could easily be generated on demand for the cells as they are encountered on a climb to make the approach more efficient.

Next, we link each local maximum to the saddle points from which it is directly reachable along an uphill climb. This is done by following the climbing path from each saddle hump until a maximum (strict or not) is reached. The saddle is then added a list stored with the maximum recording which saddle points can reach it directly along a climbing path. The maximum also stores which of the saddle's humps the climb initiated from.

We are only interested in climbing the ridges connecting Axis Segments. The connection algorithm proceeds by starting a walk from each axis endpoint cell. The following algorithm is performed for each cell encountered along the walk:

1. If the cell lies on another axis, make the connection between the two axes and terminate this walk.
2. (a) If the cell is a maximum (nonstrict), then continue the walk from each of its direct neighbors with the same distance value.
   (b) In addition, begin walk(s) from each of the maximum's associated saddle points. This corresponds to walking down the ridge to the saddle and then out of the saddle from the remaining humps.
      i. For each saddle point, we first check if any axes lie on the *downhill* path from the maximum to the saddle point. If there are any axes along this ridge, we wish to find the one closest to the maximum. Because the ridge following climbs *up* ridges, the walk is started from the saddle point hump whose climbing path leads directly to the maximum (we have recorded this value in the initial maximum-saddle linking phase). We keep track of the last axis, if any, seen on this climb. The climb terminates when the local maximum is reached. If an axis is found along this path, the algorithm proceeds to Step 1 and makes a connection.
      ii. If there are no axes along the path from the saddle to the maximum, a walk is started from each of the remaining climbing paths out of the saddle.
3. If the cell is a strict maximum, proceed as in step 2b above (there are no equal valued direct neighbors, so part 2a is unnecessary).
4. If the cell is a saddle point, start walks from all hump climbing paths out of the saddle.
5. If the cell is not one of those specially classified, continue the walk by moving to the cell's unique climbing neighbor.

Connections are added as edges in the graph structure. When a connection is made, each end of the edge is labeled with a Connectivity Type: type L specifies that the edge connects to a cell that is the left/bottom endpoint cell of an Axis Segment, type R to the right/top endpoint cell of the Axis Segment, and type T (for T-connection) to an axial cell internal to the Axis Segment.

Figure 21 illustrates the process for a portion of an object. The Axis Segments labeled 1,2, and 3 are connected by the highlighted climbing paths. The endpoint cell of axis 1 is a local maximum (labeled `L`). The walk starts at step 2) of the algorithm. The cell has no neighbors of the same distance value, so the algorithm skips 2a and proceeds to 2b. A walk is started from the maximum's single saddle point (labeled `S`). The saddle is adjacent to the local maximum, so there are no axes on the downward climb into the saddle. The walk continues out of the saddle via the single remaining hump. The rest of the path continues along the marked climbing neighbors until segment 2 is reached. The climb connecting axis 2 and 3 follows the cells highlighted on the right of the figure. The resulting graph representation is shown below. The letters L and R (for Left and Right) at the ends of the edge designate which end of the axis the edge connects to.

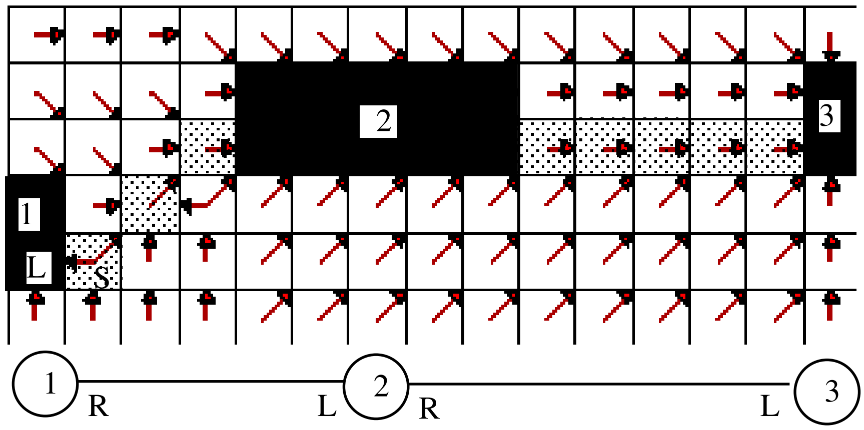


Figure 21: Connecting the Graph: The Axis Segments 1,2, and 3 are connected by following the highlighted climbing paths. The resulting graph representation is shown below: the letters L and R (for Left and Right) at the ends of the edge designate which end of the axis the edge connects to.

4.2.2 Assigning Weights to the Graph Nodes

The Axial Shape Graph is a weighted graph. Each node stores four types of weights: an *Axis Complexity Weight*, a *Left Complexity Weight*, a *Right Complexity Weight*, and a *T Complexity Weight* for each axial cell that is part of a T-connection.

The complexity of an axis can be measured three different ways: Size Complexity (C_s), Count Complexity (C_c), and Spatial Complexity (C_{bb}). These values are calculated for a single axis as follows:

1. **Size Complexity**: This is an relative estimate of the size of the subcomponent measured as one half the area of the approximating bar. If l is the length of the segment and w_{max} is the maximum half-width along the axis:
$$C_s = lw_{max}$$
2. **Count Complexity**:
$$C_c = 1$$
3. **Spatial Complexity**: This is an estimate of the area of the axis-aligned bounding box around the component.
$$C_{bb} = |(x_{max} - x_{min})(y_{max} - y_{min})|$$

The Complexity Weight of an ASG subgraph is the sum of complexities of all axes that are part of the subgraph.

The Left Complexity Weight of a node corresponds to the sum of the complexities of the set of subgraphs attached to the Left end of the node. The Right Complexity Weight is defined similarly, representing the sum of all subgraphs reachable from the edges attached to the Right end of the node. A T Complexity Weight is calculated for each T-junction axial cell by summing the complexities of all subgraphs connected through each T-edge connected to the cell.

The Complexity Weights are calculated by a running a graph traversal. To calculate the Left Complexity, the traversal starts by following all Left edges of out of the cell. The traversal continues through all edges (i.e. Left, Right, *and* T edges) of each node encountered, and the Complexity Weight is recursively summed. Once a node has been visited, it is marked. The traversal continues until there are no more edges to follow that lead to unmarked nodes. A similar traversal is run to calculate the Right and T Complexities, initially following all Right edges out of the node to determine Right Complexity, and all T edges out when calculating T Complexity. The Left, Right and T Complexity Weights are stored with the node and used by the ASG decomposition process.

4.2.3 Linking the Axial Shape Graph across Resolutions

Axial Shape Graph nodes at one level in the hierarchy are linked to nodes of the ASG one level up in the hierarchy if the corresponding Axis Segments represent the same bar component at the next resolution. To determine if an Axis Segment corresponds to an Axis Segment of the graph one level up in the hierarchy, the axial cells defining the two axes are compared. If the axial cells defining the axis at the finer (lower) level pass only through cells whose parents in the cell hierarchy are part of the Axis Segment at the coarser (higher) level, the segments are the same to the resolution of the grid cells.

For efficiency, we compare only the cells that are covered by the line segment representing the axis. This line segment is guaranteed to lie entirely within the axial cells.

Axes at subsequent levels may not exactly coincide, even if they represent the same subcomponent. Axes that differ only by a set percentage of the total number of axial cells in the finer resolution axis are considered to represent the same feature.

If the Axis Segment is matched to the next resolution level, the corresponding nodes are linked and the test continues on to the next resolution level. The *Feature Weight* of an axis corresponds to the number of levels above the axis node in the hierarchy that the match is successful.

5 Hierarchical Decomposition of the Geometry Representation

Our goal is to decompose the object into a well-balanced tree of hierarchical subcomponents that reflect its geometric structure. In a geometric decomposition, each component corresponds to a fundamental part of the object at a particular scale. The object is recursively decomposed into components representing fundamental parts at increasingly smaller scales.

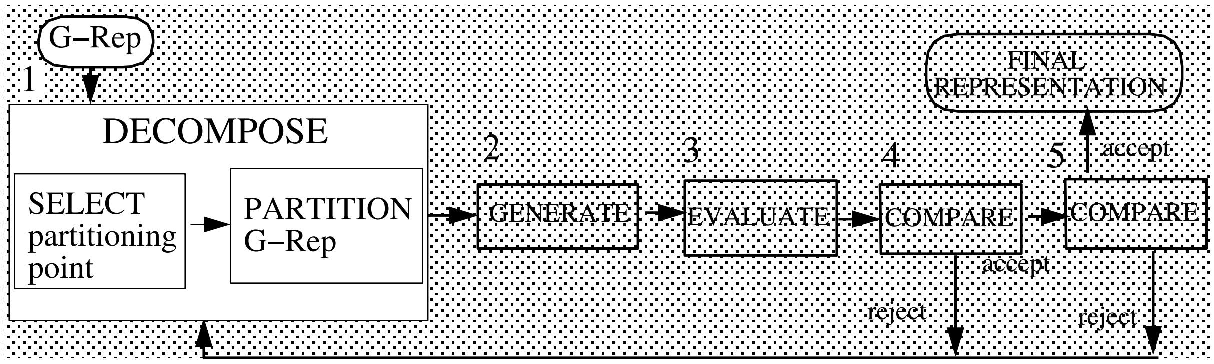


Figure 22: G-Rep Decomposition process and the Representation Generation Pipeline

This section describes a method for decomposing the Geometry Representation into a hierarchy of subcomponents based on the Axial Shape Graph. G-Rep Decomposition is the first phase of the Representation Generation Pipeline. Figure 22 shows its relationship to the overall pipeline introduced in Figure 2. The exact details of the decomposition will depend on the needs of the particular task. In this section we discuss the general process, and in section 6 we address the details of the process when utilized for the task of Collision Detection.

The partitioning is performed at the most detailed level of the G-Rep hierarchy, utilizing the Axial Shape Graph to identify the subcomponents and their connectivity relationships. G-Rep decomposition occurs in two steps:

1. **Selection**: of a Partition Point p.
2. **Partitioning**: of the G-Rep into the two subcomponents defined by p.

5.1 Selecting a Partitioning Point

Using the Axial Shape Graph, the task of shape decomposition is reduced to a graph partitioning problem. The decomposition should reflect the geometric structure of the object (i.e. satisfy Property 1 of Section 2.3), and produce a well-balanced tree of subcomponents (Property 2). To meet Property 1, we choose partitions along boundaries between bar subcomponents, which are suggested by the Axis Segment endpoints and width discontinuities along the Axis Segment. An ideal decomposition would produce a tree that is balanced according to all three criteria: Size, Count, and Spatial. It is not always possible to achieve this balance, however, especially since Property 1 requires that the decomposition be based on the shape structure of the object. The exact weighting used will depend on the application. Section 6 discusses the graph weighting used for the task of Collision Detection. In this section, for the purposes of explaining the general algorithm, we will use Size Complexity as the graph weight.

We first find a *Partition Node* at which to partition the graph structure. Once a node is selected, we choose one of the axial cells of the Axis Segment associated with the node as the *Partition Point.*

5.1.1 Choosing a Graph Partition Node

For now we assume that the graph is partitioned only at axis endpoints. We will remove this restriction later. The graph is decomposed by selecting a node and removing all edges from one side of the node (i.e. the Left edges or Right edges). This partitions the nodes of the graph into two sets. We want a balanced decomposition, and therefore we choose a Partition Node that will break the graph into two sets of nodes with roughly equal complexity weight.

If we partition at a node n by removing all Left edges, one set of the resulting two node sets will contain the Left Graph, and the other set will contain the Right Graph of n, as well as node n. We define *Left* and *Right Partition Ratio*s as the relative complexity of the resulting subgraphs if we partitioned at a node n by removing the Left or Right edges, respectively.

Given that C_l is the Left Complexity of a node n, C_r is the Right Complexity, C_t is the T Complexity, and C_a is the Axis Complexity, we define the following set of ratios:

- **Left Partition Ratio** R_l:
$$R_l = \frac{C_l}{C_r + C_t + C_a}$$
- **Right Partition Ratio** R_r:
$$R_r = \frac{C_r}{C_l + C_t + C_a}$$
- **Axis Ratio** R_a:
$$R_a = max(R_l, R_r)$$

We are interested only in the relative balance of complexities when calculating R_l and R_r and therefore constrain all ratio values to lie in $(0, 1]$: if the ratio exceeds 1, the reciprocal value is utilized instead. If the value of C_r or C_l is zero, the Ratio is set to zero.

Figure 23 is a representation of a single node and its associated axis. The Left (L) and Right (R) edges of the node are shown. The circles represent the subgraphs reachable from each edge. In this example,

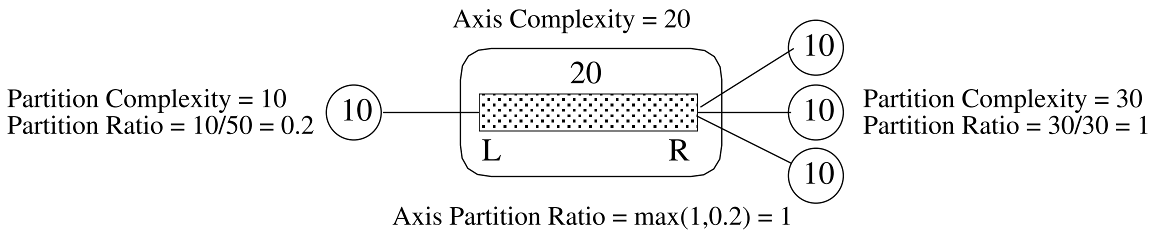


Figure 23: Calculation of Axis Partition Ratio

the Left Complexity of the node is 10 and the Right Complexity is 30. The values of the Left and Right Partition Ratios and the Axis Partition Ratios for this example are shown on the figure.

We choose nodes with high Axis Partition Ratios as potential Partition Nodes. If the ratio is 1, the removal of the edges from one side of the node will divide the graph into two sets of subgraphs of equal complexity. The process of finding the node with maximum Axis Partition Ratio for each decomposition step is very efficient. The Partition Ratios are calculated in the same graph traversals that calculate the Left and Right Complexity Weights. Once the weights have been assigned to a node in the traversal, the node is inserted into a *Node List* sorted in descending order on the Axis Partition Ratios. The head of the Node List is removed each step and utilized as the Partition Node.

5.1.2 Choosing a Axis Segment Partition Point

Once a graph node is chosen for partitioning, we select one of the axial cells defining the associated Axis Segment as the Partition Point. The Partition Point is used to define the actual cut that partitions the G-Rep into two subcomponents. If partitioning at the axis endpoints would lead to a strongly unbalanced partition, we allow points interior to the axis to be selected as Partition Points. In this case, partitioning at the node corresponds to splitting the Axis Segment at the Partition Point and generating two new sub-nodes.

We define the *balance potential* of an axial point as the estimate of the ratio of the sizes of the resulting two object subcomponents that would be produced if the graph were split at the axial point. The ratio is chosen such that the balance potential is always in $(0, 1]$. For axial endpoints, we use the Partition Ratios as an estimate of the balance ratio. For axial points interior to the axis, the axis is split into two sub-segments at the axial point, and Partition Ratios are calculated for the two new nodes produced. The Axis Size is recalculated for each new Axis Segment and the Left Complexity, Right Complexity, and T Complexities are derived from the values of the parent segment. Partition Ratios are calculated from the complexities and used as an approximation of the balance potential of the axial point.

When selecting a Partition Point, we favor axis endpoints because these points are associated with concave points on the object's boundary. If the Axis Complexity and T Complexity Weights are large relative to the Left and Right Complexities, however, partitioning at either end of the node will result in an unbalanced tree. We also wish to satisfy the balance properties, and therefore in these cases we consider points on the axis between the endpoints. If the axis endpoints do not have an acceptable balance potential, we next consider any points with adjacent T-edges as potential Partition Points. If there are no such points, or they do not have an acceptable balance potential, we next examine any points defining width discontinuities. These points also correspond to concave points on the object boundaries and thus suggest good partitions. Finally, if we have still not found an acceptable Partition Point, we choose the axial point that best balances Left and Right Complexities. This point is called the axial *balance point*.

This results in a process that prefers partitions at axis endpoints, assuming these will coincide with subcomponent boundaries, but will favor a more balanced cut if the balance potential is below the specified balance threshold.

Partition Points are therefore chosen in the following order:

1. Axis Segment endpoints
2. T-connection points
3. Width Discontinuities
4. Balance Point

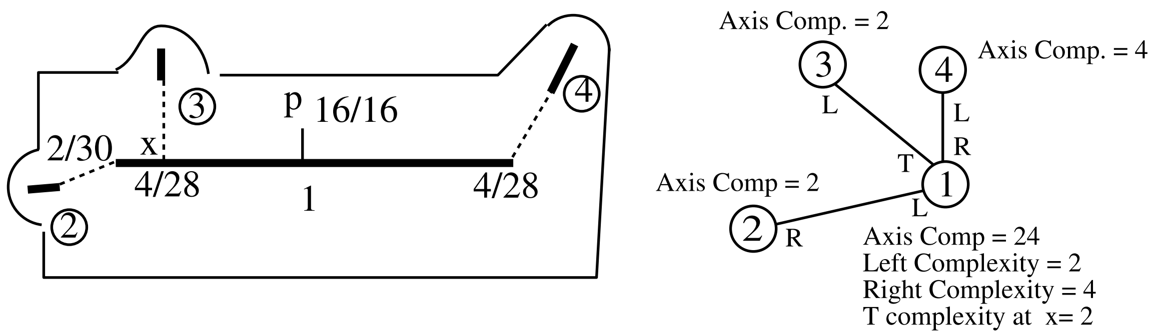


Figure 24: Selection of Partition Point

Figure 24 shows an object and its associated ASG representation. For this example, we assume that the potential balance threshold is 0.25. Axis 1 has the highest Axial Partition Ratio (0.14) and is therefore chosen as the Partition Node. The Ratio Partitions of the axial endpoints are below threshold ($2/30 = .07$ and $4/28 = .014$), so the T-connection point (labeled "x") is considered next. This point is also rejected because its balance potential is $4/28 = 0.14$. There are no width discontinuities in this case, and therefore the balance point (indicated with the label p) is chosen as the Partition Point.

5.2 Partitioning the Geometry Representation

Once an interesting Partition Point has been selected, the actual partition cut to decompose the G-Rep is determined. We want cuts to occur along boundaries between bar subcomponents, and thus at the apex of defining concavities.

The Partition Point is used to find a pair of boundary or *shore* points that define the actual cut. Every axis point is associated with at least two boundary cells on "opposite" shores. These cells are reached by following the Distance Map Direction vectors. The lower left corner of the cell (i.e. the address of the cell) is used as the actual shore point. These points constitute a natural termination of the bar associated with the Axis Segment. A cut made between such a pair of boundary points will create a partitioning that separates the bar component from the rest of the object.

There may be multiple shortest paths to the boundary, but most often they end up at the same shore points, and therefore produce a manageable number of pairs of shore points through which a cut can be made.

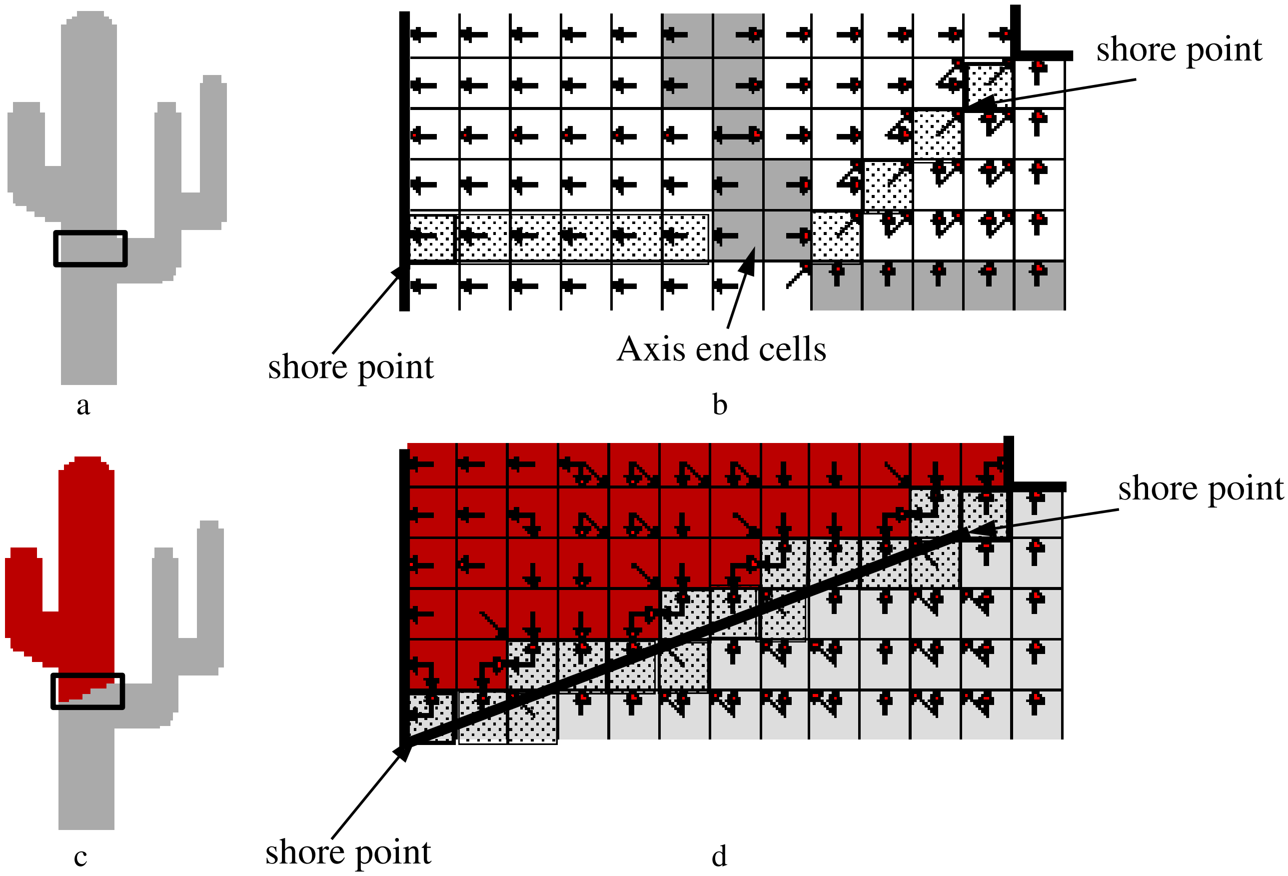


Figure 25: G-Rep Decomposition: a) Original b) Paths to shore from axis end cell c) Resulting Decomposition d) Resulting expanded cut area

Figure 25 illustrates a single G-Rep decomposition step. The box in 25a indicates the expanded region shown in Figures 25b and 25d. In Figure 25b, the composite axial end cell chosen for the Partition Point is indicated. The cells on the paths leading to the two shore cells from the axial endpoints are shown highlighted.

Once the two cut points on the boundary are selected, the object is partitioned along the line segment between the two points. Cells along the line segment belong to the part that lies to the right/top of the line. This is done for simplicity, so that after a cut, each cell belongs to exactly one of the two resulting pieces. The cutting routine walks along the line between the cut points, labeling each cell that the line intersects as belonging to the part on the right/top of the line. A flood fill is then run on either side of the cut to label all of the cells belonging to the new left and right subcomponents. Figure 25d shows the resulting cut chosen by the partitioning code. The cut line segment is shown and the cells that the line intersects are shown highlighted. Figure 25c shows the resulting two parts after the partitioning.

After the two subcomponents have been isolated, the Axial Shape Graph is reconstructed for each subcomponent.

5.3 Utilizing the Hierarchy

In addition to connections to adjacent axes, an axis may also be connected across the hierarchy to an axis representing the same feature at the next resolution. This multiresolution information is captured locally at each level by a Feature Weight indicating at how many levels of resolution the feature appears.

The Feature Weight can be utilized to order the axes on *importance*. Features that occur at multiple levels in the hierarchy are more likely to represent true strong geometric structure, and not discretization noise. Components associated with high importance are given preference when performing the decomposition. The partitioning is done in decreasing order of Feature Weight, i.e. strong components are separated first, and then later decomposed into their subcomponents.

5.4 Managing the Part Hierarchy

A hierarchical part structure is maintained to keep track of the components as they are created. This part structure is a binary tree where a node represents a part. Each part has a number and an associated G-Rep structure. Internal nodes in the part tree also have a cut. The cut is represented by two 2D points which correspond to the two cells defining the ends of the cut. The cut will break the parent part into two parts which are the children of the node (it is possible for a cut to produce a single child in objects of genus greater than zero, but here we are assuming genus zero objects because of the flood fill performed on the object).

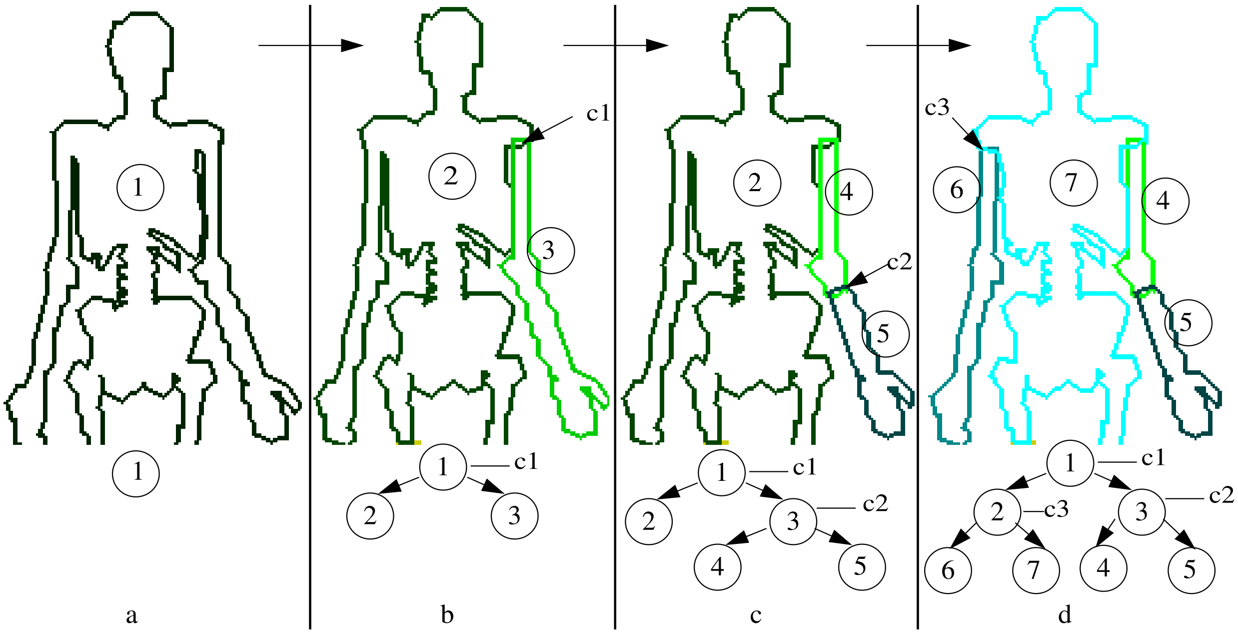


Figure 26: Part Tree Structure: a-d show the progressive decomposition of an object into parts. The upper figure shows the cut and resulting object parts and the lower figure shows the tree representation.

Each part in the tree has an unique number that stores the cut history. The part number is a collection of bits, where each pair of bits corresponds to a cut. If a part is on the left/bottom side of a cut i, then the value stored in the two bits for cut i (i.e. bits $(2i-2)$ and $(2i-1)$) will be 1. If the part lies on the right/top side of the cut, it will have the value 2. The cut number is assigned in the order that the cuts are made.

The part tree lives in conjunction with the quadtree data structure. Each cell references the part that it belongs to. The part numbers are also assimilated up the quadtree such that all parts exist in the same quadtree, but individual parts can be managed efficiently because each internal node keeps a record of which parts have cells beneath that node.

The part tree records all information relevant to the partitioning process. At any point, the cut history can be retraced, and if desired, undone.

Figure 26 shows a sequence of G-Rep partitions, and a representation of the corresponding part tree. Figure 26a shows the original object with a single part, Part 1. In Figure 26b the object has been partitioned at the shoulder by Cut c1, resulting in Parts 2 and 3. Part 3 is further partitioned at the elbow by Cut c3 as shown in Figure 26c. The next cut is shown in Figure 26d, where Part 2 is partitioned at the shoulder into Parts 6 and 7.

5.5 Comparison to Previous Work

An object's shape can be most naturally described as a hierarchy of subcomponents. Axial-based representations have been utilized to perform shape decomposition and description. In this section we compare relevant work in 2D shape segmentation to our decomposition approach.

Di Baja and Thiel [11] present a skeleton decomposition method that can be utilized to represent an object as a collection of elementary regions. We have discussed their skeleton construction approach in Section 2.2.1. This approach produces a representation composed of the union of a set of overlapping regions, and is thus not a strict decomposition per se.

Rom and Medioni [30] utilize an axial representation to hierarchically decompose an object based on its shape. This approach is a combination of existing approaches. The input object is represented by a B-spline approximation. Smooth Local Symmetries (SLS) are used for the axial description of parts, but because SLS produces superfluous axes, the contour is first segmented into regions bounded by consecutive negative curvature sections. The SLS axes are computed for all such parts. These define *local ribbons*, which are potential parts of the object decomposition. Parallel symmetries are used to represent global shape, characterizing *global ribbons*. The relationship between local and global ribbons is maintained; a local ribbon may continue a global one, or local and global ribbons may conflict. Once a potential part is identified it is removed by deleting all but the first and last two boundary control points of the B-spline control polygon of the region to be segmented and generating the closing curves on the two subcomponents. The decomposition is a recursive process that proceeds by first removing small parts, and continuing with the remainder of the object. Local ribbons that conflict or continue a global ribbon are not removed until all other local ribbons are segmented. In this way, the relationship between the global and local information is used to create a hierarchical decomposition.

Our approach is structurally similar to the approach of Rom and Medioni, and our axis formation is similar to that of di Baja and Thiel, but our approach is novel in many ways. Our decomposition process had been designed for a different purpose, and therefore we discuss improvements over the previous work in the context of the tasks that we are interested in. The structure of the Axial Shape Graph provides many advantages over previous skeletons for the purpose of shape decomposition. First, axial points are selectively joined into Axis Segments based on the local shape characteristics of the object, not just by proximity. Our approach is also novel in the use of the weighted Axial Shape Graph to produce balanced part trees. Our partitioning process applied to the Cell-Based Spatial Representation produces a strict decomposition. Finally, the overall Representation Generation Framework allows the decomposition process to be driven by user-specified metrics, and provides a means of evaluating the resulting decompositions.

6 Generating Representations for Collision Detection

We have implemented an application based on the Geometry Representation that automatically generates representations specialized for the task of Collision Detection in 2D environments. In this section we first discuss the task of Collision Detection and its representational requirements. We then describe an implementation of the Representation Generation Framework presented in Section 1 (Figure 28), and present results of using this application to generate representations for a variety of objects. In the final section, we compare the Representation Generation Framework, and the representations it produces for the task of Collision Detection, to existing approaches and representations.

6.1 Representations for Collision Detection

A robust and efficient collision detection mechanism is an integral part of any animated and interactive virtual environment. As dynamic objects move through the virtual world, intersection tests must be performed quickly between the moving object and all other objects near it.

Many collision detection systems utilize a hierarchy of representations [21, 8, 24, 28]. In these approaches, intersection testing occurs in two main phases: a *broad* phase followed by a *narrow* phase [21]. In the *broad* phase, intersection tests are made against simplified approximations of the object. If a possible intersection is detected from these approximate tests, the actual object is tested in the final *narrow* phase. The intersection test is first performed against the coarsest representation at the root of the hierarchy. If such a test reports that there is no intersection, all objects below that representation in the hierarchy can be removed from further consideration. If, on the other hand, a possible intersection is detected, the test moves to the next, more detailed and accurate representation found at the next level of the hierarchy. In this paradigm, finding a true intersection will require multiple levels of testing, but the extra computation required is usually more than balanced by the performance gain provided by the fast intersection testing and culling based on the approximations in the case of a miss.

Axis-aligned bounding boxes are often used as the components of these representation hierarchies. An object's bounding box is easy to construct and to perform intersection tests against. Axis-aligned bounding boxes, however, are often very poor approximations of an object. This means that many false hits will be detected at the coarse level, and several additional intersection tests will need to be performed before the collision detection mechanism is able to determine whether or not the object has been hit. In addition, axis-aligned bounding boxes are not rotation invariant. If the object is not a circle, and is allowed to rotate, the bounding box representation will either need to be set as the worst case bounds for any orientation, or be recalculated.

The convex hull of the object is also often used as one of the approximations in the hierarchy of representations. The convex hull is rotation invariant, and more closely approximates a larger class of objects than the axis-aligned bounding box. We are assuming that the collision detection algorithm used is some variant of the Lin-Canny approach [24], such that intersection testing is performed very efficiently between convex objects. In addition, we assume that the system can handle non-convex objects as well, although at a significantly higher cost (such as I-Collide [8]). If the objects are convex, then the intersection test is extremely fast. If the objects are non-convex, but well-approximated by their convex hull, then the test is still efficient.

Different types of representations may be combined in the hierarchy of representations for collision detection. The representation at the most detailed level may or may not be the original representation. It may be possible to replace the original with a representation that is simpler, but equivalent for the collision detection task. For example, if the task is to insert a part into an engine, and the minimum feature size of the part is known, the *probe* size, then concave features of the engine smaller than this size do not affect the process. The original object can therefore be replaced with one that has all such features removed. An object may also contain a complicated network of internal cavities. This information is irrelevant in determining if the outer hull of the object (assuming that the simulation does not utilize other physical information about the object) contacts another solid object, and can therefore be filtered out.

To satisfy the requirements of the collision detection task, a representation must meet the following **Representation Criteria** :

1. **Correctness**: The representation must produce conservatively correct results when tested with a probe of a specified size. False positives are acceptable and only represent a performance loss, but the test must not miss when there is an actual intersection.

2. **Efficient Intersection Testing**: The representation should allow inexpensive intersection testing.

3. **Efficiency of Representation**:

   - **Detail**: The representation should be minimal in terms of detail, i.e. it should only contain those features relevant to the task.
   - **Redundancy**: There should be minimal overlap between the representations for subcomponents of the object.
   - **Size**: The representation should not cover a large (relative to the application domain) spatial extent. If there is a pipe running through a building, for example, the entire pipe will have to be considered for potential intersection whenever the object of interest is anywhere within the building. It would be more efficient to partition the pipe into smaller components such that if the object of interest is in a certain room, it only needs to be tested against the small portion of pipe that passes through the room.

For our study, we utilize the following representations:

1. Conservative Hull (CH)

2. Axis-aligned Bounding Box (BB)

Representation 1, the Conservative Hull, is a convex representation that fully encloses the object. The tightest fitting Conservative Hull corresponds to the object's convex hull. The Conservative Hull could potentially consist of multiple levels of detail if the object is very complex (e.g. a finely tesselated representation of a circle). The most detailed level Conservative Hull could be either the convex hull of the original object or a conservative convex approximation if the detail of the original object exceeds the resolution of the collision detection task. For the intermediate level Conservative Hulls, we construct the convex hull of a simplified version of the object. An alternative conservative convex approximation could also be utilized, such as Oriented Bounding Boxes (OBBs) [18].

The top row of Figure 27 shows an input object and the corresponding Conservative Hull and axis-aligned Bounding Box, marked as level 0 in the figure.

If the original object is not convex, the Conservative Hull may be a very poor approximation. To produce more efficient representations for the task, we decompose the object and generate convex representations for the resulting subcomponents such that the object is better approximated by the composite Conservative Hulls.

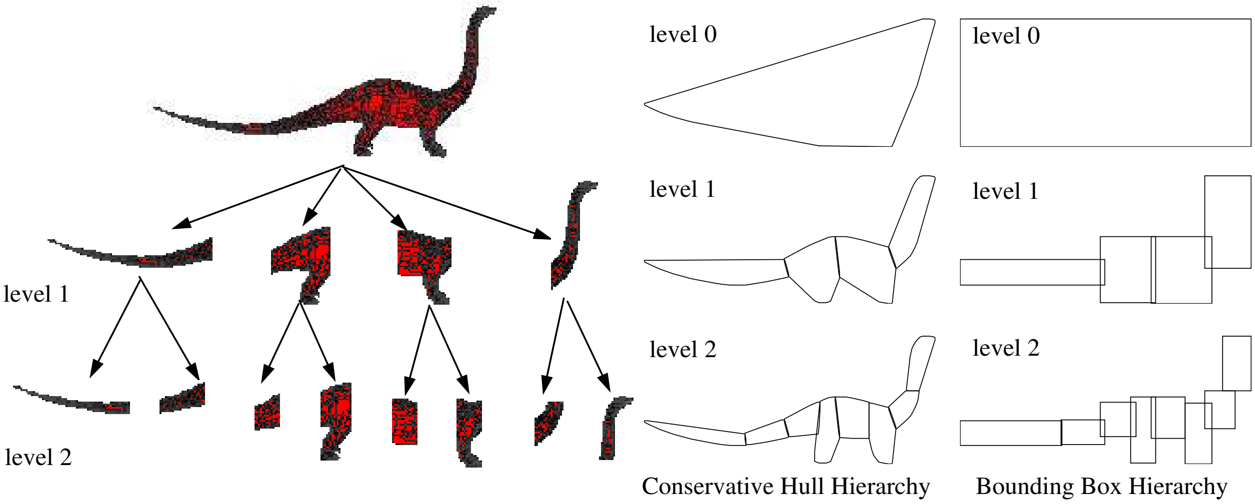


Figure 27: Hierarchical Representation for Collision Detection

The object in Figure 27 is an example of an object that is not approximated very well by its Conservative Hull and Bounding Box. Figure 27 shows a possible representation of the same object after it has been recursively decomposed into components better approximated by the Conservative Hull and Bounding Box representations. The middle figure shows three levels of convex representations resulting from the decomposition. The rightmost figure shows the three levels of Bounding Boxes resulting from the decomposition.

In the following section we show how the Representation Generation Framework can be used to produce a hierarchy of Conservative Hull and Bounding Box representations that meet the Representation Criteria of correctness (criterion 1) and efficiency (criteria 2-3) for the task of Collision Detection.

6.2 Overview of the Representation Generation Framework

We have implemented an application based on G-Rep decomposition that produces a hierarchy of approximate representations to be used in the *broad* phase of a Collision Detection approach. For this task, the desired output representation is a balanced hierarchy of convex approximations that conservatively and efficiently bound the input object. In addition, the axis-aligned bounding boxes of the approximations have aspect ratios close to unity and should not vary too much under object rotation. To achieve these goals, the process of representation generation is guided by a quality/cost metric that favors simplified subcomponents that both minimize the area difference between the object and its composite Conservative Hulls and produce Bounding Boxes of unit aspect ratio.

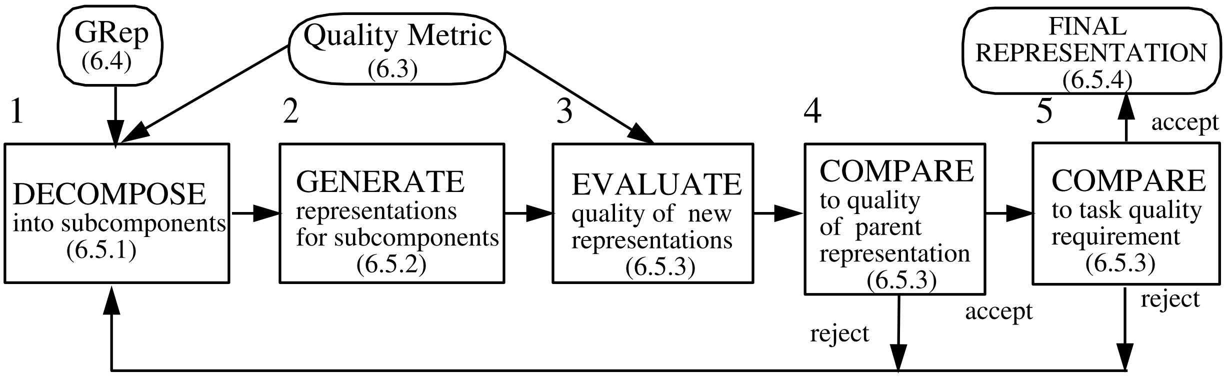


Figure 28: Representation Generation Framework for Collision Detection

The application we have developed is structured as a feedback loop that recursively generates representations for subcomponents of the object based upon its geometric structure. The resulting representations are evaluated at each step, and this information is used to guide the process.

The algorithm is based on the following *Representation Generation Framework*:

1. **Decompose** an object component into two subcomponents.
2. **Generate** a representation for each subcomponent.
3. **Evaluate** the new representations.
4. **Compare** the value of the new representations to the representation value of the parent.
   - If the improvement exceeds a specified threshold, the decomposition and resulting representations are accepted.
   - If the improvement is below threshold, the next iteration is started.
5. **Compare** each new representation value against the specified quality threshold.
   - If the threshold is exceeded, the representation is accepted as the final representation and the subcomponent is removed from further processing.
   - If the quality is below threshold, the subcomponent is sent back for further decomposition.

Figure 28 shows the basic Representation Generation Framework schematically. The following sections discuss the implementation of the various Framework modules. The corresponding section number is indicated for each module in the figure. In Section 6.6 we present the results of utilizing this framework for representation generation for a variety of input objects. In Section 6.7 we compare both the representations generated and the generation process to other approaches.

6.3 Quality Metrics

In order to generate representations for Collision Detection that fit the specified requirements of the task, we must first develop a measure of the quality of a representation. Each of the modules in the Representation Generation Framework is dependent upon task-specific metrics: we must evaluate representations to decide if an object component should be decomposed further (Module 1,5), to determine how it should be decomposed (Module 1), and to evaluate if a potential decomposition produces a better representation (Module 4). This evaluation is based on a quality/cost metric.

In the context of the Collision Detection task, quality is measured as area difference between the original and the representations. In addition, representations should also have approximately unit aspect ratio, i.e. be close to circular in shape.

We also characterize the cost of a representation hierarchy. We want to decompose the object into components that are well-approximated by Conservative Hulls and axis-aligned Bounding Boxes. Each time a subcomponent is decomposed, however, an additional level is added to the representation hierarchy. There is a cost associated with the number of levels that must be traversed on a hierarchical intersection test. There is therefore a tradeoff between the number of levels in the hierarchy, and thus the number of decomposition steps performed, and the tightness of the representation. The decomposition process is guided by this cost consideration. A partition is rejected if the resulting representations do not improve the quality function by a given amount. Partitioning is also terminated if the size of the resulting subcomponents falls below a given threshold. In this way, the cost of adding an additional level in the representation is weighed against the quality gain. While it is clearly wasteful to store a convex hull and a bounding box for a square object, other cases are more subtle and require further inspection. The quality/cost metric together weighs the accuracy of the representations against the amount of information that needs to be stored and processed. For example, if a decomposition improved the value of the representation by 5%, but increased the representation cost by 80%, it might be rejected.

The following values are calculated for each Conservative Hull and Bounding Box:

- **Representation Quality $\mathcal{Q}_{CH}, \mathcal{Q}_{BB}, \mathcal{Q}_{AR}$**: The Representation Qualities $\mathcal{Q}_{CH}$ and $\mathcal{Q}_{BB}$ measure the relative area difference between the Conservative Hull ($\mathcal{Q}_{CH}$) and Bounding Box ($\mathcal{Q}_{BB}$) representations and the original object. If P is the object subcomponent:

$$\mathcal{Q}_{CH} = \frac{Area(P)}{Area(CH)} <= 1.0$$

$$\mathcal{Q}_{BB} = \frac{Area(P)}{Area(BB)} <= 1.0$$

  The representations are conservative, and therefore the quality values are in (0,1], with an exact representation yielding an optimal Representation Quality value of 1.

  The Representation Quality $\mathcal{Q}_{AR}$ is a measure of the aspect ratio of the Bounding Box representation. If $Length(BB)$ is defined to be the maximum extent of the Bounding Box, $\mathcal{Q}_{AR}$ is calculated as follows:

$$\mathcal{Q}_{AR} = \frac{Width(BB)}{Length(BB)} <= 1.0$$

  Quality $\mathcal{Q}_{AR}$ achieves a maximum value of 1 for a square, or Bounding Box with unit aspect ratio.

- **Decomposition Benefit**: $\mathcal{B}_{CH}, \mathcal{B}_{BB}, \mathcal{B}_{AR}$. The Decomposition Benefit expresses the relative improvement in quality resulting from decomposing an object into subcomponents. If CH_R, BB_R are the representations for the parent component and CH_{R1},BB_{R1} and CH_{R2}, BB_{R2} are the representations of the two child subcomponents, the Decomposition Benefits are calculated as follows:

$$\mathcal{B}_{CH} = \frac{Area(CH_R)}{Area(CH_{R1}) + Area(CH_{R2})}$$

$$\mathcal{B}_{BB} = \frac{Area(BB_R)}{Area(BB_{R1}) + Area(BB_{R2})}$$

$$\mathcal{B}_{AR} = \frac{min(\mathcal{Q}_{AR}(CH_{R1}), \mathcal{Q}_{AR}(CH_{R2}))}{\mathcal{Q}_{AR}(R)}$$

Decomposition Benefits $\mathcal{B}_{BB}$ and $\mathcal{B}_{CH}$ express the improvement as the relative decrease in area from the parent representation to the pair of child representations. The aspect ratio Decomposition Benefit $\mathcal{B}_{AR}$ chooses the child with the worst aspect ratio (i.e. smallest), and compares the relative improvement with the parent representation. The Decomposition Benefits may be < 1, indicating that the quality of the composite of the child representations is worse than the quality of the parent.

If the Decomposition Benefit is greater than one, the children represent the object more accurately than the parent. There are, however, practical costs associated with the number and complexity of representations in the hierarchy. The current system adopts a simple cost model. By only accepting partitions with Decomposition Benefits above a fixed percentage, we take into account the cost incurred by adding levels to the hierarchy. The cost could also be adapted to include the number of edges in the Conservative Hull, and the absolute number of sub-components making up the hierarchy.

The Hausdorff distance [29] is an additional measure of the accuracy of a representation. It is defined to be the maximum of the distances from each point on the object boundary to the closest point on the representation boundary. We calculate the perpendicular distance from each cell representing a concave point on the boundary of the object to the Conservative Hull edge covering that concavity.

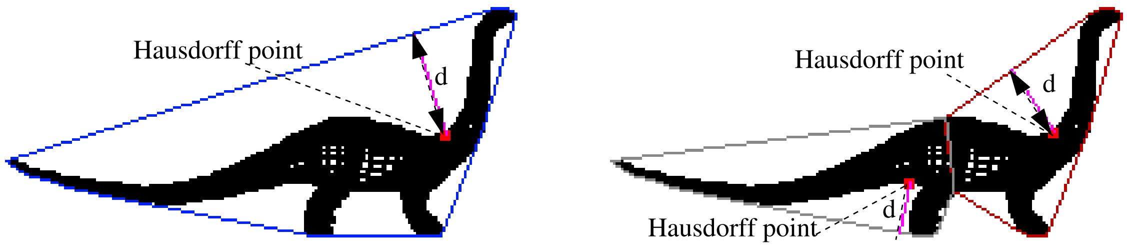


Figure 29: Hausdorff points for dinosaur parts

We utilize the Hausdorff measure only to evaluate the quality of the generated representations, but it could easily be incorporated as a metric to guide the decomposition process. The cell or *Hausdorff point* associated with the maximum distance represents a concave point. These points could be chosen as initial starting points for the decomposition process. Figure 29 illustrates the *Hausdorff points* for various parts of the dinosaur example. The resulting maximum distance is shown as a line from the point to the Conservative Hull.

The following sections describe the utilization of the Quality and Benefit metrics in the generation of representations.

6.4 Construction of Geometry Representation

The application first constructs the multi-resolution Geometry Representation hierarchy from the input object. For Collision Detection, we are primarily concerned with what space an object occupies, and therefore the G-Rep spatial representation is well-suited for this task. The grid-insertion process also filters out details not relevant to the Collision Detection task. If the probe size is known, details in areas where the probe cannot reach due to its size can be filtered out by choosing a cell size based on the probe size. In addition, the spaces interior to the object are filled. If we are probing an object to test for collision, the interior spaces of an object are irrelevant, as these areas are not reachable by a colliding probe of any size.

Our goal is to partition objects into components that are well approximated by a Conservative Hull and by a Bounding Box. A circle fits this ideal: it is convex, and has the same Bounding Box at any orientation. We can therefore cast the problem as one of decomposing the object into circular components. In practice, however, objects do not naturally decompose into a small number of minimally overlapping circles. We instead seek to find components that are as circle-like as possible, meaning that they are relatively convex and have an aspect ratio close to unity.

The Axial Shape Graph described earlier can reveal these approximate convex, or generalized bar features in an object. A circle would have a single axial point with complete symmetry (direction vectors in every direction). The bar subcomponents should have a single strong axis that is relatively short compared to its distance from the boundary to meet the bounding box criteria.

6.5 Generation of Hierarchical Representations

The final output of the program is a collection of hierarchical Bounding Boxes and Conservative Hulls that closely bound the original object. The decomposition of the object into subcomponents is guided by the evaluation of the metrics presented in Section 6.3. There are three Representation Quality metrics associated with the process, listed in order of importance:

1. Conservative Hull Representation Quality ($\mathcal{Q}_{CH}$)
2. Bounding Box Representation Quality ($\mathcal{Q}_{BB}$)
3. Aspect Ratio Representation Quality ($\mathcal{Q}_{AR}$)

The user supplies *Quality Thresholds* ($\delta\mathcal{Q}$) for each type of Representation Quality ($\delta\mathcal{Q}_{CH}, \delta\mathcal{Q}_{BB}, \delta\mathcal{Q}_{AR}$). For example, if the Quality Threshold for the Conservative Hull is $\delta\mathcal{Q}_{CH} = 0.1$, the area of the Conservative Hull representation must not differ from the area of the object by more than 10%.

The user also specifies *Benefit Thresholds* for each type of Decomposition Benefit ($\delta\mathcal{B}_{CH}, \delta\mathcal{B}_{BB}, \delta\mathcal{B}_{AR}$), and a *Minimum Size* ($\mathcal{S}_{MIN}$) and *Maximum Size* ($\mathcal{S}_{MAX}$) value indicating the smallest and largest allowable subcomponent, measured in cells.

The application makes four passes over the subcomponents, using a different evaluation metric for each pass. The order emphasizes the relative importance of the quality metrics. The decomposition proceeds in Pass 1 until all components meet the Conservative Hull Representation Quality Threshold specified by the user, or cannot be further processed by the algorithm. Pass 2 is then run on the resulting subcomponents to verify that the Bounding Box Representation Quality threshold is also met. Any subcomponents that do not meet this criterion will be further decomposed. In the third pass, the Aspect Ratio Representation Quality is utilized as the metric. A fourth pass is made to make sure that the final components do not exceed the Maximum Size limit.

It would be possible to make a single pass over the subcomponents by utilizing a single, unified metric taking into account all of the Quality and Size considerations. The criteria often present conflicting desires in the choice of decomposition and it is not clear how to appropriately weight such a metric, in order to satisfy the criteria for a good representation. We therefore chose to separate the decomposition into passes, and only consider a single criterion in each pass.

Each pass is a three step process:

1. Decomposition
2. Representation Generation
3. Evaluation

The following sections discuss these steps.

6.5.1 Decomposition

At each iteration of the process, the object subcomponent with the worst Representation Quality is chosen for possible partitioning. Subcomponents are maintained in a list based on the Representation Quality value ($\mathcal{Q}_{CH}$, $\mathcal{Q}_{BB}$ or $\mathcal{Q}_{AR}$) or size value being used in the current pass. The list is sorted with the components with the worst representation value occurring first.

The partitioning is performed utilizing the G-Rep Decomposition process presented in Section 4. We have chosen Size Complexity alone as the weighting criterion for our graph. By choosing Size as the single measure, the graph partitioning process is simplified, while still producing acceptable balance for all three criteria. To see why this is possible, we will consider each of the two remaining balance criteria individually. Because we have a minimum and maximum size threshold (i.e. $\mathcal{S}_{MIN}$ and $\mathcal{S}_{MAX}$), a part tree that is Size-balanced will tend to have approximately the same number of components in each subtree. For the Collision Detection task, Spatial balance is very important, potentially even more so than Size balance. Since partitions are chosen based on shape as the primary consideration and since the graph structure itself reflects the spatial connectivity of the object, the only types of partitions permitted are those that have some amount of spatial coherence via their shape and the graph connectivity. The result is that partitions chosen on Size balance tend to have good Spatial balance as well. Figure 30 illustrates the effect of Size balancing with the object from Figure 9. The original figure is shown in Figure 30a and its Axial Shape Graph in 30b. The initial decomposition of the ASG based on shape and Size balance is illustrated in 30c. The final hierarchy is shown in 30d,e. The resulting decomposition has good Size, Spatial, and Count balance.

The Axis Segment's end points are first considered when choosing potential Partition Points, and then its balance point. The application does not currently consider T-connections or width discontinuities when choosing Partition Points. A single Partition Point chosen by the G-Rep Decomposition process may generate more than one possible partition. In this case, all partitions are made and the one resulting in components with the greatest Representation Quality is chosen.

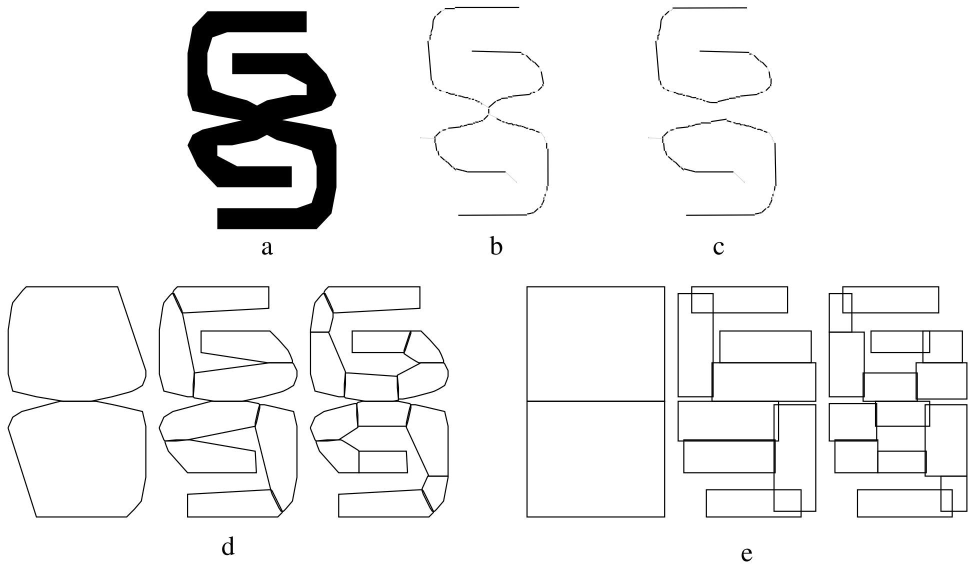


Figure 30: Size balanced Decomposition a) Input Object b) Axial Shape Graph c) ASG after single decomposition step d) Resulting Conservative Hull Hierarchy e) Resulting Bounding Box Hierarchy

6.5.2 Representation Generation

Once a Partition Node has been chosen and the proposed decomposition made, representations are generated for the new components. The Conservative Hull and Bounding Box are constructed around the cells of the subcomponents identified by the partitioning process and are therefore guaranteed to be conservative since the spatial representation is conservative.

The Conservative Hull is constructed by collecting all of the convex cell vertices on the boundary of the cell representation of the object. A convex boundary cell is one that meets the following criteria:

1. The cell lies on the object boundary, i.e. one or more of its eight nearest neighbors does not belong to the object component.
2. The cell has one or more convex vertices. The convex vertices are easy to determine in the G-Rep structure: if a boundary cell has one or more pairs of direction vectors set at 90 degrees, the cell corner that lies between the two directions is a convex vertex.

A simple package-wrapping convex hull construction approach is then run with this point set as input, producing a polygonal boundary representation of the Conservative Hull.

The axis-aligned Bounding Box is calculated by taking the extents of the Conservative Hull in the x and y directions.

The area of the Conservative Hull is calculated as the sum of the areas of the trapezoids defined by each edge and the projection of the edge onto the x-axis.

$$Area = 0.5 * \sum_{i=1}^{n} (x_i - x_{i+1})(y_i + y_{i+1})$$

The area of the object subcomponent is approximated by counting the number of cells labeled *interior* to or on the *boundary* of the subcomponent.

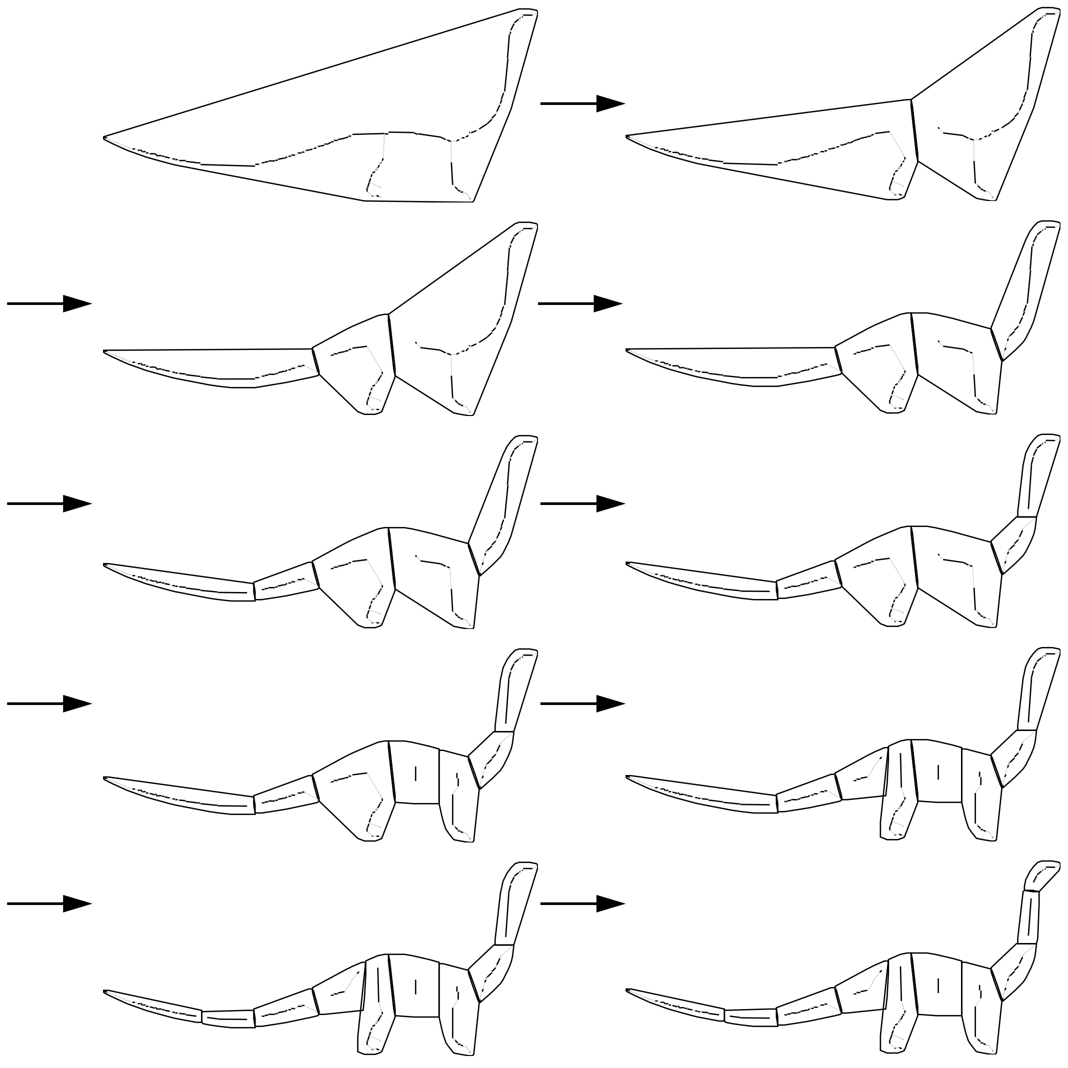


Figure 31: Step-by-step decomposition of dinosaur

6.5.3 Evaluation

The new representations are evaluated. If the quality has improved by an acceptable amount (i.e. the Benefit Threshold is exceeded), the decomposition step is accepted. If the step is rejected, then the process returns to the Decomposition step and attempts an alternate partitioning. If the partition is accepted, each of the two new representations is evaluated according to the Representation Quality value. If its value is above the allowable Quality Threshold, the representation is accepted as the final representation for that subcomponent, and the subcomponent is not processed further. If, on the other hand, the Quality value is below threshold, the new subcomponent is inserted into the subcomponent list and the process continues with the next subcomponent.

The evaluation process compares the Conservative Hull and Bounding box of the original component before the partitioning to the Conservative Hull and Bounding Box of the two resulting subcomponents. A cut is accepted if the Decomposition Benefit ($\mathcal{B}_{CH}$, $\mathcal{B}_{BB}$ or $\mathcal{B}_{AR}$) value of the representations exceeds the specified Benefit Threshold, *and* the sizes of the resulting subcomponents are above the *Minimum Size* ($\mathcal{S}_{MIN}$).

If the quality of the resulting decomposition does not pass the evaluation, the cut is rejected, and the process continues on a different subcomponent. If the partition is accepted, the process continues recursively on the two new components. The parent component is removed from the component list, and the new components are inserted into the sorted list according to their Representation Quality value. The process will stop when when the Quality Threshold is met for all components or if no more desirable cuts can be found.

The Benefit Threshold is set to be a fixed percentage higher than the specified value in the beginning, and is then reduced to the input Threshold value as the process continues. This method attempts to find the best partitions first.

Figure 31 illustrates the step-by-step decomposition of the dinosaur example. The Conservative Hull is shown for each component, in addition to a representation of the Axial Shape Graph.

6.5.4 Output Representation

The application is fully automatic, and generates two types of output files. The first file is in the UG3 format and contains polygonal descriptions of the components of the resulting Conservative Hull Hierarchy (the Bounding Box Hierarchy can be trivially regenerated from the Conservative Hulls).

The second output file can be used to visualize the decomposition process. The file contains the G-Rep, Part Tree, and generated representations for each decomposition step of the representation generation pipeline. The file is accepted as input to an interactive visualization program provided by the application that allows the user to visually step through the decomposition process, by either following the decomposition steps iteratively, or hierarchically .

6.6 Results

We have utilized this application to generate representations for a variety of 2D objects. In all cases the application was able to meet the specified Representation Quality Thresholds. Figure 32 shows the representation hierarchy produced for the dinosaur. More results are shown in Appendix 1. We utilized the same threshold values, chosen somewhat arbitrarily, for all of the examples:

Quality		*Benefit*		*Size*	
$\delta\mathcal{Q}_{CH}$	0.90	$\delta\mathcal{B}_{CH}$	0.80	$\mathcal{S}_{MIN}$	60
$\delta\mathcal{Q}_{BB}$	0.50	$\delta\mathcal{B}_{BB}$	1.02		
$\delta\mathcal{Q}_{AR}$	0.33	$\delta\mathcal{B}_{AR}$	1.02	$\mathcal{S}_{MAX}$	5000

The average and maximum Hausdorff distances ($\mathcal{H}_{AVG}$ and $\mathcal{H}_{MAX}$) for the resulting leaf parts in the hierarchy are also shown below each figure.

The representations we have generated meet the Representation Criteria specified in Section 6.1:

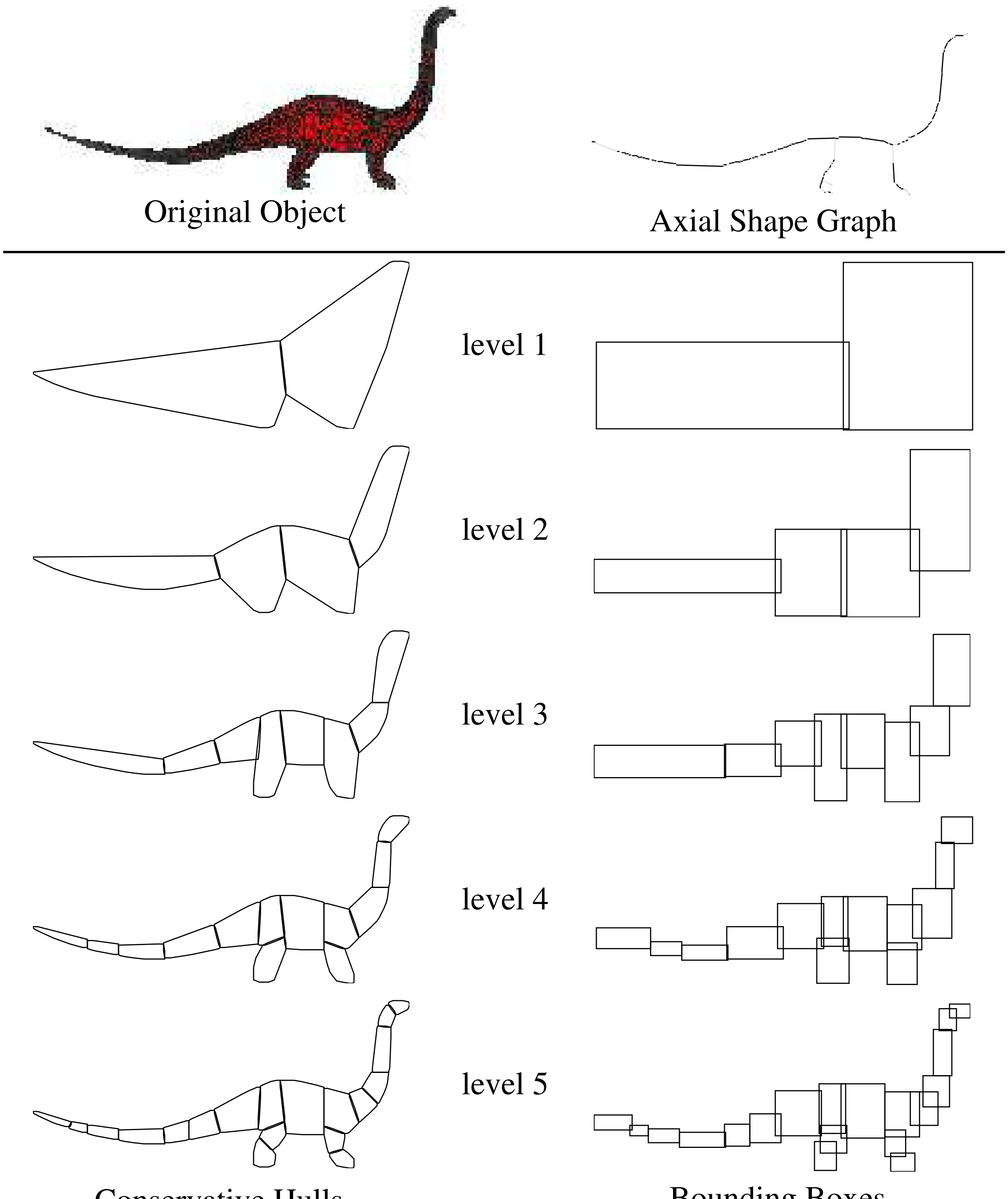


Figure 32: Dinosaur: $\mathcal{H}_{MAX} = 5.1, \mathcal{H}_{AVG} = 1.8, \mathcal{Q}_{CH} = 0.9, \mathcal{Q}_{BB} = 0.5, \mathcal{Q}_A = 0.33, \mathcal{B}_{CH} = 0.8, \mathcal{B}_{BB} = 1.02, \mathcal{B}_A = 1.02$

1. **Correctness**: The output representations are provably correct, since the Conservative Hull encloses the spatial representation which is guaranteed by construction to be conservative.

2. **Efficient Intersection Testing**: The representations are convex, and therefore because we are assuming a Collision Detection system utilizing the Lin-Canny approach, intersection testing is extremely efficient. Since we are utilizing a hierarchical intersection test, the representations must also fit the data well to be efficient, and the hierarchy should be well-balanced. The application generates representations that meet the specified Representation Quality Threshold, and produces fairly balanced trees, indicating the potential for good overall performance.

3. **Efficiency of Representation**:

   - **Detail**: The filtering provided by the spatial representation removes details irrelevant to the Collision Detection task.
   - **Redundancy**: The Conservative Hull representations produced in most cases contain very little overlap. The application does not explicitly test for overlap of subcomponent representations when performing the decomposition. Such a test could be built into the evaluation module if desired.
   - **Size**: The Absolute Size Threshold $\mathcal{S}_{MAX}$ ensures the decomposition of any relatively large components.

6.6.1 Effects of Object Rotation

The representation hierarchy is constructed by decomposing the object into subcomponents based on its shape, and constructing convex approximations for each subcomponent. Since both the shape and the convex hull of an object are invariant to rotation, the output representations generated from an ideal process should be insensitive to object rotation as well.

The decomposition process of our Representation Generation Application utilizes the Axial Shape Graph, which is based on the discretized, axis-aligned Cell-Based Spatial Representation. The CSR is intrinsically sensitive to rotations, and to noise and small perturbations in the object boundary. Figure 33 illustrates the ASG for the dinosaur rotated 5,10,15,45, and 90 degrees. It is clear that the skeleton is different, but the main structure is present in all cases. The differences appear as small branch details off the main structure. The decomposition process is rather insensitive to these extra branches. The ASG is utilized as a hint only, and the evaluation of any possible cuts generated by these branches shows them to be inappropriate partitions.

While not invariant to rotation, our method of axis construction filters out much of the noisy information found in many skeletonization approaches, and is therefore less sensitive to rotation. Many approaches in the vast skeletonization literature have addressed the issue of stability. The early work of Pizer presents a multi-resolution blurring approach to produce more stable axial representations [27]. It is also possible to produce an axial representation that is rotation invariant, such as the more recent method of *cores* presented by Pizer et al [6]. It would be possible, therefore, to replace our axial generation approach with one that is more stable. A main advantage of our ASG decomposition approach, however, is that we do not require that the axial representation be exact, or even particularly stable. The purpose of the graph structure is to capture general shape feature trends, not exact information. In the decomposition process, the ASG is utilized to provides hints about the location of potential subcomponents, and about the global structure of the object. In practice, our approach performs very well under object rotation, despite the possible changes in the underlying Geometry Representation structure.

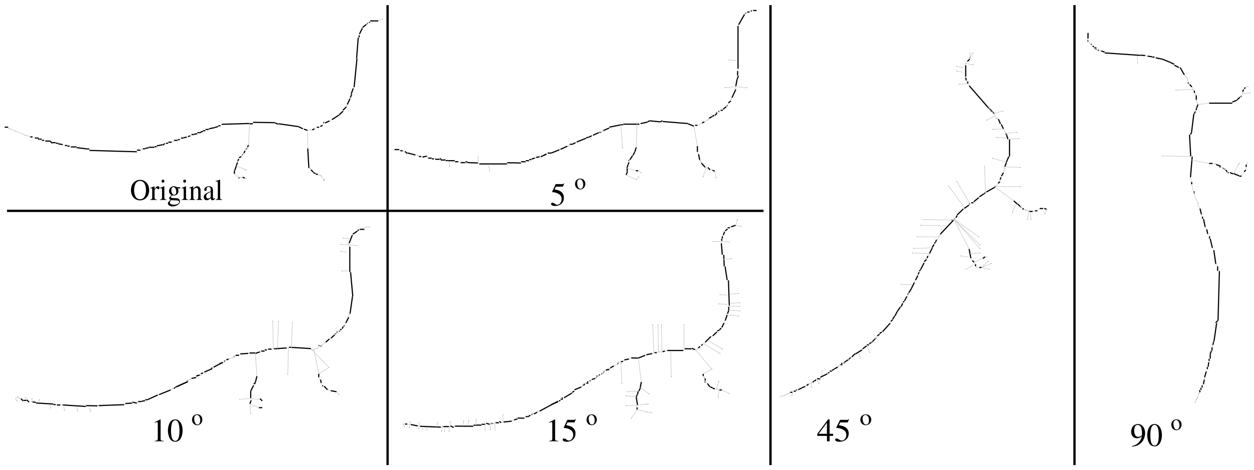


Figure 33: Effects of Rotating Object on Axial Shape Graph

To determine the sensitivity of the approach to object rotation, we have run the application on rotated versions of a single input object. We then compared the representation hierarchy generated with the rotated object as input, to a rotated version of the hierarchy generated for the original, un-rotated object. Figure 34 shows the finest level of the Conservative Hull representation hierarchy when the application is run on the object after it has undergone the different rotations. The figure on the left in each case is the output representation, and the figure on the right is created by taking the output of the application when run on the unrotated dinosaur, and then rotating it by the specified amount. The results are similar in that the same features are captured and the hierarchical decompositions are similar. The results are not exactly the same, but that is not surprising. There is no single correct representation. The application is set to the task of producing a representation that meets the specified metrics; in all cases both representations satisfy the metrics, and are therefore equivalent under the given framework.

6.6.2 Application Performance

The application running time depends on the number of cells representing the object in the Cell-Based Spatial Representation, and on the number of parts produced. The running time is also dependent on the number of partitions that are tried and rejected. This is related to the shape complexity of the object, but hard to quantify.

The amount of per-cell computation is significant in the current implementation, resulting in an overall process requiring minutes for any object of reasonable detail. At each decomposition step, the next cut is determined, made, and evaluated. In the current implementation, this involves recalculating the ASG structure for each subcomponent. It would be possible to achieve significantly faster results by utilizing the ASG information of the parent component, and performing local fix-ups at the partitioning boundaries to recalculate the ASG structure.

Figure 35 illustrates the effects of cell size and number of partitions on the application running time. A rectangle is utilized as the input object to remove any factors due to shape complexity. The largest component of the rectangle is divided approximately in half along its maximum dimension at each partitioning step. The graph in 35a shows the effects of object size on the application running time: times for a rectangle undergoing a single partition are shown along with the best fitting line on a log-log scale as the size of the rectangle in cells is increased (i.e. the cell size is decreased). In this case, the

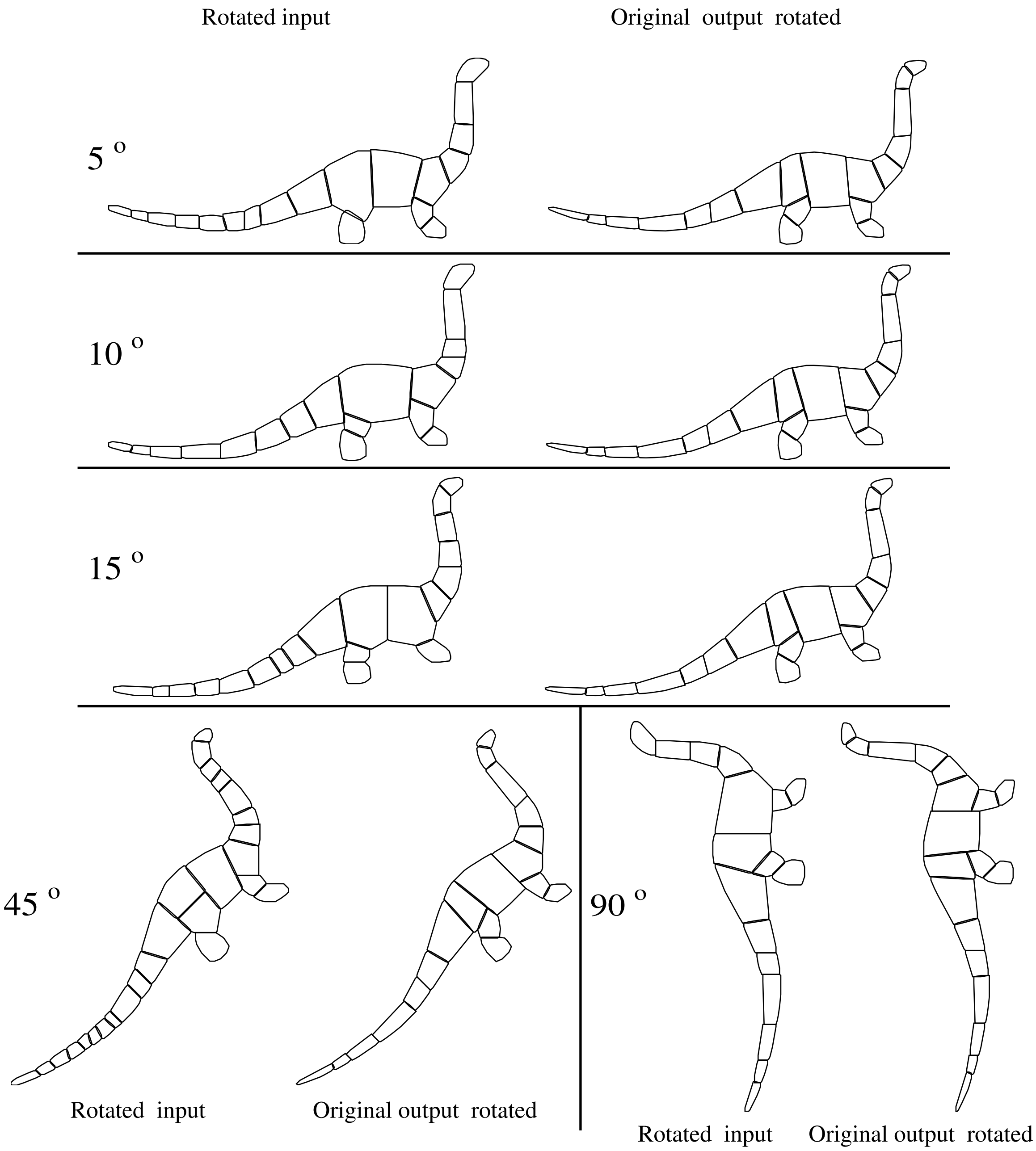


Figure 34: Effects of Rotating Object on Output Representation

running time increases linearly with the number of cells. The graph in 35b illustrates the effect on the running time when the number of partitions is increased. Again, the increase in running time is approximately linear as the number of partitions increases. Figure 35c shows the running time on a log-log scale for the various cell sizes, with a line fit to the data for each different number of partitions (1,2,4,8,16). The partition count is shown to the left of each graph.

The following table lists approximate running times of the application for some of the examples from the appendix. The overall size of the object in cells is listed, as well as the total number of parts in the resulting hierarchy, and total running time. All times were based on running the application on a Silicon Graphics 200MHZ Indigo2 with 64MB of main memory, taking the CSR representation as input.

Application Performance								
Object	Cactus	Skeleton	T-Rex	Dinosaur	Marlin	Frog	Duck	Snowflake
Size (cells)	1740	7129	7858	8527	10209	10876	14946	17434
Parts	13	67	87	37	39	45	29	101
Time (s)	42	583	2143	340	770	1055	1410	2689

6.7 Comparison to Previous Work

Since we assume the Collision Detection system utilizes a variant of the Lin-Canny intersection algorithm for convex objects, our approach generates a hierarchy of convex approximations whose union is guaranteed to conservatively cover the object. Our approach has several advantages over a strict convex decomposition of the object. First, it has been shown that the problem of optimal convex decomposition is NP-hard [7]. More importantly, the worst case lower bound on the number of resulting subcomponents is $O(N^2)$, where N is the number of concave vertices [2, 7]. Consider a polygonal approximation to a half circular arc. With a strict convex decomposition, this object would be decomposed into as many pieces as there are edges approximating the curve. There is clearly a tradeoff between the number of subcomponents and the tightness of the convex approximation.

Bounding boxes are often utilized as the sole approximating structure in the hierarchy. Ponamgi et al [28] automatically construct an axis-aligned bounding box hierarchy from the input object utilizing an approach based on a variant of octree subdivision. Optimally aligned bounding boxes have been proposed [18], offering rotation invariance at a slightly higher intersection testing cost. But whether optimally aligned or not, bounding boxes most often do not provide a tight fit of the data.

The approach of Hubbard [21] utilizes spheres as approximations for the *broad* phase of Collision Detection. Spheres are rotationally invariant, and allow very efficient intersection testing , but suffer from the same representational efficiency problem as bounding boxes; most objects do not naturally decompose into a set of components that are closely approximated by minimally overlapping spheres.

Hubbard's sphere hierarchy construction is similar to our approach in that it utilizes an approximate medial axis representation when choosing a set of spheres to represent the object. The medial axis is calculated by constructing the Voronoi diagram [29] of the object and taking points along Voronoi cell boundaries as the axial points. Each such cell boundary point is equidistant to two points on the object boundary. In 2D, a circle is associated with each axial point. The axial point is the center of the circle, and the circle passes through three points (called the *forming points*) on the object's boundary. The representation is progressively simplified by merging adjacent circles with a single circle that passes through the *forming points* of both circles. The result may not be conservative, so an additional pass is necessary to add the necessary circles to ensure full coverage.

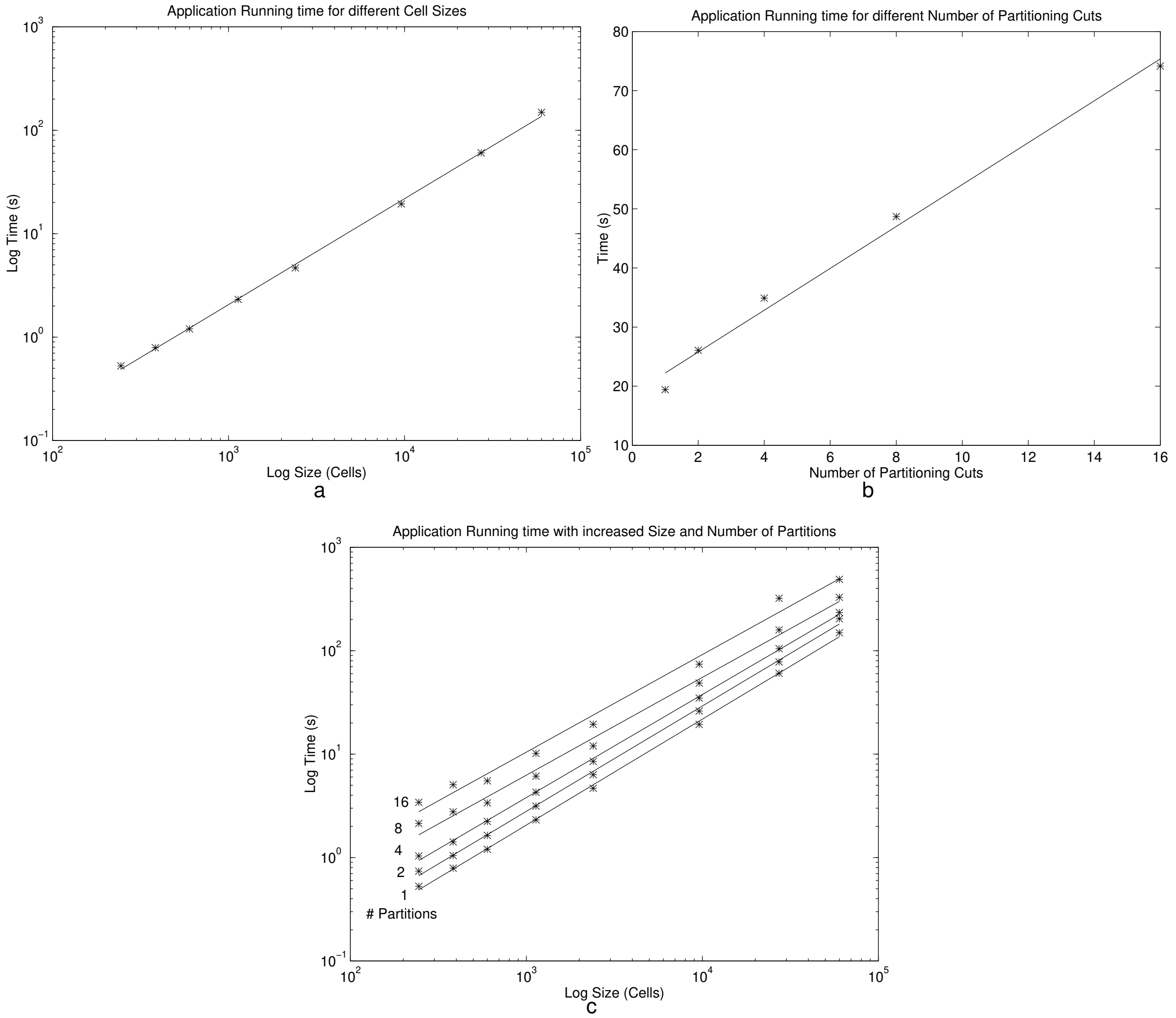


Figure 35: Application running times: a) Running times for a rectangle at different cell sizes undergoing a single partition. b)Running times for a size of 9600 cells with 1,2,4,8 and 16 partitions. c)Running times for each size and partition count.

7 Conclusion/Future Work

This thesis provides two main contributions: first, the G-Rep hierarchy is introduced as a means of encapsulating the geometric characteristics of an object, second, a general framework is presented that utilizes the G-Rep in conjunction with task-specific metrics to automatically produce representations for geometric tasks.

We have shown how this structure can be utilized to generate representations specialized for the task of Collision Detection in 2D environments. The general framework has proved successful in generating representations that meet the task specifications for a variety of 2D objects. The current implementation is a prototype, and could be improved upon and extended in many ways.

Implementation optimizations could substantially decrease the running time, both by utilizing local update operations and the multi-resolution information which is currently calculated, but not used, in the representation generation process. It is not clear, however, if even an improved approach could be utilized as anything but a preprocessing step.

To increase efficiency and stability, the Axial generation module could be replaced by an alternate construction method such as used in producing Pizer's *cores*. Pizer's approach also provides additional *exoskeleton* information which represents object concavities explicitly, and could be utilized in the decomposition process. Most importantly, this approach is far more efficient, generating axial information on demand and making it potentially feasible to have an approach that works at interactive speeds. Even if it is not possible to generate axial information for all objects in realtime, with a more efficient approach it may be possible to allow some dynamic updates on the shape of objects, or allow new objects to be inserted into the environment at runtime.

We believe this basic framework can be extended to accommodate a variety of geometric tasks. In particular, we will apply this approach to generate cell-and-portal representations to be used for visibility preprocessing, and as a means of introducing hierarchy in unstructured models. In addition we would like to investigate incorporating different classes of metrics beyond geometric measurements into the framework, most specifically perceptually-based metrics.

Moreover, this work provides a solid stepping stone towards the development of an analogous approach to perform object abstraction in 3D environments, a realm in which there is a strong need for practical and efficient task-specific representations.

8 Appendix

This appendix illustrates the representation hierarchies generated by our approach for various objects. Each figure shows the original object, its Axial Shape Graph, and the Conservative Hull and Bounding Box Representation Hierarchies. The Hausdorff distance values are noted under the figure.

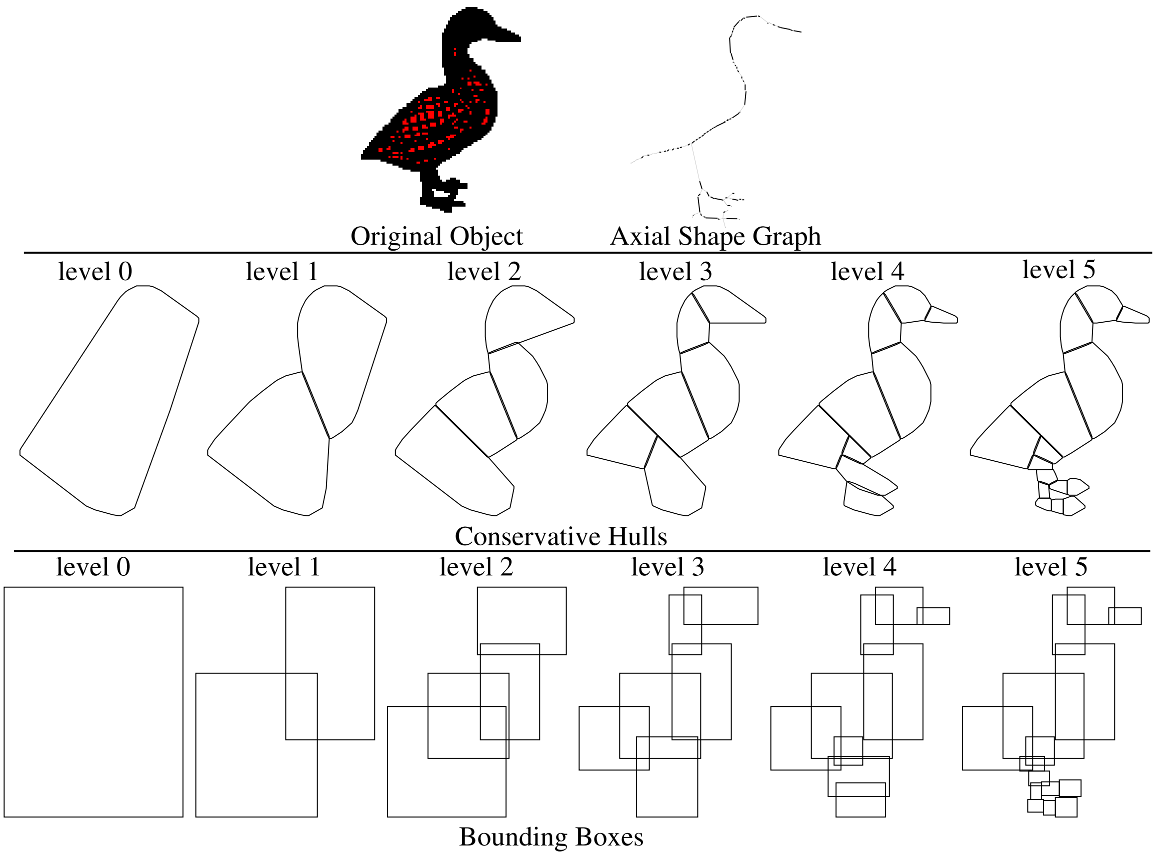


Figure 36: Duck: $\mathcal{H}_{MAX} = 7.3, \mathcal{H}_{AVG} = 2.9$

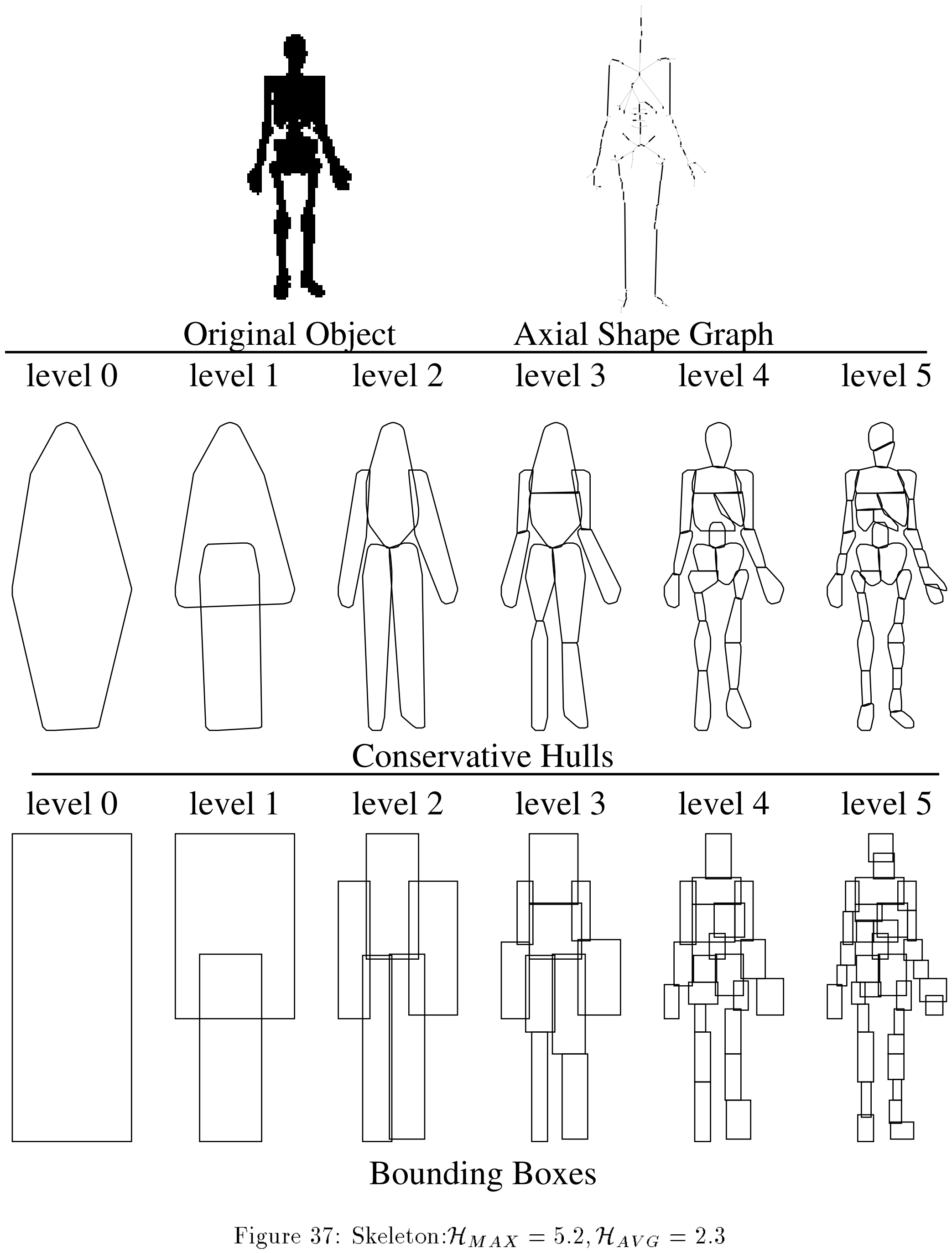


Figure 37: Skeleton:$\mathcal{H}_{MAX} = 5.2, \mathcal{H}_{AVG} = 2.3$

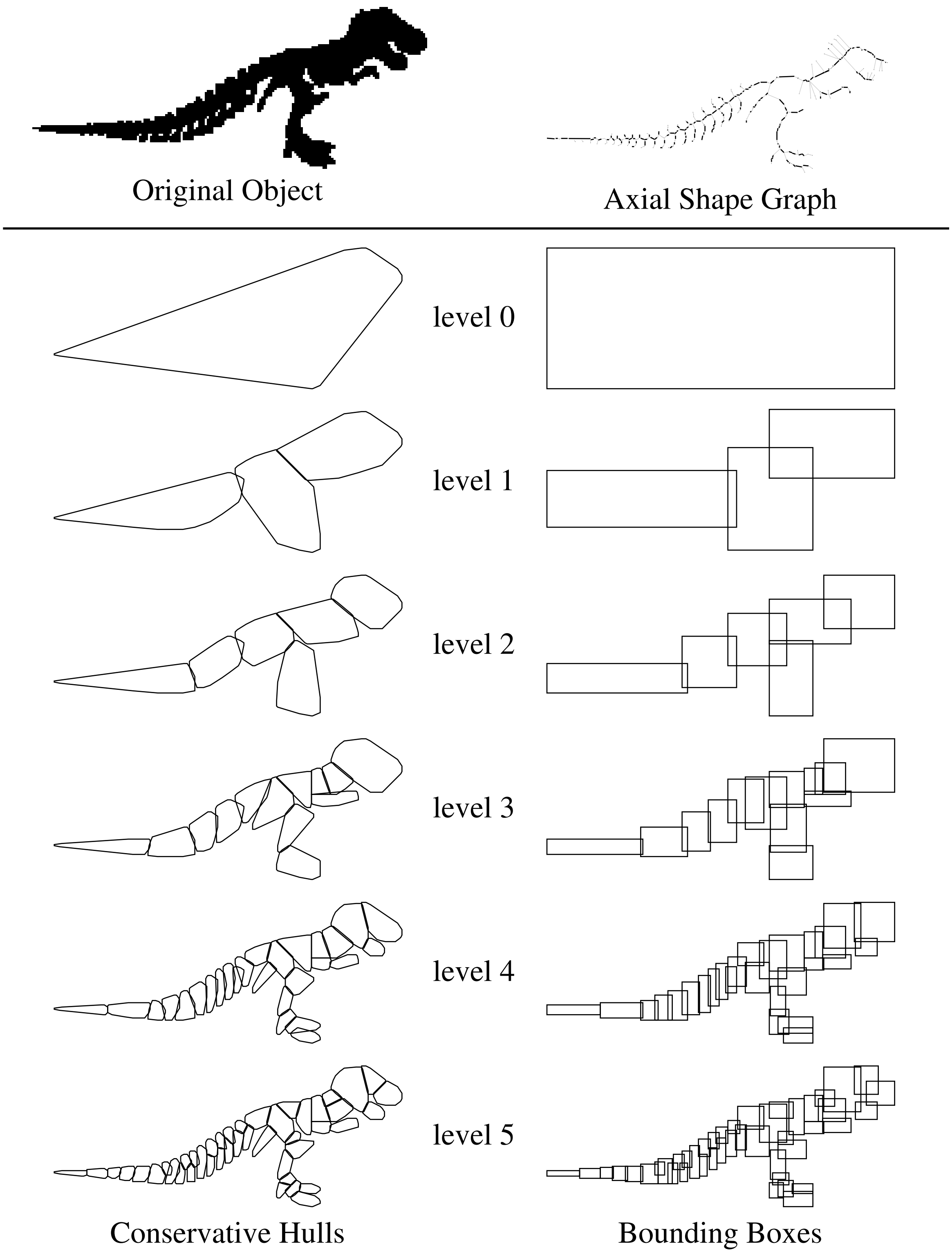


Figure 38: T-Rex: $\mathcal{H}_{MAX} = 7.3, \mathcal{H}_{AVG} = 3.0$

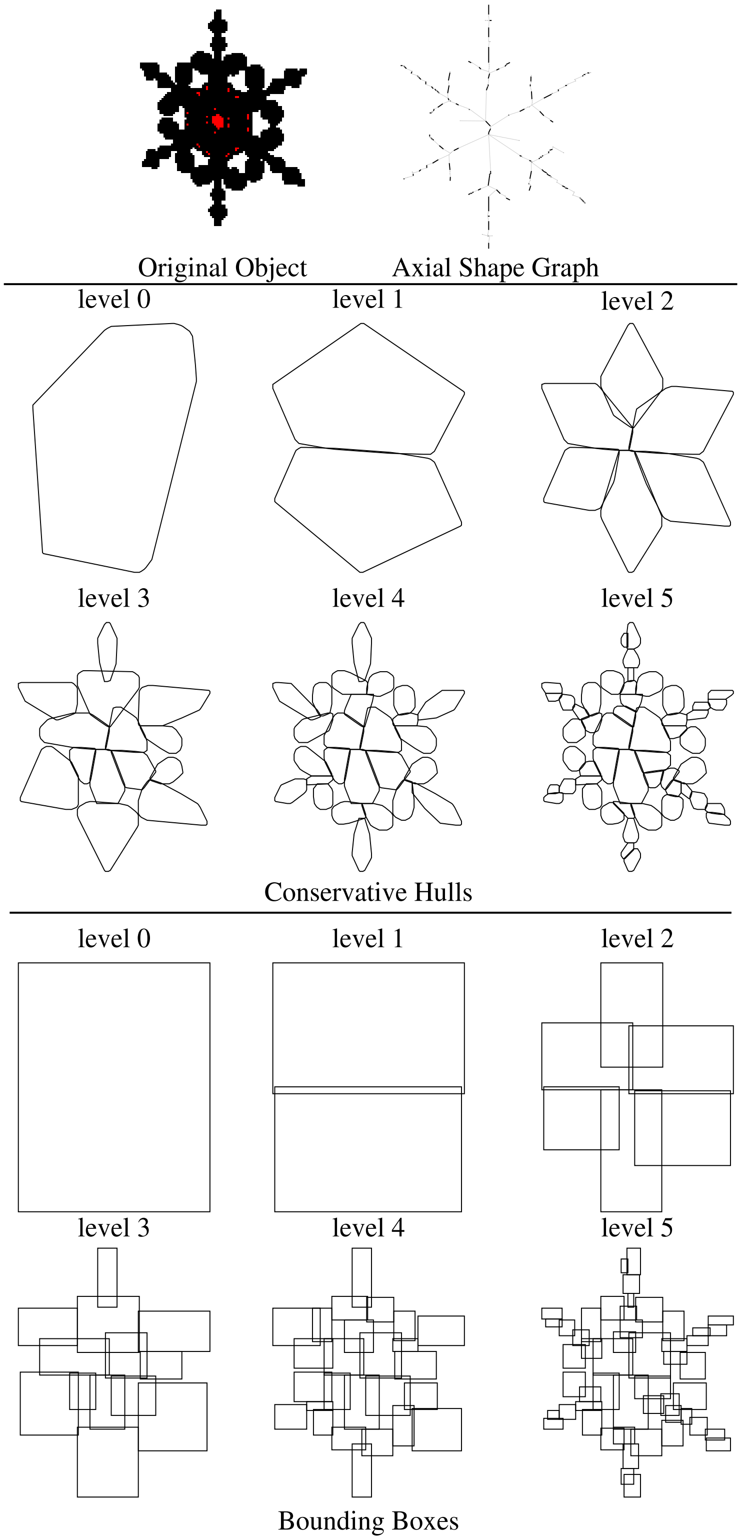


Figure 39: Snowflake:$\mathcal{H}_{MAX} = 9.1, \mathcal{H}_{AVG} = 2.4$

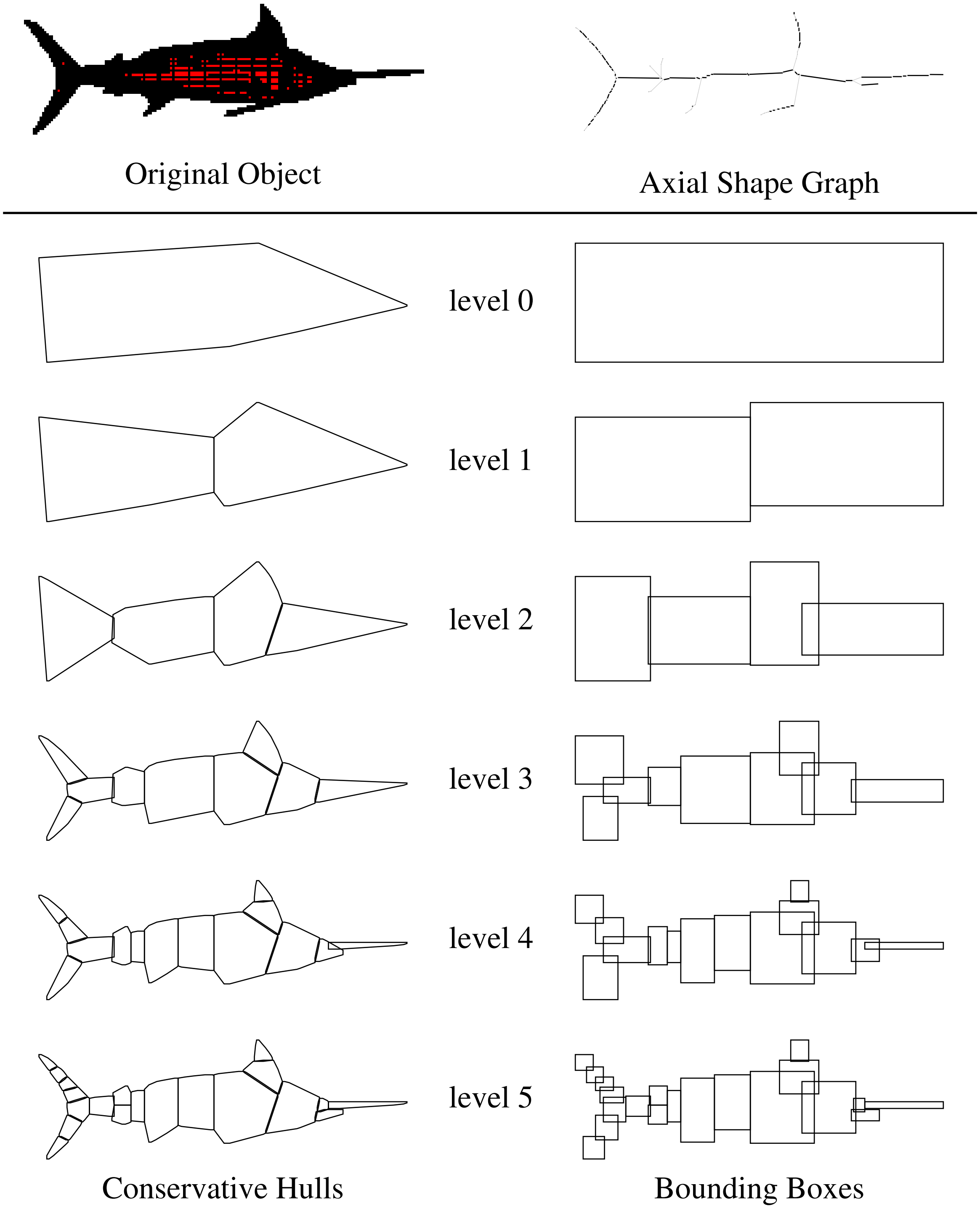


Figure 40: Marlin: $\mathcal{H}_{MAX} = 31.9, \mathcal{H}_{AVG} = 3.4$

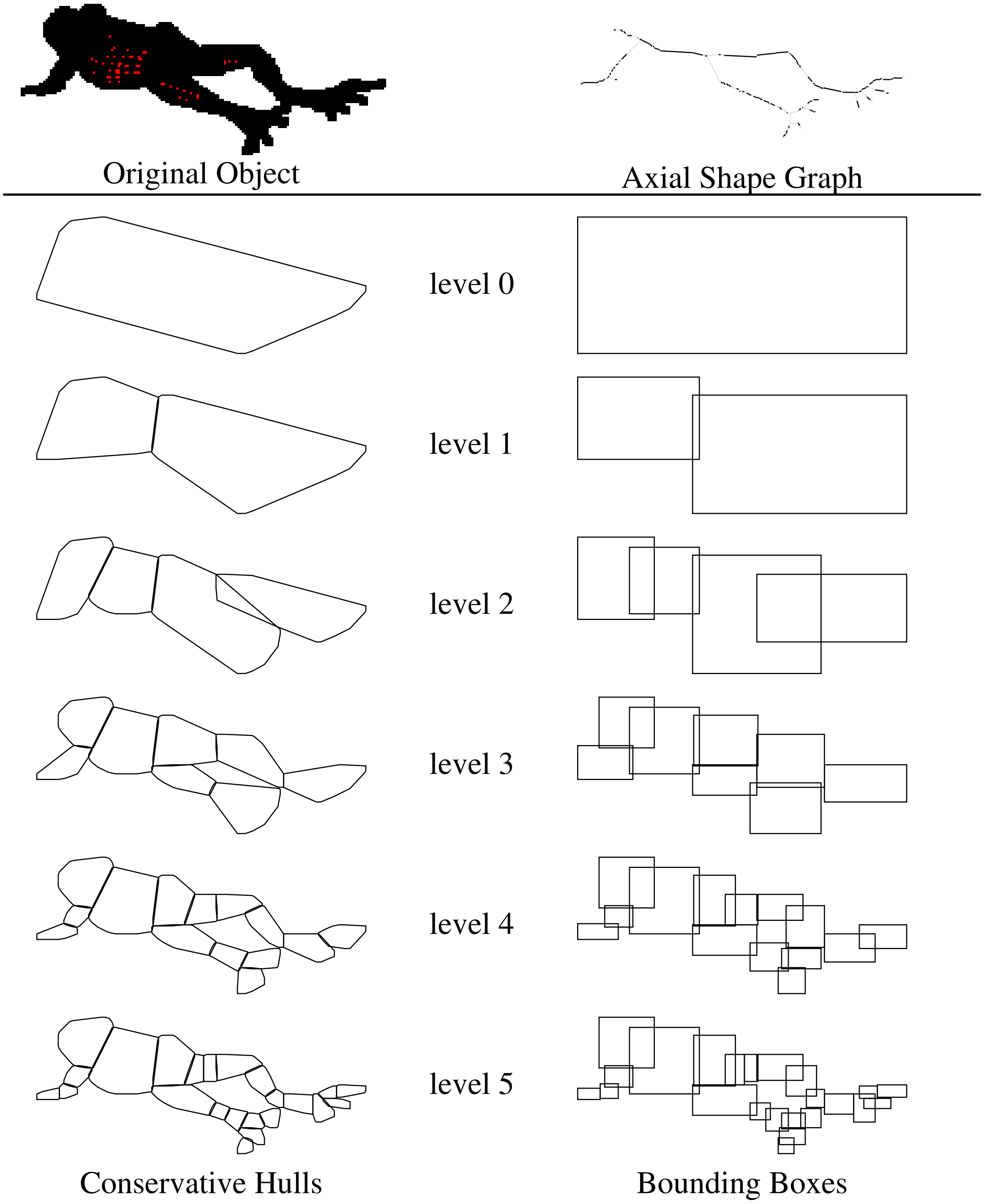


Figure 41: Frog: $\mathcal{H}_{MAX} = 8.5, \mathcal{H}_{AVG} = 2.6$

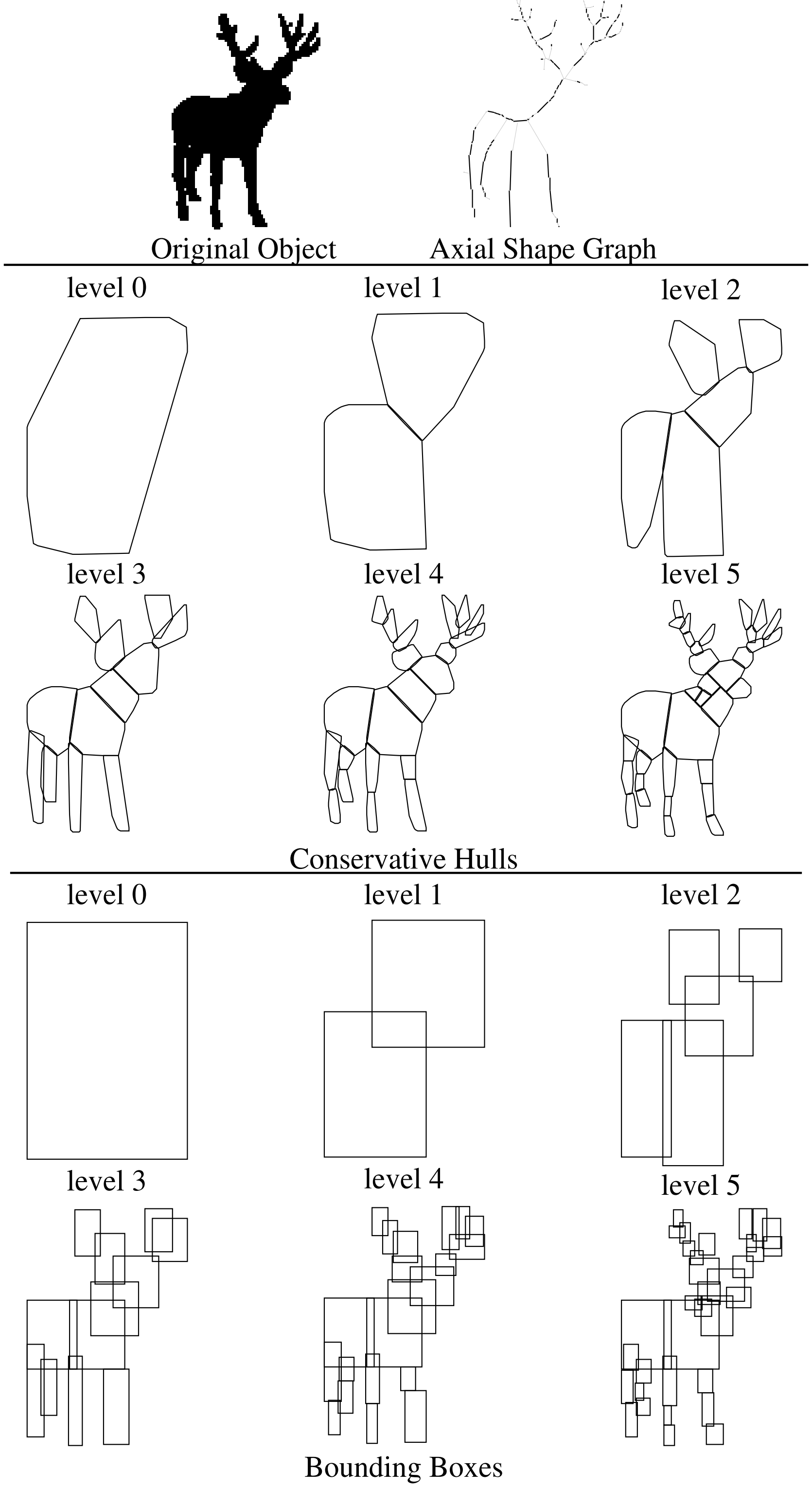


Figure 42: Muledeer: $\mathcal{H}_{MAX} = 9.14, \mathcal{H}_{AVG} = 2.7$

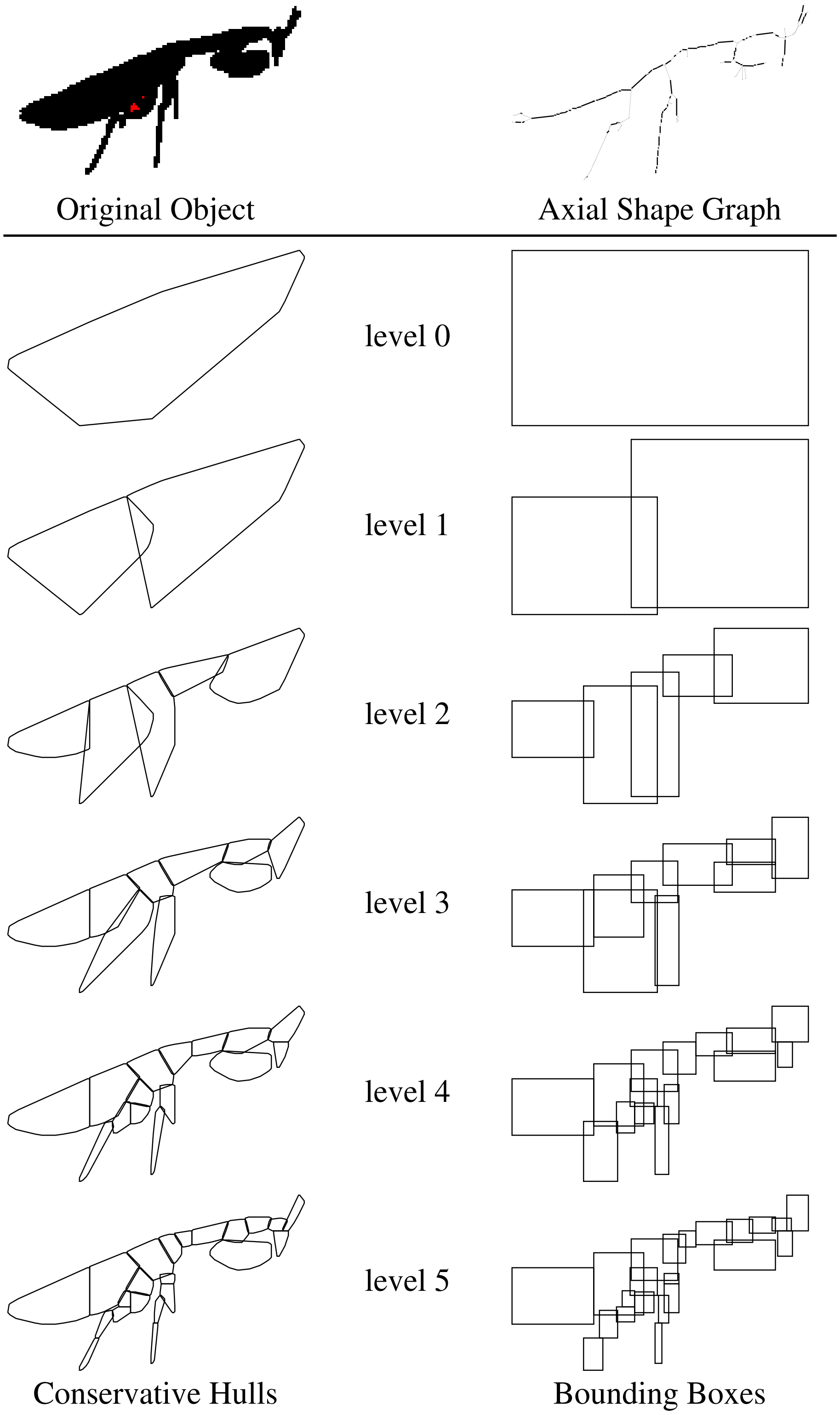


Figure 43: Praying mantis: $\mathcal{H}_{MAX} = 14.4, \mathcal{H}_{AVG} = 3.5$

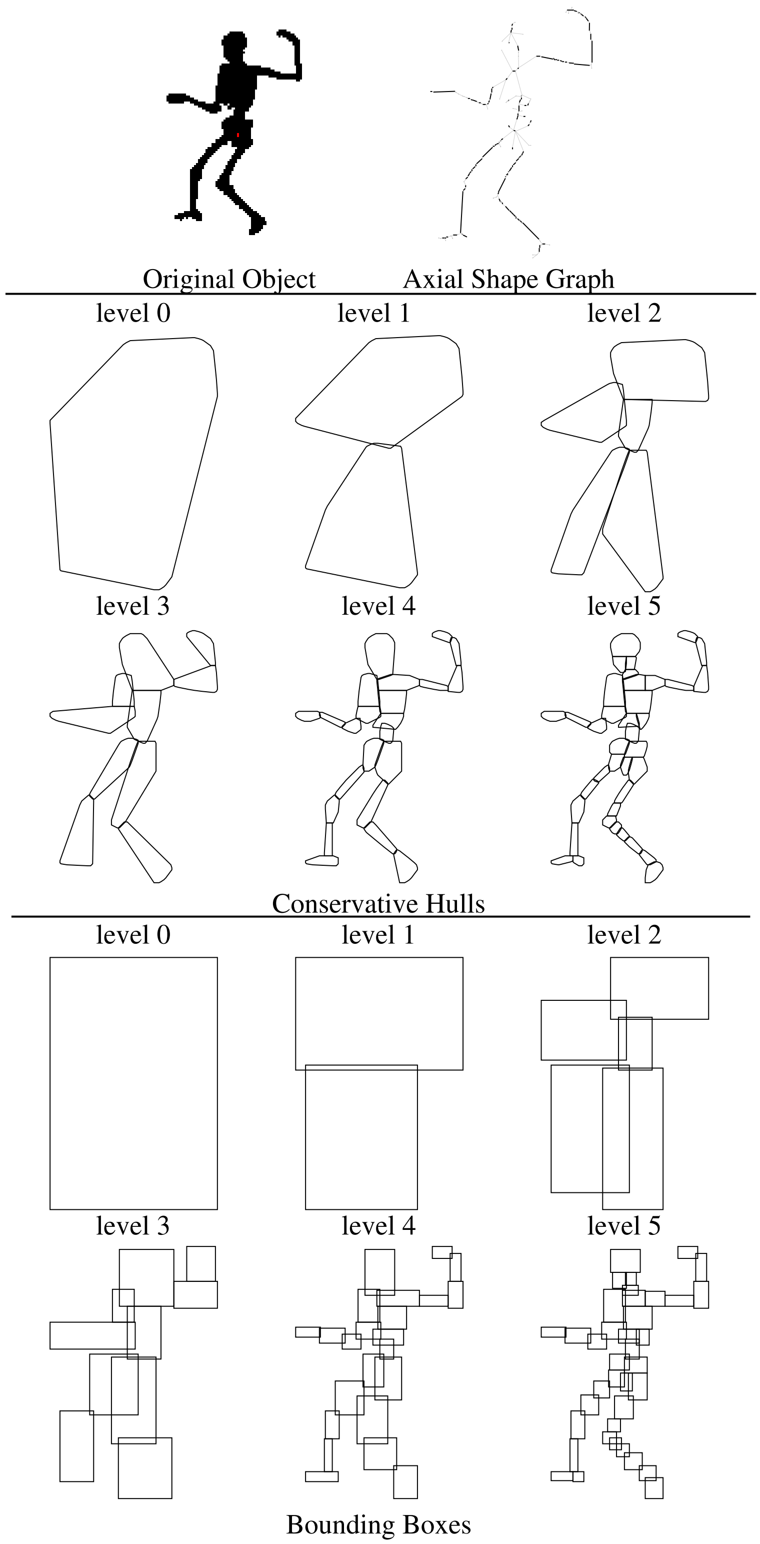


Figure 44: Dancing skeleton: $\mathcal{H}_{MAX} = 7.9, \mathcal{H}_{AVG} = 2.7$

Acknowledgements

I would like to thank my advisor Carlo Séquin for his guidance and contributions to this work, and for the inspiration he provides through his endless energy, creativity, and enthusiasm. Thanks to Jitendra Malik for being my second reader. Many thanks to the members of the Berkeley Graphics Group, past and present: Ajay Sreekanth, Randy Keller, Sara McMains, Christopher Healey, Laura Downs, Tom Funkhouser, Mark Brunkhart, Rick Lewis, Rick Bukowski, Jordan Smith, and Amy Hsu, for assistance and feedback on this work and for general camaraderie and support.

This work was partially supported by NEC and by ONR MURI grant N00014-96-1-1200.

The objects utilized as input appearing in the Appendix are captured images of 3D data sets available from Viewpoint DataLabs.

References

[1] C. Arcelli and G. Sanniti di Baja. Finding local maxima in a psuedo-euclidean distance transform. *CVGIP:Graphical Models and Image Processing*, 43:361–367, 1988.

[2] C. L. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *Siam Journal of Computing*, 21:2:339–364, April 1992.

[3] H. Blum and R.N. Nagel. Shape description using weighted symmetric axis features. *Pattern Recognition*, 10:167–180, 1978.

[4] G. Borgefors. Distance transformations in arbitrary dimensions. *CVGIP:Graphical Models and Image Processing*, 27:321–345, 1984.

[5] M. Brady and H. Asada. Smooth local symmetries and their implementation. *International Journal of Robotics Research*, 3:3:36–60, 1984.

[6] C. A. Burbeck and S. M. Pizer. Object representation by cores: Identifying and representing primitive spatial regions. *Vision Research*, 35:13:1917–1930, 1995.

[7] B. Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *Siam Journal of Computing*, 13:3:488–507, August 1984.

[8] J. D. Cohen, M. C. Ling, D. Manocha, and M. Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. In *Proceedings of the Symposium on Interactive 3D Graphics*, pages 189–196, 1995.

[9] Gregory S. Couch. Berkeley UNIGRAFIX 3.1- data structure and language. Technical Report UCB/CSD-94-830, University of California, Berkeley, September 1994.

[10] P.E. Danielson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:3:227–248, November 1980.

[11] G. Sanniti di Baja and E. Thiel. (3-4)-weighted skeleton decomposition for pattern representation and description. *Pattern Recognition*, 27:8:1039–1049, 1994.

[12] G. Sanniti di Baja and E. Thiel. Skeletonization algorithm running on path-based distance maps. *Image and Vision Computing*, 14:47–57, 1996.

[13] L. Dorst. Pseudo-euclidean skeletons. In *International Conference on Pattern Recognition (ICPR)*, pages 286–289, October 1986.

[14] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*, chapter Basic Raster Graphics Algorithms for Drawing 2D Primitives. Addison-Wesley, 1990.

[15] T. A. Funkhouser. *Database and Display Algorithms for Interactive Visualization of Architectural Models.* PhD thesis, University of California, Berkeley, 1993.

[16] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Computer Graphics (Proceedings SIGGRAPH)*, pages 247–54, 1993.

[17] T. A. Funkhouser, S. J. Teller, C. H. Séquin, and D. Khorramabadi. UCB system for interactive visualization of large architectural models. *Presence: Special Issue on Teleoperators and Virtual Environments*, 5(1):13–44, Winter 1995.

[18] S. Gottschalk, M. Lin, and D. Manocha. OBBTree:a hierarchical structure for rapid interference detection. In *Computer Graphics (Proceedings SIGGRAPH)*, pages 171–80, August 1996.

[19] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel Based Object Simplification. In *Proceedings IEEE Visualization*, pages 296–303, 1995.

[20] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, pages 171–183, June 1996.

[21] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1:3:218–230, September 1995.

[22] R. Kimmel, D. Shaked, and N. Kiryati. Skeletonization via distance maps and level sets. *CVGIP:Graphical Models and Image Processing*, 62:3:382–391, 1995.

[23] S. Lee, L. Lam, and C. Y. Suen. A systematic evaluation of skeletonization algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:5:1203–1225, 1993.

[24] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In *International Conference on Robotics and Automation*, pages 1008–1014, May 1991.

[25] C. W. Niblack, P. B. Gibbons, and D. W. Capson. Generating skeletons and centerlines from the distance transform. *CVGIP:Graphical Models and Image Processing*, 54:5, September 1992.

[26] R.L. Ogniewicz and O. Kubler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28:3:343–359, 1995.

[27] S. M. Pizer, W.R. Oliver, and S.H. Bloomberg. Hierarchical shape description via the mulitresolution symmetric axis transform. *IEEE Transactions on Pattern Recognition and Machine Intelligence-PAMI*, 9:4:505–511, 1987.

[28] M. Ponamgi, D. Manocha, and M. C. Lin. Incremental Algorithms for Collision Detection Between Solid Models. Technical Report 94/061, Department of Computer Science, University of North Carolina, Chapel Hill, 1994.

[29] F.P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, New York, 1985.

[30] H. Rom and G. Medioni. Hierarchical decomposition and axial shape description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:10:973–981, October 1993.

[31] A. Rosenfeld. Axial representations of shape. *CVGIP:Graphical Models and Image Processing*, 33:156–73, 1986.

[32] J.C. Russ. *The Image Processing Handbook*, chapter 7: Processing Binary Images. CRC Press, 1995.

[33] F. Y. Shih and C. C. Pu. A skeletonization algorithm by maxima tracking on euclidean distance transform. *Pattern Recognition*, 28:3:331–341, 1995.

[34] S. J. Teller. *Visibility Computations in Densely Occluded Polyhedral Environments.* PhD thesis, University of California, Berkeley, 1992.

[35] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics (Proceedings SIGGRAPH)*, pages 61–69, 1991.

[36] M.W. Wright and F. Fallside. Skeletonisation as model-based feature detection. In *IEEE Proceedings I (Communications, Speech and Vision)*, volume 140:1, pages 7–11, February 1993.