THE UNIVERSITY OF CALGARY

Using Multi Agent Systems for Illustrative Rendering of Implicit Surfaces

by

Pauline Elizabeth Jepp

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

September, 2007

Your file Votre référence
ISBN: 978-0-494-34178-0
Our file Notre référence
ISBN: 978-0-494-34178-0

Abstract

In this dissertation a Multi-Agent System (MAS) for illustrative rendering of implicit surfaces is presented. The system uses techniques from MAS to advance and combine previous particle-based object-space methods and agent-based image-space approaches. Previous approaches to pen-and-ink style renderings of implicit surfaces were based on particle systems, which were slow to achieve a good distribution of particles across a complex surface, and subsequently trace features. The focus was previously on creating an automatic system for feature identification and illustration.

The method proposed in this thesis develops traditional particles into semi-autonomous agents (called smarticles) that use goal directed behaviours to complete tasks. Tasks include identifying and tracing feature lines, place strokes or stipples, and explore a region. Stipples or short or long, curved or straight, strokes can be placed anywhere on the surface relative to smarticle paths. Smarticles also create colour accents or highlights using a low resolution polygonisation of the object that is also used for hidden line removal. Smarticles use attribute properties associated with individual or groups of primitives that create the model. Using this method any subtree in the data structure that represents the model can be rendered in a particular style.

Acknowledgements

Paul, thank you. Paul's support, laughing, cooking and general cheesiness has gotten me through many a late night and many late mornings.

My supervisor Brian Wyvill has been an inspiration, a motivation, a source of many good times, my introduction to climbing and my Martian-Scottish-English translation service. None of those things I will ever forget, many thanks. Mario Costa Sousa came on board as a co-supervisor quite by chance and set me off on the path that has lead me to this current topic. Thanks Mario for all your good ideas, great brainstorming sessions, and meetings! Jörg Denzinger was originally part of a brainstorming session, but became my beacon of light in the confusing waves of completing this thesis. Jörg, I appreciate all of your efforts, thank you very much. Many thanks go to the other members of my supervisory committee, Sheelagh Carpendale (who has given me some amazing food for thought), and Faramarz Samavati (who constantly inspires me to learn more maths). Also a big thanks to Joaquim Jorge whose words of wisdom helped me navigate a desert or two.

There are many others who have been an integral part of my time here in the Graphics Jungle. My heartfelt thanks go to Callum Galbraith, Ryan Schmidt and Kevin Foster for the good times, the discussions and a little bit of nonsense. Thanks also to Brendan Lane, Adam Runions, Erwin de Groot and Colin Smith, I have enjoyed being in the jungle with you all. I must also thank the administrative and technical staff, who collectively kept me on track and working. Thanks Camille, Lorraine, Debbie, Beverly, Jenny, Susan, Darcy, and last but most certainly not least Wayne Pearson.

I would also like to thank my friends who put up with me being absent and absent minded. Thank you all for keeping me laughing and reminding me about the funny and crazy side of life. The Steamers must get a mention here: the final countdown to a good time - always. Thanks especially to Callum, Ann, Genene, Daiva, Conrad, Davie, Allan, Susannah and Robin. And not forgetting Shannon - my climbing partner that has put up with my lack of climbing and become my lunchtime (much needed) distraction.

I must thank my friends at Thursday night yoga club at Ki Essentials, you have been a source of inspiration, education and calming in so many ways. Never mind the laughs and party tricks: Nora you are a true teacher, thanks. Thanks also to John Estabrooks, Dr Marie and Dr Nick for keeping me moving and mobile when my body complained about long hours in thesis-land.

My family have been a source of everlasting love, support and inspiration for me over the years and particularly throughout my PhD. To my baby sister Lindsay, I wish I could have brought you with me, I miss and love you more than words will ever express. Dad, oh dad, you gave me a few frights here and there, but in true Jepp fashion you refused to give in and taught me how to be strong and steadfast, there's a heart in this book for you, although your own obviously works better than we could have hoped for. To my mum: each and every day you become more precious to me, your love, strength and selflessness have always been inspiring. Thanks also to Bruce, Laura and Erin Jepp, for being a great example of how good family life can be. My (one of these days legally) in-laws have been fantastic. Deb, Auriel, Clare, Craig and Ryan have accepted me into their hearts and lives and treated me as a welcome part of the family, I can never thank you enough, your love and support

made the distance from my own family that little bit easier to bear.

This thesis is dedicated to my Father:

Michael Jepp.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In computer graphics, skeletal implicit surface modelling techniques have been used to create representations of a variety of objects [BBB+97]. One of the main problems associated with using implicit surfaces is the computation cost and resulting speed of rendering. A potential solution to this problem is to use a particle system to sample and render the surface. Particle systems were introduced to speed up surface visualisation, but also created the foundations for rendering implicit surfaces in various Non-Photorealistic Rendering (NPR) styles.

Particle systems commonly use oriented discs to visualise surfaces [WH94, RRS97, SH05]. Most of these surfaces are simple and relatively smooth; they are constructed using only a few simple primitives such as spheres and cylinders. Oriented discs do not, however, effectively illustrate many details of a more complex or less smooth surface. Other techniques have been used to capture these details. Pen-and-ink style strokes have been used to render both feature outlines and general surface strokes[1] [Elb98, Akl98a, Akl98b, FJW+05, SH05], which gives more information than unconnected particles alone. Particles can be rendered directly, for example

[1] In Line and Form [Cra04], Walter Crane states that "*Outline, one may say is the Alpha and Omega of Art.*" He explains that "*the function of outline is to act as the definition of the boundaries of form*". To enhance an illustration other lines are used: "*when we add lines or tints of shadow, local colour or surface, to an outline drawing, we are seeking to express form in a more complete way than can be done in outline alone.*" Features of a surface can therefore be illustrated using, what are termed in this thesis as, outlines and general surface strokes.

using discs or short lines, or their path of movement can be used, for example when a particle traces a silhouette [FJW+05]. Oil painting type strokes are created in [Akl98a], and polygonal models can also be created using the particles' final positions [RRS97, SH05].

There are two main drawbacks associated with using particle systems: the slow distribution method and the limited palette for rendering. The distribution method uses attraction and repulsion forces to constrain the particles to and spread them across a surface. This is the method presented in [WH94] and is the basis for most implicit surface particle systems [Akl98a, RRS97, FJW+05, SH05]. While this method can produce real time results for relatively simple models, complex objects (such as those created with the BlobTree [GNW02, Gal05]) can require a significant amount of time to cover the surface. A further consequence of this is related to feature outline identification methods used in most particle systems: particles identify locations of feature outlines when they are in close proximity to them [Akl98a, RRS97, FJW+05, SH05]. Local surface properties are used to identify areas of interest and subsequently steer particles.

Most of the previous approaches to creating pen and ink style renderings for implicit surfaces fully automate identification and visualisation of the object [Akl98a, RRS97, FJW+05, SH05]. The user has control over the orientation of short interior (general surface) strokes in [FJW+05] and can direct the placement of new particles to speed up identification of feature outlines.

One of the motivations for the research presented in this research is to provide the user with alternative rendering techniques. These techniques include automatic feature outline identification and tracing, and placement of general surface strokes

that cover the entire surface, as seen in the work by Foster et al [FJW+05]. They also include new methods of giving the user some control over where, how and how many strokes are placed, i.e. the position, direction and density of general surface strokes. The user indicates the position and direction of strokes, without having to draw each individual stroke, which can be very time consuming. A combination of automatic techniques and user direction allows a great deal of flexibility in creating an illustration.

1.2 Contributions

The research presented in this thesis constitutes solutions to the problems identified in previous particle-based systems. The contributions, therefore, are:

1. Management System:
    (a) Organises agents to collectively explore an implicit surface environment and visualise object data using NPR styles.
    (b) Analyses the object space to identify potential locations of feature outlines.
    (c) Interprets user requests to create appropriate tasks.
    (d) Prioritises tasks to be processed based on workload and workdone estimates.
    (e) Uses polygonisation for initial positions and voxel data thereby identifying feature outline locations faster than the previous methods [FJW+05, JWS06].

2. Smarticles:

   (a) Object space, agent-based method. Most previous object space methods are based on particle systems and most agent based methods are based in image space.

   (b) The Dual Tracking algorithm, which identifies and traces feature outlines due to discontinuities in the field or abrupt blends between primitives.

   (c) Apply steering and flocking behaviours to agents to both sample and render an implicit surface.

   (d) Create stipples from the Wander steering behaviour.

   (e) Allow the user to specify arbitrary stroke directions (not constrained by scene, view, or surface properties, such as principal directions of curvature or contour).

3. Mixed Rendering:

   (a) Supports the creation of shading for areas of an implicit surface relative to an individual or group of primitives.

   (b) Make available concept art styles of rendering for complex implicit surfaces: pen and ink style short and long strokes, stipples and colouring relative to surface feature outlines.

1.3 Overview of System Development and Design Motivation

Current techniques for NPR style rendering of complex implicit surfaces identify two main areas of improvement. The first involves increasing performance so that the time to create a rendering is decreased. Second, alternative rendering techniques can be provided to improve the palette of available styles and techniques.

In order to improve efficiency, field function evaluations should be reduced. The implicit model is evaluated (by the particle system) with every step that a particles takes. This information is not stored or evaluated, other than to identify if a feature outline has been crossed and requires tracing. Storing and evaluating this data with respect to local neighbouring samples can be used to predict (assume) the local behaviour of the surface. This would move the emphasis of the particle system to a more goal directed approach (find and trace feature outlines).

Particles as entities simply contain data about their properties, such as position and velocity. The particle *system* calculates the forces that act upon individual entities to alter these properties, for example to move them. In the context of this research, entities require additional capabilities that will enable them to perform tasks and achieve goals. A good solution to this general problem has been identified with the use of agents in a MAS. Instead of "blind" particles being used as data repositories, agents have a degree of autonomy that allows them to evaluate their situation and decide upon an action to perform. Agents can assume a great deal of responsibility of performing tasks (such as outline tracing). Spreading the workload in this way is one way of improving efficiency. Ultimately, agents can also

be distributed across a network or onto different processors, which would also show improvements.

Previous particle systems rely solely on automatic techniques for rendering, where the user may only have control over the orientation of general surface strokes and the appearance of feature outlines. Pen-and-ink rendering styles have more scope and depth. Although automatic techniques are highly desirable, providing the user with an added level of control over the image was identified to be a requirement. This would enable a user to create renderings with different appearances, for example using a more "sketchy" style to user directed focus of attention using varying levels of detail. This was designed to be independent of the methods to improve efficiency, although in this thesis they are presented in one system.

The system presented in this thesis was designed to implement and test these techniques. The design included a progressive approach that initially determined the feasibility. System development was, therefore, achieved in three steps:

1. Enhance particles to use behaviours to create new rendering and sampling methods, i.e. develop **sm**art p**articles** or smarticles.

2. Use smarticles information to examine the surface from a higher level (not limited to a smarticle's local) point of view to identify areas of interest, i.e. develop a basic manager to store and analyse data from smarticle exploration.

3. Further development of the smarticles and manager to be a MAS, which is capable of improving performance and providing a larger palette of rendering styles.

Initially, particles were given a new behaviour to identify and trace surface feature outlines due to discontinuities or abrupt blends [FJW+05], known as the Dual Tracking Algorithm. Particles were then given new steering and flocking behaviours to facilitate new methods of sampling the space and rendering images. The particle system was also enhanced to use a basic manager that analysed smarticle data to identify feature outlines. Smarticle data was organised using a voxel grid that represented the model in object space [JWS06]. The next step was to take advantage of techniques in distributed computing and use a Multi-Agent System (MAS) in place of the traditional particle system and the basic management scheme.

In creating the MAS, particles were developed into agents that have control over themselves with locomotion, awareness, analytical and communication capabilities. This gives agents self direction and autonomy and, in conjunction with the manager, allows them to actively seek out areas of interest in a goal directed manner. The manager interprets user requests to create tasks for agents. Agent data is stored by the manager in voxels and analysed to identify features and Prioritise tasks.

This is, therefore, a more sophisticated approach to representing, understanding and analysing the object space than traditional particle based systems. The final system achieves a faster distribution and, consequently, faster feature outline identification for complex models than the previous particle based methods.

An illustration is more effective with suitable cues, such as feature outlines and general surface strokes. Faster surface visualisation, however, comes at the cost of surface detail information. As a pre-processing step the MAS manager performs a low resolution polygonisation. This provides a good initial coverage of the surface and fast identification of potential features. A low resolution polygonisation is faster

than distributing particles across a complex implicit surface.

Feature outlines can easily remain undetected if a particle placement is sparse or a polygonisation resolution is too rough. There is no *general* method of guaranteeing that sampling rates (particle or agent step sizes, or polygonal voxel grid resolution) are appropriate for *all* details on the surface. More information about the surface is necessary to define an adequate sampling rate. For example the Nyquist theorem for sampling identifies that the sampling frequency must be equal to or greater than twice the highest frequency component of the analogue signal [Wei], i.e. the maximum frequency of the signal must be known. Similarly, a Lipschitz constant represents the maximum rate of change of a function in its domain [Wei]. Therefore, further knowledge about the behaviour of the function, or surface it represents, is required to guarantee an accurate sampling rate/level.

Smarticles offer a method of sampling the object space in variable user-selected Levels Of Detail (LOD) to identify features that are not found by the initial polygonisation. This means that limitations identified in [SIJ+07] regarding the assumption that the base mesh captures the functional surface topology do not apply here. This control allows the user to change the sampling rate without being expected to know the size of details on the surface.

In most pen-and-ink style renderers, the direction of strokes is calculated using principal directions of curvature [Elb98, BH98, FJW+05], contours [FJW+05], axes-aligned planar slices [Ric73] or using a function designed to steer particles to cover the surface [Akl98a]. In this research, strokes can also be placed using a random wander technique, or in directions specifically identified by the user (their orientation is not determined by surface details).

The system presented in this thesis also makes use of the polygonal data for hidden line removal and to create colour accents and highlights in a style reminiscent of concept art. Concept art is a term that has been used to identify a form of illustration that is used to convey a preliminary visual representation of an object. It is also known as *visual development* in traditional animation; the term has been used since the 1930s and, more recently, popularised in the automobile and video games industry. Shading, or colouring, of objects is limited to highlights or accents on limited parts of the surface. Shading, in this research, is calculated around a line, or from individual or groups of primitives. In the context of tree-like data structures, such as the BlobTree, any sub-tree can be visualised using this method. The method is extendable to other types of modelling systems that allow attribute values to be associated with points, primitives or subtrees.

Fine tuning the appearance of strokes and therefore the final image can be achieved using a number of parameters. The number of parameters, however may introduce a level of confusion and obfuscation for the user. Therefore, it is important to consider the number and presentation of parameters. The system presented in this thesis was designed to reduce the number of parameters in relation to the previous work presented in [FJW+05] and [JWS06], while facilitating varying levels of user control over the image.

1.4 Organization

In Chapter 2 a summary of the **previous work** related to implicit surface modelling (ISM), Non-Photorealistic Rendering (NPR), particle and agent based systems, and

Multi-Agent Systems (MAS) are presented.

The **MAS management scheme** is presented in Chapter 3 with an introduction to the concepts of user requests and agent tasks. The mechanism of creating priority ratings for these tasks and assigning them to agents is also discussed. The structure and organisation of the blackboard data repository, which is an abstraction of the implicit object is also included here.

In Chapter 4 the agents, known as **smarticles**, are introduced. They are described in terms of performing the tasks assigned by the manager. This includes a description of their environment, the situations they can be in, the actions they can perform, their internal data values, their decision function and teams they can be associated with[2].

Chapter 5 describes the effect of **behaviours** on the steering direction of smarticles and consequently the paths they take. Smarticles have behaviours to process polygons, steering and flocking to create paths for sampling or strokes, and finding and tracing feature outlines.

In Chapter 6 the **rendering** techniques used to position strokes from smarticle's paths and colour polygons are presented. Strokes can be long or short, dotted lines or stipples and can be subject to density and lighting calculations. Polygons can be coloured relative to a single point, feature outline or user placed line, or can be related to a specific primitive of the model.

Chapter 7 brings together the techniques presented in previous chapters to illustrate the various **results** achieved using a number of different models. This chapter also includes a **discussion** and future work.

[2]"This is the sort of English up with which I will not put." Winston Churchill.

Chapter 2

Related Work

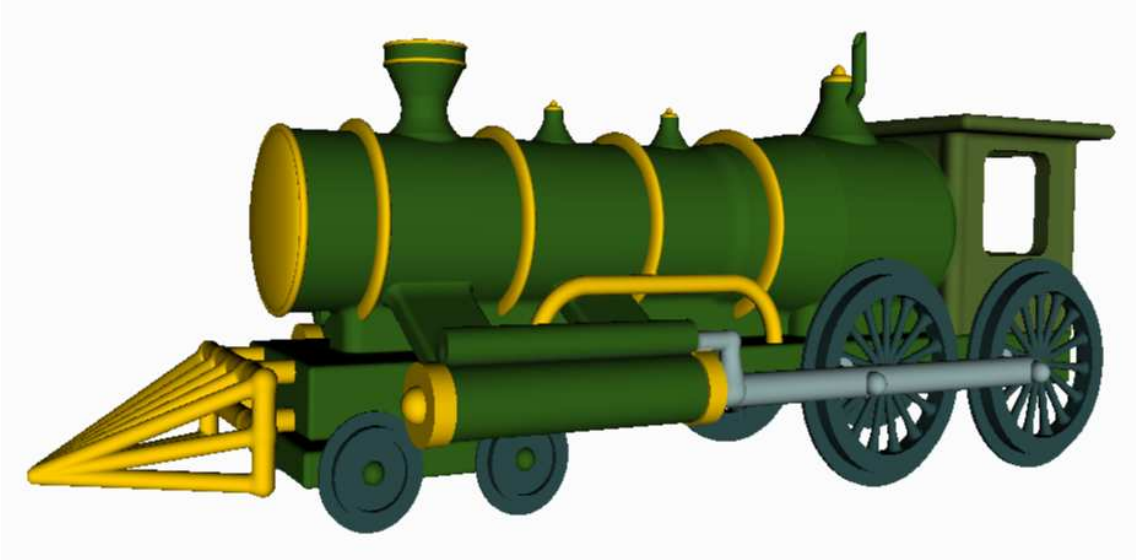

Figure 2.1: Train Modelled using the BlobTree Implicit Surface Modelling system by Callum Galbraith.

2.1 Introduction

This chapter contains a summary of the previous work[1] that is related to the research presented in this thesis. Section 2.2 introduces the field of Implicit Surface Modelling (ISM) with a description of data structures commonly used in modelling and rendering stages and includes some of the methods previously used to find and render discontinuities of algebraic surfaces. Particle and agent-based systems are outlined

[1] "... a group of French monks and their involvement with sheep rearing helped to give the modern world the computer..." James Burke, Connections.

in Section 2.4, with the focus on: sampling and rendering of implicit surfaces using the Witkin-Heckbert model, and systems that use smart particles or agents. An extension to particle systems to include steering and flocking behaviours is outlined in Section 2.5. Finally in Section 2.6 Multi-Agent systems are defined and the concepts relevant to the research presented in this thesis are described.

2.2 Implicit Surface Modelling (ISM)

In 1973 A. Ricci published the seminal paper "A constructive geometry for computer graphics" [Ric73]. In this paper Ricci presented a general approach to the definition of complex 3D objects constructed from simpler primitives using boolean set operations.

Solid modelling operations are dependant on volume information, they are not practical using surface information alone. Surface based techniques generally use a collection of patches with connectivity information to maintain continuity and avoid unwanted discontinuities or creases. For example, when using surface based techniques to model a human hand, it is important to avoid obvious joins or unwanted artifacts between the fingers [BBB+97]. Contrast this to using solid modelling techniques where each finger can be blended with the hand to create a smoothly branching surface.

2.2.1 Implicit Surfaces

In [Sab68] Sabin introduced using *Potential Surfaces*, which use scalar functions, to describe machined components for arbitrary but fair surfaces (with only essential

features). The potential surface is described as the locus of points at which a scalar function of position p is zero $f(p) = 0$. Essentially, this is an offset surface that uses a base surface to calculate the potential scalar function value.

Whilst visualising electron density fields in the early 1980's, Jim Blinn presented modelling with iso-surfaces in a scalar field [Bli82]. An iso-surface is a two-manifold that exists in three-dimensional space. Blinn introduced this as *Blobby Modelling*. Later techniques were *Metaballs* [NHK$^+$85], which used ray tracing, and *Soft Objects* [GWW86] introduced a method to create a polygonisation.

All of the above methods use algebraic distance rather than Euclidean, and each method uses a different distribution function[2] to describe the field. *Implicit Surfaces* is the current most commonly used general term for such modelling techniques and is the one used in this thesis.

The functions used in implicit modelling define a volume around a point in space. The extent or range of the volume is defined by the Radius of influence, R. The iso-surface is a contour found at a certain distance from the originating point, where the field function results in some particular value.

Definition: An *Implicit surface* [BBB$^+$97] S, is composed of the set of points derived from a field function $f(p)$ as follows:

$$S = \{p \in \Re^3 : f(p) = iso\} \tag{2.1}$$

where iso is a constant value defining the iso-surface and $p = (x, y, z)$. $f(p)$ is known as the *Implicit Function* (or the field function).

[2]The terms potential, distribution and field function are used interchangeably in the literature.

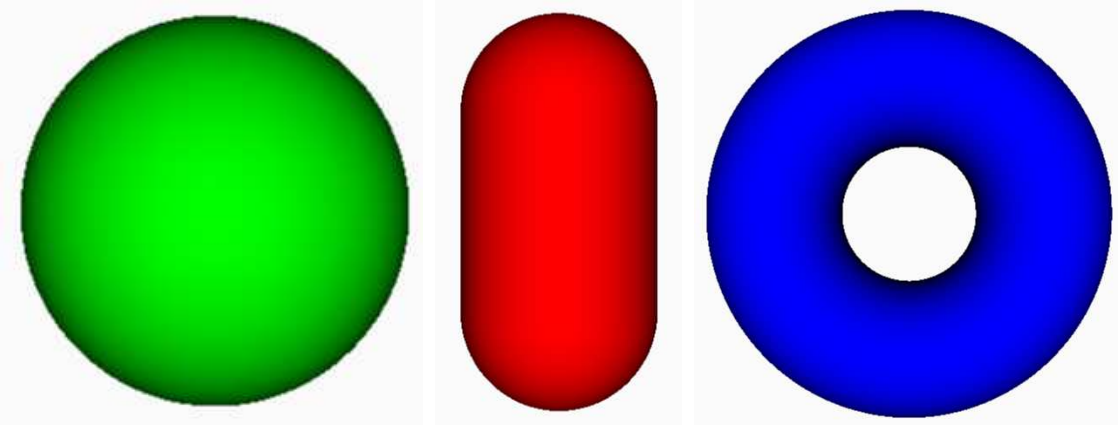

Figure 2.2: Objects generated from skeletal elements: a point produces a sphere, a line a cylinder and a circle skeleton produces a torus.

The primitives introduced by Blinn [Bli82] are defined by a single point that creates a spherical surface. Other primitives can be generated using different base shapes, or skeletons [BBB+97]. For example a line skeleton creates a cylinder and a circle skeleton creates a torus, see Figure 2.2. In this research, a primitive has a maximum function value of one exactly at the centre and smoothly drops to zero at R.

When discussing volume, it should be noted that there is the volume defined by the implicit function which extends to the Radius of influence, and there is also the volume associated with the visualised object, which is enclosed by the iso-surface. These volumes are termed *function related volume* and *primitive related volume* for clarity [Gal05].

The volume associated with a function is described as the set of points where the field function value is greater than zero.

$$V^f = \{p \in \Re^3 : f(p) > 0\} \tag{2.2}$$

Outside of the radius of influence R the field function value is zero.

The volume associated with a primitive is described as the set of points where the field function value is equal to (on the surface) and greater than (inside) the iso-surface value *iso*.

$$V^p = \{p \in \Re^3 : f(p) \geq iso\} \tag{2.3}$$

2.2.2 Blending

Complex implicit models can be constructed by combining several simple primitives using smooth blending or set operations. This differs from parametric methods where the surface is shaped by interaction with control points [BBB+97]. With Implicit Surface Modelling (ISM) the volume of the object is important, not simply the surface. Information describing the volume of all contributing objects are used in the calculations of the final shape.

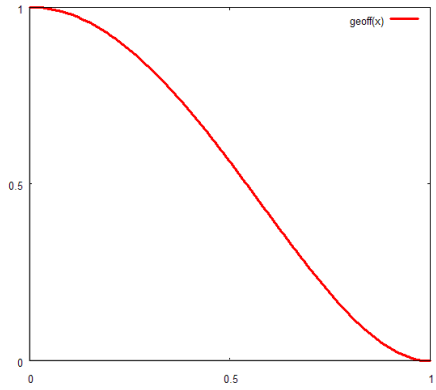

Figure 2.3: A symmetric function.

Methods of combining primitives are broadly characterised in this thesis as blends resulting in smooth surfaces or blends resulting in surfaces with discontinuities. Smooth blending involves summation of field function values from contributing prim-

itives. Such smooth blends characterise implicit surfaces and identify them as being particularly well suited to modelling complex organic objects. Discontinuities in the field, and therefore on the surface, result from blends such as boolean set operations which are used in Constructive Solid Geometry (CSG), see below. Adding CSG to the set of blend operations allows the construction of more complex objects, such as those common in CAD applications [BBB+97].

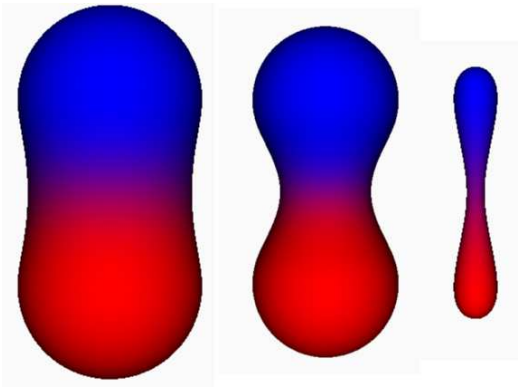

Figure 2.4: When two implicit spheres are close together the Radii of influence overlap and their iso-surfaces is a blend of the two. This iso-surface value is (from left to right) **Left:** $iso = 0.1$, **Centre:** $iso = 0.5$, **Right:** $iso = 0.9$

The field function and consequently the value of iso used to define primitives is chosen to reflect certain blending characteristics, for example $iso = 0.5$ in a symmetric function (Figure 2.3) [GWW86] . If the value of iso is changed different blending characteristics are observed, see Figure 2.4.

Smooth Blends

Smooth blending of several primitives involves the summation of their field function values [BBB+97]:

$$F(p) = \sum_{i=0}^{n-1} f_i(p) \tag{2.4}$$

Where n is the number of primitives and p is the point in space being queried.

Two key advantages of modelling with implicit surfaces relate to blending characteristics. The first is the ease with which smooth or discontinuous blends of implicit models are achieved using simple functional compositions (relating to the component primitives used in the object construction $f_i(p)$). The second advantage is derived from the fact that Equation 2.1 is easily modified to define an object volume: instead of an equality to denote a point being on the surface, we use $f(\mathbf{x}) < iso$ to identify a point as being outside of the surface or $f(\mathbf{x}) > iso$ for inside.

Constructive Solid Geometry

Smooth blending properties of implicit surfaces mean they are well suited to creating complex, organic and branching objects. CSG is used to create more defined, or sharp, features.

In [Ric73] the concepts known as CSG were introduced. These are straightforward methods of using simple geometric primitives in boolean set operations to create more complex 3D objects. Sharp features resulting from CSG operations identify discontinuities in the resulting field. A bottom up binary tree evaluation is detailed in Ricci's paper, where leaf nodes are the geometric primitives and non-terminal nodes are boolean set-theoretic operations known as union and intersection, see Figure 2.5.

Early systems combining implicit surface modelling and CSG were developed by

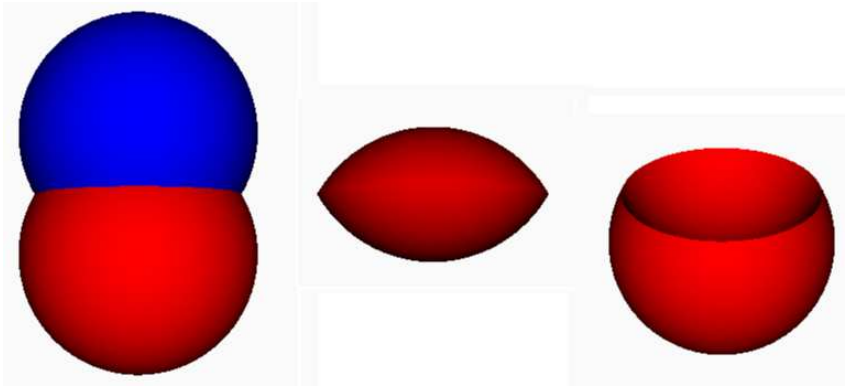

Figure 2.5: **(a)** The union surface of two point primitives. **(b)** The resulting intersection. **(c)** Difference.

Wyvill and Trotman [WT90], Wyvill and van Overveld [WvO95] and Pasko [PS94]. Note that in the original Ricci paper there was no mention of the difference operator. This can, however, be expressed as the union of one object with the negation of the second [WGG99], see Fig 2.5. The union, intersection and difference operators are expressed as:

$$f^U(P) = max(f_1(P), ..., f_n(P)) \tag{2.5}$$

$$f^I(P) = min(f_1(P), ..., f_n(P)) \tag{2.6}$$

$$f^D(P) = max(f_1(P), (1 - f_2(P))) \tag{2.7}$$

2.2.3 Data Structures

Efficient data structures are essential for handling complex implicit models. A complex model can involve a large number of primitives and operations, for example the train model in Fig 2.1 has more than 700 nodes in its tree. There are, therefore,

a number of important considerations when developing effective and efficient data structures. In the context of this thesis, they are grouped into two distinct categories. The first is the representation of the model itself: how to organise and evaluate the primitives and the blend operations. Secondly, the actual visualisation of the model considering rendering strategies.

Modelling

Implicit Surface Modelling (ISM) systems often employ a tree-like data structure for the model representation [WGG99], [PA02], [SWG05], [AGCA06].

In [WGG99], Wyvill et al. propose a structure called the BlobTree, which is a hierarchical tree-based method that allows arbitrary compositions of models using blending, warping and boolean operations. A primitive is described as the shape defined by a skeletal element with its associated volume defined by the scalar field function. Primitives are represented as leaf nodes, while combination operations are non-terminal nodes. The BlobTree was extended for animating implicit surfaces in [GNW02].

In [PA02], Pasko describes a function representation modelling system known as FRep. Pasko developed a unified representation of definitions of primitives using continuous real-valued field functions (as above) and including discrete-value volume information. FRep also extends the method to include multi-dimensional shapes (for example: as could be used in animation with time as the fourth dimension).

In [AGCA06], the authors present another hybrid modelling framework for constructing complex objects from simpler primitives called the HybridTree. The HybridTree is an extended CSG tree that assembles primitives, triangle meshes and

point set models in a coherent manner. Allègre et al. exploit the relative abilities of implicit models and polygonal meshes for editing operations. They use the implicit formulation for combining shapes with smooth blending and CSG operations; and the polygonal mesh for Free Form Deformation (FFD) and fast visualisation.

Calculating the field function value for any point in space involves traversing the defining tree (and its subtrees). Field function evaluations are often the main bottleneck of ISM systems. Speeding up this stage can be achieved using hierarchical spatial caching [SWG05]. Hierarchical spatial caching introduces the concept of a caching node inserted into the tree structure. This node stores the exact field function value at nodes of a voxel grid and uses tri-linear and tri-quadratic reconstruction filters to locally approximate the field value of a sub tree.

Another method used to speed up this stage uses a discrete voxel grid representation [GH91, FCG00]. The values at voxel corners are incremented according to any operations performed. This inherently changes the nature of the implicit object from being a continuous representation to a discrete one. The voxel grid in both systems are used for both modelling and rendering.

Rendering

A secondary data structure is often required to render an implicit surface. Generally, such a data structure is independent of the one used to store the description of the model [BBB+97]. The most accurate method of rendering an implicit surface is to use ray tracing [KB89]. The object's defining functions are used in ray intersection tests and a data structure stores the values of resulting pixels for the final image. Particle systems can also be used for visualisation of the surface, they query the

model definition and use their own data structure for the rendering [WH94].

Voxels are the most commonly used elements in data structures for polygonisation of implicit surfaces [GWW86, Blo88, SWG05, SIJ+07]. They are used to subdivide space and calculate polygons to approximate the part of the surface they contain. There are four types of voxels, which are identifiable by field function values. There are:

1. **voxels that contain surface intersections**, which are identified as containing points that are evaluated as being:

   - inside the surface ($f(p) > iso$)
   - on the surface ($f(p) = iso$)
   - outside the surface($f(p) < iso$)

   these are known as *surface voxels*.

2. **voxels completely inside the surface**, therefore, every point will have $f(p) > iso$.

3. **voxels outside the surface** but have some field function contribution as they are still within the Radius of influence R, these have points with $f(p) < iso$ and $f(p) >= 0$.

4. **external voxels** that are wholly outside the surface *and* the Radius of influence R, all points in such voxels have $f(p) = 0$.

In most circumstances, voxels containing part of the surface are the most useful. These voxels are easily identified by examining field function values at the corners

(assuming the voxel dimensions are appropriate for sampling details on the surface).

There are two general methods for finding the surface and associating it with voxels. The first converges on the surface and thus creates the voxels. The second uses the abstract notion of a 3D voxel grid where the surface voxels have to be identified.

Convergence. In [Blo88], a hierarchical space partitioning is used to converge upon the object. The initial voxel bounds the surface and converges using subdivision. Voxels are recursively subdivided when they contain a surface intersection, until a specified depth of recursion has been reached.

Continuation algorithm. The continuation algorithm works by first identifying a voxel in the grid that contains a surface intersection, also known as a *seed voxel*. The rest of the surface is tracked by examining neighbouring voxels across shared edges [GWW86].

Surface Voxels. When using voxels for polygonisation, there are a comparatively small number that contain surface intersections. It is unnecessary to implement the voxel grid as a three-dimensional data structure when it is only the surface voxels that are required. In [Blo88] (convergence method), Bloomenthal uses an octree to store only the voxels with surface intersections. In [GWW86] (continuation method) a hash table is used to store the surface voxels. The hash address is calculated from the integer value index of the voxel, i.e. the vertex of lowest (x, y, z) or left back bottom.

Surface Polygonisation. Surface voxels are examined, and polygons are generated according to the pattern of corners identified to be either inside or outside the surface [GWW86], [Blo88]. Surface intersection points must be found for each edge

that has one vertex identified as inside and the other outside the surface. Where speed is more important than accuracy, linear interpolation can be used. Linear interpolation, however, can result in unreliable approximations because the variation in the field function is not linear. Although, as noted in [GWW86], if the grid has a high enough resolution the approximation is a reasonable one. Where accuracy is more important than speed, the intersection point should be found using the exact field function value. The pattern of intersections are then matched to a polygon generating algorithm or a pattern look up table [GWW86].

Sampling Considerations. Finding and illustrating features or small scale details on a surface is dependent on the level of sampling. Surface details will only be identified where the sampling level is small enough to identify the feature. Guaranteeing feature identification requires additional information about the surface being rendered, Kalra and Barr discuss sampling for implicit surfaces in [KB89]. The Nyquist theorem states that the maximum frequency of the signal must be known in order to define a sampling rate that is greater than or equal to twice this value [Wei]. Lipschitz constants also require information about the maximum rate of change of the function [Wei]. Therefore, in order to guarantee that all details are identified, the distance between samples should be related to the size of the smallest surface feature. Although too many samples will have a considerable time overhead [vOW04]. A balance between speed and accuracy is important. As mentioned above, surface voxels are identified where at least one corner (and at most seven) is inside the surface. If the voxel grid is not fine enough, a detail may be overlooked, for example in Figure 2.6, the resolution of the voxel is 10 units cubed and the radius of a feature is 4 units, the feature is identified only if it is located at a corner of the voxel.

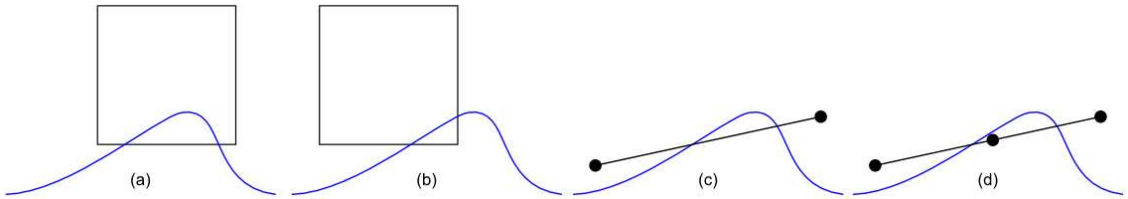

Figure 2.6: **(a)** If the resolution of the voxel grid is larger than a detail on the surface it can be missed. **(b)** If the detail is coincident with a corner of the voxel it will be found. **(c)** Is a step size between two samples is too large a detail can be missed. **(d)** A smaller step size will find the feature.

2.2.4 A Black-Box view of the Implicit Function

A distinction should be made between the requirements of the function to generate an object and those to visualise it. The functional description is necessary for definition of the shape and blending of component primitives. Visualising the result, however, has less strict functional requirements. The functional description may be necessary for techniques such as such as Precise Contact Modelling (PCM) [BBB+97], or (in some cases) finding the intersections of surfaces [SZ89]. Alternatively, for rendering, the defining implicit function (and related operations) can be treated as a black-box. This means that the visualisation method is general and can apply to any implicit model definition where the gradient is computable everywhere (although it does not have to be continuous, [FJW+05]). For a black-box method the only requirement is that given a point in space the field function value and the gradient are provided. This information can be used to calculate curvature, principal directions of curvature [Ebe99], contour directions and many other useful properties of the surface and its related volume.

2.2.5 Finding and Displaying Discontinuities and Singularities

In [RRS97], Rosch et al present two methods for interactive visualisation of implicit surfaces with singularities. The methods are physically based sampling using particle systems and polygonisation with mesh improvements. Simple implicit surfaces are used along with the Kumar family of algebraic surfaces to illustrate these methods. The particle system is based on the Witkin-Heckbert (WH) model [WH94], which tends to repel particles more at singularities. In order to achieve a more uniform distribution of particles a number of heuristics are proposed. The first method modifies the energy function to allow the particles to intersect one another at a singularity. The second method applies an adaptive repulsion radius for each particle. The adaptive factor is chosen depending on either the curvature of the surface or the proximity to a singularity.

The algorithm presented in [SZ89] deals with scan line display of algebraic surfaces. It can correctly display any singularities (of any complexity) by using the underlying definition of the surface. The algorithm finds the intersection curve of two surfaces by using the resultant and discriminant of the implicit equations that describe the model. The resultant eliminates the variable z from the equations and is a two dimensional curve. The discriminant is found from the resultant of the surface and its polar. The polar surface is defined as the surface that results from the dot product of the eye vector with the normal of a point on the surface. For example the silhouette is defined where this dot product equals zero. This polar surface is a plane that intersects the shape and is perpendicular to the view direction.

An early system, [Wei66] uses the underlying mathematical surface definition

to render silhouettes and curves of intersection. It also introduces orthographic projection as a theorem for visibility.

The stochastic sampling method (SSM) proposed in [TSYK01] can detect intersections of implicit surfaces using the surface definitions. This technique generates uniform sampling points on a complex implicit surface. It also uses a generalised stochastic differential equation where the solutions can satisfy plural constraint conditions simultaneously. This generalization is the means by which the intersections can be viewed.

In [JBS03] a point based rendering technique is used in conjunction with iterated function systems (IFS) to render both parametric and implicit surfaces with non-manifold features and singularities. Only evaluations of the surface functions and gradients are used to render an image. For arbitrary implicit surfaces, rather than using iterative solutions of differential equations, points are distributed directly over the surface. It is a brute force method and is therefore slow.

In [BPK98] Belyaev et. al derive formulas to detect creases on a surface in implicit form. They state an intuitive definition of a surface crease as "*curves on a surface along which the surface bends sharply*". These ridges and ravines are mathematically described as the extrema of the principal directions of curvature along the curvature lines. Ridges are the local positive maxima and ravines are the local negative minima. They rely on the defining functions for derivation of the crease.

2.3 Pen and Ink Rendering

Pen-and-ink has been explored significantly in NPR, including 2D image-based techniques [SWHS97, DHvOS00, SGS05] (which calculate the final rendering using a 2D viewing plane), and 3D object space methods, which either apply stroke textures [WS94, PHWF01] or procedurally generate strokes on surfaces [DS00, HZ00, SFWS03, PFS03]. Relatively few papers have been devoted to pen-and-ink for implicit surfaces [BH98, Elb98, Ric73].

In one of the earliest works on implicit surfaces, Ricci used line drawings to visualize constructive solid models [Ric73]. His system traces lines following the intersection of planar cross-sections of the surface and also performs Hidden Line Removal (HLR).

Bremer and Hughes presented an approach for rendering implicit surfaces in a pen-and-ink style [BH98]. This method finds silhouettes by first intersecting random rays with the implicit model, then uses numerical integration to step from successful intersections toward the silhouette. Once the silhouette is found, it is traced using a second numerical integration. This method uses ray intersection tests for positioning short interior (general surface) strokes at successful intersection points and for performing HLR. There are few options for stylization and the dependence on random rays does not support specific interior stroke placement. Furthermore, the silhouette tracing algorithm requires a smooth surface. This method is not guaranteed for more complex implicit surfaces which contain abrupt blends and constructive solid geometry (CSG). The techniques presented by Bremer and Hughes are the inspiration for the approach presented in [FJW+05] and the silhouette tracing method in this

research.

Elber presented a particle based method for rendering implicit surfaces in a pen-and-ink style [Elb98]. In this method long flowing strokes are traced and stylized along the interior of the implicit model or along silhouette lines. Tracing of the strokes uses either the principal directions of curvature, or lines of constant angle between the surface normal and the view vector.

Akleman described a method for creating painterly renderings of implicit surfaces [Akl98a, Akl98b]. In this technique particles trace the surface leaving a paint stroke in their path. Paint is simulated by considering the interaction of a paint brush with paper. This method creates expressive effects by allowing the particles to move off the surface. Although this gives an impression of a line drawing, it offers no pen-and-ink stylization. In [Akl98a] Akleman describes implicit models built using CSG operations. The rendering is directly related to the set-operations that define the underlying surface. This means that the junction is not traced but computed directly from the model definition.

In [GIHL00] the authors present a psychologically based argument for using principal directions of curvature to perceive shape. This pen-and-ink style rendering technique can be applied to polygonal models or 3D volume data.

2.3.1 The Non-PhotoRealistic BlobTree (NPR-BT

In [FJW+05] we attempt to improve the quality of rendering results by implementing pen-and-ink style rendering for complex implicit surfaces. We draw on point relaxation concepts of Pastor et al. [PFS03] which use particles attached to a polygonal mesh and a point hierarchy to create frame-coherent stipples. We also use

shape-measure/threshold concepts to create and stylize strokes [WS94, PHWF01, SFWS03]. Concepts of curvature-oriented hatching [RKS00] are also used. The method of identifying and tracing feature outlines due to discontinuities or abrupt blends is known as the Dual Tracking Algorithm. This is a novel technique and has been adapted in this research to be achieved using smarticle behaviours (see Chapter 5.7.1, this was my main contribution to this publication).

To create short directional strokes over the surface, a two-step process is used. First, the system calculates the stroke direction. Then, it can then optionally curve the stroke based on the amount of surface curvature in the stroke direction.

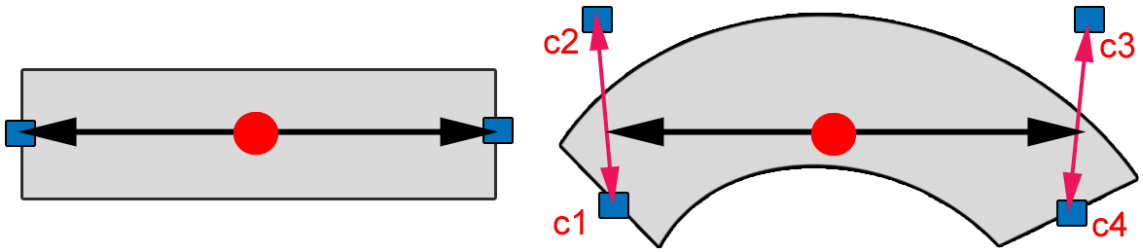

Figure 2.7: Left: To create a straight stroke, a line is created between two endpoints projected out from the particle's position. Right: A curved stroke is created by extruding 4 points from each particle applying them to a Bzier function.

To create strokes, our system projects two points g_0 and g_1 from each particle's position in the stroke direction as illustrated in Figure 2.7, left. The length of this line can be controlled by surface curvature or by the user. When this length is kept short, the system can produce reasonable output. Unfortunately, straight strokes with longer lengths do not adequately describe the surface. To allow for longer strokes, we curve strokes to follow the surface. The amount that a stroke bends is directly related to the curvature so that straight strokes are drawn in areas of little curvature, while curved strokes appear in convex/concave areas (Figure 2.7). To

accomplish this, the system draws a Bézier curve with four *control-points*, c_1, c_2, c_3, c_4 defined as follows:

$$c_1 = g_0 - (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \tag{2.8}$$

$$c_4 = g_1 - (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \tag{2.9}$$

$$c_2 = g_0 + (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \tag{2.10}$$

$$c_3 = g_1 + (\nabla f(\mathbf{x}) * scl * \gamma(\mathbf{x}_i)) \tag{2.11}$$

where *scl* is the distance between g_0 and g_1 multiplied by a user selected scalar and $\gamma(\mathbf{x})$ is the surface curvature.

2.4 Particle and Agent Based Systems

Particles used in physically based modelling [WBK97] can be used as individual entities, such as elements of fireworks or sparks [Ree83], or as a single "monolith" when considered as a group, for example cloth, fire, smoke or liquids [Tur91, SF95, Sta99]. They can be susceptible to physical forces such as gravity and viscous drag, or have constraints applied to them, such as the springs that connect particles to create a cloth simulation. Furthermore, particles can also be used for collision detection and particle dynamics are also used as a basis for rigid body simulations. Particle systems are, therefore, versatile and can be used in a variety of manners, they can be based on anything from physical laws to artificial rules and anything in between.

A particle is an object that can be described as having a few basic attributes such as a position and a velocity, they can have associated behaviours such as responding

to forces. These behaviours are not, however, calculated by the particle, they are applied to each in turn by the particle system. The particle system examines all the forces that are exerted upon a particle and references each particle's data to calculate the effect of these forces. This results in new values, such as the new position and velocity.

In "Depicting Fire and Other Gaseous Phenomena" [SF95] Stam uses particle systems and allows the user to define velocity fields that are used to visualise flow fields. Stam uses a similar technique to update a velocity field that is dynamically animated to simulate fluid flows in "Stable Fluids" [Sta99].

In "Generating textures on arbitrary surfaces using reaction-diffusion" [Tur91], Turk presents a method for creating a regularly spaced mesh that is used to calculate reaction-diffusion texture patters. This method involves placing points randomly on a model's surface and then distributes them using a repulsion force. A polygonisation is used to determine initial positions for points. The relaxation method used for distribution uses a repulsion radius determined by the area of the surface and the number of points. Points are always corrected to lie on the plane defined by the local polygon

In "Reaction-diffusion textures" [WK91], Witkin and Kass develop a method that adapts traditional reaction-diffusion approaches to allow anisotropic and spatially non-uniform diffusion. This also includes multiple competing directions of diffusion and as such was a precursor for Witkin's particle system research [WH94] (presented in the following Section).

A particle system is used for dynamic path planning for groups of characters in "Continuum Crowds" [TCP06]. A per-particle energy minimisation approach using

continuum dynamics is used for individual characters' motion. Forces that act upon each character are calculated assuming that each character has a dynamic or motion based goal.

In "Organic labyrinths and mazes" [PS06] Pedersen and Singh use the point distribution techniques developed by Turk in [Tur91]. Labyrinths and mazes are represented as configurations of a set of piecewise linear curves that evolve using an iterative algorithm similar to Turk's repulsion method.

2.4.1 Witkin-Heckbert Style Particle Systems for Implicit Surfaces

Witkin and Heckbert [WH94] presented one of the seminal particle system frameworks for sampling and controlling implicit surfaces. They were motivated to improve the time taken to sample and render an implicit surface; to provide a visualisation method that would work in real-time.

Many other particle systems for implicit surfaces are based on the principles described in [WH94]. There are two main forces, or constraints, that dictate the movement of the particles. These are an attractor force that constrains the particle to the implicit surface, and a repulsion force that spaces the particles from one another. The repulsion force acts locally to distribute the particles across the surface. The repulsion force is also used to control density, birth and death of particles. Attraction and repulsion are fundamental forces that are used in many later systems.

The attractor force $\mathbf{A}$ represents the orthogonal projection of the particle's position onto the surface's tangent plane:

$$\mathbf{A} = \frac{\Delta F^i(p) \cdot \mathbf{v}^i}{\Delta F^i p \cdot \Delta F^i(p)} F_p^i \tag{2.12}$$

where $\mathbf{v}^i$ is the velocity of particle i; and ΔF_p^i is the gradient evaluated at position p^i.

The repulsion force is a Gaussian energy function based on the distance between particles. It defines a particle's energy E as:

$$\mathbf{E^{ij}} = \alpha \; exp \; \left(-\frac{|\mathbf{u^{ij}}|^2}{2(\rho^i)^2} \right) \tag{2.13}$$

where $\mathbf{u^{ij}} = p^i - p^j$ is the vector between particles, α is a global repulsion amplitude parameter and ρ is the local repulsion radius or standard deviation of the Gaussian function.

The repulsion force is therefore negatively proportional to the gradient of energy with respect to a particle's position:

$$\mathbf{R^i} = (\rho^i)^2 \sum_{j=1}^{n} \left(\frac{\mathbf{u^{ij}}}{(\rho^i)^2} \mathbf{E^{ij}} + \frac{\mathbf{u^{ij}}}{(\rho^j)^2} \mathbf{E^{ji}} \right) \tag{2.14}$$

The repulsion force has been modified in later research to take account of properties of the surface, such as curvature.

In [RRS97], Rosch et. al use a Witkin Heckbert (WH) based particle system to illustrate implicit surfaces with singularities, cusp points and self intersections. Their aim was to achieve a uniform sampling that would ultimately be used for polygonisation. The authors modify the WH repulsion energy force (Equation 2.14) to avoid particle interference or repulsion near a self intersection, and to to take into account surface curvature and proximity to a singularity.

Another direct modification of the WH model is described in [CA97]. Crossno et al modify the repulsion force to be based on local curvature of the surface. Curvature

is also used in birth and death calculations, more particles appear in regions of high curvature and less in low curvature. Ultimately the particles are used to create a polygonal representation.

In [SH05] Su and Hart use a WH style particle system and extend it to use behaviours, attributes and shaders to sense, extract and render surface information. This method incorporates the techniques from [RRS97] and [CA97]. The definition of *behaviours* in [SH05] refers to the forces applied to the particles, which includes the attractor and repulsion forces, the integration step, and the birth and death of particles as in the WH model. The main difference from the WH approach is the method of decomposition of these forces. There are also behaviours that cause particles to move towards silhouettes or singularities. The singularity behaviour is based on the assumption that the gradient vanishes at a singularity and therefore moves particles towards it when the magnitude falls below a preset threshold or when the direction changes faster than the preset threshold. The silhouette adhesion behaviour uses gradient descent search to move particles in the opposite direction of the gradient of the squared magnitude of the silhouette function.

The Non-PhotoRealistic BlobTree (NPR-BT)

In [FJW+05] a WH based particle system was used to render complex hierarchical skeletal implicit models in several pen-and-ink styles. The system uses the BlobTree and is called the Non-PhotoRealistic BlobTree (NPR-BT). Particles identify interesting areas of the surface and features using local surface properties. Interesting areas include silhouette outlines and lines following local shape features, such as those caused by CSG junctions. Local surface properties include examining the curvature

of the surface at each particle's position. The mean curvature γ_i is computed at each particle position as follows:

$$\delta_i = 1 - \frac{\gamma_i}{\gamma_{max}} \tag{2.15}$$

where γ_{max} is the maximum of all γ_i, with γ_i defined by the Hessian $h(\mathbf{x})$ of the potential function. γ_i is defined as :

$$\gamma_i = \frac{1}{2}\left(\frac{trace(h(\mathbf{x}_i))}{\|\nabla f(\mathbf{x}_i)\|} - \frac{\nabla f(\mathbf{x}_i) h(\mathbf{x}_i) \nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)^3\|}\right) \tag{2.16}$$

.

The Hessian is the 3 by 3 matrix of second partial derivatives of the potential function, which are calculated numerically from the gradient. Additionally, *trace()* defines the vector created from the principal diagonal of the matrix. The Hessian is used to trace the principal directions of curvature [Ebe99].

The silhouette curve for an implicit surface is defined in terms of the view vector $\bar{V}$, which for a point in space $\mathbf{x}$ is defined as:

$$\bar{V} = \frac{(\mathbf{x} - \mathbf{x}_{eye})}{\|(\mathbf{x} - \mathbf{x}_{eye})\|} \tag{2.17}$$

A complete continuous silhouette curve $c(t)$ exists on an implicit surface S if $c(t) \in S$ and the tangent plane to the surface at $c(t)$ contains $\bar{V}$ where $\mathbf{x} = c(t)$ for all t [BH98]. These conditions translate to:

$$f(c(t)) = iso, \forall t \tag{2.18}$$

$$\nabla f(c(t)) \bullet \bar{V} = 0, \forall t \tag{2.19}$$

To trace silhouettes, the numerical integration from Bremer and Hughes [BH98] is used. Unlike their technique, tracing begins by identifying particles P_i that lie near

the silhouette using the following inequality:

$$|\bar{V} \bullet \nabla f(\mathbf{x}_i)| < \kappa_{threshold} \tag{2.20}$$

where $\kappa_{threshold}$ is user defined (we use 0.05).

Once silhouette particles have been identified, Equation. 2.18 and 2.19 are used to to step along the silhouette using a predictor/corrector method. For a given point $\mathbf{x}$ on the silhouette (obtained from the list of silhouette particles), an estimate of the silhouette direction $\bar{U}$ is defined as follows:

$$\bar{U} = \kappa_{step} \frac{\nabla f(\mathbf{x}) \times \bar{V}}{\|\nabla f(\mathbf{x}) \times \bar{V}\|} \tag{2.21}$$

where κ_{step} is a user defined step size. This estimate is used to step along the silhouette to a new point $\mathbf{x}' = \mathbf{x} + \bar{U}$. To ensure that the stroke trace remains on track, two correction vectors are used. The first correction $U_{surface}$, corrects in the direction of the surface as follows:

$$\bar{U}_{surface} = \kappa_{surface} \nabla f(\mathbf{x}')(iso - f(\mathbf{x}')) \tag{2.22}$$

where $\kappa_{surface}$ is a user defined scalar value (we use $\kappa_{surface} = 0.5$). This equation uses the field value to scale the size of the surface corrector, as in:

$$\bar{A}_i = (f(\mathbf{x}_i) - iso) \nabla f(\mathbf{x}_i). \tag{2.23}$$

The second correction $\bar{U}_{sil}$, is used to ensure that integration follows the silhouette:

$$\bar{U}_{sil} = \kappa_{sil}(\bar{U} \times \nabla f(\mathbf{x}')(\bar{V} \bullet \nabla f(\mathbf{x}'))) \tag{2.24}$$

where κ_{sil} is a user defined scalar value. As $\mathbf{x}'$ moves off the silhouette, the magnitude of $\bar{U}_{sil}$ increases, thus pulling the stroke back toward silhouette. A new silhouette

point $\mathbf{x}''$ is then defined as follows:

$$\mathbf{x}'' = \mathbf{x}' + \bar{U}_{sil} + \bar{U}_{surface} \tag{2.25}$$

Subsequent iterations repeat this procedure (Equations 2.21 to 2.25).

As points are calculated, their positions are saved into a fine spatial grid, denoting that a silhouette has been found in that position. This is used to prevent multiple silhouettes from being traced, and to determine when a silhouette loops.

Linking Silhouettes: Silhouette tracing will generate looping and non-looping silhouettes. Looping silhouettes are identified when the silhouette trace enters a grid cell where (1) its starting point exists and where (2) the angle between the direction towards the starting point and the $\bar{U}$ is less than a threshold. A non-looping silhouette is created when $\bar{U}_{sil}$ enters a cell where another silhouette already exists or when $\bar{U}_{sil}$ enters an area of an abrupt blend or a CSG junction. In both of these cases, the system returns to the original silhouette point and traces in the other direction. After all silhouettes have been traced, we link the endpoints from any two silhouette chains that are within a certain distance of one other. If there are still incomplete silhouettes after this step, the K step sizes are lowered, and tracing is attempted again.

We trace short **interior strokes** directly at particle positions. Selection of the particles that are used to draw strokes is accomplished by evaluating the following particle measures: depth, angle from the silhouette, mean curvature and lighting. A brief summary of lighting and silhouette angle measures is now provided.

Lighting: We simulate point lights in our system. These are evaluated as:

$$light_i = \frac{\mathbf{x}_i - \mathbf{light}}{\|\mathbf{x}_i - \mathbf{light}\|} \bullet \frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|} \tag{2.26}$$

where **light** is the light position.

Angle from Silhouette: The angle from the silhouette creates a measure which is the same as a point-light at the eye position in the scene. It is calculated as the angle between the gradient to the view direction:

$$silAngle_i = \frac{\mathbf{x}_i - \mathbf{x}_{eye}}{\|\mathbf{x}_i - \mathbf{x}_{eye}\|} \bullet \frac{\nabla f(\mathbf{x}_i)}{\|\nabla f(\mathbf{x}_i)\|} \tag{2.27}$$

Notice that to gain efficiency, we only calculate particle measures for front-facing particles as back-facing particles will not be visible when rendered. A particle is back-facing if $(\mathbf{x}_i - \mathbf{x}_{eye}) \bullet \nabla f(\mathbf{x}_i) > 0$.

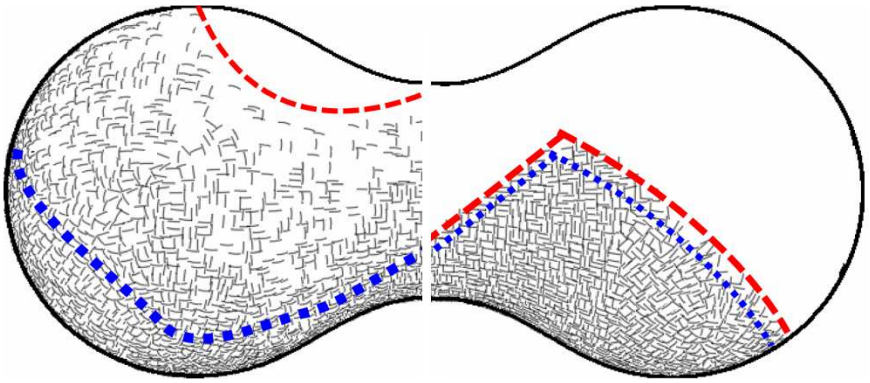

Figure 2.8: Comparing different values for **ut** and **lt**: the blue dotted line denotes *ut* and the red dotted line denotes **lt**. Left: a large difference between **ut** and **lt** creates a smooth blend between areas with strokes and those without. Right: close **ut** and **lt** values creates a sharper transition.

Applying Measures to Select Particles: Our system allows the user to select two values, an upper-threshold *ut* and a lower threshold *lt*, which define shape-measure areas (Figure 2.8). Any particle whose *measure*, *et* is less than *lt* is removed and any particle whose measure is greater than *ut* is used to create a stroke. Particles between these two values are blended for inclusion (Figure 2.8). The system must blend particles in a way that appears to be random, but does not create flickering

as the scene is redrawn. To do this, we use the particle's index i into the particle set P to create:

$$et = \frac{i}{n} * (ut - lt) + lt \tag{2.28}$$

where n is the number of particles. To create a blended effect between ut and lt, particles are included for rendering when the shape measure for particle i is greater than lt.

We provide two methods for implicit model hidden line removal: an efficient approximation using surfels [PZvBG00] with z-buffer occlusion and a more computationally expensive exact approach using raytracing [BH98]. Only the surfel technique is presented here, see [FJW+05] for more details.

2.4.2 Smart Particle and Agent Based Systems

The motivation for the smart particles or *sparts* introduced by Pang and Smith in [PS93] is to provide a programmable tool for visualisation of scientific data. They are used in the context of virtual spray cans, which are user controlled and are filled with different types of sparts that perform different tasks. Spray rendering can render arbitrary data sets in various styles and allows progressive refinement of the image. Characteristics can be assigned to the sparts such as surface seeking, volume penetrating, flow tracking, and meta-sparts. They allow the user to visualise a data set by interacting with it using a collection of specific-task sparts, or spray cans. The sparts system provides certain structured behaviours, such as surface seeking, but aims to provide the user with a mechanism of defining behaviours for themselves. This requires an understanding of the underlying data and the potential behaviours that could be created to visualise it. For example, sparts can be used to highlight

data values themselves or relationships between data values, "*a spart may be defined to manifest itself only if the wind speed is between 12 and 30 knots*". Sparts can be thought of as small spherical balls or light emitting points. The individual sparts may, however, be rendered in a variety of styles and shapes, for example each spart can be shaped like a leaf and the group of sparts is "thrown" into the wind to visualise the direction of the wind.

The techniques presented in [FJW+05] identified shortcomings in terms of the time taken to distribute particles across a surface to illustrate feature outlines. In [JWS06] the system was enhanced to use a data structure that represented the Blob-Tree object. Analysis of this data structure identified potential locations of surface features and areas that had been under sampled. Smarticles presented in [JWS06] used steering and flocking behaviours to seek out interesting areas of the surface and achieve variations in stroke appearance. The basic management system was the first step and proof of concept in creating the management system and smarticles presented in this thesis.

A method that operates in image space is [SOD04]. User collaboration with an artificial ant colony progressively transforms photographs into stylised pictures. The user is not responsible for positioning strokes, the ants create the strokes in response to navigation and stroke parameters set by the user. Ants are autonomous agents that use only local information and can navigate edges, fill and hatch regions, and smudge regions of the image. There is no communication or cooperative concepts applied to them. They have a shared memory in terms of the image to be created and user interaction sets navigation and mark (stroke) parameters.

In [MDC04] an agent based system is presented to help artists express ideas. The

semi-autonomous agents represent graphical elements that are combined through rules provided by an artist working with an algorithm. The authors aim to find a balance between artistic expression and algorithmic support. Coalition forming is used to create larger elements that an artist can manipulate: *"Instead of dealing with masses of tiny elements the artist can deal with a smaller number of coalitions (groups) of elements."* Creating an image involves training the agents about the conditions under which they should form coalitions.

RenderBots [SGS05] are autonomous agents that represent one stroke in one style. There are a number of different styles of strokes and so a number of RenderBots are used to create an image. The RenderBots can draw edges, shade an image using hatching and stippling, and use styles such as mosaic or painting. The main contribution is a unified approach to stoke based rendering, where many styles can be created using the same framework. The agent space consists of a source image and possibly additional G-buffers. These are layered to create a RenderBot's *universe*. This is an image space method with the focus on a 2D environment, rather than 3D object space. Although 3D information may be available in terms of information from G-buffers, it is not made available to the RenderBots when generating an image.

2.5 Flocking and Steering Behaviours

Craig Reynolds introduced flocking [Rey87] and steering [Rey99] behaviours to give autonomous characters more life-like actions. The behaviours are independent of the visual representation of the characters and describe their interaction with each other and their environment.

Steering behaviours are applied at the individual level and flocking behaviours are applied to groups. Steering applies to how individuals navigate their environment. Whereas flocking creates effects that are dependent on other individuals in the flock. Individuals are referred to as *boids* in Reynold's work.

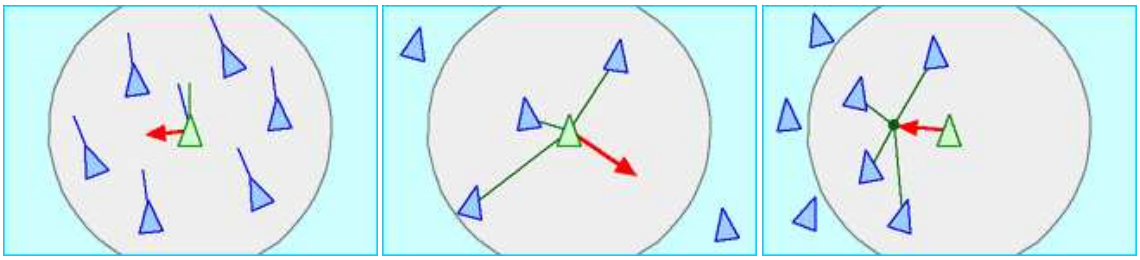

Figure 2.9: The red arrows indicate the steering directions central particles should apply to adhere to the relevant flocking behaviour. **Left** Alignment. **Centre** Separation. **Right** Cohesion. *(Images courtesy of Craig Reynolds.)*

2.5.1 Flocking Behaviours

In [Rey87], Reynolds presents a generalisation of a particle system that is used to simulate group behaviour, i.e. flocks, herds or schools. The individual members of the group are modelled as above, with the group behaviour emerging from the application of straightforward interaction rules. There are three main flocking behaviours: collision avoidance (or separation), velocity matching (or cohesion) and flock centring (or alignment), see Figure 2.9. In [Rey87] the flocking behaviours and related neighbourhoods are used to simulate flocks of birds, shoals of fish or swarms of insects.

All three flocking behaviours rely on finding local flockmates (others in the same flock). The neighbourhood may include a few flockmates, or can be large enough to include all of the members of the flock.

Alignment with local flockmates causes all relevant members of the flock to follow similar direction headings. An individual's velocity is progressively altered to match those of local flockmates. The alignment steering vector is calculated from the difference between the current individual's velocity and the average velocity of neighbours:

$$\mathbf{a} = \frac{\mathbf{v^a}}{\mid \mathbf{v^a} \mid} - \mathbf{v} \tag{2.29}$$

where $\mathbf{v}$ is the current velocity and

$$\mathbf{v^a} = \sum_{i=0}^{n} \mathbf{v^i} \tag{2.30}$$

$\mathbf{v^i}$ refers to a neighbour's velocity and n is the number of flockmates in the neighbourhood.

Separation from local flockmates eliminates overcrowding and can be used for collision avoidance. A repulsive steering force is calculated that moves an individual away from local neighbouring flockmates. The separation force is calculated from the relative positions of neighbours, which is normalised and weighted. The weights use the difference in position between the character and the flockmate being considered:

$$\mathbf{s} = \sum_{i=0}^{n} \left(\frac{1}{\mid p - p^i \mid^2} * \frac{p - p^i}{\mid p - p^i \mid} \right) \tag{2.31}$$

where p is the current individual's position and p^i is a neighbouring flockmate's position.

Cohesion steers the characters into a group. The steering force is calculated as the difference between the vehicle's current position and the average position of the local flockmates.

$$\mathbf{c} = p^a - p \tag{2.32}$$

where p^a is the average position of local flockmates: $p^a = \Sigma_{i=0}^{n} p^i$.

The three flocking behaviours define three forces that are combined to create a single steering (flocking) force. Each force is given a weight that assigns how much influence that behaviour will have on the final steering direction. The values of the weights are dependant on the desired effect.

$$\mathbf{f_i} = \omega_s \ \mathbf{f_i^s} + \omega_a \ \mathbf{f_i^a} + \omega_c \ \mathbf{f_i^c} \tag{2.33}$$

where ω_s, ω_a and ω_c are the weights for separation alignment and cohesion respectively.

2.5.2 Steering Behaviours

In [Rey99], the term *behaviour* refers to *"the improvisational and life-like actions of an autonomous character"*; in contrast to a scripted set of actions. Combination of steering behaviours determine the total steering force, which can generate complex behavioural patterns. The method of combination is typically linear, and is achieved by associating weights with each of the steering direction vectors. A boid is a simple point mass approximation with a position, mass, velocity and orientation. Boids have a visual representation that requires an orientable local coordinate frame. This basis is in terms of its local origin and local forward, side and up vectors. This frame is incrementally rotated to maintain coherency in orientation throughout the simulation.

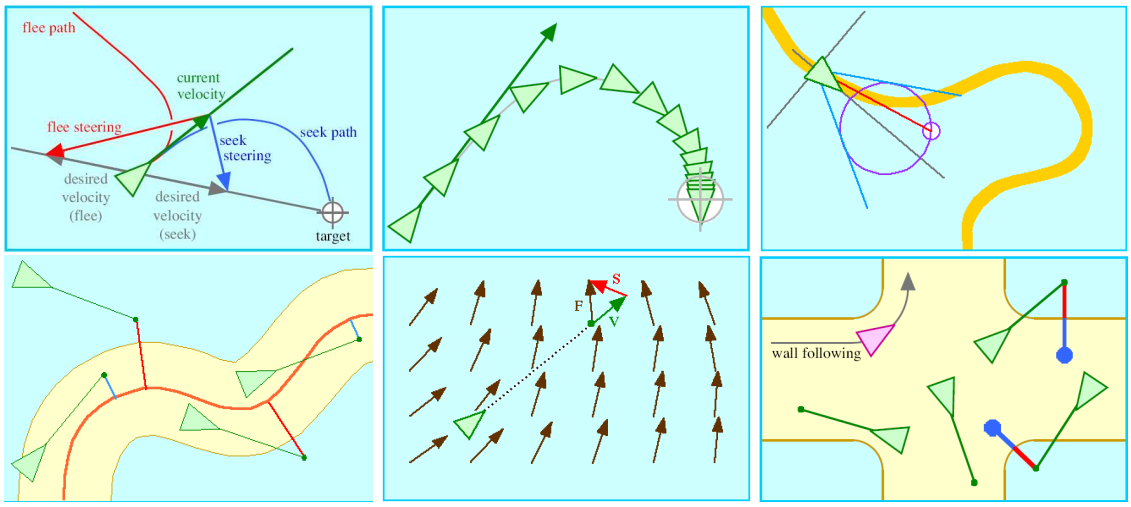

Figure 2.10: Behaviours. **(Top Row:) (Left)** Seek. **(Centre)** Arrival. **(Right)** Wander. **(Bottom Row:) (Left)** Path following. **(Centre)** Flow field following. **(Right)** Containment. (*Images courtesy of Craig Reynolds*)

The **seek** behaviour steers the boid towards a specified position in space, see Figure 2.10 (**Top Left**). The boid's velocity is incrementally adjusted at each step so that it is aligned towards the target. The desired velocity is therefore in the direction from the boid to the target.

The **arrival** behaviour is identical to seek until the boid reaches within a certain distance of its target, see Figure 2.10 (**Top Centre**). As the boid approaches its target it slows down and finally stops when it has reached it.

The **wander** behaviour (Figure 2.10**(Top Right)**) calculates a random vector that is used with some constraints to ensure the path is not too erratic. A sphere is positioned close to the front of the vehicle with the radius equal to the maximum wander *strength* (Figure 2.10 **(Top Right)**: large circle). The steering force is constrained to the surface of this sphere. The new steering force is calculated using a random displacement with a maximum length equal to the wander *rate* (Figure

2.10 **(Top Right)**: small circle). This displacement is added to the previous steering force before being constrained to the surface of the larger sphere. In this way, the steering direction will not be in the opposite direction regardless of the orientation of the randomly generated displacement, which means the path is less erratic.

Path following (Figure 2.10 **(Bottom Left)** describes an overall behaviour that provides a pre-determined steering path for an individual. The path is not defined as rigidly as rails for a train, it is more akin to a corridor or footpath. This enables a certain amount of freedom of movement without the path being too prescriptive and restricted.

Flow field following (Figure 2.10 **(Bottom Centre)**) is a subset of **Path following** where the path is defined using a flow field vector that is dependant on either user requirements or environmental specifications.

Containment (Figure 2.10 **(Bottom Right)**) is another sub set of path following and is used to constrain an individual within a particular region. Containment generally describes a steer-to-avoid strategy that alters the steering force of the particle to ensure that the path does not take the particle outwith the specified containment region.

2.6 Multi Agent Systems (MAS)

A Multi Agent System (MAS) usually consists of a group of computing elements (known as agents) in an environment. Agents have a method of collaboration that defines them as being part of a system rather than unrelated individuals. One of the main goals of a MAS is synergy [Den99]: although a single agent may be capable

of autonomous problem-solving behaviour, a collection of agents is often recognised as having a better chance of achieving a higher quality solution. The collection of agents is used to examine a problem space and work towards finding possible solutions. Collaboration (co-operation or competition) between agents should increase the effectiveness, efficiency and quality of a solution. An important consideration, however, is that the level of collaboration does not create prohibitively large volumes of communication.

2.6.1 An agent

There are many definitions of an agent and a MAS, it is a much debated topic. Wooldridge's [Woo02] definition is:

> "An *Agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives."

An accepted generalisation is that an agent is defined by a 4-tuple consisting of situations, actions, internal data and decision functions [Den99]. The situations are those in which an agent may find itself. The internal data is the set of possible values the data areas can have. The set of actions are those that the agent can perform. And finally, the decision function is what connects the situation and relevant action. This definition of an agent is therefore:

Definition: Agent

$Ag = (Sit, Act, Dat, f_{Ag})$

where:

Ag: the agent.

Sit: the set of situations Ag can be in.

Act: the set of actions Ag can perform.

Dat: Ag's internal data (set of all possible values).

f_{Ag}: Ag's decision function (Sit * Data = Act).

The decision function can be represented using a decision network framework, which is a generalisation of Bayesian networks. Probability theory is used along with graph theory to represent uncertain knowledge in a probabilistic graphical model [YT07].

The notion of an intelligent agent is as difficult to define as the concept of intelligence itself. In [WJ95] the authors list a number of capabilities that may be expected of an intelligent agent, these are: **reactivity**, where agents should be able to perceive and respond to changes in their environment; **pro-activeness** in terms of using some initiative for goal directed behaviour; and an agent's **social ability** in terms of communicating with other agents (and potentially users).

2.6.2 A Multi-Agent System

As with most human systems, fundamental properties of a MAS are organisation and collaboration [Den99]. Organisation is required to take a group from a set of individuals to a system of agents. The organisational structure identifies the roles of agents in the system. An organisation requires a communication protocol and a reporting structure for the group of agents. Collaboration can be in the form of co-operation or competition.

2.6.3 Organisation

It has been stated that one of the main goals of an MAS is synergy; where the whole is greater than the sum of its parts [Den99]. A group of agents can be used to solve problems that may be beyond the ability of one individual [DLC89]. The group may also provide a better, faster or more appropriate solution. An organisational structure is, therefore, required for the collection of agents to function as a system. This organisation will deal with task identification and assignment as well as the method of agent collaboration. As mentioned above, agent collaboration can be in the form of competition or co-operation [Den99], [Woo02]. Whatever the collaboration style of an agent, the ultimate aim of the system is to find a solution.

In order to find a solution a control strategy is required. A control strategy addresses the problem of deciding upon appropriate actions to perform in particular situations during the tasks of the problem solving process [HR85]. The control strategy is, therefore, related to an agent's decision function.

There are a number of tasks associated with creating an organisation of co-operative problem solving agents. First, the tasks and subtasks should be well defined, created and distributed. Knowing how to organise (and identify or categorise) both task and subtasks is very important and can be a very difficult process. Next, agents are assigned to work on and solve the tasks. And finally, the achievements of the agents should be collated, examined and communicated to other (relevant) agents. This process is repeated until a satisfying solution to the problem is achieved [Den99]. Note, however, after a few system iterations, the definition of the problem or task may be quite different from the original description.

2.6.4 Communication

In order for agents to co-operate, they must communicate. Communication between agents should be designed in such a way as to maximise useful information sharing whilst minimising unnecessary data flow [Den99]. Communication can, therefore, be described as an action that is undertaken to influence other agents [Woo02].

In order to have effective communication between any two entities, be they agents in a system or human beings in situations, a method of communication must be agreed upon. This method includes both the language to be used and the form of the communication itself.

Two basic communication protocols used in MAS are through message passing or via shared memory. Message passing is one-to-one communication between agents. This communication usually has a clearly defined and agreed upon structure which is turn based and forms a dialogue between the agents involved [Wer88]. Shared memory, alternatively, is a common area, or database, where both reading and writing are permitted. This is also known as a *blackboard* [HR85]. An agent can choose to ignore it or act upon any information being communicated to it.

In a traditional blackboard system a central control unit known as a *scheduler* is an integral element. The scheduler creates control strategies which dictate the state, availability and accessibility of agents and information, and also compiles a strategy to solve the problem at hand [HR85]. In [Cra93], Craig introduces a new interpretation of the blackboard metaphor where control is more distributed. The blackboard is still used as a central repository of information, but the agents themselves decide which information to make available. The agents, therefore, maintain

control of their information, state, history, status, strategy and also have knowledge of other agents. Agents are, however, required to maintain communication with the blackboard regarding agreed upon topics. The blackboard can then inform other agents of state changes or any other information that is relevant.

2.6.5 Co-operative Concepts

Durfee et al [DLC89] identify four general goals of co-operation. These are:

1. Increase the task completion rate through parallelism.

2. Increase the set or scope of achievable tasks by sharing resources (information, expertise, physical devices, etc.).

3. Increase the likelihood of completing tasks (reliably) by undertaking duplicate tasks, possibly using different methods to perform those tasks.

4. Decrease the interference between tasks by avoiding harmful interactions.

The authors note that the importance of each of these goals is very much dependent on the system being created.

Organisations often employ several concepts for co-operative problem solving. Amalgamation of these concepts allow different tasks to be assigned to the most relevant concept. Two examples of concepts are co-operation by making (selected) information available and master-slave relationships.

Sharing useful information (via a blackboard or message passing) can help agents achieve better or faster solutions. Deciding what is "useful", however, can be a time

consuming task. It is important to consider that many agents engaging in high volumes of communication can create a great deal confusion. It is extremely desirable therefore to properly filter this information and reduce the volume of communication. In [DF99] filters, or *referees* are used to help clarify the situation and the communication between agents.

2.6.6 The Vigo Multi-Agent System

Vigo is a framework for simulating and visualizing three dimensional scenes [Bur]. It is being developed as a research tool at the ESD Group.

Vigo was originally created as a platform for modelling dynamic systems of interacting agents. As agents move through three dimensional space they perceive ("see") objects and other agents in their neighbourhood. Agents decide how to act based on their own internal data and information from the environment, i.e. what they perceive.

Vigo combines all of the physical components of the simulation into a scene. The scene then assumes responsibility for agent dynamics in the simulation. A developer derives agents from the vigo::space::Node class and tailors them to the requirements of the simulation. The scene is used to control the simulation and therefore the agents.

2.6.7 Applied MAS techniques

"Artificial Fishes: physics, locomotion, perception, behaviour" presented by [TT94] uses Reynold's flocking and steering behaviours in a MAS that simulates fish in an environment that simulates certain natural ecosystems. One of the aims of the

framework was to provide realistic appearance, movement and behaviour of individual fish as well as model complex group behaviours. The fish have behaviours that rely on their perception of their dynamic environment. They have three mental states: hunger, libido and fear. An intention generator checks sensory information and mental states to decide on an appropriate intention. Control is then passed to a behaviour routine that will satisfy the intention. Any relevant perceptual information is also passed to the behaviour routine.

In "Improv: a system for scripting interactive actors in virtual worlds" [PG96], Perlin and Goldberg describe actors in a manner akin to agents that have real-time behaviour based actions in terms of their individual physical movements and their group behaviours. Actors, or agents, directly represent the character being displayed and animated. Characters have a behaviour engine that maintains an internal model of the character (similar to Internal Data Values (IDV)) and makes decisions about the appropriate action to be performed in particular situations (related to the Decision Function). Co-ordination of multiple actors involves using a blackboard. When actors interact, they can alter the contents of each others IDV. The blackboard is distributed along with each actor if multi-processing or network implementations are used.

In "Planning cooperative motion for distributed mobile agents" [FS96], autonomous distributed mobile robots co-operative to to create a map of an environment. Robots are of different sizes and therefore have access to different areas. They communicate with one another to resolve problems, such as requesting a smaller robot access a small region.

Yu and Terzopoulos present an agent based framework for advanced behavioural

animation in virtual humans in "A decision network framework for the behavioral animation of virtual humans" [YT07]. The paper deals with decision making that selects appropriate actions for the agents' perceived environment. The actions are chosen even though their interpretation of the environment may be imprecise, which results in more realistic simulations. The authors use a combination of techniques from decision networks, probability and graph theories for complex behaviour modelling and intelligent action selection.

In "Autonomous Robots and Agents" [MG07] techniques are presented for collaborating autonomous agents used for techniques such as motion planning and map making. Methods presented include techniques for autonomous agents to create, for example, cognitive maps of environments, and special trajectories for mobile robots.

Chapter 3

The Management Scheme

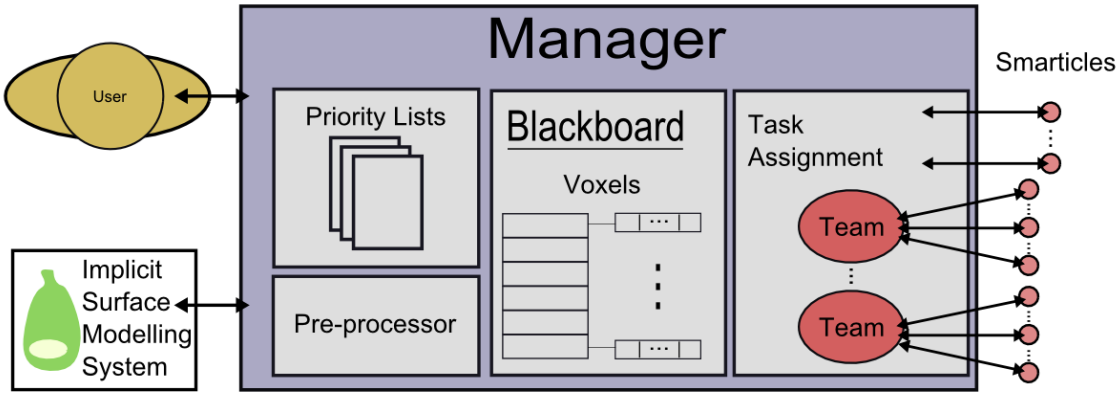

Figure 3.1: The manager: initialises the data structure, which is the repository known as a blackboard; queries the implicit surface modelling system on behalf of the smarticles (organised into teams); and interfaces with the user to create the priority lists.

Previous approaches to pen-and-ink rendering of implicit surfaces are, in general, based on particle systems. Particles are distributed across a surface and can identify feature outlines when they are close to them. Particles know only their position, velocity, acceleration, and other attributes (in some systems). Smarticles are agents (in a MAS) that have been developed from traditional particles (WH style) to include behaviours, which allow them to semi-autonomously explore their environment. Smarticles are described in full in the following chapter (Chapter 4). They are used to overcome some of the shortcomings associated with the feature proximity requirement of previous systems. The development of smarticles alone, however, does not address all of the problems identified with previous particle-based

approaches.

A particle system is described as a "monolith" [WBK97], which means that despite having many component parts, it acts as a single entity designed to perform a particular task.

In the context of implicit surfaces many of the previous approaches used particles to visualise a surface by adhering to it and distributing across it. The particle system calculates and assigns attribute values, such as position and velocity, for each individual particle. Particles do not have any control over their locomotion or changes in attribute values, they simply store the data that represents them. Particles also do not have any autonomy or inter-particle communication or co-operation. All calculations involving the particles are done at the particle system level.

Smarticles develop WH style particles to include agent-based techniques. The management system interprets user requirements and creates tasks that are assigned to the smarticles. For example, if the user requests that a silhouette outline is traced, the manager initialises a smarticle and instructs it to perform the task. Other tasks include feature outline tracing, exploring a region and creating paths for strokes, see Section 3.2 for details.

The management system was developed to complete tasks in a flexible manner. The order of completion is not predefined and consistent, it changes every time the system is used in accordance with the user's specifications. Although there is a certain set of tasks the manager can assign and smarticles can perform, the ways and means of completing these tasks vary according to the user's requirements. To avoid ambiguity, user specifications and requirements are called *requests* and smarticles are

assigned *Tasks* to fulfil these requests.

The user (e.g. an illustrator) has an overall view (either mental or physical) of how an object should be represented when creating an image. The illustrator may choose certain areas of the image for focus of attention or to be rendered in a higher (or lower) Level Of Detail (LOD). The illustration being created is viewed as a single object that is created piecewise, where the whole is greater than the sum of its parts.

The manager was therefore designed to represent this holistic view. The manager assigns task and collects and co-ordinates the results of smarticle explorations. The management and smarticle scheme combines the benefits of local, distributed components, such as agents or particles, with an overall view of the object. For this reason, wholly autonomous agents are not appropriate, and the manager (necessarily) acts as a central controller.

Manager and smarticle development are modelled on the rough-to-detail process many illustrators use. In early stages of an illustration, rough outlines are used to suggest the shape and form of an object. The illustrator decides which parts of the model should be drawn with more (and how much) detail. Smarticles were, therefore, developed as a tool rather than being wholly automatic. Smarticles can automate tasks where relevant, such as feature finding, but the overall direction remains with the user. The manager interprets the user's requests. The user can direct the smarticles to any region of the surface and specify *what* should be performed with the manager taking care of the details of *how* it will be performed.

The contributions of this chapter are the method of organising agents to explore an implicit model; the method of organising the object space to identify potential locations of features and outlines; interpreting user requests to create tasks; and

prioritising tasks and areas of the surface to identify features and trace the outlines.

3.1 Introduction

The concepts of *tasks* are central to the understanding of the working of the manager. Tasks include tracing feature outlines, positioning stroke samples, shading around an outline or a single primitive, exploring a region (identifying features), see Section 3.3. The manager interprets user requests to create a list of tasks, illustrated in Figure 3.2. Smarticles are initialised with relevant task information i.e. initial position, velocity, steering direction calculation method, maximum step length and termination criteria, before being instructed to perform their assigned tasks, (see Chapter 4 for a full description of smarticles). Upon completion of the task the voxel-based public data repository known as the *blackboard* and the task list are updated. The manager accesses the Implicit Surface Modelling (ISM) system on behalf of the smarticles for information about the model.

This chapter is organised as follows: first an overview of the management scheme framework is given, which is followed by an explanation of task identification from user requests, and subsequently the method of assigning tasks to smarticles. Task prioritisation is then described in terms of workload, workdone and the resulting priority ratings. Finally, the blackboard data structure, communication pipelines and initialisation are described.

Note, a smarticle's movement is described by its *path*, which is constructed from steps taken at each system iteration. The direction of the steps is dependant on the task, e.g. to explore a region a constrained random wander steering behaviour

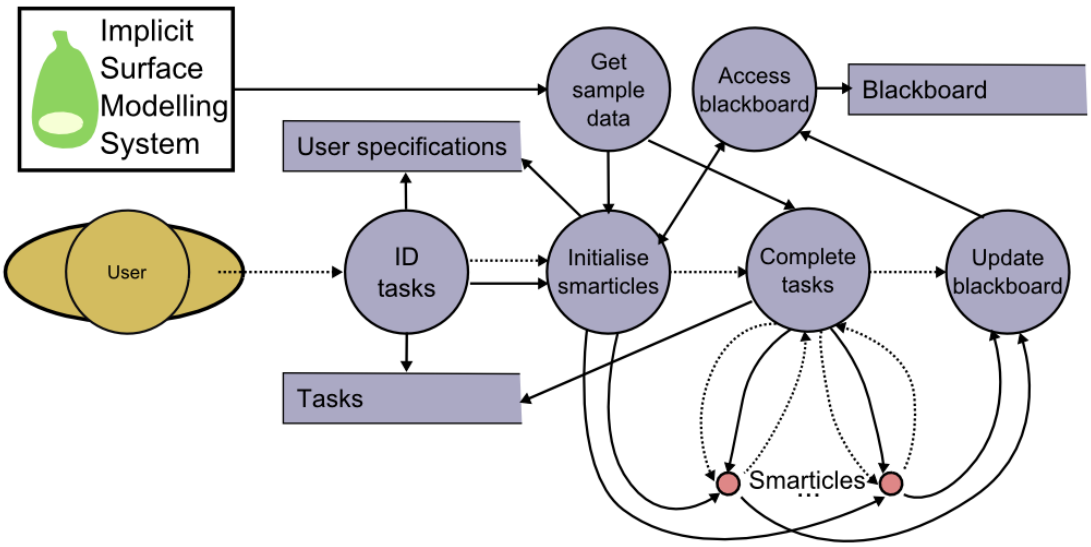

Figure 3.2: The manager interprets user requests to create tasks that are assigned for smarticles to complete. The dotted lines show the process flow and the solid lines show data flow.

(direction) is used with the length of the vector, or the size of the step, related to the level of detail. For a full discussion of smarticle's behaviours creating paths see Chapter 4.

3.2 Management Scheme Framework

The manager is central to the system operation and has several components (Figure 3.1). During the initialisation stage a low resolution polygonisation of the model is created, which provides initial data for the voxel based data structure known as the blackboard and smarticles, see Section 3.6. After system initialisation, the user can specify where and how they would like smarticle activity to be concentrated. The manager interprets the user's requests (Section 3.3) and references the blackboard to

generate a list of prioritised tasks. This list is then used to create smarticles, assign their relevant data values, initiate their performance of the tasks, and finally collect, collate and analyse the resulting data in the blackboard.

The prioritised task list is based on estimates of the *workload* (the perceived amount of work to be undertaken) and *workdone* (the amount of work completed). Note that both workload and workdone are estimates that are also dependent on the level of detail specified by the user.

An overview of a typical run of the system to identify and complete tasks is:

CompleteTasks()

$Initialise(blackboard)$ ▷ data structure (see Section 3.6.4)

$IdentifyTasks(user, blackboard)$ ▷ (see Section 3.3)

$Prioritise(tasks)$ ▷ (see Section 3.5)

$Assign(tasks, smarticles)$ ▷ (see Section 3.4)

$Initiate(smarticles)$ ▷ complete tasks (see Section 3.4.1)

$Collatesmarticledata$ ▷ (see Section 3.6)

3.3 User Requests and Task Identification

Table 3.1 describes the relationship between user requests and smarticle tasks. The user makes requests and the manager identifies the corresponding smarticle task. A single user request can fire multiple tasks. For example, if the user selects to create more detail in a region, the manager will use a team of smarticles to explore the region, which will also identify and trace any feature outlines found.

User requests	**Smarticle tasks**
Trace feature outlines	Trace feature outlines (Section 5.7)
Create a single line or stroke sample	Position a stroke sample (Sections 5.7 and 5.4)
Place stroke samples	Position stroke samples (Section 5.4 and 5.5)
Shade around an outline or line	Shade around an outline or line (Chapter 6.3.1)
Shade a single primitive	Shade a single primitive (Chapter 6.3.2)
Find features in a region	Explore a region (Section 5.4) Identify features or points on a feature outline (Section 5.6)

Table 3.1: User requests and their associated smarticle tasks.

User requests include identification of parts of the object or object space to define the region of interest:

- **point**: the user directly specifies the coordinates, (x, y, z), of a point
- **neighbourhood**: the number of voxels neighbouring the current voxel, e.g. 1: all voxels within $1 < i, j, k >$ (integer-value voxel dimension) of current voxel
- **region**: a region is an area of the surface (or volume) that is identified by a smarticles through examination of values at each path step point. A region is defined by the user as follows:
    - *The angle between the gradient at the first point on the path and the current point.* The region is identified to contain all path points where this gradient comparison is less than the specified angle. For example, in Figure 3.3 **(b)** the silhouette outline is drawn in the region where the angle between the gradient at the user specified point (blue square) and the

gradient at each path step point is less than 90 degrees. The extent of the region is reached and the path is terminated when this angle comparison exceeds the user-specified value.

- *The number of steps in the path.* This identifies a region that will have a size and shape dependent on the local properties of the surface. Smarticles take smaller step sizes in areas of high curvature, so a region delineated by 100 steps, for example, will be smaller in area on a curved part of the surface versus a relatively flat part.
- *If the curvature sign changes*, e.g. from concave to convex or vice versa. Using curvature change in this way smarticle paths can be limited to concave areas of a surface, for example smarticle paths could be limited to the concave region that represents the bite from the pear in Figure 3.7.

- **level of detail**: dictates the size of the steps the smarticle should take.

In the absence of specific user requests, the manager can, to a certain extent, automate task identification using statistics from system initialisation. In such a case task identification is restricted to tracing feature outlines.

In the remainder of this section, the manager's responsibilities in terms of task information is explained. The related smarticle operations are covered in Chapters 4 and 5.

3.3.1 Feature Outline Tracing

The user can request silhouettes and discontinuities tracing. Unless a region is specified, a feature outline is traced until it is no longer discernible or until it loops (the

end of the path meets the beginning). Not all feature outlines are guaranteed to be identified at any point in time, so the system can only trace those that are identified when the user made the selection. The types of feature outline identification are to trace:

1. **all feature outlines of all types to completion** (see Figure 3.3 **d**): this is accomplished by selecting "Trace all identified feature outlines" in the User Interface (UI). The manager automatically looks at all of the surface voxel data where feature outlines have been identified and systematically traces each to completion.

2. **all outlines of a particular type to completion** (e.g. silhouette or discontinuity): the UI allows the user to select tracing of all currently identified silhouettes, discontinuities or curvature change regions.

3. **all outlines in a region** (regardless of type): e.g. (Figure 3.3 (**b**)) the user identifies a region by specifying a 3D point in object space and termination criteria, as described above, in this example the angle was chosen to be 90 degrees. The region can be used to identify where tracing starts (user identified point) and stops (termination criteria), or where it starts with outlines being traced to completion (looping or lost) by not specifying any termination conditions.

4. **a single feature outline**: the user identifies a point in the object space (as above) and selects the "Trace closest feature outline" option. As above, the region constraint (termination criteria) can be set to limit the length of the

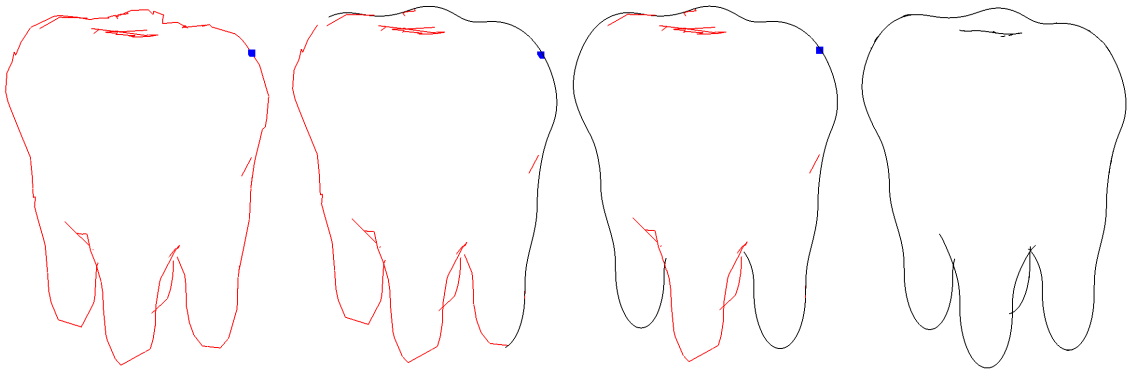

Figure 3.3: (**a**) The outlines from the initialisation of the tooth model in red. (**b**) The user identifies the silhouette (the blue square) and the region inside which it is traced. (**c**) The same silhouette has been traced to completion. (**d**) All silhouettes traced.

trace.

For full details of how smarticles trace specific feature outlines see Chapter 5.7.

3.3.2 Stroke samples

Creating stroke samples includes specification of where the stroke samples are to start, the direction they should take and where (or under what conditions) they should stop. Stroke samples are initiated:

- from existing outlines, such as silhouettes or discontinuities
- in particular regions (such as concave or convex areas)
- in a region around a user specified point
- at random positions on the surface, using polygon vertices with random offsets.

The user can identify outlines and regions using the UI. The user also selects the direction for stroke samples. Directions relate to the behaviours described in Chapter 5.

The user can identify a region on the surface by specifying the centre point and termination criteria. The user can specify further constraints, such as the density of stroke samples.

3.3.3 Shading

The user can request certain areas of the surface to be shaded as an indication of the importance of an outline or region. The shading is not intended to be a complete and accurate method, rather, it is used as a way of drawing attention to a region of the surface or indicating colour, shape or detail. Full details of rendering around an outline or a single primitive are covered in Chapter 6.

The user selects a feature outline using the UI (as above), specifies the voxel neighbourhood, and selects a colour using the colour selection tool. Upon selecting "Shade around feature outline" the shaded triangles are calculated and displayed, see Chapter 6.3.1 for full details.

Alternatively, where the user wishes to shade a particular primitive, they select it by positioning a point in the viewing space or selecting an attribute value, see Chapter 6.3.2 for full details. The colour is chosen in the same manner as above and the shading is calculated upon selecting the "Shade a single primitive" option.

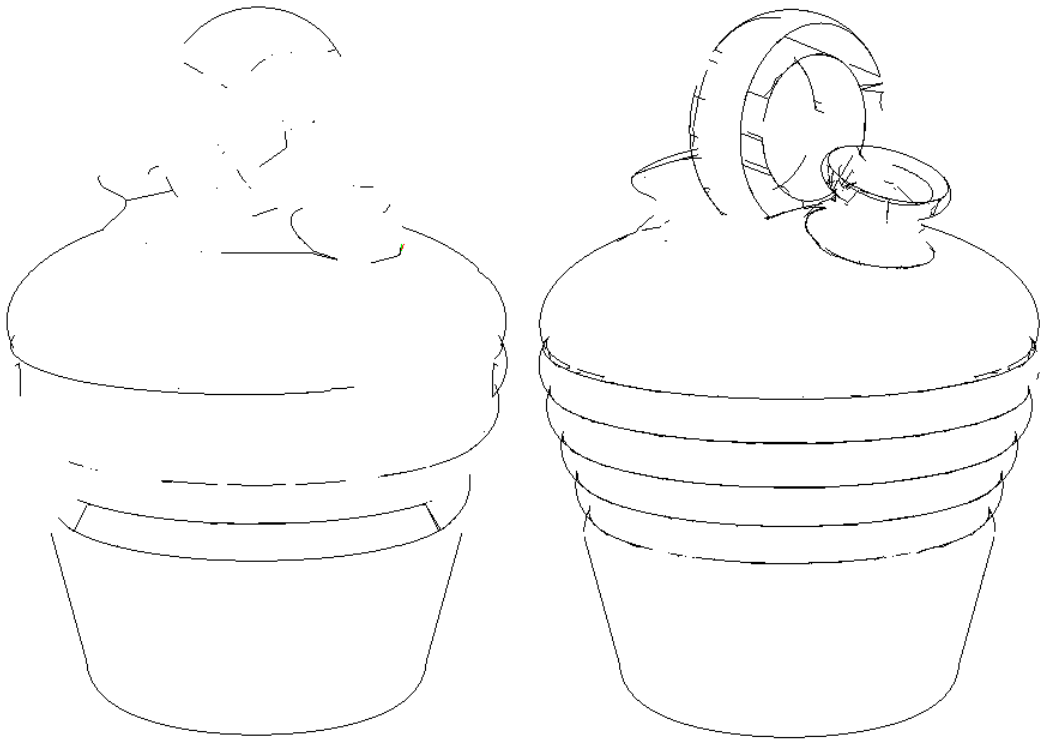

Figure 3.4: The jug on the left has outlines traced at a low level of detail (LOD = 1). Whereas the jug on the right has a higher level of detail for the tracing (LOD = 10).

3.3.4 Explore a region

Smarticles can be given tasks that relate to finding out more information about a region. This is usually done under one of two circumstances: where surface voxels have not been identified by the initial polygonisation, or where feature outlines have not yet been identified.

Sample voxel space

When a smarticle is given the task of sampling voxel space, its movements are constrained to lie within that voxel. *Sampling* is also the general term for evaluating

the field function value and the gradient at a particular point. Sampling voxel space can have two distinct purposes: ensuring voxel corner and edge data is evaluated and processed; and also to explore the space inside the voxel, potentially identifying a feature or surface detail.

Evaluation and processing of corners and edge-surface intersections usually occurs where a surface voxel has not been identified by the initial polygonisation (due to a small scale detail and large voxel size, see discussion on sampling in Section 2.2.3) or where a neighbourhood voxel is newly sampled and does not contain a part of the surface. This evaluation step provides the manager with the basic information to be used in calculation of statistics. Corner sampling is carried out as a first step when new voxels are identified.

A smarticle can steer (using wander, Section 5.4.1) inside a voxel and examine field function and gradient information at each of its steps. Relevant sample and curvature information is stored by the smarticles for the manager to calculate voxels' processing priorities. This can be carried out on any voxel that has previously had its corner information evaluated.

Sample a neighbourhood

Sampling a neighbourhood is similar to sampling a voxel. Different measurements are used, however, to describe the region (rather than using a voxel). The smarticle is given steering behaviours that direct it, and termination criteria to identify the region. The constraints are the same as above and those outlined for smarticle containment in Chapter 5.4.

3.4 Assigning Tasks

Assigning tasks to smarticles involves creating either an individual smarticle or a team, initialising Internal Data Values (IDV), initiating actual performance of the task, and collecting and storing any relevant information from the smarticles in the blackboard.

Smarticles were designed to complete tasks semi-autonomously with the aim of multi-threading the tasks where relevant. This would allow the smarticle to perform certain tasks independently and in parallel. The places where this would not be the case would be where they were constrained by density calculations or some other situation where a conflict in accessing the blackboard would arise. Multi-threading was not fully implemented, however, due to constraints with the MAS engine Vigo.

In this section, the life-cycle of a smarticle is explained, from the point of view of the manager and completing assigned tasks. Teams of smarticles are also introduced, before describing an example of assigning tasks from user requests.

3.4.1 The Life Cycle of a Smarticle.

A smarticle is created by the manager when it is needed, see Figure 3.5. It is given initial values depending on the task it should perform, for example if the smarticle is to trace a silhouette, the initial position and sample values will be provided by a point identified from the pre-processor to be on or close to a silhouette.

The following Algorithm outlines the managers method of initialising smarticles:

```
InitialiseSmarticles()
    WHILE tasks
```

$CreateTeam(numberSmarticles, task)$

FORALL smarticles $\triangleright$ in team

$SetInitialPosition()$

$SetInitialVelocity()$

$SetSteeringDirectionCalculationMethod()$

$SetMaximumSpeed()$

$SetTerminationCriteria()$

ENDFOR

ENDWHILE

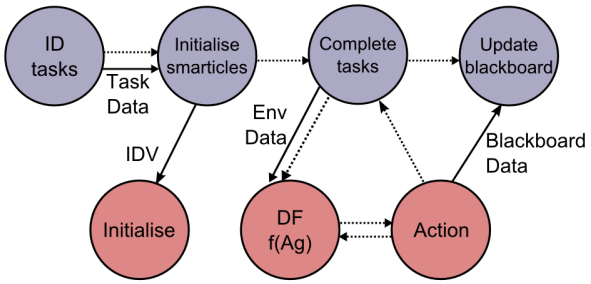

Figure 3.5: A smarticle is created by the manager in *initialise smarticle* and then performs actions to complete the assigned task. Upon completion of the task the smarticle reports any relevant information to the manager to update the Blackboard and informs the manager that the task has been completed. Blue circles represent manager processes and red ones represent smarticle processes. Dotted lines represent process flow, solid lines represent data.

The initial point is determined according to the request made by the user. It can be the point identified by the user, a point on a user selected outline, or a point identified to approximate a feature, from the initial polygonisation.

The initial velocity is also related to the specific task. It is calculated as part of the relevant initial steering direction. The steering direction is related to the behaviours (see Chapter 5 for full details and examples). Therefore, as the steering direction calculation method is set, the initial velocity is calculated using this method.

The maximum speed is directly related to the detail level requested by the user. This value is between 1 and 10 for user selection, which is normalised (0:1) for use in scaling the steering direction vector when calculating a smarticle's acceleration (see Chapter 4 for details).

The smarticle's termination criteria is set directly in accordance with the values specified by the user, as described above.

Once a smarticle has been assigned a task and its internal data is initialised, it performs the actions necessary to complete the task. The action it performs is dependant on the situation the smarticle is in. The situation is defined by information such as environmental and internal data (for details see Section 4.2).

When a smarticle has finished its assigned task it reports to the manager with any relevant information to be updated in the blackboard. The manager then places the smarticle on a free-smarticle list. Members of the list can be re-assigned to tasks where and when relevant, rather than having to create a new one each time. The free smarticles list was created in anticipation of multi-threading and parallel processing. Potentially, each smarticle could be associated with a processor, which would be assigned a new task without requiring new creation and association overheads.

3.4.2 Teams of smarticles.

The manager is responsible for forming teams that will perform particular tasks. Smarticles can be organised into teams that can achieve successful results more readily than individuals working alone. Teams are often described as behaving like one single agent, or smarticle, with more abilities, resources and better efficiency. A team is used to trace a discontinuity-based feature outline, (Section 5.7.1) which is not possible using a single smarticle. Teams can also be treated as flocks to create variations on the effects of basic steering behaviours (Section 5.5). The user can select the number of members of a team using the UI. Alternatively, where the user makes no such request the manager assumes a default value that corresponds to the minimum number of smarticles required to complete a task.

Team members do not necessarily all have the same behaviours or tasks. For example, the first team formed processes the polygonal mesh: one smarticle analyses the information for silhouette crossings, another for discontinuities, and a third for changes in sign of curvature.

Team mates (other members of the same team) communicate with each other through the manager. The manager can identify and communicate where other team members are and what they are doing. Team mates' position and velocity are used to calculate the flocking behaviours mentioned above, see Chapter 5.5. Teams do not, however, communicate with other teams, the manager takes care of co-ordination of tasks and team assignment. As smarticles update and query voxels in the blackboard with sample and path information they become aware of the exploration results of other smarticles not in their team.

A team is disbanded when it has completed the task it was assigned. Members of the team are then placed on the free-smarticle list, awaiting re-assignment. Although the team is initiated and dissolved by the manager it does not control it; the actions of member smarticles define the overall outcome.

3.4.3 An Example of Assigning Tasks

The user has requested that silhouettes are traced followed by sharp junctions (or discontinuities). The manager uses the number of available agents on the free smarticles list, e.g. ten. First the manager assigns each smarticle to trace a silhouette. When a smarticle detects that it has looped, met another silhouette, or the silhouette is no longer traceable, its task is finished. It reports back to the manager and is removed from the active agent list and placed on the free agent list. For the second task on the priority list, two smarticles are needed to trace each feature outline (a sharp junction). The manager, therefore, creates teams of two smarticles to trace the outline. As smarticles become available teams of two are formed. Where only one smarticle is available, a new one will be created to form the team. Both of the smarticles are associated with their team and any relevant internal data is initialised. When the teams identify that the tracing has looped or been lost, their task is complete. The team is then dissolved and the two smarticles are placed on the free agent list.

3.5 Task Priorities

Priority lists are compiled when more than one task is to be performed, or where one task is made up of many sub-tasks (such as trace all feature outlines in a region). They are calculated from user requests and data from voxels. New prioritised task lists are calculated whenever the set of current tasks is complete, or when the user changes their requests or makes a new one. User interaction identifies the regions of interest, and therefore the voxels to be considered. The data from the voxels provides estimates of the workload and workdone from voxel statistics, which are then used to calculate the priority rating for tasks. The current priority list is then used to assign appropriate tasks and perform them in the correct order.

The algorithm to create priority lists is:

```
CalculatePriorityList()
    FORALL VoxelsInNeighbourhood
        CalculateWorkload()
        CalculateWorkdone()
        CalculatePriorityRating()
    ENDFOR
    CreatePrioritisedTaskList()
```

This algorithm is used in the $Prioritise(tasks)$ function used in the $CompleteTasks$ algorithm in Section 3.2.

Statistical analysis is primarily concerned with variations in the data stored in the voxels. These variations give the manager an indication of the nature of the surface

inside the voxel: how large the surface area is, how flat it is, and if there are any features or details. The statistics also show how well a voxel has been sampled and the likelihood that all features have been found. Note that workload and workdone are estimates, the behaviour of the surface inside a voxel is subject to the limitations of sampling. It is not possible to guarantee that all features of a surface are found by sampling alone, unless other information such as a Lipschitz constant is known for the surface. The system allows the user to change the sampling rate by changing the detail level, which adjusts the step sizes smarticles take to make their paths.

The algorithm for *CompleteTasks()* shows where the priority list is used in the context of identifying tasks.

3.5.1 Workload

The workload estimate is composed of several parts that are combined to create an overall value. Maximum values for a neighbourhood (or for the entire object space) are also calculated to normalise the data. A neighbourhood refers to adjacent voxels within a particular region.

Work estimate calculations use two main sources: the potential area of surface inside the voxel; and the likely behaviour of this surface, such as high curvature, changes in curvature (such as from concave to convex) or any features that may be present. The calculations are updated as more information becomes available, through smarticle exploration.

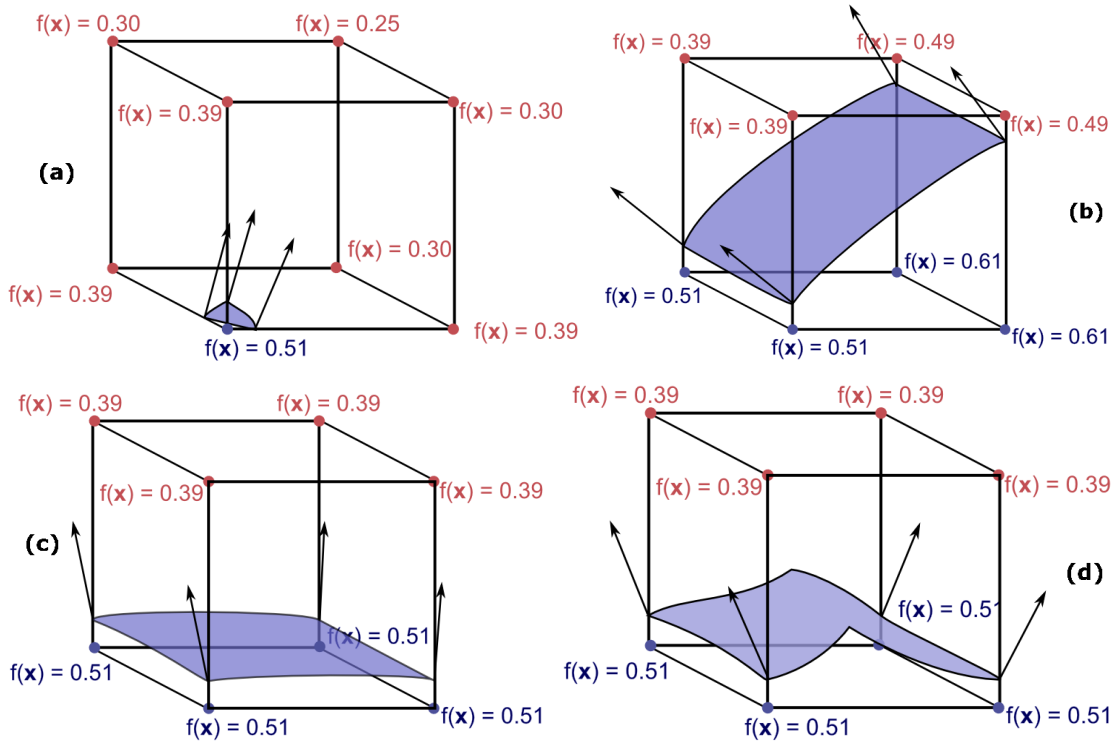

Figure 3.6: Voxel corners are identified to be inside (red) or outside (blue) of the surface ($iso = 0.5$). Including field function and gradient information can identify, for example: (**a**) a small portion of the surface in the voxel. (**b**) a large area of the surface in the voxel. (**c**) a smaller portion of the surface is present. (**d**) there is a discontinuity or small scale detail on the surface in the voxel.

Corners inside

Each corner of a voxel is identified as being inside, outside or on the surface. The number of corners identified as inside can provide valuable information about the surface. For example, if one corner is identified as being inside (or on) the surface and its field function value is very close (or equal) to *iso* then it can be assumed a relatively small portion of the surface is contained in this voxel (Figure 3.6 (**a**)). If four corners are identified as being inside and the other four outside the assumption

is that the voxel contains a relatively large part of the surface (Figure 3.6 **b**, **c** and **d**). More information is required to learn about the behaviour of the surface in such a voxel. The number of corners inside, therefore, provides the first estimate of workload ψ^c for this voxel (see Table 3.2). There are only four discrete possible values, therefore, the normalised workload values represent this. The number of corners inside is chosen because situations may arise where the only information that is available about a voxel is the field function and gradient information at the corners. The perceived behaviour of the surface in the voxel is, therefore, the second means of estimating work load.

corners inside	workload ψ^c
1 or 7	0.25
2 or 6	0.50
3 or 5	0.75
4	1.00

Table 3.2: Workload estimate from the number of corners inside the surface

Gradient Information

In a similar manner, gradient values at corners and on surface intersection points identify much about the behaviour of the surface within the voxel (Figure 3.6 (**b**), (**c**) and (**d**)). If the angle between the gradients is relatively small then it can be assumed the part of the surface in the voxel is relatively flat (Figure 3.6 (**b**) and (**c**)). Where the angle between the gradients is divergent (Figure 3.6 (**d**)), this identifies a potential feature or detailed part of the surface in the voxel. Again, initial data may not identify all surface details within voxels. As more information becomes available a better understanding of the nature of the surface in the voxel emerges. The second

estimate of workload is therefore calculated using gradient information:

$$\psi^g = \left(\frac{\theta_i}{\Theta_{max}} \right) \tag{3.1}$$

where ψ^g is the normalised workload estimate due to the gradient, θ_i is the maximum angle between gradients in voxel i, and Θ_{max} is the maximum angle between gradients (in all neighbourhood voxels).

Level of Detail

Another important piece of information is the current level of detail (LOD). The system process mimics the rough-to-detail and piecewise approach many illustrators use, therefore there must be a concept of the current local LOD. The LOD identifies the level to which voxels are sampled. Where the user identifies a particular region to be rendered in more detail, this level is local and does not apply to the whole model. For example, smarticle path step sizes will be smaller in a region with a higher LOD. The local normalised LOD information is stored in each voxel.

The user can specify the level of detail, therefore, the last measurement of workload is related to the level of detail. The levels are normalised so that $0 \leq \psi^d \leq 1$.

Total Workload Estimate

All of these workload estimates are summed to provide the total workload estimate ψ_i for each voxel:

$$\psi_i = \left(\frac{\psi_i^c + \psi_i^g + \psi_i^d}{\psi_{max}^c + \psi_{max}^g + \psi_{max}^d} \right) \tag{3.2}$$

where ψ^c is the number of corners inside and ψ_{max}^c is the maximum number of corners inside. ψ^g is the workload estimate due to the angle between the gradients

and ψ^g_{max} the maximum divergence of the gradients. ψ^d_i and ψ^d_{max} are the current and maximum levels of detail, respectively. All maximum values are calculated for the user specified neighbourhood.

3.5.2 Workdone

The estimate of workdone is re-evaluated as more information about the contents of the voxels are collected. Much of this information is in the form of tallies, which record the number of smarticles; smarticles' steps (or samples); and feature points identified.

The statistics can identify whether a voxel has been sufficiently sampled or if it requires further investigation. For example, a voxel has four corners identified to be inside the surface and the gradient values indicate a possible feature, yet only four samples or path steps have been taken inside the voxel, and no feature has been traced. This voxel would be identified as being insufficiently sampled.

Information about the maximum and minimum tallies are recorded for contextualising single voxel information. For example, the maximum number of smarticle steps or samples in single voxels and over all voxels are recorded. This is used to calculate the context of workdone and ultimately priority. Consider a voxel that has been identified through user request and has ten surface samples. Without information about the other selected voxels, there is no context to determine whether this is a poor or a well sampled voxel. If the other voxels have an average of two surface samples then the voxel is relatively well sampled. Conversely, if the other voxels have an average of two hundred surface samples then the voxel is poorly sampled.

The workdone sample-estimate σ_i is normalised for the neighbourhood:

$$\sigma_i = \frac{\sigma_i^{'}}{\sigma_{max}} \tag{3.3}$$

where $\sigma_i^{'}$ is the un-normalised workdone estimate and σ_{max} is the maximum value in the neighbourhood.

An important piece of information is whether a feature has been identified in the relevant voxel. In order to calculate the workdone for each feature found the following heuristic is used:

$$\sigma^g = \left(\frac{\psi_i^g}{2^{\mu_i}}\right) \tag{3.4}$$

where μ_i is the number of feature points in the voxel i.

This is used to modify the results from Equation 3.2, so that:

$$\psi_i^{'} = \left(\frac{\psi_i^c + \sigma^g + \psi_i^d}{\psi_{max}^c + \psi_{max}^g + \psi_{max}^d}\right) \tag{3.5}$$

.

The situation where no features are identified from the initial polygonisation is extremely rare, in tests this situation has not occurred. In such a case, however, (and in the absence of user direction) the manager will create a feature outline tracing priority list based on the gradient information statistics from polygonal data.

3.5.3 Priorities

Tasks are prioritised to identify the order in which they will be performed. The workload must be offset against the workdone to avoid repeatedly recomputing the same information, or examining the same voxel.

Priorities are dependent on user requirements and subsequent voxel data population. The management scheme does, however, have a general priority scheme, which is used in the absence of user interaction:

1. possible feature outlines:

    a silhouettes,

    b sharp junctions (discontinuities or change in curvature);

2. not enough sample information.

Priorities Φ are created by looking at the difference of workloads ψ_i compared with the tallies that estimate workdone:

$$\Phi^i = \psi_i - (\psi_i^{'} * \sigma_i) \tag{3.6}$$

where σ_i is the number of smarticle samples in voxel i from equation 3.3.

In Figure 3.7 the priorities of voxels are illustrated. The priorities are calculated per voxel and as such, each triangle in the voxel is the same colour. The darker the colour the higher the priority. In **(a)** the priorities are calculated before any tracing has occurred. In **(b)** the priorities are recalculated after tracing the discontinuous region around the bite of the pear.

3.5.4 Numerical Examples

The data compiled in Table 3.3 relates to the example images shown in Figure 3.7. The orange box outlines the first voxel used in these examples, the red box shows the second. This numerical example presents a detailed view of calculating the priority

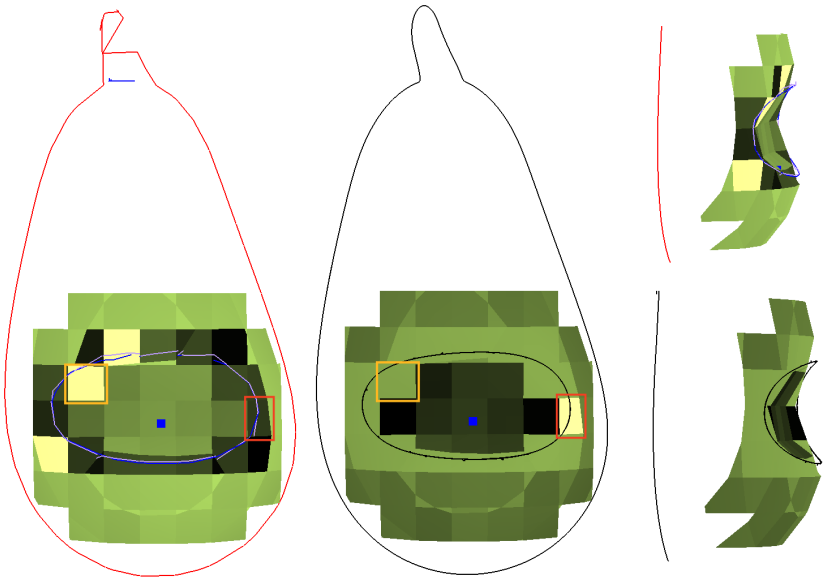

Figure 3.7: The user identifies a point (blue square) around which the voxel priorities are calculated. The higher the priority of the voxel the darker the colour of its polygons. In relation to Section 3.5.4: the orange box is example 1, and the red box is example 2. **(a)** The priority before tracing. **(b)** The priority is different after the feature has been traced. **(c)** Before tracing side view. **(d)** After tracing side view.

rating for each of these voxels. The workload and workdone estimates are calculated using the above methods and the values shown in the table. There are two sets of values that represent the workload, workdone and priority ratings for each voxel before and after the discontinuity has been traced. The resolution of the initial polygonisation was 20^3. The neighbourhood was set to 3 voxels in each direction around the currently identified voxel, therefore the total number of voxels in the neighbourhood was 99.

Before the trace the voxel identified by the orange box has four corners inside the surface so it is assumed that it contains a relatively large part of the surface. The divergence of the gradients is about half that of the maximum value identified in the

neighbourhood. The workdone estimate identifies that this voxel has the maximum number of samples in the neighbourhood, so the workdone estimate is is high.

The voxel identified by the red square has only two corners inside and an average gradient divergence, which amounts to a slightly lower priority rating than the voxel identified by the orange box. The workdone estimate is slightly lower than for the orange because there have been fewer samples taken in the voxel identified by the red box.

The priority rating for the orange box before the tracing was calculated to be $\Phi^i = 0$ because of the workdone estimate being at the relative maximum.

After the feature outline has been traced, the priority ratings are reversed (relative to one another) because of the number of samples taken in each voxel. The voxel identified by the orange box has half the number of samples as the red, which incidentally has the maximum number of samples in the neighbourhood. In Figure 3.7 you can see that the voxel identified by the red box has a larger part of the feature tracing than the voxel identified by the orange box.

3.6 Data repository: Blackboard.

A blackboard is a public database [HR85] (Section 2.6.4), which is an abstraction of the smarticles' environment (Section 4.3.2) that contains the results of the smarticles' exploration. The information stored in it is accessible for reading and writing through the manager. This abstraction is in the form of a collection of voxels, which organise data in a coherent and useful manner. The voxels are updated by the manager with information provided by smarticles.

Trace	Orange before	Orange after	Red before	Red after
Number of corners inside				
max corners inside	4	4	4	4
corners inside	4	4	2	2
ψ^c	**1.0**	**1.0**	**0.5**	**0.5**
Gradient divergence				
max in n'hood	1.239	1.239	1.239	1.239
max in voxel	0.789	0.789	0.861	0.861
feature pts IDd (#steps)	0	7	0	9
ψ^g	**0.637**	**0.005**	**0.695**	**0.005**
Level of detail				
max	1.0	1.0	1.0	1.0
in voxel	0.5	0.5	0.5	0.5
ψ^d	**0.5**	**0.5**	**0.5**	**0.5**
Total Workload estimate				—
ψ^i	**0.712**	**0.335**	**0.565**	**0.335**
Workdone estimates: Number of samples				
maximum number	6	149	6	149
number in voxel	6	70	4	149
σ_i**:**	**1**	**0.470**	**0.667**	**1.0**
Priority				
Φ^i	**0**	**0.157**	**0.565**	**0**

Table 3.3: Numerical example of priority ratings

Subdividing the space into voxels allows for varying levels of detail across the image, and it can be used to meaningfully isolate regions that require more (or less) attention. As with pen-and-ink illustrations, not every area needs to be rendered to the same level of detail. In order to provide similar functionality, the voxels allow local levels of detail and local estimations of workload and workdone. Also, not all of the contents of a voxel need be at the same LOD. Changes in LOD are graded to achieve a smooth blend so as to avoid a detectable change, see Chapter 6 for

examples. Alternatively LOD are applied at the smarticle path level (in the form of step sizes and accuracy) so are naturally coded into the sub-path information that each voxel contains.

3.6.1 Voxel Organisation.

The granularity of the voxel grid is the same as that of the initial polygonisation. This is purely for convenience as the content is continually updated with information from smarticles' movement.

Most smarticles are located in regions close to the surface, which means that there is a large portion of the object space that is not generally sampled or used. For this reason a hash function with buckets [GWW86] is used to store the relevant voxels. A voxel is identified from either its identifier or by a point that is contained within it. The identifier is a real value triple (i, j, k) that identifies the voxels position in the object space, with the back bottom left corner of the voxel space identified as $(0, 0, 0)$. The blackboard can also identify a voxel from a point in space by determining in which voxel the point lies. Once the voxel identifier has been found, this is used in the hash function (Chapter 2.2.3) to identify a voxel-buckets list, see Figure 3.8. A pointer to the actual voxel data is stored in one of these buckets. Typically there is only one voxel bucket pointing to one single voxel. There is, however, no guarantee that this is a unique identifier and it is not uncommon for there to be more than one voxel bucket present.

The algorithm for finding the voxel is:

FindVoxel()

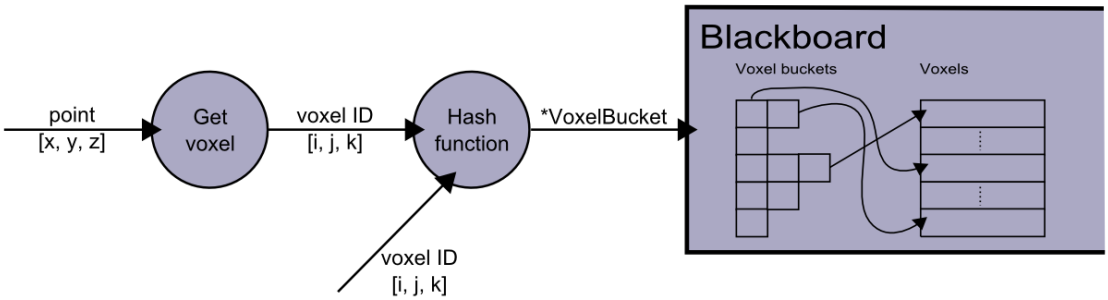

Figure 3.8: Accessing a voxel in the blackboard. A voxel can be identified directly or from a point contained within it. The voxel ID is then used in a hash function to identify the list of voxel-buckets that will contain the address of the actual voxel data.

$$voxelID = GetVoxelID(Path[LastPoint].xyz)$$

$$VoxelBucketList = Hash(voxelID)$$

$$VoxelFound = false$$

$$Find(voxelID, VoxelBucketList)$$

$$Update(VoxelID, PathInfo)$$

Note: not all voxels in the data structure are guaranteed to contain parts of the surface. Smarticles that explore regions (without being constrained to the surface) may stray into a voxel that does not have a part of the surface, but neighbours a voxel that does. Also, only voxels identified by the initial polygonisation or smarticle exploration will be on the *initial* list, which means that sampling levels can lead to small scale details being missed (see Chapter 2.2.3). Voxels that are missed in the initialisation stage can be identified through smarticle exploration and automatically added to the list. A complete list of surface voxels can not always be guaranteed.

Where the list is incomplete after initialisation smarticles must explore regions to identify new surface voxels.

3.6.2 Voxel information.

Initially, data in the voxel is provided by the polygoniser, i.e. the number of polygons and pointers to their vertex positions and gradient values. Each of the identified surface voxels are examined to compile statistics that estimate the behaviour of the surface inside the voxel. As a smarticle moves along its path it samples the implicit object at each step, the data in the voxels is updated with this information. The statistics are periodically re-calculated when a set of tasks have been completed.

Voxels contain a list of the triangles generated from the polygonisation process. The polygon data includes the number of polygons, and the position and gradient information for each vertex.

Smarticle step information is also stored; this includes the number of smarticles located in the voxel at any point in time and the total number of samples inside the voxel.

The information stored in each voxel is:

- voxel ID: integer value triple [i,j,k]
- voxel position: real value back bottom left corner co-ordinate (x,y,z)
- sample data for:
    * voxel corners
    * polygon vertices

    * smarticle sub-path steps (including feature traces)

- number of corners inside (i.e. indicate a surface voxel)
- voxel corner, polygon vertex and path-step sample statistics (field function value and gradient) for priority rating:

    - average
    - range
    - standard deviation.

Note: sample data is field function value, gradient, mean and Gaussian curvature, and the principal directions of curvature. Also, for easy identification of voxels that contain parts of the surface, the number of voxels corners inside the surface are stored.

Smarticles have private information which they do not always post to voxels, for example the length of their entire path or the number of voxels visited. It is not necessary to have all information publicly available. Note, however, that when smarticles complete their assigned task their private data is discarded. Any data that is relevant to the system is, therefore, communicated to the voxels.

Note that assumptions made about the contents of a voxel, and therefore about the behaviour of the surface in that region in earlier stages can not be guaranteed, a small scale detail may be missed. As more information is accumulated from smarticle samples, assumptions about voxel contents are updated and refined using newly calculated statistics. More information means better sampling which gives a better likelihood of identifying small scale details.

3.6.3 Communication Pipelines.

The manager is responsible for communication with the user, and between smarticles and the ISM system.

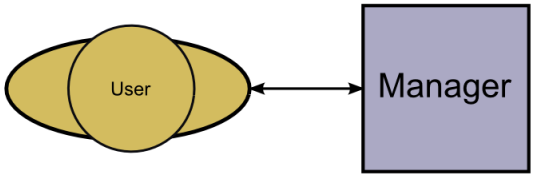

Figure 3.9: The user creates the illustration by specifying their requirements, such as areas to explore or feature outlines to trace, in a chosen level of detail.

The Manager and the User.

The user's interaction with the system directly affects the progress and process of the sampling and rendering. User requirements are interpreted by the manager to identify the tasks smarticles will perform. One of the main purposes of the management scheme is to co-ordinate completion of these tasks without overburdening the user, for example, the user can identify a particular feature outline to be traced in more detail without being concerned about how this is achieved. In Section 3.3 user requests are related to smarticle tasks.

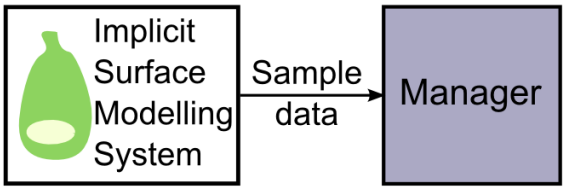

Figure 3.10: The manager obtains field function values and gradients for any point in space from the Implicit Surface Modelling (ISM) system.

The Manager and Implicit Surface Modelling System.

Field function and gradient information is fundamental to all aspects of the system. It is used to identify the surface, calculate steering forces and find features.

The manager communicates queries between smarticles and the implicit surface modelling system. This single line of communication means the Implicit Surface Modelling (ISM) system can be treated as a *black box*, i.e. the only requirement is: given a point in space, a field function value and gradient is returned.

As mentioned in Chapter 2.4.1 (WH style particle systems) one of the reasons for developing particle systems for implicit surfaces was for fast visualisation. Field function and gradient evaluations are costly and constitute a bottleneck in the system [BBB+97].

Introducing the manager as a central controller does not, therefore, create a new bottleneck. In contrast, the manager organises smarticle sampling so as to reduce the number of times that field function values and gradients are evaluated while identifying and tracing as many feature outlines as possible. The workload and workload estimates identify voxels that have been sampled avoiding unnecessary recalculation.

3.6.4 Initialising the System.

As a pre-processing step the manager creates a rough polygonisation of the object (Figure 3.11 **a**). Using the polygon vertices as initial positions for smarticles (Figure 3.11 **b**) provides a sufficient coverage of smarticles, both in terms of area and density (adding a random vector reduces the regularity of the initial placement).

The list of voxels that contain surface intersections is not guaranteed to be com-

plete at initialisation. It does, however, identify locations of many of the features, or areas that require exploration. The manager assigns a team of smarticles to analyse the polygonal data, to identify features and surface details. Each voxel and its associated polygons are examined to provide a rough approximation to the features on the surface (Figure 3.11 **c** and Section 5.3), and a preliminary set of voxel statistics.

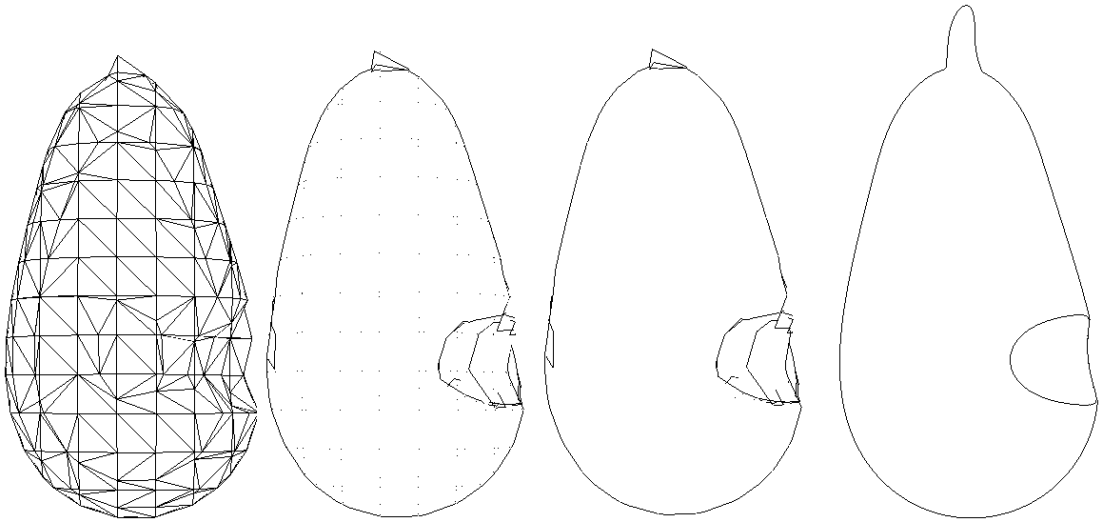

Figure 3.11: (**a**) The initial rough polygonisation of the pear. (**b**) The polygon vertices can be used as the starting points for smarticles. (**c**) The rough approximations of the feature lines. (**d**) The final traced, higher quality feature lines.

This pre-processing step provides a great deal of information for these voxels. Each voxel has corner, edge and polygonal surface intersection details. The polygonisation is, in general, not fine enough to guarantee that all features are found; as in Figure 3.11 **a** to **c** where the stalk of the pear has not been identified or polygonised. Neither does it provide enough information about the voxels to be a definitive representation of the model; for example in Figure 3.11 **b** and **c** where the bite from the pear, or discontinuous region is not accurately illustrated. Figure 3.11 **d** shows

a high level of detail tracing of the feature outlines for comparison.

Features that are not identified from the polygonisation step, such as the stalk of the pear in Figure 3.11 are identified and traced through smarticle exploration.

Using an initial rough polygonisation is a faster method of achieving a good coverage of smarticles than using a Witkin-Heckbert style attractor-repulsion method (see Chapter 2.4.1). Figure 3.12 shows the train model after initialisation of a WH style particle system and the manager and smarticles system.

The WH style particle system initialises (loads the object, places 1000 initial particles and starts the distribution process). Figure 3.12 **a** shows the result after initialisation, which takes approximately ten seconds, Figure **b** shows the progression after sixty seconds. These times include the time taken to query the BlobTree.

Figure 3.12 **c** and **d** shows the results of initialisation of the manager and smarticle system. The manager has polygonised the object with a grid resolution of 15^3 and the first team of (three) smarticles have processed the mesh to identify potential feature outlines (red lines are silhouettes, blue lines are discontinuous regions and purple lines are changes from positive to negative curvature). Initialisation took 10 seconds.

Referencing the results from Figure 3.12 identifies the results after initialisation when both systems take approximately ten seconds. Figures **c** and **d** show that the management and smarticle system identifies many more features that the NPR-BT system, shown in Figure **a**. In fact, even when left to run for sixty seconds (Figure **b**) the NPR-BT system has not identified as many feature outlines as the management and smarticle system.

Note that the train model has 723 nodes. For models with significantly fewer

nodes initialisation times are similar. For example the pear has six nodes, using the WH style particle system a good coverage of particles is achieved and the silhouette and discontinuity is identified in four seconds. Using the smarticle system initialisation also takes four seconds and the silhouette and discontinuity are identified, although in a low LOD.

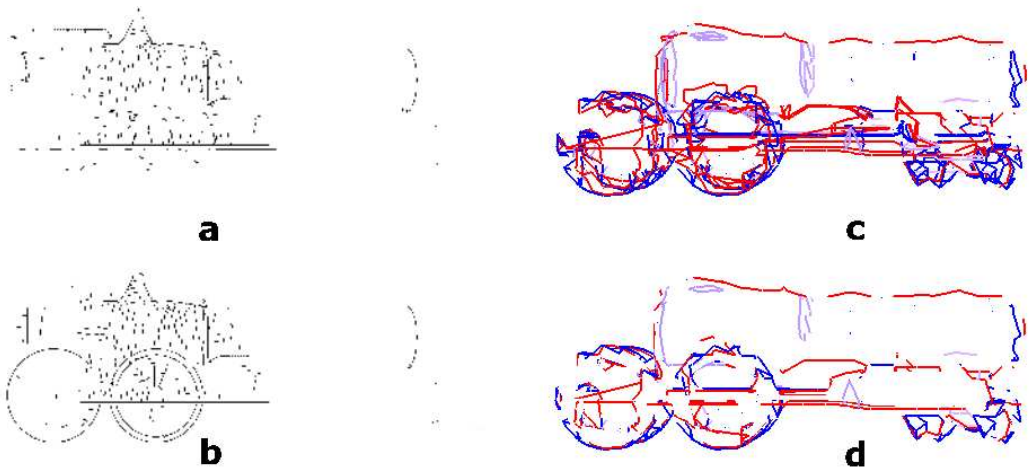

Figure 3.12: (**a**) The WH-style particle system after initialisation and using 1000 particles. (**b**) The WH method after 60 seconds. (**c**) The smarticles method with no HLR, after system initialisation which took 10 seconds. (**d**) The smarticles method with polygons drawn white for HLR.

3.7 An Example of User Guided Illustration Creation

Figure 3.13 shows the creation of an illustration. The pre-processing stage used a polygonisation grid resolution of ten. This results in a few rough details that show the basic shape of the engine without any details (Figure 3.13 **(a)**, the red lines

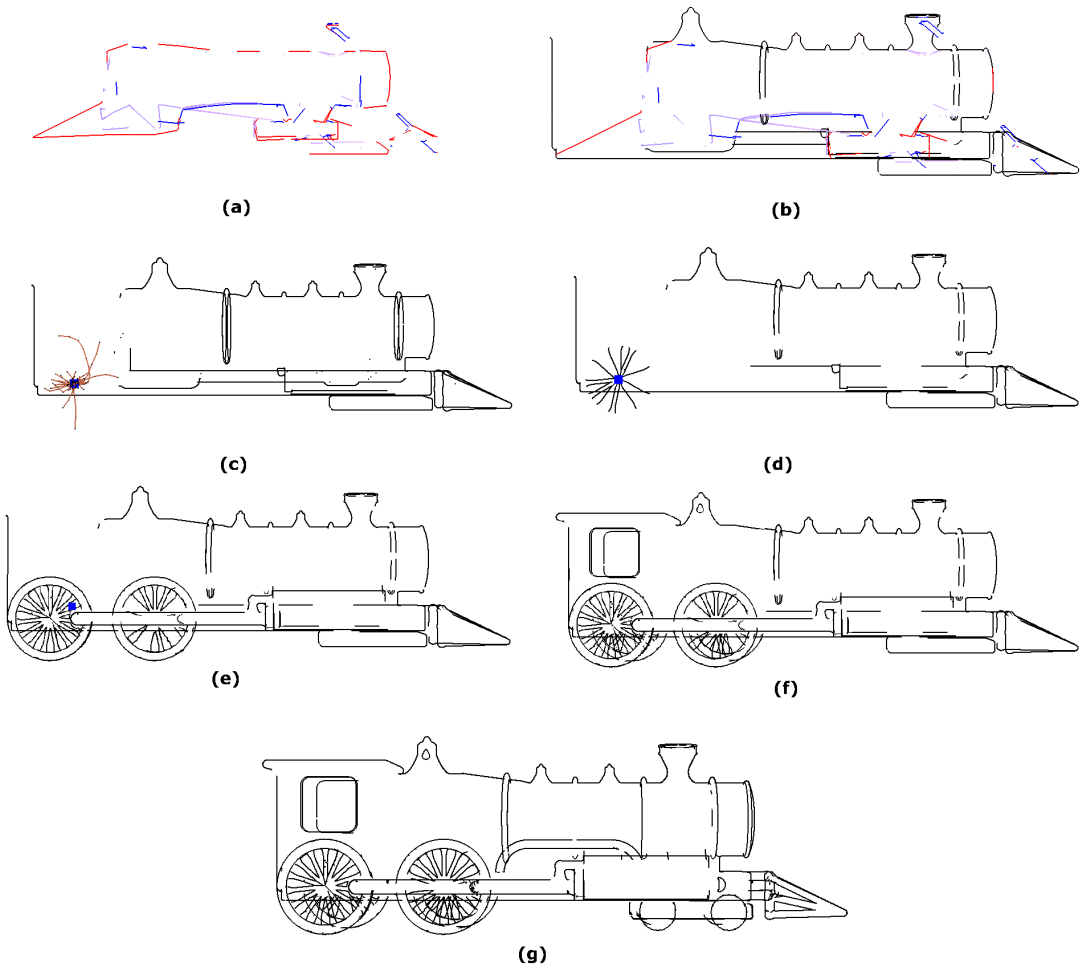

Figure 3.13: An Example of a user guided illustration creation, see Section 3.7.

are the silhouette approximations, blue lines are the rapid changes in gradient, and purple lines are the changes in sign of curvature). Figure 3.13 **(b)** shows the initial approximations with the results from the first outline tracings. In image **(c)** the user has specified attention to be concentrated in the region around the blue square. The brown lines are the result of a group of 100 smarticles using the wander behaviour (not constrained to the surface) to explore and sample their local neighbourhood.

Image **(d)** illustrates where some more surface features have been found (spokes of the wheel), without the smarticles paths drawn. The user specified point is moved and the smarticles find more feature outlines to trace, more of the engine and part of another wheel is identified, **(e)**. All of the back wheels are visible, the engine details, the cattle guard and the front wheels, this is the user's final image, and does not necessarily capture every detail from the final model.

Note the initial polygonisation was used for Hidden Line Removal (HLR). The polygons are offset slightly along the direction of their normal to the inside of the model. This is not a robust or accurate method as it is only as reliable as the initial polygonisation. It is used only as a means of achieving fast HLR. This method means there are a number of places where overlapping lines are visible. A polygonal subdivision method would create a more accurate approximation to the surface, and voxel subdivision would create polygons where none exist in the model, this is an important area of future work.

3.8 Conclusions and Discussion

In this chapter a management scheme has been presented that acts as a central controller for a MAS. This approach is designed to address the shortcomings of particle-based systems that rely on distributing particles over a surface and identify features through proximity.

The manager uses teams of smarticles and priority ratings to automate relevant *tasks*, such as feature outline tracing, whilst allowing the user overall control of the rendering process. The manager represents a holistic view of the object which

interprets user requests to create appropriate tasks to assign to smarticles. It then collects, analyses and presents the findings of the smarticles, which is used to explore the object space and create strokes. The management scheme, therefore, combines the benefits of local, distributed components, such as smarticles, with an overall view of the object. The manager facilitates co-operation and communication between smarticles whilst not dictating their behaviour.

Using the combination of the manager and smarticles, features on complex models are identified and traced faster than using the previous method [FJW+05]. The polygonisation pre-processing step is useful in a number of ways. The smarticles have a good distribution across the surface at system initialisation, with a higher density of points in regions of higher curvature. Also, examining the triangles identifies more features in a shorter space of time than using a Witkin-Heckbert style distribution and proximity requirements, see Figure 3.12.

The issues raised by Pang and Smith in [PS93] about lack of management and co-ordination are addressed by using the agent engine Vigo and the manager with the blackboard, using the estimates of workload and workdone and the resulting priority ratings. The Vigo agent engine takes care of neighbourhood considerations and the blackboard tracks the relevant history of smarticles and facilitates inter-smarticle communication.

The manager interfaces with Vigo by directly relating to the scene class, which is responsible for agent dynamics. The manager creates smarticles which are inherited from Vigo agents and registers them with the scene. Vigo is mainly used for scene representation and neighbour identification.

One of the most needed additions to the management system is an effective user

interface. The user interface is basic and parameter driven, which is not ideal. A more direct user interface that would facilitate user investigation and exploration is desirable. The number of parameters is useful in terms of the variety of results that can be produced, but a better presentation of them, their control and the results of their variance would benefit the system.

The concepts of workload and workdone are heuristics inspired from consideration of load balancing problems. A formal analysis of the effectiveness of these techniques to create priority lists that satisfy user requests would also be advantageous. This would however, be partly dependant on a more direct user interface.

The contributions in this chapter are:

1. Organising agents to collectively explore an implicit surface environment and visualise object data using NPR styles.

2. Analysing the object space to identify potential locations of features.

3. Interpreting user requests to create appropriate tasks.

4. Prioritising tasks to be processed based on workload and workdone estimates.

5. Using polygonisation for initial positions and voxel data thereby identifying features faster than the previous method.

Chapter 4

Smarticles

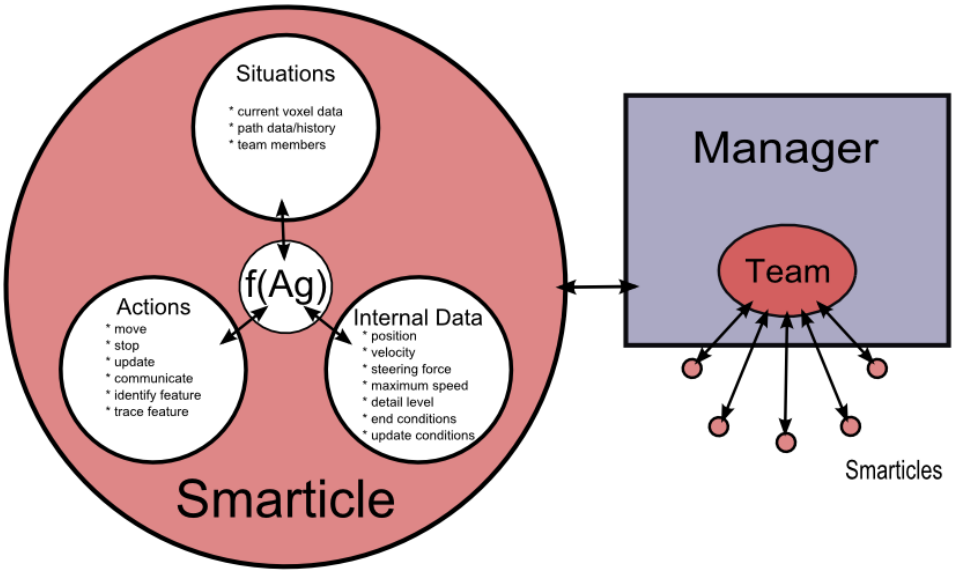

Figure 4.1: Smarticles explore their environment using situations, actions, internal data and a relating function.

In this chapter a new method for creating pen-and-ink illustrations by extending current particle-based systems is presented. In this method particles are extended to become agents, referred to as **smarticles**, or **sm**art p**articles**. These agents are extensions to the smarticles described in [JWS06], which were particles in a particle system. Previous methods rely on particle proximity or local surface properties to identify feature outlines and explore the surface. Particle distribution can be very time consuming for complex implicit objects and subsequent feature identification

is reliant on a good coverage of particles. In contrast smarticles complete these tasks in a goal directed manner with no costly distribution method. A **manager** (Chapter 3) assigns **tasks** to the **smarticles**, such as finding or tracing feature outlines (Figure 4.1). A smarticle's *situation* along with its *internal data values*, which represent behaviours (covered in Chapter 5), are used to determine *actions* that will accomplish the assigned task. Co-ordination, collection and analysis of data from smarticles is carried out by the manager (Chapter 3).

In this chapter, a smarticle is defined in the context of an *agent* in a Multi Agent System (MAS), with respect to performing tasks assigned by the manager. Chapter 5 describes the behaviours agents use (represented by internal data values, such as the steering direction vector) to perform actions, which ultimately complete tasks.

4.1 Introduction

In previous work (Chapter 2.4.1), Witkin-Heckbert [WH94] style attractor-repulsion based particle systems were used to sample and render implicit models. In these systems particles identify features as they are being distributed across the surface. Distribution is based on density of particles in a region, properties of the surface (such as curvature), or with user direction. The main drawbacks with such techniques are associated with the distribution method. Distributing the particles over a complex surface to obtain a good coverage can be time consuming. A further consequence of this is the time taken to identify features. Particles identify a feature by either being moved across it or by using local surface properties to converge upon it[RRS97, CA97, SH05]. In both cases particles are assumed to be in close proximity to the

feature.

Smarticles are designed to use new methods to find, trace and illustrate surface details. They investigate their environment in a goal directed attempt to find details, rather than relying on proximity. The smarticles use the global view provided by the manager (querying the ISM system and the blackboard abstraction of the model) along with behaviours (such as seek and wander) to actively seek out areas of the surface that contain features or details.

The contributions presented in this chapter relate to expanding the traditional particle-based system to an agent-based system that uses behaviours to modify actions that are used to complete tasks.

In this chapter, first the smarticles are explained in terms of their development and expansion to be agents in a Multi-Agent System. Next, the method of performing tasks is described in the context of the smarticle's environment, and therefore the situation it is in, any team members it has, and how internal data values and its decision function are used to determine the appropriate action.

4.2 From Particles to Smarticles

A *Smarticle* is an extension of a particle based on the models presented in physically based simulations [WBK97], (Section 2.4). Basic particles have attributes for position and velocity, whereas smarticles also include behaviours and a history, which can be used for data analysis. Smarticles have been developed as agents in a Multi-Agent System (MAS). The addition of behaviours, a history and the global view provided by the manager (using the blackboard and analysis of its contents)

gives the smarticles more information about potential locations of surface features and how to find and trace them. The reliance on simple proximity for identifying a feature is removed. They use the field function value, gradient, curvature (mean and Gaussian) and principal directions of curvature along with user specified stroke direction preferences (such as horizontal, vertical or diagonal) for their investigation. This information is used to determine the path that a smarticle takes through its environment. A path is constructed from steps that the smarticle takes at each iteration (time step) of the system. For clarification: *path steps* are those used to construct the smarticle's path (positional) and *time steps* are system iterations (temporal). In most circumstances, each system time step would result in an additional path step for a smarticle or terminate the path. The information in a smarticle's path is generated during a run of the system, during which a task is completed.

In Figure 4.2 a run of the system is depicted from the point of view of a smarticle. Smarticle level processes are depicted by pink circles and rectangular data storage areas. First the manager assigns a task by initialising relevant internal data values (IDV). Next, the potential new position is evaluated for termination criteria, if the path is not to be terminated, then the move action is performed. When termination conditions have been met all relevant information is updated with the manager, i.e. the report action is performed. Each smarticle's path is represented in its history, which is updated as each action is performed and is referenced when performing and evaluating the result of actions or reporting back to the manager. Note that environment data passed from the manager to the smarticle is field function and gradient information from the ISM system.

The fact that smarticles are defined as agents in an MAS (as outlined in Chapter

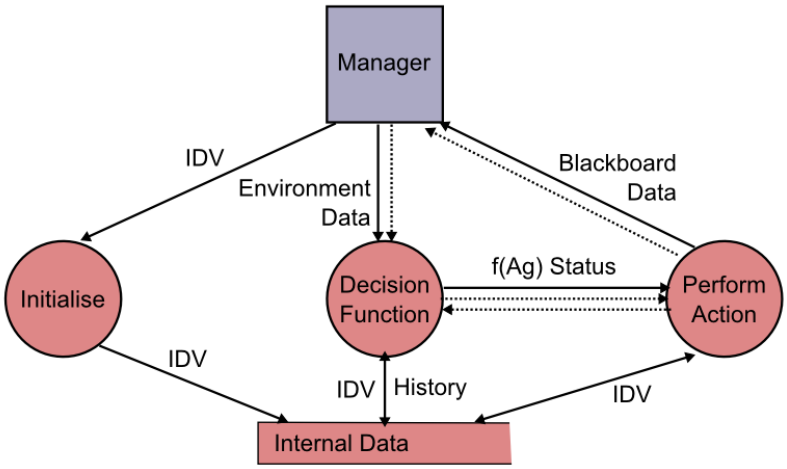

Figure 4.2: A smarticle's view of a run of the system (smarticle processes and data storage are denoted by pink circles and rectangles respectively). Data flows are black lines with arrows representing the direction of data flow and system operation flows are depicted as dotted lines with green text.

2.6) implies they are framed in the context of an agent acting in an environment, having a decision function which relates the sets of valid situations, actions and internal data values $Ag = (Sit, Act, Dat, f_{Ag})$ (Chapter 2.6.1).

Smarticles are assigned tasks and relevant attribute values from the manager. Current attribute values or IDV represent part of a smarticle's knowledge about its assigned task. This knowledge is used together with the smarticle's situation for the Decision Function (DF) to determine the action it should take, see Section 4.3.7.

Smarticles are semi-autonomous, they are assigned a general task from the manager but actual performance of the task is carried out either on their own or as part of a team. Smarticles can be described as social because of their group behaviours and their co-operative approach to problem solving and task completion. They can also be described as reactive, or behavioural, as they are closely linked to

the representation of their environment, see Section 4.3.2.

The abstraction of the environment, although based on a static model, is continually evolving in direct response to the actions of the smarticles. As smarticles investigate the environment the abstraction progressively contains more accurate and useful information, see Section 4.3.2.

Tasks, or subtasks, are carried out by performing actions, which are defined, in part, by information associated with behaviours. The five actions a smarticle can perform are move, stop, report (or update), communicate and identify feature points, covered in Section 4.3.5. There are four behaviours related to the move action, which are known as polygon processing, steering, flocking and feature line tracing (covered in detail in Chapter 5).

A smarticle's initial position is determined either from vertices of a polygon (see Chapter 5.3) or by sample points identified by other smarticles or the user. As a smarticle moves it creates a path, which is made up of a series of steps. The size of the steps are dependant on the level of detail requested by the user. The length of the path is determined by constraints to regions, which vary according to the task assigned to the smarticle. In this research a **region** is defined by the curvature of the surface, a volume (possibly a voxel), or a distance from an existing feature line, see Chapter 5.4.4.

4.3 Performing Tasks

Tasks are identified and assigned by the manager based on user requirements (Chapter 3.3 and 3.4). The manager assigns a task by activating a smarticle and providing

relevant data, such as the initial position, steering direction and path termination criteria. This data is stored in the smarticle's IDV.

A smarticle carries out a task by examining its environment (and other team members) to assess its situation before using a Decision Function (DF) to find an appropriate action. The values of internal data are provided, in part, by the behaviours and are used as properties, or modifiers, of the action.

A task is completed when certain conditions are met or path termination criteria are fulfilled. For example, where the task is to initialise the blackboard, the task is completed when the polygonisation has been performed, the data is stored and analysed, and initial features are identified (as far as possible).

Tasks assigned to smarticles are (from Chapter 3.4):

- initialise the blackboard data structure (Section 3.6) using the polygon processing behaviour (Section 5.3) and voxel statistics (Section 3.5)
- investigate a region (or explore the environment), e.g., using the wander behaviour (Chapter 5.4.1)
- identify features (or points on a feature) (Chapter 5.6)
- trace feature outlines(Chapter 5.7)
- position stroke samples, using steering behaviours (Chapter 5.4)
- report relevant information at appropriate times, such as when a smarticle changes voxel or completes its task.

In this section smarticles are described in terms of what is required to complete their assigned task. First, an overview example is given, which explains how a

smarticle carries out a task. Then, the smarticles are described in the context of the environment and their view of it, their team-mates (if any), and how the DF and IDV are used to decide upon actions.

Completion of tasks is dependent on communication, either between smarticles or with the manager. Smarticles report information to the manager (Section 3.6), which is used to gain a better understanding of the environment. Where teams are used to complete tasks, members also communicate with one another to share data (see Chapter 3.4.2 and Section 4.3.4).

4.3.1 An Example

Figure 4.3 illustrates the process and data flow of performing a task: the processes involved in the decision function and the relevant actions that are performed. For example, where a smarticle is assigned the task of tracing a silhouette its initial task data includes a starting point on the silhouette. The steering direction and initial velocity are calculated using the cross product of the surface gradient and the view vector, which is a good approximation to the direction of the silhouette. This data is stored in the smarticle's IDV.

When the signal to start processing is received from the manager the smarticle evaluates its situation, i.e. it references the IDV that identifies how to calculate the steering direction, calculates the potential new position and asks the manager what the gradient is at this point in space.

The smarticle then uses the decision function to evaluate if the move action is appropriate. This evaluation checks to see if the step will satisfy any of the termination criteria to end the path. If the termination criteria have not been satisfied, the

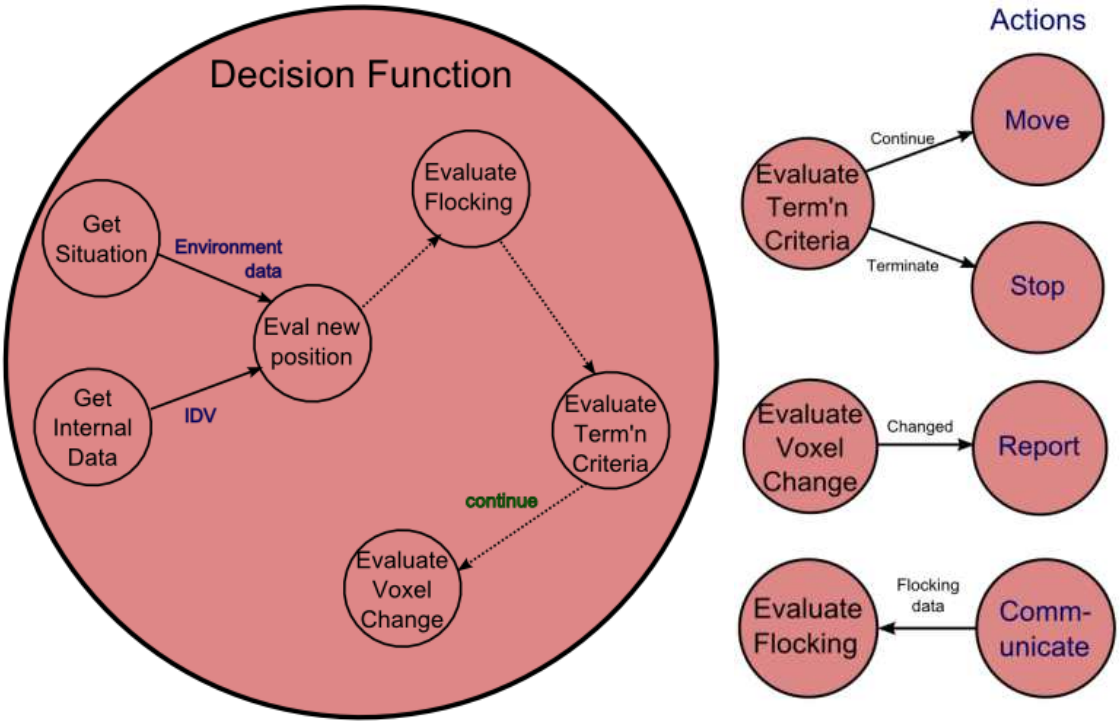

Figure 4.3: **Left**: Details of the processes involved in the decision function. **Right**: The actions that are evoked as results of the decision function.

smarticle confirms the move action and adds the current step to its path history.

Next, the smarticle evaluates if in the current step it has changed voxel from the previous step, if it has the segment of the path that is contained in the previously identified voxel is reported to the manager for inclusion in the blackboard.

The smarticle's history is also updated to identify that the smarticle has changed voxel, so a new path segment reference is created. The path segment references allow communication of only the relevant segment of the path to the correct voxel.

This process continues until one of the termination criteria are fulfilled, i.e. the silhouette loops or is lost. In which case, the smarticle flags the IDV to identify the

path is complete, without inclusion of the current path step, and communicates the (unreported) path information to the manager for inclusion in the blackboard.

4.3.2 The Environment

The agents' environment is made up of information about the object and the abstract view of the object, i.e. the Blackboard. A smarticle's view of its environment is provided via the manager.

Smarticles collectively develop the abstract view of their environment from the ISM system queries. This abstraction is in the form of a *Blackboard* (Chapter 2.6.4 [Cra93]). The blackboard is a voxel-based public repository of information that holds the data from smarticles exploration (Chapter 3.6). Although the 3D model does not change (the model is static), the abstract data structure representing it is continually updated. Smarticles use relevant information from their investigations to update the abstraction (by updating the relevant voxels), and therefore their interpretation of the environment. For example, as a smarticle takes steps along its path the *sample* (field function and gradient) and curvature information is periodically updated and stored in the relevant voxels. The information individual smarticles store about their environment is local, it is limited to the path. Sharing this information provides a better understanding of the environment on a global scale. The blackboard is used to collect this information and is subsequently used to examine the environment. As each smarticle investigates and updates the blackboard, more information is available to *all* smarticles for interpretation of the object.

Although the 3D object is not discrete, the abstraction of the environment (the Blackboard) can be seen as having a set Env of k possible abstraction states:

$$Env = \{env_0, env_1, \ldots, env_k\}. \tag{4.1}$$

A smarticle's path is represented in its history, at each step more is known about the environment. Adding smarticle data from a system run (up to time n) produces the sequence $(env^0, env^1, \ldots, env^n)$ with $env^i \in Env$ and $0 \leq i \leq n$.

The view of the environment is the information available about it at a particular point in time. The view at env^n is therefore more detailed than at env^0. As more information becomes available more accurate estimates, priority ratings and interpretations of the environment can be made.

For example, at env^0 the blackboard voxels contain nothing, at env^1 the voxels contain the results of the initial polygonisation, at env^2 the voxels also contain the results of the initial smarticles' examination of the polygonal data to identify potential silhouettes and other feature outlines. At env^n the user has traced feature outlines and shaded regions around them, so the relevant voxels also have smarticle path information, colour details for relevant polygons and other associated data and statistics.

4.3.3 Situations

A situation is defined by task related environmental information available at any time step. There are two considerations for a situation: what a smarticle sees, in terms of its environment; and who a smarticle can see, in terms of team members. A smarticle's environment includes information, such as the number of samples in the voxel, from any voxel its path intersects, and any direct neighbours.

The set of all possible situations Sit a smarticle can be in is:

$$Sit = \{sit_0, sit_1, \ldots, sit_l\}. \tag{4.2}$$

A situation sit^i is dependant on the view of, or the information available about, the environment env^i at each time step i and the current team $team^j$ (if any):

$$sit^i = view(env^i, team^j). \tag{4.3}$$

The sequence of smarticle situations from a run is $(sit^0, sit^1, \ldots, sit^n)$ where $sit^i \in Sit$ for $0 \leq i \leq n$.

For example, a team of 100 smarticles have been assigned the task of creating stroke samples on a region of the surface, at time step $i = 0$, sit^0 identifies that a smarticle has its IDV initialised and knows that it has 99 team members. At $i = 1$, sit^1, the smarticles have all taken their first step, so each knows one more piece of information about their environment (from the sample information at the path step) and can communicate with any team members for any relevant information they might have, such as their position and velocity.

4.3.4 Teams

Teams of smarticles are required to complete certain tasks. These tasks are initial, low Level of Detail (LOD) identification of feature outlines, (Section 5.3) tracing feature outlines (Section 5.7), and groups of stroke samples that are created using flocking behaviours (Section 5.5).

Initial identification of feature outlines does not require inter-smarticle communication, the group is used to complete a task that is actually a collection of sub-tasks

(i.e. examining the initial polygonisation involves examining each polygon edge for crossing a silhouette, a discontinuity and a change in curvature), which could be completed by individuals working alone. A team is used in this context for speed of results.

Smarticles communicate with other team members through the manager. Communication is generally used to orient themselves relative to one another by sharing position and velocity information. Processing of this information is carried out at the individual smarticle level. For example, where flocking behaviours are to be applied, smarticles ask the manager where their local flockmates are (their positions) and what are their current velocities. The individual smarticles use this information to calculate the steering direction vectors due to the flocking behaviours. See Chapter 5.5 for full details and examples of using the flocking behaviours.

4.3.5 Actions

The action a smarticle will perform depends on the situation it is in and any behaviours that have been assigned to it. Actions a smarticle can perform are:

- move
- stop
- update, report
- communicate
- identify feature point

Behaviours provide IDV that are used for performing actions. For example, where a smarticle's action is to take a step, the direction, size and success of that step is dependant upon the behaviours. A path following, flow field following or wander behaviour determines the direction and the detail level specifies the size of the step, and the constrained or containment behaviour determines whether the step is to be included in the smarticle's path (or the path is terminated). Furthermore, the smarticle can examine the IDV from each path step to see if a feature outline has been crossed. The behaviours are described in more detail in Section 5.2. The method of examining the smarticle's path is covered in Section 5.6.

A smarticle's action can also be to update the blackboard. This action is triggered in response to certain situations. When a smarticle moves from one voxel to another, the previous voxel is updated with the path information of the entire subsection of the path that is contained within it. Other information that is updated relates directly to the steps of the smarticle's path, such as the history of sample and curvature information. This update also occurs when a smarticle has finished its assigned task.

The communication action is activated in group situations where a smarticle requires information about its team members, such as when flocking behaviours are used (Section 5.5)

The set of potential actions Act that a smarticle can perform is composed of all m actions available to it:

$$Act = \{act_0, act_1, \ldots, act_m\}. \tag{4.4}$$

During a run of the system a smarticle produces the sequence of actions $(act^0, act^1, \ldots, act^n)$ with $act^i \in Act$ for $0 \leq i \leq n$.

The move, stop and identify feature point actions are motivated by behaviours, which are explained in Chapter 5.

In the example in Section 4.3.1 above, the smarticle performed the move, stop and report actions.

4.3.6 Internal Data

A smarticle's internal data values (IDV) includes its assigned behaviour attributes, history, and identification of both the team it belongs to (if any) and the task it has been assigned. The manager sets the smarticle a task, which provides the starting position, velocity, and the appropriate behaviour information, which is in the form of IDV values, such as steering direction and termination criteria. Behaviours define the way in which actions are to be carried out. Identification of the team is necessary because certain actions are performed at the group level.

A smarticle collects data from investigations of its environment. This data includes field function value, gradient, mean and Gaussian curvature values and principal directions of curvature. As a smarticle moves through its environment it stores this information for all path steps.

The set of all possible IDV due to performing tasks is therefore:

$$Dat = \{dat_0, dat_1, \ldots, dat_m\}. \tag{4.5}$$

The current value of a smarticle's IDV dat^i at time step i reflects the task it has been assigned:

$$dat^i = \{task^j, team^j, history^i, behaviours^i\}. \tag{4.6}$$

The sequence of IDV resulting from a system run are $(dat^0, dat^1, \ldots, dat^n)$ where $dat_i \in Dat$ and $0 \leq i \leq n$. The task and team information does not change over time, rather it is dependant on the current task assignment from the manager.

A smarticle's history represents unique knowledge about its interaction with the environment, in the form of sample information taken at each step in it's path. Any data that is useful or relevant to other smarticles is periodically communicated to the manager for inclusion in the blackboard. For example, all path information is communicated from the smarticles IDV back to the manager for inclusion in the blackboard when the smarticles path changes voxel or when the task is complete. Other information the IDV store include the steering direction calculation method, or behaviour, (e.g. one of the principal directions of curvature, see Chapter 5.4 for a full list of the different steering directions available to smarticles); the number of steps in the smarticle's complete path; and the maximum step size, or detail level;

4.3.7 The Decision Function

The decision function (DF) relates the situation and IDV to the action that should be performed. The DF looks at the smarticle's situation and references relevant IDV and provides the necessary information to perform the action. If the action is to move (change the particles position and velocity) then the DF will provide the parameters for the movement. If the action is to report to the blackboard (through the manager) the function will identify the information to be reported.

An example of using the DF to identify an action would be where the smarticle's path is restricted to be within a certain range of its initial position. The distance from the initial path step position to the current one is calculated. If the distance

is within the specified range the smarticle should perform the action defined by the behaviour that guides its movement, i.e. take the next step. Whereas, if the distance is outside this range the action to be performed is to terminate the stroke and update the data structure. A full explanation of the termination criteria is discussed in the constrained behaviour Section 5.4.

The DF, f_{Ag}, relates the situation to the appropriate action:

$$act^i = f_{Ag}(sit^i, dat^i) \tag{4.7}$$

The example in Section 4.3.1 above illustrates how the decision function works by evaluating the smarticle's current situation and the proposed path step validity before initiating the relevant actions. For example the move action is taken when the proposed step has been identified not to satisfy any termination criteria. The stop action is dependant on the termination criteria, which also performs a report action. The report action is also instigated after a voxel change has been identified.

4.4 Conclusions and Discussion

Previous approaches to pen and ink rendering of implicit surfaces generally use particle based systems. The main drawback with such approaches is the requirement of proximity to a feature to identify it. Particles are, in general, unaware of their environment or each other, all particle calculations are carried out by the particle system. In this chapter, smarticles were presented as semi-autonomous agents that not only are aware of their surroundings but are also capable of exploring them and identifying feature outlines or interesting areas of the surface. The manager pre-

sented in the previous chapter (Chapter 3) interprets user requests to create tasks that are assigned to smarticles for completion. Tasks are carried out using smarticle assigned behaviours in the form of internal data values, covered in the following chapter (Chapter 5).

Smarticles benefit from being semi-autonomous agents in that they control their movements and store their results. It is up to the smarticle to decide what information should be reported back to the manager for inclusion in the blackboard (common data structure representing the environment). Future work could be aimed at providing smarticles with learning behaviours. This could identify further uses for the data collected by smarticles' exploration, and alter the selection of data that is deemed relevant.

One of the benefits of such a distributed approach is the potential to operate in a multi-processing environment. Although the smarticles were designed with this in mind, the agent engine Vigo does not have these capabilities. It would be an interesting avenue of future research to develop a multi-processing environment for the smarticles to operate in.

The contributions presented in this Chapter relate to the development of particles into agents that operate in object space and are motivated by goals to complete tasks.

Chapter 5

Behaviours

In this chapter the behaviours that smarticles (presented in Chapter 4) use to perform actions are described. Behaviours are introduced as a method of creating goal directed agents, or *smarticles*, which actively seek out features or interesting areas of the surface. The costly distribution method used in previous approaches is replaced with the method of using the initial low level of detail polygonisation to calculate initial surface positions. The information gained from the polygonisation identifies potential features or areas of interest that should be explored further.

In the previous chapter smarticles were introduced as agents in a MAS. Smarticles use behaviours to modify actions, and respectively their parameters, in order to complete tasks assigned by the manager. The behaviours are used to identify regions of interest, trace feature outlines and position smarticle paths (Note that the final appearance of strokes from smarticle paths is covered later in the rendering chapter (Chapter 6).

5.1 Introduction

Smarticles are based on particles, with the addition of behaviours, and framed in the context of a MAS. Smarticles investigate their environment using these behaviours to define forces for steering and to analyse their paths. Behaviours are dependant on sampling the 3D object space. *Sampling*, in this context, is the process of determining

Task		Behaviours
Initialise the Blackboard	⇒	Polygon processing
Investigate the environment / Create Smarticle Paths	⇒	Steering and flocking
Trace feature outlines	⇒	Trace features

Table 5.1: Relationship between tasks and behaviours.

the field function value and gradient for a particular point in space. Examination of the samples is used to identify interesting regions of the surface, i.e. where there are feature outlines or other details.

There are two types of samples referred to in this research. The first are surface samples, that have field function values close to the value of *iso*. The second type are more general and are called object space samples. These samples determine information about a particular region of space and do not necessarily adhere to the surface.

There are two contributions presented in this chapter. The first is a method for applying steering and flocking behaviours to agents to both generate stroke positions for and sample a 3D implicit model. The second is feature outline identification and tracing sharp junctions or discontinuities.

5.2 Behaviours and Actions

Agents' actions are defined, in part, by a set of behaviours, several of which can be active at any one time. Behaviours are part of a smarticle's IDV and are used as modifiers of actions, such as the steering direction when moving.

Behaviours are used to perform actions that relate to a smarticle's assigned task, see Table 5.1.

Behaviours are applied to individual smarticles and also to teams. Those applied at the individual level are: processing; steering; and feature outline tracing behaviours. Some tasks require more than one smarticle, or a team. Teams are used for flocking and sharp junction feature outline tracing. Feature line tracing requires both individuals and teams, silhouette tracing requires only a single smarticle, whereas sharp junction or discontinuity tracing requires a team. Similarly, the feature point identification action uses individuals for silhouettes and teams for identifying discontinuities. Teams are also used to create groups of strokes, in this case the user decides the number of members in the team.

During the pre-processing stage, the object is roughly polygonised using a continuation method (Chapter 2.2.3 [GWW86]). Polygon processing behaviours are then used to initialise the data structure and provide a low resolution illustration of the surface feature outlines.

5.3 Polygon Processing

As identified in Chapter 4, previous methods of distributing particles across a surface can be time consuming. High resolution polygonisation can also be time consuming for complex implicit surfaces, in many cases in order to correctly identify surface features ray tracing must be used [Gal05, SWG05]. This research, therefore, uses a low resolution polygonisation to provide an initial distribution of points on the surface. This is a faster method of obtaining a good coverage of the surface than relying on a WH distribution method. Smarticle initial positions are provided by the vertices

of triangles produced from the polygonisation. The vertices also provide initial data about the smarticle's environment such as gradient and curvature information. The data from polygon vertices identifies (to a certain extent) the behaviour of the surface inside the voxel. This information is used by the manager to make assumptions about the workload associated with processing the voxel.

A continuation method is used (Chapter 2.2.3) which also uses a voxel-based data structure, so the vertices are already associated with voxels. All of the data from the polygonisation is transferred to the blackboard which provides a good initial view of the environment.

After the rough polygonisation is performed a team of smarticles processes the resulting mesh to identify feature outlines such as silhouettes, changes in curvature, and discontinuities. Smarticles process the mesh by examining each polygon and the sample and curvature information for each vertex, similar to the method in [SWS05]. These smarticles use individual behaviours, they are treated as a team simply to complete the task.

5.3.1 Finding silhouette approximations

In this research the definition of a silhouette outline is where the dot product of the surface normal (normalised gradient) and the view vector is equal to zero:

$$\mathbf{n} \cdot \mathbf{v} = 0 \tag{5.1}$$

A smarticle computes this dot product for each of the polygon's vertices. Once calculated, the values are stored to avoid re-calculation. Similar to the method in [SIJ+07] the vertices are examined in pairs that represent polygon edges (Figure 5.1).

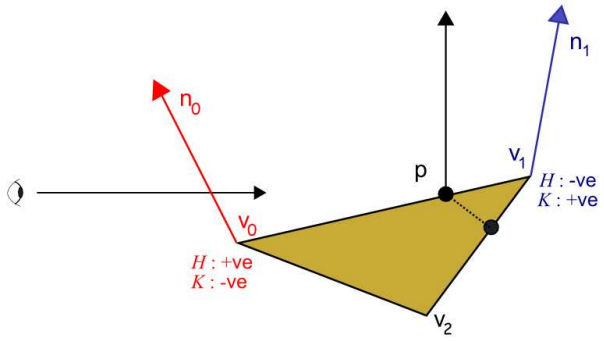

Figure 5.1: When triangle vertices identify a silhouette cross or a change in sign of curvature, linear interpolation can be used to approximate the point on the silhouette or where the curvature changes.

If a dot product for one vertex is negative and another is positive, this identifies that a silhouette edge has been crossed. The dot products at each of the vertices are used in an initial linear interpolation to evaluate the value of the interpolation coefficient, α:

$$\alpha = \frac{0 - \mathbf{n_0} \cdot \mathbf{v}}{\mathbf{n_1} \cdot \mathbf{v} - \mathbf{n_0} \cdot \mathbf{v}} \tag{5.2}$$

where $\mathbf{n_0}$ and $\mathbf{n_1}$ are the surface normals at vertices v_0 and v_1 respectively. The value of 0 is used as the silhouette is found where $\mathbf{n}.\mathbf{v} = 0$ as per Equation 5.1.

The interpolation coefficient, α, is then used in a second linear interpolation to approximate the position of the silhouette point, p:

$$p = (1 - \alpha)V_0 + \alpha\ V_1 \tag{5.3}$$

This rough approximation to the actual silhouette outline is used for low levels of detail. As higher levels of detail are required smarticles will trace the feature outline,

producing a more accurate silhouette.

5.3.2 Finding changes in sign of curvature

In the same manner as silhouette outlines are found, triangle vertices are examined for the possibility of identifying a change in the mean H or Gaussian K curvature of the surface. A change in sign indicates the change from a convex to a concave region of the surface (or vice versa). In the same manner as above, linear interpolation is used to identify the point of change across a polygon's edge. Equations 5.2 is changed to:

$$\alpha = \frac{0 - H^0}{H^1 - H^0} \tag{5.4}$$

where: H and K are interchangeable.

Equation 5.3 is still used, instead the resulting point p is the approximation to the point where the curvature changes.

In some circumstances changes in curvature can be treated as discontinuities. The change in sign of the curvature is checked using the following method to determine if it also identifies a discontinuity. If there is a discontinuity present in the field then the points are treated as such rather than curvature change points.

5.3.3 Finding Discontinuities and Sharp Junctions.

Discontinuities or sharp junctions can be identified by the information from pairs of vertices of the triangles, or edges. If a large angle between the gradients is identified the vertices are used to find a point on the edge that corresponds to the discontinuity, see Section 5.7.

5.4 Steering

Steering behaviours affect a smarticle in two ways: how it moves and how it stops. The movement stage uses concepts from Reynold's wander, flow field following and path following behaviours [Rey99]. Stopping a smarticle, or terminating its paths, is inspired in part by Reynold's concepts of containment and arrival behaviours [Rey99].

Steering behaviours are used to determine a steering force that produces a change in direction of a smarticle. The steering force is part of a smarticle's IDV, and the change of position is a result of the move action. Steering behaviours can be used individually or combined. The steering force is used as the acceleration because all smarticles in the system currently have equal mass ($m = 1$). The steering force $\mathbf{s_i}$ is used to update the smarticle's velocity $\mathbf{v_i}$:

$$\mathbf{v_{i+1}} = \mathbf{v_i} + \mathbf{s_i} \tag{5.5}$$

where $\mathbf{v_{i+1}}$ is the new velocity and $\mathbf{v_i}$ the current velocity, note that $0 \leq \mathbf{v} \leq \mathbf{v^{max}}$ where $\mathbf{v^{max}}$ is the maximum velocity of the smarticle, which is related to the level of detail. Higher levels of detail have lower maximum velocities to specify shorter steps, and vice versa.

The velocity is used to update its position p_i:

$$p_{i+1} = p_i + \mathbf{v_i} \tag{5.6}$$

where p_{i+1} is the next position and p_i is the current position.

5.4.1 The Wander behaviour

The wander behaviour can be used for a smarticle to take steps in (slightly constrained) random directions, see Section 2.5.2. At each path step the steering force has a truncated random vector added to it. Adding a truncated random vector ensures that the changes in the steering force are incremental rather than potentially being a completely different direction at each path step. This produces a more smooth path avoiding erratic direction changes. The initial steering direction can be determined from a random vector or it can use any of the flow field following or path following behaviours mentioned below.

In Figure 5.2 smarticles were given initial positions provided by the silhouette and initial steering directions use a horizontal path combined with a random component. Subsequent steering directions were obtained by adding a truncated random vector. Two smarticles start from each initial position on the silhouette and trace their paths in both directions, ME on the front, or visible, side of the surface and the back. Although both are shown here for illustration, it is not necessary to calculate them for parts of the surface that aren't visible. The paths in this image were constrained so that the surface normals of the initial and current path step are within sixty degrees of each other (for constraint details see Section 5.4.4).

5.4.2 The Flow Field Following behaviour

Flow field following is used to calculate the steering behaviour through definition of a vector field. A flow field is a region of space where direction vectors can be evaluated according to some property of the environment. For example, the first principal direction of curvature can define a flow field, see Figure 5.3. At any point

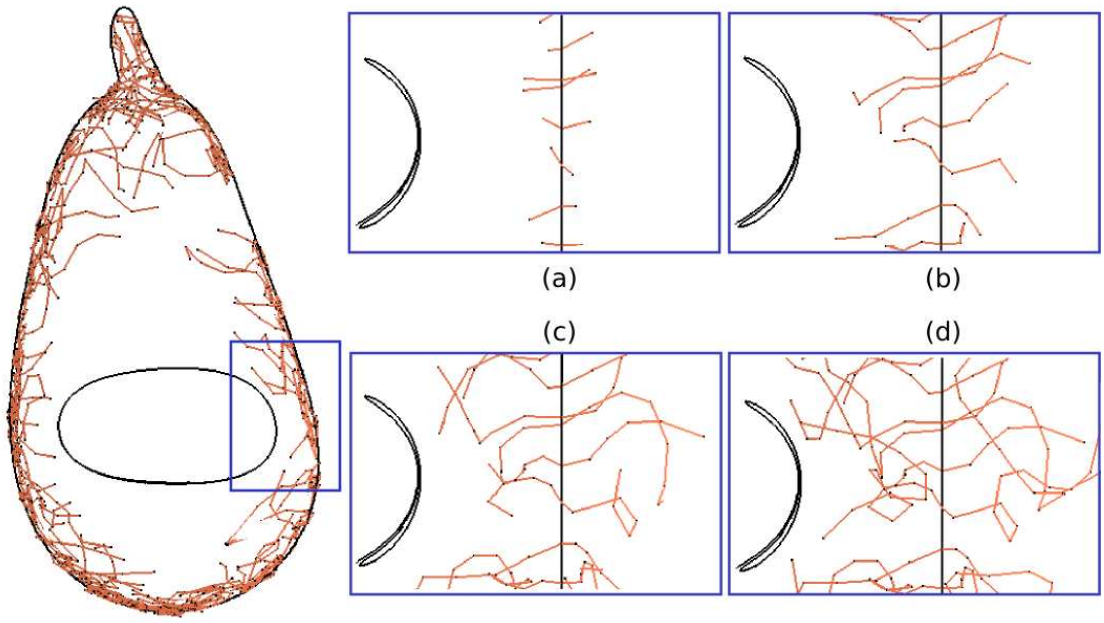

Figure 5.2: The wander behaviour. **(a)** The first step in the smarticles' paths. **(b)** Step three. **(c)** Step six. **(d)** The smarticles paths completed.

in space a smarticle can determine the first principal direction of curvature and use it as the steering direction. As with the previous example (wander Figure 5.2) the paths start from the silhouette with two smarticles each tracing the front and the back of the object. The paths are again constrained to be within sixty degrees of the initial positions. Note also that a smarticle's path is terminated in regions where paths already exist, in Figure 5.3 **(d)** not all paths are of the same length.

The steering force $\mathbf{s}$ is calculated using the flow field vector $\mathbf{d}$ and the velocity vector $\mathbf{v}$:

$$\mathbf{s} = \mathbf{d} - \mathbf{v} \tag{5.7}$$

The flow field vector $\mathbf{d}$ is defined according to the effect desired by the user. The

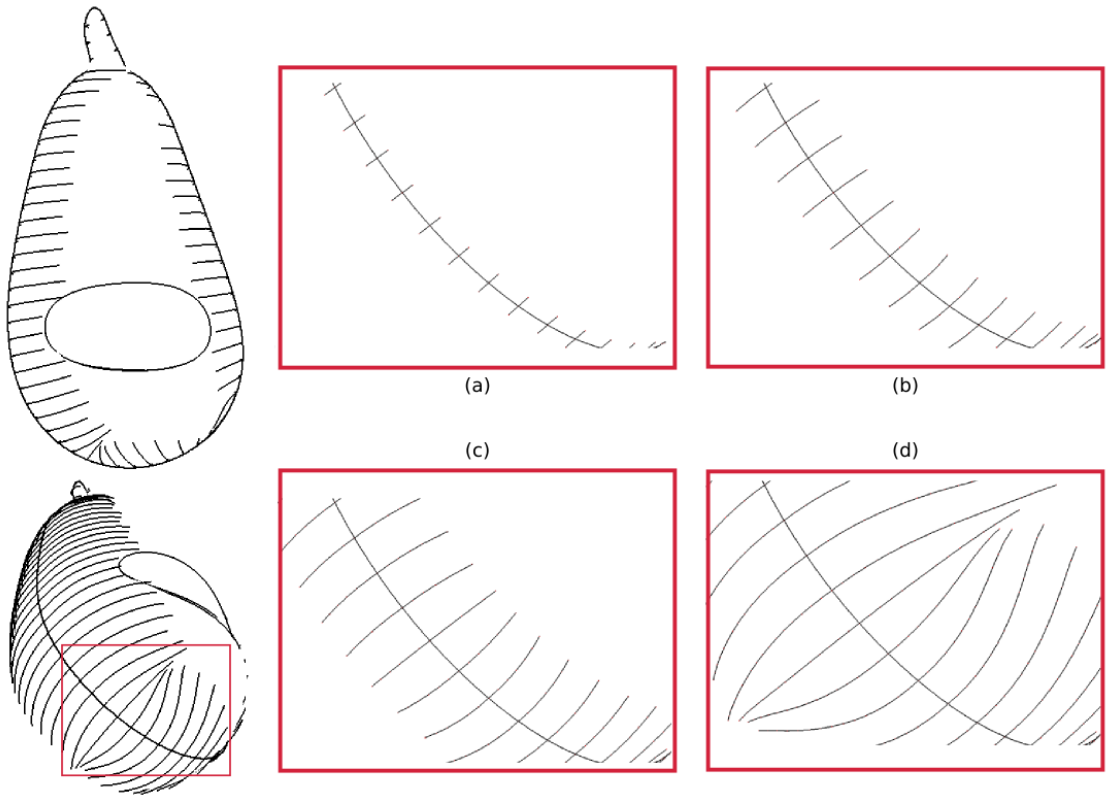

Figure 5.3: The flow field following behaviour using the first principal direction of curvature. **(a)** The first step in the smarticles' paths. **(b)** Step three. **(c)** Step six. **(d)** The smarticles paths completed.

flow field can be in the direction of either of the principal directions of curvature or the contour direction.

Changes in the flow field are typically small or smooth enough that no constraints are necessary to ensure a smooth path. Discontinuities in the implicit field are the main areas where the flow field changes may not produce a smooth (or desirable) path.

5.4.3 The Path Following behaviour

The path following behaviour is primarily used to define globally oriented vertical horizontal and diagonal steering forces. This path is called a *steering path* to avoid ambiguity with the *smarticle's path*. A (horizontal, vertical or diagonal) plane is defined (Figure 5.4 **(a)**) using the smarticle's initial path position, or starting point. The normal to this plane $\mathbf{n}$ is defined using the steering direction $\mathbf{s}$ and the view vector $\mathbf{v}$ to ensure that it is correctly oriented in terms of the user:

$$\mathbf{n} = \mathbf{v} \times \mathbf{s} \tag{5.8}$$

The steering path is then defined as the intersection of this plane and the surface (Figure 5.4 **(b)**). The smarticle's path is then constrained to be within some tolerance region τ of this steering path (Figure 5.4 **(c)**). If the smarticle's next step is estimated to be outside of this region, then the smarticle's position is corrected to keep the smarticle's path within the tolerance range. This is achieved by applying a force $\mathbf{c}$ that is equivalent to projecting the point back into the region, p_i'':

$$\mathbf{c} = ((\mathbf{n} \cdot \mathbf{u}) - \tau)\mathbf{n} \tag{5.9}$$

where $\mathbf{u} = p_{i+1} - p_0$, p_{i+1} is the current uncorrected position and p_0 is the initial position on the smarticle's path. τ is the radius around the path that represents the tolerance region, which is dependent on the level of detail.

Diagonal paths are created using a combination of the horizontal and vertical steering directions. Similar conditions and constraints were applied as for Figures 5.2 and 5.3. Note that in the figure the diagonal path can result in some interesting

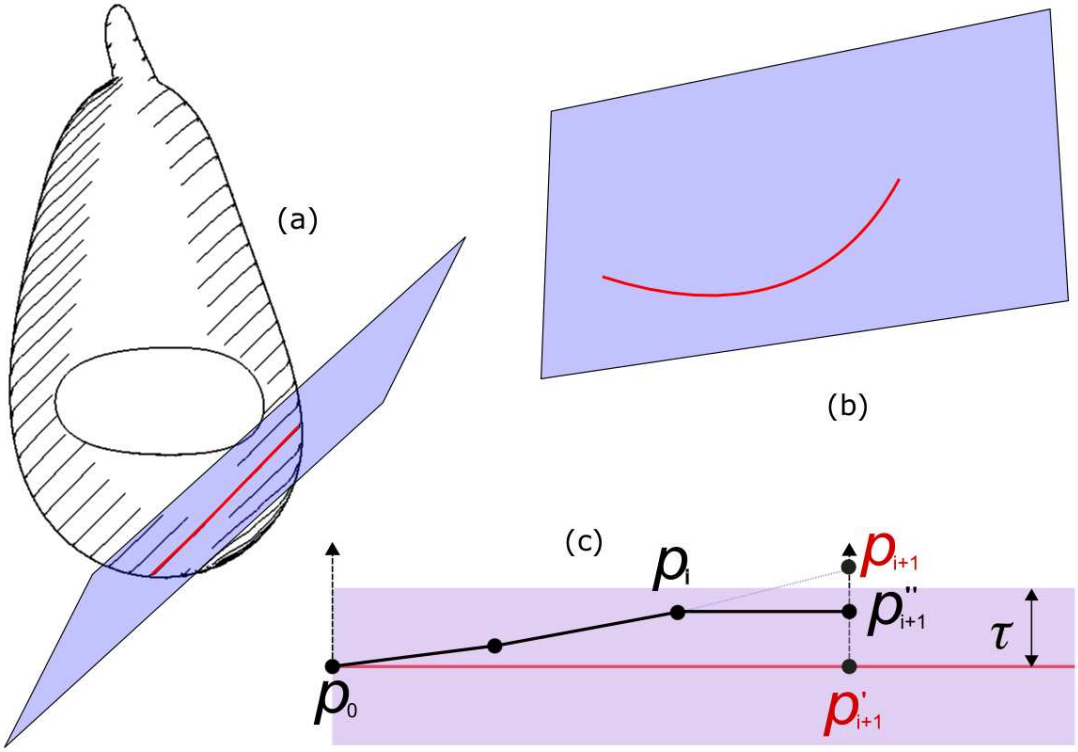

Figure 5.4: **(a)** The intersection of a diagonal plane (blue rectangle) and the surface (pear) defines the smarticle's path (red curve). **(b)** the smarticle's path seen from a different angle. **(c)** If the smarticle's path strays outside the tolerance region τ it is corrected.

results when the image is viewed from an angle other than the one the path was defined for.

5.4.4 The Constrained Behaviour

A new **constrained** behaviour pattern has been developed for this research. It uses concepts from Reynolds original arrival and containment behaviours [Rey99], to constrain a smarticle, or a group, within a particular region.

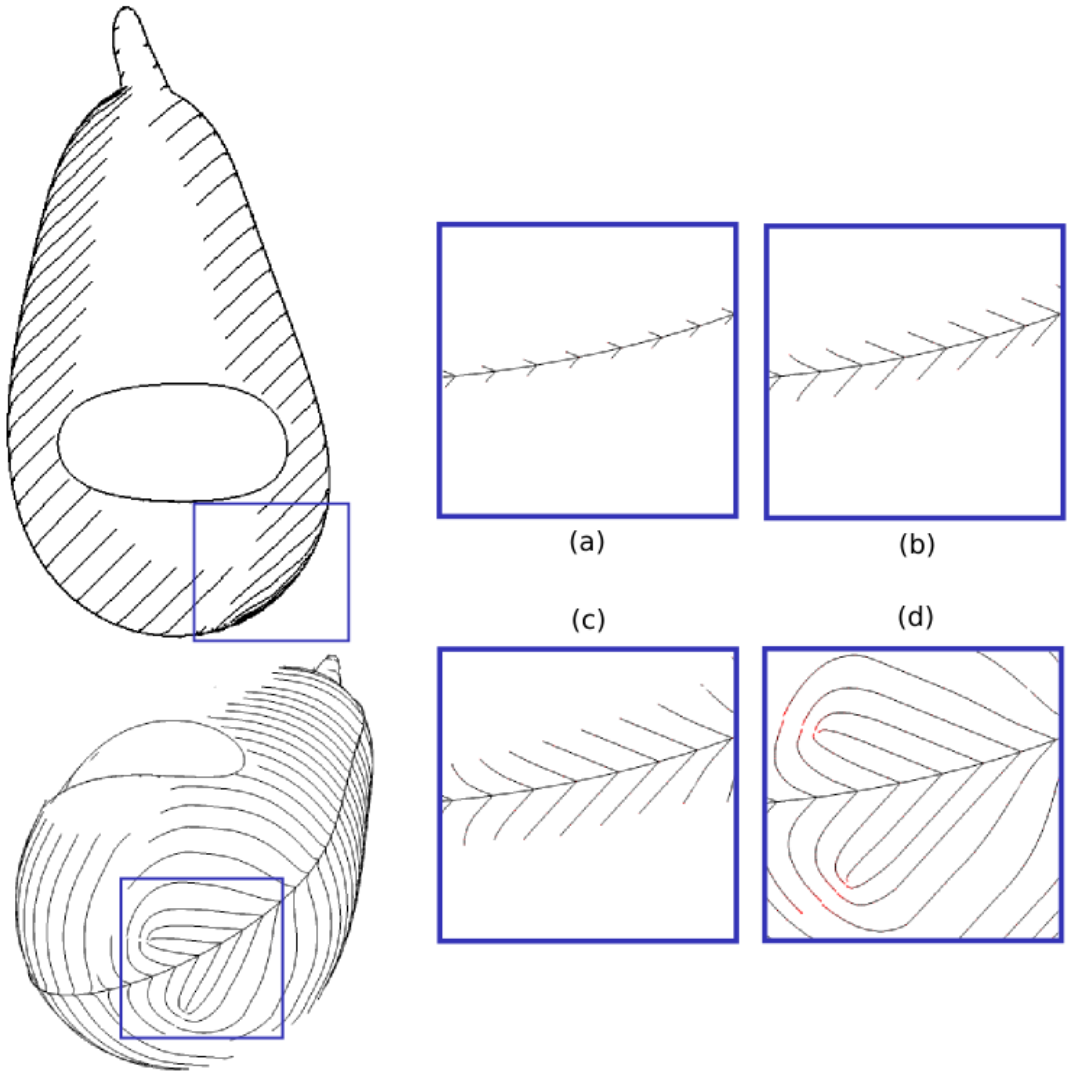

Figure 5.5: The path following behaviour using the intersection of a horizontal plane and the surface as the path. **(a)** The first step. **(b)** Step three. **(c)** Step six. **(d)** The completed path

At any path step, the smarticle's current position can be used to evaluate surface properties for constraints. If there is a change in the relevant property then the path will not include the current step. Where the smarticle is constrained to lie

in a convex or concave region, the Gaussian and mean curvature of the surface is calculated; if the sign changes (indicating a change in the convexity or concavity), the path is terminated. Similarly, if a discontinuity in the field is detected the path is terminated.

The path can also be constrained by the curvature from its starting point. The surface gradient at the smarticle's current position is compared with the gradient at its initial position. If the angle between the two vectors is greater than desired then the path is terminated without inclusion of the next step. Similarly, if the containment refers to a region around a feature line, the distance between the initial point on the feature line and the current position is the measurement; where the distance is measured by the lengths (or the number) of the steps. The choice of containment is generally left to the user.

In Figures 5.2, 5.3 and 5.4 above, all of the smarticle paths are constrained so that the angle between the surface normal at the initial position and the normal at the current position are within sixty degrees. The smarticle paths are simply terminated when they come into contact with an unknown or undesirable feature or element, rather than using a steer to avoid strategy.

Where a smarticle's next step is identified as leaving a region, the result is smaller step sizes rather than a change in steering direction. A smaller step size allows the final path step to more closely approach the actual change in region. This is analogous to Reynolds **arrival** behaviour [Rey99], where a particle slows down as it approaches its goal. In the context of smarticles it is achieved by using a binary subdivision method. Where the next point is identified as being outside of the region, a binary subdivision of the step is performed until a point inside the region

is obtained. This process is repeated until an acceptable level of detail is achieved or a number of subdivisions has been performed. In Figure 5.6, the smarticle's next position p_{i+1} is determined to be outside of the valid region. The first level of binary subdivision of the vector results in point p'_{i+1} (which is corrected to the surface). This point is within the region but the level of detail has not yet been reached. The second level results in point p''_{i+1} which also satisfies the level of detail measurement. This subdivision is not guaranteed to converge on the feature outline, rather it is used to obtain a closer approximation than merely terminating the stroke sample. It is also only used where a higher level of detail is required.

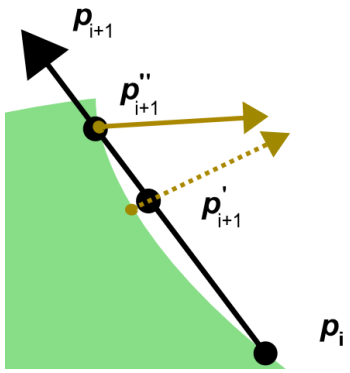

Figure 5.6: When the smarticle's next position is outside the region, binary subdivision of this step is performed to more closely approximate the edge of the region.

Where a smarticle is constrained to lie on the surface its position must be corrected to lie on the surface (using the general surface correction method, see Equation 2.23 ($F_i = (f(\mathbf{x}_i) - iso)\nabla f(\mathbf{x}_i))$).

5.4.5 The Seek behaviour

The seek behaviour is used to create a stroke in an orientation requested by the user, rather than using any of the methods outlined above. To create a user defined stoke orientation the user specifies two points, a start and a target point and uses the seek behaviour to create a path between the two points. This path can be in any direction the user wishes, the only constraint is that the path remains on the surface.

The user can also create a group of strokes from two traced seek paths. For example, in Figure 5.7 (centre) the user created both of the upper (red) and lower (blue) strokes using the seek behaviour. With both of these selected the user requested to "Create strokes between" them. In Figure 6.12 (top), the train has a group of strokes drawn on the engine.

Figure 5.7: (**Left**) A group of strokes is created using user placed paths. (**Centre**) Close up of the user created paths (red) and (blue). (**Right**) A second set of strokes created inside the bite of the pear.

5.5 Flocking

Flocking behaviours are applied to a group of smarticles. The effects are calculated using information about other members of the group. The results, however, vary for each individual. These behaviours are: **separation**, where flockmates steer to avoid each other; **alignment**, where flockmates align their velocities to stay in formation; and **cohesion**, where flockmates steer to stay together as a group.

Flockmates are defined by a neighbourhood. In this research, the neighbourhood is defined as either a geographical, spatial region (as with Reynolds work in [Rey87]), or as an area of the surface that is delimited by surface details, such as changes in curvature or distance from the initiating point.

The concept of a *neighbourhood* is an important one when dealing with flocks. The neighbourhood as defined by Vigo (see Section 2.6.6) uses a hierarchical spatial partitioning method. This identifies the region around any individual where valid neighbours are found. This reduces the range of a search for flockmates, so that an exhaustive search of all individuals to determine proximity is not required.

As with the steering behaviours, a final surface correction step is applied to the result of the flocking behaviours to ensure that the smarticle's paths remain on the surface, where appropriate.

Applying *both* steering and flocking behaviours is carried out in two stages. First, the steering behaviours calculate an interim position and velocity. Next, the flocking behaviours are calculated using the interim position and velocity information for all relevant smarticles. This results in the final position and velocity for the next step in the smarticles path.

5.5.1 Separation

The separation behaviour uses the positions of neighbours to create a force that moves the smarticle in the opposite direction, or away from, its neighbours.

The separation behaviour $\mathbf{f_i^s}$ for smarticle i is calculated as:

$$\mathbf{f_i^s} = \sum_{j=0}^{n, j \neq i} \frac{p_i - p_j}{|p_i - p_j|^2} \tag{5.10}$$

where n is the number of smarticles in the neighbourhood, p_i and p_j are the i^{th} and j^{th} smarticles' positions after both have had their individual steering behaviours been applied.

5.5.2 Alignment

The alignment behaviour calculates the average velocity of flockmates and uses this force to move the smarticle so that ultimately all flockmates velocities are the same.

The alignment behaviour $\mathbf{f_i^a}$ for smarticle i is calculated as:

$$\mathbf{f_i^a} = \frac{\sum_{j=0}^{n, j \neq i} \mathbf{v_j}}{n} \tag{5.11}$$

5.5.3 Cohesion

The cohesion behaviour calculates the average position of the flockmates and moves the smarticle closer to them.

The cohesion behaviour $\mathbf{f_i^c}$ for smarticle i is calculated as:

$$\mathbf{f_i^c} = \frac{\sum_{j=0}^{n, j \neq i} p_j}{n} \tag{5.12}$$

5.5.4 Flocking

The final flocking vector is calculated using the separation $\mathbf{f_i^s}$, alignment $\mathbf{f_i^a}$ and cohesion $\mathbf{f_i^c}$ behaviours as:

$$\mathbf{f_i} = \omega_s \ \mathbf{f_i^s} + \omega_a \ \mathbf{f_i^a} + \omega_c \ \mathbf{f_i^c} \tag{5.13}$$

where ω_s, ω_a and ω_c are the weights for separation alignment and cohesion respectively.

The following images have been created to illustrate the effects of the flocking behaviours on smarticle paths. In each of the images a group of twenty five smarticles were created (using triangle vertices from the polygonisation stage) around a user identified point (illustrated as a blue square in the images). Note that a low level of detail has been used in all of the images. A low level of detail means that the step sizes are relatively large. The density of smarticle paths in a region is calculated using distances of step positions to one another.

Using the separation behaviour (Figure 5.8) the smarticles' paths spread out and move away from one another. The smarticle's initial velocity uses the separation vector calculated for the initial positions. The behaviour weights are $\omega_s = 1.0$, $\omega_a = 0.0$ and $\omega_c = 0.0$ to illustrate the singular effect of separation. Note that some smarticles' paths are terminated in dense regions, whereas others start to spread over the surface. This behaviour is useful for examining a region of the surface that was missed by the low level of detail polygonisation.

With the alignment behaviour (Figure 5.9) the smarticles' velocities are altered, over the course of their path, so that they are all the same. In this example the initial

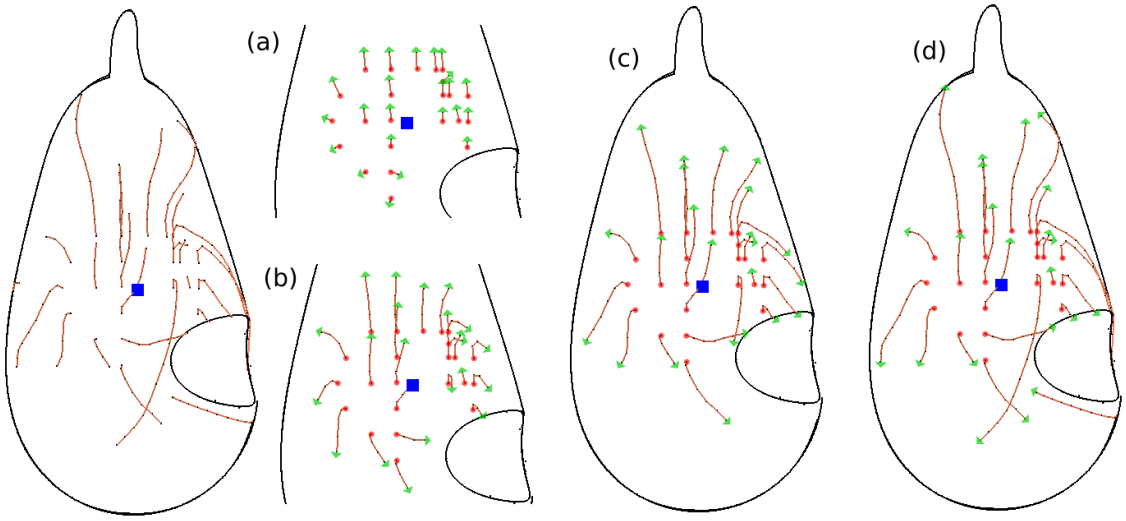

Figure 5.8: The separation behaviour causes smarticles to move away from each other. **(b)** Step three. **(c)** Step six. **(d)** The completed path

velocities were provided using a horizontal path on the front side of the pear moving from the starting point towards the centre of the object. The behaviour weights are $\omega_s = 0.0$, $\omega_a = 1.0$ and $\omega_c = 0.0$

In Figure 5.10 the effects of the cohesion behaviour are illustrated. Using this behaviour all of the smarticles converge on the same, central, position. The initial velocities are provided using the cohesion behaviour on the initial positions. The behaviour weights are $\omega_s = 0.0$, $\omega_a = 0.0$ and $\omega_c = 1.0$. Only six steps were required to show the effects of this behaviour, as the smarticles' paths are terminated when a region is too densely populated.

The effects of these behaviours on strokes are illustrated in Figure 5.11. For each of these images a small flock of size six was chosen where there was no communication between the current flock (black lines emphasised with red) and neighbouring flocks (grey lines). In Figure 5.11 **(a)** no flocking behaviours were applied, the lines result

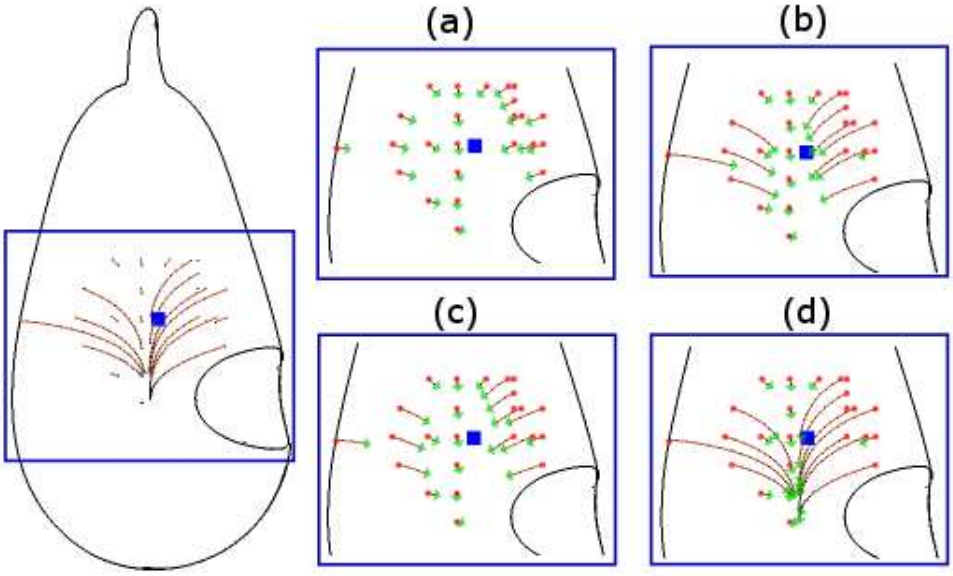

Figure 5.9: The alignment behaviour causes all smarticles to come into alignment with each other. **(b)** Step three. **(c)** Step six. **(d)** The completed path

from the individual steering direction, which was defined to start on the silhouette and proceed in a horizontal direction. Here you can see that although the steering direction of each stroke is the same, qualities of the surface, such as curvature, introduce minor variations in the stroke sample. In Figure 5.11 **(b)** the alignment behaviour was given full influence on the strokes, with no weighting given to separation or cohesion. Although the difference between this and the image in **(a)** with no flocking is small there is notable change due to the alignment of the smarticle's velocity. Figure 5.11 **(c)** illustrates the effect when the separation behaviour is used to the exclusion of the others. As the smarticles progress their paths move apart and create a spreading effect. Similarly, in 5.11 **(d)** the cohesion behaviour is dominant and the smarticle's paths ultimately converge.

Note that the initial velocities are provided by the path following behaviour with

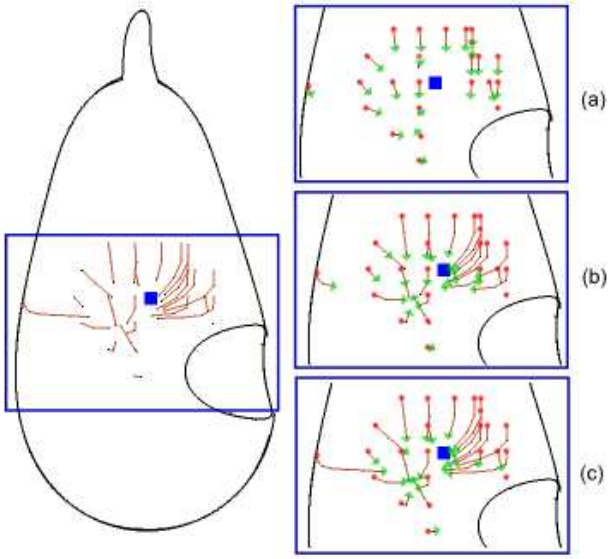

Figure 5.10: The cohesion behaviour causes all smarticles to move towards each other. **(b)** Step three. **(c)** Step six, which is also the completed path.

a horizontal path. The subsequent positions, however, are not constrained by the path, as this would terminate the strokes when they leave the allowed path region.

5.6 Finding Feature Outline Points

A smarticle's IDV are used to create a better understanding of its environment. The information from the smarticle's path can be used to identify missed voxels or untraced feature outlines (or details), that were not found by the low level polygonisation.

Rough polygonisation can miss small scale details in the field, see the sampling discussion in Chapter 2.2.3. Smarticles sample at a more refined rate than the initial polygonisation, and as such can identify details or features of the object that were

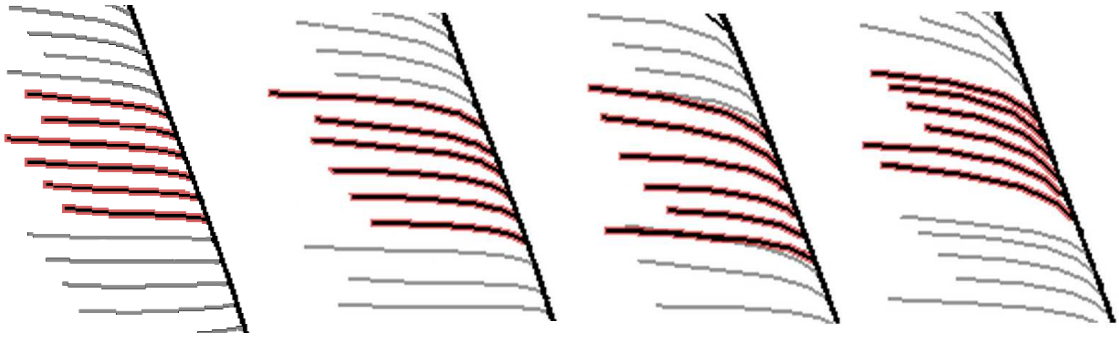

Figure 5.11: **(a)** No flocking: The smarticle's paths as they appear from only individual steering behaviours. **(b)** Alignment: The weighting for the alignment behaviour is set to one and the others are set to zero. **(c)** Separation: The paths tend to move apart from one another. **(d)** Cohesion: The paths tend to move together.

previously missed. A smarticle can find itself in a voxel that has not been identified to contain part of the surface. When this situation occurs the voxel is processed and flagged as containing part of the surface, see Chapter 3.

At each step of a smarticle's path, the sample and curvature data is examined to identify if a feature outline, such as a silhouette or sharp junction, is encountered. When a feature point is identified it is compared to other feature points in the vicinity to ascertain if this feature has been identified before. If it has not, then the outline is traced using the behaviours described in Section 5.7.

5.6.1 Identifying Silhouette Points

A smarticle uses gradient information to determine whether a triangle crosses a silhouette outline. A point on a silhouette outline has $(\mathbf{n}.\mathbf{v}) = 0$ (Equation 5.1). At each point in a smarticle's path the value of the dot product is calculated. If this value changes sign, then the path step has crossed a silhouette. The current

path step is then used to identify if a new silhouette has been found. First, a copy of the current path step is constrained to lie on the silhouette using Equation 2.24 from Chapter 2.4.1. The silhouette will then be traced if it has not been previously identified.

5.6.2 Identifying Points on a Discontinuity or Sharp Junction

Many feature outlines of the surface are defined as areas where the surface is not smooth. This means that there is a discontinuity in the implicit definition of the model. Discontinuities in implicit fields imply that the gradient is not continuous or everywhere differentiable. Therefore, this algorithm uses surface samples that straddle the junction, which avoids problems that can occur precisely *on* the discontinuity. The straddle points are kept close to the discontinuity and the resulting points are thus described merely as approximations.

For each step of a smarticle's path, the positions before and after the step are tested using a straddle function $t(p_i, p_i')$. The straddling function uses a disparity in the angle between the gradients at the two points to identify if points straddle a discontinuity. Where this test identifies straddling points, the two positions are saved p_{left} and p_{right} (the assignment of left and right is arbitrary).

A variation of the CSG junction finding method [WvO97] for approximating the intersection of two implicit contours is used. The straddling function identifies points where the angle between their gradients is larger than some threshold (see Figure 5.12 Left). The threshold is determined as being a potential limit of the gradient disparity that would be expected from a smoothly blended surface (as with the angle between the gradients at points p_{left} and p_{f0} in Figure 5.12 Centre). First, the midpoint of

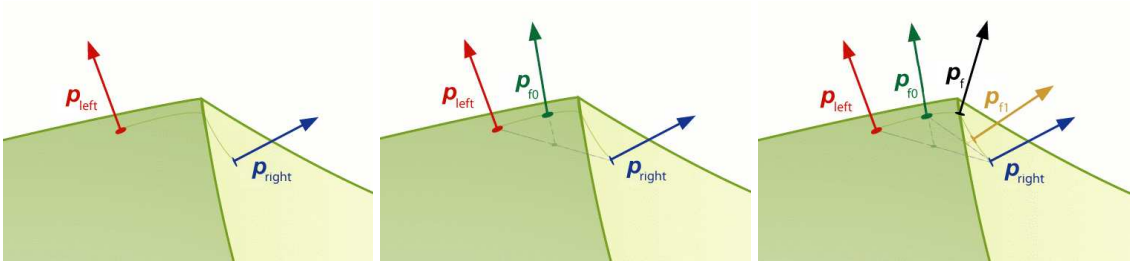

Figure 5.12: **Left**: p_{left} and p_{right} have a large angle between their respective normals, identifying a discontinuity or abrupt blend between their positions. **Centre**: *Straddle* points are used to converge on it. **Right**: The process is continued until a certain level of accuracy is reached.

these two points is found $p' = 1/2(p_{left} + p_{right})$ and corrected to lie on the surface p_{f0}. The angle between the gradients at points p_{left} and p_{f0} are consistent with a smoothly blended surface. The angle between the gradients at points p_{f0} and p_{right} identify that the feature lines in between these two points. Points p_{f0} and p_{right} are used as the straddling points and the process is repeated until a certain level of accuracy is reached and a point on the feature outline can be approximated.

Ultimately p_f approaches the discontinuity. Note that in the BlobTree the gradient is computable everywhere, although discontinuous. If the gradient is undefined, the level of accuracy is the means of achieving a close approximation to the discontinuity.

5.7 Tracing Feature Outlines

Guptill, in his book Rendering in pen and ink [Gup76], states *"actual objects have no true outlines, no definite edges or profiles bounding them that appear as lines"*, he points out that outlines appear due to differences in aspects such as colour, texture and lighting. The human perceptual system, however, finds it very easy to per-

ceive objects as if they did indeed have such outlines. These outlines are, therefore, instrumental to conveying shape and form.

Feature outlines like silhouettes are view dependant, others are aspects of a model that delineate where there is a change in colour, texture, curvature or shape. The feature outline tracing behaviour described in this section traces feature lines due to abrupt changes in curvature or shape of the model, these can also be described as sharp junctions.

An initial set of points identify a part of an outline. The feature is then traced to produce an outline that will be used for rendering. The **silhouette tracing** behaviour is based on the method of extracting silhouettes from [BH98] and [FJW$^+$05]. The **dual tracking algorithm** introduced in [FJW$^+$05], as part of this research is used to trace lines following certain view-independent features on the surface. Feature lines are determined through the use of a *straddling* function $t(p_{left}, p_{right})$, which returns true if points p_{left} and p_{right} are on opposite sides of a feature and false otherwise.

5.7.1 The Dual Tracking Algorithm

As mentioned in Section 5.6.2, straddle points are used to find a point that closely approximates a discontinuity. Feature lines are constructed by tracking the discontinuity with these straddle points. This means that the feature line that is traced will not be guaranteed to lie *precisely* on the discontinuity but is an approximation of it. The closeness of the straddling points means that the approximation is good enough for renderings of this style.

The dual tracking algorithm consists of two parts. The first is concerned with

finding the approximation to a single point on the discontinuity or sharp junction, which is covered in Section 5.6.2. This determines a point p_f on, or close to, the feature. Initial straddle points can be identified by the pre-processing stage, or from smarticles' paths as they explore the model. The second part tracks the feature to create a chain of points that can represent the outline.

Tracking the feature line

After finding an initial approximation point for the discontinuity, the entire feature is traced. This involves estimating the direction of the feature and stepping in that direction before performing the surface corrections and ensuring that the straddle points are kept within a reasonable distance of the discontinuity, see Figure 5.13.

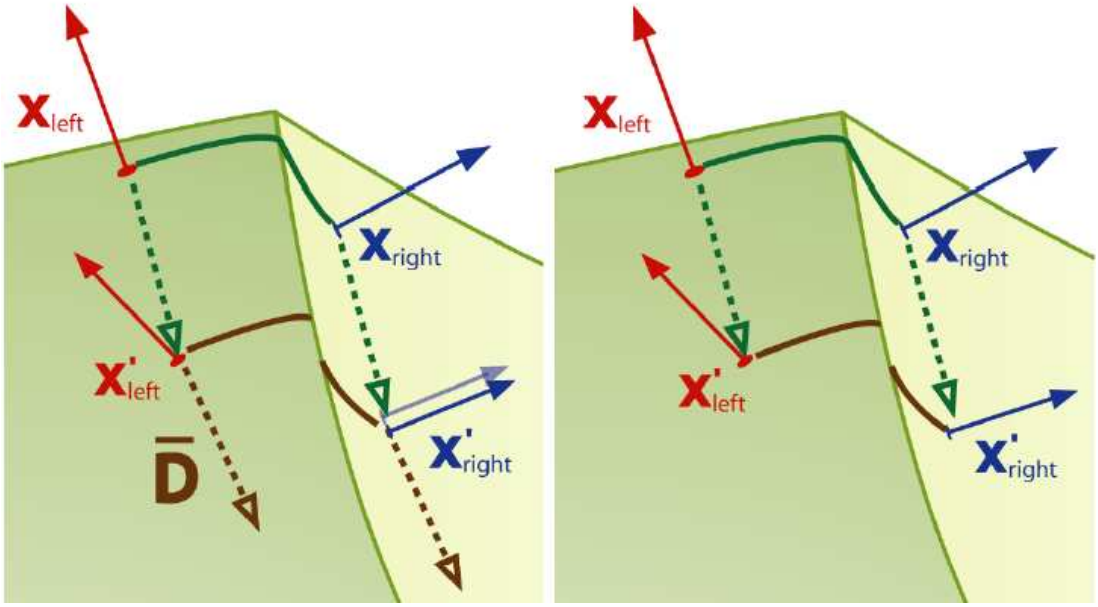

Figure 5.13: **Left**: The direction of the feature is estimated and the straddle points are stepped in that direction. In order to counteract the racetrack problem, after each step the straddle points are constrained to lie on a plane perpendicular to the feature line estimation. **Right**: The resulting stepped straddle points.

An estimate of the feature line direction is defined as the cross product of gradients at points (p_{left} and p_{right}):

$$\bar{D} = \nabla f(p_{left}) \times \nabla f(p_{right}) \tag{5.14}$$

New straddle points are found by moving in direction $\bar{D}$ and then correcting the points back onto the surface using Equation. 2.7. In addition to this, a straddle-corrector $\bar{W}$ is used to keep the straddle points close to the feature line. $\bar{W}$ defines a correction back towards the feature line as:

$$\bar{W} = \frac{p_f - p_{side}}{\|p_f - p_{side}\|} * (\|p_f - p_{side}\| - \kappa_{dist}) \tag{5.15}$$

where $side$ is $left$ or $right$ and κ_{dist} is the desired distance from the feature point. New points p'_{side} are then calculated with:

$$p'_{side} = p_{side} + \bar{D} + \bar{W} + \bar{U}_{surface} \tag{5.16}$$

The Racetrack Problem

In general, $\bar{D}$ is a good approximation to the direction of the feature line. Where the feature line is curved, however, a problem can occur. This problem is analogous to the situation on a *racetrack* where lanes on the inside are shorter than lanes on the outside. To overcome this problem, one straddle point is corrected to lie in the plane defined by the vector $\bar{D}$ Figure 5.13 and the other straddle point. This means that the straddle points are always on a plane perpendicular to the direction of the discontinuity, they will not be out of step, i.e. where one smarticle is ahead of the other.

Maintain Straddle Points

A second problem that may arise is where the displacement $\bar{D}$ can result in one of the straddle points crossing the feature line (detected using $t(p_{side}, p'_{side})$)(see Figure 5.14). This means that both straddle points would be on the same side of the feature rather than straddling it. In this case, the point p'_{side} is corrected back to the original side using the vector defined by $d = \pm(p_{left} - p_{right})$, where the direction used is dependent upon whether the left or right smarticle has crossed the feature line.

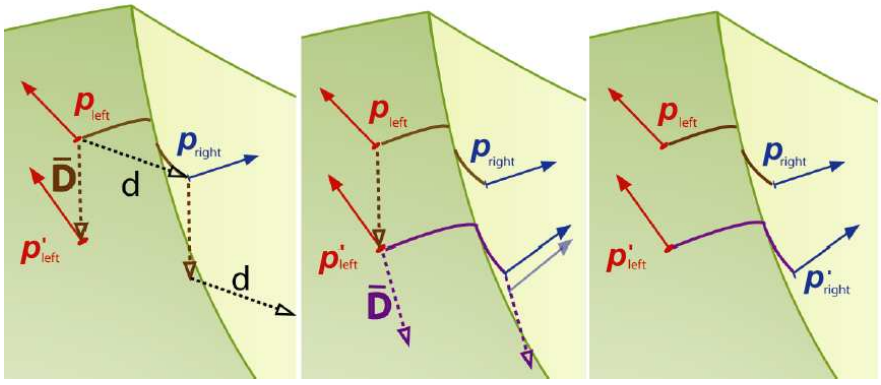

Figure 5.14: **Left**: The step direction takes one of the straddle points to the opposite side of the feature. **Centre**: The straddle points are constrained to the plane perpendicular to the feature line approximation. **Right**: The resulting straddle points.

In both of these situations, smarticles are corrected to lie on the surface using equation 2.23 and to be close to the feature line as a final step, using Equation 5.16. As points on the feature line p_f are calculated, their positions are chained to create the feature line.

5.7.2 The Silhouette Extraction Behaviour

Initial points can be identified through the polygon pre-processing stage or through analysis of a smarticle's path. The method of extracting a silhouette presented in [FJW+05] is used here to trace a silhouette, see Chapter 2.4.1. The steering direction is determined using Equation 2.21 which estimates the direction of the silhouette. Constraints are applied in the form of the surface and silhouette correctors Equations 2.22 and 2.24 respectively. The path is terminated when the silhouette is lost or where it loops.

5.8 Conclusions and Discussion

The introduction of smarticle behaviours was inspired by one of the drawbacks of previous particle based systems: achieving an adequate coverage of particles to detect features. Features (in particle systems) are detected through particle movement and proximity, for example crossing a silhouette. Steering behaviours were introduced to encourage the smarticles to actively seek out an area of interest. With the help of the manager (Chapter 3) smarticles can also identify surface voxels that need to be sampled further in order to determine if a feature is present.

The dual tracking algorithm does not specifically look for junctions or corners. In practice, most junctions are found and traced through separate smarticles identifying parts of the feature outline.

There are potentially a number of other methods of deriving steering directions to direct particles towards features that could be advantageous. An investigation into such potential steering direction calculation methods would be an interesting

area of further research.

The contributions of this chapter are:

1. Identify and trace feature outlines due to discontinuities in the field or abrupt blends between primitives.

2. Applying steering and flocking behaviours to agents to both sample and render the implicit surface.

3. Creating stipples from wander behaviour.

4. User created stoke directions (not constrained by surface properties, such as principal directions of curvature or contour).

z

Chapter 6

Mixed Rendering

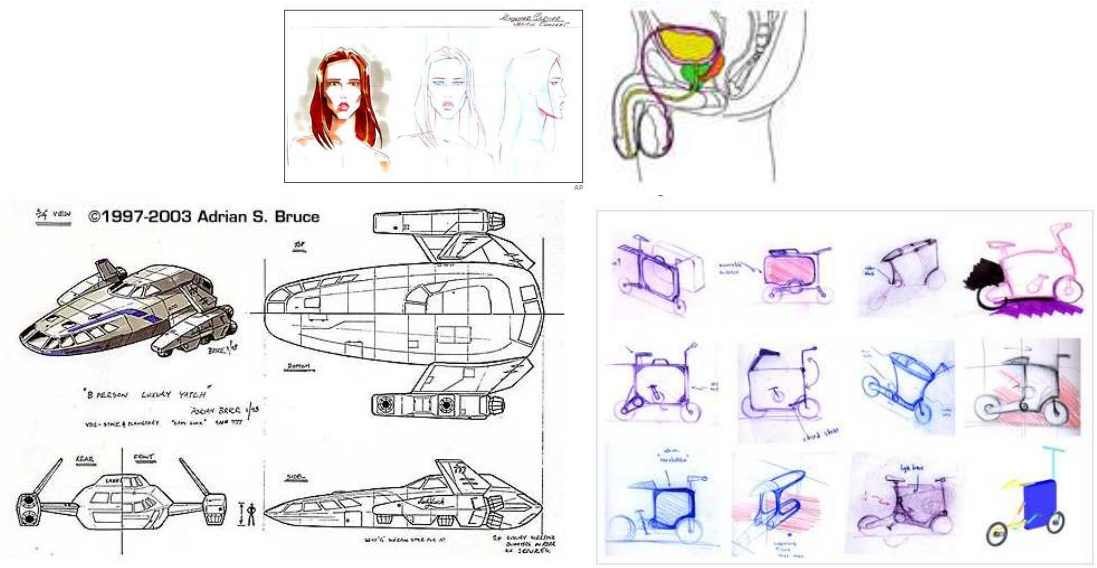

Figure 6.1: (**Top Left:**) Jennifer Garner from "Animated Alias". (**Top Right:**) Medical illustration from http://www.prostatitis.org/. (**Bottom Left:**) Courtesy of Adrian Bruce (http://www.artandtechnology.com.au/index.html). (**Bottom Right:**) Briefcase Bike (http://www.coroflot.com/.

6.1 Introduction

In many fields such as design, engineering and medical illustration mixed styles of rendering are commonly used for various purposes. Smarticles can be used to render surfaces using a combination of styles such as short and long strokes, stipples, and using colour accents (or highlights). Using strokes or stipples for rendering produces images in a pen-and-ink or sketching style, when colour is introduced the results are

reminiscent of concept art, see Figure 6.1.

In this chapter, the methods and results of using smarticles to create strokes, stipples or coloured polygons are presented. The adaptation of the techniques from [FJW+05] for use with smarticles to produce short strokes, are discussed in Section 6.2. This also includes the method of creating stipples from the wander behaviour, Section 6.2.2. The methods of shading polygons from feature outlines and primitives are described in Section 6.3. Some results combining the two methods are presented in Figure 6.12 and in Chapter 7.

6.2 Stroke Rendering

A smarticle's path can be used to render strokes with the various steering and flocking behaviours providing the different orientations. The paths can be rendered in various ways: as lines connecting each path step (creating long strokes), by drawing only the step points (for dotted lines or stipple effects), or using the step point to position a short stroke.

6.2.1 Smarticle Paths

Smarticle's paths, described in Chapter 5, can be rendered completely (i.e. connecting every path step), which is usually the choice for feature lines, user requested specific strokes, or where conveying a more rough "sketchy" appearance. For example in Figure 6.2 **(a)** the silhouette outlines of an Egyptian dyed are drawn as a continuous lines. In **(b)** the red lines are created by the user (no features are identified at these positions). (**c**) shows a sketchy version of the dyed with regular

horizontal smarticle paths creating strokes. Image **(d)** introduces some randomness in the termination point of the path steps.

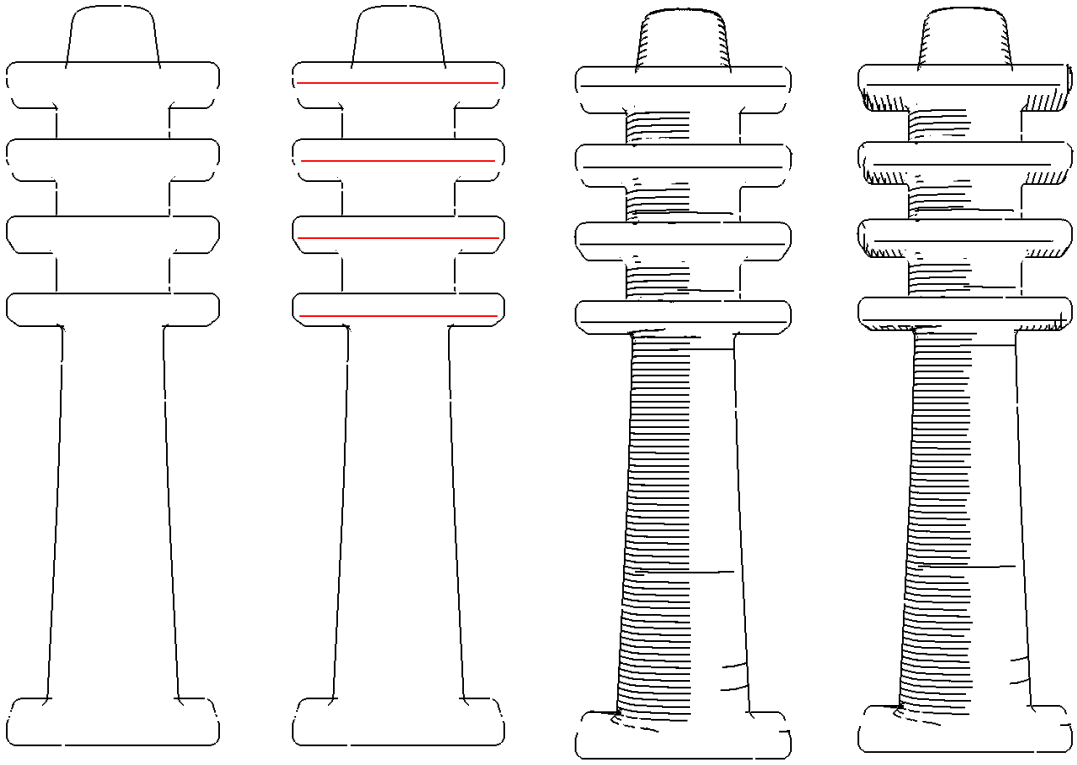

Figure 6.2: An Egyptian Dyed (**a**) The silhouette outlines are traced. (**b**) The user creates strokes (red lines) where no features are illustrated. (**c**) some regular strokes from horizontal smarticle paths drawn on the surface. (**d**) includes some more strokes and introduces a random termination value .

6.2.2 Stipples from the wander behaviour

Stippling effects can be produced by drawing only the step points from particle paths. Most of the paths are very regular (by nature) and appear as dotted lines, see Figure 6.3 (**a**), except for paths created using the wander behaviour. Using the

wander behaviour the actual path is indiscernible, instead, a random stippling effect is created, see Fig 6.3 (**b**). The strokes start from the silhouette and their paths are constrained so as the gradient at each path step is within sixty degrees of the gradient at the originating point.

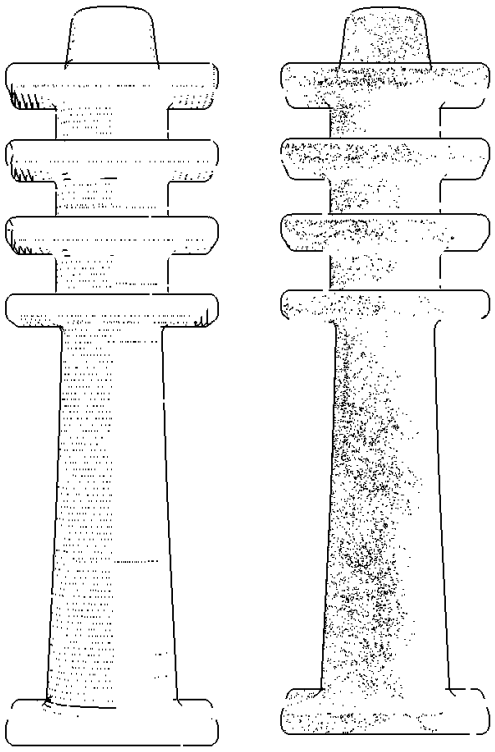

Figure 6.3: (**a**) The horizontal strokes from Figure 6.2 are rendered as dotted lines. (**b**) Smarticle paths use the wander behaviour, the path steps are used to place stipples.

6.2.3 Short strokes from paths

There are two ways that short strokes can be drawn at path steps. The first involves drawing subsets of the path, e.g. every n^{th} path step is used with m steps on either

side comprising the stroke, see Figure 6.4 (**a**).

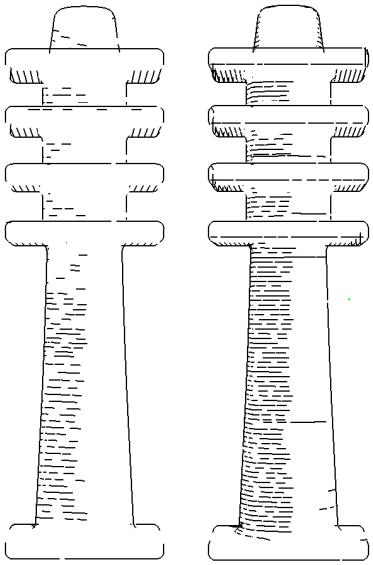

Figure 6.4: (**a**) Short strokes drawn using path steps directly. (**b**) The dyed rendered using curved strokes.

The second way is to use the method outlined in Chapter 2.4.1 where a centre point and direction vector are used to calculate the position and direction of the stroke, the degree of curvature of the surface is also used to curve it. The stroke is calculated as a Bézier curve with four control points defined using the path step and direction vector. Figure 6.4 (**c**) shows the dyed with curved strokes using the smarticle paths from Figure 6.2. Curved strokes are also used in the image in Figure 6.7 (**b**).

6.2.4 Short strokes from polygons

The style of results presented in most particle-based pen-and-ink renderers includes short strokes positioned anywhere on the surface, relative to particle positions. In

Figure 6.5 smarticles have placed short strokes all over the surface using the polygons from the initial polygonisation, the length of strokes is related to the curvature of the surface.

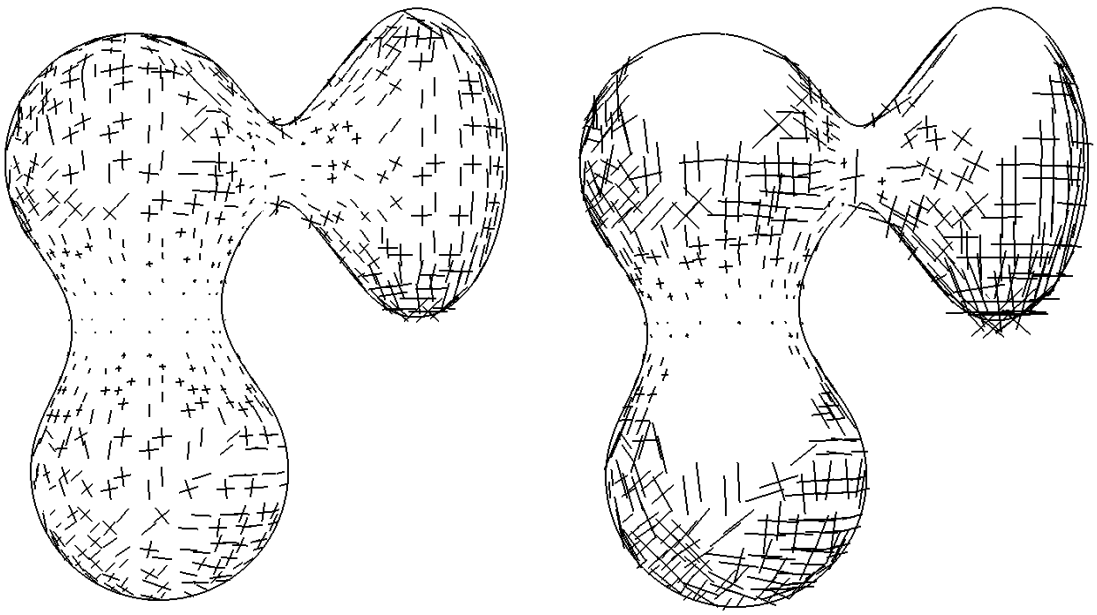

Figure 6.5: (**a**) Short strokes are placed at the centroids of the polygons and strokes in the first and second directions of curvature are drawn. (**b**) Longer strokes and lighting calculations.

The centroid, c, of the triangle is calculated from the vertices V^n using barycentric co-ordinates:

$$c = \alpha V^0 + \beta V^1 + \gamma V^2 \tag{6.1}$$

where:

$$\alpha = \beta = \gamma = \frac{1}{3} \tag{6.2}$$

and:

$$\alpha + \beta + \gamma = 1.0. \tag{6.3}$$

The position of the centre point for the stroke can be changed using different values for α, β and γ, as long as equation 6.3 holds. This ensures that the resulting point is within the convex hull of the original polygon.

6.2.5 Stroke Density

All of the above methods of visualising the stroke samples are enhanced using a density parameter. Where the path steps are used to create a dotted line, the number of path steps can be changed to achieve variations in the results, see Figure 6.6 (**a**). Where the path steps are used to create short strokes, drawing a stroke at each path step may produce either a solid line or too many strokes (visual clutter). In such a case a subset of these steps can be illustrated using every n^{th} step, Figure 6.6 (**b**). To avoid too much regularity, the step counter is chosen to be a random number (associated with each stroke so it is not calculated each time the display is refreshed).

Short strokes drawn near the silhouette outline can have the unwanted appearance of having greater density than strokes on forward facing parts of the surface. This is because the method is object-spaced rather than image spaced. The dot product of the gradient and the view vector (used to identify the silhouette region) can be used along with a stochastic probability to prune strokes in regions where the result is close to zero.

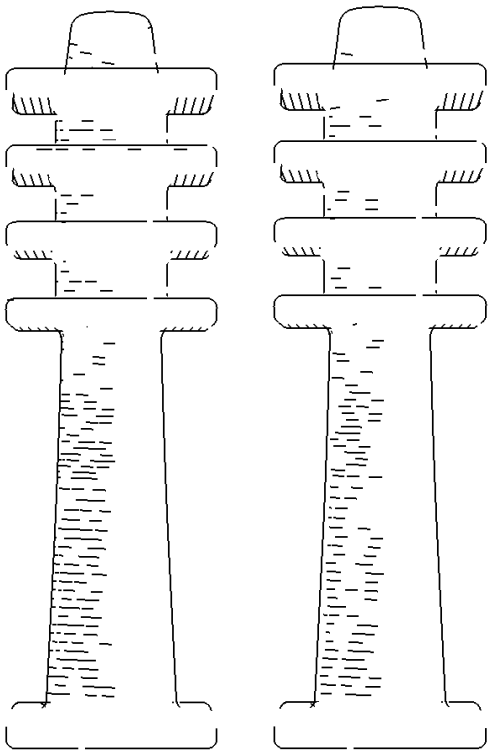

Figure 6.6: (**a**) The smarticle paths are drawn starting from the silhouette. (**b**) The paths start a short distance away from the silhouette to avoid visual clutter.

6.2.6 Lighting Calculations

Visibility of strokes due to lighting and shading is based on the method presented in Chapter 2.4.1. The concepts of upper and lower thresholding values, ut and lt, are used to define the blend region between light and shade. The thresholding values reference the dot product of the normal at the smarticle's position and the light direction.

In regions that are in shadow all strokes are drawn (subject to density calculations). In the blend region between the upper and lower threshold, strokes are chosen for rendering using a random element, see below. The dot product of the surface

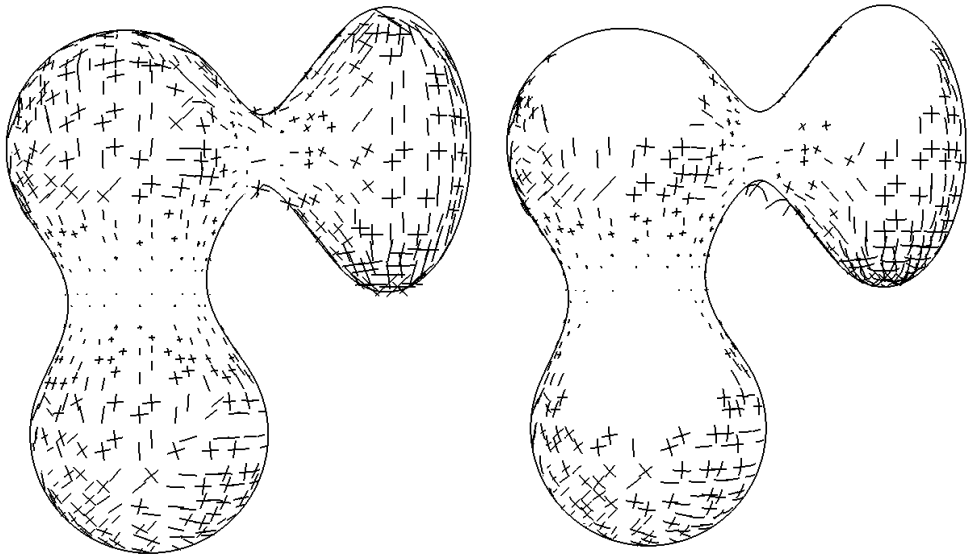

Figure 6.7: (**a**) $ut = 0.0$ and $lt = 1.0$, with straight strokes. (**b**) $ut = 0.25$ and $lt = 0.75$, with curved strokes.

normal at the smarticle's path step with the normalised direction of the light source is compared with a random value between ut and lt, if the value is less than the randomly generated value then the stroke remains visible, see Figure 6.7. Also only front facing strokes are drawn by ensuring that the dot product of the view vector with the surface normal at the path point is greater than zero. The algorithm for calculating the visibility of the stroke due to lighting is:

$\omega_{light} = point.normal \bullet lightDir$

$\omega_{rand} = rand(lt, ut)$

$point.ndotv = point.normal \bullet viewVector$

IF(($\omega_{light} < lt$ OR $\omega_{light} < \omega_{rand}$)

AND $point.ndotv > 0$)

drawStroke(*point.position*)

ENDIF

where $rand(lt, ut)$ uses Vigo's Mersenne Twister random number generator to generate a number between lt and ut.

6.2.7 Hidden Line Removal

Hidden Line Removal (HLR) is accomplished using the original polygonisation created at system initialisation. The polygons are slightly displaced in the opposite direction of the gradient, so that they are drawn just inside the iso-surface being visualised. This is a fast method, but it has drawbacks associated with the polygonisation being an approximation to the surface. There are places where the low resolution polygonisation does not approximate the surface very well, or at all, so there can be regions where there appears to be no HLR and outlines overlap, see the wheels of the train in Figure 6.12. Also, the polygonisation is not guaranteed to be inside the convex hull of the implicit surface, consequently feature outlines and strokes on the surface can be occluded, see the silhouette outline occlusion on the stalk of the pear in 6.11.

6.3 Polygonal Rendering

Illustrations and concept drawings often use colour as a means of accentuating, highlighting or focusing attention on a particular area. This suggestion of colour is, therefore, not necessarily intended to be realistic (e.g. medical illustrations) or complete (e.g. concept art), see Figure 6.1 for some examples. The manager and

smarticle colouring technique was inspired by such methods. The initial polygonisation (Chapter 3.6.4) is used to create colour accents or highlights, see Figure 6.8.

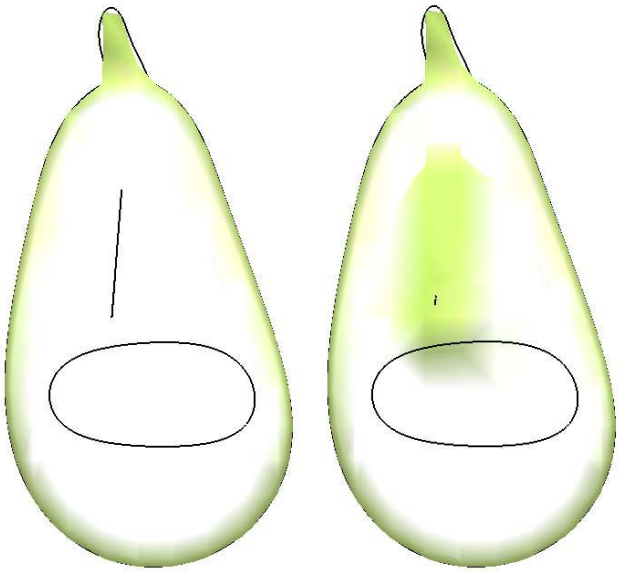

Figure 6.8: (**a**) The silhouette of the pear has been shaded and the user has drawn a single line. (**b**) The neighbourhood around the user drawn line is also coloured.

There are two ways for a user to select areas of the surface for colouring. The first is where the colour is calculated in the neighbourhood of a feature line or smarticle path, Section 6.3.1. The second method calculates colour in relation to selected primitives, Section 6.3.2.

The method of calculating the colour of the polygons was designed to be used in a multi-processing environment. A single smarticle processes a single voxel, with a variable number of smarticles active at any one time, each smarticle processes one voxel at a time. Additional voxels to be processed are assigned to smarticles

after they complete their current task, until all voxels have been examined. The colour is calculated according to an identified smarticle path, a copy of which is made available to each smarticle/voxel. This architecture is also suited to a single smarticle examining all of the voxels and calculating the colour values for all polygons in the neighbourhood.

Voxels, and their related polygons can be included in the neighbourhood of more than one colour calculation. In this case the average value of the colours is calculated.

6.3.1 Colour around a line

Any point, feature line or stroke sample can be used to identify the centre, or *skeleton*, of an area to be coloured. The skeleton is either a single point or a path of connected steps. The colour of the voxels in the neighbourhood is calculated as a measure of the distance to a point on this skeleton. In Figure 6.9, a user-selected colour is associated with the skeleton point (back dot) and smoothly blends to the background colour (white) as the distance increases.

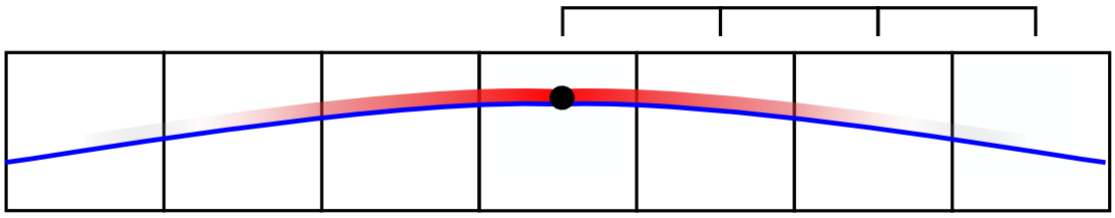

Figure 6.9: The red shaded line represents the colour polygons are shaded, which are blended with the background colour (white). The black dot indicates where the skeleton point is. The blue lines is used as a reference to the location of the surface.

The extent of the shading is defined by the voxel neighbourhood, which identifies

the distance at which the colour should be fully blended into the background. For example, if the neighbourhood is identified to include three adjacent voxels, the colour is smoothly blended to white before it reaches the farthest extent of the third voxel, see Figure 6.9. The voxel neighbourhood extends in three dimensions, which means that the maximum distance is calculated as the smallest edge dimension for the number of voxels (all voxels are evenly sized although the x, y and z dimensions may be different):

$dist^2_{max} = (numVoxels * dim_{min})^2$

Note that this distance does not describe the maximum distance from the centre of the voxel containing the skeleton point to the farthest edge of the last neighbourhood voxel. If the skeleton is in the centre of the original voxel this distance will extend to the centre of the last neighbour, see Figure 6.9. This guarantees that the polygons will be blended to the background colour at the edge of the last voxel.

The blend from highlight to background colour uses a Euclidean linear interpolation based on the distance $dist^2_{max}$:

$colour = (1 - \alpha) * skeleton.colour + \alpha * background.colour$

where the interpolation coefficient α is calculated as the distance from each of the triangle's vertices to the closest point on the skeleton:

$\alpha = \frac{DISTSQD(skeletonPt,vertex)}{dist^2_{max}}$

When the user selects "Colour neighbourhood" the manager creates a team of smarticles to process the voxels. The smarticles examine each voxel in the neighbourhood to identify which polygons will be coloured and what the colour values will be.

The algorithm to calculate the colour of triangle vertices therefore is:

FORALL triangles in voxel

FORALL vertices of triangles

$dist^2_{point} = FindClosestPtOnFeature(vertex, feature)$

$\alpha = \frac{dist^2_{point}}{dist^2_{max}}$

$vertex.colour = (1.0 - \alpha) * user.colour + \alpha * user.colour$

ENDFOR

ENDFOR

Note that the distance squared is used in calculations to avoid unnecessary square root evaluations.

As mentioned above, any point, line or stroke sample can be used as the skeleton for colour value calculations. Figure 6.8 shows examples of using an existing feature line, i.e. the silhouette, and a single stroke sample positioned by the user specifically to identify a region for colouring. The user created stroke is the black line in Figure 6.8 (**a**) and is shown with the shaded polygons in (**b**).

6.3.2 Colour a single primitive

The user can also select to colour an area of the surface relative to an individual or group of primitives that are used to construct a model. For example, in Figure 6.10 a single primitive is shaded where the colour is either limited to the individual primitive (**a**), or smoothly blended to match the background when the neighbouring primitive is encountered (**b**).

To accomplish specific primitive colouring an extension to the ISM system communication pipeline is necessary. This breaks the simple "black-box" approach to the

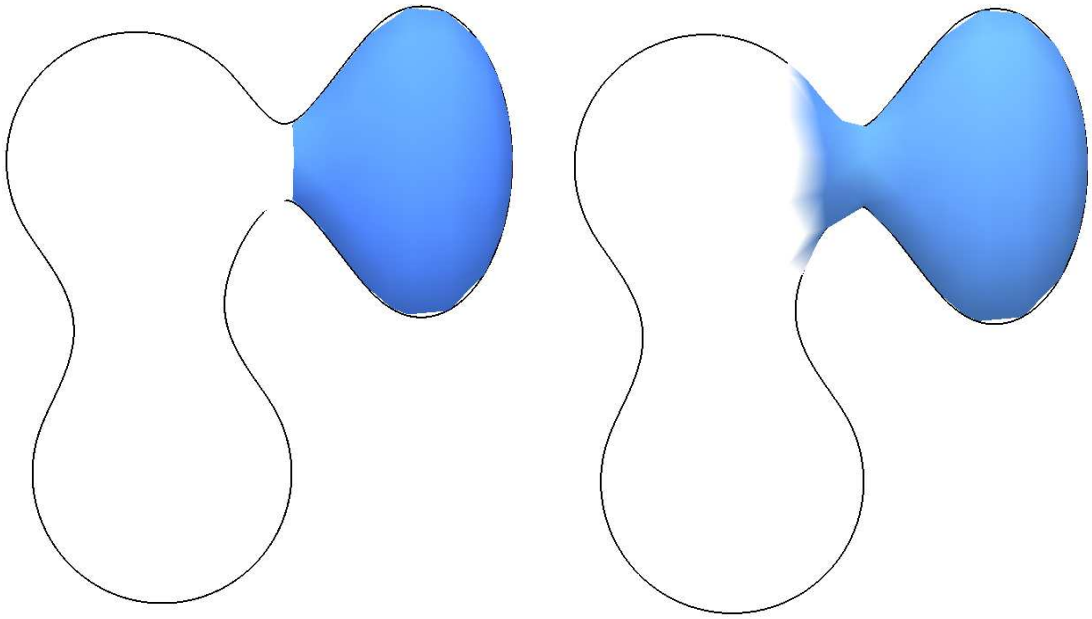

Figure 6.10: The user has identified a single primitive to be coloured. (**a**) Identifying the primitive with a point. (**b**) Identifying the primitive by it's attribute value.

ISM system using only field function and gradient information (as outlined in Chapter 2.2.4) but accomplishes it using simple attribute values that are also available with many other ISM systems and with much volume data.

In the context of the BlobTree, the person creating the model is required to create attribute values that uniquely identify each primitive, or group of primitives, at the modelling stage. For example, when the model in Figure 6.10 was created, each primitive was identified by the order in which it was added to the model, i.e. "one" (bottom), "two" (middle) and "three" (top, right), the coloured primitive. Note that numbers were used as attribute identifiers for simplicity only, these values are arbitrary, e.g. the parts of the train in Figure 6.12 were identified by "cabin" and "engine". The attribute values can be used to represent any group of primitives or

subtree that makes up the object.

The user positions a point in the object space (it does not have to be *on* the surface) and the manager queries the ISM system to identify the largest contributing primitive at that point. When the user selects "Colour single primitive" the manager queries the ISM system to identify the primitive and then assigns smarticles to examine the voxels in the neighbourhood (as above). The smarticles query the triangles in each of the voxels to determine those that identify the selected primitive as the largest contributer, these vertices have the user selected colour associated with them.

The algorithm to calculate the shading relative to a particular primitive is:

FORALL triangles in voxel

FORALL vertices of triangle

$\alpha_{colour} = GetShadeValue(vertex, attrib_{user})$

$vertex.colour = (1.0 - \alpha_{colour}) * user.colour + (\alpha_{colour}) * user.colour)$

ENDFOR

ENDFOR

Where $GetShadeValue(vertex, attrib_{user})$ is:

$contribution_{max} = CalculateMaxContributingValue(vertex, primitives)$

FORALL contributing primitives

IF $(attrib_{user} == prim_{current}.attrib)$

$prim_{user} = SetContributingPrimitive(prim_{curr})$

$contribution_{this} = GetContribution(vertex, prim_{user})$

$\alpha_{colour} = \frac{contribution_{this}}{contribution_{max}}$

ENDIF

ENDFOR

$return \alpha_{colour}$

and $CalculateMaxContributingValue(vertex, primitives)$ is:

FORALL primitives

$contribution_{this} = GetContribution(vertex, prim_{curr})$

IF $(contribution_{this} > contribution_{max})$

$contribution_{max} = contribution_{this}$

ENDIF

ENDFOR

$return contribution_{max}$

There are some situations where large differences in the sizes of primitives can have an unintended influence on the colour calculation method. In situations where an identified primitive is much smaller than a neighbouring one, the primitive identified by the user does not necessarily correspond to the largest contributing primitive. For example, Figure 6.11, the user has identified a point on the stalk of the pear. The neighbouring primitive is, however, much larger and, therefore, contributes more. In this situation a secondary method of identifying the primitive is necessary. In such a case the user must know the attribute value of the primitive they wish to illustrate. This can be selected in the UI and the colour is calculated relative to the identified primitive rather than the one with the largest contribution, in Fig 6.11 the stalk is primitive "four".

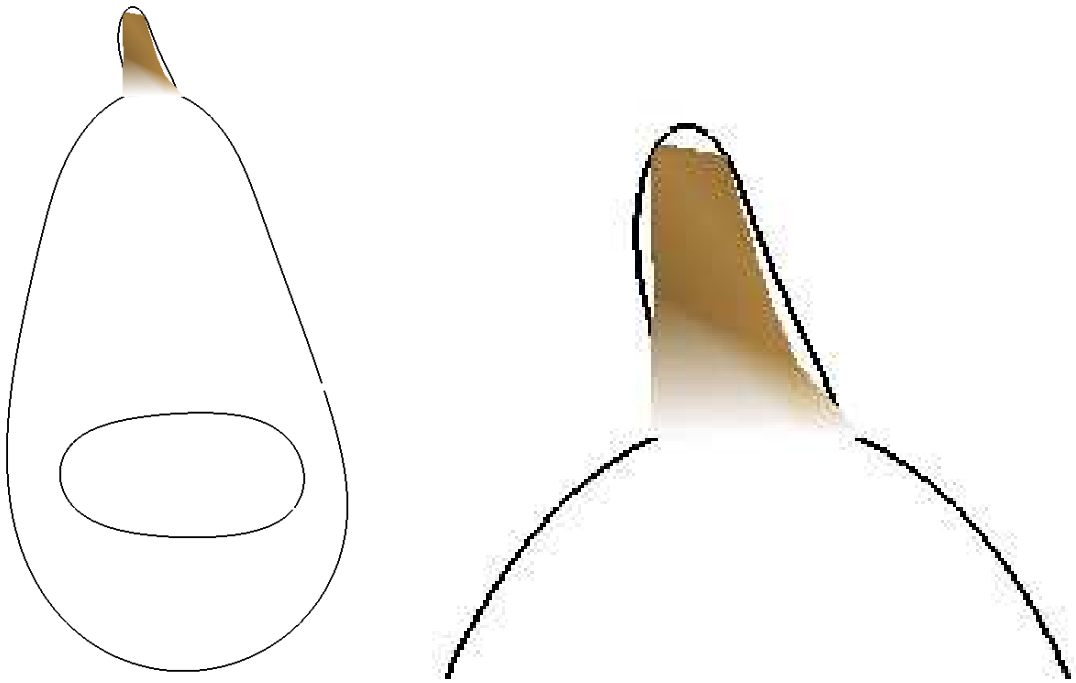

Figure 6.11: The stalk of the pear has been coloured. (**a**) The primitive attribute was required to colour the stalk. (**b**) A close up of the stalk.

This also affects the outcome of the colouring. In Figure 6.10 (**a**) where the user has selected the primitive with a point, the primitive can be explicitly recognised so the region and colour can be more accurately calculated, i.e. the colour stops where the identified primitive is no longer the largest contributing one. In Figure 6.10 (**b**) the primitive has been identified by it's attribute value, so the colour values are calculated from the contribution values.

The colour is blended in relation to the contributions of the primitives. As the selected primitive contribution becomes weaker and an adjacent ones becomes stronger the colour blends to be the same as the background, see Fig 6.10. The maximum contribution value in the neighbourhood is calculated to normalise the

individual values from the primitives. In this way some of the effects due to different sized primitives and different real-value contributions can be overcome allowing the colour selected by the user to be the colour used where the contribution is maximum.

6.4 Conclusions and Discussion

In this chapter, the methods the smarticle and manager system uses to render implicit objects are presented. Short or long strokes, or stipple type patterns can be created by visualising all or parts of the smarticles' paths. The smarticles can also be used to colour polygons created by the initial polygonisation in a simple concept art style, which is intended to give a colour accent or highlight.

The methods of drawing strokes presented here is based on using OpenGL lines of varying thicknesses. The appearance of strokes is therefore somewhat limited by the nature of GL-Lines. Using GL-Quads or a stroke texture would greatly enhance the visual quality of the strokes themselves. Likewise, using density thresholds for variability of stroke width and appearance related to the viewing angle in densely populated areas would enhance the visualisation, as in [ZISS04] and [WS94].

It would also be interesting to implement distances calculated according to surface measurements rather than using Euclidean distance. The differences, strengths and weaknesses of the various results could be evaluated in the context of concept drawings.

Using the polygons from the initial polygonisation for colouring has a couple of drawbacks. The first is related to the level of detail of the polygonisation and therefore the closeness of the approximation to the actual object. In Figure 6.11 the

triangles that approximate the stalk of the pear are coloured. The polygonisation is of a low resolution and therefore the approximation may not be close enough for the desired effect. In such situations more polygons should be created in a neighbourhood, which would result in a more accurate approximation of the surface.

The second drawback of this method of using the polygons from the initial polygonisation comes arises where voxels do not have polygons associated with them, even where they contain parts of the surface. The polygons that approximate the surface need to be found, this would require subdivision of the voxel identified to contain part of the surface (but no polygons). Each of these sub-voxels is used to generate polygons, in the same way as the original voxels were processed to generate the original triangles.

The presented method of identifying the primitive to be coloured has its flaws when considering large variances in primitive size, as mentioned in Section 6.3.2. A better way of identifying primitive rather than relying on the user to know the attribute ID value would be beneficial. There's no guarantee that the user has knowledge or access to the attribute values from the construction of the model.

The primitive colouring method can also be used to shade groups of primitives of subtrees of the tree that represents the object. IT is possible to extend this method to identify regions to be shaded using nodes such as the controlled blend node in the BlobTree [Gal05].

The contributions of this chapter are:

1. Shading areas of an implicit surface relative to an individual or group of primitives.

2. Concept style art for implicit surfaces: pen and ink style short and long strokes, stipples and colouring relative to surface feature outlines.

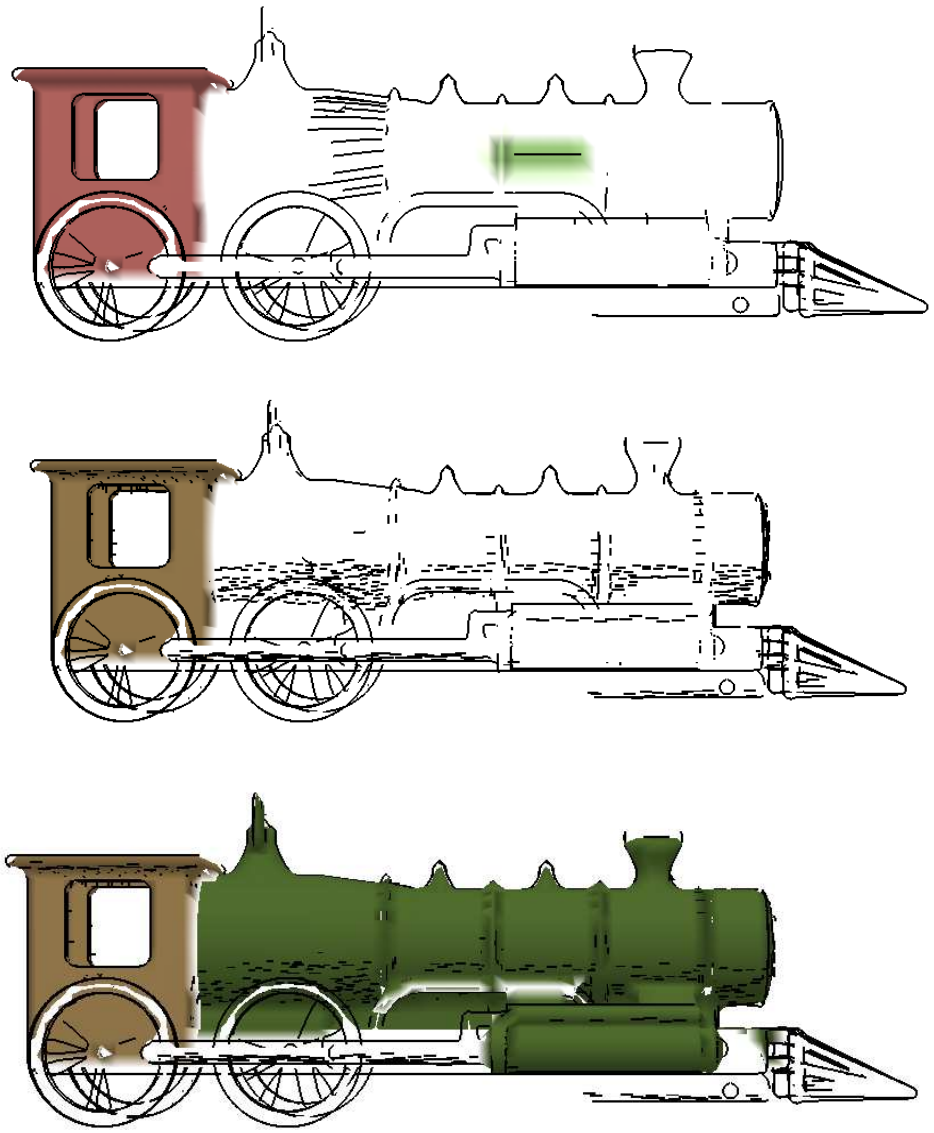

Figure 6.12: Colouring a group of primitives. (**Top**) The train's cab is shaded along with a user specified line to indicate the colour of the engine. there is also a group of user placed strokes on the engine. (**Middle**) Short strokes from polygons using lighting calculations. (**Bottom**) Both the train's cab and engine are shaded using different colours.

Chapter 7

Results and Conclusions

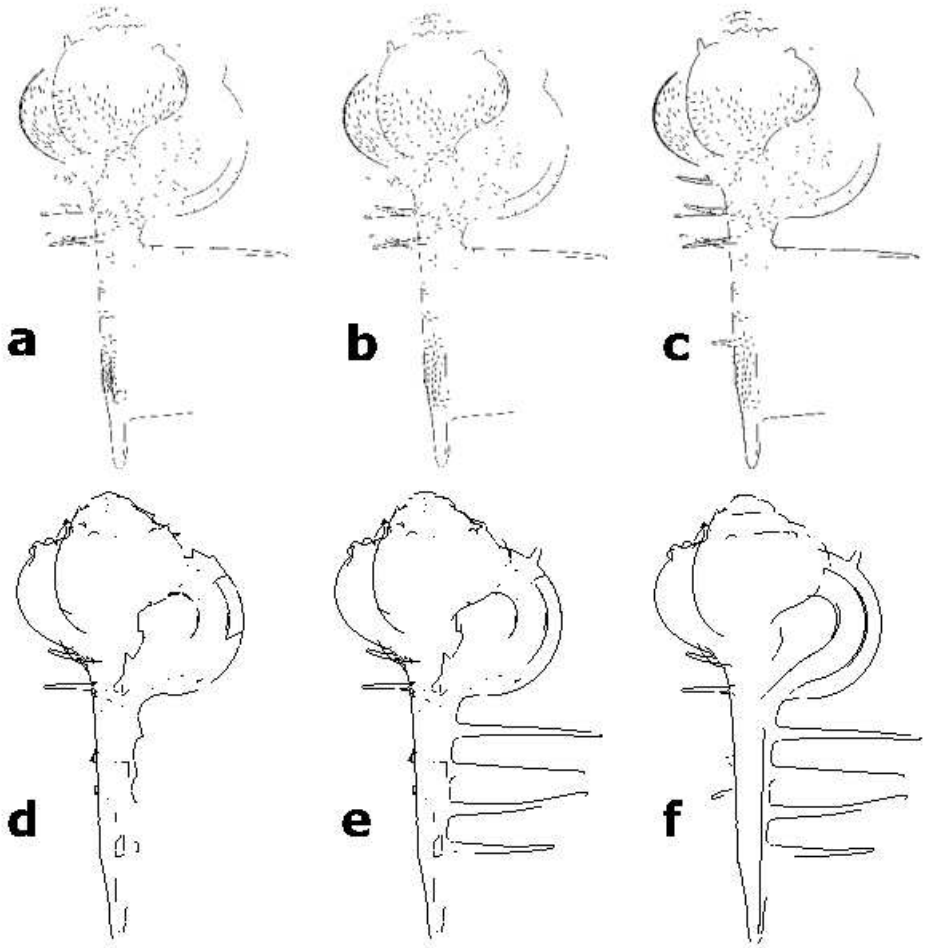

Figure 7.1: Automatic feature outline illustration. **(a)**: NPRBT after initialisation 12 seconds. **(b)**: NPRBT after 3 minutes. **(c)**: NPRBT after 5 minutes. **(d)**: Manager and Smarticles after initialisation 1 minute. **(e)**: Manager and Smarticles after 3 minutes. **(f)**: Manager and Smarticles after 5 minutes.

7.1 Results

The main drawbacks of using particle systems for rendering complex implicit surfaces are associated with the distribution method. The time taken to identify features and trace outlines is dependant upon the time taken to achieve a good coverage of particles.

There are two distinct areas of contribution from the research presented in this thesis. The first is increased efficiency of identification of features on a complex implicit surface. The second is the introduction of alternative rendering methods.

In this section these two main areas of contributions are discussed with reference to actual examples from the system presented in this thesis and the previous system, the NPRBT. The contributions are listed in the following section and summarised specifically in relation to the management system, the smarticles and mixed rendering.

7.1.1 Increased Efficiency

Previous particle based systems were effective in creating pen-and-ink style renderings for simple implicit surfaces. Complex models, however, required a significant amount of time to identify surface features and create a rendering.

In this research, a MAS comprised of the management system and smarticles is used to improve efficiency. Automatic identification and tracing of feature outlines is achieved faster than with previous particle based systems. Furthermore, a combination of automatic techniques and user direction provides a great deal of flexibility in creating an illustration.

The shell images in Figure 7.1 illustrate the increase in speed of automatic identification of feature outlines. In the top row are the images from the NPRBT system, the bottom row are the images from the system presented in this thesis. The images in **(a)** and **(d)** are generated after system initialisation. In the case of the NPRBT **(a)** that is 12 seconds, for the manager and smarticle system **(d)**, that is approximately one minute. Images **(b)** and **(e)** were both taken after approximately 3 minutes, and images **(c)** and **(f)** were taken at around six minutes. After 6 minutes the manager and smarticle method had completed all automatic tracing of initially identified feature outline locations. Note that these times are total time that include initialisation.

Both systems require user interaction to identify the remaining outlines within an acceptable time. Neither the NPRBT nor the manager and smarticle system can guarantee that *all* feature outlines will be found. Using the NPRBT, there are a number of ways the user can improve the time taken and identify more features. First, the number of particles on the surface can be increased, this is achieved either by automatic placement (using random rays from the camera position to the centre of the object) or allowing the user to identify a region where new particles should be created. Alternatively, the user can increase the particles' velocity, which conversely decreases the level of sampling detail, i.e. the particles take larger steps, and consequently are less likely to identify feature outlines. The manager and smarticle method allows the user to position a point around which a team of smarticles are created that autonomously explore the volume to identify locations of feature outlines.

Figure 7.2 illustrates the difference in time taken to trace the train's feature

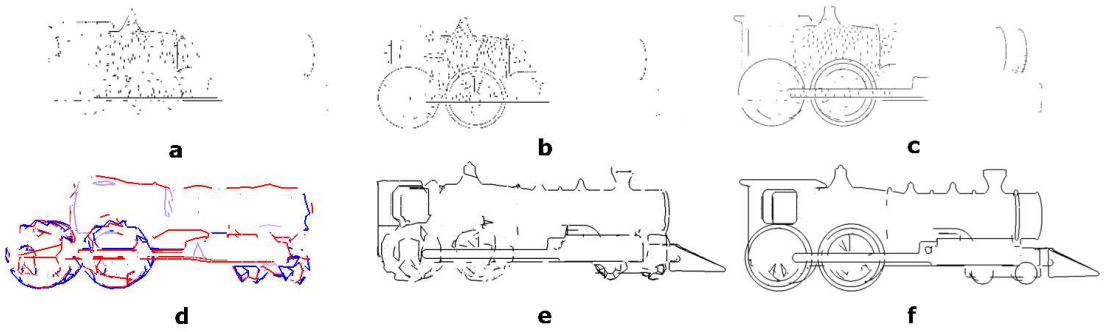

Figure 7.2: **(a)**: The NPRBT train after initialisation: 11 seconds. **(b)** NPRBT after 2 minutes. **(c)** NPRBT after 4 minutes. **(d)** The management and smarticle system after initialisation: 53 seconds. **(e)** Manager and Smarticle after 2 minutes. **(f)** Manager and Smarticle method after 4 minutes.

outlines. There was no user involvement in the NPRBT method (Figure 7.2 **(a)** to **(c)**), and no extra particles were added, 1000 particles is the default value. In Figure 7.2 **(d)** to **(f)**, the smarticle and manager method is used. The user selected "Trace All Feature Outlines" and used 10 smarticles, there was no other user interaction. Image **(c)** represents the result after five minutes using the NPRBT system. Image **(f)** took less than four minutes.

Figure 7.3 illustrates the heart model rendered with the NPRBT, and the manager and smarticle system in comparative times. Images **(a)** and **(c)** show the results after initialisation, 9 and 17 seconds respectively. Images **(b)** and **(d)** show the results after 5 minutes. In both cases user interaction was used to more fully identify the feature outlines. In **(b)** the user increased the number of particles from 1000 (the default value) to approx 2700, allowing automatic placement of new particles. Using the manager and smarticle approach in **(d)** the user directed smarticles to regions that required exploration to trace the outlines. In particular, the veins are difficult to trace as they are not constructed using simple smooth cylinders. Using

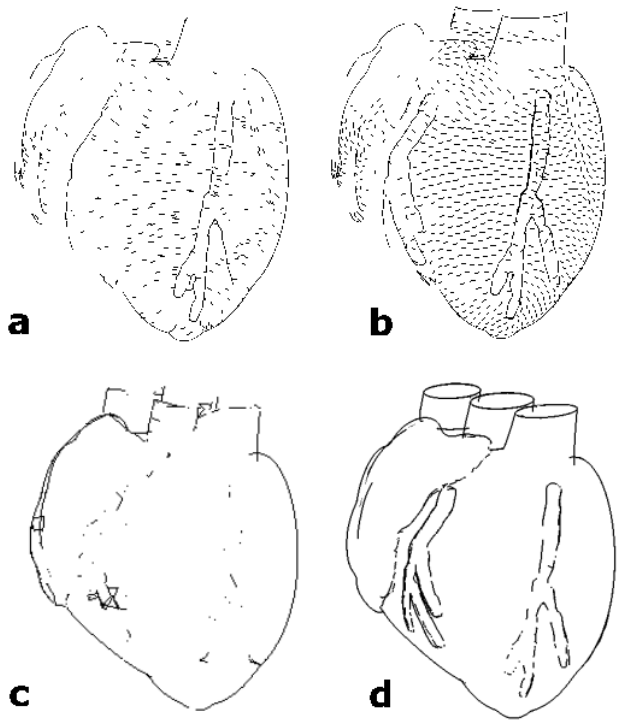

Figure 7.3: (**a**): NPRBT after initialisation 9 seconds. (**b**): NPRBT after 5 minutes. (**c**): Manager and Smarticles after initialisation 17 seconds. (**d**): Manager and Smarticles after 5 minutes.

both systems many particles or agents were required to trace parts of the outline in limited regions using short strokes. Neither system was able to trace the veins using a single particle or agent.

Table 7.1 lists the number of BlobTree nodes in each of the three main models used to illustrate efficiency results in this chapter: the shell, the train and the heart.

Examination of the numbers of system iterations, after the initialisation stage, identifies that although a single system iteration may take longer using the manager and smarticle approach, the movements are more directed and therefore result in

model	BlobTree nodes
shell	1497
train	723
heart	76

Table 7.1: Number of BlobTree nodes in model.

identification and tracing of more feature outlines. The time for one system iteration is generally longer because of the analysis and reporting elements.

The following tables list the number of system iterations that produce the above images. The main difference to note in calculating the system iterations concerns the method of tracing feature outlines. In the NPRBT, when an outline has to be traced, the particle immediately traces it to completion (either loop or lost) in the same system iteration. The management and smarticle system, however, traces them by taking one step with each system iteration.

The final images in each sequence were created in equal lengths of time, including the initialisation stage. In most cases a single system iteration takes longer with the manager and smarticle system, but because these are goal directed movements rather than based on a simple repelling function, more features are identified.

Initialisation of the NPRBT uses 1000 particles randomly placed using a ray tracing method. Initialisation of the management and smarticle system involves a team of three smarticles processing the polygonal mesh for edges that cross silhouettes, discontinuities and changes in sign of curvature. Initialisation of the man anger and smarticle system takes longer and involves more steps, but identifies more features. Another important distinction to make is that with the manager and smarticle system, the count of system iterations starts from zero after initialisation. This is

Shell	Initialisation	3 minutes	5 minutes
NPRBT	5	233	495
Mgr & Sm	2366	189	317

Table 7.2: Number of system iterations for the shell model. The Manager and Smarticles method uses more evaluations for initialisation (which takes longer than the NPRBT), but identifies more features. In general system runs, the number of iterations is slightly smaller for the Manager and Smarticle system (i.e. one iteration takes longer) but more features are identified. After 5 minutes (including initialisation) the manager and smarticle method identifies and traces more features than the NPRBT, see Figure 7.1

because the smarticles' tasks are very different between the initialisation and exploration stages. This also gives a better indication of system iterations involved in exploring the space and creating the images. The number of iterations after the initialisation stage is cumulative. The final times, are also cumulative, including the initialisation stage for both systems.

In Table 7.2 the results are shown for rendering the shell images in Figure 7.1. Although only 5 NPRBT system iteration steps have been taken, some particles have also traced features, i.e. the silhouette outlines. Therefore, most particles have only taken 5 steps, but a few of them have taken more. The manager and smarticle system uses the team of 3 smarticles, which amounts to 2366 examinations of polygon edges to identify feature outline crossings. The resolution of the polygonisation grid was 25^3.

Although there are more particles and (slightly) more system iterations for the NPRBT system, most of these evaluations are for general surface strokes and do not necessarily identify features. For the cost of performing the initial polygonisation to identify potential locations, and the cost of the manager examining the data from the

Train	Initialisation	2 minutes	4 minutes
NPRBT	3	249	403
Mgr & Sm	1640	233	481

Table 7.3: Number of system iterations for the train model. After 4 minutes the Manager and smarticle method has identified more features.

heart	Initialisation	5 minutes
NPRBT	15	298
Mgr & Sm	860	184

Table 7.4: Number of system iterations for the heart model.

blackboard, more features are identified using the manager and smarticle method.

In Table 7.3 the results of both systems rendering the train in Figure 7.2 are shown. The resolution for the manager and smarticle system was 15^3. The results show that a much larger number of smarticle operations are necessary for examining the polygonal model. Overall, the number of system iterations for exploration are similar, although the smarticles are more directed towards areas where potential feature outlines are identified.

Although the shell and the train have a large variance in the number of BlobTree nodes, the number of polygon edge examinations are closer than would be expected. The train has a lower number of BlobTree nodes than the shell, but most of the train's primitives are identified from the initial polygonisation. This is not the case for the shell, many of the nodes construct features that have not been identified by the low resolution polygonisation, i.e. the spikes.

The manager and smarticle resolution of the heart model in Figure 7.3 was 20^3, the results of the comparison are in Table 7.4. The same pattern as the shell and

train models is continued: a larger number of operations in the preprocessing stage, and fewer iterations for exploration and tracing, which results in greater feature identification.

Examination of the numbers of system iterations, after the initialisation stage, identifies that although a single system iteration may take longer using the manager and smarticle approach, the movements are more directed and therefore result in identification and tracing of more feature outlines. The time for one system iteration is generally longer because of the analysis and reporting elements.

7.1.2 Alternative Rendering

The second major contribution of the work presented in this thesis, is in the area of mixed rendering. Alternative rendering techniques are provided in the form of new methods of placing strokes, stipples and creating colour highlights.

The heart model shown in Figure 7.4 has been created by tracing the feature outlines **(a)**; shading the right auricle and atrium (subtree or group of primitives from the BlobTree) using stipples from the wander behaviour **(b)**; shading the polygons **(c)**; and using the same polygonal shading and adding short strokes and using lighting calculations **(d)**.

One of the drawbacks to using the original polygonisation for shading with colour is the visibility of the polygonal edges, as can be seen in the images in Figure 7.4 on the bottom row. A higher resolution polygonisation (through adaptive subdivision) in these visible transition areas would be desirable.

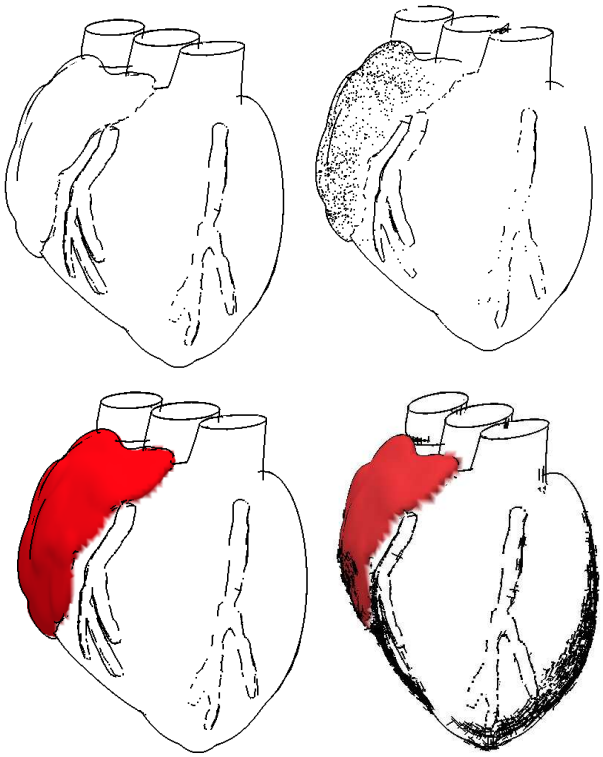

Figure 7.4: (**a**): A heart model with the main feature outlines traced. (**b**): The heart model with the right atrium and auricle shaded using a stipple pattern. (**c**): The heart model with the atrium and auricle polygons shaded red. (**d**): The heart model with the auricle and atrium shaded red and short strokes.

7.2 Conclusions

In this section, the list of contributions from the introduction is reproduced with a discussion of the results, strengths and weaknesses of the approach.

7.2.1 Contributions

1. The Management System:

a **Organises agents to collectively explore an implicit surface environment and visualise object data using NPR styles.**

Most previous systems use a particle based approach to NPR for implicit surfaces, and most previous agent-based systems are restricted to image space. The only other object-space agent-based method presented by Pang et. al. [PS93] (used to visualise scientific data) identifies that problem areas are related to agent control and co-ordination. The manager presented in this thesis addresses these concerns by organising inter-agent communication and co-operation, and controls the overall system.

In the research presented in this thesis, the majority of feature outlines are identified by processing an initial low resolution polygonisation. This can not, however, *guarantee* that *all* outlines are identified. Features can be missed due to sampling disparities between the size of voxels used in the polygonisation and the size of details on the surface. Feature identification and tracing is not, however, tied to the LOD provided by the polygonal representation as in [SIJ+07]. Features not identified by processing the polygonisation can be identified and traced through smarticle exploration, although identification of *all* features can not be guaranteed.

b **Analyses the object space to identify potential locations of features.**

The workload estimation method presented in Chapter 3.5.1 can identify voxels that are likely to contain feature outlines. The measure of workdone (Chapter 3.5.2) identifies the likelihood that any features in the voxel have already been traced. This identifies potential locations

directly rather than relying on the distribution method. A suitable data structure is required to facilitate data collection and analysis. The voxel based shared memory known as the blackboard is used for this.

c **Interprets user requests to create appropriate tasks.**

An assumption of the research in this thesis is that the user is involved in image creation to a greater extent than simply deciding the orientation of short internal strokes and lighting effects. Chapter 3.3 describes the method of interpreting user requests to create tasks for agents. Agents are designed to complete tasks that are directly related to user requirements. The specification of user requests and related tasks, shift reliance from solely automatic measures to include the user more in creating the image. These tasks are assigned by the manager and smarticles do not have to be managed individually or directly.

d **Prioritises tasks to be processed based on workload and work-done estimates.**

The user identifies regions or neighbourhoods for performing the tasks. Voxels with potential feature outlines or where not enough information is known are processed before voxels with traced outlines or a large number of sample data from smarticle paths, see Chapter 3.5.4

e **Uses polygonisation for initial positions and voxel data thereby identifying feature outline locations faster than the previous method.**

Figures 7.1 **(a)** and **(d)** (the NPRBT and the manager method after sys-

tem initialisation) illustrate that the manager and smarticles method of using a low resolution polygonisation for system initialisation can identify feature outlines faster than relying on the Witkin-Heckbert (WH) method [WH94] (for complex implicit surfaces). The NPRBT has traced a few outlines whereas the management system has identified the presence of more of the features (not yet traced). These results can be seen clearly in Figure 7.2 where Image **(d)** illustrates the features that are identified after system initialisation. Compare this with the results after system initialisation with the NPRBT in Image **(a)**.

Upon system initialisation, the surface can immediately be covered with sort strokes. They are placed on the surface using the polygonal information; barycentric combinations of vertices define the centre point of the stroke rather than particle positions, as in [FJW+05].

2. Smarticles:

   a **Object space, agent-based method.**

   Smarticles operate and create renderings in object space, rather than image space as in [SOD04, MDC04, SGS05]. Most object space methods use a particle system approach. Smarticles approach the task in a more goal directed way than particle systems. They have a history which allows further analysis of the space, unlike particle based systems which generally store only current positions of particles (with their associated properties) and locations of previously identified feature outlines. This means that information gained from smarticles' movements and queries

about the behaviour of the surface are retained and used as a valuable asset in examining the surface and finding features.

b **The Dual Tracking Algorithm**, which identifies and traces feature outlines due to discontinuities in the field or abrupt blends between primitives.

The method presented in Chapter 5.7.1 uses a black-box approach to the ISM system to identify and trace a feature due to a discontinuity or abrupt blend, see Figures 5.7, 3.12, 7.1 and 7.4.

c **Apply steering and flocking behaviours to agents to both sample and render the implicit surface.**

In Chapter 5 examples are given of using flocking and steering behaviours to both sample the object space and direct smarticle paths. Previous methods have used silhouette edge ([BH98, FJW+05]), principal directions of curvature ([Elb98, BH98, FJW+05]), contour ([FJW+05]) and planar slices for stroke directions ([Ric73]). The long strokes presented in this research are similar to streamlines created in [ZISS04, HZ00], but do not require pre-processing. The paths can be created individually, or using flocking behaviours (which require teams) to create variations in the appearance of strokes.

d **Create stipples from wander behaviour.**

Stipples are normally created relative to particle positions [FJW+05], or in image space using voronoi diagrams [DHvOS00] or graph based relaxation algorithms [PS04]. Using the wander behaviour is a novel approach to

creating stippling patterns (see Figures 6.3 and 7.4 for examples).

e **Allow the user to specify arbitrary stroke directions** (not constrained by surface properties, such as principal directions of curvature or contour).

The user can create strokes in any orientation they choose, using the seek behaviour 5.4.5, see Figures 6.2, 6.8 and 6.12 **(a)**. The user can also use two such strokes as guides and have the system create strokes in between these guidelines, see Figure 5.7 and 6.12 **(a)**.

3. Mixed Rendering:

a **Supports creation of shading for areas of an implicit surface relative to an individual or group of primitives.**

The method presented in 6.3.2 is a novel method of selecting individual or groups of primitives for rendering in a particular style. Models with a tree-like data structure can have any subtrees rendered (for examples see Figures 6.12 and 7.4). The object, or parts of it can be rendered by colouring some of the polygons. Colours are calculated using the original polygonisation, which is limited to the original LOD. Subdividing relevant voxels and polygons would give a better approximation to the actual surface where it is needed. Colour calculations are made relative to a feature line or a single primitive (or a group). Identifying a primitive breaks the "black-box" approach to the ISM system by using an attribute value for each primitive (or group). This is the most general and straightforward way of achieving such an effect whilst maintaining a "black-box" approach

to the ISM system.

b **Make available concept art styles of rendering for complex implicit surfaces: pen and ink style short and long strokes, stipples and colouring relative to surface feature outlines.**

Previous systems have rendered implicit surfaces in pen-and-ink styles [Elb98, Akl98a, Akl98b, FJW+05]. The current system makes use of many rendering techniques, i.e. strokes related to feature outlines, strokes placed by the user for effect, short general surface strokes and includes shaded polygons, to create an image (see Figures 6.12 and 7.4).

7.2.2 Discussion

If the WH style distribution method were to accept polygon vertices as initial positions for particles it would require alterations. The method is based on repulsion radii around each particle and aims to reach equilibrium in particle placement. Using polygon vertices the placement is already close to equilibrium, which is why some techniques use the final particle placement to create a polygonal model.

Identifying and tracing *all* feature outlines of a complex model, used in this research, requires user interaction. It is assumed that the user will request more detail in regions they are interested in, rather than expect the manager to automatically identify and explore *all* areas. For example, all of the hearts in Figure 7.4 took less than 15 minutes to create using an iterative user-directed approach. The session was not scripted and there was no particular desired outcome, some of this time was spent making decisions about appearance and which features to trace and in how much detail. The resolution of the heart model was increased to 100 voxels

cubed to achieve a better visual representation of the polygons that are shaded. The atrium/auricle region was also identified by its attribute number rather than positioning a point in space, as the neighbouring primitives are much larger and contribute more. This means that all of the voxels must be examined to determine if the selected primitives have a contribution. The method would benefit from a polygonal subdivision method which would allow the initial surface to be processed at a lower resolution, thereby making it faster. Also this would mean that less voxels would have to be examined to determine primitive contributions. Another useful optimisation strategy would be to limit the number of voxels examined by using a method analogous to the continuation method. In which case, the voxels examined could be extended to include those that do not contain parts of the surface. It is possible that there are groups of primitives where the visible surface areas are not contiguous, but are connected internally, or the connection is not visible.

If the polygonisation doesn't identify all feature outline locations, those outlines will only be found by the user identifying where they ought to be. Which means that although the user does not need to understand the underlying data they need to be aware of the object being represented. The user directs a team of smarticles to a neighbourhood to explore the region. The smarticles then identify whether they can find any features at the specified location.

7.3 Future Work

Smarticles were designed to operate in a multi-processing environment, but unfortunately, the MAS engine Vigo [Bur] was not capable of this. An important direction

of future work would be to develop the smarticle and management system to operate in a multi-processing environment.

The smarticle and management system was developed with an aim of moving away from purely automated tools to encouraging exploration of the object space. The user interface is basic and is not currently conducive to this type of exploration. A more appropriate user interface would be a great asset and could also be used to help explore the parameter space.

The smarticle locomotion method has been deliberately designed with the aim of easily adding new behaviours. Behaviours being developed include tracing suggestive contours, regions of curvature ign change and exploring ridges and ravines.

Although the method is object-space based, it has been developed with the aim of producing a single image. A valuable addition would be to maintain coherency when changing the view of the object. There are also many avenues of interest in dealing with animation or changing topologies. The agents could be used to sample regions of the surface that change over time, concentrating in areas that are identified as not being well sampled. Or, alternatively agents could be assigned to specific voxels and spawn exploratory teams when the surface changes and requires further investigation.

Shading polygons relative to particular lines has given some promising results. The method, however, is limited due to the reliance on the original resolution of the polygonisation. A better method of subdividing the voxels and polygons, and creating polygons where none have been identified would improve the results. This would also improve the HLR, as the polygons would more closely approximate the surface and would be less likely to occlude feature outlines. Another way to achieve better accuracy relative to individual outlines or primitives would be to change the

shading method so that it does not use linear interpolation to calculate the colour. The interpolation across a triangle, relative to its vertices can sometimes create a different gradient in adjacent triangles. The primitive identification method could easily be extended to include restriction of areas to be rendered using techniques similar to a bounded blend [Gal05].

One of the most exciting avenues to pursue as future work is to apply the method to other modelling systems and object data. Especially real-time or interactive modelling systems such as ShapeShop presented in [SWSJ05]. The basic NPRBT system was adapted to an earlier version of ShapeShop and some results were presented in [WFJ+05]. There is also considerable scope for applying the manager and smarticle method to point based graphics.

Point Based Graphics (PBG) are used to overcome problems associated with rendering models with a high level of polygonal complexity.

Algorithms take unstructured point clouds as input and create a visually continuous surface for rendering. Implicit surface fitting and particle simulations are two techniques commonly used to reconstruct surfaces from irregular point samples.

Point Based Graphics (PBG) are used to overcome problems associated with rendering models with a high level of polygonal complexity. Algorithms take unstructured point clouds as input and create a continuous surface for rendering. Implicit surface fitting, particularly Radial Basis Functions (RBF), are commonly used to reconstruct surfaces from irregular point samples. The manager and smarticle method could be used to analyse samples and estimate the behaviour of the surface. A common problem with PBG is related to reconstruction of sharp edges. The Dual Tracking Algorithm could be adapted to identify and trace feature outlines due to

discontinuities or creases in the model.

PBG also often use techniques such as particle simulations to create suitable surface reconstructions, which can then be used to create a mesh that is a good approximation of the surface. Such techniques rely on distribution methods and energy minimisation techniques that are fundamental in particle systems. The management and smarticle system could be adapted to perform energy minimisation on component surface samples, or indeed any capabilities offered with particle systems. The power of agents in a MAS comes with their additional capabilities in terms of a more cognizant approach to problem solving than most particle systems.

Bibliography

[AGCA06] Remi Allegre, Eric Galin, Raphaelle Chaine, and Samir Akkouche. The hybridtree: Mixing skeletal implicit surfaces, triangle meshes, and point sets in a free-form modeling system. *Graphical Models*, 68(1):42–64, January 2006.

[Akl98a] E. Akleman. Implicit painting of csg solids. In *Proc. of CSG '98, Set-Theoretic Solid Modelling: Techniques and Applications*, pages 99–113, 1998.

[Akl98b] E. Akleman. Implicit surface painting. In *Proc. of Implicit Surfaces '98*, pages 63–68, 1998.

[BBB+97] J. Bloomenthal, C. Bajaj, J. Blinn, M.P. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997. ISBN: 155860233X.

[BH98] D. Bremer and J.F. Hughes. Rapid approximate silhouette rendering of implicit surfaces. In *Proc. of Implicit Surfaces '98*, pages 155–164, 1998.

[Bli82] J Blinn. A generalisation of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[Blo88] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[BPK98] A. G. Belyaev, A. A. Pasko, and T. L. Kunii. Ridges and ravines on implicit surfaces. In *CGI '98: Proceedings of the Computer Graphics International 1998*, pages 530–535, Washington, DC, USA, 1998. IEEE Computer Society.

[Bur] Ian Burleigh. Vigo::3d, http://vigo.sourceforge.net/.

[CA97] Patricia Crossno and Edward Angel. Isosurface extraction using particle systems. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 495–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[Cra04] Walter Crane. *Line and Form.* George Bell and sons, 1904.

[Cra93] I.D Craig. A new interpretation of the blackboard architecture, technical report 254 dept computer science, university of warwick, 1993.

[Den99] Jörg Denzinger. Foundations of multi-agent systems, Univeristy of Calgary Course Notes, 1999.

[DF99] Jörg Denzinger and Dirk Fuchs. Cooperation of heterogeneous provers. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 10–15, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[DHvOS00] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3):40–51, 2000.

[DLC89] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989.

[DS00] O. Deussen and T. Strothotte. Computer-generated pen-and-ink illustration of trees. In *Proc. of SIGGRAPH '00*, pages 13–18, 2000.

[Ebe99] David Eberly. Principal curvature of surfaces, Geometric Tools, http://www.geometrictools.com/documentation/documentation.html., 1999.

[Elb98] G. Elber. Line art illustrations of parametric and implicit forms. In *IEEE Transactions on Visualization and Computer Graphics, Vol. 4, No. 1*, pages 71–81, 1998.

[FCG00] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Practical volumetric sculpting. *The Visual Computer*, 16(7):469–480, 2000.

[FJW+05] Kevin Foster, Pauline Jepp, Brian Wyvill, Mario Costa Sousa, Callum Galbraith, and Joaquim A. Jorge. Pen-and-Ink for BlobTree Implicit Models. *EUROGRAPHICS2005*, 24(3):267–276, September 2005.

[FS96] Kikuo Fujimura and Karan Singh. Planning cooperative motion for distributed mobile agents. *Journal of Robotics and Mechatronics*, 8(1):75–80, 1996.

[Gal05] Callum Galbraith. *PhD Thesis.* PhD thesis, University of Calgary, Calgary, Canada, 2005.

[GH91] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, volume 25, pages 267–274, July 1991.

[GIHL00] A. Girshick, V. Interrante, S. Haker, and T. Lemoine. Line direction matters: an argument for the use of principal directions in 3d line drawings, 2000.

[GNW02] Callum Galbraith, Mai Ali Nur, and Brian Wyvill. Data Structures for Animating Implicit Surfaces. *Proc. Graphics Interface*, May 2002.

[Gup76] Arthur L. Guptill. *Rendering in pen and ink*. Watson-Guptill Publications, 1515 Broadway, New York, NY 10036, 1976. ISBN: 0-8230-4530-7.

[GWW86] C. McPheeters G. Wyvill and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[HR85] Barbera Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[HZ00] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proc. of SIGGRAPH 2000*, pages 517–526, 2000.

[JBS03] Huw Jones, Ron J. Balsys, and Kevin G. Suffernt. Point based rendering of non manifold surfaces. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 267–ff. ACM Press, 2003.

[JWS06] Pauline Jepp, Brian Wyvill, and Mario Costa Sousa. Smarticles for sampling and rendering implicit models. In *Theory and Practice of Computer Graphics*, pages 39–46, 2006.

[KB89] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 297–306, New York, NY, USA, 1989. ACM Press.

[MDC04] Kaye Mason, Jörg Denzinger, and Sheelagh Carpendale. Negotiating gestalt: Artistic expression and coalition formation in multiagent systems. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1350–1351, Washington, DC, USA, 2004. IEEE Computer Society.

[MG07] Subhas Mukhopadhyay and Gourab Sen Gupta, editors. *Autonomous Robots and Agents*. Springer, 2007. ISBN: 978-3-540-73423-9.

[NHK+85] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, and K. Omura. Object modelling by distribution function and a method of image generation. *Transactions of the institute of Electronics and Communcation Engineers of Japan*, J68-D(4):718–725, 1985.

[PA02] Alexander A. Pasko and Valery Adzhiev. Function-based shape modeling: Mathematical framework and specialized language. In *Automated Deduction in Geometry*, pages 132–160, 2002.

[PFS03] O.E. Meruvia Pastor, B. Freundenberg, and T. Strothotte. Animated, real-time stippling. In *IEEE Computer Graphics and Applications (Special Issue on Non-Photorealistic Rendering)*, pages 62–68, 2003.

[PG96] Ken Perlin and Athomas Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216, New York, NY, USA, 1996. ACM Press.

[PHWF01] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proc. of SIGGRAPH '01*, pages 579–584, August 2001.

[PS93] A. Pang and K. Smith. Spray rendering: Visualization using smart particles. pages 283–290, 1993.

[PS94] A. A. Pasko and V. V. Savchenko. Blending operations for the functionally based constructive geometry. In *CSG 94 Set-theoretic Solid Modeling: Techniques and Applications*, pages 151–161, Winchester, UK, 1994. INFORMATION GEOMETERS.

[PS04] Oscar Meruvia Pastor and Thomas Strotthote. Graph-based point relaxation for 3d stippling. In *ENC '04: Proceedings of the Fifth Mexican International Conference in Computer Science (ENC'04)*, pages 141–150, Washington, DC, USA, 2004. IEEE Computer Society.

[PS06] Hans Pedersen and Karan Singh. Organic labyrinths and mazes. In *NPAR '06: Proceedings of the 4th international symposium on Non-*

photorealistic animation and rendering, pages 79–86, New York, NY, USA, 2006. ACM Press.

[PZvBG00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[Ree83] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, 1983.

[Rey87] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Rey99] Craig Reynolds. Steering behaviors for autonomous characters, 1999.

[Ric73] A. Ricci. A constuctive geometry for ccomputer graphics. *The Computer Journal*, 16(2):157–160, 1973.

[RKS00] C. R¨ssl, L. Kobbelt, and H. Seidel. Line art rendering of triangulated surfaces using discrete lines of curvature. In *Proc. of WSCG 2000*, pages 168–175, 2000.

[RRS97] Angela Rösch, Matthias Ruhl, and Dietmar Saupe. Interactive visualization of implicit surfaces with singularities. *Computer Graphics Forum*, 16(5):295–306, Dec 1997.

[Sab68] Malcolm Sabin. The use of potential surfaces for numerical geometry. *Technical Report no. BAC/VTO/MS/157*, 1968.

[SF95] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 129–136, New York, NY, USA, 1995. ACM Press.

[SFWS03] M. C. Sousa, K. Foster, B. Wyvill, and F. Samavati. Precise ink drawing of 3d models. *Computer Graphics Forum (Proc. of EuroGraphics '03)*, 22(3), 2003.

[SGS05] Stefan Schlechtweg, Tobias Germer, and Thomas Strothotte. RenderBots—Multi Agent Systems for Direct Image. *j-CGF*, 24(2):137–148, June 2005.

[SH05] Wen Y. Su and John C. Hart. A programmable particle system framework for shape modelling. In *Shape Modeling and Applications*, pages 114–123, 2005.

[SIJ+07] Ryan Schmidt, Tobias Isenberg, Pauline Jepp, Karan Singh, and Brian Wyvill. Sketching, scaffolding and inking: A visual history for interactive modelling. In *5th International Symposium on Non-Photorealistic Animation and Rendering*, 2007.

[SOD04] Yann Semet, Una-May O'Reilly, and Fredo Durand. An interactive artificial ant approach to non-photorealistic rendering, 2004.

[Sta99] Jos Stam. Stable fluids. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 121–128, August 1999.

[SWG05] R. Schmidt, B. Wyvill, and E. Galin. Interactive implicit modeling with hierarchical spatial caching, 2005.

[SWHS97] M.P. Salisbury, M.T. Wong, J.F. Hughes, and D.H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proc. of SIGGRAPH '97*, pages 401–406, 1997.

[SWS05] Ryan Schmidt, Brian Wyvill, and Mario Costa Sousa. Sketch-based modelling with the blobtree, 2005.

[SWSJ05] Ryan Schmidt, Brian Wyvill, Mario Costa Sousa, and Joaquim Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In *2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–62, 2005.

[SZ89] Thomas W. Sederberg and Alan K. Zundel. Scan line display of algebraic surfaces. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 147–156. ACM Press, 1989.

[TCP06] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1160–1168, New York, NY, USA, 2006. ACM Press.

[TSYK01] Satoshi Tanaka, Akihiro Shibata, Hiroaki Yamamoto, and Hisakiyo Kotsuru. Generalized stochastic sampling method for visualization and investigation of implicit surfaces. *Comput. Graph. Forum*, 20(3), 2001.

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 43–50, New York, NY, USA, 1994. ACM Press.

[Tur91] Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *SIGGRAPH Comput. Graph.*, 25(4):289–298, 1991.

[vOW04] Kees van Overveld and Brian Wyvill. Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface. *The Visual Computer*, 20(6):362–379, 2004.

[WBK97] A. Witkin, D. Baraff, and M. Kass. An introduction to physically based modeling. updated from siggraph '95 course, 1997.

[Wei] Eric Weisstein. Mathworld.

[Wei66] Ruth A. Weiss. Be vision, a package of ibm 7090 fortran programs to draw orthographic views of combinations of plane and quadric surfaces. *J. ACM*, 13(2):194–204, 1966.

[Wer88] E. Werner. Toward a theory of communication and cooperation for multiagent planning. In *Theoretical Aspects of Reasoning About Knowledge: Proceedings of the Second Conference*, pages 129–143, 1988.

[WFJ+05] Brian Wyvill, Kevin Foster, Pauline Jepp, Ryan Schmidt, Mario Costa Sousa, and Joaquim A. Jorge. Sketch based construction and rendering of implicit models., 2005.

[WGG99] B. Wyvill, E. Galin, and A. Guy. Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum*, 18(2):149–158, June 1999.

[WH94] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics*, 28(Annual Conference Series):269–277, 1994.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[WK91] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 299–308, New York, NY, USA, 1991. ACM Press.

[Woo02] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd, Chichester, West Sussex, England, 2002. ISBN: 047149691X.

[WS94] G. Winkenbach and D.H. Salesin. Computer-generated pen-and-ink illustration. In *Proc. of SIGGRAPH '94*, pages 91–100, 1994.

[WT90] G. Wyvill and A. Trotman. Ray tracing soft objects. In *Proceedings of Computer Graphics International 1990*, pages 469–476, 1990.

[WvO95] B. Wyvill and K. van Overveld. Constructive soft geometry: The unification of csg and implicit surfaces., 1995.

[WvO97] B. Wyvill and K. van Overveld. Polygonisation of implicit surfaces with constructive solid geometry. *Journal of Shape Modeling*, 2(4):257–273, 1997.

[YT07] Qinxin Yu and Demetri Terzopoulos. A decision network framework for the behavioral animation of virtual humans. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 119–128, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[ZISS04] Johannes Zander, Tobias Isenberg, Stefan Schlechtweg, and Thomas Strothotte. High quality hatching. *Comput. Graph. Forum*, 23(3):421–430, 2004.