# Using Multi-Agent Systems for Sampling and Rendering Implicit Surfaces

Pauline Jepp
INESC-ID
Lisbon, Portugal
pjepp@inesc-id.pt

Jörg Denzinger
University of Calgary
Canada
denzinge@ucalgary.ca

Brian Wyvill
University of Victoria
Canada
blob@cs.uvic.ca

Mario Costa Sousa
University of Calgary
Canada
smcosta@ucalgary.ca

## Abstract

*In this paper a Multi-Agent System for Sampling and Rendering Implicit Surfaces is presented (MASSRIS). Previous approaches to pen-and-ink style renderings of implicit surfaces were based on particle systems, which, for a complex surface, are slow to achieve a good distribution of particles and subsequently to trace features. The method proposed in this research extends traditional particles into semi-autonomous agents that sample the implicit model and illustrate surface features. Agents use goal directed behaviours to achieve a good coverage of surface strokes and feature outline identification faster than with previous particle-based methods.*

## 1. Introduction

In computer graphics, skeletal **I**mplicit **S**urface (IS) modelling techniques have been used to create representations of a variety of objects. Complex IS are slow to evaluate [3] therefore some important work has been done to speed up evaluation of the model definition, such as using spatial caching [21]. **P**article **S**ystems (PS) are used for both faster visualisation and creating Non-Photorealistic Rendering (NPR) styles [8, 19, 10, 23]. Real-time interaction and viewing rates are achieved for relatively simple models [24, 19, 6, 23]. Sampling and rendering more complex models, however, does not achieve comparable results [10, 9]. Speed limitations arise from the method of distributing particles by repulsion.

Traditional PS use a repulsion based distribution approach to cover a surface and also to evaluate local surface properties that determine features and feature outline locations [24, 19, 10, 23]. Particle distribution is not directed towards particular areas of the surface, therefore identification of surface details is haphazard in this respect. When the aim of a rendering is to identify feature outlines a complete coverage of particles is not essential, it is required only inasmuch to identify the location of lines. Similarly, where short strokes or stipples are required, particles produce good results but the PS is neither mandatory nor the only solution.

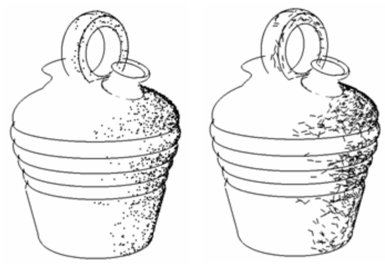
**Figure 1. Jug with stipples and short strokes.**

In this research, the traditional PS is extended into a Multi-Agent System (MAS) that improves on the efficiency of previous particle based methods. Agents identify and trace surface feature outlines faster by using highly improved methods of goal direction and co-ordination.

Efficiency is improved using a combination of object space agents and a manager agent, see Fig 2. A rough polygonisation is used to initialise the MAS, with more precision achieved by querying the implicit model directly. Initially agents analyse polygon and voxel data to estimate surface behaviour and identify potential feature outline locations. This data is stored in an abstraction of the model space known as a blackboard [11, 5]. A manager agent then analyses this data and creates prioritised lists of tasks (e.g. trace a feature outline or explore a region) to which semi-autonomous explorer agents attend. The results of agents exploration is stored in the blackboard and included in subsequent iterations of manager agent analysis and task list preparation. The manager also interprets user requests when creating prioritised task lists. The task descriptions and the method of completing them are therefore more goal directed when compared with a traditional PS approach where feature outlines are only identified when particles happen upon them [19, 6, 10, 23].

The contributions of this research are an object-space agent-based sampling and rendering method that analyses an IS to identify and target feature outline locations rather than using a repulsion based PS approach.

## 2. Background

Complex IS are frequently modelled using hierarchical tree-based methods such as the BlobTree [26], which allows arbitrary compositions of models using blending, warping and boolean operations. Efficient data structures are essential for handling complex implicit models. It is common to use two different structures: (1) model definition (2) visualisation considering rendering strategies [3, 16, 1, 21]. Ray tracing is the most accurate method of visualising an implicit surface, but it is costly [13, 3]. Polygonisation is commonly used for faster visualisation and is usually voxel-based [27, 2, 21]. For faster surface visualisation and stylization PS have been used [24, 19, 6, 10, 23].

Many PS for IS [19, 6, 10, 23] are based on the Witkin-Heckbert (WH) attraction-repulsion model [24]. Particles can be rendered directly, e.g. using discs, short strokes [8, 19, 23] or surface texture elements [9]. Alternatively, their path can be used, e.g. when illustrating a silhouette [10]. Particle density can be varied, they either appear visually disconnected [24] or provide continuous coverage [9].

"Smarter" particles were developed to overcome some of the limitations associated with basic PS. The motivation for the smart particles (*sparts*) introduced in [15] is to provide a programmable tool for visualisation of scientific data. User controlled spray cans are filled with different types of sparts that perform different tasks. The *smarticles* used in [12] were particles from [10] that used an abstract view of a BlobTree object and were also given behaviours to place strokes.

Cellular Texture Generation (CTG) [9] is achieved using a PS to create patterns of geometric elements that are generated using a biologically motivated simulation, which is based on the principles of morphogenesis. A WH style PS is extended to include cell-cell interactions, or behaviours, (based on Reynolds' work [17]). Cells have an environment and combinations of cell programs that define a cell's behaviour, e.g. go to a surface, divide to cover and reaction diffusion.

Agent based methods are commonly used in image space methods. In [22] user collaboration with an artificial ant colony progressively transforms photographs into stylised pictures. In [14] an agent based system helps artists to express ideas. The semi-autonomous agents represent graphical elements that are combined with a balance between artistic expression and algorithmic support. RenderBots [20] are autonomous agents that represent one stroke in one style, multiple RenderBots create an image.

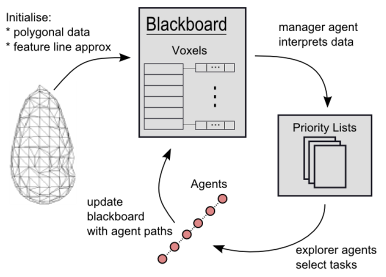

**Figure 2. System process flow overview.**

In MAS literature a *blackboard* [11] is a shared memory or common area where agents are permitted to both read and write. In a traditional blackboard system [11] a central control unit (*scheduler*) is an integral element. In [5], Craig introduces a new interpretation of the blackboard metaphor where control is more distributed, i.e. the agents themselves decide which information to make available and to act upon. Agents are also commonly used in groups to solve problems that may be beyond the ability of one individual [25, 11].
**Definition:** An **Implicit Surface** [3] $S$ is composed of the set of points derived from a field function (*Implicit Function*) $f()$ as: $S = \{p \in \Re^3 : f(p) = iso\}$ where $iso$ is a constant value defining the iso-surface and $p = (x, y, z)$.
**Definition: A Multi-Agent System** (MAS) usually consists of a group of computing elements (agents) in an environment. Agents have a method of collaboration that defines them as being part of a system rather than unrelated individuals. An accepted generalisation states an agent **Ag** is defined by a 4-tuple $Ag = (Sit, Act, Dat, f_{Ag})$, consisting of situations **Sit**, actions **Act**, internal data **Dat** and a decision function $f_{Ag}$ [7]. Situations are those in which an agent may find itself. Internal data is the set of possible values the data can have. The set of actions are those that the agent can perform. And the decision function connects the situation and relevant action $f_{Ag} : Sit \times Dat \rightarrow Act$.

## 3. System overview

In contrast to a conventional PS, the MASSRIS approach does not aim to cover the entire surface nor does it use repulsion based distribution. Agents are given a degree of autonomy that allows them to identify potential locations of feature outlines and suitable positions for placing stipples or strokes. A global view of the model is analysed to identify potential features, which is then used to direct agents.

The MASSRIS system uses not only the IS model (*model space*) and the representation of the rendered object (*rendering space*), but also an abstract space in between these, described as a blackboard (*blackboard space*). A manager agent (hereafter called the *manager*) is placed between the IS model and the task performing agents (called *agents*).

The manager administers the blackboard to identify and prioritise tasks, and also co-ordinate agents' activities.

In this paper, first we concentrate on the structure of the blackboard (Sec. 4), then we explain how the manager creates and prioritises tasks for the agents (Sec. 5). Finally we describe the structure of the agents and their decision making process using the blackboard and behaviours (Sec. 6). Stippling results are shown in Sec. 7 and results showing efficiency improvements are presented in Sec. 8.

## 4. Blackboard Space

The blackboard space is a voxel based public repository that is an abstract representation of the IS model space. It contains the results of agents' examination and exploration of the model space. Some of these results are used to create the final image, i.e. feature outline traces and surface marks (stipples and strokes), whilst others are not rendered i.e. agent exploration results. Thus, rendering space can be seen as a subspace of blackboard space.

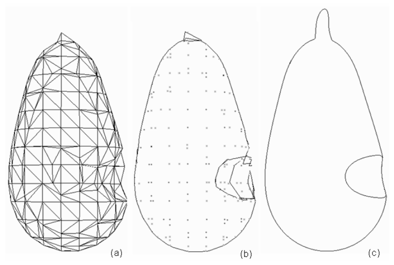

**Figure 3. (a) The initial rough polygonisation of the pear. (b) Polygon vertices and approximations to feature lines. (c) Traced feature lines.**

The manager initialises the blackboard with data from a low resolution polygonisation that has been examined by a team of agents to identify potential feature outline locations (Fig. 3). Each polygon edge is examined for crossing a silhouette edge, a discontinuity (or small scale detail) and for changes in the sign of the curvature (convex to concave and vice versa). Linear interpolation is used for a fast approximation to the feature outline location.

Using the polygon vertices as initial positions for agents (Fig. 3 **b**) provides sufficient coverage (both in terms of area and density) of the surface. Note that when particles in previous systems reach equilibrium they are, by nature of the algorithm, regularly spaced. Therefore, the regularity of the polygon vertices does not introduce a new problem.

The polygonisation is, in general, not fine enough to guarantee that all features are found; e.g. in Fig. 3 **a** and **b** the stalk of the pear has not been identified. Neither does it provide enough information about the voxels to be a definitive representation of the model; e.g. in Fig. 3 **b** the bite from the pear, a discontinuous region, is not accurately illustrated. Using a low resolution polygonisation is, however, a faster method of achieving a good surface coverage and identifying potential feature locations than using a WH style attractor-repulsion method (see Sec. 8 for timing results).

During runs of the system, the blackboard is updated with results from agents performing tasks. More accurate feature outline traces (referencing the model space) replace the approximations from the initialisation (polygonisation) stage (Fig. 3 **c**). Assumptions made about the contents of a voxel, and therefore about the behaviour of that part of the surface can not be guaranteed. Small scale details may be missed due to the sampling frequency. As more information is accumulated from agent samples, assumptions are updated and refined using newly calculated statistics. More information means better sampling which gives a better likelihood of identifying details (Fig. 3 **c**).

The information stored in each voxel includes the integer-triple voxel identifier (back bottom left corner), and its real-valued co-ordinate $(x, y, z)$. Samples (field function, gradient, mean and Gaussian curvature, and principal directions of curvature) and statistics of samples (average, range and standard deviation) are stored for each corner, polygon vertex and agent path step.

In most tasks, agents are constrained to the region close to the surface of the object, therefore, only surface voxels (and potentially their neighbours) are stored. The relevant voxels are stored using a hash table (with buckets) with the voxel identifier used as the key [27]. A complete list of surface voxels can not always be guaranteed. Where the list is incomplete agents can identify other voxels through exploration.

## 5. The Manager

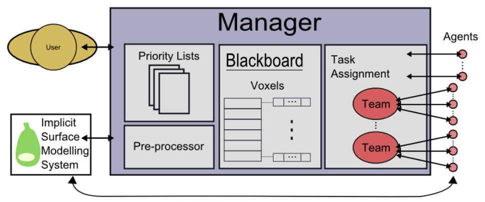

**Figure 4. The manager agent overview.**

The manager agent is responsible for administering the blackboard and interpreting user requests into prioritised tasks that are assigned to agents for completion. The manger, therefore acts as an interpreter be-

| User requests | Agent tasks |
|---|---|
| Trace feature outlines | Trace feature outlines |
| Create a single stroke | Create a rendering path |
| Place strokes | Create rendering paths |
| Find features (more detail) in a region | Explore a region<br>Identify features or points on a feature outline |

**Table 1. User requests and associated tasks.**

tween the model, blackboard and rendering spaces. When agents change voxels or complete tasks, they report all relevant path sample information to the manager for inclusion in the blackboard. Agents indicate whether data they report is relevant to rendering or exploration (not constrained to the surface). The manager interprets this, thereby creating the rendering space as a subset of the blackboard space.

## 5.1. User Requests and Task Identification

Tasks are central to the MASSRIS system. Table 1 describes the relationship the manager assumes between user requests and agent tasks. A single user request can fire multiple tasks, e.g. the user selects to create more detail in a region, the manager requests a team of agents explore the region (blackboard space), which will also identify and trace any feature outlines found (rendering space), and report back to the manager with data for the blackboard.

## 5.2. Task Priorities

Agents are capable of performing many tasks, at most points in time there are a number of tasks to be performed. If there are only a limited number of agents, e.g. one agent per available processor, then the order of task performance must be decided. The manager agent makes this decision by maintaining one or more task priority lists.

Priorities are calculated, and respectively updated, for voxels identified to be affected by the completion of a task by an agent (neighbourhood). For example when tracing the bite (discontinuous region) in the pear (Fig. 3), all of the voxels identified to contain part of the feature outline are evaluated and given a priority rating. Task *priorities* are calculated in blackboard space using estimates of *workload* and *workdone*. The estimates use field function and gradient values for voxel corners and agent path steps.

**5.2.1. Workload** Workload is estimated in reference to the perceived size and behaviour of the surface inside a voxel. Surface voxels are identified using corner and sample information, i.e. where the voxel contains field function values both above and below $iso$. In general, surface voxels can be identified by examining their corners. The number of corners inside the surface can be used to represent the approximate size of the surface inside the voxel. The variance of the angles between corner gradients can estimate the behaviour of the surface, i.e. if it contains a discontinuity, small scale detail, or is relatively smooth, see Fig. 5. Surface voxels can also be identified by agent samples.

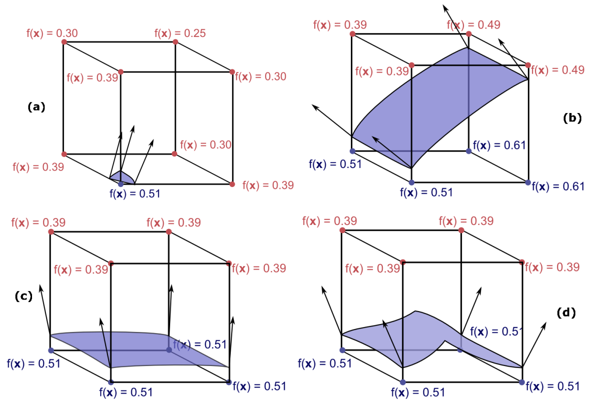

**Figure 5. Voxel corners are inside (red) or outside (blue) the surface. Field function and gradients can identify, e.g.: (a) a small part of the surface, (b & c) a larger smooth area, (d) a discontinuity or small scale detail.**

The number of corners inside the surface provides the first step in estimating the workload $wl^c$ for a voxel. If the number of corners inside the surface is either 1 or 7 then $wl^c \equiv 0.25$, [2 or 6]: $wl^c \equiv 0.5$, [3 or 5]: $wl^c \equiv 0.75$, 4: $wl^c \equiv 1.00$. If a voxel does not have any corners inside the surface and at least one sample has identified part of the surface, the value of $wl^c$ is set to $0.25$, i.e. it is assumed to contain a relatively small part of the surface, which is acceptable as the polygonisation has not identified the voxel.

The perceived behaviour of the surface in the voxel is the second means of estimating workload. If the angle between gradients is relatively small then it can be assumed the surface in the voxel is relatively flat (Fig. 5 (**a**) to (**c**)). Where the angle between the gradients is divergent (Fig. 5 (**d**)), this identifies a potential feature or detailed part of the surface. The second estimate used to calculate workload $wl^g$ therefore uses (normalised) gradient information : $wl^g = \left(\frac{\theta_i}{\Theta_{max}}\right)$ where $\theta_i$ is the maximum angle between gradients in voxel $i$, and $\Theta_{max}$ is the maximum angle in all neighbourhood voxels.

The last workload calculation uses the normalised (user specified) Level Of Detail (LOD): $0 \leq wl^d \leq 1$.

These three values are summed to provide the total workload estimate $wl_i$ for each voxel $i$:

$$wl_i = \left( \frac{wl_i^c + wl_i^g + wl_i^d}{wl_{max}^c + wl_{max}^g + wl_{max}^d} \right) \tag{1}$$

where $wl_{max}^c$, $wl_{max}^g$ and $wl_{max}^d$ are the maximum values of corners inside, gradient divergence and LOD, respectively, for the neighbourhood.

**5.2.2. Workdone** Workdone is estimated using tallies of the number of samples and feature outline points identified. The number of samples $wd_i$ can identify whether a voxel has been well sampled (relative to the neighbourhood) or requires further investigation. The workdone sample-estimate $wd_i^{'}$ is normalised: $wd_i^{'} = \frac{wd_i}{wd_{max}}$ where $wd_{max}$ is the maximum number of samples in the neighbourhood.

An important piece of information is whether a feature has been identified in the voxel. In order to calculate the workdone for each feature found the following heuristic is used: $wd^g = \left( \frac{wl_i^g}{2^{\mu_i}} \right)$ where $\mu_i$ is the number of feature points in the voxel $i$ (or the number of steps on the trace's path).

This is used to modify the results from Eq. 1, so that:

$$wl_i^{'} = \left( \frac{wl_i^c + wd^g + wl_i^d}{wl_{max}^c + wl_{max}^g + wl_{max}^d} \right) \tag{2}$$

.

**5.2.3. Priority** To calculate the final priority rating, $P$, the workload, $wl_i$, must be offset against the workdone, $wd_i$:

$$P^i = wl_i - (wl_i^{'} * wd_i^{'}) \tag{3}$$

This gives each voxel a priority rating that identifies where agent activity should be applied first.

## 6. Task Performing Agents

An agent is an extension of a particle with added abilities that allows it to act autonomously in reaction to its situation. Although agents are aware of their neighbours their movements are not dictated by others i.e. they do not rely on a repulsion distribution method, they are self motivated. Agents are designed to perform tasks by referencing accurate sample data from the model space rather than a polygonal approximation. When an agent changes voxels or finishes a task, it reports any relevant information (path history sample data) to the manager. If the task is to create paths for strokes, this information will also be used for rendering. Otherwise, the task is exploratory so the data will remain solely in blackboard space.

In this research, the actions available to an agent are **move**, **stop**, **report**, and **identify feature point**. When an agent moves it references the model space to determine the sample information at its new position. At voxel change or task completion the agent updates its personal current-voxel path history to the manager for inclusion in the blackboard space.

A situation is defined by *what* an agent can see i.e. its view of the blackboard and model spaces and *who* it can see, i.e. any teammates (only for tasks which need a team).

An agent's Internal Data Values (IDV) include its history, assigned behaviour attributes, and team and task identification. Behaviour attributes are position and velocity, steering direction calculation method (e.g. $normal \cdot viewVector$ to follow the silhouette), visibility of the agent's actions (i.e. include results in rendering space) and termination criteria, see below for a description of the behaviours.

The decision function $f_{Ag}$ relates the situation and IDV to the action to be performed e.g., if the task is to explore and the termination criteria have not been met, then $f_{Ag}$ will first select the move action, and then compute the parameters that define the exact move using data from IDV and referencing the model space.

**Teams and Behaviours** :

**Teams** of agents are used to complete certain tasks, e.g. Dual Tracking a discontinuity outline [10], or flocking strokes [12]. Agents query team members properties (e.g. position and velocity) through the manager.

Agents use **behaviours** to perform actions that complete tasks. These behaviours include: polygon processing; steering and flocking; finding feature points; and tracing feature outlines. An agent can be influenced by more than a single behaviour. The behaviours that are active in an agent mostly influence the **move** action and they determine if the consequences of an action are only realised in blackboard space or also in rendering space.

The *polygon processing* behaviour is used to analyse the initial polygonisation to determine if a feature outline has been crossed. Feature outlines identified include silhouettes, discontinuities (or abrupt blends) and changes in the sign of curvature.

*Steering* behaviours determine how a particle moves and when it stops. Steering behaviours are inspired by Reynolds' steer, wander, flow field following and path following behaviours [18]. Stopping a particle is inspired in part by Reynolds' containment and arrival behaviours [18].

Agents can *identify feature points* in a similar manner as basic particles: they examine their current and previous step to identify if a feature outline has been crossed [10]. They do not rely, however, on a chance encounter and actively seek out a region that potentially contains a feature outline location.

Agents *trace feature outlines* using a similar numerical integration predictor-corrector method as previous methods [4, 10]. The main difference is that the next step direction (move) is calculated by the agent itself as its steering direction.

## 7. Rendering

Rendering space is a subspace of blackboard space, the manager interprets agents' paths to determine its contents. The user also has a degree of control over the visibility of the path steps or strokes used in the rendering.

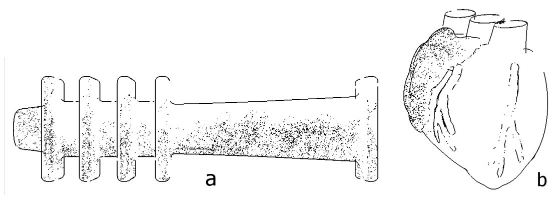

**Figure 6. (a) Dyed with wander behaviour stipples. (b): Stipples on the heart model.**

Short strokes similar to those presented in [10] and long strokes based on those from [12] are both available with the MASSRIS system. Short *general* surface strokes (curved and straight) are created using the original polygonisation to identify the center of the polygon and the center of the stroke. Methods similar to those in [10] are used for calculating lighting and stroke visibility. Longer agent paths are placed using behaviours in user specified regions.

Stippling effects can be produced by only drawing step points (or a subset) from agent paths. Most of the paths are regular (by nature) and appear as dotted lines, the wander behaviour, however, creates paths that are indiscernible and are used to create stipples or short strokes see Figs 1 and 6.

## 8. Results

Although WH [24] used a particle based method for fast visualisation of IS the models they use are relatively simple, i.e. they use only a few primitives. The method used in CTG [9] is extremely slow (in the manner of hours) for large datasets. In more current research, the PS described in [23] by Su and Hart also uses only simple implicit models with few primitives, or a Steiner surface that is defined by only one quadratic equation. There is very little relevant material in the current literature dealing with complex implicit models, which makes comparison with other methods difficult, therefore the results of this research are compared to previous systems described in [10] and [12], hereafter called NPRBT.

The BlobTree models used in Fig. 7 have many nodes: the shell has 1497, the train has 723 and the heart has 76. Rendering times (using the NPRBT) for the shell and the train frequently exceed 10 minutes. The BlobTree evaluations are therefore a major bottleneck in rendering the objects.

Fig. 7 illustrates the increase in speed of automatic visualisation of feature outlines. A similar pattern emerges for all models: the MASSRIS system uses a larger number of operations in the pre-processing stage, and fewer iterations for exploration and outline tracing, which results in faster visualisation. Initialisation of the NPRBT uses 1000 particles randomly placed using a ray tracing method. Initialisation of the MASSRIS system involves a team of three agents processing a low resolution polygonal mesh.

In Fig. 7 (top) the results are shown for the shell images. The MASSRIS system uses the team of 3 agents for initialisation, with the resolution of the polygonisation grid at $25^3$. Initialisation has a much larger number of system iterations using MASSRIS, which enables identification of many feature locations. Both results are produced automatically and use the default settings.

In Fig. 7 (middle) **(a)** to **(c)** the train is modelled with the NPRBT. In **(d)** to **(f)**, the MASSRIS system is used with a voxel resolution of $15^3$. The user selected "Trace All Feature Outlines" with 10 agents, there was no other user interaction. Again we see that MASSRIS accomplishes more feature outline identification and tracing than the NPRBT in the same time.

Fig. 7 (lower) shows the results of the heart model. In both cases user interaction was used to more fully identify the feature outlines. In **(b)** the user increased the number of particles to approx 2700, allowing automatic placement of new particles, but new particles are not ideally placed to identify outlines. Using the MASSRIS approach in **(d)** (grid resolution $20^3$) the user directed agents to the veins, which are difficult to trace as they are not constructed using simple smooth primitives. Using both systems many particles or agents were required to trace the vein outlines using many short strokes. Neither system was able to trace the vein outlines using a single particle or agent.

The main difference to note regarding 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). The MASSRIS system, however, traces by taking one step with each system iteration. Another important distinction to make is that the MASSRIS system iteration count starts from zero after initialisation. This is because the agents' tasks are very different between the initialisation and exploration stages. This also gives a better indication of sys-

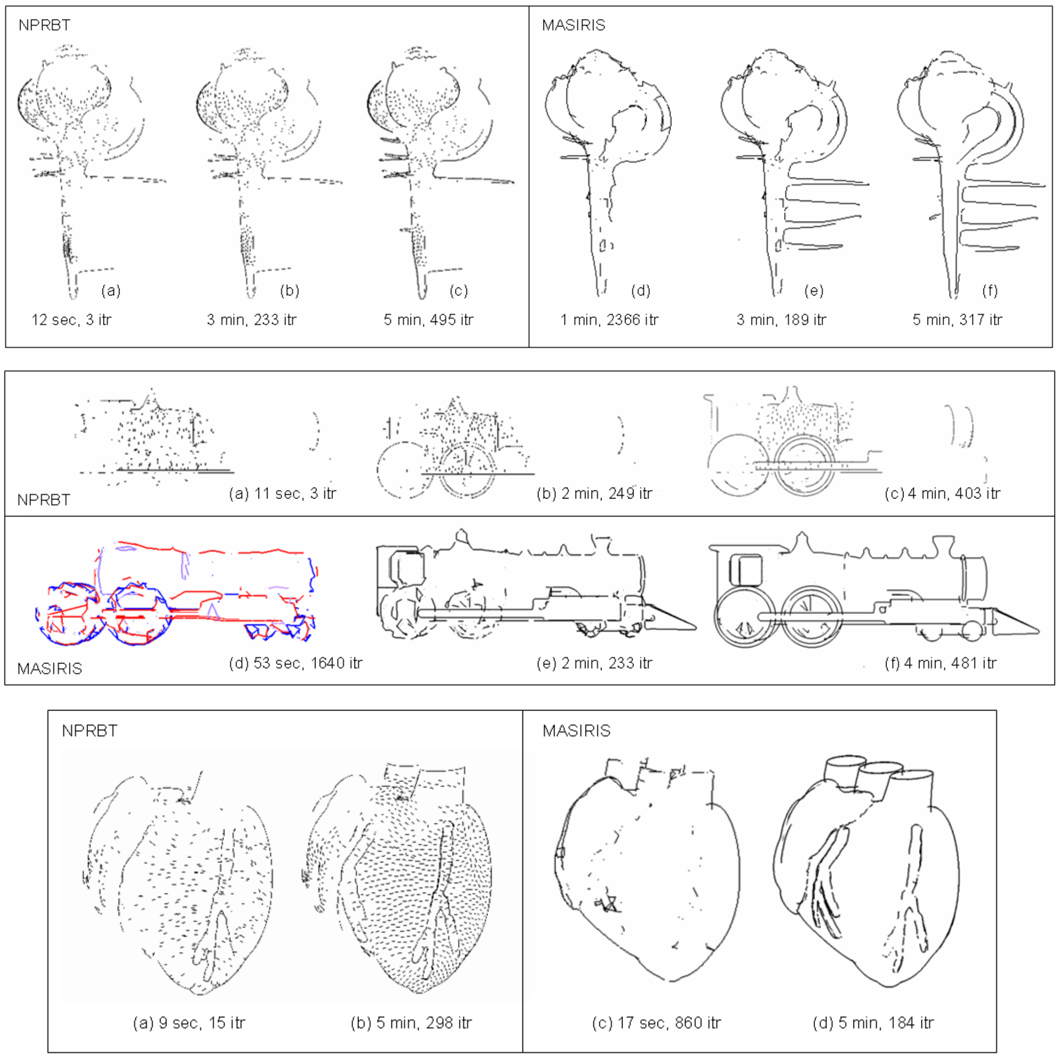

**Figure 7. Automatic feature outline illustration. After initialisation. Number of system iterations for the models.**

tem iterations involved in exploring the space and creating the images. The number of iterations after the initialisation stage is cumulative.

The final images in each sequence were created in equal lengths of time (including initialisation) on a single machine. In most cases a single system iteration takes longer with the MASSRIS system, but, as each movement is directed towards finding a feature rather than using simple repulsion, more features are identified.

## 9. Conclusions and Future Work

In this research, we have presented a MAS that improves the efficiency of identifying and tracing feature outlines on a complex implicit surface and provides an alternate method of positioning strokes and stipples. Our technique uses a manager to administer a blackboard space, which is used to store and analyse data about the model space in order to create a rendering space. Our experiments showed that the efficiency is improved over previous approaches, and the method of placing general surface strokes is facilitated by the pre-processing stage of a low resolution polygonisation of the implicit object. The method of creating stipple patterns using a wander steering behaviour also expands the palette of previous pen-and-ink style renderers that use particles covering a surface.

The main drawback of using the initial low resolution polygonisation for hidden line removal is that the polygonisation is not an accurate approximation of the surface and at silhouette regions artifacts are visible. Our solution to this is to use a voxel subdivision method where polygons will be recalculated using subdivided voxels. This will create a better approximation to the surface locally, and also will create polygons in regions that were missed due to the low resolution polygonal sampling.

The MASSRIS system could be adapted to use other modelling systems or sources of data. 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 is commonly used). The MASSRIS method could be used to analyse samples and estimate the behaviour of the surface.

There are particular applications of the MASSRIS method to animation and changing topologies. 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.

Agents' actions are deliberately designed to easily include more behaviours. A number of additional behaviours have also been developed, such as agents identifying regions of the surface to be coloured in a style reminiscent of concept art. The feature outline tracing behaviours are easily adapted to trace apparent ridges and suggestive contours.

## References

[1] R. Allegre, E. Galin, R. Chaine, and S. Akkouche. The hybridtree: Mixing skeletal implicit surfaces, triangle meshes, and point sets in a free-form modeling system. *Graphical Models*, 68(1):42–64, 2006.

[2] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[3] J. Bloomenthal, C. Bajaj, J. Blinn, M. Cani-Gascuel, A. Rockwood, B. Wyvill, and G. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

[4] D. Bremer and J. Hughes. Rapid approximate silhouette rendering of implicit surfaces. In *Proc. of Implicit Surfaces '98*, pages 155–164, 1998.

[5] I. Craig. A new interpretation of the blackboard architecture, technical report 254 dept computer science, university of warwick, 1993.

[6] P. Crossno and E. Angel. Isosurface extraction using particle systems. In *Proc. of Visualization '97*, pages 495–ff., 1997.

[7] J. Denzinger and J. Hamdan. Improving observation-based modeling of other agents using tentative stereotyping and compactification through kd-tree structuring. *Web Intelligence and Agent Systems: An International Journal*, 4(3):255–270, 1999.

[8] G. Elber. Line art illustrations of parametric and implicit forms. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):71–81, Jan. 1998.

[9] K. Fleischer, D. Laidlaw, B. Currin, and A. Barr. Cellular texture generation. In *Proc. of SIGGRAPH '95*, pages 239–248, New York, NY, USA, 1995. ACM.

[10] K. Foster, P. Jepp, B. Wyvill, M. Sousa, C. Galbraith, and J. Jorge. Pen-and-ink for blobtree implicit models. *Computer Graphics Forum (EG '05)*, 24(3):267–276, 2005.

[11] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26:251–321, 1985.

[12] P. Jepp, B. Wyvill, and M. C. Sousa. Smarticles for Sampling and Rendering Implicit Models. In *Theory and Practice of Computer Graphics, Eurographics UK Chapter Proc. (EGUK '06)*, pages 39–46, 2006.

[13] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *Proc. of SIGGRAPH '89*, pages 297–306, 1989.

[14] K. Mason, J. Denzinger, and S. Carpendale. Negotiating gestalt: Artistic expression and coalition formation in multiagent systems. In *Proc. of 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '04)*, pages 1350–1351, 2004.

[15] A. Pang and K. Smith. Spray rendering: visualization using smart particles. In *VIS '93: Proceedings of the 4th conference on Visualization*, pages 283–290, 1993.

[16] A. Pasko and V. Adzhiev. Function-based shape modeling: Mathematical framework and specialized language. In *Automated Deduction in Geometry*, pages 132–160, 2002.

[17] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, volume 21, pages 25–34, 1987.

[18] C. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, 1999.

[19] A. Rosch, M. Ruhl, and D. Saupe. Interactive visualization of implicit surfaces with singularities. *Computer Graphics Forum*, 16(5):295–306, 1997.

[20] S. Schlechtweg, T. Germer, and T. Strothotte. RenderBots—Multi Agent Systems for Direct Image. *Computer Graphics Forum*, 24(2):137–148, 2005.

[21] R. Schmidt, B. Wyvill, and E. Galin. Interactive implicit modeling with hierarchical spatial caching. In *Proc. of Shape Modeling and Applications*, pages 104–113, 2005.

[22] Y. Semet, U. O'Reilly, and F. Durand. An interactive artificial ant approach to non-photorealistic rendering. In *Genetic and Evolutionary Computation*, pages 188–200, 2004.

[23] W. Su and J. Hart. A programmable particle system framework for shape modeling. In *SIGGRAPH '05 Course*, page 277, 2005.

[24] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. *Proc. of SIGGRAPH '94*, pages 269–277, 1994.

[25] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd, Chichester, West Sussex, England, 2002. ISBN: 047149691X.

[26] 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.

[27] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, 1986.