Topological Correction of Hypertextured Implicit Surfaces for Ray Casting

Manuel N. Gamito
Steve C. Maddock
The University of Sheffield
Department of Computer Science
211 Portobello Street, Sheffield S1 4DP, UK
{M.Gamito,S.Maddock}@dcs.shef.ac.uk

Abstract

Hypertextures are a useful modelling tool in that they can add three-dimensional detail to the surface of otherwise smooth objects. Hypertextures can be rendered as implicit surfaces, resulting in objects with a complex but well defined boundary. However, representing a hypertexture as an implicit surface often results in many small parts being detached from the main surface, turning an object into a disconnected set. Depending on the context, this can detract from the realism in a scene where one usually does not expect a solid object to have clouds of smaller objects floating around it. We present a topology correction technique, integrated in a ray casting algorithm for hypertextured implicit surfaces, that detects and removes all the surface components that have become disconnected from the main surface. Our method works with implicit surfaces that are C^2 continuous and uses Morse theory to find the critical points of the surface. The method follows the separatrix lines joining the critical points to isolate disconnected components.

1. Introduction

Hypertexturing is a procedural technique proposed by Perlin & Hoffert to add three-dimensional small-scale detail to the surface of smooth objects [17]. It can be used to model a large collection of materials such as fur, fire, glass, fluids and eroded rock. As a procedural modelling and texturing tool, hypertexturing can be regarded as an improvement over solid texturing [15]. Using hypertextures, it becomes possible to actually deform the surface of an object instead of merely modifying its material shading properties.

Following Perlin & Hoffert, we give here a definition of hypertexture. An object is defined in three dimensional space with an *object density function* $D : \mathbb{R}^3 \to [0,1]$, which associates a density value $D(\mathbf{x})$ with every point $\mathbf{x}$ in space. The shape of the object can then be deformed through the composition of D with one or more *density modulation functions* $f_i : [0,1] \to [0,1]$ such that the final object density H is given by:

$$H(D(\mathbf{x}),\mathbf{x}) = f_n(\dots f_2(f_1(f_0(D(\mathbf{x}))))). \qquad (1)$$

The visualisation of a hypertextured object is a volume rendering task. For every pixel, a ray must be marched by taking a sequence of small steps and accumulating density and opacity values along the way [10]. This rendering method shows objects with a fuzzy appearance, which do not have an exact surface separating an inside from an outside volume.

It is possible to increase the sharpness of the surface for a hypertextured object by using a *gain function* g_α as the last density modulation function in the composition sequence (1) for H [17]. The sharpened hypertexture is:

$$H_\alpha(D(\mathbf{x}),\mathbf{x}) = g_\alpha(H(D(\mathbf{x}),\mathbf{x})) = \\ = g_\alpha(f_n(\dots f_2(f_1(f_0(D(\mathbf{x})))))). \qquad (2)$$

The gain function g_α pushes the density values in the range $[0,0.5]$ towards 0 and the values in the range $[0.5,1]$ towards 1. The gain $0 \le \alpha \le 1$ controls the amount of sharpening. When $\alpha = 1$, g_α becomes a step function, switching from 0 to 1 at the middle of the $[0,1]$ interval. In this case, the hypertextured object becomes an *implicit surface*. If we define $F : \mathbb{R}^3 \to [-0.5,+0.5]$ such that $F(D(\mathbf{x}),\mathbf{x}) = H_{\alpha=1}(D(\mathbf{x}),\mathbf{x}) - 0.5$, the surface of the object with density $H_{\alpha=1}$ will be the locus of points given by the set $\{\mathbf{x} \in \mathbb{R}^3 : F(D(\mathbf{x}),\mathbf{x}) = 0\}$. The hypertextured object is now a hypertextured implicit surface with a well defined boundary, where $F > 0$ is verified for points inside the surface and, correspondingly, $F < 0$ for points outside.

Visualising a hypertextured implicit surface with ray marching leads to a rendering method that is both inefficient and unreliable. It is inefficient because too many samples

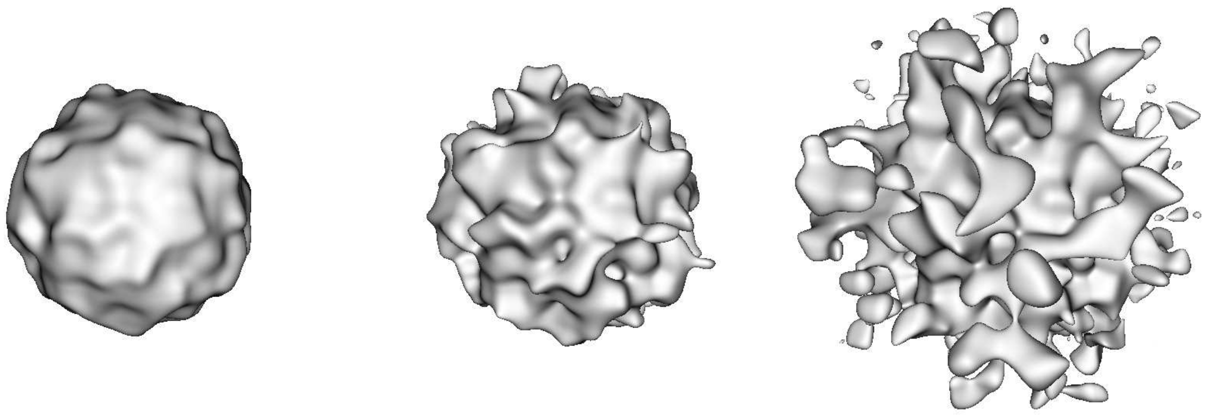

Figure 1. A sphere rendered with increasing amounts of hypertexture ($\alpha = 0.1$, 0.3 and 0.8).

of F must be taken along a ray, trying to find a transition from $F < 0$ to $F > 0$. It is also unreliable because two or more roots $F = 0$ may easily pass unnoticed inbetween two negative samples. Robust methods for finding the intersection between a ray and the surface use range estimation techniques based on either interval arithmetic or Lipschitz bounds, the latter case being used when F is Lipschitz continuous [13, 9, 2]. Since, for ray-surface intersection purposes, we are only interested in finding a transition from negative to positive F, it is no longer necessary to enforce the restriction that density functions must return values in $[0, 1]$ and all of D, H and H_α (and, consequently, F) can now be defined from $\mathbb{R}^3$ to $\mathbb{R}$.

2. The Surface Splitting Effect

We illustrate the splitting effect of hypertextured surfaces with an example. Figure 1 shows three implicit surfaces that have been generated by adding increasing amounts of hypertexture. The function that generates these surfaces is:

$$F(D(\mathbf{x}), \mathbf{x}) = D(\mathbf{x}) + \varepsilon\, n(4\mathbf{x}), \tag{3}$$

where $D(\mathbf{x}) = 1 - \|\mathbf{x}\|$ defines an implicit sphere of unit radius and n is Perlin's improved gradient noise function [16]. The amplitude ε of the hypertexture takes values of 0.1, 0.3 and 0.8 for the three surfaces in Figure 1. The case $\varepsilon = 0.1$ shows a surface with a small amount of perturbation relative to the initially smooth sphere. This type of surface could more easily have been modelled as a procedural displacement map [6]. The case $\varepsilon = 0.3$ generates an object with more pronounced surface features but which, from a topological point of view, is still homeomorphic to a sphere. The case $\varepsilon = 0.8$ generates an object with the interesting overhanging and arching features that only the implicit surface approach can give. At the same time, it also causes the surface to split, generating a cloud of small objects that are seen floating at fixed locations around the main object in the centre. Depending on the context, this surface splitting effect may be desirable or not. If one is using hypertextured implicit surfaces to model splashing fluids, for example, then the surface splitting effect is actually beneficial. If, on the other hand, one is trying to model a solid object with a complex surface structure, e.g. a rock, the case $\varepsilon = 0.8$ of equation (3) leads to physically non-plausible results.

The splitting effect places an upper bound on the amount of hypertexture that can be added to an object while keeping it as a topologically connected set. This bound depends on the particular function F that generates the surface and can only be found by trial and error. It is not an uncommon practice, when an excessive amount of hypertexture has been applied over a solid object, to digitally remove disconnected parts from the final rendering as a post-processing step.

In this paper, we propose a technique that allows us to go beyond the upper amplitude bound that exists for connected hypertextured surfaces by detecting and removing disconnected surface components other than the main surface. To achieve this goal we rely on Morse theory to analyse the topology of the surface. The surface connectivity, in particular, can be completely determined by studying the critical points of the function F and the way they are joined together. We apply our technique as part of a ray casting algorithm for hypertextured implicit surfaces.

Section 3 presents two simple methods to solve the surface connectivity problem and explains their limitations. Section 4 explains the necessary concepts from Morse theory that will be required in Section 5, where we present our proposed algorithm. Section 6 shows results and Section 7 presents our conclusions. Finally, Section 8 suggests an extension of our algorithm that can be applied to implicit surface polygonisers.

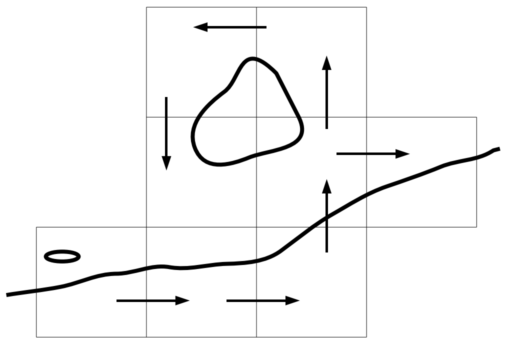

Figure 2. Two surface components incorrectly determined to be part of the main surface. The arrows show the region growing sequence, starting from the voxel on the bottom left.

3. Alternative Approaches

A simple but inaccurate way to perform topological correction on hypertextured surfaces is to employ a voxel grid, where the function $F(D(\mathbf{x}), \mathbf{x})$ is sampled at the voxel corners. A voxel is known to straddle the surface if the function changes sign at some of the voxel's eight corners. One can perform a discrete three dimensional region growing process to segment the voxel space into disjoint volumes, each enclosing a particular disconnected component of the surface. If the original data is already discrete, e.g. a series of MRI scans, then this is probably the best approach to take. When generating an isosurface from the volume data, this voxel-based method can be used to remove outlying surface components that may be the result of measurement error. In our case, we are interested in performing topological correction of *procedurally defined surfaces*. Sampling $F(D(\mathbf{x}), \mathbf{x})$ onto a grid implies loss of information unless the function happens to be bandlimited and the sampling frequency is above the Nyquist limit. This loss of information leads to incorrect connectivity results, as shown in Figure 2, which can occur for surface components that are too small or too close to the main surface, relative to the sampling distance.

Another possible approach is to first convert the implicit surface into a polygonal mesh before performing any topology correction. Stander & Hart have presented a meshing algorithm for implicit surfaces that is guaranteed to preserve surface topology [21]. Once the polygonal mesh has been generated, one can perform region growing by jumping across the edges shared by neighbouring polygons to obtain a set of disjoint polygonal objects. One of these objects approximates the main implicit surface and the others represent the outliers that should be eliminated. One objection against this approach is that it cannot be used for direct rendering of implicit surfaces with ray casting – it is only meaningful for applications where implicit surfaces are converted to polygonal meshes and subsequently rendered on a GPU board. Another objection is that it is wasteful of CPU cycles since it takes time to correctly polygonise surface components that are later found to be disconnected and which must then be removed. Topology correction should occur before the meshing process rather than after.

4. Morse Theory and the CW-Complex

Morse theory studies the behaviour of functions that are defined over a manifold [12]. The theory was first introduced to computer graphics by Shinagawa et al. and was later shown by Hart to be relevant for the topological study of implicit surfaces [19, 3]. When the theory is applied to implicit surfaces, the manifold becomes the whole of the $\mathbb{R}^3$ space and the function defined over this space is our function F that generates the surface. Central to the Morse theory is the notion of a *critical point* of F. A critical point $\mathbf{x}_C$ is such that:

$$\nabla F(D(\mathbf{x}_C), \mathbf{x}_C) = 0. \tag{4}$$

A critical point can be further classified by studying the eigenvalues of the Hessian matrix of F at $\mathbf{x}_C$. The Hessian matrix $\mathcal{H}\{F\}$ collects all the second partial derivatives of the function F:

$$\mathcal{H}\{F\} = \begin{bmatrix} \dfrac{\partial F^2}{\partial x_1^2} & \dfrac{\partial F^2}{\partial x_1 \partial x_2} & \dfrac{\partial F^2}{\partial x_1 \partial x_3} \\ \dfrac{\partial F^2}{\partial x_2 \partial x_1} & \dfrac{\partial F^2}{\partial x_2^2} & \dfrac{\partial F^2}{\partial x_2 \partial x_3} \\ \dfrac{\partial F^2}{\partial x_3 \partial x_1} & \dfrac{\partial F^2}{\partial x_3 \partial x_2} & \dfrac{\partial F^2}{\partial x_3^2} \end{bmatrix}. \tag{5}$$

If F is C^2 continuous then we have $\partial F^2/\partial x_i \partial x_j = \partial F^2/\partial x_j \partial x_i$ and the Hessian is symmetric. The spectral theorem then guarantees that all three eigenvalues of $\mathcal{H}\{F\}$ will be real. Depending on the signs of the eigenvalues λ_1, λ_2 and λ_3, sorted in increasing order, a critical point can be classified as follows:

λ_1	λ_2	λ_3	**Type**
$-$	$-$	$-$	Maximum
$-$	$-$	$+$	2-saddle
$-$	$+$	$+$	1-saddle
$+$	$+$	$+$	Minimum

The type of a critical point gives an indication of the topology of the surface around that point. For example, maxima occur near the local centroids of the surface while 2-saddles occur at the points where two surface components are joined together. In this paper, we only need to be concerned with maxima and 2-saddles in order to characterise the connectivity of the surface.

The case where one or more of the eigenvalues is zero leads to a *degenerate critical point*. Morse theory breaks down in these circumstances. Degenerate critical points, however, are unstable and can easily be removed by introducing a small perturbation in the parameters defining the function. A function F that contains no degenerate critical points is then said to be a *Morse function*. Morse functions need to be C^2 continuous, considering that both first and second partial derivatives of F are required by the Morse theory. It is possible to relax this restriction and work with C^1 functions, provided that second derivatives are continuous at least over the critical points [5].

By taking the gradient ∇F, one obtains a vector flow field whose structure is intimately related to the topology of the implicit surface. From equation (4), the critical points of the surface are also the stagnation points of the flow field. A streamline of this field is a path that is obtained by following the local gradient vector, according to the ordinary differential equation:

$$\frac{d\mathbf{x}}{dt} = \nabla F(D(\mathbf{x}), \mathbf{x}). \tag{6}$$

A streamline is called a *separatrix* if it separates two regions of the flow with different characteristics [7]. Separatrices are important as they also give information about the topology of the surface. All the separatrices originate and terminate at maxima of F. For every separatrix there is always a 2-saddle somewhere along its path. The separatrix is locally tangent to the $\mathbf{v}_3$ eigenvector (associated with the λ_3 eigenvalue) at the 2-saddle.

Figure 3 shows a simple case of two implicit blobs connected as a single surface. There are two maxima close to the centroids of each blob and a 2-saddle at the junction of the two blobs. The separatrix, in this simple case, is a straight line segment joining the two maxima and passing through the 2-saddle. In a more general situation the separatrix would be curvilinear. Knowing the position $\mathbf{x}_S$ of the 2-saddle, it is possible to locate the two maxima sharing this critical point by integrating equation (6) backwards and forwards from $\mathbf{x}_S$, following a direction that is initially coincident with the $\mathbf{v}_3$ eigenvector of the 2-saddle. It is also possible to determine the connectivity of the two blobs by checking the sign of $F(D(\mathbf{x}_S), \mathbf{x}_S)$. If this sign is positive, the blobs are connected and the separatrix is known to travel exclusively through the interior of the surface. If the sign is negative, the two blobs are disconnected and the separatrix must exit the surface at some point.

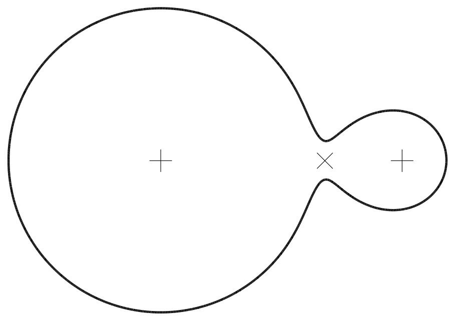

Figure 3. An implicit surface formed from two blobs. The "$+$" signs mark the two maxima and the "$\times$" sign marks the 2-saddle.

The separatrices defined by F form a network of lines that partition the $\mathbb{R}^3$ space into a *CW-complex* [4]. The CW-complex is a data structure that encodes all the topology of the implicit surface. It consists of a disjoint partitioning of the space into curved cells. The maxima are located at the corners of these cells and the separatrices form the edges of the same cells. Connectivity information can be obtained by following only the network of separatrices that are interior to the surface. This process will partition the maxima into a number of separate sets, which reflects the number of disconnected components of the surface.

5. The Topology Correction Method

The method for correcting the topology of hypertextured implicit surfaces proceeds by identifying all disconnected components of the surface. Of all the components detected, the larger one is considered to be the main surface, which is rendered as part of the ray casting algorithm. The remaining surface components are ignored during ray-surface intersection tests. The detection of disconnected surface components proceeds in two steps:

1. Build a set of all maxima and 2-saddles that are located inside the surface.

2. Segment the previous set into disjoint subsets by following the separatrices from the 2-saddles towards the maxima.

Steps 1 and 2 are performed before any surface rendering occurs. The outcome of step 2 is a sequence of sets S_i, with $i = 1, 2, \ldots, N$, where each set contains all the maxima that exist inside some particular component. The number N of sets is equal to the total number of surface components. One of these sets is the main set, corresponding to the main surface to be rendered. During a ray-surface

```
push bounding box $V_0$ onto stack;
while stack not empty
    pop voxel $V$ from stack;
    let $\mathbf{x}_V$ = centre of $V$;
    let $y = F(D(\mathbf{x}_V), \mathbf{x}_V)$;
    let $r$ = radius of bounding sphere for $V$;
    if $y < 0$ and $|y|/\lambda > r$
        continue;
    let $X_V$ = interval extent of $V$;
    if $\nabla F(D(\mathbf{X}_V), \mathbf{X}_V) \not\ni \mathbf{0}$
        continue;
    if $r < \epsilon$
        Test $V$;
        continue;
    subdivide $V$;
    push children onto stack;
```

Figure 4. The `Subdivision` algorithm.

intersection test, the set S_i that corresponds to the surface component to which the intersection point belongs is identified. If this is not the main set, the intersection point is ignored and another point is searched further along the ray. The following sections describe the relevant steps of the topology correction method.

5.1 Locating Critical Points

Location of critical points is made by recursive subdivision of an initial bounding box that surrounds the surface. For every cubical voxel resulting from this subdivision, a series of tests is made to determine, first, if the voxel contains part of the surface and, second, if a critical point may be contained within it. If these tests pass, the voxel is subdivided and the children are tested in turn, down to a minimum specified voxel size. Similar techniques to locate critical points by spatial subdivision have previously been used by Stander & Hart and Hart et al. for implicit surface meshing algorithms [21, 5].

To check if a voxel, centred at location $\mathbf{x}_V$, is part of any of the components of the surface, the following condition is first tested:

$$F(D(\mathbf{x}_V), \mathbf{x}_V) > 0. \tag{7}$$

If this condition is true, the voxel is either inside or straddling the surface. If, on the other hand, the condition evaluates to false, the voxel may still intersect with the surface. To check for this, we use a second test that is similar to the one performed by Kalra & Barr for their LG surfaces [9]:

$$|F(D(\mathbf{x}_V), \mathbf{x}_V)|/\lambda > r, \tag{8}$$

where λ is a Lipschitz bound of F and r is the radius of the smallest bounding sphere, centred at $\mathbf{x}_V$, for the voxel. Test (8) requires that F be Lipschitz continuous but this is a trivial consequence of the fact that F is already C^2 continuous. The Lipschitz bound is also necessary during the rendering stage since we use the sphere tracing method of Hart for that purpose [2]. Test (8) guarantees that the function does not change sign inside the voxel. If this test holds and if $F < 0$ at the point $\mathbf{x}_V$ we know that the voxel is entirely outside the surface and can be discarded.

A voxel is checked for the existence of critical points once it is known to be either inside or straddling the surface. This is achieved by evaluating the function gradient with interval arithmetic [14]. Let $\mathbf{X}_V$ be an interval vector that spans the spatial extent of the voxel. The test is:

$$\nabla F(D(\mathbf{X}_V), \mathbf{X}_V) \ni \mathbf{0}. \tag{9}$$

If the null vector $\mathbf{0}$ is contained inside the interval vector for ∇F, there is the possibility that one or more critical points may be contained in the voxel. The voxel is then either subdivided or an explicit test is made for the presence of maxima and 2-saddles, once a minimum voxel size has been reached. Figure 4 shows in pseudo-code the `Subdivision` algorithm that implements the sequence of tests for each voxel. The voxels are kept in a stack, which is initialised with the bounding box V_0 for the object.

Because of the conservative properties of interval arithmetic, it often happens that voxels neighbouring a voxel that contains critical points are also incorrectly flagged by condition (9) to contain such points. The `Test` routine, that is invoked in the listing of Figure 4, performs the final stage in the search for critical points, weeding out the false positives output by (9). We assume at this stage that a voxel is small enough to contain only one critical point. This should be true provided that the threshold ϵ for the minimum voxel size is appropriately chosen. Starting from the voxel centre $\mathbf{x}_V$, the following sequence of multi-dimensional Newton iterations is performed towards the position of the critical point:

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_i + \delta\mathbf{x}_i, \\ \mathcal{H}\{F\}\delta\mathbf{x}_i &= -\nabla F. \end{aligned} \tag{10}$$

Both the Hessian matrix $\mathcal{H}\{F\}$ and the gradient ∇F are evaluated at the point $\mathbf{x}_i$ to solve for $\delta\mathbf{x}_i$. The iteration is stopped if the sequence of points $\mathbf{x}_i$ goes outside the voxel. Otherwise, the sequence will converge to some point $\mathbf{x}_C$ inside the voxel where a critical point is known to exist. If the critical point is inside the surface such that $F(D(\mathbf{x}_C), \mathbf{x}_C) > 0$, and if it is a maximum or a 2-saddle (which is found after the eigenvalues of $\mathcal{H}\{F\}$ at $\mathbf{x}_C$ have been computed), the point is added to a set S of critical points interior to the surface. Each element in S stores the following information regarding a critical point:

```
for every $\mathbf{x}_C \in S$ by decreasing
        order of $F(D(\mathbf{x}_C), \mathbf{x}_C)$
    if $\mathbf{x}_C$ is a 2-saddle
        let $\mathbf{x}_i, \mathbf{x}_j$ be maxima reached from $\mathbf{x}_C$;
        let $S_i \ni \mathbf{x}_i$ and $S_j \ni \mathbf{x}_j$;
        if $S_i \neq S_j$
            create $S_k = S_i \cup S_j$;
            discard $S_i$ and $S_j$;
    else
        create $S_i = \{\mathbf{x}_C\}$;
```

Figure 5. The `Segmentation` **algorithm.**

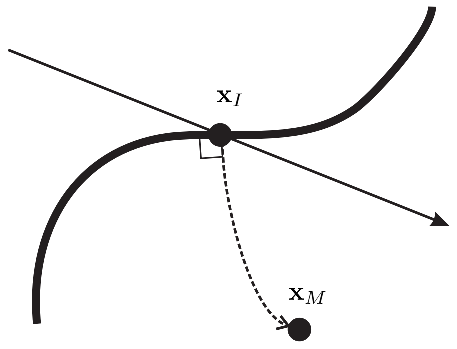

Figure 6. The intersection between a ray and the surface. The streamline originating at the intersection point is shown as a dotted line.

- The position $\mathbf{x}_C$.
- The value $F(D(\mathbf{x}_C), \mathbf{x}_C)$, which must be positive.
- A flag indicating if $\mathbf{x}_C$ is a maximum or a 2-saddle.
- The eigenvector $\mathbf{v}_3$ if $\mathbf{x}_C$ is a 2-saddle.

When the `Subdivision` algorithm completes, the set S will contain all the maxima and 2-saddles that were found inside every disconnected component of the surface.

5.2 Locating Disconnected Components

The set S is segmented into the sequence S_i, where each set S_i contains the maxima for one surface component. The algorithm `Segmentation` is shown in Figure 5. Each critical point $\mathbf{x}_C$ of S is considered at a time, by decreasing order of $F(D(\mathbf{x}_C), \mathbf{x}_C)$. If $\mathbf{x}_C$ is a maximum then a new set $S_i = \{\mathbf{x}_C\}$ is created. If, on the other hand, $\mathbf{x}_C$ is a 2-saddle, the two maxima $\mathbf{x}_i$ and $\mathbf{x}_j$ connected to it are determined by integrating the separatrix backwards and forwards with equation (6), starting from $\mathbf{x}_C$ and going initially along the direction of the $\mathbf{v}_3$ eigenvector for the 2-saddle. The sets S_i and S_j that contain $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, are then joined together to form a new set. The 2-saddle is ignored, however, if both $\mathbf{x}_i$ and $\mathbf{x}_j$ are found to be part of the same set.

Once `Segmentation` completes, all disconnected surface components will have been identified through the S_i sets. As a final step, the indices i are reassigned so that they run sequentially from $i = 1$ to $i = N$, where N is the number of disconnected components. The main surface is identified by the set S_m that contains the largest number of maxima, where the index m is:

$$m = \max_i \ \#S_i. \tag{11}$$

This criterion for selecting the main surface to be rendered may fail for objects with an excessively large amount of hypertexture. If there is too much hypertexture, the object will break into a cloud of many smaller objects of approximately equal size. It is not clear in these conditions which of these smaller objects should be selected for rendering. Our purpose is to study hypertextured functions $F(D(\mathbf{x}), \mathbf{x})$ where the geometry of the original object $D(\mathbf{x})$ is still discernible after the hypertexture has been applied. The criterion (11) will then identify the correct surface component for rendering since the majority of the maxima will be contained inside the main surface – only a smaller number of maxima will exist outside the main surface, being responsible for the disconnected fragments.

5.3 Computing Ray Intersections

The computation of ray intersections with the implicit surface is performed with the sphere tracing algorithm [2]. Once an intersection point $\mathbf{x}_I$ has been found along a ray, a test is performed to determine if it belongs to the main surface or not. To that effect, a streamline is followed with equation (6), starting from $\mathbf{x}_I$, which will converge towards some maximum $\mathbf{x}_M$ interior to the surface. Figure 6 shows an example. The streamline starts off along a direction that is initially orthogonal to the implicit surface and converges towards the point $\mathbf{x}_M$. Having found the maximum $\mathbf{x}_M$, the set S_i to which it belongs is retrieved. If this is the main set S_m, the intersection point $\mathbf{x}_I$ is rendered, otherwise sphere tracing continues along the ray to try to find another intersection point further along. A small increment is added to the length of the ray, up to the intersection point $\mathbf{x}_I$, to cause the sphere tracing algorithm to ignore that point and converge to the next available intersection point. Following every intersection that is found not to be part of the main surface, the connectivity test need not be performed again for the next intersection point, given that this will be the exit point of the ray from a disconnected component.

5.4 Tracking Streamlines

The path of a streamline needs to be tracked as part of the ray-surface intersection procedure of Section 5.3 and as part of the `Segmentation` algorithm of Section 5.2 where, in the latter case, the streamline is also a separatrix of the surface. Special care needs to be taken when performing this path tracking procedure because the endpoint of the streamline (and also the starting point, in the case of a separatrix) is a critical point where $\nabla F = 0$ occurs.

When tracking a separatrix, the path originates from a 2-saddle located at some point $\mathbf{x}_S$. If one were to integrate equation (6) with the initial condition $\mathbf{x}(0) = \mathbf{x}_S$, the path would never leave $\mathbf{x}_S$ since this is a stagnation point of the flow. To start off the integration from a 2-saddle, the following initial condition must be used instead:

$$\mathbf{x}(0) = \mathbf{x}_S \pm \epsilon \mathbf{v}_3, \tag{12}$$

where ϵ is a small displacement. The displacements of $\pm\epsilon$ along the $\mathbf{v}_3$ eigenvector will enable the integrator to move away from $\mathbf{x}_S$ and to converge towards the two maxima that connect with the 2-saddle through the separatrix. The maxima, however, are also stagnation points and path tracking would have to proceed from $t = 0$ up to $t = \pm\infty$ if the two maxima were to be reached exactly. In practice, one proceeds with the integration for as long as possible and then finds the maxima that are nearest to the points where the integrator left off.

We use the `lsode` ordinary differential equation solver from the ODEPACK Fortran package to perform path integration [8]. When given an upper limit of $+\infty$ or $-\infty$, `lsode` inevitably finishes with an error status as it tries to get close to one of the maxima. It also returns the farthest point $\mathbf{x}(t)$ that could be computed along the path. By controlling the numerical precision requested from `lsode`, it is possible for $\mathbf{x}(t)$ to be as close to the correct maximum point as desired. We then search among all the maxima of all the S_i sets for the one that is closest to $\mathbf{x}(t)$, thus identifying the particular set S_i to which the separatrix has converged. The procedure is similar when tracking streamlines as part of the ray-surface intersection tests except that we are now only interested in following the path from $t = 0$ to $t = +\infty$ and the starting condition $\mathbf{x}(0) = \mathbf{x}_I$ is used, instead of equation (12).

Currently, the search for the maximum point that is nearest to $\mathbf{x}(t)$ is performed exhaustively by computing the squared distance to every possible maximum. This search method has linear time complexity and can become slow for a surface with a large number of maxima inside. Although we have not implemented it for this paper, it is possible to perform the search for a maximum in average logarithmic time with the help of a kd-tree [1, 20].

6 Results

We demonstrate the application of the topology correction algorithm with hypertextures that are generated from scaled sums of a basis procedural noise function. The hypertexture function is:

$$F(D(\mathbf{x}), \mathbf{x}) = D(\mathbf{x}) + 0.8 \sum_{i=0}^{L-1} 2^{-0.8i} n(2^i \mathbf{x}). \tag{13}$$

The function D generates a sphere of unit radius, as in the example of Figure 1, and n is a sparse convolution noise function [11]. The summation in (13) models a fractional Brownian motion process with a Hurst parameter $H = 0.8$ [18]. The number of layers of noise that are added to the sphere is given by L. As this number increases, the surface of the sphere becomes increasingly more irregular and, in the limit, attains a fractal dimension of $3 - H = 2.2$.

Figure 7 shows the network of separatrices for a hypertextured object computed from equation (13), with $L = 1$, after the `Subdivision` and `Segmentation` algorithms have been applied. The network is shown superimposed over an image of the object. This network represents a partial visualisation of the CW-complex for the object's surface since only the separatrices that are inside the surface are shown. Maximum points are also shown as dots and are located at the endpoints of one or more separatrices. Several of these points, however, are isolated and correspond to small disconnected surface components that can be seen surrounding the main surface.

Table 1 lists the number of maxima, 2-saddles and disconnected components of the surface as the number of noise layers increases. These numbers follow a roughly geometrical progression with L, which causes the `Subdivision` algorithm to become increasingly less efficient as it needs to identify an ever denser cloud of critical points. The application of the topology correction method to a fractal hypertexture is, therefore, impractical since a surface needs to have five or more layers of noise to become recognisably fractal. Figure 8 shows the cases $L = 1$ and $L = 3$ of the hypertexture generated from equation (13). The original surface is first shown, without any topological correction. The disconnected components are then identified and visualised in red. Finally, the same disconnected components are ignored during the ray-surface intersection procedure.

A more efficient method than spatial subdivision for the localisation of critical points was proposed by Wu & de Gomensoro Malheiros for implicit surfaces that are made from sums of radial basis functions [22]. With this method, simple heuristics are used to estimate the position of the critical points. The application of several relaxation steps then causes the critical points to converge towards their correct positions. Sparse convolution noise is an example of

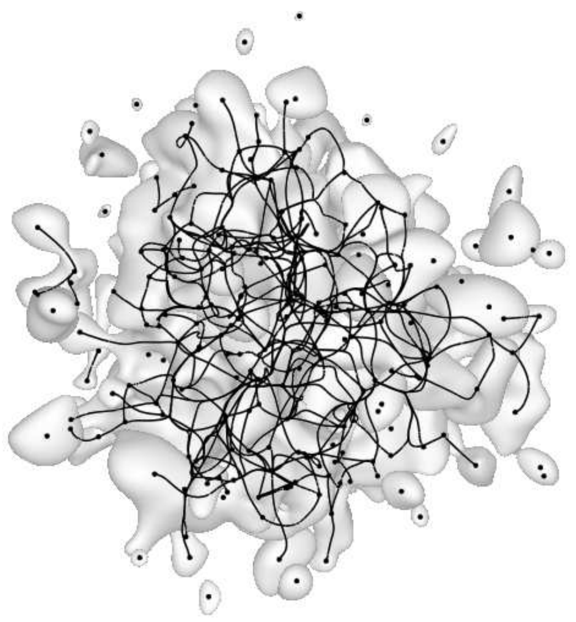

Figure 7. The network of separatrices and maxima interior to a hypertextured surface.

a hypertexturing function that could use the improved localisation method by Wu & de Gomensoro Malheiros since it consists of the sum of an infinite number of radial basis functions that follow a Poisson-disc distribution in space. The same method, however, cannot be applied to Perlin noise functions. For that reason, we have adopted spatial subdivision as our critical point localisation method, which, although being less efficient, is quite general and can be applied to any C^2 or even C^1 function. Spatial subdivision is also an easily parallelisable algorithm where disjoint regions of space can be assigned to different CPUs.

A minimum voxel size $\epsilon = 10^{-8}$ was used in the `Subdivision` algorithm to obtain the results shown in this section. The iterations (10) for the multi-dimensional Newton root finder were stopped when $\|\mathbf{x}_{i+1} - \mathbf{x}_i\| < 10^{-12}$. After determining the connectivity information, the component sets S_i, with $i = 1, \ldots N$, were stored to a file so that they could be reused for different renderings of the same surface. This is especially helpful when performing computer animation as the `Subdivision` and the `Segmentation` algorithms need to be run only once for each surface.

7 Conclusions

Morse theory provides all the connectivity information about an implicit surface that is necessary to determine how many components it is split into. This property of Morse theory finds application in the hypertexturing of implicit surfaces as it enables disconnected components other than the desired main surface to be detected and removed during rendering. In this way, one can add much greater amounts of hypertexture than previously possible to a solid object without the inconvenience of fracturing it into many smaller objects. Our technique can be applied to C^2 continuous hypertextured surfaces generated from equation (2) for the case $\alpha = 1$. In the most general situation, our technique can be applied to any C^2 continuous implicit surface whenever it may be desirable to identify and isolate disconnected components of the surface.

L	Maxima	2-saddles	Components
1	214	304	58
2	1006	1585	182
3	8408	4567	418

Table 1. Statistics for a hypertextured sphere with an increasing number of layers of noise.

The topological correction method is robust and will detect any disconnected component, no matter how small or how close it may be to the main surface. This robustness is again a consequence of the application of Morse theory. The accuracy of the method is only limited by the numerical tolerance factors and threshold values that are chosen for the algorithms described in the paper. We have used values that are equal to or smaller than 10^{-8}, giving the topology correction method an overall accuracy similar to that of single precision floating point arithmetic.

Although the proposed method can guarantee that a hypertextured implicit surface is topologically connected, it cannot guarantee that it is physically stable. Consider the case of a surface component that is attached to the main surface by a very thin bridge of material. If the rigidity of the material is not sufficient, the application of even the smallest force to the component will cause it to break at the junction point. This has consequences if one tries to use hypertextures to model terrain landscapes, for example, as some of the terrain features, although connected, may be unstable under the action of gravity. The modelling of hypertextured surfaces that are both connected and stable would require stress analysis tools and is beyond the scope of this paper.

8 Further Developments

The topology correction method that was here presented in the context of a ray casting rendering algorithm for implicit surfaces can, with little extra coding effort, be adapted to work in the context of the topologically correct polygonal meshing algorithm of Stander & Hart [21]. As a prelimin-

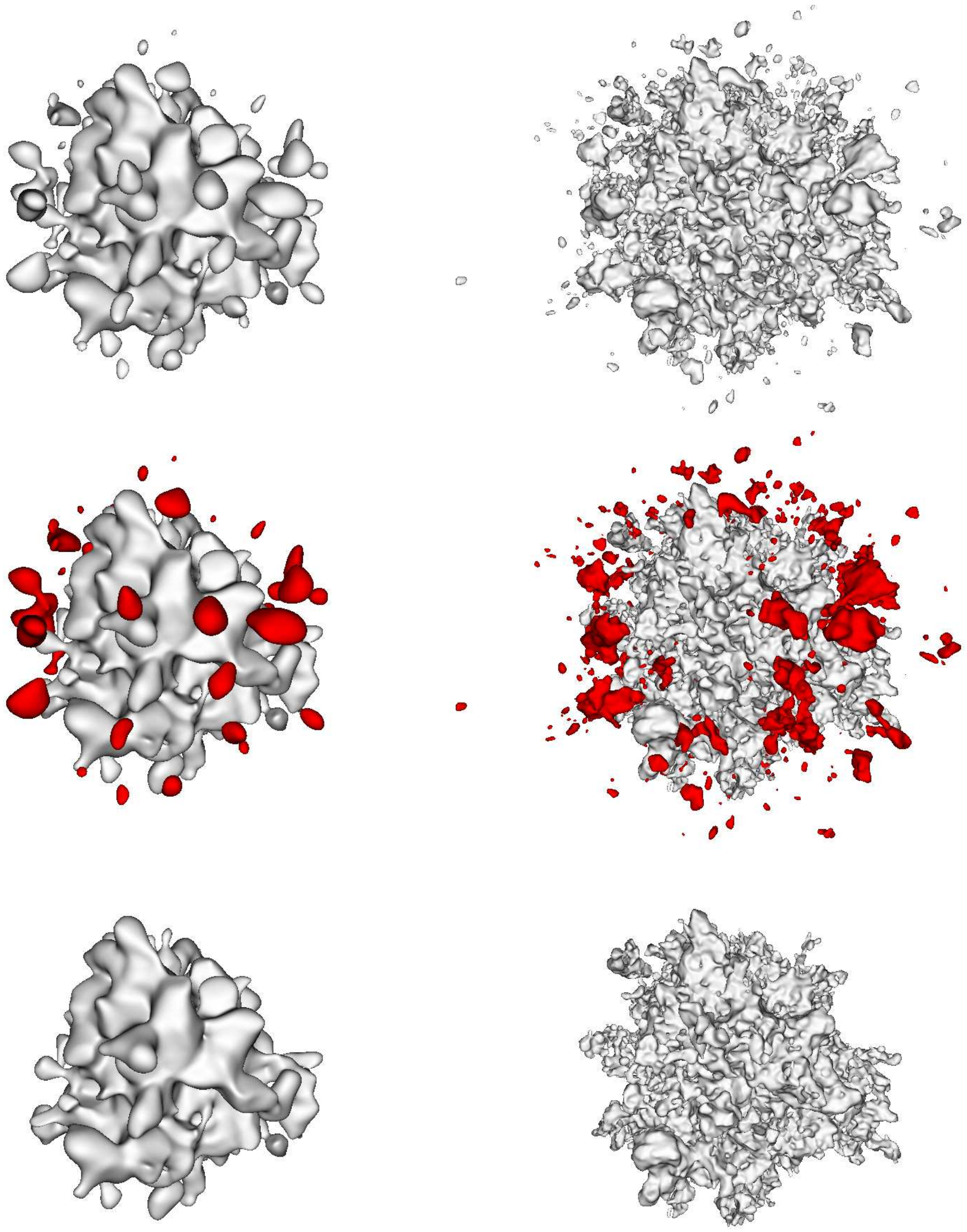

Figure 8. A hypertextured sphere with one layer (left) and three layers (right) of a sparse convolution noise function. Top row shows original surfaces. Middle row shows disconnected components in red. Bottom row shows surfaces after topological correction.

ary step of that algorithm, all the critical points of a surface are first located. The polygonal mesh that approximates the surface is then progressively inflated until it reaches its correct position. Whenever the mesh passes through one of the critical points, an appropriate mesh correction operation is performed to account for the topology change that has just occurred.

To perform topology correction for hypertextures as part of the method by Stander & Hart, it is necessary to include all critical points in the component sets S_i during the `Segmentation` algorithm, besides the maxima and 2-saddles that are already included. Just as before, all critical points are considered in decreasing order of their $F(D(\mathbf{x}), \mathbf{x})$ values. To include a 1-saddle, a streamline is followed towards one of the maxima. The streamline must be initially tangent to some arbitrary vector that is contained in the plane formed by the $\mathbf{v}_2$ and $\mathbf{v}_3$ eigenvectors of the 1-saddle. The maximum that is found at the end of the streamline then identifies the set S_i to which the 1-saddle belongs. To include a minimum, a streamline is followed which can be initially tangent to any desired direction around the minimum.

Once all the components sets have been identified, the main set S_m is chosen according to any preferred criterium and passed to the meshing algorithm. In this way, a polygonal mesh will only be computed for the main surface component. No effort will be wasted polygonising surface components that have already been found to be disconnected.

Acknowledgements

The first author acknowledges the support of the Fundação para a Ciência e a Tecnologia, Portugal, through grant SFRH/BD/16249/2004. The authors are also indebted to their anonymous reviewers for the helpful comments provided.

References

[1] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, Sept. 1977.

[2] J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(9):527–545, 1996.

[3] J. C. Hart. Morse theory for implicit surface modeling. In H.-C. Hege and K. Polthier, editors, *Mathematical Visualization*, pages 257–268. Springer Verlag, Heidelberg, 1998.

[4] J. C. Hart. Using the CW-complex to represent the topological structure of implicit surfaces and solids. In *Proc. Implicit Surfaces '99*, pages 107–112. Eurographics/SIGGRAPH, Sept. 1999.

[5] J. C. Hart, A. Durr, and D. Arsch. Critical points of polynomial metaballs. In *Proc. Implicit Surfaces '98*, pages 69–76. Eurographics/SIGGRAPH, June 1998.

[6] W. Heidrich and H.-P. Seidel. Ray-tracing procedural displacement shaders. In W. Davis, K. Booth, and A. Fournier, editors, *Proceedings of Graphics Interface '98*, pages 8–16. Canadian Information Processing Society, Morgan Kaufmann Publishers, June 18–20 1998.

[7] J. L. Helman and L. Hesselink. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3):36–46, May 1991.

[8] A. C. Hindmarsh. ODEPACK: A systematized collection of ODE solvers. In R. S. Stepleman, editor, *Scientific Computing*, pages 55–64. North-Holland, Amsterdam, 1983. Package available at www.netlib.org.

[9] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 297–306. ACM Press, July 1989.

[10] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[11] J.-P. Lewis. Algorithms for solid noise synthesis. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 263–270. ACM Press, July 1989.

[12] J. Milnor. *Morse Theory*, volume 51 of *Annals of mathematics studies*. Princeton University Press, Princeton, 1963.

[13] D. P. Mitchell. Robust ray intersection with interval arithmetic. In *Proceedings of Graphics Interface '90*, pages 68–74. Canadian Information Processing Society, Morgan Kaufmann Publishers, May 14–18 1990.

[14] R. Moore. *Interval Arithmetic*. Prentice-Hall, 1966.

[15] K. Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, pages 287–296. ACM Press, July 1985.

[16] K. Perlin. Improving noise. *ACM Transactions on Graphics (SIGGRAPH '02 Proceedings)*, 21(3):681–682, July 2002.

[17] K. Perlin and E. M. Hoffert. Hypertexture. In J. Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 253–262. ACM Press, July 1989.

[18] D. Saupe. Point evaluation of multi-variable random fractals. In H. Jüergens and D. Saupe, editors, *Visualisierung in Mathematik und Naturissenschaften - Bremer Computergraphik Tage*, pages 114–126. Springer-Verlag, 1989.

[19] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface coding based on morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, Sept. 1991.

[20] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[21] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In T. Whitted, editor, *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 279–286. ACM Press, Aug. 1997.

[22] S.-T. Wu and M. de Gomensoro Malheiros. On improving the search for critical points of implicit functions. In *Proc. Implicit Surfaces '99*, pages 73–80. Eurographics/SIGGRAPH, Sept. 1999.