The Visual Computer 24(6):397–409 (June 2008)

Manuel N. Gamito · **Steve C. Maddock**

Topological Correction of Hypertextured Implicit Surfaces for Ray Casting

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.

Keywords Morse Theory · Implicit Surface · Hypertexturing · Ray Casting

1 Introduction

Hypertexturing is a procedural technique proposed by Perlin & Hoffert to add three-dimensional small-scale detail to the surface of smooth objects [18]. 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 [16]. Using hypertextures, it becomes possible to actually deform the surface of an object instead of merely modifying its material shading properties. Hypertextures were initially presented as a technique to model *fuzzy objects* rather than implicit surfaces, i.e. objects such as a cloud that are represented by a density function and which do not have a well-defined boundary. Hypertextures, however, can also be applied to implicit surfaces.

In the original definition of hypertexture, an object is defined in three-dimensional space with an *object density function* $f_0 : \mathbb{R}^3 \to [0,1]$, which associates a density value $f_0(\mathbf{x})$ with every point $\mathbf{x}$ in space. A density of 0 means total transparency while a density of 1 means total opacity. The shape of the object can then be deformed through the composition of f_0 with one or more *density modulation functions* $f_i : [0,1] \times \mathbb{R}^3 \to [0,1]$, $i = 1,\dots n$ such that the final object density f is given by:

$$f(f_0(\mathbf{x}),\mathbf{x}) = f_n(\dots f_2(f_1(f_0(\mathbf{x}),\mathbf{x}),\mathbf{x}),\dots,\mathbf{x}). \quad (1)$$

It is possible to apply (1) similarly to implicit surfaces by considering $f_0 : \mathbb{R}^3 \to \mathbb{R}$ to be a function such that $f_0 = 0$ signals the surface and $f_0 > 0$ signals the inside of the surface. After this change, $f(f_0(\mathbf{x}),\mathbf{x})$ also becomes a function that generates a *hypertextured implicit surface*. The restriction that density modulation functions return opacity values in the range $[0,1]$ is no longer necessary and we can define them as functions $f_i : \mathbb{R} \times \mathbb{R}^3 \to \mathbb{R}$ for $i = 1,\dots,n$.

Compared to displacement mapping, hypertexturing is a more flexible technique for adding geometric detail to an otherwise smooth implicit surface [1,20]. Hypertextures can generate surface overhangs and arches, which the displacement mapping technique is incapable of producing. Figure 1 illustrates the difference between the two techniques. Although it is not demonstrated here, any displacement map can also be expressed as a particular form of hypertexture. If f_0 generates a sphere of unit radius, for example, any hypertexture written as $f(f_0(\mathbf{x}),\mathbf{x}/\|\mathbf{x}\|)$ is equivalent to a displacement mapped sphere. Furthermore, it is possible to add

The first author is supported by grant SFRH/BD/16249/2004 from the Fundação para a Ciência e a Tecnologia, Portugal.

Manuel N. Gamito
Department of Computer Science
The University of Sheffield
211 Portobello Street
Sheffield S1 4DP
E-mail: M.Gamito@dcs.shef.ac.uk

Steve C. Maddock
Department of Computer Science
The University of Sheffield
211 Portobello Street
Sheffield S1 4DP
E-mail: S.Maddock@dcs.shef.ac.uk

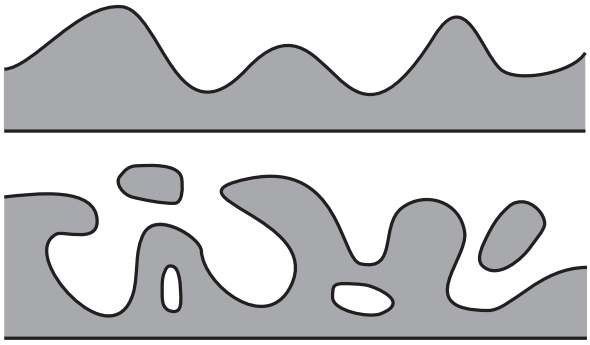

Fig. 1 Comparison between the displacement map (top) and hypertexturing (bottom) techniques when adding detail to a horizontal plane.

hypertexturing details on top of a displacement mapped surface, all within the framework expressed by equation (1).

One drawback of the modelling flexibility provided by hypertextures is that a surface can also become fragmented into several disconnected parts (this is illustrated in Figure 1). 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 disconnected state of the surface leads to physically non-plausible results.

This paper presents a solution to the surface splitting effect of hypertextures that are defined with C^2 continuous functions. Disconnected surface parts are detected in an initial connectivity analysis step and then removed during rendering. 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.

We make a distinction between *global methods* and *local methods* for querying surface connectivity. Global methods must analyse the entire surface as a pre-processing step before they can determine the connectivity state of any arbitrary surface point. Local methods, by contrast, can perform a localised connectivity analysis for every point. In this paper we concentrate on global methods. Our proposed method is global because it needs to locate all the critical points of the surface as a first step. We also give suggestions of how a local topology correction method may be implemented and show initial results in that direction.

Section 2 demonstrates the surface splitting effect and shows why it is hard to control in a general way. Section 3 presents two simple global methods that can be used 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. Section 8 suggests an extension of our algorithm that can be applied to implicit surface polygonisers. Finally, Appendix A presents formulae for evaluating the gradient and the Hessian of the procedural noise functions used in this paper. Evaluation of these functions is a common task but the same does not happen with their gradients or Hessians. These, however, are essential if one wants to apply Morse theory to noise functions.

2 The Surface Splitting Effect

We illustrate the splitting effect of hypertextured surfaces with an example. Figure 2 shows three implicit surfaces that have been generated by adding increasing amounts of hypertexture. The function that generates these surfaces is:

$$f(f_0(\mathbf{x}),\mathbf{x}) = f_0(\mathbf{x}) + \varepsilon\, n(4\mathbf{x}), \tag{2}$$

where $f_0(\mathbf{x}) = 1 - \|\mathbf{x}\|$ defines an implicit sphere of unit radius and n is Perlin's improved gradient noise function [17]. The amplitude ε of the hypertexture takes values of 0.1, 0.3 and 0.8 for the three surfaces in Figure 2. 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 [9]. 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.

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. It can occur for any hypertexture and not just the additive hypertexture given by function (2). The only exceptions consist of hypertextures that, by construction, are equivalent to displacement maps. The maximum amount of hypertexture depends on the particular function f that generates the surface and, without recourse to the Morse theory used in this paper, can only be found by trial and error. For solid modelling purposes, one would often like to use stronger hypertexturing effects than those allowed if a surface is to remain simply connected. It is not an uncommon practice, when excessive hypertexturing has been applied over a solid object, to digitally remove disconnected parts from the final rendering as a post-processing step. This simple trick can only be applied, however, provided that no disconnected component occludes the main surface.

3 Alternative Approaches

A simple but inaccurate way to perform topological correction on hypertextured surfaces is to employ a voxel grid, where the function $f(f_0(\mathbf{x}),\mathbf{x})$ is sampled at the corners of

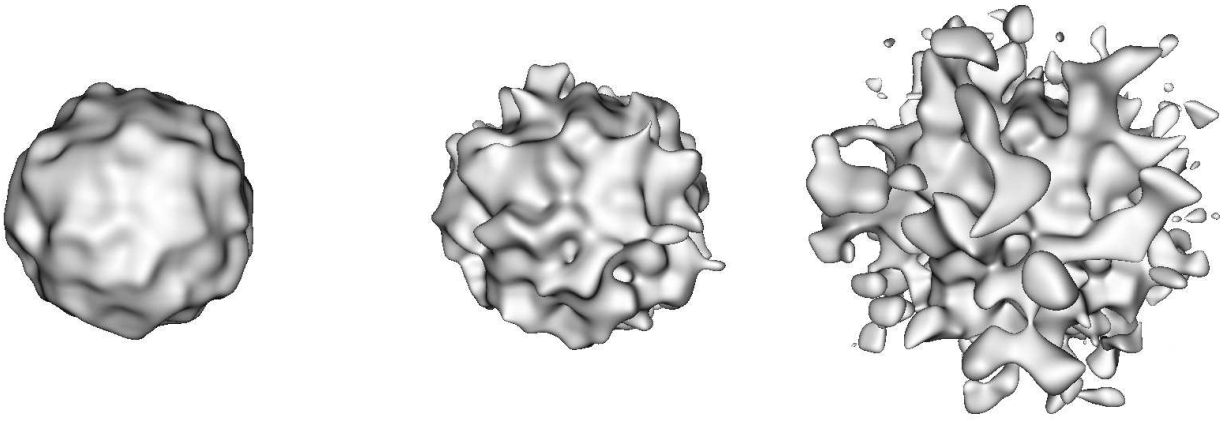

Fig. 2 A sphere rendered with increasing amounts of hypertexture ($\varepsilon = 0.1$, 0.3 and 0.8).

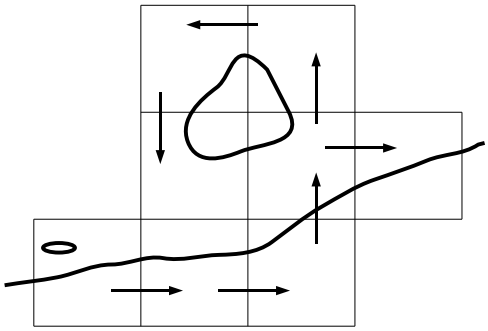

Fig. 3 Two surface components incorrectly determined to be part of the main surface. The arrows show the region growing sequence, starting from the voxel on the bottom left.

the voxels. A voxel is known to straddle the surface when 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 $f(f_0(\mathbf{x}),\mathbf{x})$ 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 3, 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 [24]. 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 over a manifold [13]. 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 [21, 6]. When the theory is applied to implicit surfaces, the manifold becomes the entire $\mathbb{R}^3$ 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:

$$\nabla f(f_0(\mathbf{x}_C),\mathbf{x}_C) = 0. \tag{3}$$

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

$$\mathscr{H}\{f\} = \left[\frac{\partial f^2}{\partial x_i \partial x_j}\right]_{i,j\in\{1,2,3\}}. \tag{4}$$

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

Table 1 Distinct types of critical points are determined by combinations of the signs of the eigenvalues of the Hessian matrix $\mathscr{H}\{f\}$.

If f is C^2 continuous then we have that $\partial f^2/\partial x_i \partial x_j = \partial f^2/\partial x_j \partial x_i$ and the Hessian is symmetric. The spectral theorem then guarantees that all three eigenvalues of $\mathscr{H}\{f\}$ 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 shown in Table 1. The type of a critical point gives an indication of the topology of the surface around that point. For example, the maxima occur near the local centroids of the surface while the 2-saddles occur at points where two surface components are joined together. In this paper, we only need to be concerned with the maxima and the 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. However, degenerate critical points 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 [8].

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 (3), 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:

$$\frac{d\mathbf{x}}{dt} = \nabla f(f_0(\mathbf{x}), \mathbf{x}). \tag{5}$$

A streamline is called a *separatrix* if it separates two regions of the flow with different characteristics [10]. 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 4 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 (5) 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 $f(f_0(\mathbf{x}_S), \mathbf{x}_S)$. 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 and enter the surface again at some points.

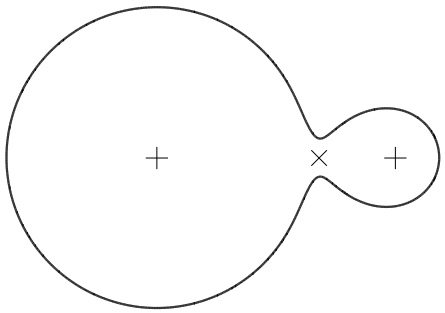

Fig. 4 An implicit surface formed from two blobs. The "$+$" signs mark the two maxima and the "$\times$" sign marks the 2-saddle.

The separatrices defined by f form a network of lines that partition the $\mathbb{R}^3$ space into a topological entity called the *CW-complex* [7]. 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.

```
push bounding box V_0 onto stack;
while stack not empty
  pop voxel V from stack;
  let X_V = interval extent of V;
  if f(f_0(X_V), X_V) < 0
    continue;
  if ∇f(f_0(X_V), X_V) ∌ 0
    continue;
  let r = radius of bounding sphere for V;
  if r < ε
    Test V;
    continue;
  subdivide V;
  push children onto stack;
```

Fig. 5 The `Subdivision` algorithm.

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 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. We employ the technique that was first proposed by Stander & Hart and later improved by Hart et al. [24,8]. For every cubical voxel resulting from the 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.

Interval arithmetic is used to check if a voxel is part of any of the components of the surface [14,22]. An interval estimate for the variation of F inside the voxel is used in the following test to determine if the voxel lies completely outside the surface:

$$f(f_0(\mathbf{X}_V),\mathbf{X}_V) < 0, \tag{6}$$

where $\mathbf{X}_V$ is an interval vector that spans the spatial extent of the voxel. Because interval arithmetic is a conservative range estimation technique, this test is always guaranteed to return a correct result for outside voxels.

A voxel is checked for the existence of critical points once it is known from test (6) that it may be either inside or straddling the surface. The test for the existence of critical points is achieved by obtaining an interval vector estimate of the function gradient, using again the $\mathbf{X}_V$ interval extent:

$$\nabla f(f_0(\mathbf{X}_V),\mathbf{X}_V) \ni \mathbf{0}. \tag{7}$$

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 5 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.

Due to the conservative properties of interval arithmetic, it often happens that voxels neighbouring a voxel that contains critical points are also incorrectly flagged by the interval arithmetic tests to contain such points. The `Test` routine, that is invoked in the listing of Figure 5, performs the final stage in the search for critical points, weeding out the false positives output by the interval tests. 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 Newton iterations is performed towards the critical point:

$$\begin{aligned} &\mathscr{H}\{f\}\delta\mathbf{x}_i = -\nabla f, \\ &\mathbf{x}_{i+1} = \mathbf{x}_i + \delta\mathbf{x}_i. \end{aligned} \tag{8}$$

Both the Hessian matrix $\mathscr{H}\{f\}$ 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 $f(f_0(\mathbf{x}_C),\mathbf{x}_C) > 0$, and if it is a maximum or a 2-saddle (which is found after the eigenvalues of $\mathscr{H}\{f\}$ 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:

- The position $\mathbf{x}_C$.
- The value $f(f_0(\mathbf{x}_C),\mathbf{x}_C)$, 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 6. Each critical point $\mathbf{x}_C$ of S is considered at a time, by decreasing order of $f(f_0(\mathbf{x}_C),\mathbf{x}_C)$. If $\mathbf{x}_C$ is a maximum then a new set $S_i = \{\mathbf{x}_C\}$

```
for every x_C ∈ S by decreasing
        order of f(f_0(x_C), x_C)
  if x_C is a 2-saddle
    let x_i, x_j be the maxima reached from x_C;
    let S_i ∋ x_i and S_j ∋ x_j;
    if S_i ≠ S_j
      create S_k = S_i ∪ S_j;
      discard S_i and S_j;
  else
    create S_i = {x_C};
```

Fig. 6 The `Segmentation` algorithm.

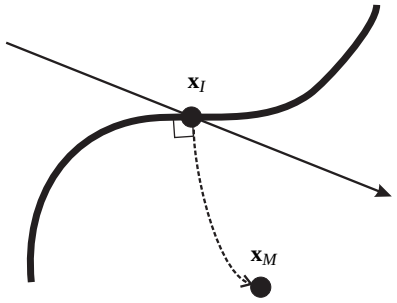

Fig. 7 The intersection between a ray and the surface. The streamline originating at the intersection point is shown as a dotted line.

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 (5), starting from $\mathbf{x}_C$ and going initially along the direction of the $\mathbf{v}_3$ eigenvector for the 2-saddle. Because the critical points are evaluated by decreasing order of f, it is certain that by the time a 2-saddle is considered, the two maxima $\mathbf{x}_i$ and $\mathbf{x}_j$ to which it connects will already have been processed by the algorithm. 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 already.

Once `Segmentation` completes, all disconnected surface components will have been identified through the S_i sets. The main surface is identified by the set S_m that contains the largest number of maxima, where the index m is:

$$m = \max_i \#S_i. \tag{9}$$

This criterion for selecting the main surface that is 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 $f(f_0(\mathbf{x}),\mathbf{x})$ where the geometry of the original object $D(\mathbf{x})$ is still discernible after the hypertexture has been applied. The criterion (9) 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 an affine arithmetic range estimation algorithm [4]. Affine arithmetic is an extension of the simpler interval arithmetic and provides tighter bounds for the estimation of unknown quantities [2]. This affine arithmetic intersection algorithm, like all interval based intersection algorithms, is capable of finding every intersection point between the ray and the surface, sorted by increasing distance along the ray. The sphere tracing method, by comparison, can only find the first intersection point with reliability [5]. 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 (5), starting from $\mathbf{x}_I$, which will converge towards some maximum $\mathbf{x}_M$ interior to the surface.

Figure 7 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 intersection testing continues along the ray to try to find another intersection point further along. 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 (5) 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:

$$\mathbf{x}(0) = \mathbf{x}_S \pm \varepsilon \mathbf{v}_3, \tag{10}$$

where ε is a small displacement. The displacements of $\pm\varepsilon$ 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 `lsodar` ordinary differential equation solver from the ODEPACK Fortran package to perform path integration [11]. The `lsodar` solver is able to select between a stiff and a non-stiff integration method, depending on the local conditions of the flow. When given an upper limit of $+\infty$ or $-\infty$, `lsodar` 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 `lsodar`, 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 (10).

Currently, the search for the maximum point 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 [3,23].

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:

$$f(f_0(\mathbf{x}),\mathbf{x}) = f_0(\mathbf{x}) + 0.8\sum_{i=0}^{L-1} 2^{-0.8i} n(2^{i+2}\mathbf{x}). \qquad (11)$$

The function f_0 generates a sphere of unit radius, as in the example of Figure 2, and n is a sparse convolution noise function [12]. The summation in (11) models a fractional Brownian motion process with a Hurst parameter given by $H = 0.8$ [19]. 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 8 shows the network of separatrices for a hypertextured object computed from equation (11), 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.

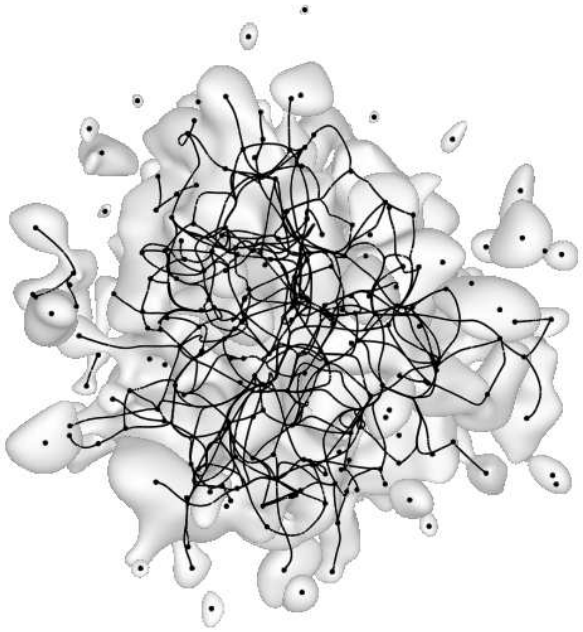

Fig. 8 The network of separatrices and maxima interior to a hypertextured surface.

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

Table 2 Statistics for a hypertextured sphere with an increasing number of layers of noise.

Table 2 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 9 shows the cases $L = 1$ and $L = 3$ of the hypertexture generated from equation (11). 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 for implicit surfaces that are made from sums of radial basis functions by Wu & de Gomensoro Malheiros [25]. With their method, simple heuristics are used to estimate the position of the crit-

ical 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 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 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 $\varepsilon = 10^{-8}$ was used as part of the `Subdivision` algorithm to obtain the results shown in Figure 9. The iterations (8) for the multi-dimensional Newton root finder were stopped when $\|\mathbf{x}_{i+1} - \mathbf{x}_i\| < 10^{-12}$. The numerical precision requested from the `lsodar` ODE solver was also equal to 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.

Figure 10 shows results that we have recently achieved and is a rendering of the surface of a procedural planet with overhangs and arches, represented as an implicit surface. Although the terrain appears to be defined over a flat surface, it is actually a sphere seen from a very close range. Procedural planet modelling is a powerful technique that can generate terrain details over the entire surface of a planet, with a range of scales similar to the one that exists on Earth [15]. A function similar to (11) was used that combines two procedural noise functions. A Perlin noise function provides the basic terrain pattern and is then modulated by a sparse convolution noise function to create the appearance of rocky outcrops over an otherwise smooth terrain. The evaluation of the gradient and Hessian of these two noise functions is presented in Appendix A. The detection of surface connectivity is shown in the middle image of Figure 10 with the disconnected surface components coloured in green. The removal of these components is then shown in the bottom image. It is possible to see that the shadows cast on the ground by disconnected components, which are visible in the lower left corner of the top and middle images, have disappeared in the bottom image due to those surface components having been removed. This effect is easily achieved by performing connectivity testing for shadow rays, similar to what is done for view rays. Disconnected components are ignored for shadow rays and a point is only in shadow if its shadow ray intersects with the main surface.

The results of Figure 10, when compared with the results of Figure 9, illustrate the difference between a global method and a local method for the determination of surface connectivity. The method presented in this paper is global because it must first locate all surface critical points as a first step. Clearly, this method, when applied to the surface of Figure 10, would be intractable, given the extreme range of scales that is present and the consequently large number of critical points that would have to be located. A Morse-based local method finds critical points on demand and only inside a small neighbourhood centred at the ray-surface intersection point for which a query is made about surface connectivity. The size of the neighbourhood is progressively enlarged, and more critical points are located, until a definite answer can be given about the connectivity state of the intersection point. Critical points can then be cached and reused for nearby ray intersection points on the surface. The local method for topological correction is more flexible but the global method is simpler to implement. Research efforts to finalise a local topology correction method are ongoing. Figure 10 was obtained with our current implementation of this local method.

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 (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.

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.

The method represents a global approach to topological correction where all the surface critical points must first be located in order for connectivity testing to be performed. This method is suitable for hypertextured surfaces where the ratio between the size of the hypertexture details and the size of the original surface is large. When this ratio is small, as is the case of procedural planets defined as hypertextures, a local topological correction method needs to be used instead.

Although the proposed method can guarantee that a hypertextured implicit surface is topologically connected, it cannot guarantee that it is physically stable. Consider the

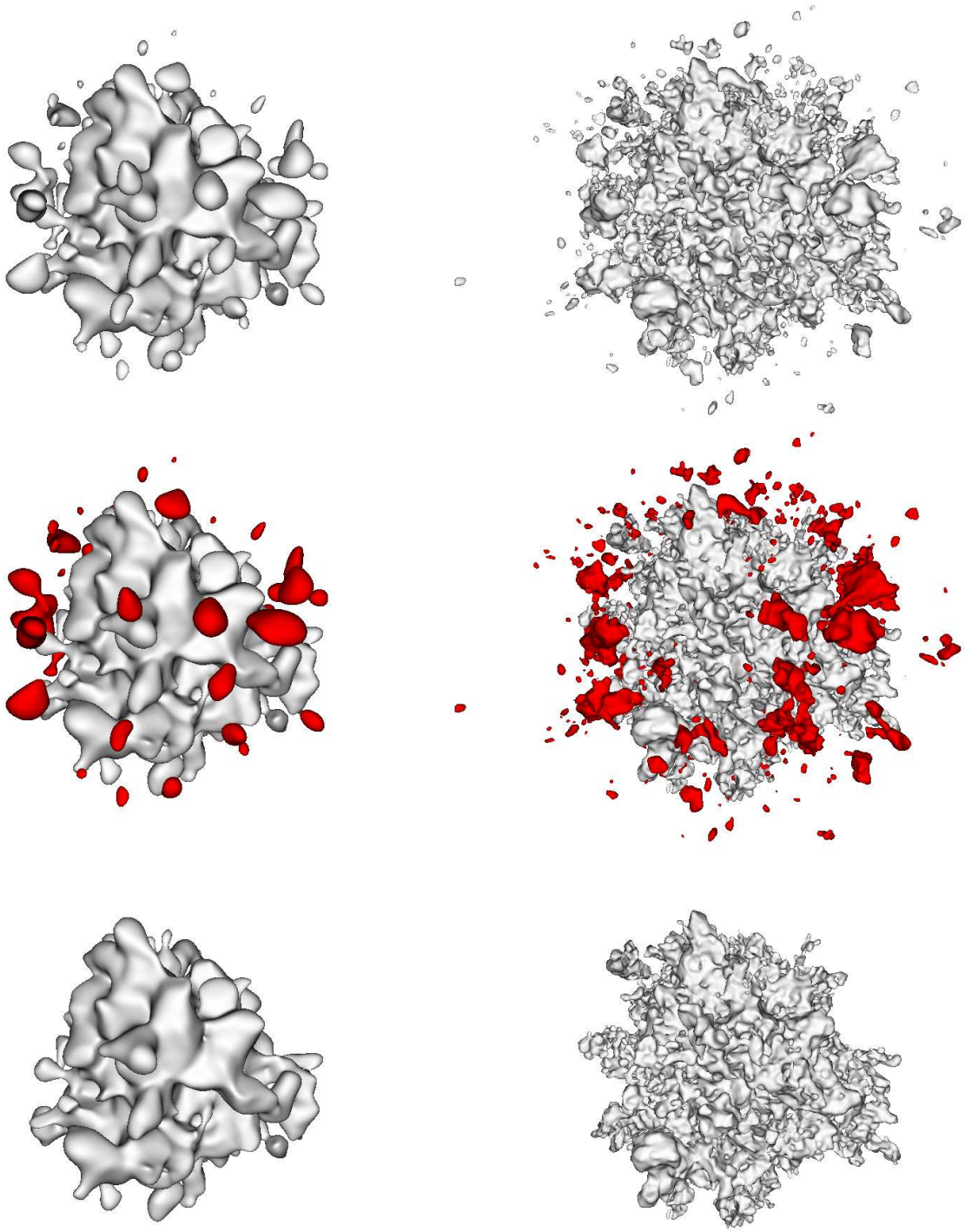

Fig. 9 A hypertextured sphere with one layer (left) and three layers (right) of a sparse convolution noise function. Top row shows original surfaces. Middle row shows disconnected components in red. Bottom row shows surfaces after topological correction.

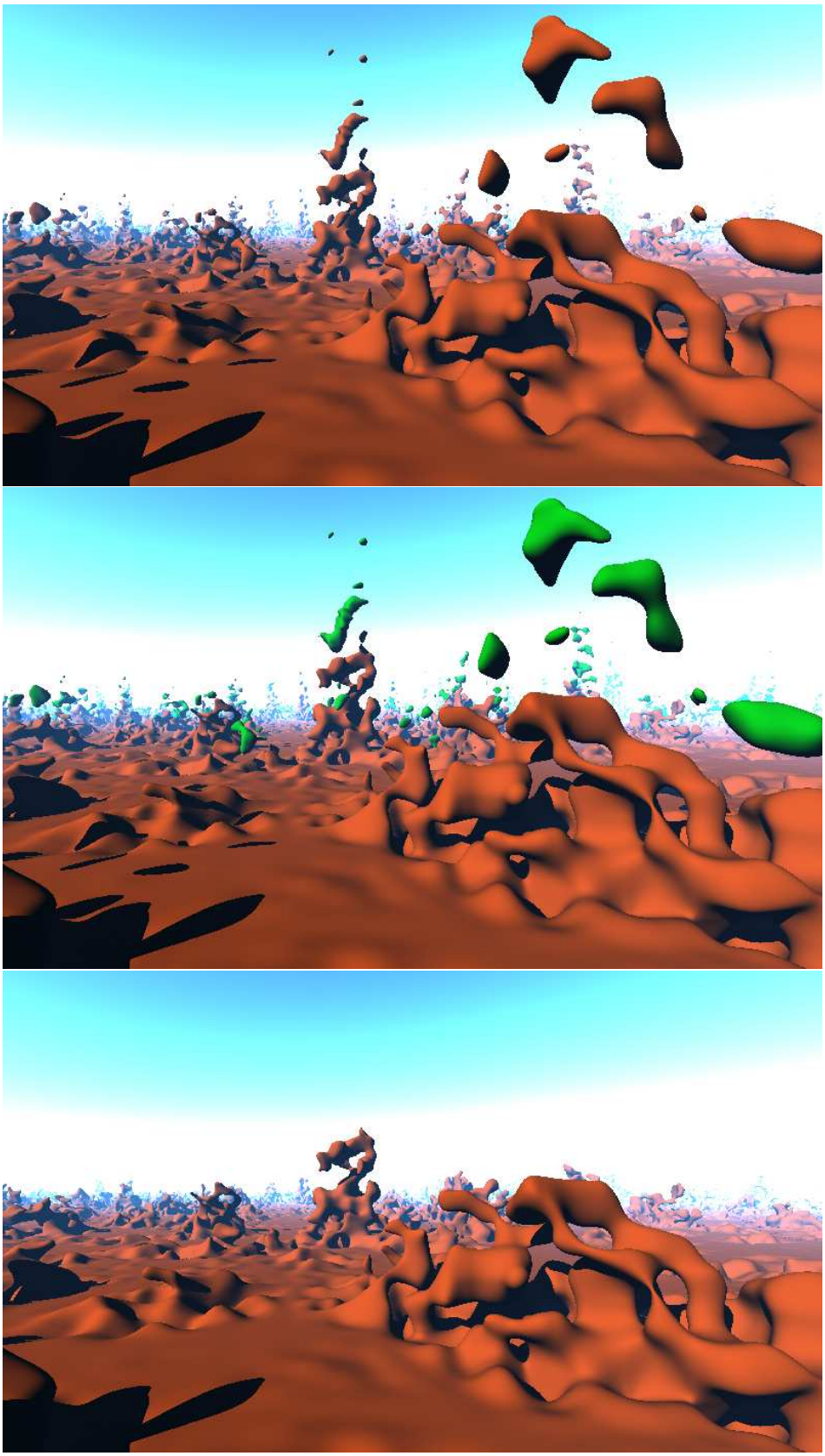

Fig. 10 A hypertextured planet featuring terrain overhangs and arches. The top image shows the original surface. The middle image shows disconnected components in green. The bottom image shows the terrain after topological correction.

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 topologically connected and physically stable would require stress analysis tools and goes 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 [24]. As a preliminary 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 in the context of the method by Stander & Hart, it is necessary to include all critical points in the component sets S_i as part of the `Segmentation` algorithm, together with the maxima and the 2-saddles that are already included by our approach. Just as before, all critical points are considered in decreasing order of their $f(f_0(\mathbf{x}),\mathbf{x})$ 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 started along any desired direction around the minimum.

Once all the component sets have been identified, the main set S_m is chosen, according to any preferred criterion, 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.

A Derivatives of Procedural Noise Functions

The application of Morse theory to hypertextured implicit surfaces made with procedural noise functions requires that formulae be available for the evaluation of the gradient vector and the Hessian matrix of such functions. The two widely available procedural noise functions that are known to be C^2 continuous are gradient noise and sparse convolution noise [17,12]. This appendix provides analytic formulae to evaluate their gradient and Hessian at any point in space.

The value of a procedural noise function n at some point $\mathbf{x}$ in $\mathbb{R}^3$ depends on the position of $\mathbf{x}$ relative to a discrete but infinite set $S = \{\mathbf{x}_i \in \mathbb{R}^3 : i = 0,1,2,\dots\}$ of node points $\mathbf{x}_i$ that are distributed throughout space. Because S has an infinite number of node points, the evaluation of $n(\mathbf{x})$ is feasible when $n(\mathbf{x})$ is made to depend only on a small subset $S(\mathbf{x})$ of S. At each location $\mathbf{x}$, the subset $S(\mathbf{x})$ is the finite set of node points in S that surround $\mathbf{x}$ according to some specified criterion.

We can define the value of a procedural noise function n at $\mathbf{x}$ as a sum of translated copies of a kernel function ϕ that depend on the displacement vectors between $\mathbf{x}$ and the node points in $S(\mathbf{x})$:

$$n(\mathbf{x}) = \sum_{i \in S(\mathbf{x})} \phi(\mathbf{x} - \mathbf{x}_i). \tag{A.1}$$

What distinguishes gradient noise and sparse convolution noise is the shape of the kernel ϕ, the criterion used to define $S(\mathbf{x})$ and the distribution of the $\mathbf{x}_i$ in space to form S. The gradient and Hessian are then given by:

$$\nabla n(\mathbf{x}) = \sum_{i \in S(\mathbf{x})} \nabla\phi(\mathbf{x} - \mathbf{x}_i), \tag{A.2a}$$

$$\mathscr{H}\{n\}(\mathbf{x}) = \sum_{i \in S(\mathbf{x})} \mathscr{H}\{\phi\}(\mathbf{x} - \mathbf{x}_i). \tag{A.2b}$$

A.1 Gradient Noise

For gradient noise, the set of node points forms a cubic integer lattice $S = \{(u,v,w) : u,v,w \in \mathbb{Z}\}$. For each location $\mathbf{x}$, the set $S(\mathbf{x})$ is made of the eight node points at the vertices of the lattice cell in which $\mathbf{x}$ resides. The kernel is given by:

$$\phi(\mathbf{x}) = \phi(x_1,x_2,x_3) = (\xi_1 x_1 + \xi_2 x_2 + \xi_3 x_3)\, h(x_1) h(x_2) h(x_3), \tag{A.3}$$

where ξ_1, ξ_2 and ξ_3 are random variables. The degree five polynomial h has support in $[-1,+1]$ and is such that $h(0) = 1$ and $h(-1) = h(+1) = 0$ [17]. For compactness, we write the polynomial for coordinate x_1 as $h_1 = h(x_1)$, the first derivative as $h'_1 = h'(x_1)$ and the second derivative as $h''_1 = h''(x_1)$ and similarly for x_2 and x_3. The gradient vector and Hessian matrix are given by:

$$\begin{aligned}\nabla\phi(x_1,x_2,x_3) \;&=\\ &= h_1 h_2 h_3 \begin{bmatrix}\xi_1\\ \xi_2\\ \xi_3\end{bmatrix} + (\xi_1 x_1 + \xi_2 x_2 + \xi_3 x_3)\begin{bmatrix}h'_1 h_2 h_3\\ h_1 h'_2 h_3\\ h_1 h_2 h'_3\end{bmatrix},\end{aligned} \tag{A.4a}$$

$$\begin{aligned}\mathscr{H}\{\phi\}(x_1,x_2,x_3) \;&=\\ &= \begin{bmatrix} 2\xi_1 h'_1 h_2 h_3 & (\xi_1 h_1 h'_2 + \xi_2 h'_1 h_2) h_3 & (\xi_1 h_1 h'_3 + \xi_3 h'_1 h_3) h_2\\ (\xi_2 h'_1 h_2 + \xi_1 h_1 h'_2) h_3 & 2\xi_2 h_1 h'_2 h_3 & (\xi_2 h_2 h'_3 + \xi_3 h'_2 h_3) h_1\\ (\xi_3 h'_1 h_3 + \xi_1 h_1 h'_3) h_2 & (\xi_3 h'_2 h_3 + \xi_2 h_2 h'_3) h_1 & 2\xi_3 h_1 h_2 h'_3 \end{bmatrix}\\ &\quad + \quad (\xi_1 x_1 + \xi_2 x_2 + \xi_3 x_3)\begin{bmatrix} h''_1 h_2 h_3 & h'_1 h'_2 h_3 & h'_1 h_2 h'_3\\ h'_1 h'_2 h_3 & h_1 h''_2 h_3 & h_1 h'_2 h'_3\\ h'_1 h_2 h'_3 & h_1 h'_2 h'_3 & h_1 h_2 h''_3 \end{bmatrix}.\end{aligned} \tag{A.4b}$$

A.2 Sparse Convolution Noise

As with gradient noise, a regular lattice placed at integer positions is used. Inside each cell in this lattice, K node points are uniformly distributed. This simple scheme generates an infinite Poisson distribution of node points. The value of n at each location $\mathbf{x}$ depends on the node points of the cell that contains $\mathbf{x}$ plus the node points in the twenty six surrounding cells. The set $S(\mathbf{x})$, therefore, always contains $27K$ node

points. The kernel ϕ depends only on the distance $r = \|\mathbf{x}\|$ and on a single random variable ξ:

$$\phi(\mathbf{x}) = \xi h(r). \tag{A.5}$$

For the function h, we have used the same degree five polynomial that was used for gradient noise, now evaluated only for positive arguments. Any other function can be used for h provided that it is C^2 continuous and with compact support in the interval $[0,1]$. The gradient vector and Hessian matrix are given by:

$$\nabla\phi(\mathbf{x}) = \xi\frac{h'(r)}{r}\mathbf{x}, \tag{A.6a}$$

$$\mathscr{H}\{\phi\}(\mathbf{x}) = \xi\Big(\frac{h''(r)}{r^2} - \frac{h'(r)}{r^3}\Big)\big(\mathbf{x}\cdot\mathbf{x}^T\big) + \xi\frac{h'(r)}{r}\mathbf{I}. \tag{A.6b}$$

The matrix $\mathbf{I}$ is a 3×3 identity matrix. The Hessian matrix is symmetric because the matrix $\mathbf{x}\cdot\mathbf{x}^T$ is also symmetric.

References

1. Cook, R.L.: Shade trees. In: H. Christiansen (ed.) Computer Graphics (SIGGRAPH '84 Proceedings), vol. 18, pp. 223–231. ACM Press (1984)
2. de Figueiredo, L.H., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms **37**(1–4), 147–158 (2004)
3. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software **3**(3), 209–226 (1977)
4. Gamito, M.N., Maddock, S.C.: Ray casting implicit fractal surfaces with reduced affine arithmetic. The Visual Computer **23**(3), 155–165 (2007)
5. Hart, J.C.: Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer **12**(9), 527–545 (1996)
6. Hart, J.C.: Morse theory for implicit surface modeling. In: H.C. Hege, K. Polthier (eds.) Mathematical Visualization, pp. 257–268. Springer Verlag, Heidelberg (1998)
7. Hart, J.C.: Using the CW-complex to represent the topological structure of implicit surfaces and solids. In: Proc. Implicit Surfaces '99, pp. 107–112. Eurographics/SIGGRAPH (1999)
8. Hart, J.C., Durr, A., Arsch, D.: Critical points of polynomial metaballs. In: Proc. Implicit Surfaces '98, pp. 69–76. Eurographics/SIGGRAPH (1998)
9. Heidrich, W., Seidel, H.P.: Ray-tracing procedural displacement shaders. In: W. Davis, K. Booth, A. Fournier (eds.) Proceedings of Graphics Interface '98, pp. 8–16. Canadian Information Processing Society, Morgan Kaufmann Publishers (1998)
10. Helman, J.L., Hesselink, L.: Visualizing vector field topology in fluid flows. IEEE Computer Graphics and Applications **11**(3), 36–46 (1991)
11. Hindmarsh, A.C.: ODEPACK: A systematized collection of ODE solvers. In: R.S. Stepleman (ed.) Scientific Computing, pp. 55–64. North-Holland, Amsterdam (1983). Package available at www.netlib.org
12. Lewis, J.P.: Algorithms for solid noise synthesis. In: J. Lane (ed.) Computer Graphics (SIGGRAPH '89 Proceedings), vol. 23, pp. 263–270. ACM Press (1989)
13. Milnor, J.: Morse Theory, *Annals of mathematics studies*, vol. 51. Princeton University Press, Princeton (1963)
14. Moore, R.: Interval Arithmetic. Prentice-Hall (1966)
15. Musgrave, F.K.: Mojoworld: Building procedural planets. In: D.S. Ebert, F.K. Musgrave (eds.) Texturing & Modeling: A Procedural Approach, 3rd edn., chap. 20, pp. 565–615. Morgan Kauffman Publishers Inc. (2003)
16. Perlin, K.: An image synthesizer. In: B.A. Barsky (ed.) Computer Graphics (SIGGRAPH '85 Proceedings), vol. 19, pp. 287–296. ACM Press (1985)
17. Perlin, K.: Improving noise. ACM Transactions on Graphics (SIGGRAPH '02 Proceedings) **21**(3), 681–682 (2002)
18. Perlin, K., Hoffert, E.M.: Hypertexture. In: J. Lane (ed.) Computer Graphics (SIGGRAPH '89 Proceedings), vol. 23, pp. 253–262. ACM Press (1989)
19. Saupe, D.: Point evaluation of multi-variable random fractals. In: H. Jüergens, D. Saupe (eds.) Visualisierung in Mathematik und Naturissenschaften - Bremer Computergraphik Tage, pp. 114–126. Springer-Verlag (1989)
20. Sclaroff, S., Pentland, A.: Generalized implicit functions for computer graphics. In: T.W. Sederberg (ed.) Computer Graphics (SIGGRAPH '91 Proceedings), vol. 25, pp. 247–250. ACM Press (1991)
21. Shinagawa, Y., Kunii, T.L., Kergosien, Y.L.: Surface coding based on morse theory. IEEE Computer Graphics and Applications **11**(5), 66–78 (1991)
22. Snyder, J.M.: Interval analysis for computer graphics. In: E.E. Catmull (ed.) Computer Graphics (SIGGRAPH '92 Proceedings), vol. 26, pp. 121–130. ACM Press (1992)
23. Sproull, R.F.: Refinements to nearest-neighbor searching in *k*-dimensional trees. Algorithmica **6**, 579–589 (1991)
24. Stander, B.T., Hart, J.C.: Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In: T. Whitted (ed.) Computer Graphics (SIGGRAPH '97 Proceedings), vol. 31, pp. 279–286. ACM Press (1997)
25. Wu, S.T., de Gomensoro Malheiros, M.: On improving the search for critical points of implicit functions. In: Proc. Implicit Surfaces '99, pp. 73–80. Eurographics/SIGGRAPH (1999)

Manuel N. Gamito is currently a PhD student at the University of Sheffield. He received a MSc in Electrotechnical Engineering from Lisbon Technical University in 1996. His research interests are in procedural modelling, landscape modelling and the visual simulation of natural phenomena. He is a member of ACM SIGGRAPH and Eurographics.

Steve C. Maddock is a senior lecturer in computer science at the University of Sheffield. His research interests are in computer facial modelling and animation, human figure animation, procedural modelling, and surface deformation techniques. He received a PhD in computer science from the University of Sheffield in 1999. He is a member of ACM SIGGRAPH and Eurographics.