Abstract
Level set models may be interactively edited. Due to numerical dissipation and movements in the surface’s normal directions level set modifications smooth out surface details. In order to overcome this problem, we have developed techniques that identify surface details prior to level set surface changes and incorporate them back afterwards; thus providing a detail-preserving level set editing capability. The surface details are stored in particles lying on the level set surface and are maintained on the surface as it is edited. The particles are then used to drive a detail-adding level set evolution process. These detail particles may also be transferred from one surface to another to provide a geometric texture transfer functionality for level set models.

Keywords: Detail-preserving modeling, geometric textures, level set surfaces, implicit modeling, particles

1. Introduction
While not widespread, volumetric implicit models are generated and utilized in a number of important computer graphics applications. These types of models are acquired with 3D scanners, e.g. MRI and CT, are generated during surface reconstruction methods, and are used for animation sequences and physical simulations. Level set models [1, 2] are one type of volumetric implicit model, and as a modeling representation they offer benefits over more popular explicit surface models. Level set models easily support CSG manipulations [3] and change of topology during editing and animation operations.
[4]. They do not require remeshing or reparameterization while undergoing drastic shape deformations. These properties make them well-suited for applications that utilize complex surfaces with dynamically changing topology, such as animating soft, fluid-like characters [5, 6], modeling fracturing or exploding objects [7], simulating water, clouds and smoke [8, 9], as well as representing surfaces of unknown genus acquired from scanned or reconstructed volumetric data [10, 11, 12]. They are also guaranteed to be closed and simple (non-self-intersecting), a property critical for 3D printing applications.

1.1. Motivation

As noted above, level set models have been used in many special effect sequences for physics-based simulations and for morphing animations of amorphous, shape-changing objects. These computed sequences often do not generate the desired results and may require post-processing and clean-ups. Furthermore, the intermediate steps of the simulation or morphing processes may also require control and redirection from the user in order to create specific outcomes. In order to meet the need for direct modification of level set models, a variety of computational technologies and techniques have been developed that provide numerous interactive editing capabilities for level set surface models. These include both CSG operations with smoothing and freeform surface manipulations [13, 14, 15]. The work described here extends the previously developed tools for level set editing by incorporating detail preservation into them.

Figure 1 demonstrates how interactive level set editing can be incorporated into the level set morphing process. An initial sequence creates an animation that morphs a double torus into an ellipsoid. The intermediate level set result shown in Figure 1(b) (within the red box) is then edited by the user to create an intermediate target shape (drawn within a blue box) that changes the shape of the object during the transition. Having level set editing capabilities allows the user to directly modify the intermediate level set model and removes the need for extracting a surface, importing it into a commercial surface modeling system, editing it, and re-scan-converting it back into a level set volume [3]. Surface extraction and scan-conversion are time-consuming processes that can introduce changes and errors into the model. With level set editing this expensive and troublesome component of the work-flow becomes unnecessary.
Figure 1: Using interactive surface editing to control and direct level set morphing.
1.2. Shortcomings and New Capabilities

The editing operators used in the previous example are part of a suite of operators [3, 14, 15] that implements a wide variety of expressive modeling functions for level set models. See Section 3.2 for more details. As with all level set evolutions, the editing operators move the surface with an algorithmically-generated speed field in directions normal to the surface. While the previous work on level set editing has created novel modeling capabilities for these types of implicit surfaces, they unfortunately have some shortcomings. Local motion in the normal direction can cause regions of high curvature to collapse and merge, resulting in the smoothing of the surface; thus removing small-scale details. It is also well-known that the computational methods used to evolve a level set surface smooth out the details of the interface because of numerical dissipation [2]. Therefore it is necessary to develop techniques for maintaining geometric details during level set editing operations. Methods have been developed to overcome/correct numerical errors through the use of higher order, hybrid and adaptive techniques. However, higher order methods [16] are computationally complex, which make them undesirable for interactive applications.

In order to address the current shortcomings of level set modeling, we have developed techniques that preserve surface details during editing and additionally may be used to transfer these details between level set models. Further technical details may be found in the first author’s PhD thesis [17]. These techniques are the main contributions of the following, described work. Inspired by the previous research on multiresolution modeling [18, 19] and particle level set methods [20], the techniques capture and store level set surface details in particle sets. These particles are used to add surface details back to the models after they have been removed by the inherent smoothing of level set evolution. The particle sets are dynamic and are kept on the surface after each operation. This allows us to use the particles during interactive surface modifications to maintain small-scale surface details. Additionally, the detail particles may be copied from one surface to another; thus supporting geometric texture transfer between level set models. Our work provides novel capabilities for directly modifying level set models. While these capabilities have been available for mesh models, we have extended them so they may be applied to volumetric implicit models within a level set framework. Our level set modeling techniques offer new benefits to users working with volumetric models.

Figure 2 shows results from our detail-preserving level set surface editing
Figure 2: (a,b) A level set model is modified with a detail-preserving editing operator. (c,d) Geometric texture transfer is used to place hair texture on a level set model of a lion after an editing operation.
and geometric texture transfer techniques. In subfigures (a) and (b) the shape of the armadillo’s face, represented with a level set model, has been expanded. Details on the face have been preserved during the editing operation. In subfigures (c) and (d) the capital has been removed from the top of the lion level set model, and geometric hair texture has been transferred from another part of the model to provide geometric detail where none previously existed.

2. Related Work

The separation of a surface into a lower resolution, smoothed base model and high-resolution details is central to multiresolution meshes. Zorin et al. [18] described a multiresolution representation for meshes based on subdivision. Their work combined smoothing, refinement and coarsification algorithms to construct a set of techniques for interactive multiresolution editing of complex hierarchical meshes. Another multiresolution mesh representation was presented by Kobbelt et al. [19]. Mesh editing is done at a coarse level and then mapped/translated to finer levels. Guskov et al. [21] and Lee et al. [22] both described multiresolution mesh representations (normal meshes and displaced subdivision surfaces, respectively) where each level of detail can be specified as normal offsets/displacements from a coarser version. Biermann et al. [23] utilized a similar base and detail separation to produce a cut-and-paste capability for multiresolution meshes. Sorkine et al. [24] presented an intrinsic surface representation based on the Laplacian of a polygonal mesh, which supports a number of detail-preserving editing operations, e.g. interactive free-form deformations, and the transfer and mixing of geometric details between two surfaces. Lipman et al. [25] introduced a rigid motion invariant mesh representation based on discrete forms defined on the mesh, and demonstrated its effectiveness with various detail-preserving editing operators and shape morphing. Pauly et al. [26] defined smoothing and decomposition operators for point clouds that transform the points into a multiscale surface representation, with the different scale levels being connected via scalar offsets. We have extended these concepts from multiresolution/multiscale mesh/point surfaces to level set models.

Enright et al. [20] proposed the particle level set method (PLSM) for improving the mass conservation properties of the level set method when the interface is passively advected in a flow field. They later presented the semi-Lagrangian based particle level set method for fast and accurate capturing of interfaces [27]. The particle level set method has been used during the
animation and simulation of smoke and complex water surfaces [8, 28, 29]. Unlike previous work our particles are kept on the level set surface during surface updates and are used to store and restore surface detail information.

Implicit models [30] are a widely used representation for geometric modeling applications. Wyvill et al. [31] created an implicit modeling system that combines constructive solid geometry (CSG) operations with blending and warping. They used a tree-based representation, called the BlobTree, where leaves of the tree are the primitives and inner nodes are the operations, i.e. warp, deform, blend, union, intersection and difference. ShapeShop [32], which uses BlobTrees as the underlying shape representation, extended this work with sketch-based editing techniques. Mullen et al. [33] proposed a mass-preserving variational approach for geometry processing of volumetric implicit surfaces and foliations using an Eulerian formulation. Level set methods have been used for volume sculpting [34], CSG-based surface editing, automatic blending and curvature-based smoothing [3, 13]. Lawrence and Funkhouser [35] proposed a painting paradigm for specifying surface deformations for level set surfaces and triangle meshes. Our detail-preserving techniques go beyond these methods in that they correct some of the shortcomings of previously-developed level set editing operators, namely the smoothing of surface details during surface evolution.

Geometric texture mapping is the 3-dimensional extension of traditional image-based texture mapping. Here, surface characteristics of a 3D model are skinned and automatically applied onto another model to create a variety of geometric details without the effort needed to manually specify them. Elber [36] and Zhou et al. [37] used a stitching technique to create more geometrically complex surfaces by tiling patterns over thin-shell triangle meshes. Lai et al. [38] presented an explicit texture transfer method for meshes based on geometry images. As mentioned above, Biermann et al. [23] developed a cut-and-paste capability for multiresolution meshes. Takayama et al. [39] presented GeoBrush, a method for interactive cloning of 3D surface geometry using a paintbrush interface. GeoBrush supports real-time continuous copying of arbitrary high-resolution surface features between irregular meshes, including topological handles. Lin et al. [40] added extracted surface details to tessellated implicit surfaces utilizing Laplacian coordinates [24]. Andersen et al. [41] extended the height field texture representation by incorporating displacements in the tangential plane in the form of a normal tilt.

Bhat et al. [42] presented a volumetric approach to tiling patterns in order to create more complex textures. Shell maps [43] provide a mapping
between shell space and texture space that can be used to generate small-scale features on surfaces using a variety of modeling techniques. The method is based upon the generation of an offset surface and the construction of a tetrahedral mesh that fills the space between the base surface and its offset. Brodersen et al. [44] extended geometry mapping techniques to level set models. They can warp and blend geometric models, represented either as a mesh or a level set surface, onto level set surfaces, providing a geometric stamping capability. Our techniques improve on previous work in that they allow a user to interactively specify source and target regions, which provides a more customized 3D texture transfer between level set models. This is a capability that has not been previously available for level set models.

While it is possible to implement detail-preserving editing and geometric texture transfer for an object defined as a level set model by converting the model into a mesh, then utilizing mesh-based techniques to perform the operation, and then re-scan-converting the modified mesh back into a level set model, it is our belief, and the belief of the implicit modeling professionals with whom we interact, that these conversion steps should be avoided, because they are slow, costly and each conversion step incorporates small errors into the model. The goal of our work is to develop modeling techniques that directly work on level set models, without the need for converting the underlying surface representation.

3. Level Set Models

3.1. Level Set Methods

The Level Set Method [1, 2] is a technique for tracking the evolution of a deforming interface/surface. It represents the deforming surface as an iso-surface

\[ S = \{ x \mid \phi(x) = k \}, \tag{1} \]

where \( k \in \mathbb{R} \) is the iso-value, \( x \in \mathbb{R}^3 \) is a point in 3-space on the iso-surface and \( \phi : \mathbb{R}^3 \to \mathbb{R} \) is a scalar function. In other words \( S \) is the set of points which forms the \( k^{th} \) iso-surface of \( \phi \). The choice of \( k \) is arbitrary. Deformations are defined by the change of \( S \) over time. The dynamic level set equation defines \( k \) to be constant (usually 0) and \( \phi \) to evolve with time,

\[ \phi(x, t) = k. \tag{2} \]
This equation can be transformed into a partial differential equation that can be solved by well-established numerical methods. Taking the time derivative of both sides of Equation 2 produces

\[
\frac{\partial \phi(x, t)}{\partial t} + \nabla \phi(x, t) \frac{ds}{dt} = 0.
\]  

Equation 3 is a Hamilton-Jacobi type equation and defines an initial value problem for the time-dependent scalar function \( \phi \). There are several numerical approaches for solving the dynamic level set equation \([1, 2]\). Equation 3 can be rewritten as

\[
\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot F(x, D\phi, D^2\phi\ldots),
\]  

where \( F \) is a user-defined speed term which depends on a set of order-n derivatives of \( \phi \), as well as other functions of \( x \). \( F \) defines the speed of the level set surface at point \( x \) in the direction of the local surface normal \((\nabla \phi/|\nabla \phi|)\). The surface is deformed over time by moving it either in or out in the direction of the normal. The magnitude of the movement is specified by the \( F \) function. An iterative process solves the level set PDE (Eq. 4) and updates the signed distance volume that represents the level set model.

### 3.2. Level Set Surface Editing

A set of free-form editing operators have been developed that provide direct implicit surface manipulations within a level set framework \([14, 15]\). These operators allow a user to add or remove surface detail from a level set model by interactively moving geometric handles attached to the surface. The level set surface can be edited through a click-sketch-and-pull interface that allows a user to modify the surface within a Region-Of-Influence (ROI). The ROI may be specified by drawing a closed boundary curve on the surface, with a superellipsoid, or with a distance function and defines what portion of the surface is to be edited. The user may then pull a point or a curve within the ROI to produce a free-form surface manipulation. Sketch-based interactions may also be used to specify multiple curves \([45]\) on, above and around a level set surface that define features, contours and constraints to which a level set model can move and fit. Since a level set model can only be modified via solving the level set equation, a speed function \( F \) has been devised for each editing operator. Figure 3 shows a level set model created using these editing operators.
A level set surface is defined as the zero crossings of a signed distance volume, where each voxel stores a floating point value that represents the signed shortest distance to the surface. The distance is positive or negative depending on the voxel’s location with respect to the surface, i.e. inside/outside of the closed manifold. Narrow-band techniques [12, 47, 48] provide means for performing the evolution of the level set equation only in a small band of voxels around the interface, thus reducing the amount of computation needed to deform a level set model. This work has been extended [49, 50] to not only limit computations to a narrow band around the interface, but to also limit the storage utilized during level set evolution. In our work we employ a sparse volume data structure based on spatial hashing [46] to store only the voxels within the narrow band. These sparse representations of narrow-band voxels allow for the creation and modification of high resolution level set models with modest memory requirements, while supporting the fast random data access/modifications needed for interactive applications.

4. Extracting Surface Details

Surface details are extracted from a level set model by a successive set of smoothing and differencing operations. In the first stage a high resolution volumetric model is smoothed and subtracted from the original model. Information about the difference between the two models is then stored in particles lying on the smoothed surface.

Curvature-based surface smoothing was first considered. However, since
Figure 4: (a) A noisy model is scan converted from a triangle mesh. (b) Model after 100 steps of curvature-based smoothing. (c) Model after a single application of the binomial filter.

The amount of smoothing produced by this method is directly proportional to the surface curvature, it is unable to generate uniform smoothing results for a wide variety of models. Additionally curvature-based smoothing requires numerous iterations to produce the desired result. Therefore direct volume filtering is used since it produces a superior, consistent smoothing result in less time. Youssef [51] examines several classes of filters, and finds that binomial filters produce the best results in terms of signal-to-noise ratio (SNR) between the original and the filtered-then-reconstructed images. Besides their excellent SNR performance, binomial filters offer the added advantage of having rational coefficients with powers of two denominators that improve computational efficiency.

Figure 4 shows a comparison of two surfaces obtained through curvature-based smoothing and binomial filtering applied to a level set model of an armchair that has been scan-converted from a noisy mesh model (Figure 4(a)). Noise is removed from the model with curvature-based smoothing. The result after 100 iterations of the level set evolution is shown in Figure 4(b). Figure 4(c) is created via a single application of the $3 \times 3 \times 3$ binomial filter. The amount of smoothing obtained via 100 steps of curvature-based smoothing is comparable to a single application of binomial filtering. 100 steps of level set smoothing needs 10.3 CPU-minutes and one application of binomial filtering only requires 9.79 CPU-seconds. The binomial filter clearly removes surface artifacts, and consequently geometric details, more efficiently than curvature-based smoothing.

A high resolution sampling of “detail” particles is then created that reside directly on the smoothed level set surface produced after binomial fil-
tering. The spatial hash table that holds the level set’s scalar values [46] was extended to also store the particle data structures. The particles are associated with the surface crossing voxels of the model. A voxel is called “surface crossing” if the distance value associated with the voxel has a sign (positive or negative) that is opposite to one or more of the distance values stored at the three voxels adjacent to it in the positive $x, y, z$ directions.

The process that generates detail particles is diagrammed in Figure 5. Particles are initially placed on the smoothed surface, one per surface voxel. The particles are projected in the direction of the local surface normal ($N$) onto the detailed surface. The signed distance field representing the detailed model facilitates finding the intersection point, as well as determining the inside/outside status of the particle with respect to the detailed model. An offset value (the distance between the smooth and detailed surface) is calculated and stored with $N$ in the particle. Additional particles are added to the detailed surface where necessary to produce a sampling of one particle per surface voxel. The offset direction for the new particles are calculated using a Gaussian weighted average of the directions of its neighboring particles on the detailed surface, with the weights defined by Equation 8. The particles in the 1-ring, 26-connected neighboring voxels are used in this calculation. Each new particle is projected back onto the smoothed surface using its $N$ direction, and stored in the surface’s spatial hash table along with the initial particles. An offset value is calculated for these new particles during the
back-projection step. Once completed this process will produce more than one particle per surface voxel in many regions of the smoothed surface. While there have been a number of techniques developed to adaptively sample implicit models [52, 53] we have found that our approach to generating detail particles has produced satisfactory sampling and reconstruction results in our examples.

Once these calculations are completed, each particle contains its 3D location on the smoothed surface, a scalar representing the signed Euclidean distance to the pre-filtered detailed surface, as well as the surface normal of the smooth surface at that particle’s location. These last two quantities may also be viewed as an offset value and a direction vector that capture the difference between the filtered and unfiltered level set surfaces. This representation of the surface details might seem inefficient considering one can combine the unit normal vector and the scalar into a single vector. However, we later show that this representation facilitates the construction of the speed function required to evolve the level set surface when adding geometric details back into the model.

5. Detail-Preserving Surface Editing

As an editing operation modifies the detailed model and smooths out small-scale features, its detail particles are maintained on the altered surface. Once the editing operation is completed, an iterative level set evolution is performed which restores the removed details. The evolution’s speed function is based on the offset values and direction vectors stored at the particles.

Figure 6 presents a flowchart of the detail-creation and model-reconstruction processes, using a portion of the armadillo model. The original model is filtered to create a smooth surface. The smoothed model is “subtracted” from the detailed model to produce the surface details, as defined in the previous section. The signs of the detail particles’ offset values are shown on the lower left, green for positive and red for negative. Details may be added back to a smoothed model via a level set evolution to produce a surface with small-scale features.

5.1. Maintaining Detail Particles on the Surface

Two techniques were developed to maintain the detail particles on a level set surface as it is being modified, an iterative method that updates the particles at each step of the level set evolution, and a projection method
that places the particles onto the reshaped surface after the completion of a single editing operation. A comparison of run times for the methods is presented in Section 8.

5.1.1. The Iterative Method

Using this method, detail particles are kept on the surface at all times while modifications are being made to the model. At every step of a level set evolution during surface editing, the new locations of the detail particles can be easily determined using the signed-distance value of the level set field and the offset vectors associated with the particles. Each particle is moved along its offset vector until a zero value is found in the field. The 3D location of the zero value defines an intersection between the vector and the level set surface. The particle is then moved to this intersection point on the surface. The CFL condition that restricts the surface to move less than one voxel every time step [2] ensures that the particles will always reside within the narrow band of the surface after each iteration of the level set evolution.
5.1.2. The Projection Method

Even though the method of Section 5.1.1 can effectively keep track of the detail particles as the surface moves, it requires that all of the particles within the ROI be moved with the surface at each incremental update of the level set field. This process is slow at higher resolutions that involve a large number of particles, and hinders interactivity. In order to improve interactive performance, a non-incremental method was developed that projects the detail particles onto the surface after it has been edited. In order to keep the local relationships of particles intact, we introduce a spring system that connects each particle to all of the particles in its 1-ring, 26-connected neighboring voxels. The rest length of each spring is set as the initial Euclidean distance between particles, placing the system in a stable steady state. Once the connections are made, any movement of one or more of these particles from the steady state configuration triggers a response from their neighbors which cascades through the entire set of particles until the system once again reaches equilibrium.

Two concepts presented in Section 3.2 are pertinent for the projection method. The first concept is the editing boundary which defines the extent of a local surface modification (i.e. the ROI). This boundary curve provides the first constraint in the spring system. The particles located within a voxel intersected by a boundary curve are called boundary particles. These particles are considered frozen and are immobile during the entire process that takes the spring system to a steady state. The second concept is the use of 3D points and curves placed on the surface as handles in order to enable user interaction. Particles located within a voxel intersected by a handle curve or point are called handle particles. Handle points and curves move with the surface at each step of interactive editing. The final position of the handles are used to move the handle particles onto the edited surface. These particles also stay fixed at their new location during the following step, which repositions the remaining particles as they move to a new equilibrium with the spring system.

Particles that are not positioned either on the ROI boundary or on the handles move under the influence of two “forces” during the projection step, the force created by the springs connected to them and the constraint keeping them on the level set surface. The first force is calculated from the potential
energy of a spring. For each particle \( i \) located at \( x \), the energy \( E_i \) is

\[
E_i(x) = \sum_{s \in S_i} k \cdot (d_s - d_s^0)^2, \tag{5}
\]

where \( S_i \) is the set of springs that are attached to particle \( i \). \( k = 1 \) for all springs, \( d_s \) is the spring’s current length and \( d_s^0 \) is the initial rest length of the spring. The particles move in the direction of the spring energy gradient until they are evenly spread between the boundary and handle particles, i.e. the system reaches a steady state. These particles should also remain on the level set surface. Therefore, following several spring-relaxation steps the particles are projected onto the surface in the direction of the surface’s distance field gradient.

These two processes are repeated, 1 projection step followed by 20 steps of spring energy minimization, until a state is reached where all particles are in close proximity to the surface and the spring energy gradient is close to zero. The first condition is satisfied when the maximum closest distance to the surface for all particles is under 0.1 voxels and the second condition is satisfied when the maximum magnitude of the gradient of Equation 5 for all particles is under 0.1. Once the stopping criteria are satisfied, a final projection step places all particles on the surface.

### 5.2. Detail Restoration Speed Function

The speed function that restores small-scale features utilizes the offset values stored at the detail particles. Details are only added to the modified portion of the surface and only the detail particles that cover the modified area contribute to the speed function. The detail-adding speed function is defined as:

\[
F(x) = D(x) \sum_{p \in P(x)} \omega_p \frac{|O_p|}{\max(|O_{ROI}|)} \tag{6}
\]

\[
D(x) = \begin{cases} 
-1 & \frac{\sum_{p \in P(x)} \omega_p \vec{V}_p}{\sum_{p \in P(x)} \omega_p \vec{V}_p} \cdot \vec{n}(x) < 0, \\
+1 & \text{otherwise}
\end{cases} \tag{7}
\]

\[
\omega_p = e^{-\frac{(G(x_p, x))^2}{2\sigma_p^2}} \tag{8}
\]
where \( x \) is a point on the surface, \( P(x) \) is the set of 8 nearest particles to \( x \), \( O_p \) and \( \vec{V}_p \) are the scalar offset \((O)\) and the unit vector \((\vec{V})\) associated with particle \( p \), and \( \text{max}(|O_{ROI}|) \) is the maximum of the \(|O|\) values stored at all of the particles within the ROI. We sum the influence of the 8 nearest particles assuming that these would be the 8-connected neighbors on a regular grid on the surface.

\( D(x) \) in Equation 7 determines the direction (-1 for inwards and +1 for outwards) of the movement along the surface normal \( \vec{n}(x) \) using the dot product of the normal and the weighted sum of the \( V \)s stored at all contributing particles. The sign of the dot product indicates if the weighted sum of the \( V \) vectors is in the direction of the local normal or not. The weight \( \omega_p \) for each particle in \( P(x) \) is calculated using Equation 8, based on the geodesic distance between \( x \) and the 3D location of each particle \((x_p)\) in set \( P(x) \).

\( G(x0, x1) \) is a function that returns the geodesic distance between two 3D points on the surface \((x0 \text{ and } x1)\). The Gaussian function provides a smooth blending of all neighboring offsets. \( \sigma_p \) is set to one half of the geodesic distance to the farthest particle in set \( P(x) \). This weight function has a positive but rapidly reducing value as the geodesic distance increases and allows the closest particles to contribute more to the speed function at point \( x \).

Equation 6 uses a weighted sum of the scalar offset values from a small neighborhood of particles to determine how much the level set field stored at each surface crossing voxel will change. The weighted sum is divided by the sum of \( \omega_p \times \text{max}(|O_p|) \) to ensure that the magnitude of \( F() \) is less than 1, in order to satisfy the CFL condition during the level set value updates [2]. While the \( D() \) term is discontinuous, the blending nature of the remaining portion of Equation 6 leads the speed function \( F() \) to transition smoothly from inward motion to outward motion, and vice versa. Overall, the \( F() \) function computes a floating point value between \(-1.0\) and \(1.0\) that, once plugged into Equation 4 and solved, implicitly moves the level set surface either in the direction of the surface normal at \( x \), or in the opposite direction.

Figure 7 presents a 2D illustration of detail preservation using the projection method. Subfigure (a) shows a level set surface with its detail particles. In (b) a handle point \( P_s \) is selected within an ROI bounded by particles \( B_s \). Pulling \( P_s \) changes and smooths the surface within the ROI, as seen in (c). The springs connecting \( P_s \) to its neighboring particles are maintained and the \( B_s \) particles are immobile. In (d) the spring forces, along with an explicit projection step, move the detail particles onto the new surface. In the last
step (e) the details are restored to the surface via a level set evolution using information from the particles.

5.3. Sampling

While constrained to lie on the surface, the particles might clump together or stray away from each other as the surface shrinks or stretches. In order to address the latter, we keep a minimum sampling of one particle per surface crossing voxel by adding new particles as the surface stretches. A 3D point on the level set surface within an empty surface crossing voxel can be computed from the signed distance values at the 8 corners of the voxel using tri-linear interpolation. A new particle is placed at this point and its offset direction and magnitude is calculated by interpolating the offsets of the 6 nearest particles to the new location using the Gaussian weights of Equation 8.

Some particles may clump together in a single surface crossing voxel if the surface is shrinking. Removing or blending some of these clumping particles would result in a loss of information. We do not do this. Since the hash table can store multiple particles per \((X, Y, Z)\) index, all detail particles associated with a single voxel are maintained during surface editing.

5.4. Adjusting Detail Restoration

All particles keep track of how far they have moved during the detail-addition process using a path variable. The path is set to zero for all particles initially and it is updated each time the interface and the particles move.
The particle positions are updated by finding the intersection of the level set surface with one of the two vectors \((x_p + \vec{V}_p)\) or \((x_p - \vec{V}_p)\), where \(x_p\) is the current position of the particle and \(\vec{V}_p\) is the particle’s offset direction. The particle is then moved to the intersection point. Since the speed at a given point on the surface is based on information from a number of particles, it is possible that any point on the surface may move opposite to the offset direction of the nearest detail particle. The bidirectional computation of the intersection point ensures that the particles stay on the surface in such cases. The particle’s displacement is subtracted from the path variable if it moves in \(-\vec{V}_p\) direction and added otherwise.

The path variable is used to determine when the detail-addition process should stop. The level set surface is moved with the speed function defined in Equation 6 until all particles have moved to within 0.1 voxels of the offset value \((O)\) associated with each particle, i.e. when \(O - \text{path} < 0.1\). Particles reaching these limits no longer contribute to the speed function, thus slowing down and/or stopping the surface within their influence. If all particles no longer contribute to the speed function the interface comes to a complete stop and no further details are added.

This approach restores details that are derived from the binomial filtering and subtraction process described in Section 4. Since the smoothing produced by the level set edit could be significantly different than one produced by filtering, it may be necessary to adjust the number of level set evolution steps in order to restore the correct amount of surface detail. In this case the evolution may be rerun with fewer steps when the automatic process boosts the added details, or additional evolution steps may be performed if the initial detail addition proved insufficient.

We have observed that it is possible for particles to oscillate in close proximity to the surface due to the smoothing nature of the level set evolution. A small amount of added detail may be subsequently smoothed during an evolution. This happens rarely at very sharp corners, because of sampling issues that produce steep gradients around these features. An integer counter is used to keep track of the number of times a particle consecutively crosses the surface. Each time a particle changes direction the counter is increased, otherwise the counter is set to 0. A particle is considered “oscillating” when the counter reaches 5. Oscillating particles are labeled to be at their destination when evaluating the stopping criteria for the level set evolution.
6. Geometric Texture Transfer

Geometric texture mapping is the 3-dimensional extension of traditional texture mapping, which defines the surface color of an object with a 2D image. With geometric texture mapping, surface characteristics of one 3D model are extracted and applied onto another model to create a variety of geometric details. The techniques developed for preserving surface detail can also be used to transfer geometric detail from one part of a level set surface to another or between two different level set surfaces. Surface details are extracted from an ROI on a source level set surface using the smoothing and differencing method described in Section 4. The particles are then transferred to a new location near a destination level set surface. A spring system is used to place the particles onto the destination surface. A surface evolution using the speed function in Equation 6 is then computed to add these details onto the new surface.

6.1. Initial Positioning of Detail Particles

The user specifies the details to be transferred by clicking a point on the source surface \(x_s\) and choosing a source radius \(R_1\). A flood-fill algorithm is used to identify all detail particles within a geodesic distance smaller than \(R_1\) to \(x_s\). The surface within this radial distance becomes the source region of influence (ROI). Similarly the destination ROI is defined by specifying a point \(x_d\) and a radius \(R_2\). The particles within the source ROI are connected to neighboring particles via springs, as described in Section 5.1.2. Particles lying in voxels that are distance \(R_1\) from \(x_s\) on the source surface and distance \(R_2\) from \(x_d\) on the destination surface are designated as boundary particles.

Initially all source detail particles (except the boundary particles) are moved close to the destination ROI by using the translation vector \(T = x_d - x_s\). Taking the cross product of the surface normals at \(x_s\) and \(x_d\) produces an axis of rotation \(\vec{V}_r\) and the dot product of these normals produces the angle between them \(\theta\). All particles are rotated by \(\theta\) around \(\vec{V}_r\). Additionally the user may specify a rotation around the surface normal to change the orientation of the details. Note that the aim here is not to move the particles exactly onto the destination surface, but to get them close to the destination ROI in order to prepare them for the next stage. After these transformations, the particle \(P_s\) at \(x_s\) has been moved to \(x_d\) (then labelled \(P_d\)). Boundary particles are not positioned with the method described above, but are moved to the destination surface using two different algorithms described in the
following section. $P_d$ and the boundary particles stay fixed in the destination ROI for the remainder of the next, projection stage.

6.2. Projecting Detail Particles

Two methods are available for projecting the source detail particles into the destination ROI. The first method does a direct one-to-one mapping, distributing one copy of all of the source detail particles into the destination ROI. The source and destination ROIs are not necessarily the same size, i.e. $R_1 \neq R_2$. When $R_2$ is less than $R_1$ the details are contracted, and multiple particles are stored per voxel by the spatial hash table of the level set data structure. When $R_2$ is greater than $R_1$, the detail particles are stretched over the destination ROI. The second method may be used only when the destination ROI is larger than the source ROI, and it tiles multiple copies of the source details into the destination ROI.

6.2.1. Transferring Detail Particles Between ROIs

In order to map the source detail particles into the destination ROI, the boundary particles on the source surface are transformed to match the boundary particles on the destination surface. The matching is performed as the following. Two tangent planes to the level set surface(s) are created, one at $x_s$ and the other at $x_d$. The source boundary particles ($B_s$) are projected, in the direction of the plane normal, onto the tangent plane created at $x_s$ and the destination boundary particles ($B_d$) are projected onto the tangent plane created at $x_d$. The projected source and destination particles maintain references back to the particles from which they originated in $B_s$ and $B_d$. All particles forming the projected $B_s$ are then translated and rotated onto the tangent plane at $x_d$, using the rotation that maps the surface normal at $x_s$ into the surface normal at $x_d$, as well as the rotation around the surface normal.

A series of rays connecting $P_d$ to each projected and transformed $B_s$ particle is defined and each ray is “intersected” with the projected $B_d$ particles. For each ray the closest particle in the set of projected $B_d$ particles is determined. The projected $B_s$ particle that defined the ray is then linked/matched with the closest projected $B_d$ particle. Via this linkage the original $B_s$ particle is moved to the original $B_d$ particle’s location on the destination surface. The $P_d$ and the moved $B_s$ boundary particles are held fixed at their locations, and the spring system moves all the other nearby source particles, distributing them over the destination surface as described in Section 5.1.2.
New particles are added, if necessary, to create a uniform sampling using techniques explained in Section 5.3.

Figure 8 illustrates the geometric texture mapping process in 2D. Frame (a) displays the source surface and its detail particles. (b) Clicking on the source surface identifies \( P_s \) and the ROI boundary particles \( B_s \). These particles are projected into a local tangent plane. (c) \( P_d \) and the \( B_d \) particles are projected onto a tangent plane near the destination surface. Using the two tangent planes the \( B_s \) particles (blue dots) are matched to \( B_d \) particles and moved to locations on the destination surface. \( P_s \) (green dot) is moved to \( P_d \)’s location. Note that the tangent planes at \( P_s \) and \( P_d \) are drawn below the actual surfaces to present a clearer image of the projection process. (d) The projected source detail particles are transformed to lie on the tangent plane near the destination surface, and are connected to the \( P_d \) and \( B_s \) particles lying on the destination surface. (e) The spring relaxation process is executed and the source detail particles are moved onto the destination surface. (f) The new particles are then used to add geometric details to the destination surface, as described in Section 5.2.

6.2.2. Tiling Detail Particles

If \( R_2 \) is greater than \( R_1 \), another approach may be utilized to tile the detail particles as many times as needed to fill in the larger destination ROI.
In this solution a set of center points \( (C_p) d = 1.5 \ast R_1 \) away from each other are computed on the destination surface. This sampling ensures that the destination ROI is completely covered by detail particles. Initially, \( P_d \) is added to \( C_p \) and its location is marked along with all surface voxels within a geodesic distance \( d \) to \( P_d \). The closest unmarked surface voxel in the destination ROI is then added to \( C_p \) and all unmarked surface voxels within \( d \) around this voxel are marked. This algorithm continues to add points to \( C_p \) until all surface voxels within the destination ROI are marked, i.e. the destination ROI has been covered by tiles of the source ROI. The next step transfers the detail particles, using the technique described in Section 6.2.1, from the source ROI to each of the sub-ROIs on the destination surface defined by the points in \( C_p \) and the radius \( R_1 \). If multiple particles fall within a surface voxel, they are combined into a single particle and their offset values and direction vectors are averaged. The geometric details are then incorporated into the destination ROI using the methods described in Section 5.2.

7. Results

In Figure 9 the back shell of the armadillo model (resolution 512\(^3\)) is edited to give it a larger hump. The edit expands the surface outwards and smooths out the details that define the turtle shell (Figure 9(b)). The detail particles created on the original model are maintained on the surface during the editing operation with the iterative method explained in Section 5.1.1. These particles are used to reconstruct the shell details on the edited model. Figures 9(c) and (d) provide a comparison of the edited model with and without details, and Figure 9(e) shows the complete and final state of the modified model. Figure 10 presents an edit of the armadillo’s snout. Some of the surface details are lost when the snout is pulled upwards. The projection method of Section 5.1.2 is used to add these details back onto the edited model.

Figure 11 presents an example of details being stretched and tiled over a surface during geometric texture transfer. First, a source ROI is specified on the leg of the armadillo model. The details from this region are captured and transferred to a region on the back of the model that has been smoothed by an editing operation. In Figure 11(c) the details have been stretched over the complete destination ROI. In Figure 11(d) the details are tiled to fill the destination ROI.
Figure 9: (a) Scan converted armadillo model. (b) Modifications to the model smooth out surface details on the back. (c) Two different views of the smooth surface. (d-e) Different views of the modified armadillo model after the surface details are added back.
Figure 10: A detail-preserving editing operation applied to the Armadillo’s snout. (a) Original model. (b) Applying an editing operator that smooths out details. (c) After adding back the details.

Figure 11: (a) Detail particles are taken from the Armadillo leg and (b) transferred to a larger region on the back of a smoothed version of the same model. (c) The details are stretched over the larger region. (d) The details are tiled over the larger region.
Figure 12: (a) Original scan converted model. (b) An edit removes the top part and smooths the head. (c) Back view of the original scan converted model. The details are extracted from within the highlighted ROI. (d) Top: The detailed surface. Bottom: Color-coded detail particles (Green - positive offsets, red - negative offsets). (e-f) The details that are extracted from the original model are added on top of the edited model.
Another example of geometric texture transfer is given in Figure 12. The lion head is edited at a resolution of $512^3$ to remove the capital from its head. Figure 12(b) shows the resulting smooth area on top of the head. Geometric surface details are extracted from the back of the same model as shown in Figure 12(c) and added to the smoothed area using geometric texture transfer as explained in Section 6. The detail particles are color-coded, green representing positive (outward) offsets and red representing negative (inward) offsets (Figure 12(d)). The final lion model with added hair details on top of its head is shown in the final two frames.

8. Discussion

8.1. Reconstruction Error

We evaluated the accuracy of the detail-preserving techniques by reconstructing a model from a smooth base surface and detail particles. The reconstructed detailed model was then compared with the original model, with the results shown in Figure 13. The high resolution surface details are removed using a binomial filter and added back with the methods explained in Sections 5.1.2 through 5.4. The reconstruction error is measured by calculating the Euclidean distance between the original and the reconstructed surfaces at the reconstructed surface’s volume grid points. The distance at a grid point is computed as the difference between the distance field values of the original and the reconstructed surfaces at that point. The minimum, maximum and average distances for the armadillo reconstruction are $4.09 \times 10^{-8}$, 1.48 and 0.07 voxels respectively, and are color-coded in Figure 13(f), with green representing minimum and red representing the maximum distances. 99% of the reconstructed surface was less than 0.5 voxels away from the original surface and only 0.025% of the reconstructed surface was more than 1 voxel away from the original surface.

8.2. Comparison of Run Times

The armadillo model shown in Figure 9 was edited twice, once using the iterative particle update method (Section 5.1.1), and once using the projection method (Section 5.1.2). Both edits were performed on both a high resolution $512^3$ model and somewhat lower resolution $256^3$ model. Table 1 shows run times for each step of both approaches and for both models on an Apple Mac Pro with a 3.2 GHz Intel Xeon CPU and 16GB of RAM.
Figure 13: (a) Filtered model (b) Reconstructed model, created by adding surface details to the filtered volume shown in (a). (c) Original high resolution model. (d) A closer view of the reconstructed head. (e) A closer view of the original head. (f) Distance between the reconstructed and the original model is color-coded, with red representing the maximum (1.48 voxels) and green representing the minimum distances (4.09 × 10^8 voxels).
<table>
<thead>
<tr>
<th>Size</th>
<th>Method</th>
<th>Steps</th>
<th>Run Times</th>
<th>Total Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>256(^3)</td>
<td>Iterative</td>
<td>Editing model &amp; Moving particles, Adding details</td>
<td>4.8 fps 1.8 sec</td>
<td>1.8 sec</td>
</tr>
<tr>
<td></td>
<td>Projection</td>
<td>Editing model, Projecting particles, Adding details</td>
<td>37 fps 5.6 sec 1.8 sec</td>
<td>7.4 sec</td>
</tr>
<tr>
<td>512(^3)</td>
<td>Iterative</td>
<td>Editing model &amp; Moving particles, Adding details</td>
<td>1.9 fps 12 sec</td>
<td>12 sec</td>
</tr>
<tr>
<td></td>
<td>Projection</td>
<td>Editing model, Projecting particles, Adding details</td>
<td>5 fps 31.6 sec 12 sec</td>
<td>43.6 sec</td>
</tr>
</tbody>
</table>

Table 1: A run-time comparison of the iterative and projection methods.

In the first (iterative) approach the details particles were moved along with the surface as it was being edited. This editing operation was computed and updated at a frame rate of 4.8 fps for the 256\(^3\) model and 1.9 fps for the 512\(^3\) model. In the second (projection) approach, the model is edited without moving the detail particles. This operation was performed at a frame rate of 37 fps for the 256\(^3\) model and 5 fps for the 512\(^3\) model. Once the editing operation was completed, the detail particles are projected onto the modified, smoothed surface and are used to restore details on the surface. The final step to add the details using the particles explained in Section 5.2 is the same for both approaches and takes 1.8 seconds for the 256\(^3\) model and 12 seconds for the 512\(^3\) model.

These run times show that for lower resolutions both methods (iterative and projection) of our modeling approach provide acceptable to passable interactive performance. But if the editing of a higher resolution model is required, then only the projection method offers a usable alternative. We acknowledge that waiting several seconds for a modeling operation to complete is not ideal or truly interactive. We expect that porting our techniques onto a GPU should significantly improve their performance, but for now we have separated the various computational components of our approach in a way that provides some kind of interactive experience to the user.
8.3. Limitations and Future Work

The techniques in their current state have a number of limitations, which point to issues requiring future work. Currently our techniques can only adequately capture details defined in the surface normal direction (displacement maps). It would be beneficial to represent / store / transfer more complex surface details. An application of Shell Maps [43] that captures the level set scalar field within a band near the surface could accomplish this; thus producing a variation of Brodersen et al.'s [44] technique.

A spatially uniform sampling of detail particles was used to produce the examples in this paper. While this sampling produced acceptable results, we acknowledge that adaptive sampling of the level set surface, e.g. based on curvature [52], should produce superior results with less loss of detail. This type of sampling is planned for future developments of this project.

Two methods have been described for maintaining detail particles on the level set surface as it is being edited, the iterative method (Section 5.1.1) and the projection method (Section 5.1.2). We used the projection method for most models, given its superior interactivity. Unfortunately, the projection method suffers from the major limitation that it is based on a local spring network to maintain particle connectivity during projection. This prevents it from being used with editing operators that change the topology of the model. The iterative method does not have this limitation. The model may change topology and the detail particles follow the different sections of the surface. This method though has unacceptably low frame rates when editing large-scale models. To address both of these issues (limits to change of topology and interactivity) we intend to explore GPU-based methods for level set evolution and updating the detail particles during editing. This should improve interactivity for larger models and offer a detail-preserving modeling method that supports change of topology. Additionally, GPU-based methods should also allow us to use higher particle resolutions; thus alleviating the sampling and reconstruction errors inherent in our approach.

9. Conclusion

We have developed novel detail-preserving techniques for level set modeling, that capture and store surface details in particle sets. These particles may then be used to add the surface details back to the models after they have been smoothed by an editing operation. The particle sets are dynamic and are kept on the surface after each operation. This allows us to use the
particles during interactive surface modifications to maintain small-scale surface details. Additionally, the detail particles may be copied from one surface to another; thus producing a level set geometric texture transfer capability.

Acknowledgements

This research was supported by NSF grant CCF-0702441. We would like to thank Ross Whitaker for the VISPACK level set software. We would also like to thank Ken Museth for providing Dr. Eyiyurekli with an internship at Dreamworks Animation. This experience provided valuable insight into working with large-scale level set models.

References


