Interactive Implicit Modeling With Hierarchical Spatial Caching

Ryan Schmidt
University of Calgary
Computer Science
rms@cpsc.ucalgary.ca

Brian Wyvill
University of Calgary
Computer Science
blob@cpsc.ucalgary.ca

Eric Galin
LIRIS - CNRS
Université Claude Bernard Lyon 1
eric.galin@liris.cnrs.fr

Abstract

Complex implicit CSG models can be represented hierarchically as a tree of nodes (the BlobTree) . However, current methods cannot be used to visualize changes made to these models at interactive rates due to the large number of potential field evaluations required. A heirarchical spatial caching technique is presented which accelerates evaluations of the potential function. This method introduces the concept of a caching node inserted into the implicit model tree. Caching nodes store exact potential field values at the nodes of a voxel grid and rely on tri-linear and tri-quadratic reconstruction filters to locally approximate the potential field of a sub-tree. A lazy evaluation scheme is used to avoid expensive pre-computation.

Polygonization timings with and without caching are compared for a complex model undergoing manipulation in an interactive modeling tool. An order-of-magnitude improvement in visualization time is achieved for complex implicit models containing thousands of primitives.

1. Introduction

Interactive modeling is an iterative, essentially trial-and-error process. Components of complex models are created independently and then assembled. A significant portion of the designer's time is spent making small changes to model parameters and examining the results. To facilitate this kind of interaction, interactive visual feedback is necessary.

Implicit surface modeling has been hampered by the lack of a fast interactive visualization method. Local-update methods [9] are effective only when the underlying model tree is simple. When assembling the components of a complex model, computing a single field value can require a very large number of leaf-node field evaluations and composition operations.

Standard implicit surface visualization methods rely on computing the value and gradient of the scalar field many times. In profiling implicit surface polygonizers, we observe that over 95% of the computation time is spent in field value evaluations. Potential field function evaluation cost must be reduced in order to provide interactive visual feedback for complex models.

We propose a hierarchical spatial caching technique that can be used to greatly decrease the number of tree nodes traversed when computing a single field value or gradient. We have implemented spatial caches using uniform 3D grids and store field values at grid vertices. Tri-linear and tri-quadratic reconstruction filters are applied to these cached field values to locally approximate the potential field that defines the implicit surface. Using this process the computational complexity of a single field evaluation for a cached sub-tree is reduced from $O(n)$ to $O(1)$, where n refers to the number of nodes in the subtree. Our results show that this caching technique is capable of providing an order-of-magnitude decrease in interactive polygonization time for complex hierarchical skeletal implicit models.

The remainder of this paper will proceed as follows. In Section 2 we will describe related work. Section 3 describes the fundamentals of the BlobTree modeling system. Section 4 introduces spatial caching nodes to the BlobTree and addresses the spatial caching process. Section 5 describes some implementation details regarding memory considerations. Results and future work are covered in Sections 6 and 7, respectively.

2. Related Work

A variety of implicit surface modeling systems have been proposed in the last decade. These systems use various underlying implicit representations. A small sample includes skeletal elements [19], levels sets [14], convolution surfaces [4] [11], adaptive distance fields [10] variational implicit surfaces [16] and functional representations [15].

All of these systems can be used to model complex shapes. Recent interactive sculpting [7] [3] and sketch-based modeling [12] [1] systems provide interactive visualization rates while updating the implicit model. How-

ever, fundamentally these systems construct global models. By this we mean that the final implicit model is not composed of smaller components that can be individually manipulated, but rather a single volume. Components of a complex model can be created individually but must eventually be composited into the global volume. This step is essentially irreversible, later local editing or animation of components is very difficult. Composition operators can instead be applied in the context of a general implicit modeling system, such as the BlobTree [19]. A BlobTree model with hierarchical affine transformation nodes is essentially a scene graph, permitting animation of model components.

Unfortunately, no existing implicit surface visualization technique provides interactive performance for complex BlobTree models. The key issue is that a functionally-defined implicit surface must be converted to a discrete representation, such as a triangle mesh or point set, to be visualized on commodity graphics hardware. This surface extraction process is too computationally demanding to execute in real time as a designer interactively modifies a complex implicit model.

One avenue for reducing surface extraction time is to incrementally compute a new surface discretization as local changes are made by the designer. Some proposed techniques include incremental octree subdivision of space [8], adaptive marching triangles approaches [9] and surface constrained particle systems [18]. Interactive performance has been demonstrated for relatively simple models. However, incremental update schemes do not scale to complex hierarchical models because the cost of evaluating even the local update region becomes too expensive. In addition, large-scale model changes are not accelerated.

Level-of-detail schemes for implicit surfaces [2] [11] attempt to speed up potential field queries by dynamically reducing the complexity of the implicit model tree. These techniques reduce interactive visualization times when lower levels of detail are used. Our technique can be used in conjunction with these methods to reduce polygonization time at all levels of detail.

Our goal is to enable interactive modeling of complex implicit surfaces by accelerating potential field queries. We use potential field function approximations similar to [3], where the implicit model was stored as a set of potential field value samples in a uniform 3D grid. A smooth C^1 continuous surface was extracted from these samples using reconstruction filters. We adapt this technique to dynamically construct spatial caches in our heirarchical model. We approximate the potential field function whenever possible, avoiding the complex tree traversals required when evaluating a potential field query. This work is in spirit related to the render cache system [17] that exist for speeding up interactive ray-racing applications.

3. The BlobTree

An implicit surface is mathematically defined as the points in space that satisfy the equation

$$\mathcal{S} = \{\mathbf{p} \in \mathbb{R}^3, f(\mathbf{p}) - T = 0\}$$

where $f(\mathbf{p})$ denotes a scalar field function in space and T a threshold value. The BlobTree model [19] is characterized by a hierarchical combination of primitives organized in a tree data-structure (Figure 1). The nodes of the tree include blending, CSG and warping nodes, whereas the leaves are skeletal elements.

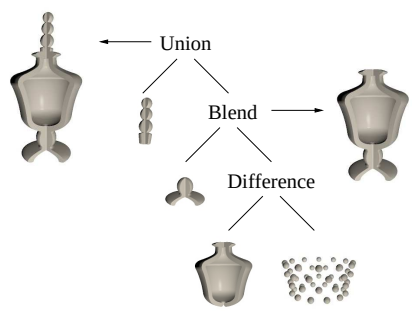


Figure 1. A simplified representation of the tree structure of a bottle

Skeletal elements are defined by a skeleton, a distance function and a potential field function with a compact support. Therefore, every skeletal element has a bounded region of influence in space. Every node incorporates a bounding box as to rapidly discard useless field function evaluations when queries are performed in empty regions of space.

The computation of the field function $f(\mathbf{p})$ at a given point in space is performed by recursively traversing the BlobTree structure. The skeletal primitives at the leaves of the tree return potential field values, which are combined by the operators at the nodes of the tree.

Field function evaluation is the most computationally demanding step in implicit surface visualization. Ray tracing techniques require many field function evaluations along a ray to isolate the roots and converge to the ray-implicit surface intersection. Polygonization techniques also query the BlobTree with many potential field function evaluations. Therefore, it is crucial to be able to accelerate the computation of $f(\mathbf{p})$ for interactive modeling applications.

4. Hierarchical spatial caching

Hierarchical spatial caching is integrated into the Blob-Tree implicit modeling framework by introducing the caching node. A caching node $\mathcal{C}$ is a unary operator with a single subtree $\mathcal{T}$. Each caching node stores a set of exact potential field values determined by querying it's subtree $\mathcal{T}$. When evaluating the potential field of $\mathcal{T}$ at a point $\mathbf{p}$ inside the bounding box of $\mathcal{T}$, an approximation $f_{\mathcal{C}}(\mathbf{p})$ to the exact potential field value $f_{\mathcal{T}}(\mathbf{p})$ is reconstructed from the cached potential values. The subtree $\mathcal{T}$ is not traversed.

In the next sections we will describe in detail our implementation of caching nodes.

4.1. Data structure

In our system caching nodes are inserted above composition nodes. We specifically do not place a caching node at the root of the tree, as this cache would require continuous updating in an interactive system. Our caching nodes store the exact potential field values at the nodes of a voxel grid (Figure 2). In the remainder of this section $\mathbf{p}_{ijk}$, $(i, j, k) \in [0, n]^3$ will refer to a point in the voxel grid. The corresponding potential field value stored at $\mathbf{p}_{ijk}$ will be denoted as $v_{ijk} = f_{\mathcal{T}}(\mathbf{p}_{ijk})$.

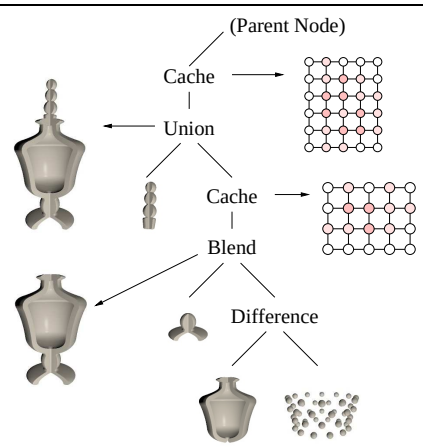


Figure 2. A bottle model with some cache nodes. The cache resolution for a subtree $\mathcal{T}$ is dependent on the subtree size

Each caching node has a local reference frame, allowing the uniform grid to be rotated and translated in line with the orientation of it's subtree $\mathcal{T}$. This avoids expensive cache recalculation when the entire subtree is manipulated.

When some descendent node of a caching node changes, the bounding box of the descendent node is used to locally invalidate cached values in all parent caches. Invalidation is carried out at each cache by clearing a *valid* flag for all cached values inside the bounding box.

4.2. Caching algorithm

Caching nodes are implemented using reconstruction filters that approximate the field function $f_{\mathcal{T}}(\mathbf{p})$ using nearby exact field value samples $(\mathbf{p}_{ijk}, v_{ijk})$ of the subtree $\mathcal{T}$. Approximating the field function of the subtree $f_{\mathcal{T}}(\mathbf{p})$ by $f_{\mathcal{C}}(\mathbf{p})$ is more efficient only if the necessary field value samples $(\mathbf{p}_{ijk}, v_{ijk})$ have been pre-computed.

The steps taken by a cache node to compute $f_{\mathcal{C}}(\mathbf{p})$ are as follows:

1. Transform $\mathbf{p}$ into local cache coordinates $\mathbf{p}'$
2. If the necessary field value samples $(\mathbf{p}_{ijk}, v_{ijk})$ required to evaluate $f_{\mathcal{C}}(\mathbf{p}')$ are cached, go to step 4
3. Cache any missing field value samples $(\mathbf{p}_{ijk}, v_{ijk})$ by evaluating the subtree.
4. Evaluate $f_{\mathcal{C}}(\mathbf{p}')$

4.3. Lazy evaluation

The initial field values used by spatial cache nodes to calculate interpolated field values are found by evaluating the child node's field value. For complex subtrees these evaluations can be quite expensive. The overhead necessary to create and maintain a fully-evaluated uniform grid has a significant impact on interactivity.

Surface-following continuation methods [6] generally only evaluate field values at points near the surface. We exploit this by introducing lazy evaluation into our spatial cache nodes. Initially the spatial cache data structure is empty. When a field value at some point inside the spatial cache is requested, the cached values necessary to compute the interpolated field value are evaluated and stored for future use (Figure 3).

4.4. Potential field reconstruction

Tri-quadratic reconstruction filter The tri-quadratic reconstruction filter [3] is a separable, C^1 continuous filter. Evaluation of the filter at a point $\mathbf{p}$ requires 27 neighbouring samples. The filter is an approximating filter and hence does not necessarily pass through the sample points. We will first describe the one-dimensional quadratic reconstruction filter, as the three-dimensional filter is defined in terms of 13 applications of the one-dimensional filter.

Evaluation of the 1D quadratic B-spline filter $\mathcal{R}_q$ requires 3 sample points s_{i-1}, s_i, and s_{i+1} to reconstruct a

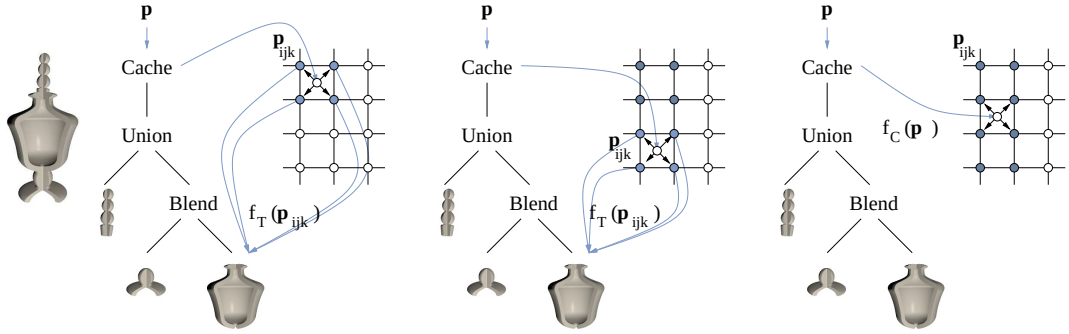


Figure 3. Lazy evaluation process at three consecutive points. For the first two points (left and center images), missing cache samples (empty circles) must be evaluated (filled circles) by traversing the subtree. All necessary cache samples are evaluated for the right point, so the subtree is not traversed.

signal over the interval $\mathcal{I} = [i - 0.5, i + 0.5]$. $\mathcal{R}_q$ is a function of one parameter $t \in [0, 1]$ defined over $\mathcal{I}$.

We define the 1D quadratic B-spline filter $\mathcal{R}_q$ as

$$\mathcal{R}_q\left(s_{i-1}, s_i, s_{i+1}, t\right) = \left(\frac{s_{i-1} + s_{i+1}}{2} - s_i\right) t^2 + \left(s_i - s_{i-1}\right) t + \frac{s_{i-1} + s_i}{2}$$

We can now define the tri-quadratic reconstruction filter $\mathcal{R}_{q^3}$. Evaluation of the filter at a 3D sample point requires 27 neighbouring voxels. For a 3D sample point $\mathbf{p}$, we find the nearest voxel v_{ijk}. The one-dimensional parameter t is computed along each grid axis, these are denoted t_i, t_j, and t_k. $\mathcal{R}_{q^3}(\mathbf{p})$ is calculated by repeated applications of $\mathcal{R}_q$, defined in the the following set of equations.

$$\begin{aligned}
\mathcal{R}_{q^3}(\mathbf{p}) &= \mathcal{R}_q\left(\mathcal{R}_j(k-1), \mathcal{R}_j(k), \mathcal{R}_j(k+1), t_k\right) \\
\mathcal{R}_j(\rho) &= \mathcal{R}_q\left(\mathcal{R}_i(j-1, \rho), \mathcal{R}_i(j, \rho), \mathcal{R}_i(j+1, \rho), t_j\right) \\
\mathcal{R}_i(\phi, \rho) &= \mathcal{R}_q\left(v_{(i-1)\phi\rho}, v_{i\phi\rho}, v_{(i+1)\phi\rho}, t_i\right)
\end{aligned}$$

Here ϕ and ρ are placeholders. $\mathcal{R}_i(\phi, \rho)$ is an evaluation of $\mathcal{R}_q$ in the i direction, and $\mathcal{R}_j(\rho)$ is an evaluation of $\mathcal{R}_q$ in the j direction.

Reconstruction filter implementation We reconstruct a smooth scalar value field by applying both reconstruction filters to the potential field values stored in the uniform grid. We have evaluated both tri-linear and tri-quadratic reconstruction filters. Our current system uses tri-linear interpolation to reconstruct the potential field and tri-quadratic approximation to reconstruct the field gradients. We found this to be the best trade-off between polygonization time and appearance of surface smoothness.

Tri-linear interpolation is very efficient, requiring only 8 grid values. However, as shown in [13], the tri-linear filter is only C^0 continuous. This causes significant gradient error, shown in Figure 4.

The tri-quadratic reconstruction filter produces a smooth gradient, however it is significantly more expensive than the tri-linear filter and requires more adjacent potential field value samples. High-frequency details of the original potential value field can also be lost due to the approximating property of the filter.

Polygonization times with the tri-quadratic filter were approximately twice the tri-linear filter times. However, we found that using tri-linear reconstruction for field values and tri-quadratic reconstruction for gradients was only 10% more expensive than a pure tri-linear approach. The resulting mesh is visually much smoother, as seen in Figure 4.

4.5. Approximation Accuracy

The accuracy of the potential field reconstruction is entirely dependent on the uniform grid resolution. In our current implementation the resolution of a uniform grid caching a subtree $\mathcal{T}$ is dependent on the axis-aligned bound-

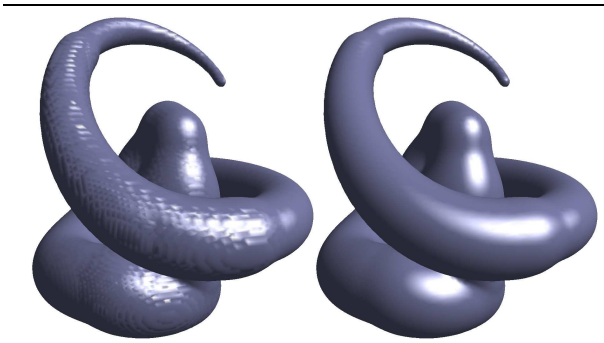


Figure 4. Tri-linear gradient (left image) and tri-quadratic gradient (right image).

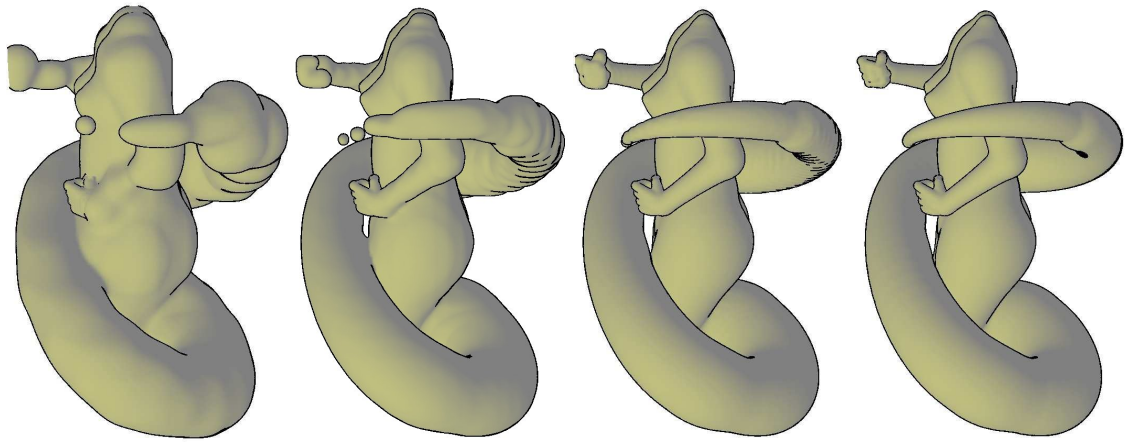


Figure 5. Varying cache resolution with tri-quadratic value reconstruction. There are three cached components in this model - upper body, lower body, and left hand. Cache resolution for each component is varied from left to right 16^3, 32^3, 64^3, and 128^3.

ing box of the subtree and a user-defined grid resolution r. Let s denote the longest side of the bounding box of $\mathcal{T}$. The size of each grid cell is then defined as

$$c = s/r$$

If some descendent of the cache node $\mathcal{C}$ is modifed, the bounding box of $\mathcal{T}$ will change. In this case we compute a new cell size c'. If the ratio c'/c is greater than 2 or less than 0.5, the current cache is destroyed. The cache will be automatically re-populated by the lazy evaluation scheme. This prevents cache resolution both from getting too low, which would result in a coarse approximation of the surface, as well as from getting too high and wasting memory.

The effect of varying the grid resolution parameter r is demonstrated in Figure 5. In these images we use tri-quadratic reconstruction for both the value and gradient, as it generates smoother surfaces at low cache resolutions. Examining the left hand component of the model is instructive. Even at a grid resolution of 16^3, the hand is reasonably well-represented for this viewing distance, while 64^3 is clearly too coarse for the body components. These images indicate that an automatic cache level-of-detail algorithm could reduce both memory use and computation time.

One potential issue with caching nodes is a loss of sharp features. The potential field around several sharp features is shown in Figure 6. The image pixels are colored by calculating $sin(11\pi f(\mathbf{p}))$ along a plane slice and converting the result to grayscale; pixels near the iso-value 0.5 are colored red. At this resolution tri-linear reconstruction is capable of properly reconstructing sharp features that lay on voxel edges. Tri-quadratic reconstruction smooths out all sharp features. The reconstruction accuracy improves in both cases if cache resolution is increased.

An important error metric for evaluating the caching scheme is to measure the absolute difference between the cached field value $f_{\mathcal{C}}(\mathbf{p})$ and the exact field value $f(\mathbf{p})$. We denote this difference $error_{\mathcal{C}}$:

$$error_{\mathcal{C}} = |f_{\mathcal{C}}(\mathbf{p}) - f(\mathbf{p})|$$

We evaluate the approximation error visually in figure 7. In these images, the model is rendered using a triangle mesh with each vertex colored based on $error_{\mathcal{C}}$. The range of the color scale is held constant across the four images. It is clear that trilinear interpolation has less error that tri-quadratic approximation. The “ringing” in the trilinear images corresponds to the areas of the largest gradient error in figure 4.

Using this metric we can detect when part of the surface is not adequately represented by the cache. The right hand of the model has significant error even at 256^3 resolution. However, the right hand is stored in the same cache as the upper body. Clearly the right hand portion of the model tree should be split into a separate cache. By randomly sampling $error_{\mathcal{C}}$ at mesh vertices, it is possible to detect undersampling and automatically insert new caches into the model tree where necessary, or increase the resolution of an existing cache. Currently we have not implemented this feature.

5. Implementation details

Uniform 3D grids can require significant amounts of memory. We store our cached field values in single-precision floating point, so a 128^3 cache requires 8MB of memory. Current hardware limitations prevent stor-

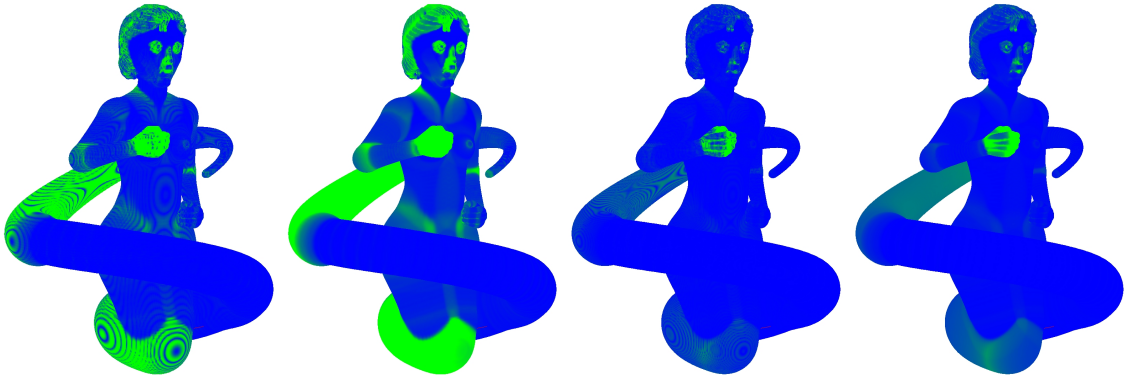
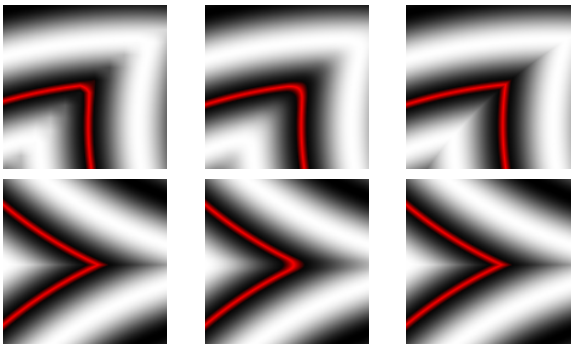


Figure 7. Approximation error. Mesh vertices are colored based on the absolute difference between the approximated field value and the accurate field value. The color scale runs from blue (low error) to green (high error). The images shown are (a) 128^3 trilinear reconstruction, (b) 128^3 triquadratic, (c) 256^3 trilinear, and (d) 256^3 triquadratic.



(a) (b) (c)

Figure 6. Sharp feature reconstruction using 128^3 cache. Tri-linear reconstruction (a), tri-quadratic reconstruction (b), and non-cached evaluation (c).

ing a significant number of fixed caches at this resolution. In addition, the volume represented by a single cache can expand, making a fixed grid data structure undesirable.

To reduce memory usage we implement our uniform 3D grid using a blocked memory scheme. We divide the uniform grid into 8x8x8 blocks of voxels. A uniform grid block is allocated only when one of the voxels it contains is needed to compute a field value. The blocks are quickly identified using a 30-bit hash, using 10 bits per integer grid axis coordinate. This limits us to 1024 blocks along each grid axis, and a total grid size of 8096^3 voxels.

Continuation methods for polygonizing an implicit surface [6] are designed to follow the surface. Most field evaluations are near the surface, hence the required voxels will also be near the surface. In this case our data structure reduces memory use while still permitting efficient evaluation of the reconstruction filters.

Current workstation processors contain several levels of hardware memory cache to reduce memory access latency. The traditional static allocation of a large 3D uniform grid causes frequent cache misses, particularly when accessing adjacent voxels along the z axis. Our blocked allocation scheme potentially increases processor cache coherency, however this effect is difficult to isolate.

Memory requirements can be reduced even further by using an encoding scheme, at the expense of some reconstruction accuracy. The range of potential field values that can occur in our system is small. Memory usage can be reduced 50 to 75 percent by encoding floating point values as one or two-byte integers. This technique is slightly more computationally expensive and decreases accuracy, so we do not use it in our evaluation.

6. Results

Cache efficiency is evaluated by comparing polygonization times between two versions of the same model, one with cache nodes and one without. We use an optimized version of the implicit surface polygonizer described in [5] with the optional cubical decomposition enabled. When computing a mesh vertex on a cube edge, 10 bisections are performed to locate the implicit surface.

The software is compiled with Microsoft Visual Studio .NET 2003 in Release mode with default optimization flags. All timings are performed with an Intel 1.6Ghz Mobile Pentium 4 processor with 512MB of RAM.

6.1. Medusa model

We have tested the system with a complex hierarchical Medusa model. The model is composed of 9490 point skeletal elements segmented into 7 major components. The names and point counts are shown in the table below.

Tail	Body	Chest	Left Hand	Neck	Head	Hair
810	840	40	570	20	700	6510

Each major component is modeled as a blend of point skeletal elements distributed along a set of splines. All the points along each individual spline are grouped together into a single optimized blend node to avoid excessive tree traversal.

A caching node with a resolution of 128^3 voxels was placed above each major component in the model tree. We approximate potential field reconstruction accuracy by comparing reconstructed and exact field values at triangle mesh vertices. Using this method, the mean tri-linear reconstruction error for the Medusa model with 128^3 voxel caches is estimated to be approximately 3%. The error is concentrated in high-frequency regions, particularly the head. We have not devised a global measure of reconstruction accuracy, however it is suspected that the global error will be similar to the near-surface error because the cache spacing is uniform.

6.2. Static polygonization time

We compare polygonization times for the Medusa model with and without caching nodes. All caches are cleared before each polygonization. Our results are shown in Table 1. In all cases the cached polygonization contained less than 1% more triangles that the non-cached polygonization.

Cubes	Cache	No Cache	Ratio
32^3	5.77	4.90	$0.8\times$
64^3	10.34	14.36	$1.4\times$
128^3	17.40	51.97	$3\times$
256^3	29.23	199.37	$6.5\times$
512^3	49.83	809.66	$16\times$

Table 1. Comparison of cached and non-cached polygonization times (in seconds) for Medusa model at different polygonization resolutions.

Polygonization time without caching increases by approximately a factor of 4 when resolution doubles. Polygonization time with caching is initially slower than without because we require 8 voxels to compute a single tri-linear interpolation. Subsequently, polygonization time reduces as the cache is populated. The ratio between cached times at consective resolutions decreases because more cached voxels are re-used. At 512^3 polygonizer resolution, approximately 33% of all cache voxels have been evaluated.

We have repeated these tests with a variety of other models constructed from skeletal elements more complex than points, including those shown in Figure 9. Similar results were observed.

6.3. Interactive polygonization time

We have shown that caching nodes are an effective means for increasing static polygonization time. However, the main benefit of caching nodes becomes apparent when interactively manipulationg model components. Figure 8 compares several different tests we performed on the Medusa model with and without caching nodes.

In each test we simulated translation of the Medusa head component. The head was moved 25 steps towards the tail, then 25 steps back to the original position. Along this path the head intersects all other components of the model. An initial polygonization was computed before running each test, hence the cache begins partially evaluated. Polygonizer resolution was fixed at 60^3 cubes, based on the initial model bounding box.

Polygonization time is relatively constant for the non-cached cases, which is expected. With caching nodes the polygonization time rapidly drops over the first few frames. Many of the caching node voxels for the head component are being evaluated over these frames. During the rest of the downward path small numbers of potential field values are cached for other components as the head intersects them. This results in a relatively stable polygonization time. At step 25 the upward path begins. The stable polygonization time decreases at this point because no non-cached potential field evaluations are necessary.

Tests were performed with and without the hair component. The first few frames are computed more quickly without the hair component, however the stable polygonization time is essentially identical in both cases. This comparison shows that after the caches are populated, polygonization time is insensitive to the underlying model complexity and depends primarily on surface complexity.

Finally, polygonizer resolution is doubled to 120^3 cubes. The cached polygonization converges to a stable time of approximately 1.25 seconds per frame. Non-cached timings were over 50 seconds per frame.

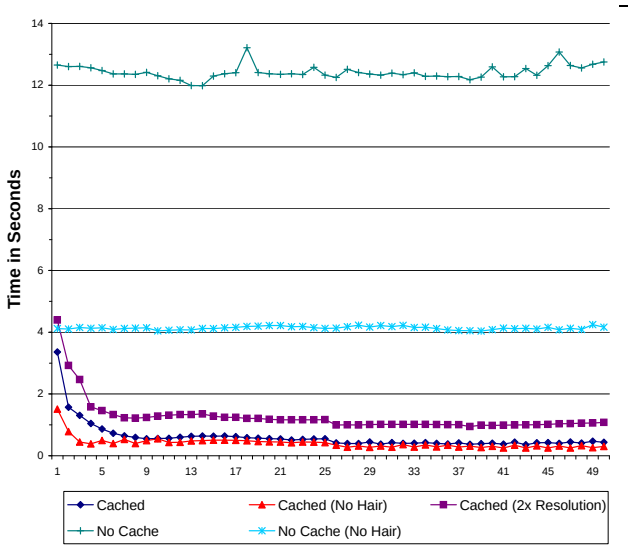


Figure 8. Comparison of cached and non-cached polgyonization times recorded while simulating interactive translation of Medusa head.

Figure 9. Some models used for evaluating our caching system

Similar results have been observed while repeating these tests with other model components.

6.4. Local update polygonization

A common technique for improving interactive visualization time in implicit modeling systems is to only re-polygonize the model in areas that have been modified. Several tests with and without caching nodes have been performed to simulate this behavior using our cubical decomposition polygonizer [5]. The update region is restricted using the bounding box of model components that have changed. Because the cubes used are a subset of those used for full polygonization, the same triangles are generated and we can directly compare times between the two cases.

The interactive assembly task simulated described in Section 6.3 was performed using local updates. The average timing improvements are shown in Table 2. Results from another test are also shown, in which we loaded only the hair component and moved a small point skeletal element through it. The point bounding box covers less than 5% of the total hair volume.

Test	Cubes	Improvement
Head Translation	60^3	$7\times$
Head Translation	120^3	$12\times$
Point / Hair Test	60^3	$30\times$
Point / Hair Test	120^3	$47\times$

Table 2. Comparison of cached and non-cached polygonization times for local-update polygonization tests. *Cubes* column refers to polygonizer resolution, *Improvement* column shows number of times speed-up with caching nodes.

7. Conclusion and future work

We have described a new type of BlobTree node, the spatial caching node. Our caching node approximates the potential field of a caching node subtree using a set of exact potential field value samples. The approximation process reduces potential field evaluation complexity for the subtree from $O(n)$ to $O(1)$, where n refers to the number of nodes in the subtree.

We implement caching nodes using uniform grids with subtree-dependent resolution. Our implementation uses tri-linear reconstruction for potential field values and tri-quadratic reconstruction for potential field gradients. This combination of reconstruction filters produces visually smooth meshes suitable for interactive visualisation in an implicit modeling tool. Complex hierarchical implicit models created using our interactive modeling tool are shown in figure 10.

We demonstrate an order-of-magnitude decrease in polygonization time when inserting caching nodes into a complex BlobTree model containing over 9000 blended point primitives. The benefit of caching nodes increases significantly at higher polygonization resolutions. Our analysis shows that caching nodes are particularly effective when used in conjunction with local-update polygonization.

Caching nodes are an effective tool for reducing interactive polygonization time when using continuation methods. Interactive visualization benefits both shape modeling and

animation prototyping. We anticipate that caching nodes will provide a similar performance benefit with other implicit surface operations, including ray-tracing, collision detection and other polygonization schemes.

There are a variety of avenues for future work on caching nodes. A primary concern is memory management. Our current implicit modeling tool requires the model designer to manage placement of caching nodes. Ideally, an interactive modeling system would infer appropriate cache placement and resolution based on the designer's actions.

Our system uses a blocked uniform grid scheme to obtain maximum computational efficiency while still maintaining some flexibility. It may be acceptable to trade some of that efficiency for other parameters, such as increased reconstruction accuracy. Alternate spatial data structures, particularly multi-grid methods, may provide this trade-off.

References

[1] B. Araújo and J. Jorge. BlobMaker: Free-Form Modeling with Variational Implicit Surfaces. *Comunicação ao 12º Encontro Português de Computação Gráfica* , 2003.

[2] A. Barbier, E. Galin and S. Akkouche. Complex Skeletal Implicit Surfaces with Levels of Detail. *Proceedings of WSCG*, **12**(4), 35–42, 2004.

[3] L. Barthe, B. Mora, N. Dodgson and M. Sabin. Interactive implicit modelling based on C^1 reconstruction of regular grids. *International Journal of Shape Modeling*, **8**(2), 99–117, 2002.

[4] J. Bloomenthal and K. Shoemake. Convlution Surfaces. *Computer Graphics (Proceedings of SIGGRAPH 91)*, **25**(4), 251–256, 1991.

[5] J. Bloomenthal. An Implicit Surface Polygonizer. *Graphics Gems IV*, Academic Press Professional Inc., 324–349, 1994.

[6] J. Bloomenthal (Ed.). Introduction to Implicit Surfaces. Morgan Kaufmann, ISBN 1-55860-233-X, 1997.

[7] E. Ferley, M.-P. Cani, J.-D. Gascuel. Practical Volumetric Sculpting. *The Visual Computer*, **16**(8), 469–480, 2000.

[8] E. Galin and S. Akkouche. Incremental Polygonization of Implicit Surfaces, *Graphical Models*, **62**, 19–39, 2000.

[9] S. Akkouche and E. Galin. Adaptive Implicit Surface Polygonization using Marching Triangles. *Computer Graphics Forum*, **20**(2), 67–80, 2001.

[10] S. Frisken, R. Perry, A. Rockwood, T. Jones. Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics. *Proceedings of SIGGRAPH 2000*, 249–254, 2000.

[11] S. Hornus, A. Angelidis and M.-P. Cani. Implicit Modeling using Subdivision-curves. *The Visual Computer*, **2**(3), 94–101, 2003.

[12] O. Karpenko, J. Hughes and R. Raskar. Free-Form Sketching with Variational Implicit Surfaces. *Computer Graphics Forum*, **21**(3), 585–594, 2002.

[13] S. Marschner and R. Lobb. An Evaluation of Reconstruction Filters for Volume Rendering. *Proceedings of Visualization 1994*, 100–107, 1994.

[14] K. Museth, D. Breen, R. Whitaker and A. Barr. Level Set Surface Editing Operators. *ACM Transactions on Graphics*, **21**(3), 330–338, 2002.

[15] A. Pasko, V. Adzhiev, A. Sourin and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, **11**(8), 429–446, 1995.

[16] G. Turk and J. O'Brien. Shape Transformation Using Variational Implicit Functions. *Proceedings of SIGGRAPH 99*, 335–342, 1999.

[17] B. Walter, G. Drettakis and S. Parker. Interactive Rendering using the Render Cache. *Proceedings of the 10th Eurographics Workshop on Rendering*, **10**, 235–246, 1999.

[18] A. Witkin and P. Heckbert. Using Particles to Sample and Contol Implicit Surfaces. *Proceedings of SIGGRAPH 94*, 269–278, 1994.

[19] B. Wyvill, A. Guy and E. Galin. Extending the CSG Tree (Warping, Blending and Boolean Operations in an Implicit Surface Modeling System). *Computer Graphics Forum*, **18**(2), 149–158, 1999.

Figure 10. Complex hierarchical implicit models created interactively using our system.