ICAT 2005
December 5-8, Christchurch, New Zealand

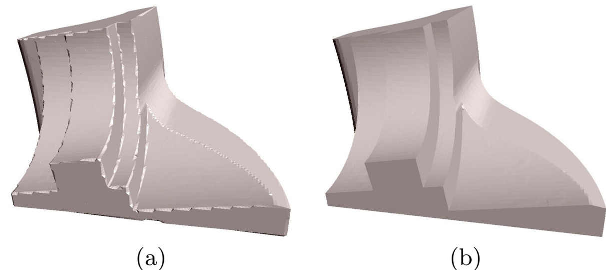
Detail Sculpting using Cubical Marching Squares

Chien-Chang Ho, Cheng-Han Tu and Ming Ouhyoung
National Taiwan University
[murphyho,toshock]@cmlab.csie.ntu.edu.tw, ming@csie.ntu.edu.tw

Abstract

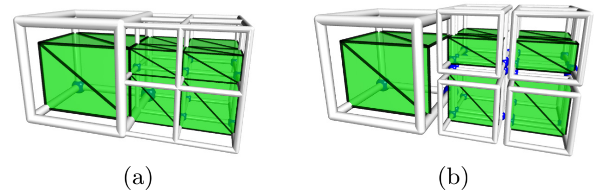
The demand for customizing models is increasing rapidly, but the related techniques for sculpting in the volumetric form raise several issues that need to be addressed. These issues include preserving sharp features, inter-cell independency, maintaining consistent topology, and adaptive resolution. These issues affect the quality of resulting shapes and the execution speed while manipulating models. Traditionally, marching cubes algorithm provides satisfactory performance for visualizing volume data in sculpting applications. However, sharp features, which are important characteristics of models, are lost while using marching cubes to visualize the resulting volume data. Furthermore, we need to perform crack patching operations to fill up the gaps between different resolutions in adaptive resolution. In contrast, these issues could be easily fulfilled by replacing the underlying isosurfacing algorithm with cubical marching squares algorithm. In this paper, we propose data structures for storing volume data and methods for Boolean operations to manipulate these data. We archive highly detailed models while sculpting at interactive speed by using proposed data structures and methods in conjunction with cubical marching squares algorithm.

Keywords: marching cubes, Boolean operations, volume sculpting



(a) (b)

Figure 1: A sharp features preserving algorithms restores edges or corners of a fandisk model in volumetric form. (a) The result of marching cubes. (b) The result of a surface extracting algorithm with sharp features preserving ability.



(a) (b)

Figure 2: Inter-cell independency. (a) Cells in different resolutions. (b) Cells are disjoint in space which is an witness of inter-cell independency.

1. Introduction

Recently, due to the more and more widely use of three-dimensional digital animations, the demand for various models is increasing rapidly. To produce a three-dimensional model, we need different kinds of specialties or special equipments to make a model. The requirements for customizing a three-dimensional model are the same as producing them. In contrast, sculpting [2, 9, 12, 13, 15] in the virtual world provides a natural way to let people intuitively customize models.

To perform sculpting in the virtual world, volumetric techniques [7, 8, 10] are usually involved to provide a smooth and efficient sculpting environment. One benefit of using volumetric techniques is that we can easily predict the resulting performance according to the space we used. However, the related techniques for sculpting in the volumetric form raise several issues that need to be addressed. The first issue is preserving sharp features. Comparing to original marching cubes algorithm, sharp features preserving algorithms provide better appearances if the shapes of volume data contain edges or corners, Fig. 1 shows an example. The second issue is inter-cell dependency. Although sharp feature preserving algorithms preserve sharp features of models, unfortunately, they introduce another problem, the dependency among cells. This problem limited the parallelizability of applications using these techniques. Hence, the ability to accelerate by GPU becomes harder. Fig. 2 shows the visualization of inter-cell independency, where cells are disjoint in space. The third issue is maintaining consistent topology. There have been many ambiguities found in certain marching cubes configurations where there are more than one ways to triangulate, Fig. 3 shows an example. To more precisely represent a shape inside a cell, different kinds of topology should be detected and rendered. Finally, adaptive resolution is required to produce more detail using the same size of memory. However, they can result in cracks [14] at the interfaces of grid cells at different resolutions. Fig. 4 shows an example of cracks.

However, these issues could be easily fulfilled by replacing the underlying isosurfacing algorithm with cubical marching squares algorithm [6]. Our sculpting process has three stages: storing volume data in a special data structure, applying Boolean operations, and, finally, extracting surfaces from the stored volume data. To sculpt under cubical marching squares algorithm, we proposed data structures and methods which can operate at interactive speed while performing Boolean operations. The outline of this paper is organized as following. First, section 2 describes related work. Second, section 3 describes the data structure for the volume data that reduces the storage and makes the computation faster. Third, section 4 describes the algorithms to

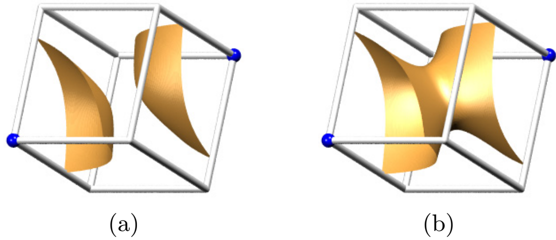


Figure 3: An example shows different topologies co-existing in the same configuration.

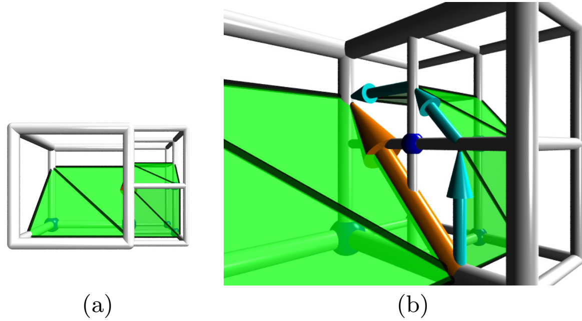


Figure 4: Adaptive resolution causes cracks. (a) The front view of the scene. (b) The close-up view of the scene, where orange and cyan arrows highlight the crack.

perform Boolean operations on the data structure. Fourth, section 5 describes the way we render the volume data. Finally, the rest of the sections discuss the result, conclusion, and future work.

2. Related Work

Sculpting in a virtual world is one of the applications that require fast rendering from polygonal representation and fast CSG operations from volumetric representation. The marching cubes algorithm[10] provides a convenient and efficient way to convert volume data to isosurfaces, hence permitting us to represent objects as volume data, manipulate them volumetrically and efficiently display them by converting to isosurfaces on the fly. Galyean *et al.* [4] propose an volumetric modeling technique based on the marching cubes algorithm. However, although the original marching cubes algorithm is generally effective, it has the problems described above and will affect the correctness and the accuracy of the surface representation. To reduce the number of triangles, Ferley *et al.* [2] propose a system for resolution adaptive volume sculpting. However, crack patching is performed to fill cracks where two cells of different resolutions meet [14]. Heidrich *et al.* also propose a real-time adaptive isosurfacing method [5]. In addition to adaptive resolution, the original marching cubes algorithm does not represent sharp features well. By using extra information of normals, Kobbelt *et al.* [8] propose the extended marching cubes (EMC) algorithm which preserves sharp features. Perng *et al.* [12] propose a volume sculpting system using EMC in uni-resolution.

In addition to marching cubes based methods, Frisken *et al.* [3] propose a sculpting system with adaptively sampled distance fields (ADFs) to preserve sharp features with an efficient refinement, but this was limited to local modification. Perry *et al.* [13] propose a system for sculpting digital characters based on ADFs. Pointshop 3D [15] is able to perform modeling, which is limited to normal displacement, for point-based geometry. Adams *et al.* [1] propose methods for Boolean operations which apply to point-based geometry. Ohtake *et al.* [11] use a partition-of-unity method to construct models from sets of points.

Although many of these contributions deal with volume sculpting, these works have concentrated on part of the problems for volume sculpting. Recently, Ho *et al.* [6] propose a cubical marching squares algorithm (CMS) to deal with preserving sharp features, inter-cell independency, maintaining consistent topology, and adaptive resolution. We obtain these benefits from incorporating with CMS to archive highly detailed models for sculpting.

3. Storing Volume Data

To represent volume data as precise as possible, we choose a mixed form representation instead of a traditional used scalar field. We use arrays of rays to store samples of a volume data. Each sample is a point located at the surface of a volume data and it stores the following information: the position of itself and the normal direction at this position. Each ray contains a list of samples. First, considering one-dimensional cases as shown in Fig. 5, a ray contains pairs of sample points to represent regions. A pair of samples defines a region, the first sample marks the left boundary and the second sample marks the right boundary. We define a sample as a *left bounding sample* if it marks the left boundary of a region. Also, we define a sample as a *right bounding sample* if it marks the right boundary. Thus, the ray itself stores data in vector form.

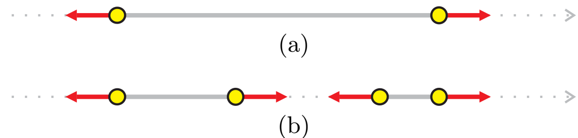


Figure 5: The data structure of a ray in 1D, where yellow points represent sample points, red arrows represent normal directions of sample points, and solid lines represent regions. (a) A ray intercepting a region. (b) A ray intercepting two regions.

Second, considering a two-dimensional case as shown in Fig. 6, an array of rays consist pieces of one-dimensional regions. These one-dimensional regions discretely represent the shape of the gray solid triangle in Fig. 6.

Third, to represent a three-dimensional volume data with rays discretely, we use a two dimensional array of rays to store each pieces of the data as one-dimensional regions. Thus, this representation is actually an image with each pixel stores a list of samples. In compare to scalar field, this representation requires much smaller space to store data and preserves all the detail of the volume data. In addition, we can store exact normal direction in this representation to meet the requirement of sharp feature preserving techniques.

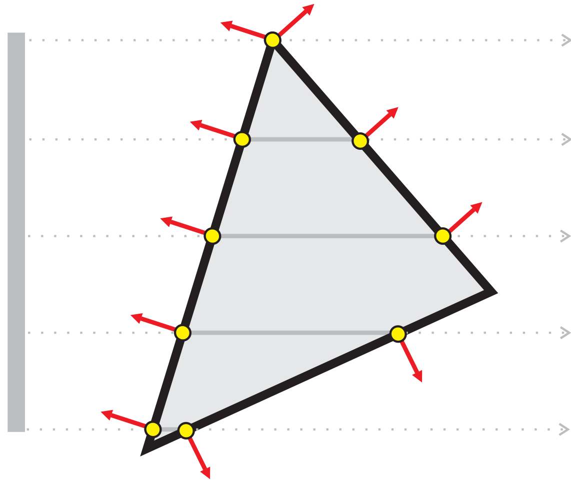


Figure 6: A 2D analogy of our data structures.

Geometric representation, such as a polygonal mesh, an implicit surface, or a set of point clouds, could easily converts into our volumetric representation. Since our goal is to represent the volume data as precisely as possible, the Hermite data is acquired at a very fine resolution, say, a uniform $n_k \times n_k \times n_k$ grid. For these geometric representations, we shoot an array of axis-aligned rays, along one of x, y, and z axes. For each ray, we store all the intersection points of this ray and the volume data, and their normals. We refer to the intersection points as sample points and their normals as sample normals. The resolution of the array is $(n_k + 1) \times (n_k + 1)$, where n_k is the finest resolution. A straightforward method to generate an array of rays with intersections is to use a modified ray tracing algorithm. However, since all rays are axis-aligned, we found that it is faster to scan convert the geometry using the orthographic projection and a Z-buffer with a Z-list for each entry. Thus, we call this representation as *Z-list-buffer.*

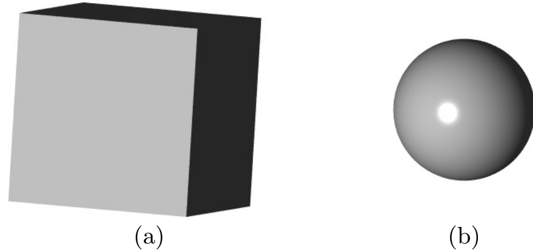
Finally, to visualize volume data using cubical marching squares algorithm, we use three arrays of rays to store samples. Each array contains rays along an axis, say, x-axis, y-axis, and z-axis. Using three axis-aligned arrays of rays is convenient for querying intersections along each axis for the cubical marching squares algorithm.

In our experiences, the average number of samples in a ray is approximately equal to 2.86 for a dragon model, and 2.05 samples for a fandisk model. Hence, the resulting performance should be better than performing Boolean operations in scalar fields. For instance, a $512 \times 512 \times 512$ distance field requires 512 samples in a ray. This can be reduced to approximately 2 samples using our ray structure.

4. Boolean Operations

Sculpting in virtual world is the same as performing continuous subtraction operations to a target object. To mimic the sculpting behavior, we store the tools and the target objects in the Z-list-buffers format. Then, we perform Boolean operations on objects in Z-list-buffers format. Because we use array of rays to store volume data, Boolean operations could perform in one dimension, which is much simpler than perform these operations in higher dimension. In this section, we describe algorithms to perform Boolean operations of two rays.

First, union operations in sculpting application add materials to a target object. Fig. 7 shows two objects, a cube and a sphere, for performing the following Boolean operations. Fig. 8 shows an example of uniting these two objects.



(a) (b)

Figure 7: Two objects for performing Boolean operations. (a) A cube. (b) A sphere.

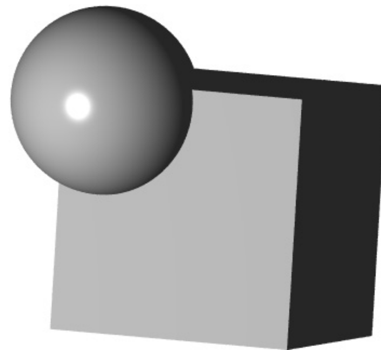


Figure 8: The resulting object after performing a union operation of objects in Fig. 7.

We use the procedure RayUnion in Algorithm 1 to perform union operation of two rays. In this procedure, we merge all samples in the list of second ray into the list of first ray and we sort the resulting list according to the z-value of each sample. Then, we remove regions which are inside another region by the procedure MergeRegion in Algorithm 2. The resulting samples in the list of first ray are the result after performing union operation.

Algorithm 1 RayUnion. *This procedure performs a union operation of two rays.*

```
1: procedure RayUnion(Ray A, Ray B)
2:     for each sample s in B.list
3:         A.list ← A.list ∪ s;
4:     end for
5:     Sort(A.list);
6:     MergeRegion(A);
7: end procedure
```

Second, subtraction operations in sculpting application remove materials from a target object. Fig. 9 shows an example of subtracting a sphere from a cube.

Algorithm 2 MergeRegion. *This procedure performs an operation to remove inner regions inside a region.*

```
 1: procedure MERGEREGION(Ray R)
 2:     level ← 0;
 3:     T ← ∅;
 4:     for each sample s in R.list
 5:         if DOT(s.normal, R.direction) < 0 then
 6:             level ← level − 1;
 7:             if level == 0 then
 8:                 T ← T ∪ s;
 9:             end if
10:         else
11:             if level == 0 then
12:                 T ← T ∪ s;
13:             end if
14:             level ← level + 1;
15:         end if
16:     end forR.list ← T;
17: end procedure
```

Algorithm 3 RaySubstraction. *This procedure performs a subtraction operation of two rays.*

```
1: procedure RAYSUBSTRACTION(Ray A, Ray B)
2:     for each sample s in B.list
3:         s.normal ← −s.normal;
4:         A.list ← A.list ∪ s;
5:     end for
6:     SORT(A.list);
7:     MERGEREGION(A);
8: end procedure
```

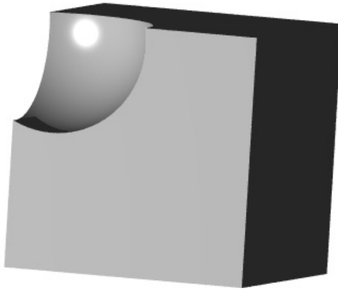


Figure 9: The resulting object after performing a subtraction operation of objects in Fig. 7.

Figure 10: The resulting object after performing an intersection operation of objects in Fig. 7.

We use the procedure RaySubtraction in Algorithm 3 to perform subtraction operation of two rays. This procedure is similar to the procedure RayUnion in Algorithm 1. The only difference is the normal direction of each sample is inverted before we merge the sample into the list of first ray. After that, we sort the resulting list according to the z-value of each sample. Then, we remove regions which are inside another region by the procedure MergeRegion in Algorithm 2. The resulting samples in the list of first ray are the result after performing subtraction operation.

Finally, there are other Boolean operations, such as intersection and complement, could perform in similar steps. Fig. 10 shows an example of intersection of a sphere and a cube. The intersection operation could be achieved by replacing the code "level==0" to "level==1" at line 6 and line 8 in Algorithm 2.

5. Rendering Volume Data

To visualize volume data, we use cubical marching squares to extract the surface of the volume data. We adaptively partition the space in a binary fashion. The space is subdivided by a plane in the following sequence: X-Y plane, Y-Z plane, and X-Z plane, then circulate back to X-Y plane and so on. The subdivision is processed in a top-down manner. We start from a very coarse uniform $n_0 \times n_0 \times n_0$ base grid, in our implementation, $n_0 = 8$. Then, we exam each cell to decide should we going further or not. We continuous subdivide a cell if it has the tendency to contain a complicated surface, or it has an ambiguity. We detect the tendency by a heuristic, checking whether the maximal spanning angle of all pairs of sample normals inside this cell exceeds a predefined angle threshold. When this happens, it means that the surface inside a cell might not be flat enough and should be subdivided.

The extracting process requires to determine the following information: a vertex is inside or outside the volume, and which intersection samples are on an edge. For each vertex, we use the procedure IsInside in Algorithm 4 to check a vertex is inside or outside the volume. Basically, a vertex is inside if it is in the middle of a pair of neighboring samples which the first sample is a left bounding sample and the other is a right bounding sample. For each edge, we use the procedure GetSample in Algorithm 5 to retrieve the samples which are on an edge. Because all samples on a ray are sorted in their z-order, retrieving samples between two vertices of an edge is straight forward. First, we calculate the z-values of these two vertices regarding to the origin and the direction of the ray. Then, we get samples which are between these z-values.

6. Results

To render the same volume data, in our experiences, we observed the distance error of the results using cubical marching squares are about one-third of the distance error using the other algorithms. Thus, we think the data structures

Algorithm 4 IsInside. *This procedure checks a vertex is inside or outside.*

1: **procedure** IsInside(Ray R, Vertex v)
2: $dist \leftarrow$ Dist$(v, R.origin)$; ▷ signed dist. between 2 points
3: **find** sample $s \in R.list$ **where**
4: Dist$(s.position, R.origin) \leq dist$
5: Dist$(s.position, R.origin)$ is maximum
6: **end find**
7: **if** Dot$(s.normal, R.direction) < 0$ **then**
8: **return** False;
9: **else**
10: **return** True;
11: **end if**
12: **end procedure**

Algorithm 5 GetSample. *This procedure retrieves the samples which are on an edge.*

1: **procedure** GetSample(Ray R, Vertex v_1, Vertex v_2)
2: $S \leftarrow \emptyset$;
3: $dist_1 \leftarrow$ Dist$(v_1, R.origin)$;
4: $dist_2 \leftarrow$ Dist$(v_2, R.origin)$;
5: **for each** sample $s \in B.list$
6: $dist \leftarrow$ Dist$(s.position, R.origin)$;
7: **if** $dist_1 \leq dist \leq dist_2$ **then**
8: $S \leftarrow S \cup s$;
9: **end if**
10: **end for**
11: **return** S;
12: **end procedure**

and the methods discussed in this paper can provide better visual quality for a sculpting application. To compare these method in visual, we demonstrate some results of our sculpting experiences.

To show the effectiveness of preserving sharp features, we perform several operations and compare the resulting shapes of marching cubes and cubical marching squares. First, we subtract a three dimensional shape of the text "CMS" from a cube. Then, we subtract a sphere from the previous result. Fig. 11(a) shows the resulting shape of the previously described operations extracted using marching cubes algorithm. Fig. 11(b) shows the result using cubical marching squares. We can see the sharp features are well preserved by comparing these two figures.

To show the effectiveness of preserving topology, we create a thin region by subtracting several spheres from a cube, two of these spheres are very close to each other. Fig. 12(a) shows the front view of the result. We observed there is a thin region at the left-top part of the figure. Fig. 12(b) shows the close-up view of the result extracted using dual contouring. We observed some defects cause by extracting wrong topologies in this figure. Fig. 12(c) shows the close-up view of the result extracted using cubical marching squares, the resulting shape is well preserved by determining the topologies inside cubes.

To show crack-free feature in adaptive resolution, we perform the following task. We create cells of a sphere using previously described methods. Then, we merge several cells

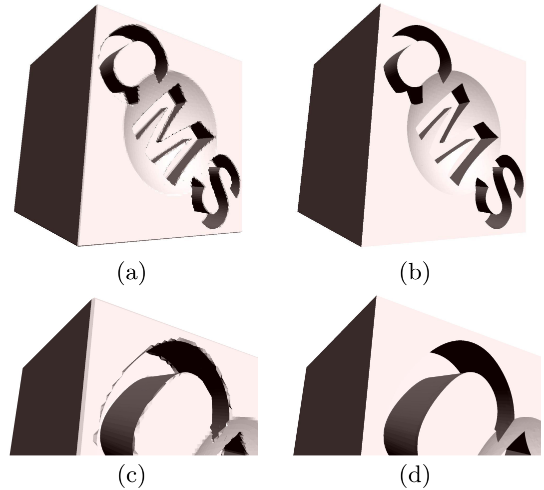


Figure 11: A cube subtracted by the text "CMS" and a sphere. (a) the resulting shape extracted using marching cubes. (b) the resulting shape extracted using cubical marching squares. (c) the close-up view of (a). (d) the close-up view of (b).

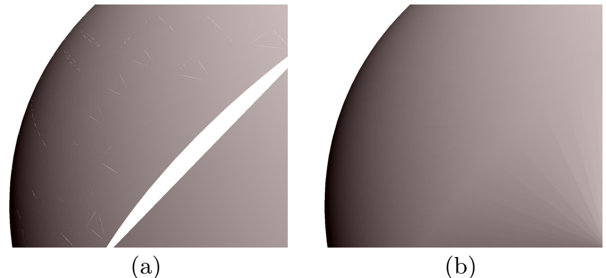
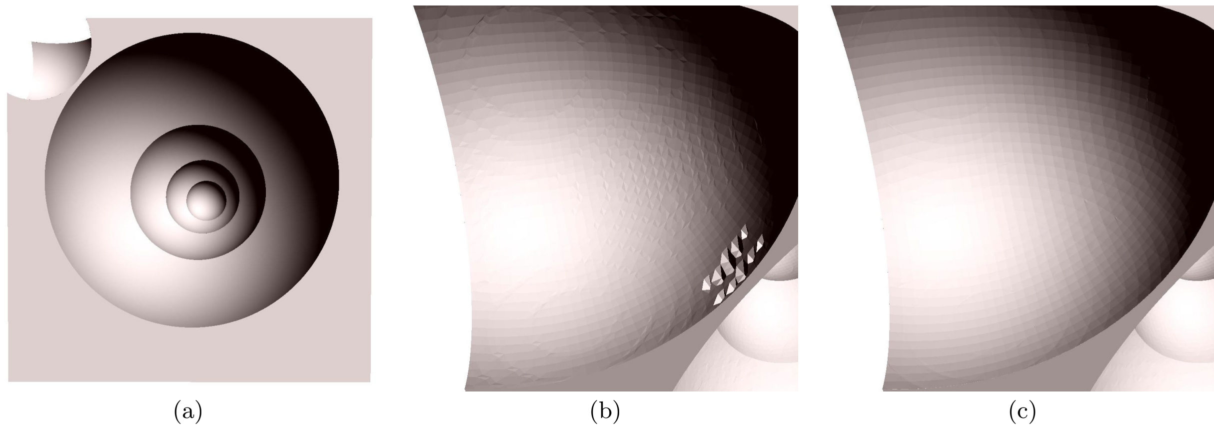


Figure 13: A result of sphere and part of the cells are merged. (a) the result of marching cubes, the white regions are the cracks between different resolutions. (b) the result of cubical marching squares.

into a bigger cell. The merge operation makes a larger depth difference between this cell and its neighboring cells. Fig. 13(a) shows the result of marching cubes, where several cracks can be observed in this figure. Fig. 13(b) shows the result of cubical marching squares. This result shows cracks are eliminated.

7. Conclusion and future work

In this paper, we propose a virtual sculpting application using the Z-list-buffers and the cubical marching squares algorithm for volume sculpting. The resulting shapes of sculpting have better visual quality due to the following facts hold: (1) sharp features are well preserved; (2) topological ambiguity can be determined and solved by the detected sharp features; (3) the problem of cracks between adjacent cells when using a multiresolution representation for the data is solved. Also, the performance of sculpting reaches at a interactive speed due to the following features: (1) 3D features can be reconstructed starting from the 2D features located on the faces of the cells; this avoids intercell dependencies; hence, it has potential to perform in parallel;



(a) (b) (c)

Figure 12: A cube subtracted by five spheres, a thin region is created at the left-top portion. (a) The front view of the result. (b) The close-up view of the result in flat shading using dual contouring. (c) The close-up view of the result in flat shading using cubical marching squares.

(2) the ray structure reduces the need of a linear sequence of samples into a fewer number of samples; this makes a faster computation and a smaller storage. These features make our method quite simple, relatively easy to implement and, at the same time, effective. Currently, we have implemented part of our algorithm on a GPU. We observed a 8-to-16-time speedup in computation using a GPU. However, the resulting speed is only comparable to our CPU implementation. As many other GPU algorithms, the bottleneck is the data transfer between CPU and GPU. In the future, we plan to fully implement our algorithm on a GPU to decrease the requirement of bus bandwidth. We believe that our algorithm will benefit from the improvement on the bus bandwidth between CPU and GPU.

References

[1] Bart Adams and Philip Dutré. Interactive boolean operations on surfel-bounded solids. *ACM Trans. Graph.*, 22(3):651–656, 2003. 2

[2] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. Resolution adaptive volume sculpting. *Graphical Models*, 63(6):459–478, 2001. 1, 2

[3] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH*, pages 249–254, 2000. 2

[4] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Proceedings of ACM SIGGRAPH*, pages 267–274, 1991. 2

[5] W. Heidrich, R. Westermannand H-P. Seidel, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, 1999. 2

[6] Chien-Chang Ho, Fu-Che Wu, Bing-Yu Chen, Yung-Yu Chuang, and Ming Ouhyoung. Cubical marching squares: Adaptive feature preserving surface extraction from volume data. *Computer Graphics Forum*, 24 (2), 2005. 1, 2

[7] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of ACM SIGGRAPH*, pages 339–346, 2002. 1

[8] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH*, pages 57–66, 2001. 1, 2

[9] Frederick W. B. Li, Rynson W. H. Lau, and Frederick F. C. Ng. Collaborative distributed virtual sculpting. In *Proceedings of Virtual Reality*, page 217, 2001. 1

[10] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of ACM SIGGRAPH*, pages 163–169, 1987. 1, 2

[11] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003. 2

[12] K.-L. Perng, W.-T. Wang, M. Flanagan, and M. Ouhyoung. A real-time 3d virtual sculpting tool based on modified marching cubes. In *Proceedings of International Conference on Artificial Reality and Tele-existence*, pages 64–72, 2001. 1, 2

[13] Ronald N. Perry and Sarah F. Frisken. Kizamu: a system for sculpting digital characters. In *Profroceedings of ACM SIGGRAPH*, pages 47–56, 2001. 1, 2

[14] Raj Shekhar, Elias Fayyad, Roni Yagel, and J. Fredrick Cornhill. Octree-based decimation of marching cubes surfaces. In *Proc. of IEEE Visualization*, pages 335–342, 1996. ISBN 0-89791-864-9. 1, 2

[15] Matthias Zwicker, Mark Pauly, Oliver Knoll, and Markus Gross. Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of ACM SIGGRAPH*, pages 322–329, 2002. 1, 2