The Voxel-Sweep: A Boundary-Based Algorithm for Object Segmentation and Connected-Components Detection

Itay Cohen, Dan Gordon

Dept. of Computer Science
University of Haifa
Haifa 31905, Israel
Email: {icohen26, gordon}@cs.haifa.ac.il

Abstract

A new approach to the problem of object segmentation and isosurface detection is introduced. Drawing on the fundamental marching cubes algorithm of Lorensen and Cline, it operates with one space sweep through the voxels (hence it is called the *voxel-sweep*). It detects all isosurfaces and partitions the objects into connected components on the basis of the continuity of the surfaces. The surface of each connected component is obtained as an oriented triangular mesh, and is thus amenable as input to mesh-processing programs and hardware. Each separate component is automatically associated with its topological holes. These properties enable accurate volume and surface-area estimation of each component, as well as noise reduction (by eliminating "small" components). The runtime of the voxel-sweep is just a negligible increase over the runtime of the marching cubes algorithm. Another option of the voxel-sweep is to visualize the resulting surfaces at the same time as they are being formed. Ideally, this option should enable real-time modification of the iso-values defining the surfaces.

1 Introduction

The issue of volume visualization is of prime importance in many industrial, scientific and medical applications. The raw data is usually available as a 3D matrix of numbers which are the discrete values of a *density* function $F(x,y,z)$, representing some physical property. For example, in computerized tomography, the property is that of X-ray attenuation; different types of objects within a body attenuate X-rays differently. In such an application, a user wishes to specify threshold values for some type of material (e.g., bone) and then visualize the shape of the bone(s).

Another important objective is that of *object segmentation*, i.e., the user wishes to distinguish between disconnected parts of the same type (the different parts will have similar function values). It would be most convenient if disconnected pieces could be isolated and displayed separately or in any combination of parts. Even if different parts are displayed together, better visual cues are obtained if the separate pieces are assigned different colors. Other useful objectives include interactive manipulation of the visible objects, and estimation of volume and surface areas.

There exist two fundamentally different approaches to visualizing volumetric data: Volume-based display, and surface-based display. Volume-based methods operate on the entire set of voxels and display the objects directly according to some method – see [5, 8, 7, 9, 13, 14, 15, 18, 20, 24, 28]. These methods have a disadvantage of requiring the entire dataset in some suitable format, but, on the other hand, the original data is available at all times for various purposes.

Surface-based methods first extract the surfaces in a preprocessing step, and then display the surfaces [2, 17]. There are advantages and disadvantages of surface-based methods as compared to volume-based methods – see [4]. However, the amount of surface data is proportional to the surface of the objects while the volume data is proportional to its volume, so surface data usually takes up less space than volume data and it is more easily manipulated and displayed. Furthermore, surface data are more suitable for most of today's graphics engines, which are geared towards polygonal meshes.

The "boundary-detection" (BD) algorithm of

Artzy et al. [2] treats the voxels as cubes with a uniform value inside. Starting from a seed, which is a user-specified boundary face, it detects the visible surface connected to the seed. BD provides the volume enclosed by the surface, but that volume may include internal holes. Different objects, as well as holes, require new seeds. Since the polygons produced by BD are faces of cubes, the display occasionally seems jagged. BD was speeded up by altering the definition of "adjacent boundary faces" [11], but it contained basically the same limitations.

The "Marching Cubes" (MC) algorithm of Lorensen and Cline [17] considers the voxel values as being assigned to grid points in 3D space, and assumes that at any other point (x,y,z), the value of $F(x,y,z)$ is a trilinear interpolation of the grid points surrounding (x,y,z). Given a certain threshold (or isovalue), MC sweeps once through all the volumetric dataset and outputs all polygons defined by the isovalue. Thus, MC produces not just one connected object but all the objects. Much research has also been done on speeding up the Marching Cubes algorithm through the employment of various data structures – see, for example, [10, 27, 12, 23, 22, 3, 16, 6, 25]

Separating the data into connected components is done by volume-based methods – see [21]. Starting from a user-specified seed voxel, all voxels that are connected to it are identified by some classical search method such as breadth-first or depth-first search. Such methods require a new seed for every object. Another problem is that the result is again volumetric, thus requiring large storage space. Also, such data needs volume-based methods for its display, or a surface-extraction step to isolate the surface.

In this paper, we introduce a new method, called the *Voxel Sweep* – or VS for short – with the following properties:

- Similarly to MC, VS operates in just one space sweep through the volume data.
- VS detects all surfaces of all the objects defined by the isovalue, including internal holes.
- All the objects are automatically separated into connected components.
- Internal holes are associated with their respective surrounding objects.
- The runtime of VS is only about 2% more than that of MC.

The above properties enable us to calculate the surface area and volume of each component taking the holes into consideration. Noise can be eliminated by avoiding the display of "small" components. The mesh format of the components is useful both for modern display architecture and for state-of-the-art compression methods – see [26]. Work on speeding up the performance of VS is currently in progress.

The rest of the paper is organized as follows. Section 2 explains the Voxel Sweep in terms of the Marching Cubes. Section 3 presents the results of several test cases, and Section 4 concludes with a discussion and potential extensions.

2 From Marching Cubes to the Voxel Sweep

Using the terminology of the Marching Cubes algorithm (MC) [17], we henceforth use the term "voxels" to refer to the grid points with which the discrete values of the data are associated. MC creates a representation, consisting of triangles, of an isovalue surface. It uses a divide and conquer approach to locate the surface in a logical cube created from eight voxels; four each from two adjacent slices. The algorithm determines how the surface intersects this cube, and then "marches" to the next cube in scanline order. To find the surface intersection in a cube, MC assigns the value '1' to a cube's vertex if the data value at that vertex exceeds or equals the given isovalue, and the value of '0' to cube vertices with values below the isovalue. The surface intersects those cube edges where one vertex is outside the surface (1) and the other is inside the surface (0). MC thus determines the topology of the surface within a cube, and by using linear interpolation between voxel values computes the location of the triangles vertices. The result of considering all subcubes in this way is a collection of triangles which approximates the location of an isosurface. In considering the eight voxels of a subcube, there are 256 possible combinations of voxels either being inside or outside the isosurface. MC takes advantage of rotation and reflection to reduce the number of unique cases from 256 to 15. The original MC algorithm did not resolve ambiguous cases, resulting in occasional holes in the surface representation. Ambiguous cases can be resolved using the gradient of voxel values – see [19].

The difference between MC and VS is that VS

uses a union-find (aka "merge-find") data structure [1] to keep the connected triangles together in one mesh. The union-find data structure is applied in the following type of situation: We start out with n objects which are initially placed in n different sets. At all times, the sets are disjoint. After that, we process $O(n)$ operations of the following type:

- Find(a) – returns the (unique) set to which a belongs.
- Union(A,B) – merges the two sets A and B.

It is well-known that with this data structure one can perform a sequence of $O(n)$ union and find operations on n objects in amortized time of $O(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. This function grows so slowly that for all practical applications, $\alpha(n) \leq 4$ [1, p. 189].

In broad terms, the union-find method is applied in VS as follows: Each part (or segment) of a boundary that is considered as an object, and entire closed surfaces will be the sets. Initially, each new boundary segment that is encountered is placed in a set of its own. As the sweep progresses, we encounter cases where two separate segments join together. We apply the Find operator to each segment to get the two sets containing the segments. We then apply the Union operator to the two sets. At the end of the sweep, all connected segments are in one unique set.

2.1 The Voxel Sweep

2.1.1 2D Analogy of the Voxel Sweep

In order to explain the Voxel Sweep, we first present a 2D analogy. Figure 1 shows a grid of points at which the values of the function $F(x,y)$ are given. We assume that all the values at the extremal grid points are zero, and that the values inside define the three boundaries shown in the figure. As in the Marching Cubes algorithm, the vertices of the boundaries lie between two grid points whose values span the isovalue; the exact position of each boundary vertex is a linear interpolation of the values at the grid points.

We assume that the horizontal sweep direction goes upwards and at each horizontal front, the grid is swept from left to right. As the sweep proceeds upwards, new boundary segments are encountered. In Figure 1, vertices marked with a "1" denote the start of new boundary segments. Initially, three segments, starting from vertices of type "1", are associated with different surfaces, but eventually, the sweep encounters a vertex marked "2" and the two rightmost segments unite (or merge). This is where the union-find data structure is used.

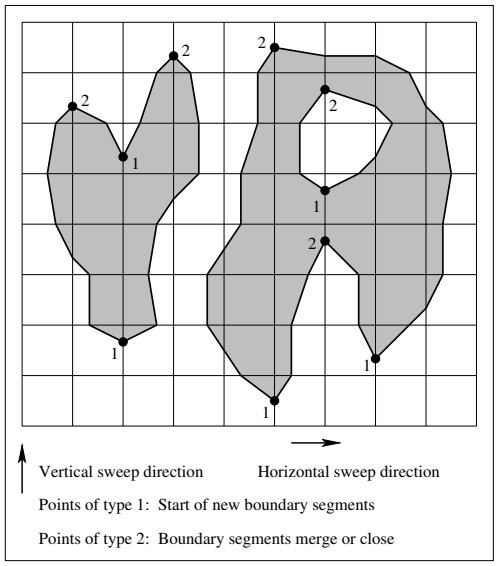

Figure 1: 2D analogy of the Voxel Sweep.

2.2 The 3D Voxel Sweep

The Voxel-Sweep algorithm creates a representation of an isosurface from volume data, as in the marching cubes. The algorithm considers subcubes of eight voxels at a time, at each cube the algorithm constructs the triangles that represent the isosurface (if the isosurface intersect that cube). Then, similarly to MC, the Voxel Sweep "marches" to the next cube in scanline order. In general, the sweep direction proceeds through slices of the dataset, and, in each slice, it proceeds as in the 2D analogy.

The root of each tree in the union-find data structure represents a different connected object. To use this data structure, each triangle vertex needs to be calculated only once. To achieve this, VS saves the previous cube line and slice intersections. Each new vertex receives a value that represents its parent in the union-find data structure. By using the Find operation on that value, the algorithm receives the value of the object that this vertex belongs to.

As the sweep advances, new triangles are detected, and each new triangle is constructed from

three vertices. These vertices can be new or old, and they can belong to the same object or to a different one. The characteristics of the three vertices determine whether the new triangle is the start of a new object, or needs to be added to an existing object, or whether it unites two objects that until this point were considered as separate. It is also possible to detect a new object that belongs to an existing outer surface; it may later turn out to be an inner surface (a hole), or it can later unite with the outer surface.

In the next subsections, which describe the different cases, we will refer to the new triangle vertices as a, b, c, as shown in Figure 2.

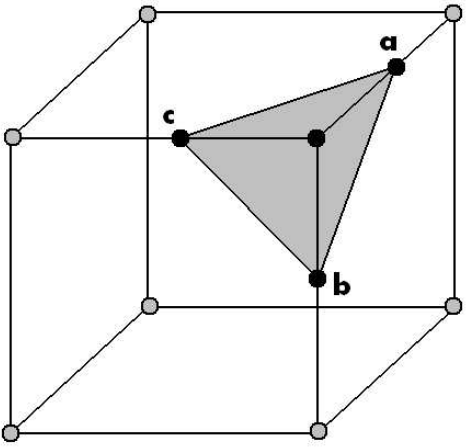

Figure 2: A new triangle and its vertices in a cube.

2.2.1 Case 1

All the vertices a, b, c are new. This case describes the detection of a new surface. As shown in Figure 3, the new triangle can only be in the far corner of the cube (in the scan direction). Because those are the only new vertices that are calculated in this cube; all the other vertices (if existing) have already being calculated in previous cubes. This observation is important to the understanding of the detection of holes (inner surfaces), that will be described later.

The operations that needs to be taken in this case are: add a new object to the union-find data structure (and place it in a new set) and then assign the new object value to the new vertices.

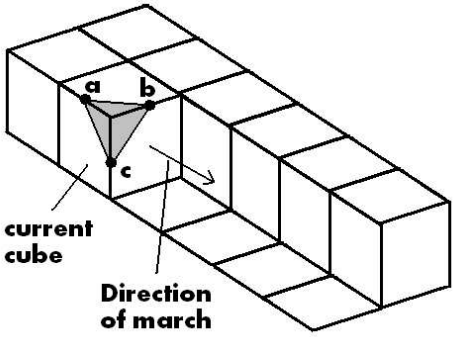

Figure 3: Case 1: The new triangle is in the far corner of the current cube.

2.2.2 Case 2

As described in the example of Figure 4, in this case at least one vertex is old, but all of the old vertices have the same object value. This case describes a new triangle that needs to be added to an existing object. If there are new vertices in the triangle, we also need to assign to them the existing object value.

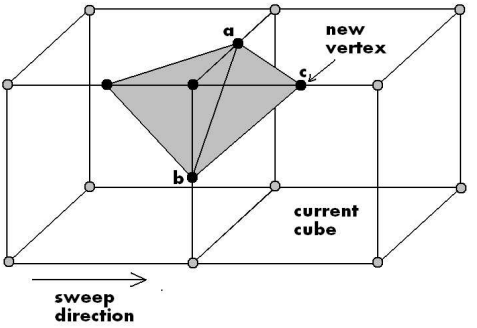

Figure 4: Case 2: Vertices a and b were calculated previously and belong to the same object. c is new and is now assigned to that object.

2.2.3 Case 3

As described in the 2D example of Figure 5, in this case at least two old vertices belong to different objects. This means that there are two (or more) old vertices in the new triangle and not all of them belong to the same object. This case describes the situation where two different objects are united by the

new triangle that connects them. In this case we perform the Find operation on the two objects and then the Union operation on the sets which contain them.

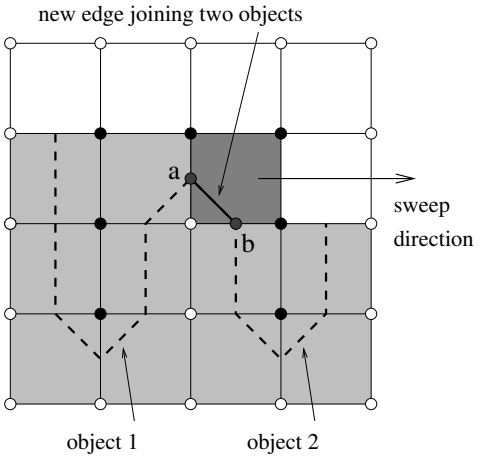

Figure 5: A 2D example of Case 3: Vertices a, b were calculated previously, but they belong to different objects (1 and 2), which now unite.

2.2.4 Detection and Association of Holes

As shown in Case 1, detection of a new object will always be at the far corner in the scan direction (Figure 3). This property makes it easy to determine if a new object is an inner surface or an outer surface: If the value at the corner is higher than the isovalue, the new object is an outer surface and needs to be assigned a new object number. If the corner value is lower than the isovalue, the new object is an inner surface of an existing object and need to be associated with it.

To understand how an inner surface is associated with its outer surface, we need to understand another property. As shown in Figure 6, if we shoot a ray from the outside and it intersects an inner surface, then it must have already intersected its outer surface, and the last ray-surface intersection before the current one was the intersection with this outer surface. As we sweep through a row of cubes, we keep the last outer surface that we intersect in the line of the far corners. When we intersect a new inner surface we associate it with the last outer surface that we found. In principle, this idea could also be extended to enable association of an object with its nested objects (if any).

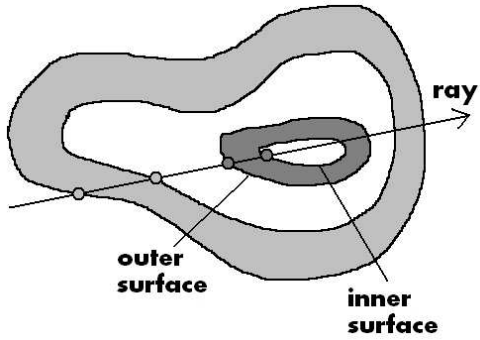

Figure 6: Ray intersection with inner end outer surfaces.

3 Results

The Voxel Sweep and the Marching Cubes were implemented in C++ in the .NET environment. The program was run on an Intel Pentium 4 with 1GB of memory and clock speed of 2.67 MHz, with an Nvidia G4-Ti4600 graphics card. The results were displayed with OpenGL, using the hardware Z-buffer, with Gouraud shading.

Figure 7 shows the result of the VS algorithm on a CT scan of a human head (CThead). The separate components are shown in different colors – we can see that the skull, the lower jaw and the upper jaw are all separated. In figure 8 (Chain) we can see three non-intersecting, linked toruses. Each torus was identified as a separate component and assigned a different color. Figure 4 describe three steps of VS on the "Tree" example. Initially, all the "roots" are shown in different colors since they are in different sets. As the sweep proceeds upwards, some sets combine and the color of the roots also combine. Figure 10 is the result of VS on the Boston-Teapot example with a cut plane. Inside the teapot we can see a lobster whose claws are separated.

Table 1 shows the run-times of MC and VS (in seconds) on the above datasets. Also shown in the table are some of the parameters of the datasets. It can be seen that VS requires, on average, only about 2% more time than MC.

Dataset	Size	Triangles	MC time	VS time
Chain	$64 \times 64 \times 64$	10,560	0.109	0.11
Tree	$64 \times 64 \times 64$	151,824	0.836	0.88
CThead	$256 \times 256 \times 113$	464,414	4.49	4.58
Teapot	$178 \times 256 \times 256$	543,524	8.34	8.36

Table 1: Runtimes (in sec.) of Marching Cubes and Voxel Sweep

4 Conclusions

The Voxel Sweep extends the Marching Cubes by separating the objects into connected components and associating holes with their surrounding objects. Besides the obvious benefits for visualization, this feature enables logically correct estimations of volume size and surface area of the different components. The additional required runtime is negligible.

Another option is that the objects can be visualized at the same time as the VS operates. This option can be utilized to enable interactive modification of the isovalues defining the surfaces.

Future research will concentrate in the following directions:

- Speeding up the Voxel-Sweep by the use of applicable data structures.
- Implement more user-oriented capabilities, such as cut planes.
- Incorporate several isovalues into the Voxel Sweep, as well as automatic surface determination defined by "large jumps" in the values between adjacent grid points (such jumps correspond to discontinuities in the function values and represent object boundaries).
- Extend the Voxel Sweep so that it operates on *voxel* connectivity instead of surface connectivity. Different types of objects could be detected in one sweep, and the union-find method will enable the separation of all object types into their respective connected components.

Acknowledgments. This research is part of the first author's M.Sc. thesis in Computer Science carried out under the second author's supervision at the University of Haifa. Research supported by grant no. 01-01-01509 from the Israeli Ministry of Science. The CThead data was taken on a General Electric CT Scanner and provided courtesy of North Carolina Memorial Hospital. The Boston teapot data is due to Terarecon Inc., MERL, Brigham and Women's Hospital.

References

[1] A. Aho, J.E.Hopcroft, and J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, Reading, Mass., 1983.

[2] E. Artzy, G. Frieder, and G. Herman. The theory, design, implementation, and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15:1–24, 1981.

[3] C. L. Bajaj, V. Pascucci, and D. Schikore. Fast isocontouring for improved visibility. In *VVS '96: Proceeding of the 1996 Symposium on Volume Visualization*, San Francisco, October 1996.

[4] D. Bartz and M. Meissner. Voxels versus polygons: A comparative approach for volume graphics. In *Volume Graphics '99*, International Workshop Proceedings, Swansea, U.K., May 1999.

[5] W. Cai and G. Sakas. Maximum intensity projection using splatting in sheared object space. *Computer Graphics Forum*, 17(3):113–124, 1998.

[6] P. Cignoni, P. Marino, C. Montani, E. PUPPO, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.

[7] D. Cohen-Or and S. Fleishman. An incremental alignment algorithm for parallel volume rendering. *Computer Graphics Forum*, 14(3):123–133, 1995.

[8] D. Cohen-Or and Z. Sheffer. Proximity clouds – an acceleration technique for 3D grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[9] G. Frieder, D. Gordon, and R. Reynolds. Back-to-front display of voxel-based objects. *IEEE Computer Graphics & Applications*, 5(1):52–60, January 1985.

[10] M. Giles and R. Haimes. Advanced interactive visualization for cfd. *Computing Systems in Education*, 1(1):51–62, 1990.

[11] D. Gordon and J. Udupa. Fast surface tracking in three-dimensional binary images. *Computer Vision, Graphics, and Image Processing*, 45:196–214, 1989.

[12] T. Itoh and K. Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, December 1995.

[13] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Computer Graphics Proceedings*, Annual Conference, pages 451–458. ACM SIGGRAPH, August 1994.

[14] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[15] M. Levoy. Efficient ray tracing of volume data. *ACM Trans. on Graphics*, 9(3):245–261, July 1990.

[16] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[17] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21 (ACM SIGGRAPH Annual Conf. Proc.)(4):163–169, July 1987.

[18] D. Meagher. Interactive solids processing for medical analysis and planning. In *Proc. National Computer Graphics Assoc.*, pages 96–106, 1984.

[19] Nielson.G.M. and B.Hamann. The asymptotic decider: Resolving the ambiguity in marching cubes. *Proceedings of the IEEE Visualization '91 Conference*, pages 83–91, October 1991.

[20] R. Reynolds, D. Gordon, and L.-S. Chen. A dynamic screen technique for shaded graphics display of slice-represented objects. *Computer Vision, Graphics, and Image Processing*, 38(3):275–298, June 1987.

[21] T. Schiemann, J. Nuthmann, U. Tiede, and K. Höhne. Segmentation of the visible human for high quality volume based visualization. In K. Höhne and R. Kikinis, editors, *Visualization in Biomedical Computing: 4th. Intern'l Conf. Proc. / VBC '96, Hamburg, Germany, Sept. 22-25, 1996.*, Lecture Notes in Computer Science, pages 13–22, Berlin, August 1996. Springer-Verlag.

[22] H.-W. Shen, C. D. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In R. Yagel and G. M. Nielson, editors, *IEEE Visualization '96*, Annual Conference, pages 287–294. IEEE Computer Science Press, 1996.

[23] H.-W. Shen and C. R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization '95*, Annual Conference, pages 143–150. IEEE Computer Science Press, 1995.

[24] L. Sobierajski, D. Cohen, A. Kaufman, R. Yagel, and D. Acker. A fast display method for volumetric data. *The Visual Computer*, 10(2):116–124, 1993.

[25] P. Sutton, C. Hansen, H.-W. Shen, and D. Schikore. A case study of isosurface extraction algorithm performance. In *VisSym'00: Joint Eurographics–IEEE TCVG Symposium on Visualization*, 2000.

[26] C. Touma and C. Gotsman. Triangle mesh compression. *Proceedings of Graphics Interface*, 1998.

[27] J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[28] J. Ylä-Jääski, F. Klein, and O. Kübler. Fast direct display of volume data for medical diagnosis. *CVGIP: Graphical Models and Image Processing*, 53(1):7–18, January 1991.

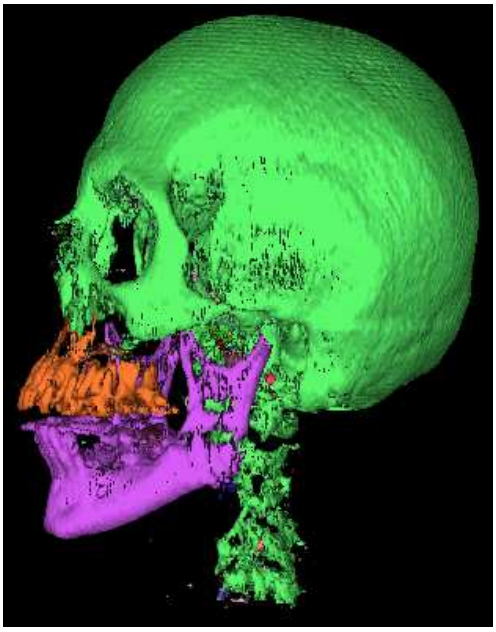

Figure 7: CThead, showing connected components.

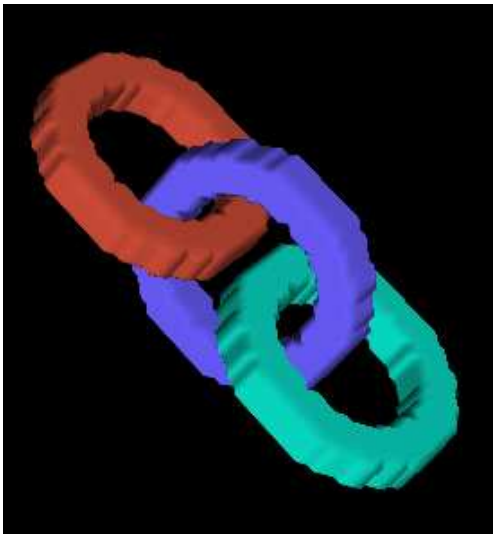

Figure 8: Chain: Three non-intersecting, linked toruses.

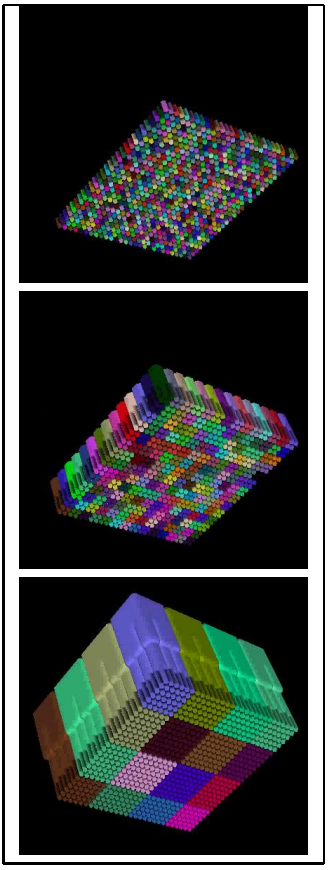

Figure 9: Three steps of the tree example, showing union of components.

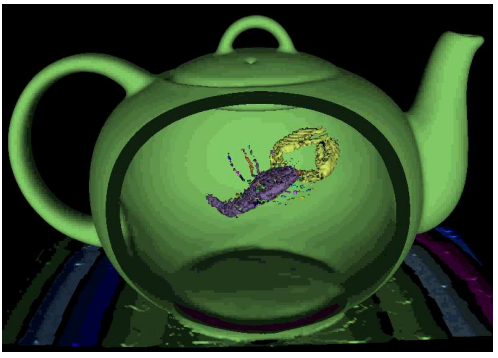

Figure 10: Boston teapot with a lobster inside.