Interactive Boolean Operations on Surfel-Bounded Solids

Bart Adams[*] Philip Dutré[*]

Department of Computer Science
Katholieke Universiteit Leuven

Abstract

In this paper we present an algorithm to perform interactive boolean operations on free-form solids bounded by surfels. We introduce a fast inside-outside test to check whether surfels lie within the bounds of another surfel-bounded solid. This enables us to add, subtract and intersect complex solids at interactive rates. Our algorithm is fast both in displaying and constructing the new geometry resulting from the boolean operation.

We present a resampling operator to solve problems resulting from sharp edges in the resulting solid. The operator resamples the surfels intersecting with the surface of the other solid. This enables us to represent the sharp edges with great detail.

We believe our algorithm to be an ideal tool for interactive editing of free-form solids.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types I.3.4 [Computer Graphics]: Graphics Utilities—Graphics editors

Keywords: free-form modeling, boolean operations, surfels, point-based geometry

1 Introduction

Constructive solid geometry (CSG) has been a useful tool in computer graphics for many years. Usually, CSG is applied to primitive objects (spheres, cylinders, cubes) to construct objects with a more complex geometric shape. However, boolean operations are also a versatile tool for editing free-form solids. Adding, subtracting and intersecting solids enables us to create more complex models. In this paper we present boolean operations as an intuitive and interactive editing tool for free-form solids bounded by surfels. Surfels, represented as oriented points in 3D space, approximate the local orientation of the surface they represent. Each surfel can be considered to represent a small area of this surface. As a consequence, when performing boolean operations on two solids *A* and *B*, most of the surfels of the surface of solid *A* are completely inside or outside solid *B* and vice versa. Only a small number of surfels intersect with the surface of the other solid.

Our algorithm works in two steps: in a first step we classify the surfels of both solids as inside, outside or intersecting with the surface of the other solid. In a second step we resample the surfels intersecting with the surface of the other solid. Our method is fast, both in displaying the boolean operations as in calculating the new geometry of the resulting solid. An example of a free-form surfel-bounded solid constructed with our algorithm is shown in figure 1.

[*]email:{barta,phil}@cs.kuleuven.ac.be

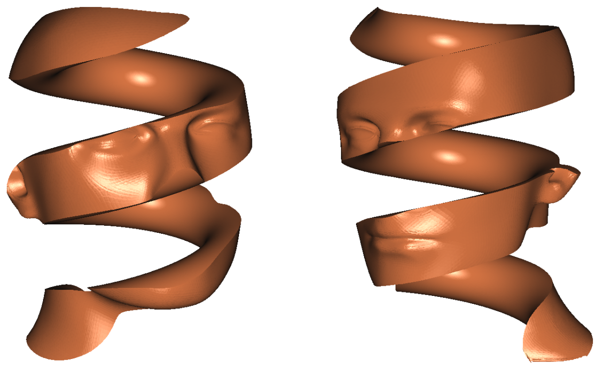
Figure 1: Two free-form surfel-bounded solids constructed using CSG (inspired by ”Bond of Union” by M.C. Escher).

This paper addresses the following important questions:

- How to test efficiently whether a surfel of one surfel-bounded solid lies inside or outside another surfel-bounded solid?
- How to represent the sharp edges typically resulting from performing boolean operations on solids, using surfels?

We start by giving a brief overview of related work in section 2. Section 3 introduces the concepts related to surfel-bounded solids. In section 4 we present a fast inside-outside test that enables us to classify the surfels of solid *A* as inside, outside or intersecting with the surface of solid *B*. In section 5 we consider the surfels intersecting with the surface of the other solid and propose the fast resampling operator. Section 6 gives implementation details and illustrates that we are able to perform boolean operations on complex solids at interactive rates. We conclude and give some topics of future research possibilities in section 7.

2 Related Work

Point-Based Geometry

The interest in using points as a display primitive in computer graphics has grown tremendously in recent years. Pfister et al. [2000] introduced the concept of surfels inspired by the work of Levoy and Whitted [1985], and more recently the work of Grossman and Dally [1998]. Significant research has been performed on efficient high quality rendering of point-based geometry. QSplat [Rusinkiewicz and Levoy 2000] uses a hierarchy of bounding spheres for progressive rendering of large models. Zwicker et al. [2001] introduce surface splatting which makes the benefits of the Elliptical Weighted Average (EWA) filter available to point-based rendering. Alexa et al. [2001] present point set surfaces and use down-sampling and up-sampling to meet the required display quality. Kalaiah and Varshney [2001]

Operation	Surface of A kept	Surface of B kept
$A \cup B$	outside B	outside A
$A \cap B$	inside B	inside A
$A - B$	outside B	inside A
$B - A$	inside B	outside A

Table 1: Part of surfaces kept when performing boolean operations.

use differential points that capture the local differential geometry in the vicinity of the sampled point. More recently Botsch et al. [2002] introduce a compact representation that uses less than two bits per point position. Cohen et al. [2001] and Chen and Nguyen [2001] introduced a hybrid system, combining polygon and point rendering. Recent approaches [Ren et al. 2002; Coconu and Hege 2002] exploit the power of current graphics hardware to render point-based geometry with high quality.

Also, work has been published on modeling and editing point-sampled geometry. Pauly and Gross [2001] introduce spectral filtering and resampling of point-based geometry. Pointshop 3D [Zwicker et al. 2002] extends 2D photo editing to 3D point clouds. They introduce a set of tools (painting, sculpting and filtering) to edit the geometry and appearance of the model. However, geometry modeling is limited to normal displacement. Pauly et al. [2002] are able to perform large model deformations on point-based geometry thanks to a dynamic resampling strategy.

Constructive Solid Geometry

Lots of research has been performed concerning constructive solid geometry. For an excellent overview we refer to [Foley et al. 1996] and [Hoffmann 1989]. Interactive rendering of CSG is often performed using graphics hardware [Goldfeather et al. 1986; Goldfeather et al. 1989; Rappoport and Spitz 1997]. Another method for CSG display is to convert the CSG structure to a boundary representation which can be rendered by all rendering systems. Interactive modification of boundary representations is often slow and difficult. Recent work however has proven that it is possible to compute the result of boolean operations on free-form solids in a reasonable amount of time. Kristjansson et al. [2001] present a framework to perform boolean operations on free-form solids bounded by multiresolution subdivision surfaces. Museth et al. [2002] present a level set framework to perform various surface editing operations.

We extend their work to solids bounded by surfels. We present boolean operations on surfel-bounded solids as an interactive editing tool. The work presented in this paper is mostly related to the work of Kristjansson et al. and Museth et al. Our algorithm can not only display the result of the boolean operation, but also compute the resulting solid at interactive rates. We also show that we are able to represent the sharp features in the resulting solid.

3 Surfel-Bounded Solids

The objects used in this paper are closed solids whose surface is represented by surfels. Each surfel s consists of a position $\mathbf{x}_s$, a radius of influence r_s and an orientation $\mathbf{n}_s$. Therefore surfels can be thought of as disks orthogonal to $\mathbf{n}_s$ with center $\mathbf{x}_s$ and radius r_s. The radius r_s should be chosen so that the projections of the disks on the image plane overlap. The surfel-bounded solids are obtained by LDC (*layered depth cube*) sampling and 3-to-1 reduction as described in [Pfister et al. 2000]. Initially each surfel will thus have a radius $r_s = \sqrt{3}h$ with h the sampling distance in each dimension chosen to match the required display resolution. Although we use uniformly sampled solids, our algorithms do not rely on this. For each solid we define $r_{max} = \max r_s$ as the radius of the largest surfel belonging to its surface.

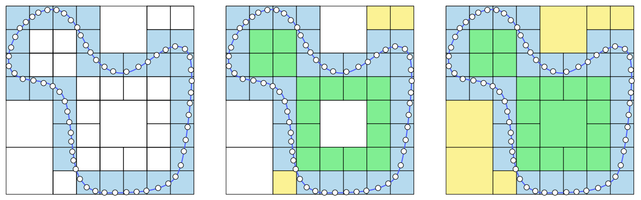

Figure 2: Constructing the quadtree of depth $d = 3$. Left: in a first step the quadtree is constructed; the blue cells are the boundary cells. Middle: classifying the empty leaf cells at depth $d = 3$. Green cells are inside the solid, yellow cells are outside the solid. Right: classifying the empty leaf cells at depth $d - 1$, i.e. 2.

4 Inside-Outside Test

When constructing a new surfel-bounded solid from two solids A and B we have to determine which surfels of A and B will be part of the surface of the resulting solid. Depending on the boolean operation different parts of the surfaces of A and B will represent the boundary of the new solid. E.g. when taking the difference $A - B$ we want to keep the part of the surface of A that is outside B and the part of the inverted surface of B that is inside A. Table 1 gives an overview for the different boolean operations.

In this section we propose a fast inside-outside test that enables us to classify the surfels of solid A as inside, outside or intersecting with the surface of solid B and vice versa. The inside-outside test is based on 3-color octrees [Samet 1990] with leaf cells classified as interior, exterior or boundary. For boundary leaf cells we partition the space even further using two parallel planes.

For clarity the ideas presented in this section are illustrated in two dimensions, but are easily extended to 3D.

4.1 Octree Construction

For each solid we construct an axis-aligned octree. We start with the bounding box containing all the surfels of the solid and subdivide it into 8 equally sized children. Each node is recursively split into 8 children as long as it contains surfels and as long as a user-chosen depth d (typically 4 or 5) is not reached.

After constructing the octree, the empty cells are classified as being inside or outside the solid, as illustrated in figure 2. The resulting octree has three types of leaf cells: boundary cells, empty cells inside the solid and empty cells outside the solid. Within a node of the octree, each cell has a neighbor in one of the principal directions. If an empty cell has a non-empty neighbor we look at the orientation of the surfels in this neighboring cell. The orientation of the surfel that is closest to the empty cell tells us if the empty cell is inside or outside the solid: if this surfel is pointing towards the empty cell, the empty cell must be outside the solid, if the surfel is pointing away from the empty cell, the empty cell must be inside the solid. More formally: let $\mathbf{c}_e$ and $\mathbf{c}_n$ be the coordinates of the centers of the empty cell and its non-empty neighbor and let s be the surfel closest to the empty cell with normal $\mathbf{n}_s$, then the empty cell is classified as inside if $(\mathbf{c}_n - \mathbf{c}_e) \cdot \mathbf{n}_s > 0$. Otherwise, the empty cell must be outside the solid.

There are three different cases when classifying an empty cell:

- the empty cell has only one non-empty neighbor,
- the empty cell has more than one non-empty neighbor (figure 3, left),
- the empty cell has no non-empty neighbor (figure 3, right).

In the first case the empty cell is classified by looking at this non-empty neighbor. In the second case, we only consider one of the non-empty neighbors. In the third case, we first classify the neighbors, and give the same classification to the empty cell as neighboring empty cells must have the same classification. Because a node in the octree has at least one non-empty cell, we can always

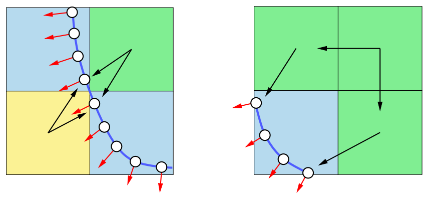

Figure 3: Classification of empty cells. Left: both empty cells have two boundary neighbors. The yellow cell is classified outside, the green cell inside. Right: the empty cell at the right top has two empty neighbors in the principal directions. It is classified as inside after the two other empty cells are classified.

classify the empty cells. In the worst case, if a node has exactly one non-empty child, we need three steps to classify the remaining seven empty cells.

4.2 Partitioning for Boundary Cells

Boundary cells are partitioned even further by constructing two parallel planes within the cell. These planes divide the cell in three areas: interior, exterior and boundary. Let $\mathbf{n} = \sum \mathbf{n}_s / \| \sum \mathbf{n}_s \|$ be the average normal over all surfels s in the boundary cell. Define the two parallel planes $\mathbf{n} \cdot \mathbf{x} - d_1 = 0$ and $\mathbf{n} \cdot \mathbf{x} - d_2 = 0$ and let $d_1 = \min(\mathbf{n} \cdot \mathbf{x}_s)$ and $d_2 = \max(\mathbf{n} \cdot \mathbf{x}_s)$ over all surfels s in the boundary cell. The offsets d_1 and d_2 are determined by the surfels $s_1 = \arg\min(\mathbf{n} \cdot \mathbf{x}_s)$ and $s_2 = \arg\max(\mathbf{n} \cdot \mathbf{x}_s)$. All the surfels s of the boundary cell are now on the positive side of the plane $\mathbf{n} \cdot \mathbf{x} - d_1 = 0$ and on the negative side of the plane $\mathbf{n} \cdot \mathbf{x} - d_2 = 0$ as can be seen in figure 4, left. The other sides of these two planes are empty and can be classified as interior or exterior. The negative half-space of $\mathbf{n} \cdot \mathbf{x} - d_1 = 0$ is inside (outside) the solid if $\mathbf{n} \cdot \mathbf{n}_{s_1}$ is positive (negative). The positive half-space of $\mathbf{n} \cdot \mathbf{x} - d_2 = 0$ is inside (outside) the solid if $\mathbf{n} \cdot \mathbf{n}_{s_2}$ is negative (positive). In figure 4 the left side (yellow) is classified as outside and the right side (green) is classified as inside.

The partitioning is best when all the surfels in the boundary cell lie in the same plane and have the same orientation. The partitioning is not always optimal as one would expect. If a boundary cell contains surfels from differently oriented surfaces there might exist a better partitioning. In the absence of noise (incorrectly oriented surfels) however, the partitioning found by the two planes is always correct. In the worst case, both planes may lie outside the boundary cell and there will be no partitioning.

4.3 Inside-Outside Test: Algorithm

We use the octrees constructed for both solids A and B, to test which part of the surface of A is inside (or outside) solid B and vice versa. We test the octrees hierarchically against each other. Starting at the root of the octree we test if the bounding box overlaps with the other octree. When testing boxes for overlap we enlarge them in all directions over a distance r_{max}, the maximal surfel radius. This compensates for the fact that surfels are not just points, but are considered to be disks. The box-box intersection test is based on the separating axis theorem [Gottschalk 1996].

There are four different cases:

1. the bounding box does not overlap with the other octree,
2. the bounding box only overlaps with exterior cells of the other octree,
3. the bounding box only overlaps with interior cells of the other octree,
4. the bounding box overlaps with at least one boundary cell of the other octree.

It is clear that it is not possible for the bounding box to intersect both with an interior and an exterior cell without intersecting with

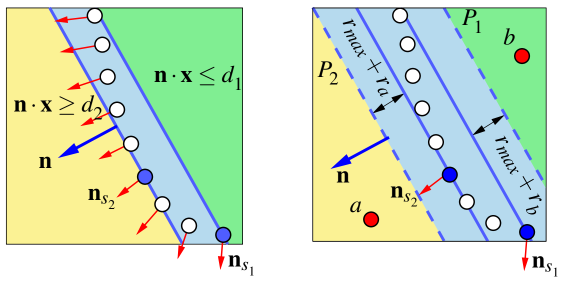

Figure 4: Left: Further partitioning the space in a boundary cell. The left part is classified as outside, the right part as inside. Right: Inside-outside test in boundary cells. Surfel a is classified as outside because it is on the positive side of P_2 and $\mathbf{n} \cdot \mathbf{n}_{s_2} > 0$. Surfel b is classified as inside because it is on the negative side of P_1 and $\mathbf{n} \cdot \mathbf{n}_{s_1} > 0$.

a boundary cell as well. In cases 1 and 2, there is no intersection of surfaces, all the surfels are classified as outside. In case 3, there is no intersection either, but here all the surfels are classified as inside. In the fourth case, we can not make a decision right away because there might be an intersection. We recursively test each of the non-empty children of the root node against the other octree. For each child we repeat the overlap test and continue the recursion as long as its bounding box intersects with a boundary cell. If we're still in the fourth case and the cell we're testing is a leaf cell we have to test each surfel s of the cell individually against the other octree.

It is clear that if a surfel s falls into an interior (exterior) cell it is classified as inside (outside) the other solid. If the surfel s falls into a boundary cell of the other octree it is tested against the two boundary planes within the leaf (see figure 4, right). If s is on the negative side of the plane $P_1 : \mathbf{n} \cdot \mathbf{x} - d_1 + (r_{max} + r_s) = 0$, we classify s as inside if $\mathbf{n} \cdot \mathbf{n}_{s_1} > 0$, otherwise we classify s as outside. If s is on the positive side of plane P_1, we test s in the same way against the plane $P_2 : \mathbf{n} \cdot \mathbf{x} - d_2 - (r_{max} + r_s) = 0$. The offsets $r_{max} + r_s$ are introduced to compensate for the fact that surfels are not points, but disks with radius r_s. Remark that this offset is conservative due to the orientation of the surfels.

If s lies between the two planes P_1 and P_2, we search for its nearest neighbor surfel t in the other solid. If the disks of s and t intersect we classify s as intersecting with the surface of the other solid. If not, s is classified as inside if s is on the negative side of the plane through t: $\mathbf{n}_t \cdot (\mathbf{x}_s - \mathbf{x}_t) < 0$ and outside if s is on the positive side of the plane through t: $\mathbf{n}_t \cdot (\mathbf{x}_s - \mathbf{x}_t) > 0$. If there is no partitioning, i.e. the planes P_1 and P_2 both lie outside the boundary cell, the algorithm always invokes the nearest neighbor search for this boundary cell. For nearest neighbor finding in the boundary cells we implemented the TINN method [Greenspan et al. 2000]. This method is based on the triangle inequality and works well for small point sets (less than 200 points).

Figure 5 illustrates the inside-outside test. The whole algorithm is given in figure 6.

5 Resampling Operator

Surfels of solid A intersecting with the surface of solid B (and vice versa) are classified as *intersecting* by the inside-outside test. They lie both inside and outside the other solid. Leaving them out would result in holes in the final solid. But adding them unchanged, results in a poor representation near the intersection of both solids (figure 7, left). As these surfels lie both inside and outside the other solid, their region of influence, given by the radius r_s, is too large.

For each surfel s classified as intersecting by the inside-outside test we perform the following resampling. During the inside-outside test we not only classify s as intersecting, we also store the closest surfel t found by the nearest neighbor method. We ap-

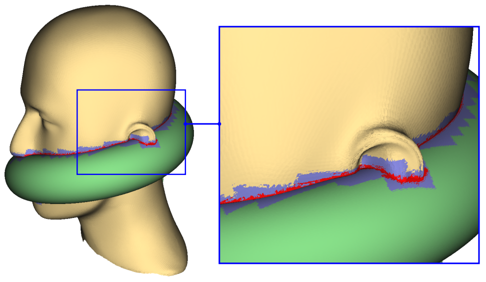

Figure 5: Union of the head and a helix. Red and blue surfels lie in boundary cells of the octree of the other solid. Red surfels lie in the region between the two planes defined for each boundary cell, the blue surfels lie outside this region. Yellow (green) surfels lie outside or in exterior cells of the octree of the helix (head).

proximate the intersection of s with the surface of the other solid by the intersection between s and the plane through t (figure 8). This intersection is a chord of the circle with center $\mathbf{x}_s$ and radius r_s in the plane: $\mathbf{n}_s \cdot (\mathbf{x} - \mathbf{x}_s) = 0$. The plane through t splits the surfel s in two parts. Depending on the boolean operation we want to keep the large part of s or the small part. Suppose we want to keep the large part. We approximate the straight cut by replacing the surfel s by the smaller surfel s'. The new surfel s' crosses the intersection chord by a user defined distance e, which we call the *overshoot error*. If h is the distance from $\mathbf{x}_s$ to the intersection chord, the radius of s' will equal $(r_s + h + e)/2$. To completely cover the intersection chord, we add surfels to the left and right of s' with radius $r_s/3$. These surfels cross the intersection chord for the same amount e. By resampling the surfel s in this way, the straight cut made by s and the plane through t is very well approximated (see figure 7, right). In the examples given in this paper we set the *overshoot error e* equal to $r_{max}/20$. Depending on the distance h, the surfel s is usually replaced by 1, 3 or 5 smaller surfels. Similar calculations hold when we want to keep the smaller part of s.

6 Implementation & Results

We implemented the data structures and algorithms described above in C++. For the visualization of the surfels we use textured triangles in OpenGL computed at run-time from the surfels position, orientation, and radius. More advanced rendering techniques [Ren et al. 2002; Coconu and Hege 2002; Zwicker et al. 2001] could be incorporated though. During interaction, we cull back facing surfels using visibility cones [Grossman and Dally 1998]. Objects can be rotated, translated and uniformly scaled. At this point, we do not have non-uniform scaling, but it could be implemented using the technique proposed in [Pauly et al. 2002]. Additionaly we implemented a local smoothing operator [Adams and Dutré 2003] which enables us, if desired, to eliminate sharp creases when adding solids together with the union operator.

We tested our algorithm on a variety of free-form solids. The timings given in this paper are obtained on an 1.6 Ghz AMD Athlon with 512MB RAM and a GeForce4 MX graphics card. In table 2 we give the timings for the creation of one head of the Bond of Union (figures 1 and 10) for different sampling rates and octree depths. During interactive manipulation we obtain average frame rates ranging from 7.7 FPS to 2 FPS depending on the number of surfels. These timings include the time performed on resampling. The update time given is the time spent on updating the octree. This update is not necessary during interaction. It is only invoked when the user has placed the objects in their final position and wants to start with a new boolean operation on the resulting geometry. During update, we not only have to add and delete surfels, we also have to create and delete new nodes in the octree. So most of the update time is used for memory allocation and deallocation. Other results are given in figures 9, 11 and 12 and in the accompanying videos.

```
InsideOutsideTest(node A, node B) {
    **if** (B does not intersect with A)
        classify all surfels of B as OUTSIDE
    **else if** (B only intersects with exterior cells of A)
        classify all surfels of B as OUTSIDE
    **else if** (B only intersects with interior cells of A)
        classify all surfels of B as INSIDE
    **else**
        **if** (B not a leaf node)
            **for each** non-empty child C of B **do**:
                InsideOutsideTest(A, C)
        **else**
            **for each** surfel s of B **do**:
                InsideOutsideForSurfel(A, s)
}

InsideOutsideForSurfel(node A, surfel s) {
    **if** (s outside A) classify s as OUTSIDE
    **else if** (s in exterior cell of A) classify s as OUTSIDE
    **else if** (s in interior cell of A) classify s as INSIDE
    **else**
        **if** ($\mathbf{x}_s \cdot \mathbf{n} - d_1 + (r_{max} + r_s) \leq 0$)
            **if** ($\mathbf{n} \cdot \mathbf{n}_{s_1} > 0$) classify s as INSIDE
            **else** classify s as OUTSIDE
        **else if** ($\mathbf{x}_s \cdot \mathbf{n} - d_2 - (r_{max} + r_s) \geq 0$)
            **if** ($\mathbf{n} \cdot \mathbf{n}_{s_2} > 0$) classify s as OUTSIDE
            **else** classify s as INSIDE
        **else**
            surfel t = nearest neighbor of s in A
            **if** (s and t do not intersect)
                **if** ($\mathbf{n}_t \cdot (\mathbf{x}_s - \mathbf{x}_t) > 0$) classify s as OUTSIDE
                **else** classify s as INSIDE
            **else** classify s as INTERSECTING
}
```

Figure 6: Pseudocode for inside-outside test. The first procedure is called with the root of the octrees of solids A and B and classifies the surfels of solid B as `INSIDE`, `OUTSIDE` or `INTERSECTING` with solid A.

7 Conclusion & Future Work

We presented boolean operations on free-form solids bounded by surfels. The simple yet efficient inside-outside test based on the octree enables us to calculate the result at interactive rates. Thanks to the resampling operator we are able to represent the sharp edges with great detail. As future work we plan to explore the following topics:

- Combining different types of boundary representations such as polygon-bounded solids with surfel-bounded solids would certainly yield new possibilities.
- Point-sampled objects are most often stored in a hierarchy like the LDC tree [Pfister et al. 2000]. We believe that our algorithms and data structures could easily be extended and used to perform boolean operations on these hierarchical representations.
- The technique presented in this paper could be used in other applications such as virtual sculpting.

Acknowledgments We would like to thank the reviewers for their constructive comments and suggestions. The dragon model is from the Stanford Computer Graphics Laboratory. The horse model is provided by Cyberware Inc. The head model is provided by the Graphics and Imaging Laboratory, University of Washington. The first author is funded as a Research Assistant by the Fund for Scientific Research - Flanders, Belgium (Aspirant F.W.O.-Vlaanderen).

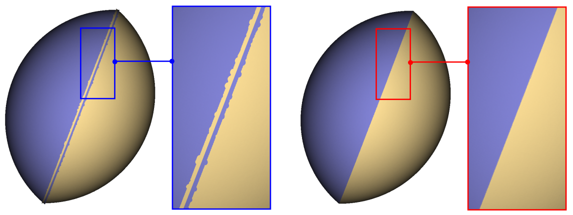
Figure 7: Intersection of two spheres. Left: no resampling. Right: resampling of surfels which intersect with the other surface. This results in sharp edges without significant overshoot, even under magnification.

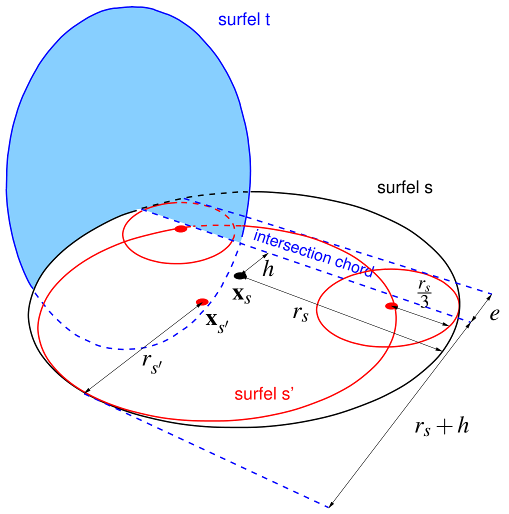

Figure 8: Intersection of the surfel s and the plane through its nearest neighbor t in the other solid. Surfel s is replaced by three smaller surfels by the resampling operator.

Head		Helix			
number of surfels	octree depth	number of surfels	octree depth	interaction time	update time
30k	4	60k	4	130 ms (7.7 FPS)	900 ms
90k	5	170k	5	240 ms (4.2 FPS)	2150 ms
200k	5	250k	5	340 ms (2.9 FPS)	2890 ms
350k	5	370k	5	500 ms (2 FPS)	4690 ms

Table 2: Timings for the head-helix difference for different numbers of surfels and octree depths.

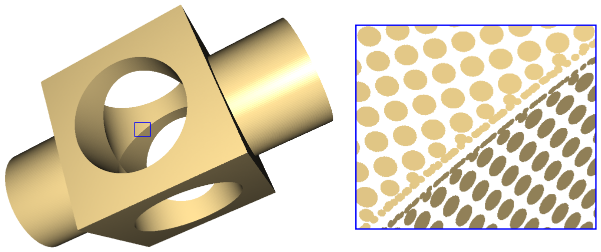
Figure 9: The classic CSG example. The cube consists of 65k surfels, the cylinder of 50k surfels. Average interaction rate is 16 FPS. Average update time is 600 ms. Right: closeup drawn using smaller disks (radii $r_s/2$) for the surfels.

References

ADAMS, B., AND DUTRÉ, P. 2003. A smoothing operator for boolean operations on surfel-bounded solids. Tech. rep., April.

ALEXA, M., BEHR, J., COHEN-OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. 2001. Point set surfaces. *IEEE Visualization 2001* (October), 21–28.

BOTSCH, M., WIRATANAYA, A., AND KOBBELT, L. 2002. Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th workshop on Rendering*, Eurographics Association, 53–64.

CHEN, B., AND NGUYEN, M. X. 2001. Pop: a hybrid point and polygon rendering system for large data. In *IEEE Visualization 2001*, 45–52.

COCONU, L., AND HEGE, H.-C. 2002. Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th workshop on Rendering*, Eurographics Association, 43–52.

COHEN, J. D., ALIAGA, D. G., AND ZHANG, W. 2001. Hybrid simplification: combining multi-resolution polygon and point rendering. In *IEEE Visualization 2001*, 37–44.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1996. *Computer graphics (2nd ed. in C): principles and practice*. Addison-Wesley Longman Publishing Co., Inc.

GOLDFEATHER, J., HULTQUIST, J. P. M., AND FUCHS, H. 1986. Fast constructive-solid geometry display in the pixel-powers graphics system. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, vol. 20, 107–116.

GOLDFEATHER, J., MOLNAR, S., TURK, G., AND FUCHS, H. 1989. Near real-time csg rendering using tree normalization and geometric pruning. *IEEE Computer Graphics & Applications 9*, 3 (May), 20–28.

GOTTSCHALK, S. 1996. Seperating axis theorem. Tech. Rep. TR96-024, Dept. of Computer Science, UNC Chapel Hill.

GREENSPAN, M., GODIN, G., AND TALBOT, J. 2000. Acceleration of binning nearest neighbor methods. In *Proceedings of Vision Interface 2000*, 337–344.

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point sample rendering. In *Eurographics Rendering Workshop 1998*, 181–192.

HOFFMANN, C. M. 1989. *Geometric and solid modeling: an introduction*. Morgan Kaufmann Publishers Inc.

KALAIAH, A., AND VARSHNEY, A. 2001. Differential point rendering. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 139–150.

KRISTJANSSON, D., BIERMANN, H., AND ZORIN, D. 2001. Approximate boolean operations on free-form solids. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 185–194. ISBN 1-58113-292-1.

LEVOY, M., AND WHITTED, T. 1985. The use of points as a display primitive. Tech. Rep. TR85-022, January.

MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. 2002. Level set surface editing operators. *ACM Transactions on Graphics 21*, 3 (July), 330–338.

PAULY, M., AND GROSS, M. 2001. Spectral processing of point-sampled geometry. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 379–386.

PAULY, M., KOBBELT, L., AND GROSS, M. 2002. Multiresolution modeling of point-sampled geometry. Tech. rep., September.

PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 335–342. ISBN 1-58113-208-5.

RAPPOPORT, A., AND SPITZ, S. 1997. Interactive boolean operations for conceptual design of 3-d solids. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 269–278.

REN, L., PFISTER, H., AND ZWICKER, M. 2002. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum 21*, 3, 461–470. ISSN 1067-7055.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 343–352. ISBN 1-58113-208-5.

SAMET, H. 1990. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc.

ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. 2001. Surface splatting. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 371–378.

ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2002. Pointshop 3d: An interactive system for point-based surface editing. *ACM Transactions on Graphics 21*, 3 (July), 322–329. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

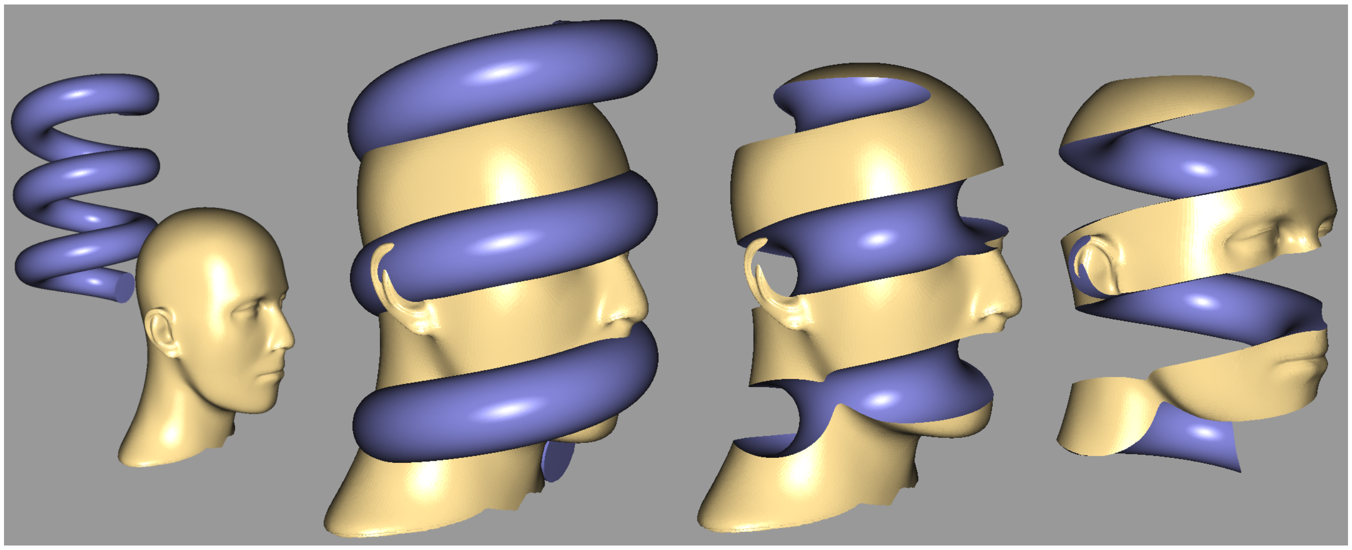
Figure 10: Constructing one head for the Bond of Union (after M.C. Escher) from the mannequin head (350k surfels, octree depth 5) and a helix (370k surfels, octree depth 5). Resulting geometry of union, difference and intersection are shown. During interactive manipulation we obtain a frame rate of 2 frames per second.

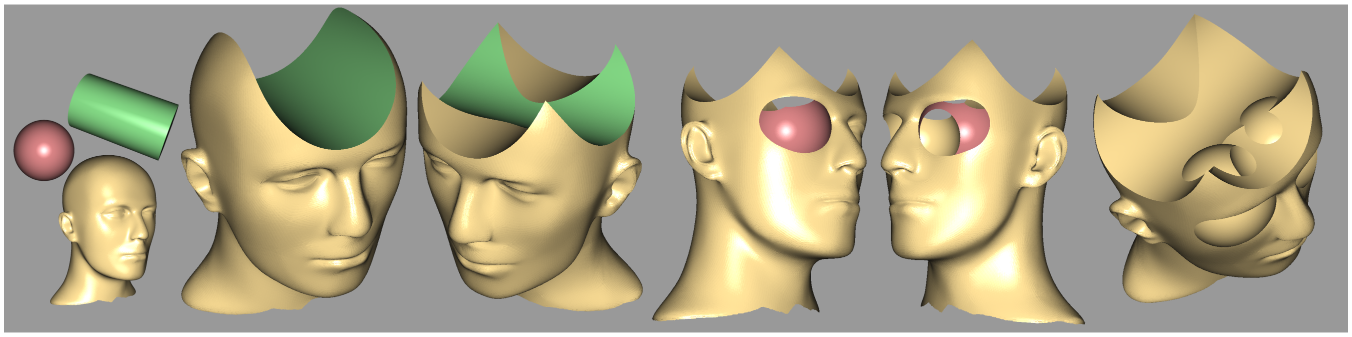
Figure 11: Subtracting 2 cylinders (230k surfels each, octree depth 4) and 2 spheres (46k surfels each, octree depth 4) from the mannequin head (350k surfels, octree depth 5). Average interaction rate is 4.4 FPS. Left: original solids. Middle: the four boolean operations. Right: resulting geometry.

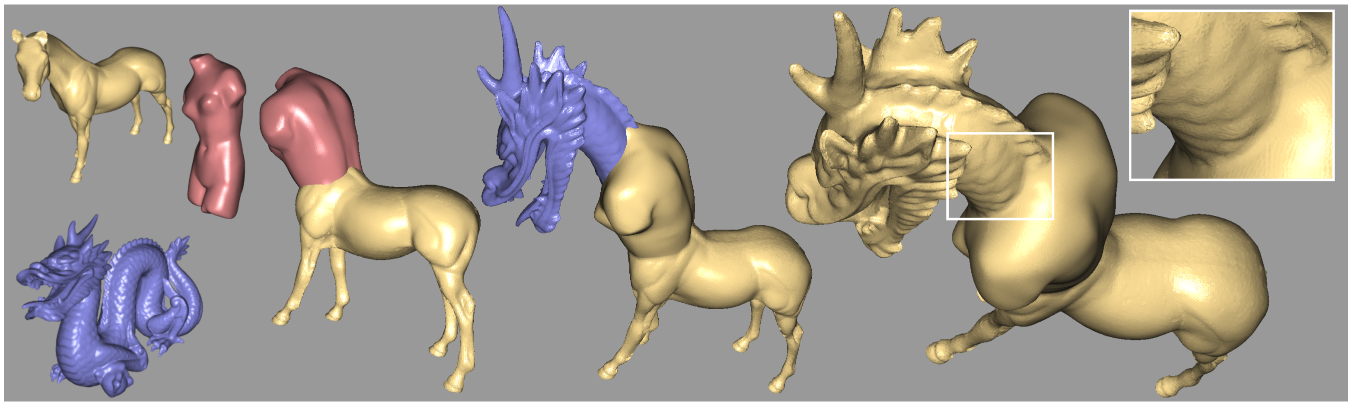
Figure 12: Mythical centaur constructed from the horse model (340k surfels), the Venus model (250k surfels) and the dragon model (650k surfels) all with octree depth 5. Average interaction time of union between horse model and Venus model was 2.5 FPS, between dragon head and centaur body was 3.3 FPS. Local smoothing is performed in the neighborhood of the surface-surface intersections.