Eurographics Symposium on Point-Based Graphics (2004)
M. Alexa, S. Rusinkiewicz, (Editors)

Boolean Operations on Surfel-Bounded Solids Using Programmable Graphics Hardware

Bart Adams Philip Dutré

Department of Computer Science
Katholieke Universiteit Leuven[†]

Abstract

In this paper we present an algorithm to compute boolean operations on free-form solids bounded by surfels using programmable graphics hardware. The intersection, union and difference of two or more solids, is calculated on the GPU using vertex and fragment programs. First, we construct an inside-outside partitioning using 3-color grids and signed distance fields. Next, we use this partitioning to classify the surfels of both solids as inside or outside the other solid. For surfels close to the boundary of the other solid, we use the distance field and its gradient to define a clipping plane, which can be used to resample or clip the surfel. Our algorithm runs at interactive rates on consumer-level graphics hardware.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1 Introduction

For many years, constructive solid geometry (CSG) has been a useful tool in computer graphics and related areas. In the CAD community CSG is applied to primitive objects (such as spheres, cylinders and cubes) to construct shapes with a more complex geometric shape. However, CSG can also be used as an editing tool for complex free-form solids. Special care has to be taken to maintain scalability and interactivity. Recent work [AD03, PKKG03] has showed that, by employing a point-sampled surface representation together with a clever acceleration structure, interactivity can be reached.

In this paper we build upon this work and increase the performance by performing the calculations on the GPU. The motivation behind this is that, when performing boolean operations, the same operation, i.e. inside-outside classification, has to be performed on all surfels. This maps well onto the so-called *single instruction, multiple data* (SIMD) architecture of programmable graphics hardware.

Our algorithm works in two steps: in a first step we compute an inside-outside partitioning for each solid using 3-color grids and signed distance fields. This partitioning is stored in texture memory which will be used as an acceleration structure in the classification step. Next, in the classification step, we classify all surfels as inside, outside or intersecting by testing each surfel against the partitioning of the other solid. For surfels close to the boundary of the other solid, we use the distance field and its gradient to define a clipping plane. This plane can be used to resample or clip the intersecting surfel to obtain sharp edges. We perform all calculations on the GPU, exploiting the SIMD architecture.

This paper builds on [AD03], our contribution is a method to perform boolean operations on surfel-bounded solids entirely on the GPU. To achieve this, we propose GPU-based algorithms to construct the inside-outside partitioning, to classify the surfels and to calculate a clipping or re-sampling plane for intersecting surfels.

We start by giving an overview of related work in section 2. Next, we briefly recapitulate the algorithms this work is based on in section 3. We discuss how these algorithms can be mapped to a GPU implementation in section 4. In section 5 we illustrate the performance and present some results. Opportunities of future research and improvement are given in section 6. Finally, we conclude in section 7.

2 Related Work

Point-Based Rendering. In recent years, researchers investigate the use of points to represent the surface of complex free-form objects. Based on software implementations of

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

© The Eurographics Association 2004.

delivered by EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

point-based rendering algorithms (such as [PZvBG00], [RL00], [ZPvBG01] and [ABCO*01]), different hardware accelerated implementations are proposed. Ren et al. [RPZ02] devise an implementation of the EWA splatting algorithm on graphics hardware. Coconu and Hege [CH02] as well as Botsch and Kobbelt [BK03] present a different hardware-accelerated approach to point rendering using a new feature of graphics hardware called *point sprites*. Dachsbacher et al. [DVS03] propose sequential point trees and allow for adaptive level of detail selection on the GPU. Recently, Zwicker et al. [ZRB*04] introduce perspective accurate splatting, a point rendering algorithm which produces correct splat shapes. They also present an extension which allows the rendering of sharp edges and corners using clipped surfels.

Point-Based Modeling. Also, work has been published on modeling and editing point-sampled objects. Pointshop 3D [ZPKG02] extends 2D photo editing to 3D point clouds, introducing painting sculpting and filtering. Reuter et al. [RSPS04] use constructive solid texturing to interactively paint a 3D object. Pauly et al. [PKKG03] are able to perform large free-form deformations on point-sampled geometry. Together they propose a method to perform boolean operations on point-sampled objects. An alternative method is proposed by Adams and Dutré [AD03]. We discuss this approach in more detail in section 3.

Constructive Solid Geometry. Lots of research has been performed concerning constructive solid geometry. For an excellent overview we refer to [FvDFH96]. Interactive rendering of CSG is often performed using graphics hardware (e.g. [GHF86] and [RS97]). Next to [AD03] and [PKKG03], various other algorithms are proposed to construct the result of a CSG operation, employing different boundary representations (e.g. multiresolution subdivision surfaces [KBZ01] and level sets [MBWB02]).

Distance Fields. Frisken et al. [FPRJ00] propose to use adaptively sampled distance fields, which can be used to perform boolean operations. Others (e.g. [IZLM01] and [SOM04]) use graphics hardware to construct a global distance field using slicing techniques. However, in the context of boolean operations, we do not need a global distance field. We use a different approach and construct a local distance field, only for the regions where we need it (i.e. for the regions close to the boundary).

3 Boolean Operations on Surfel-Bounded Solids

Our GPU implementation is based on the algorithm presented by Adams and Dutré [AD03]. They construct a 3-color octree for each solid, classifying leaf cells as *interior*, *exterior* or *boundary* (see figure 1, left). Additionally, they further partition the boundary cells using two parallel planes (see figure 1, middle). Classification of a point as being inside or outside is easily performed by testing in which space the point lies. Only when the point lies in a boundary cell between the parallel planes, there is no trivial classification. In this case, a nearest neighbor query is performed and the point is tested against the plane defined by its nearest neighbor.

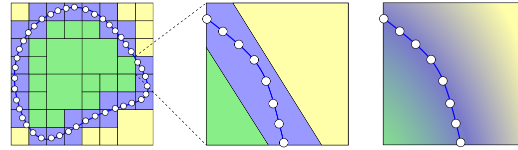

Figure 1: *Left and middle: inside-outside partitioning of a surfel-bounded solid using the technique presented in [AD03]. Right: partitioning of the boundary cell using a signed distance field.*

The constructed octree can also be used to test surfels in group: if a cell of one octree does only intersect with exterior (interior) cells of the other octree, all the surfels in the former cell can be classified as exterior (interior). By performing this test hierarchically, large numbers of surfels can be classified in group. We refer to [AD03] for more details.

4 Boolean Operations Using the GPU

There are two problems when implementing the algorithm presented by Adams and Dutré on the GPU: 1. the algorithm is hierarchical in nature and 2. for some surfels, a nearest neighbor query is necessary to make a classification. The first caveat can be solved by only testing surfels individually and not in group. Also, instead of using a 3-color octree, we propose to use a 3-color grid as this maps better to the GPU. The second problem is a more fundamental one. Both Purcell et al. [PDC*03] and Ma and McCool [MM02] propose methods to perform nearest neighbor queries in 3D space on the GPU. These methods require multiple texture fetches and rendering passes, slowing down the algorithm. We can however circumvent the nearest neighbor query by constructing a signed distance field for the boundary grid cells and using this distance field and its gradient to classify surfels falling in boundary cells.

4.1 Inside-Outside Partitioning

The space around each solid is partitioned as *inside*, *outside* or *boundary* using a 3-color grid. The space inside the boundary cells is further partitioned using a signed distance field. Empty grid cells are classified iteratively as inside or outside based on the distance values of the boundary cells.

Signed Distance Field for Boundary Cells

To avoid the nearest neighbor query present in the algorithm of [AD03], we partition each boundary cell using a signed distance field (see figure 1, right), instead of using two parallel planes (see figure 1, middle). For each corner of the cell we calculate the distance to the closest surfel in the

© The Eurographics Association 2004.

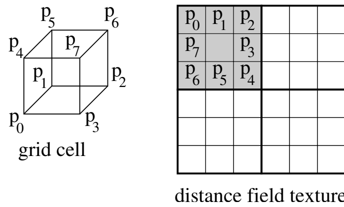

Figure 2: *Correspondence of the corners of a cell and the grid cell pixels. Each cell corresponds to a square area of 3x3 pixels in the distance field texture. Each pixel (except the middle one) corresponds to a corner of the cell.*

cell. If the corner is on the negative side of the plane defined by the closest surfel, the distance is signed to be negative, otherwise the distance is positive. For a point within the boundary cell, the signed distance to the surface can be estimated by trilinear reconstruction using the distances in the corners of the cell. The gradient of the distance field is an approximation of the surface normal for the closest point on the surface. So, by partitioning the boundary cells using a distance field, we are freed from the nearest neighbor query necessary in the algorithm presented by Adams and Dutré. The closest point on the other surface (and its orientation) can be found by evaluating the distance field and the gradient of the distance field in the query point.

Constructing the distance field for a boundary cell requires us to calculate the minimal distance of each corner of the cell to the surfels within the cell. To achieve this using vertex and fragment programs we lay out each grid cell and its corners in a 2D texture: each cell occupies a 3x3 patch of pixels with 8 of these 9 pixels corresponding to corners of the cell (see figure 2). Note that, although neighboring cells share corners in the 3D grid, these corners are duplicated (as pixels) in the distance field texture. Also, as we do not know in advance which grid cells will be boundary cells, we have to allocate space for all cells in the distance field texture.

Vertex programs allow to route a point to an arbitrary location in a buffer. The ability to write to a computed destination address is known as a scatter operation. This principle is also used by Purcell et al. [PDC*03] to construct a grid-based photon map. We use this technique to route each surfel to the center pixel of the 3x3 square corresponding to the grid cell where the surfel lies in. By rendering the surfel as a 3x3 glPoint, we can cover all pixels corresponding to the corners of the grid cell. Routing is performed in a vertex program by computing the grid cell indices where the surfel falls in and mapping them to 2D fragment coordinates corresponding to the cell's center pixel. The vertex program also passes the untransformed surfel position and normal to the fragment program.

The fragment program is responsible for computing the distance between the surfel and the corresponding cell corner $\mathbf{p}_i$. We can distinguish between each fragment of the same glPoint by using the fragment coordinate (WPOS)

```
struct vertout {
    float4 HPostion: POSITION;
    float4 Parameter: WPOS;
    float3 Position: TEXCOORD0;
    float3 Normal: TEXCOORD1;
};

fragout main(in vertout IN,
             uniform samplerRECT gridCoords,
             uniform float diagonal)
{
    fragout OUT;
    float3 corner = texRECT(gridCoords, IN.Parameter.xy);
    float dist = distance(corner, IN.Position);
    half sign = dot(corner-IN.Position, IN.Normal)>0?+1:-1;
    OUT.col = sign*dist;
    OUT.depth = dist/diagonal;
    return OUT;
}
```

Figure 3: *Cg code for the fragment program for the distance field construction for boundary cells. The signed distance is outputted as the color values and the scaled (unsigned) distance is outputted as the depth value. Cell corner coordinates are stored in the gridCoords texture for convenience.*

and the transformed point coordinate. This allows us to use a different $\mathbf{p}_i$ for each of these fragments. If the fragment profile does not support the WPOS input parameter, one could also use the NV_point_sprite extension to obtain texture coordinates for each fragment within a glPoint. The normal of the surfel is used to check if the corner is on the positive or the negative side of the surfel. The fragment program scales the computed (unsigned) distance with the inverse cell diagonal length and outputs it as the depth value for this fragment. The unscaled signed distance is outputted as the color for this fragment. By setting the glDepthFunc to GL_LEQUAL, each pixel of a boundary cell patch in the distance field texture will eventually hold the minimal distance of the corresponding cell corner to the surfels in the corresponding cell. There are no distances for cell corners corresponding to non-boundary cells. Note that the distance field construction is obtained in a single rendering pass. Figure 3 shows Cg code for the fragment program used to compute the distance field for boundary cells.

Classification of Empty Cells

After the distance field construction, we know which cells in the grid are boundary cells: the cells where we have distance values for the corner pixels. We can classify the empty cells as interior or exterior by looking at the distance values of common corners in neighboring boundary cells (see figure 4, step (1)). Iterating further enables us to classify all empty cells: empty cells take the classification of neighboring classified empty cells (see figure 4, step (2)). This procedure can easily be implemented on the GPU using fragment programs. In each iteration we render a quad textured with a 2D layed out grid texture. Each fragment

© The Eurographics Association 2004.

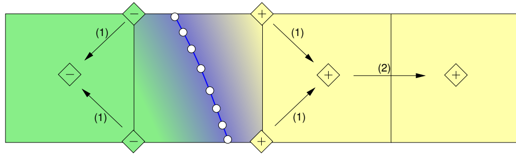

Figure 4: *Classifying the empty grid cells. After a first iteration cells with a side in common with a boundary cell are classified (1). Empty cells neighboring other empty cells take the same classification (2).*

corresponds to a cell in the 3D grid. By looking up the classification of the left, right, bottom, top, back and front neighbor cells we are able to classify all cells in a few iteration steps.

Note that, without precaution, each fragment, classified or not, is (re-)classified in each iteration step. Performance can be increased by tiling the screen with large points instead of with a single quad. Using the NV_OCCLUSION_QUERY extension, we could stop drawing a tile once all the cells covered by its pixels are finished. Purcell et al. [PDC*03] use this principle to decrease the convergence time in computing a radiance estimate in a photon mapping context. However, we did not implement this optimization.

4.2 Inside-Outside Classification

After constructing a distance field for the boundary cells, and an inside-outside classification for the empty cells (for both solids), we are able to construct the result of a boolean operation. To increase performance, we lay out the positions of the surfels in a floating point texture. Rendering a quad covered with this texture enables us to fetch the surfel position in a fragment program using the texture coordinates. Next, we can index the grid texture (containing the classification of the cells) and the distance field texture (containing the distance values for the corners of the boundary cells), by converting the surfel position to a 3D grid index. If the surfel lies in an empty grid cell, it takes the classification of this empty grid cell. Otherwise, trilinear reconstruction of the signed distance values of the grid cell corners is used to classify the surfel.

We can increase performance by using a two-pass algorithm. In a first pass we test the surfels against the 3-color grid. Surfels falling in empty cells can be trivially classified. We mark these surfels as classified by writing a depth value z_1 to the depth buffer for the corresponding fragment. For surfels falling in a boundary cell, we set the depth value to z_2 with $z_1 < z_2$. In a second pass, we disable depth writing and render the same textured quad, but now at a depth z with $z_1 < z < z_2$. In this pass we perform trilinear reconstruction to classify surfels falling in boundary cells. By setting glDepthFunc to GL_LEQUAL we ensure only surfels which correspond to unclassified surfels by the first pass are classified in the second pass. If the graphics board supports early z culling, this two-pass algorithm increases performance significantly, because most of the surfels are classified in the first pass.

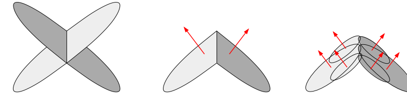

Figure 5: *Left: two surfels on the intersection curve. Middle: clipping to obtain sharp edges. Right: resampling to obtain sharp edges.*

4.3 Clipping or Resampling Plane

As discussed in [AD03] and [PKKG03], care has to be taken for surfels close to the surface of the other solid. Pauly et al. propose to clip the surfel against the plane defined by the closest surfel of the other solid (see figure 5, middle). Adams and Dutré propose to replace intersecting surfels by 1 to 5 smaller surfels along the intersection chord with the closest surfel of the other solid (see figure 5, right). Both approaches use the plane defined by the position and normal of the closest surfel. We approximate this plane by using the distance to the surface provided by the distance field and the gradient of the distance field. We use the analytic gradient resulting from the trilinear reconstruction similar to [FPRJ00]. These calculations are performed in a fragment program for all surfels lying in a boundary cell close to the other surface. The resulting clipping planes are stored in a texture and can be used to resample or clip the surfels during rendering (e.g. using the rendering algorithm of [ZRB*04]).

5 Results

All of the algorithms presented are written in Cg [MGAK03] and compiled with cgc version 1.1 to native fp30 and vp20 assembly code. We did not optimize the code by hand. The results are obtained under Linux on a 1.6Ghz PC with a GeForce FX 5600 graphics board. The distance field construction uses a vertex program of 18 instructions and a fragment program of 22 instructions. The 3-color grid construction uses a fragment program which compiles to 56 instructions. Finally, the two classification steps use fragment programs of 50 instructions and 56 instructions respectively.

We start by analyzing the time required to construct the inside-outside partitioning. Table 1 gives timings for the distance field construction for different numbers of surfels. These timings are independent of the grid resolution. Table 2 gives timings for the 3-color grid construction for different grid resolutions. Note that this step is independent of the number of surfels.

number of surfels	60k	170k	250k	370k
time	37ms	98ms	142ms	220ms

Table 1: *Timings for the distance field creation step for different numbers of surfels. This time is independent of the number of grid cells.*

© The Eurographics Association 2004.

grid size	10^3	20^3	30^3	40^3
time	3ms	22ms	104ms	214ms

Table 2: *Timings for the 3-color grid creation step for different numbers of grid cells. This time is independent of the number of surfels.*

For undeformable objects, the inside-outside partitioning needs to be computed only once, e.g. at the beginning of an editing session. Inside-outside classification of the surfels needs to be re-computed each time the position or orientation of one of the objects changes. Table 3 gives timings for the inside-outside partitioning and classification for the head-helix example (see figure 6) given in [AD03] for different numbers of surfels. The time required for inside-outside classification does not depend on the number of grid cells. This table also gives a comparison between our GPU implementation and the software implementation provided by [AD03]. Timings are run on the same PC. The GPU algorithm for inside-outside partitioning is a factor 3 to 7 faster than the software algorithm. Classification using the GPU is a factor 9 to 17 faster.

Another example (taken from [PKKG03]) is given in figure 7. Inside-outside partitioning for one dragon took 700ms, computing the classification and clipping planes took 122ms per dragon. Each dragon consists of 650k surfels and we used a grid of resolution 36^3 for this example.

Note that the timings given in this paper do not include the rendering of the resulting solid. The result of the classification is written to a texture. For surfels close to the surface of the other solid (i.e. surfels within a distance smaller than their radius) we also write the intersection plane to the same texture. This texture can be used as a lookup texture during rendering or can be read back to host memory to obtain the geometry of the resulting solid. In our implementation, rendering is performed by projecting the surfels as GL_POINTS and discarding fragments by looking up the classification in a fragment program.

6 Discussion and Future Work

Using a regular grid for the inside-outside partitioning has a few limitations. First, in order to be able to capture very fine

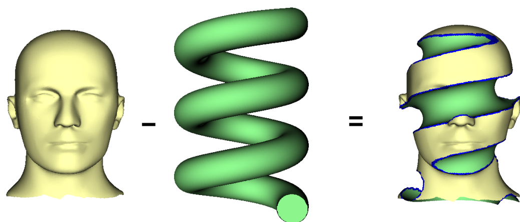

Figure 6: *Head-helix difference computed and rendered using only the GPU.*

head	helix	inside-outside partitioning	inside-outside classification
30k	60k	245ms [720ms]	19ms [330ms]
90k	170k	351ms [1980ms]	49ms [490ms]
200k	250k	492ms [3500ms]	80ms [720ms]

Table 3: *Comparison between our GPU implementation and the software implementation of [AD03] (bracketed numbers) for the head-helix difference for different numbers of surfels. For the GPU implementation, grids of size 32^3 are used, for the software implementation, we used octrees of depth 5.*

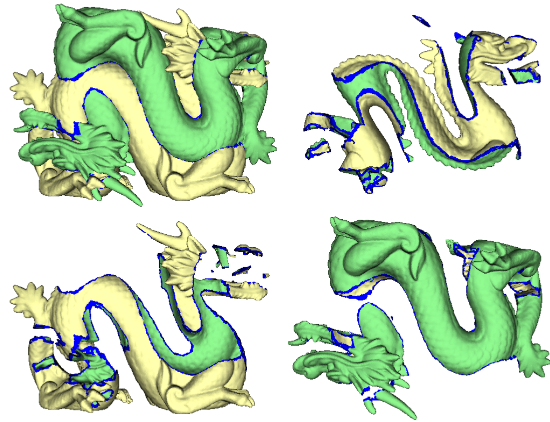

Figure 7: *Boolean operations on two dragons computed and rendered using only the GPU.*

detail using a distance field, a sufficiently high grid resolution is required. Figure 8 illustrates this with the difference of two cubes. The distance field is not able to represent the corners of the cube accurately when using a 3-color grid with 5^3 cells. Further subdivision of the grid to 20^3 cells solves this problem. However, this fine grid resolution is not required to capture the straight edges. Clearly, adaptive refinement similar to [FPRJ00] would be more appropriate. We are currently investigating how to map this on the GPU.

Second, as discussed in section 4.1, we have to allocate space in the distance field texture for all cells, not only for the boundary cells. This limits the grid resolution. For example, if we allow distance field textures of resolution 1024x1024, the number of grid cells is limited to 48^3 as each grid cell occupies a patch of 3x3 pixels in the distance field texture. For most examples, this resolution is sufficient to capture all geometric detail. However, most of the space in the distance field texture is wasted for empty grid cells.

Also, similar to [AD03], we are exploring ways to classify surfels in group. We believe that this is possible by testing bounding boxes of groups of surfels against the 3-color grid and by only testing surfels individually when the bounding box intersects with a boundary cell. This could be done by identifying vertex ranges in a vertex array and by only classifying points individually within the selected ranges.

© The Eurographics Association 2004.

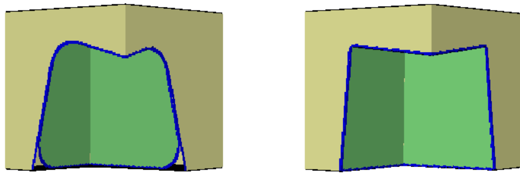

Figure 8: *Difference of two cubes. Left: using grids of size 5^3. The distance field for the boundary cells is too coarse to represent the corners. Right: using grids of size 20^3.*

Finally, our current implementation uses multiple p-buffers and therefore there are a lot of context switches. By combining multiple buffers into one buffer, we believe significant improvements are possible.

7 Conclusion

We have proposed a method to calculate the result of boolean operations on surfel-bounded solids, entirely on the GPU. In a first step, we build an inside-outside partitioning for each solid using a 3-color grid and a signed distance field per boundary cell. Next, we use a fragment program to classify the surfels as inside or outside the other solid. Along with this classification we use the distance field and its gradient to compute an approximate clipping plane for surfels close to the surface of the other solid. The result is stored in a texture on graphics memory and can either be used to render the result of the boolean operation or can be read back to host memory to obtain the geometry of the resulting solid.

Acknowledgments. We would like to thank Ares Lagae for helpful comments and for proofreading. The first author is funded as a Research Assistant by the Fund for Scientific Research - Flanders, Belgium (Aspirant F.W.O.-Vlaanderen).

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. *IEEE Visualization 2001* (2001), 21–28. 2

[AD03] ADAMS B., DUTRÉ P.: Interactive boolean operations on surfel-bounded solids. In *Proceedings of ACM SIGGRAPH 2003* (2003), pp. 651–656. 1, 2, 4, 5

[BK03] BOTSCH M., KOBBELT L.: High-quality point-based rendering on modern gpus. In *Proceedings of Pacific Graphics 2003* (2003), pp. 335–343. 2

[CH02] COCONU L., HEGE H.-C.: Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th workshop on Rendering* (2002), pp. 43–52. 2

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. pp. 657–662. 2

[FPRJ00] FRISKEN S. F., PERRY R. N., ROCKWOOD A. P., JONES T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 249–254. 2, 4, 5

[FvDFH96] FOLEY J. D., VAN DAM A., FEINER S. K., HUGHES J. F.: *Computer graphics (2nd ed. in C): principles and practice*. Addison-Wesley Longman Publishing Co., Inc., 1996. 2

[GHF86] GOLDFEATHER J., HULTQUIST J. P. M., FUCHS H.: Fast constructive-solid geometry display in the pixel-powers graphics system. In *Proceedings of ACM SIGGRAPH 1986* (1986), pp. 107–116. 2

[IZLM01] III K. E. H., ZAFERAKIS A., LIN M. C., MANOCHA D.: Fast and simple 2d geometric proximity queries using graphics hardware. In *2001 ACM Symposium on Interactive 3D Graphics* (2001), pp. 145–148. 2

[KBZ01] KRISTJANSSON D., BIERMANN H., ZORIN D.: Approximate boolean operations on free-form solids. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 185–194. 2

[MBWB02] MUSETH K., BREEN D. E., WHITAKER R. T., BARR A. H.: Level set surface editing operators. *ACM Transactions on Graphics 21*, 3 (2002), 330–338. 2

[MGAK03] MARK W. R., GLANVILLE R. S., AKELEY K., KILGARD M. J.: Cg: A system for programming graphics hardware in a c-like language. In *Proceedings of ACM SIGGRAPH 2003* (2003), pp. 896–907. 4

[MM02] MA V. C. H., MCCOOL M. D.: Low latency photon mapping using block hashing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), pp. 89–99. 2

[PDC*03] PURCELL T. J., DONNER C., CAMMARANO M., JENSEN H. W., HANRAHAN P.: Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), pp. 41–50. 2, 3, 4

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. In *Proceedings of ACM SIGGRAPH 2003* (2003), pp. 641–650. 1, 2, 4, 5

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 335–342. 2

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of ACM Siggraph 2000* (2000), pp. 343–352. 2

[RPZ02] REN L., PFISTER H., ZWICKER M.: Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum 21*, 3 (2002), 461–470. 2

[RS97] RAPPOPORT A., SPITZ S.: Interactive boolean operations for conceptual design of 3-d solids. In *Proceedings of ACM SIGGRAPH 1997* (1997), pp. 269–278. 2

[RSPS04] REUTER P., SCHMITT B., PASKO A., SCHLICK C.: Interactive solid texturing using point-based multiresolution representations. In *Journal of WSCG 2004* (2004), vol. 12, pp. 363–370. 2

[SOM04] SUD A., OTADUY M. A., MANOCHA D.: Difi: Fast 3d distance field computation using graphics hardware. In *Proc. of Eurographics 2004, to appear* (2004). 2

[ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of ACM Siggraph 2002* (2002), pp. 322–329. 2

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 371–378. 2

[ZRB*04] ZWICKER M., RASANEN J., BOTSCH M., DACHSBACHER C., PAULY M.: Perspective accurate splatting. In *Graphics Interface 2004, to appear* (2004). 2, 4

© The Eurographics Association 2004.