CSG Operations of Arbitrary Primitives with Interval Arithmetic and Real-Time Ray Casting

Younis Hijazi[1], Aaron Knoll[2,4], Mathias Schott[3], Andrew Kensler[3], Charles Hansen[3,4], and Hans Hagen[2,4]

[1] IGG-LSIIT, UMR 7005 CNRS, University of Strasbourg
[2] University of Kaiserslautern
[3] SCI Institute, University of Utah
[4] IRTG 1131

Abstract. We apply Knoll et al.'s algorithm [9] to interactively ray-cast constructive solid geometry (CSG) objects of arbitrary primitives represented as implicit functions. Whereas modeling globally with implicit surfaces suffers from a lack of control, implicits are well-suited for arbitrary primitives and can be combined through various operations. The conventional way to represent union and intersection with interval arithmetic (IA) is simply using min and max but other operations such as the product of two forms can be useful in modeling joints between multiple objects.
Typical primitives are objects of simple shape, e.g. cubes, cylinders, spheres, etc. Our method handles arbitrary primitives, e.g. superquadrics or non-algebraic implicits. Subdivision and interval arithmetic guarantee robustness whereas GPU ray casting allows for fast and aesthetic rendering. Indeed, ray casting parallelizes efficiently and trivially and thus takes advantage of the continuous increasing computational power of hardware (CPUs and GPUs); moreover it lends itself to multi-bounce effects, such as shadows and transparency, which help for the visualization of complicated objects. With our system, we are able to render multi-material CSG trees of implicits robustly, in interactive time and with good visual quality.

1 Introduction

Constructive solid geometry objects involving implicit surfaces can be an effective geometric representation. Arbitrary-form implicit surfaces can be used to model a wide variety of shapes, as well as perform interpolation and smoothing filters of multiple varieties of data. Constructive solid geometry allows for generalized trimming of these surfaces. Moreover, CSG implicits make for a compact and flexible model, in which the CSG object itself can be represented simply by implicit functions consisting of *min* and *max* operators.

Interactive, pixel-exact rendering of implicits poses a challenge to extraction and rasterization methods. Ray casting methods employing interval arithmetic have conventionally been among the most robust solutions for rendering general-form implicit surfaces, but also among the slowest. However, recent SIMD techniques for the CPU [10] and GPU [9] have shown that IA bisection can be a practical method for interactive

rendering. The contribution of this paper is to show how, in addition to conventional closed-form implicit functions, interval arithmetic methods can be employed in efficiently rendering constructive solid geometry.

2 Related work

In 1982, Roth [17] presented the first algorithm for directly rendering CSG without precomputing the combined boundary representations. His algorithm used the CSG operators to classify the intersections found by ray casting. Goldfeather et al. [2] showed in 1986 how an initial restructuring of the tree could allow CSG to be directly rendered using Z-buffer rasterization. In 1992 Duff [1] demonstrated the use of IA and subdivision for rendering CSG implicits. Nielson [14] presented applications of implicits and CSG in the context of scattered data interpolation. Kirsch et al. [8] provided an enhancement of Goldfeather's algorithm. Günter et al. [3] performed CSG modeling in real-time while Romeiro et al. [16] focused on large CSG models. Not directly related to CSG, the community Hyperfun [6] builds models using the F-rep representation which includes the CSG one.

3 Background

3.1 Ray casting implicits: a root-finding problem

An *implicit surface S* in $3D$ is defined as the set of solutions of an equation

$$f(x,y,z) = 0 \tag{1}$$

where $f : \Omega \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$. In ray casting, we seek the intersection of a ray

$$p(t) = o + td \tag{2}$$

with this surface S. By simple substitution of these position coordinates, we derive a unidimensional expression

$$f_t(t) = f(o_x + td_x, o_y + td_y, o_z + td_z) \tag{3}$$

and solve where $f_t(t) = 0$ for the smallest $t > 0$. Therefore ray casting a $3D$ implicit function reduces to a $1D$ root-finding problem.

Approaches for arbitrary implicits include:

- **Closed-form solutions**, which although fast, may suffer from numerical problems in 32-bit float arithmetic.
- **Point-sampling** [4] evaluates the function at interval endpoints and exploits the rule of signs. This is typically fast, but not generally robust (see Fig. 1(a)).
- **Sturm sequences** [18] break the ray segment into monotonic intervals by recursively bracketing zeros of all derivatives. This is slow and requires differentiability.

- **Piecewise algebraic surfaces** [11], though efficient, are limited to low-degree algebraics when relying on an analytical root-finding scheme.
- **Lipschitz methods** [7] which rely on bounding Lipschitz constants to determine where root-finding methods will converge. This works on a subclass of algebraics.
- **Distance functions** [5] require derivation of a signed distance function from an arbitrary point in space to the surface, and also requires Lipschitz.
- **Inclusion algebra methods** which evaluate an inclusion extension of the implicit (see Fig. 1(b)), and use that for spatial rejection or determining monotonicity. These work for any computable function, but require implementation of an inclusion arithmetic library.

This paper will focus on the latter approach, as it is robust and general, and requires nothing more than a function definition. Historically, it has also been the slowest, primarily due to inefficient implementation and impractical numerical assumptions.

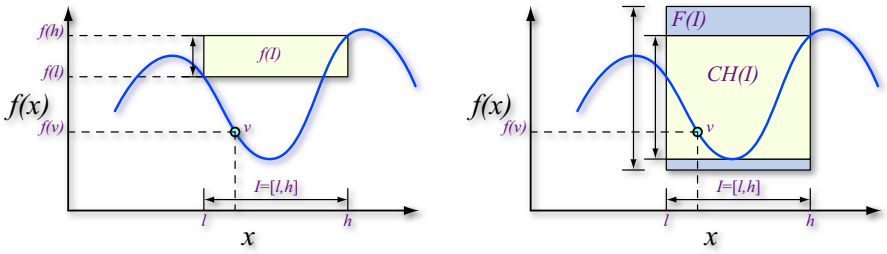

Fig. 1. *The inclusion property.* (a) **Left**: When a function f is non-monotonic on an interval I, evaluating the lower and upper components of a domain interval is insufficient to determine a convex hull over the range. (b) **Right**: This is not the case with an inclusion extension F, which, when evaluated, will enclose all minima and maxima of the function within that interval. Ideally, $F(I)$ is equal or close to the bounds of the convex hull, $CH(I)$.

3.2 CSG and implicits

The three basic operators in constructive solid geometry are the boolean union, intersection and difference. Considering two solid objects A and B respectively represented by the implicit functions f_A and f_B and with the following convention: $f < 0$ inside the solid and $f > 0$ outside the solid (here $f = 0$ defines the solid), we can easily express those operations in terms of implicit functions. Indeed the union between A and B is defined by

$$A \cup B \;=\; min(f_A, f_B). \tag{4}$$

The intersection between A and B is defined by

$$A \cap B \;=\; max(f_A, f_B). \tag{5}$$

Finally the difference between A and B is defined by

$$A \setminus B \;=\; max(f_A, -f_B). \tag{6}$$

Thus the construction of a complex CSG object using n boolean operators reduces to the expression of a single implicit function formed by *min*, *max*, and the implicit primitives.

3.3 Interval Arithmetic

Interval arithmetic (IA) was introduced by Moore [13] as an approach to bounding numerical rounding errors in floating point computation. The same way classical arithmetic operates on real numbers, interval arithmetic defines a set of operations on intervals. We denote an interval as $\overline{\underline{x}} = [\underline{x}, \overline{x}]$, and the base arithmetic operations are as follows:

$$\overline{\underline{x}} + \overline{\underline{y}} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \tag{7}$$

$$\overline{\underline{x}} - \overline{\underline{y}} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \tag{8}$$

$$\overline{\underline{x}} \times \overline{\underline{y}} = [min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy})]. \tag{9}$$

Moore's fundamental theorem of interval arithmetic [13] states that for any function f defined by an arithmetical expression, the corresponding interval evaluation function F is an *inclusion function* of f (where F is the interval *extension* of f):

$$F(\overline{\underline{x}}) \supseteq f(\overline{\underline{x}}) \;=\; \{f(x) \;\mid\; x \in \overline{\underline{x}}\} \tag{10}$$

The inclusion property provides a robust rejection test, i.e.

$$0 \notin F(\overline{\underline{x}}) \Rightarrow 0 \notin f(\overline{\underline{x}}). \tag{11}$$

Inclusion operations are powerful in that they are composable: if each component operator preserves the inclusion property, then arbitrary compositions of these operators will as well. As a result, in practice *any* computable function may be expressed as inclusion arithmetic [12]. For example, the two IA functions we are mostly interested in for performing CSG are *min* and *max* (see Algorithm 1).

3.4 Ray Casting CSG implicits with IA

The inclusion property extends to multivariate implicits as well, making it suitable for a spatial rejection test in ray casting. Moreover, by substituting the inclusion extension of the ray equation (Equation 2) into the implicit extension $CSG(x,y,z)$, we have a univariate extension $CSG_t(X,Y,Z)$. To check whether any given ray interval $\overline{\underline{t}} = [\underline{t}, \overline{t}]$ *possibly* contains our surface, we simply check if $0 \in CSG_t(\overline{\underline{t}})$. As a result, once the inclusion library is implemented, any function composed of its operators can be rendered robustly.

Algorithm 1 *min* and *max* in IA with Cg.

```
typedef float2 interval;

interval imin(interval a, interval b)
{
  return interval(min(a.x,b.x),min(a.y,b.y));
}

interval imax(interval a, interval b)
{
  return interval(max(a.x,b.x),max(a.y,b.y));
}
```

4 Ray Casting CSG implicits with IA on the GPU: results and discussion

Previously we showed how a complex CSG object reduces to a single implicit function. To render these objects efficiently, we turn to the GPU implicit IA bisection algorithm of Knoll et al. [9]. This method employs simple floating-point modulus to effect a stackless recursion method, bisecting along the ray and computing the interval extension of the implicit function along each bisected segment. The following CSG examples are obtained using this technique with relatively small ε (in the order of $1e-5$). Indeed, when dealing with multiple implicits, a precision of $1e-3$ (typically sufficient for non-CSG objects) is too large for guaranteeing good visual quality, especially around the intersections areas between the primitives (see Section 4.6). All benchmarks are measured in frames per second on an NVIDIA 8800 GTX, at 1024x1024 frame buffer resolution. The equations of the CSG primitives are provided in Table 1 of the Appendix.

4.1 Basic CSG operations

Figure 2 shows a simple example of implicit CSG functionality, using a cube (modeled as a high-order superquadric) and a sphere. We have added transparency in some figures for a better understanding of the resulting object.

4.2 More difficult examples

We can handle implicits defined by arbitrary complicated functions in the same way as simpler forms. Figure 3 demonstrates two more difficult functions: the citrus and the heart. CSG requires that its components be closed manifolds (i.e. without boundary); in other words their combination defines a solid object.

Figure 7 (in the Appendix) demonstrates a panel of CSG objects involving several primitives such as the tangle, the decocube, superquadrics, ellipsoids, etc.

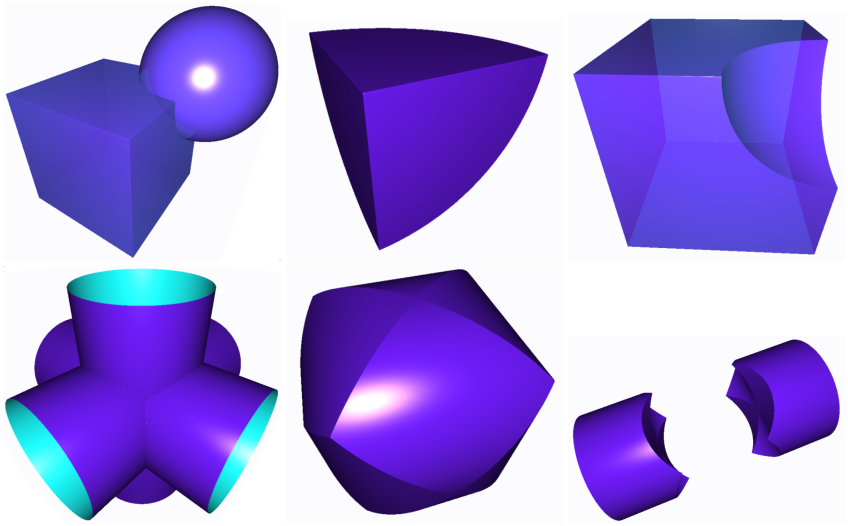

Fig. 2. *Toy examples.* **First row**: union, intersection and difference of a cube and a sphere (20, 160, and 28 fps). **Second row**: union, intersection and difference of three cylinders (91, 84, 127 fps).

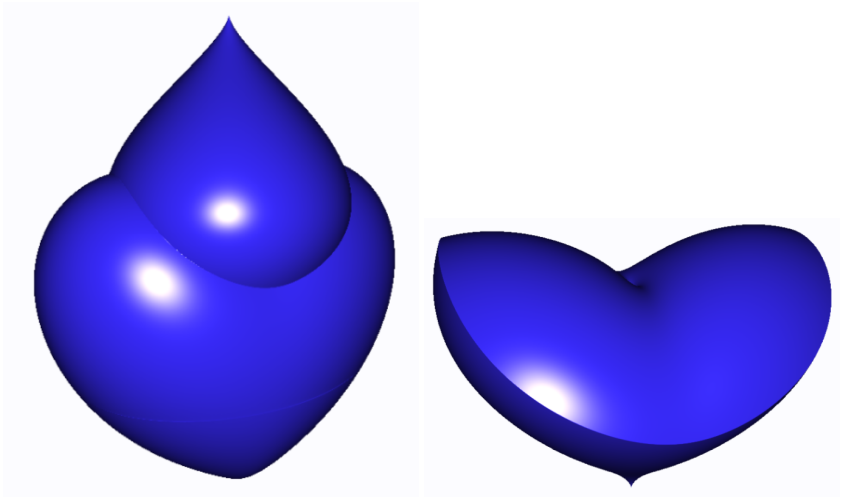

Fig. 3. *CSG of citrus and heart.***Left**: union (50 fps). **Right**: intersection (48 fps).

4.3 Arbitrary blending and dynamic CSG

Arbitrary blending: Implicits inherently support blending operations between multiple basis functions. Such forms need only be expressed as an arbitrary $4D$ implicit $f(x,y,z,w)$, where w varies over time. As ray-casting is performed purely on-the-fly with no precomputation, we have great flexibility in dynamically rendering these func-

tions. Useful morphing methods include product implicits, linear interpolation between surfaces, the hyperbolic and super-elliptic blends; and gaussian or sigmoid blending, shown in Fig. 8 (see Appendix) between the decocube and the sphere. As the blending scheme is also represented as an implicit function in our method, we are able to construct any blend we want.

Dynamic CSG: By setting variables in the CSG objects instead of fixed values, e.g. for a radius, we are able to model time-varying CSG operations. Figure 4 shows a dynamic CSG object: the union of a cube and a radius-varying sphere.

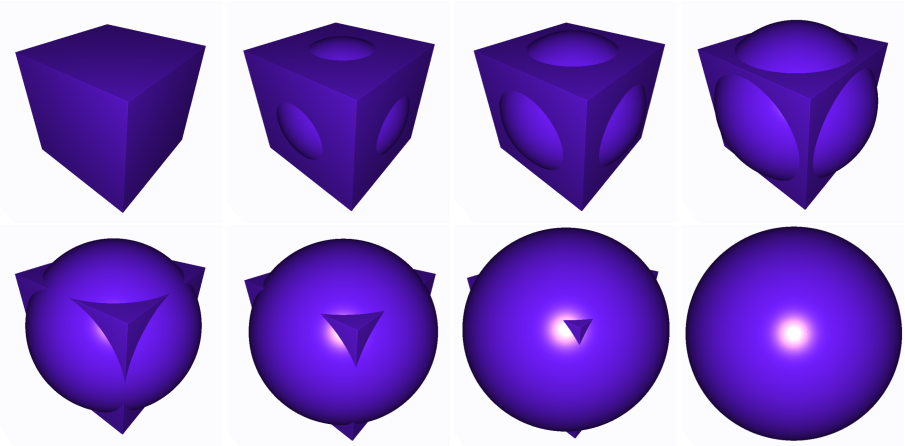

Fig. 4. *Dynamic CSG*: union of a cube and a radius-varying sphere, running at 48-117 fps.

4.4 Multi-material CSG

In addition to using the IA minimum and maximum operators to directly compute the interval extensions of CSG objects, we can evaluate the extensions separately and employ boolean arithmetic to determine which surfaces are intersected by a given ray interval. In addition, we can specify level-set conditions on the individual implicit components, similarly to the CSG methods described in F-rep literature [15]. Given an implicit $f(\omega)$ and a condition $g(\omega)$, inclusion arithmetic allows us to verify $g_+ = \{\underline{g}(\omega) \geq 0\}$ or $g_- = \{\overline{g}(\omega) \leq 0\}$, given the interval form of the inclusion extension G over an interval domain $\omega \subseteq \Omega$. Then, one can render $f \cap g_+$ or $f \cap g_-$ for arbitrary level sets of g. Boolean evaluation of 3-manifold level sets allows us to perform many of the same CSG effects, and at the same time determine which component object is intersected. This allows us to shade components differently as desired (Fig. 5). In addition, increased algorithmic sensitivity near CSG joints due to wider bounds (see Section 4.6) is not an issue using this method.

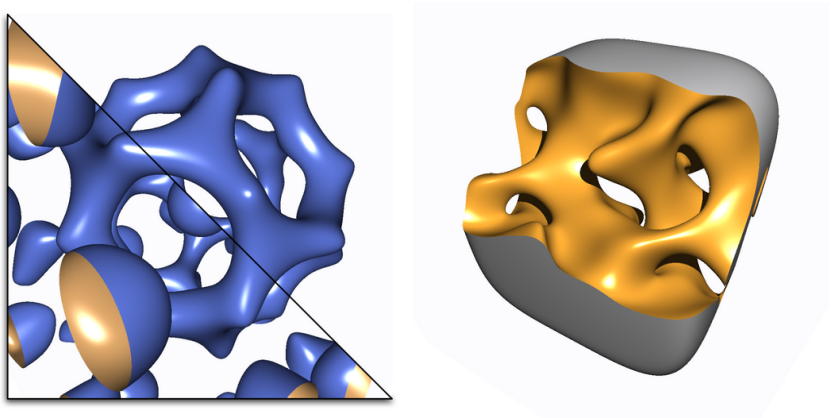

Fig. 5. *CSG objects using level-set conditions*. **Left**: icos.csg (13 fps). **Right**: sesc.csg (9 fps).

4.5 Ray casting effects

As our algorithm relies purely on ray-casting, we can easily support per-pixel lighting models and multi-bounce effects, many of which would be difficult with rasterization (Fig. 6). We briefly describe those modalities.

Transparency: Transparency is useful in visualizing implicits (see Fig. 2 and 6(a)), particularly functions with odd connectivity or disjoint features. It costs around $3\times$ as much as one primary ray per pixel.

Reflections: Reflections are a good example of how built-in features of rasterization hardware can be seamlessly combined with the implicit ray casting system. Looking up a single reflected value from a cubic environment map invokes no performance penalty. Tracing multiple reflection rays in an iterative loop is not significantly more expensive ($20-30\%$), and yields clearly superior results (see Fig. 6(a)(d)).

Gradient shading: Gradient shading is one example of features that can easily be extracted from a ray-casted object; it can help understand its topology. The gradient is computed approximately using central differences. Figure 6(b) shows the gradient shading on an intermediate blend between a decocube and a sphere.

Shadows: Shadows often entail around $20-50\%$ performance penalty. One can equally use a coarser precision for casting shadow rays than primary rays. An example of shadows is illustrated in Fig. 6(c).

4.6 Algorithmic Sensitivity

Much efficiency of the IA bisection technique is owed to the fact that fairly low sensitivity is required for accurate rendering. For many implicit forms without CSG, a termination criterion such as $\varepsilon = 2^{-11} \approx 0.0005$ is sufficient for accurate rendering.

However, in the case of CSG objects, the use of IA minimum and maximum operators cause local bounds to expand, particularly near joints. As a result, a finer discretization is required by our rendering technique to reconstruct the correct surface. Generally, this requirement is not significantly greater ($\varepsilon = 2^{-16} \approx 1e-5$ typically suffices); however this constraint is view-dependent as well as dependent on the form of the implicit itself. Nonetheless, we find IA ray bisection is less sensitive to CSG joints than to fine features in the implicit itself (for example the asymptotic features of the Steiner surface shown in [9]). Moreover, despite the moderately finer ε required to render CSG objects, this sensitivity has little impact on the frame rate (perhaps 10%-20%) compared to the costs of additional IA computation. We note that greater algorithmic sensitivity is not an issue for multi-material objects computed using the boolean evaluation method of Section 4.4.

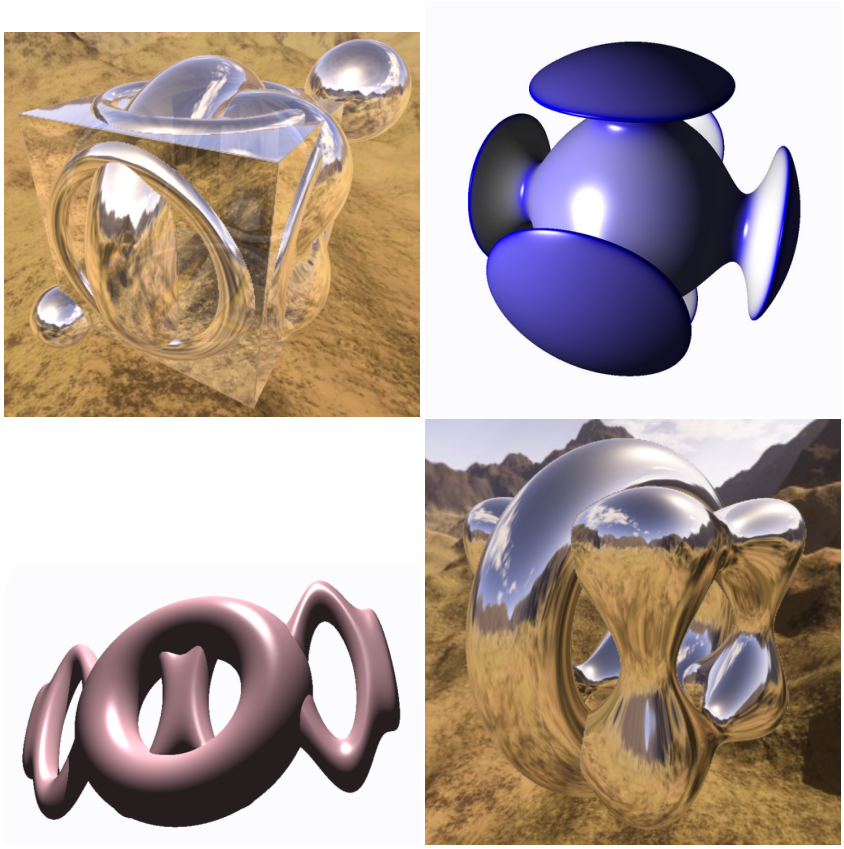

Fig. 6. *Shading Effects.* **Top left to bottom right**: (a) reflections and transparency on multiple-unions CSG object (11 fps); (b) gradient shading on a decocube/sphere blending (41 fps); (c) shadows on *4-Bretzel* $\cup$ *torus* (30 fps); and (d) *tangle* $\cup$ *torus* with up to six reflection rays (11.5 fps).

5 Conclusions and Future Work

We have demonstrated a system which can render multi-material CSG objects of implicits robustly, in interactive time and with good visual quality. Moreover we can add multi-bounce effects, such as shadows and transparency, which help for the understanding of complicated objects. Our system is general: it handles arbitrary primitives; robust: it relies on robust techniques; and efficient: it exploits recent GPU's capabilities.

There are several directions for future work. One desirable direction would be to develop a CSG language similar to [6] and adapt the existing GUI to be able to model large multi-material CSG objects. Extending the ray casting system with a bounding volume hierarchy traversal would allow for a scene graph of piecewise implicit primitives for use in modeling or visualization, and would accelerate rendering. Also comparing interval and (reduced) affine arithmetic as in [9] for the task of CSG modeling may lead to interesting observations. Another direction would be to work on the interaction paradigm of the system so that the user could intuitively build primitives, including free-form surfaces using control points. Using this system to prototype trimmed moving least squares implicits, for example, would be an interesting application. Finally, a virtual reality environment would be perfectly well-suited for such a direct-interaction CSG modeling system.

References

1. Tom Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 131–138, New York, NY, USA, 1992. ACM Press.
2. Jack Goldfeather, Jeff P M Hultquist, and Henry Fuchs. Fast constructive-solid geometry display in the pixel-powers graphics system. *SIGGRAPH Comput. Graph.*, 20(4):107–116, 1986.
3. Brian Guenter and Marcel Gavriliu. Exact procedural csg modeling for real time graphics. Technical Report at Microsoft Research.
4. Pat Hanrahan. Ray tracing algebraic surfaces. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 83–90, New York, NY, USA, 1983. ACM Press.
5. J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.
6. HyperFun team. HyperFun Project. http://www.hyperfun.org/.
7. D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 297–306, New York, NY, USA, 1989. ACM Press.
8. Florian Kirsch and Jürgen Döllner. Rendering techniques for hardware-accelerated image-based csg. In *Journal of WSCG*, pages 221–228, 2004.
9. Aaron Knoll, Younis Hijazi, Andrew Kensler, Mathias Schott, Charles D. Hansen, and Hans Hagen. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Comput. Graph. Forum*, 28(1):26–40, 2009.
10. Aaron Knoll, Younis Hijazi, Ingo Wald, Charles Hansen, and Hans Hagen. Interactive ray tracing of arbitrary implicits with simd interval arithmetic. In *Proceedings of the 2nd IEEE/EG Symposium on Interactive Ray Tracing*, pages 11–18, 2007.
11. Charles Loop and Jim Blinn. Real-time GPU rendering of piecewise algebraic surfaces. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 664–670, New York, NY, USA, 2006. ACM Press.
12. Don Mitchell. Robust ray intersection with interval arithmetic. In *Proceedings on Graphics Interface 1990*, pages 68–74, 1990.
13. R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1966.
14. Gregory M. Nielson. Radial hermite operators for scattered point cloud data with normal vectors and applications to implicitizing polygon mesh surfaces for generalized csg operations and smoothing. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 203–210, Washington, DC, USA, 2004. IEEE Computer Society.
15. Alexander A. Pasko, Valery Adzhiev, Alexei Sourin, and Vladimir V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8):429–446, 1995.
16. Fabiano Romeiro, Luiz Velho, and Luiz Henrique de Figueiredo. Hardware-assisted rendering of csg models. In *XIX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'06)*, pages 139–146, 2006.
17. Scott D. Roth. Ray Casting for Modeling Solids. *Computer Graphics and Image Processing*, 18(2):109–144, 1982.
18. J.J. van Wijk. Ray tracing objects defined by sweeping a sphere. *Computers & Graphics*, 9:283–290, 1985.

A Equations of the implicit primitives

sphere	$x^2+y^2+z^2-r^2$
pseudo-cube	$x^{500}+y^{500}+z^{500}-r^2$
cylinder	x^2+y^2-1
torus	$(1-\sqrt{(x^2+y^2)})^2+z^2-.125$
4-bretzel	$\frac{1}{10}(x^2(1.21-x^2)^2(3.8-x^2)^3-10y^2)^2+60z^2-2$
tangle	$x^4-rx^2+y^4-5y^2+z^4-5z^2+11.8$
decocube	$((x^2+y^2-0.8^2)^2+(z^2-1)^2)((y^2+z^2-0.8^2)^2+$ $(x^2-1)^2)((z^2+x^2-0.8^2)^2+(y^2-1)^2)-0.02$
superquadric	$x^{200}+(.5y^4+.5z^4)^4-1$
ellipsoid	$.25x^2+.25y^2+z^2-1$
heart	$(2x^2+y^2+z^2-1)^3-(.1x^2+y^2)z^3$
citrus	$x^2+z^2-4y^3(1-.5y)^3$
trigonometric	$(1-\sqrt{(x^2+y^2)})^2+\sin(z)^3-.125$
icos.csg	$ic(x,y,z)=2-(\cos(x+\tau y)+\cos(x-\tau y)+\cos(y+\tau z)+$ $\cos(y-\tau z)+\cos(z-\tau x)+\cos(z+\tau x))\ ,\qquad \tau=\frac{1+\sqrt{5}}{2}$ CSG condition (on inclusion intervals): $(0\in ic)$ and $sphere_{inner}<0$ and $sphere_{outer}>0$
sesc.csg	CSG of superellipsoid (se) and sinusoid convolution (sc) $se(x,y,z)=x^6+\frac{1}{2}(y^4+z^4)^4-20$ $sc(x,y,z)=xy+\cos(z)+1.741\sin(2x)\sin(z)\cos(y)+\sin(2y)\sin(x)\cos(z)$ $+\sin(2z)\sin(y)\cos(x)-\cos(2x)\cos(2y)$ $+\cos(2y)\cos(2z)+\cos(2z)\cos(2x)+0.05$ CSG condition (on inclusion intervals): $((sc>0)$ and $(0\in se))$ or $((se<0)$ and $(0\in sc))$
multiple-unions csg	$\min(\min(\min(\min(\min(x^{500}+y^{500}+z^{500}-.25,$ $(x-1)^2+(y-1)^2+(z-1)^2-.2),((x^2+y^2-0.8^2)^2+(z^2-1)^2)$ $((y^2+z^2-0.8^2)^2+(x^2-1)^2)((z^2+x^2-0.8^2)^2+(y^2-1)^2)-0.02),$ $(2x^2+y^2+z^2-1)^3-(.1x^2+y^2)z^3),$ $(1-\sqrt{(x^2+y^2)})^2+z^2-.125),(x+1)^2+(y+1)^2+(z+1)^2-.1)$

Table 1. Formulas of the CSG primitives.

B More examples of CSG implicits

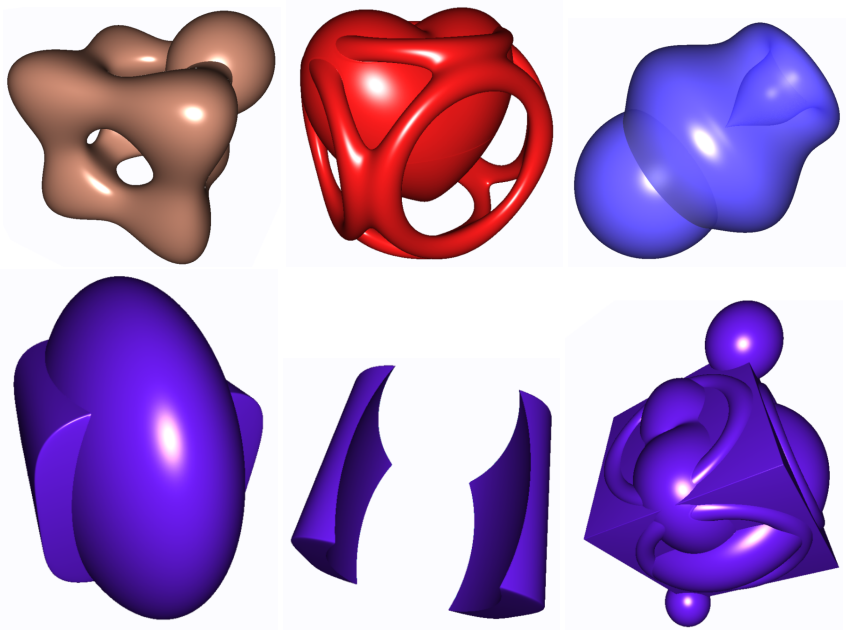

Fig. 7. *CSG with arbitrary primitives.* **First row**: *tangle* $\cup$ *sphere* (12.7 fps), *decocube* $\cup$ *heart* (22 fps) and *trigonometric function* $\cup$ *sphere* (16 fps). **Second row**: *superquadric* $\cup$ *ellipsoid* (41 fps), *superquadric* $\setminus$ *ellipsoid* (60 fps) and multiple-unions CSG object (21 fps).

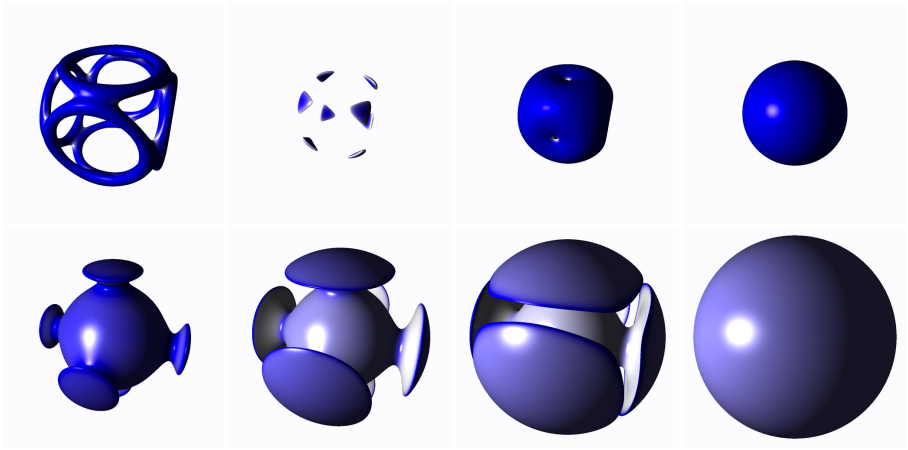

Fig. 8. 4*D sigmoid blending of the decocube and a sphere* running at $33-50$ fps.