High-Performance Graphics 2021
N. Binder and T. Ritschel
(Guest Editors)

Volume 40 (2021), Number 8

Sampling from Quadric-Based CSG Surfaces

P. Trettner[1] and L. Kobbelt[2]

[1]RWTH Aachen University, Germany

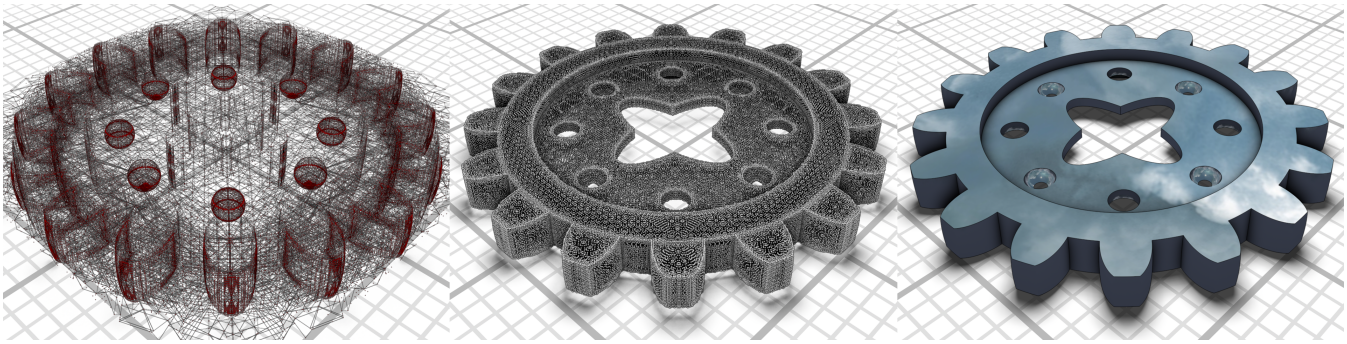


Figure 1: *Our method takes a CSG object defined by quadric surfaces and computes a set of per-surface low-distortion parametrizations. On these, we perform an adaptive subdivision guided by a conservative distance function (left). Once all CSG surface patches are identified, we can generate point samples on them (middle). Our splat-based rendering guarantees perfect surface reconstruction (right). In this example,* 327 000 *samples across 80 quadric primitives were generated. For models of this complexity, we can generate about a million surface samples per second per CPU core. Render time is* 0.5 ms *at 1080p.*

Abstract

We present an efficient method to create samples directly on surfaces defined by constructive solid geometry (CSG) trees or graphs. The generated samples can be used for visualization or as an approximation to the actual surface with strong guarantees. We chose to use quadric surfaces as CSG primitives as they can model classical primitives such as planes, cubes, spheres, cylinders, and ellipsoids, but also certain saddle surfaces. More importantly, they are closed under affine transformations, a desirable property for a modeling system. We also propose a rendering method that performs local quadric ray-tracing and clipping to achieve pixel-perfect accuracy and hole-free rendering.

CCS Concepts

• ***Computing methodologies*** → ***Point-based models; Volumetric models;*** *Ray tracing;*

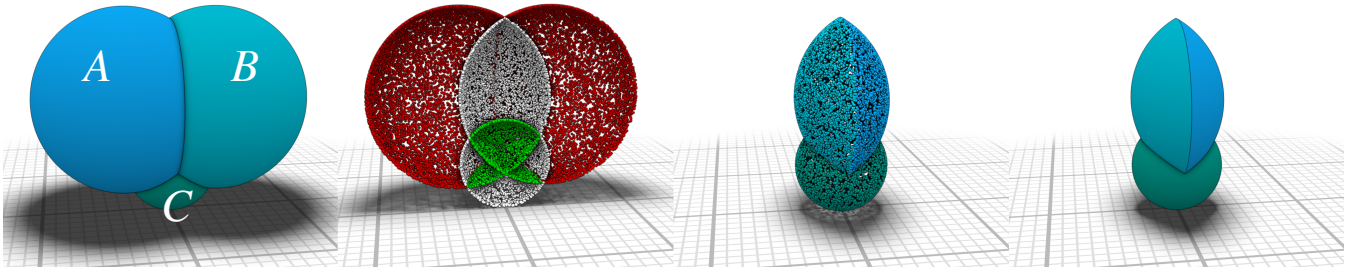
1. Introduction

Constructive solid geometry (CSG) is a popular methodology for defining geometric objects. Given a set of volumetrically defined base objects, so called *primitives*, the resulting CSG object is constructed by combining the primitives using Boolean operations such as union, intersection, or difference. CSG is used in milling simulations, video game level design, CAD tooling, fabrication, 3D modelling, and many other mesh processing pipelines. It is often seen as a natural way to describe complex geometry.

While the primitives are usually quite simple, computing and representing the resulting CSG object tends to be challenging. This typically involves computing the pairwise intersection of all participating primitives. For higher-order primitives or freeform surfaces, many intersections are extraordinarily hard to compute, often no analytic expression is known. Even when restricted to triangle meshes, robustness and performance issues are abundant.

Depending on the use case, working with CSG objects is approached by a wide spectrum of methods. Often, an explicit representation of the object is required. Some need a volumetric representation, leading to Boolean operations on binary space partitioning (BSP) trees. Others only need a surface, resulting in boundary representation (B-rep) methods with analytic surfaces, or mesh-based methods usually working with triangles. When coarse approximations are sufficient, volumetric methods based on implicit surfaces are popular. Sometimes, only visualization is required and rendering-only approaches are employed. These can range from ray-tracing the primitives and applying a form of 1D CSG, over ray-marching implicit functions, to depth-peeling inspired approaches.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.



(a) (b) (c) (d)

Figure 2: *Our* sampling-based CSG *approach creates point samples on CSG surfaces. In this example, the represented surface is* $(A \cap B) \cup C$*, where each primitive is a sphere, shown in (a). Conceptually, point samples from each primitive can be classified as outside (red), inside (green), or surface (white) using the CSG tree (b, front half clipped for clarity). The result is a point cloud representing the CSG surface (c). Additionally, we present a splat-based rendering using local quadric ray-tracing and clipping to achieve pixel-perfect visualization (d). Note that (b) is only conceptual and our actual method uses an adaptive sampling strategy to skip large parts of the red and green regions.*

1.1. Method Overview

In this paper, we present *sampling-based CSG*, a highly efficient method for creating point or splat samples on the surface of a CSG object. The input is a directed acyclic graph (DAG) where leaf nodes are primitives and inner nodes are (variadic) boolean operations. Edges can be annotated by affine and projective transformation matrices. The output is a point cloud that represents the CSG surface. Optionally, we also emit per-point metadata such as surface normal, primitive ID, and clipping information, e.g. for a splat renderer.

Our method is motivated by the observation that the CSG surface consists exclusively of parts of the (transformed) input primitives. A fact that is actually used by many mesh-based CSG approaches: Cut all input triangles along self-intersections and then remove those that do not belong to the CSG surface, i.e. are inside or outside the defined CSG object. We exploit this to create a conceptually simple method for creating point samples on a CSG surface:

- for each (transformed) input primitive P:
  - create point samples $\{s_{P,i}\}$ on the surface of P
  - discard all samples $s_{P,i}$ that are not part of the CSG surface

The set of all non-discarded samples $s_{P,i}$ is a point cloud representing the CSG surface (cf. Figure 2). While this template provides an intuition of why our approach works, our main contribution is to mold this into a practical method and to provide pixel-perfect rendering of the surface.

In its basic formulation, our method has very few constraints on the primitive types. The only requirements are that point samples can be created and that a position can be classified as *inside*, *outside*, or *surface*. The first is needed to create the per-primitive samples, the second to discard samples. For example, in $A \cup B$, a sample s_A on the surface of A is discarded if it is *inside* of B. In $A \cap B$, we would discard s_A if it is *outside* of B.

In practice, this leads to a very inefficient method as often large portions of the primitive surfaces are discarded. This formulation is also problematic for CSG surfaces that are finite but consist of infinite primitives, e.g. a cube modeled as the intersection of six half-spaces. Thus, we also require a per-primitive conservative signed distance function $d_P(x)$, i.e. $d_P(x) > 0$ *outside*, $d_P(x) < 0$ *inside* the surface, and $|d_P(x)|$ is a lower bound for the shortest distance from point x to primitive P.

Classical primitives are spheres, cubes, cylinders, cones, or half-spaces. For the rest of this paper, we focus on a single type of primitive that can represent all these—and more: a quadric surface. Quadrics are closed under affine and even projective transformations and subsumes the classical primitives, while still being reasonably efficient to handle. Section 3 is dedicated to quadric fundamentals, how quadric surfaces can be formulated as a set of heightfields and how to compute a conservative signed distance. In Section 4, we decompose the heightfield domains into a set of low-distortion regions and formulate an adaptive sampling scheme with strong guarantees. In particular, our sampling can guarantee a maximum distance to the next sample on the surface and thus hole-free rendering. How these parts are assembled to sampling-based CSG is explained in Section 5. In Section 6, we present a splat-based renderer with local per-fragment quadric ray-tracing and clipping. Together with the watertight sampling, this results in pixel-perfect renderings, even of complex CSG objects with sharp and small features. The rendering is highly efficient and only exhibits minimal viewpoint dependence.

1.2. Contribution

In summary, we contribute:

- Analytical insight into quadric surfaces, including a surface decomposition into a set of low-distortion heightfields and a useful lower bound for distance-to-quadric-surface that is extremely fast to compute.
- From that, we derive an efficient adaptive sampling scheme to generate point samples on CSG surfaces with strong coverage guarantees.
- Finally, we present a splat-based renderer that uses local quadric ray-tracing and clipping to achieve pixel-perfect real-time renderings of the sampled CSG surface.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

2. Related Work

CSG, solid modeling, and geometric Booleans have an extensive and varied history.

For geometric modeling, the desired result is an explicit computation and representation of the result. It is extremely challenging to design methods that are fast, robust, and accurate. Methods that focus on triangle meshes typically use a spatial acceleration structure to find triangle-triangle intersections, subdivide the input triangles along those cuts, and then classify every triangle as *inside*, *outside*, or *boundary* [BGF15, HKM07, MT13]. The fastest methods use clever classification schemes, for example using ray-tracing [DFR17]. [ZGZJ16] guarantees exact results using rational numbers but has still reasonably performance due to a winding number classification approach. Performance can be increased by using floating point predicates [CLSA20].

A different approach to explicit CSG of triangular or flat polygonal meshes is based on binary space partitionings (BSPs) [NAT90]. Vertices are represented implicitly as the intersection of three planes and Booleans are formulated on BSP trees. This implicit formulation circumvents the need for exact construction. [BF09] uses filtered floating-point predicates to guarantee exact results. Pure BSP-based methods tend to have scaling problems with large models. [CK10] performs mesh Booleans by inserting the meshes in a temporary shared octree and only switch to the BSP representation in octree cells that contain intersections. [NWTK21] introduce octree-embedded BSPs as a solution for the iterated CSG problem. They also use fixed-width integer arithmetic instead of floating point predicates for increased performance.

As a rough rule, mesh-based approaches lack robustness or speed, but are typically accurate. Plane-based approaches tend to be accurate and robust, but slow. When accuracy can be sacrificed, voxel or other volumetric approaches are used. Given implicit surfaces for the primitives, the CSG surface can be defined using min and max. An isosurface extraction algorithm is then used to create the explicit surface representation. Dual Contouring [JLSW02] or Extended Marching Cubes [KBSS01] are popular to preserve some sharp features. For a more thorough survey, we refer to [dALJ*15].

These three paradigms can also be combined to create hybrid approaches. [SLL*18] uses the classical triangle mesh approach but has a dual triangle representation via explicit vertices and plane-based geometry. [PCK10] mixes a mesh-based approach with a volumetric one at the intersections. [SB16] treats the mesh as an approximation of a smooth surface and performs adaptive subdivision close to intersections.

For large and complex models, none of these techniques is fast enough for real-time visualization, though some come close to support interactive modification. Often, the explicit representation is not required and a visualization technique is sufficient. These can be roughly classified into ray-tracing-based and rasterization-based ones.

The basic idea behind raytraced CSG is simple: Tracing a ray against all primitives results in a list of intersection intervals, on which a 1D CSG problem can be solved efficiently [Rot82]. Still, acceleration schemes are required to scale with the number of primitives. Spatial subdivisions of the primitives are popular [RVD06, RVd08, MDG*17]. This can be coupled with balancing the CSG tree [UBT17]. When analytic intersection with the primitives cannot be provided, sphere-tracing can be employed [KB89, SJNJ19].

The rasterization-based approaches render the primitives using the classical rasterization pipeline and then solve the 1D CSG problems using screen-space techniques. One of the earliest approaches was proposed by [GMTF89], who normalized the CSG tree and employed a full-screen pass per primitive to composite the correct result. Later, [SLJ98] used an approach similar to depth peeling, that uses the z-buffer and stencil masks to create correct renderings of normalized CSG trees. Many methods try to optimize the CSG tree to improve the compositing performance [HR05, HR07, Ros11]. Rasterization-based CSG has many similarities with transparency computation [MCTB11] and pixel-lists or k-buffers are popular. [ZCL18] use hashing to speed up the classification of intervals in the 1D CSG problem. Layered depth-normal images can also be used to compute screen-space CSG and reconstruct an approximation of the represented surface [Wan11, WLC10]. When the input models are point-sampled, e.g. acquired from a laser scanner, one can still perform approximate CSG on those [PKKG03, WGT04]. This must not be confused with our approach, which has an analytical input and creates point samples on the exact surface.

Quadrics are a popular tool in computer graphics. Their value can be used as an error metric, which has been used in mesh decimation and smoothing [GH97, LTB19, TK20] and segmentation [TGB13]. [JLSW02] use the minimizer of error quadrics for feature-sensitive isosurface extraction.

Our use case is the quadric surface, i.e. the isosurface of the quadric error function. They are a natural generalization of classical primitives like planes, spheres, ellipsoids, cylinders, and cones. In an early work, [Kle92] constructed a quadric-specific scanline algorithm for visualization purposes. Quadric surfaces can be fitted to triangle meshes [YWLY12, YLW06, AS14]. They can be used in collision detection [CWM*14]. Intersections of two quadric surfaces have been characterized analytically [CWJ02, DLLP08, TWMW09]. However, the resulting implementation is highly complex and not immediately suitable for CSG computation. A basic problem is computing the distance from point to quadric surface, which, except for certain special cases, is non-trivial and must be done iteratively [MES03, Lot14].

Our method is a mixture between explicit computation and the rendering-centric approaches. The generated samples are an explicit, view-independent representation of the CSG surface and come with strong guarantees, suitable for some tasks such as collision detection. However, we still maintain a strong focus on rendering with the same pixel-perfect result that ray-tracing usually produces. A certain analytical breakdown of quadrics is performed to find low-distortion parametrizations and a conservative signed distance function, but nothing as complex as the explicit intersection of two quadrics. More discussion and some quantitative results can be found in Section 7.2.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

3. Quadric Fundamentals

As described in Section 1.1, while the general approach supports a large variety of primitives, for this paper we concentrate on a single, yet powerful type: the quadric surface (cf. Figure 3). Given a symmetric matrix $Q \in \mathbb{R}^{4\times4}$ and a 3D position $x \in \mathbb{R}^4$ in homogeneous coordinates, the quadric surface of Q is defined as the set of points satisfying

$$x^T Q x = 0. \tag{1}$$

It is often convenient to decompose Q via

$$Q = \begin{bmatrix} A & -b \\ -b^T & c \end{bmatrix} \tag{2}$$

and define the quadric surface as all points x that satisfy

$$x^T A x - 2x^T b + c = 0. \tag{3}$$

3.1. Properties

The *value* of the quadric Q is $x^T Q x$ and, unless ambiguous, will be written as $Q(x)$. Thus, the quadric surface is the isosurface $Q(x) = 0$. The *normal* of an isosurface is its normalized gradient:

$$N(x) = \frac{\nabla Q(x)}{||\nabla Q(x)||}, \quad \nabla Q(x) = 2 \cdot (Ax - b) \tag{4}$$

Given a line $p + tv$ with $p, v \in \mathbb{R}^3, t \in \mathbb{R}$ and $||v|| = 1$, we can compute the line-quadric intersection by solving $Q(p + tv) = 0$. This quadratic equation with a single unknown t has up to two real solutions:

$$t_{1,2} = \frac{v^T b - v^T A p}{v^T A v} \pm \sqrt{\left(\frac{v^T b - v^T A p}{v^T A v}\right)^2 - \frac{p^T A p - 2p^T b + c}{v^T A v}} \tag{5}$$

For some quadrics, A is singular and $v^T A v$ can be (close to) zero for some directions. In those cases, the quadratic equation becomes linear and we only get a single solution

$$t = \frac{1}{2} \cdot \frac{p^T A p - 2p^T b + c}{v^T b - v^T A p}. \tag{6}$$

For the sake of simplicity, we will only present the non-degenerate cases in the rest of this paper.

3.2. Sampling Planes and Heightfields

In this subsection, we develop an efficient way to work with the quadric surface. For that, we define three *sampling planes*, from which we orthogonally project onto the quadric surface. This results in three different ways to parametrize the surface, basically heightfields from three different directions.

We call the point q with $Aq = b$ the *center* of the quadric and use a singular value decomposition (SVD) to compute it robustly. For most quadrics, this is the "natural" choice, such as the center of an ellipsoid or the apex of a double cone. For some types, this point is not unique, e.g. for the various cylinder types. In those cases, the actual choice matters little and we take the min-norm solution.

The eigenvectors u_i and eigenvalues $\lambda_i, i \in \{1,2,3\}$, of A are of special importance. The eigenvalues can be used to classify the quadric types. The eigenvectors, together with the center q, form a convenient local coordinate system that we heavily use in our sampling process. We will call u_i the *principal axes* of the quadric Q. For almost all quadrics, all three planes $(q, u_i), i \in \{1,2,3\}$, are mirror symmetries. A selection of quadric types together with their centers and principal axes are shown in Figure 3.

Given the quadric center q and two different principal axes u_i and u_j of Q, we call the plane $q + x \cdot u_i + y \cdot u_j$ with $x, y \in \mathbb{R}$ the *sampling plane* P_{ij}. The P_{ij} can be seen as the coordinate planes of a local coordinate space. These planes cut the quadric into two parts and have a special property: any ray that starts on the plane and is orthogonal to it will intersect with the quadric surface at most once. Such rays have the form $p + tu_k$, where p is restricted to $p = q + x \cdot u_i + y \cdot u_j \in P_{ij}, t \in \mathbb{R}_{\geq 0}$. As u_i, u_j, u_k are eigenvectors, $Q(p + tu_k) = 0$ can be simplified significantly. In particular, we exploit $u_i^T u_k = 0$, $u_j^T u_k = 0$, and $u_i^T A u_i = \lambda_i$ to obtain

$$t = \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c}, \quad \alpha_x = -\frac{\lambda_i}{\lambda_k}, \quad \alpha_y = -\frac{\lambda_j}{\lambda_k}, \quad \alpha_c = \frac{q^T b - c}{\lambda_k}. \tag{7}$$

This can be interpreted as a heightfield $h(x,y) = \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c}$ and the quadric surface (in the local space) as the function graph $(x, y, \pm h(x,y))^T$ where $h(x,y)$ is real-valued. An example quadric and its three sampling planes are shown in Figure 4. Different quadrics lead to different heightfield types, depending on the signs of α_x, α_y, and α_c, as depicted in Figure 5.

3.3. Efficient Conservative Distance

For an efficient sampling and subdivision procedure, we need a distance-to-primitive function, or at least a conservative approximation thereof. Distance between point and quadric surface is a non-trivial problem [MES03, Lot14] and only a few special cases have efficient analytic formulas, like point-to-sphere or point-to-plane. The main use of the distance function is to skip large regions of cut-away surfaces. In this case, even a conservative approximation, a lower bound for the distance-to-quadric-surface, is effective.

Given a point $x_0 \in \mathbb{R}^3$, our goal is to derive a conservative distance function $d(x_0)$ that guarantees that there is no quadric surface within a sphere of radius $d(x_0)$ around x_0. This function should be fast to compute but as close to the true distance as possible.

The value $Q(x)$ of the quadric is quadratic in x and can be rewritten to be "centered" around x_0:

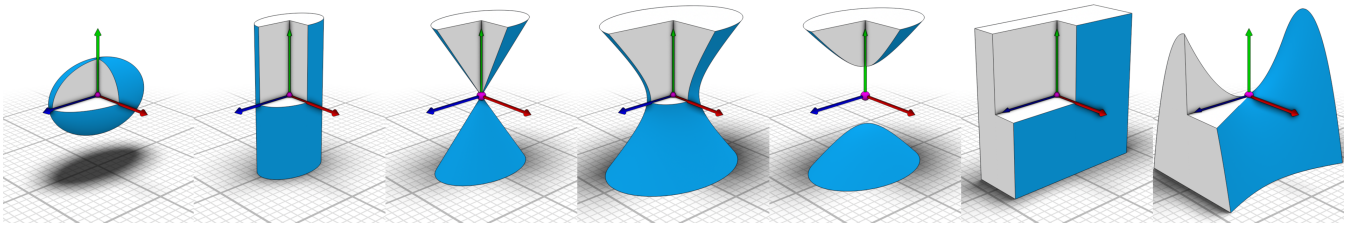
$$\begin{aligned} Q(x) &= Q(x_0) + \nabla Q(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T H Q(x_0)(x - x_0) \\ &= Q(x_0) + 2 \cdot (Ax_0 - b)^T (x - x_0) + (x - x_0)^T A (x - x_0) \end{aligned} \tag{8}$$

We can write x as $x_0 + t \cdot v$ with $t \in \mathbb{R}_{\geq 0}$ and $v \in \mathbb{R}^3, ||v|| = 1$, i.e. in particular $x - x_0 = t \cdot v$:

$$Q(x_0 + t \cdot v) = Q(x_0) + 2 \cdot t \cdot (Ax_0 - b)^T v + t^2 v^T A v \tag{9}$$

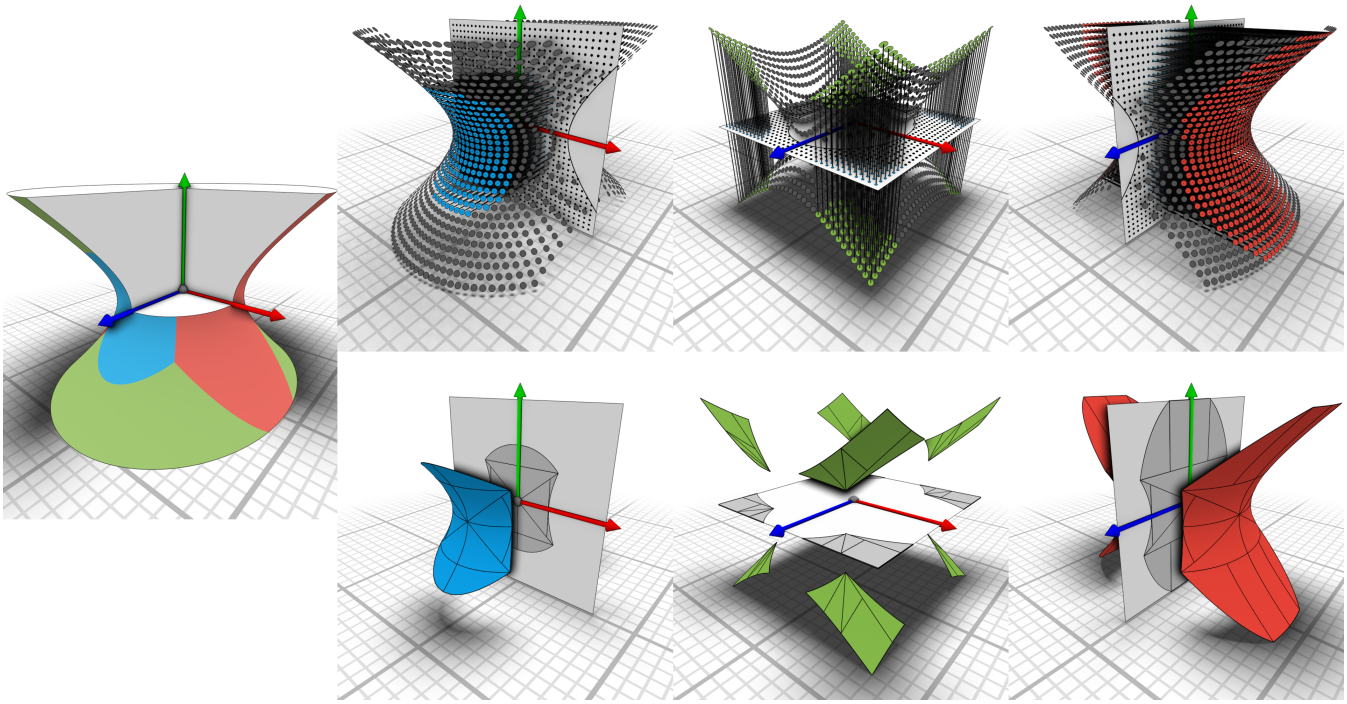
As v has unit length, $-||Ax_0 - b|| \leq (Ax_0 - b)^T v \leq ||Ax_0 - b||$ and $\min \lambda_i \leq v^T A v \leq \max \lambda_i$, where λ_i are the eigenvalues of A. Thus,

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d) (e) (f) (g)

Figure 3: *Quadric surfaces (blue) are quite flexible and, together with CSG operations, can represent a wide selection of objects. A few notable types of quadrics are shown: ellipsoid (a), elliptic cylinder (b), elliptic cone (c), one-sheet hyperboloid (d), two-sheet hyperboloid (e), two parallel planes (f), and the hyperbolic paraboloid (g). Using principal component analysis, we also define a quadric center (magenta) and quadric principal axes.*



(a) (b) (c) (d)

Figure 4: *Each quadric has a special coordinate system derived from a principal component analysis of its coefficient matrix, an exemplary elliptic hyperboloid is shown in (a). Every pair of principal axes defines a* sampling plane*, from which we can orthogonally project onto the quadric surface (b, c, d). Each surface point can be reached by multiple sampling planes with varying distortions. In the top row, a regular sampling of each sampling plane is shown. Samples are colored if they are the projection with the lowest distortion. The bottom row shows the analytical partitioning into low-distortion regions. Each region is decomposed into 2D basic regions that are later used as the starting point for our adaptive sampling scheme. These basic regions come in four different types: triangles, axis-aligned rectangles, "elliptic triangles", and "hyperbolic triangles".*

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

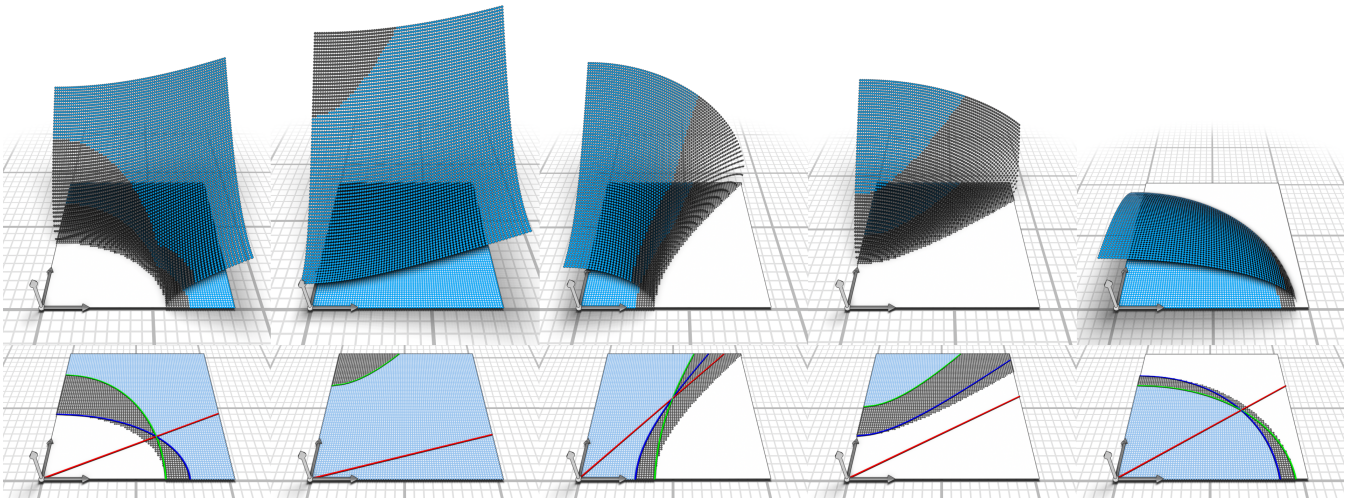


Figure 5: *On a sampling plane, the quadric surface is a heightfield of the form $h(x,y) = \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c}$. The non-degenerate cases are shown here and depend on the signs of the coefficients. From the 8 possible cases only 5 are depicted as one has no surface ($\alpha_x, \alpha_y, \alpha_c < 0$) and the two belonging to $\alpha_x < 0, \alpha_y > 0$ are mirrored versions of $\alpha_x > 0, \alpha_y < 0$. The top row shows samples on the heightfield: blue, if this is the lowest distortion amongst the three sampling planes, and gray otherwise. The bottom row shows the analytical construction of the low distortion regions: Pairwise equating of two normal components results in either a line (red) or a conic section (green, blue). The desired region is always bounded by (a subsection of) these curves and the coordinate axes.*

the following bounds hold:

$$Q(x_0 + t \cdot v) \leq Q(x_0) + t \cdot 2||Ax_0 - b|| + t^2 \cdot \max \lambda_i = Q^+_{x_0}(t) \quad (10)$$

$$Q(x_0 + t \cdot v) \geq Q(x_0) - t \cdot 2||Ax_0 - b|| + t^2 \cdot \min \lambda_i = Q^-_{x_0}(t) \quad (11)$$

For each bound, we have a simple quadratic equation in t. In particular, the bounds are independent of v. Depending on the sign of $Q(x_0)$, we can now define $d(x_0)$: if $Q(x_0) < 0$, then x_0 is inside the quadric and we can use the upper bound (Equation 10) to compute the smallest positive t where the $Q^+_{x_0}(t) = 0$, called t^+_0. Analogously, if $Q(x_0) > 0$ we use the lower bound to define $t^-_0 = \min\{t \mid t > 0 \wedge Q^-_{x_0}(t) = 0\}$. Thus, the function

$$d(x_0) = \begin{cases} t^+_0 & \text{if } Q(x_0) < 0 \\ t^-_0 & \text{otherwise} \end{cases} \quad (12)$$

is a conservative distance function.

A few 2D examples are shown in Figure 6. Empirically, $d(x_0)$ seems to be a good approximation of the true distance. In the supplemental material, we prove a desirable property: not only the absolute error, but also the relative error converges to zero close to the surface.

3.4. Heightfield Distortion

Due to the way we partition the heightfield domains in Section 4.1, the mapping from 2D to 3D has a maximum distortion of $\sqrt{3}$. Using this bound results in pessimistic assumptions during subdivision and distance evaluation, ultimately resulting in more subdivision levels and more emitted samples than strictly necessary. A more nuanced understanding of the local distortion leads to more accurate subdivision and less superfluous samples. Given the (unnormalized) heightfield normal $n(x,y)$ (Equation 14), the distortion $\delta(x,y)$ is given by

$$\delta(x,y) = \left(\frac{n(x,y)_2}{|n(x,y)|}\right)^{-1} = \sqrt{\frac{(\alpha_x + \alpha_x^2)x^2 + (\alpha_y + \alpha_y^2)y^2 + \alpha_c}{\alpha_x x^2 + \alpha_y y^2 + \alpha_c}}. \quad (13)$$

For the subdivision and distance estimates, we need the maximum distortion of a basic region. Assuming a constant distortion over such a region might work for small regions of relatively smooth quadrics, but is wrong in the general case. Fortunately, we were able to prove a useful property of $\delta(x,y)$: Given an axis-aligned rectangle R in a 2D quadrant, the extrema of $\delta(x,y)$ for $(x,y) \in R$ are assumed at one of the four vertices of R. Thus, we obtain efficient and useful distortion bounds for any 2D region by computing an axis-aligned bounding box and evaluating the distortion at the corners. The supplemental material contains the full derivation.

4. Adaptive Sampling

Blindly sampling the primitive surfaces and discarding samples that do not lie on the actual CSG surface leads to an efficient method with weak guarantees. Instead, we use the sampling planes defined in Section 3.2 to have a parametric description of the quadric surfaces via sets of heightfields. In Section 4.1, we partition the domains of the heightfields such that the quadric surface is represented by 2D regions with low-distortion heightfields (cf. Figure 4). In the domain of these heightfields, we define our adaptive sampling. Given a user-provided radius, our goal is to guarantee that

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

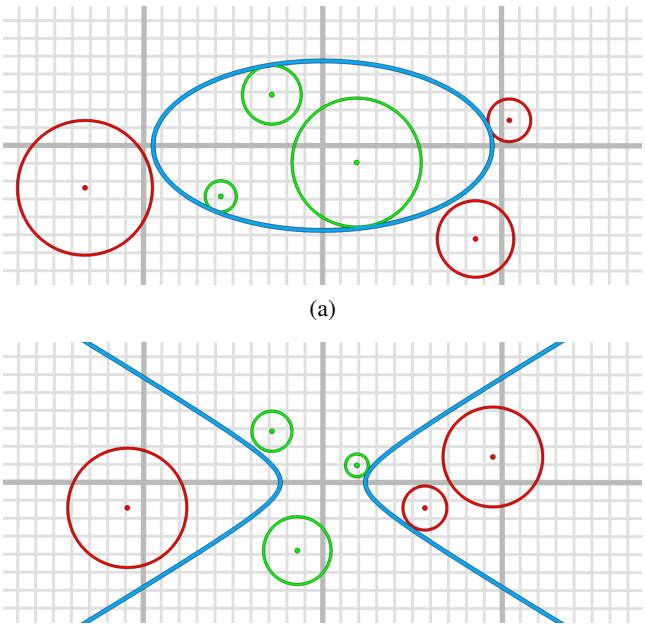


Figure 6: *We derive an extremely efficient conservative signed distance-to-quadric-surface function using second order Taylor expansion and the min-max theorem for eigenvalues. A few examples are shown here in the 2D setting, where the non-degenerate quadrics are ellipses and hyperbolas (blue). Negative signed distances are "inside" (green), positive means "outside" (red). These roles can be inverted by negating the quadric coefficient matrix and is used for the Boolean operation "not". Note that, while not exact, these distances are still quite useful in determining large regions of space that contain no surface intersection.*

for each point on the CSG surface, at least one sample is generated within the provided radius. Formulated the other way around: each generated sample "covers" a certain radius in the heightfield domain. Our adaptive subdivision and sampling approach of Section 4.2 ensures that each point in the heightfield domain is either covered by a generated sample or not part of the CSG surface. Simultaneously, large regions outside and even on the CSG surface can be skipped during the subdivision.

4.1. Low-Distortion Regions

While we could take any of the three sampling planes and use its heightfield as a parametrization, for most quadrics, this leads to problems in certain areas. Consider the leftmost example in Figure 5. In the gray region, the quadric surface perpendicularly intersects the sampling plane. Close to the intersection, a unit area on the quadric projects to almost zero area on the sampling plane. Thus, we would need smaller and smaller steps on the sampling plane to have a uniform sampling on the quadric surface, each generated sample would "cover" less and less of the 2D domain.

Given a sampling plane P_{ij} and the orthogonal principal axis u_k, we can compute the distortion using the normal vector. If p is a point sample in world space, then $|u_k^T N(p)|^{-1}$ measures the distortion if a local surface patch on P_{ij} is orthogonally projected onto the quadric surface at p. We can circumvent the distortion problem by using all three sampling planes and always choose the "best fitting" plane, i.e. the one with the lowest distortion. This is always the plane where $N(p)$ aligns best with the plane normal, the one maximizing $|u_k^T N(p)|$. As the principal axes form an orthonormal system, we can always find a plane with at most $\sqrt{3}$ distortion. This is reminiscent of how triplanar texturing (e.g. [Gol17]) constructs a low-distortion texture mapping based on the surface normal. The result is a set of 2D patches on the sampling planes that, when applying the heightfield function, partition the quadric surface as depicted by the different colored regions in Figure 4. These patches will be the starting point of our adaptive subdivision and sampling scheme described in Section 4.2. In the following, we only describe the positive quadrant, all other regions follow from symmetry.

We now want to find an analytic description of these low-distortion regions, i.e. the blue regions of Figure 5. This can be derived from the heightfield function $h(x,y) = \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c}$. The (unnormalized) surface normal $n(x,y)$ is given by

$$n(x,y) = \begin{pmatrix} -\alpha_x x \\ -\alpha_y y \\ \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c} \end{pmatrix}. \tag{14}$$

The heightfield is defined in the local coordinate space of the sampling plane P_{ij}. Thus, the blue low-distortion region is the set of points (x,y) where $|n(x,y)_2| \geq |n(x,y)_0| \wedge n(x,y)_2 \geq |n(x,y)_1|$, i.e. where "up" is the largest component of the heightfield normal. As $n(x,y)$ is continuous, the low-distortion region boundary is at $|n(x,y)_2| = |n(x,y)_0|$ or $|n(x,y)_2| = |n(x,y)_1|$. The first condition yields

$$\begin{aligned} |\alpha_x x| &= \sqrt{\alpha_x x^2 + \alpha_y y^2 + \alpha_c} \\ \Leftrightarrow \quad 0 &= (\alpha_x - \alpha_x^2) x^2 + \alpha_y y^2 + \alpha_c, \end{aligned} \tag{15}$$

which is an axis-aligned conic section centered at the origin. Similarly, the second condition yields

$$0 = \alpha_x x^2 + (\alpha_y - \alpha_y^2) y^2 + \alpha_c. \tag{16}$$

These two conics already properly describe the searched-for boundary, but are slightly unwieldy in practice. Thus, we further subdivide this region. Solving $|n(x,y)_0| = |n(x,y)_1|$ yields $\alpha_x x = \pm \alpha_y y$, i.e. two lines through the origin, one per quadrant, resulting in "octants". These three curves describe all points where two normal components have equal magnitude. They partition the plane into regions where the order of normal component magnitudes is fixed. If two of them intersect, they all intersect in the same points, which correspond to the normals $(\pm 1, \pm 1, \pm 1)^T / \sqrt{3}$. This construction is shown in the lower row of Figure 5.

If the coordinate axes are used for demarcation as well, we obtain a partitioning of the blue region into manageable subregions. No subregion can be bordered by more than one of the conics. An elliptic conic will always eventually intersect with the $|n(x,y)_0| = |n(x,y)_1|$ line. The hyperbolas can either intersect the x- or y-axis and they might intersect the $|n(x,y)_0| = |n(x,y)_1|$ line or not. For each case there is an "inside" and an "outside" sub-case, i.e. if the subregion with or without the origin is meant. The resulting 9 non-degenerate cases are shown in Figure 7.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

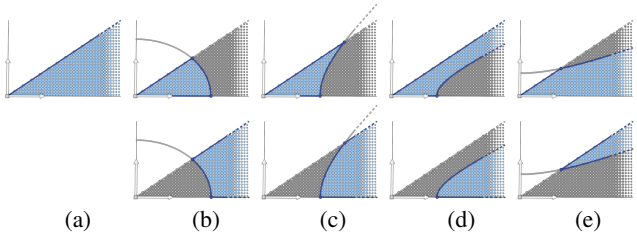


Figure 7: *The analytic partitioning of the low-distortion heightfield regions results in five cases, four of which have two sub-cases each. These are all combinations of how an ellipse or hyperbola that is axis-aligned and centered at the origin can intersect with a 2D octant (i.e. the region between an axis and a line through the origin).*

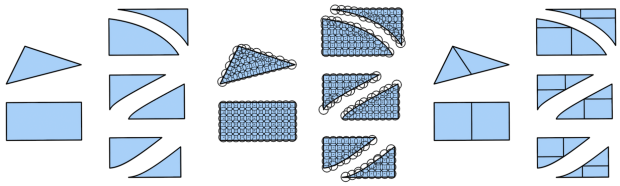


Figure 8: *Given a bounding box, the potentially infinite heightfield regions are decomposed into a set of basic regions: triangles, axis-aligned rectangles, "elliptic" triangles and "hyperbolic" triangles (a). During our adaptive sampling, we classify these regions. If a region is fully on the CSG surface, it is covered by splats (b). If it is fully outside or inside, it is discarded. Otherwise, we subdivide the region and re-classify (c).*

These subregions can still be infinite, the same way that quadric surfaces might be infinite. In practice, we provide a bounding sphere and no surface outside the sphere needs to be generated. We translate this bound into an axis-aligned bounding box (AABB) in the local coordinate system of the sampling plane. The subregions are then intersected with this AABB and decomposed into a set of basic regions, chosen for ease-of-use in our adaptive sampling. Each basic region can be either (cf. Figure 8):

1. a triangle
2. an axis-aligned rectangle
3. an "elliptic triangle", an axis-aligned right triangle where the diagonal is part of an ellipse centered at the origin
4. a "hyperbolic triangle", a triangle where one side is part of a hyperbola centered at the origin and opening either towards the x- or y-axis

While this construction might seem unnecessarily complex, it serves an important purpose: We now have a decomposition of the quadric surface into heightfields bound by reasonably simple regions. Each heightfield has the guarantee that within the defined region, the mapping from 2D to 3D has a maximum distortion of $\sqrt{3}$. These properties are important for the adaptive sampling in Section 4.2. An example quadric and its decomposition into basic regions is shown in Figure 4.

4.2. Adaptive Sampling

This section describes the core of our method: the adaptive sampling with strong coverage guarantee. Our CSG surface is defined by a set of quadrics and Boolean operations between them (cf. Section 5). Each quadric surface is decomposed into a set of low-distortion 2D basic regions with heightfield functions (cf. Section 4.1 and Figure 4).

Given a quadric Q and a 2D region on a sampling plane, the goal of our adaptive sampling is to generate samples in this region and thus, via the heightfield, on the quadric surface. Additionally, each sample must be on the CSG surface and each point of the region must either be "covered" by a sample (i.e. within a user-defined world-space radius) or the point must be proven to not be part of the CSG surface.

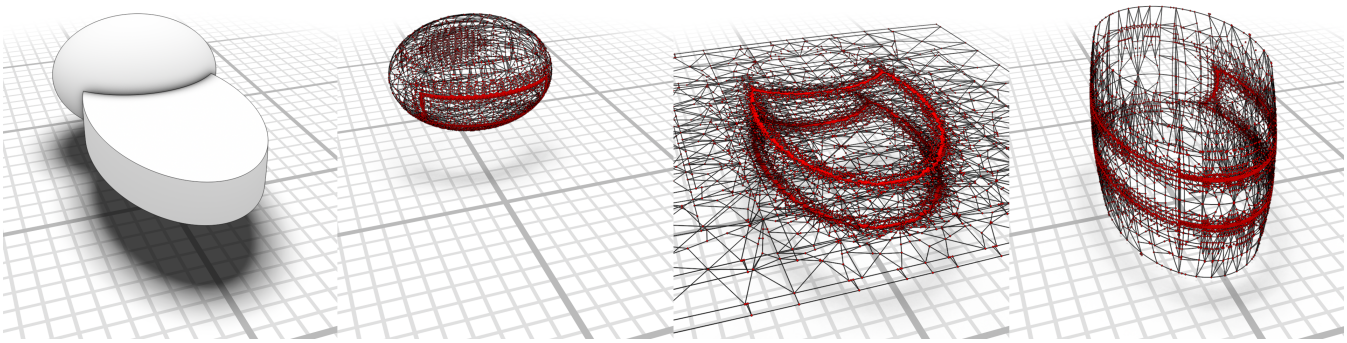
An essential tool to achieve this efficiently is Equation 22 in Section 5, which defines the quadric-relative conservative signed distance function $d_Q(S,p)$: For quadric Q of CSG surface S, this function provides classification for points p (that lie on Q) and their neighborhood. It has the following property: p is only part of the CSG surface if $d_Q(S,p)$ is negative. Furthermore, the same classification is valid for all points on Q within radius $|d_Q(S,p)|$ of p.

Given a basic region, i.e. one of the shapes depicted in Figure 8, we evaluate $d_Q(S,p)$ on all corners of the region. These distances are in world space and we use our distortion guarantees to project the radius into the 2D domain. For this, we can either use the conservative $\sqrt{3}$ value or the distortion bounds described in Section 3.4. The result is a set of circles at the corners of the region. Inside each circle, all points have a known classification provided by the sign of $d_Q(S,p)$. If the circles cover the whole region, the region can be classified as belonging (as a whole) to the CSG surface or not. In the former case, we cover the region with samples (cf. Figure 8 (b)). Otherwise, no samples are generated. If the circles do not cover the whole region, we subdivide according to a simple longest-edge strategy (cf. Figure 8 (c)). This whole strategy is then applied recursively.

There is one special case: If any corner is part of the CSG surface ($d_Q(S,p) < 0$) and the region is small enough to be covered by a single sample at that corner, we terminate regardless of $|d_Q(S,p)|$. Generating that sample will already provide our guarantee without classifying the rest of the region.

Figure 9 shows the subdivision pattern on an example CSG object. Using the conservative distance, this adaptive sampling only subdivides where multiple quadrics interact. The subdivision quickly terminates when the region actually belongs to the CSG surface, i.e. when d_v is negative. In that case, we can stop at the latest when the region is smaller than a single splat as the rendering ensures proper pixel-perfect covering. Outside but close to the CSG surface, usually many subdivisions are required to prove that the region is actually completely outside and does not contain small surface parts. In practice, a threshold can be employed: discard regions that are not yet classified but are smaller than for example $1/10$th of the splat size. This speeds up termination but can theoretically result in small holes (which we did not observe in practice, except when explicitly forced).

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d)

Figure 9: *Subdivision pattern of our adaptive sampling on an actual CSG object (a). The (recursively subdivided) basic regions on each quadric surface are shown (b,c,d). Each red point is an evaluation of our quadric-relative conservative signed distance function. With its help, large regions can be classified at once and subdivision is only needed in the vicinity of feature curves.*

5. CSG

The input is a directed acyclic graph (DAG) representing the CSG object. Leaf nodes are annotated with a primitive, i.e. a quadric. Inner nodes are Boolean operations such as union, intersection or difference. Operations are variadic where sensible. Additionally, each graph edge can be annotated with a 4×4 transformation matrix.

We start by computing a canonical form of the CSG graph: A union of intersections, also known as disjunctive normal form (DNF). Sub-graphs are duplicated as needed, transformations are pushed into leaf nodes and are applied to the quadric matrix. Given a quadric coefficient matrix $Q \in \mathbb{R}^{4\times4}$ and a transformation matrix $M \in \mathbb{R}^{4\times4}$, the transformed quadric Q' is obtained via

$$Q' = M^{-T} Q M^{-1}. \tag{17}$$

The represented object S is thus simply

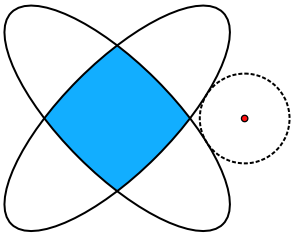
$$S = \bigcup_i \bigcap_j Q_{ij}, \tag{18}$$

where Q_{ij} are the leaf quadrics with transformations applied. This transformation is quite similar to [GMTF89].

This canonicalization serves three purposes. First, a tree form is required as each path from root to leaf creates a unique instance of the primitive, and thus of its surface, anyways. For each instance, different regions of the primitive surface might belong to the CSG surface, which means that samples cannot be reused across instances. Secondly, the conservative signed distance function that we construct for the adaptive sampling in Section 4.2 currently relies on the normal form. And finally, the local clipping employed during rendering in Section 6 is highly efficient as it only needs to know about quadrics from the same intersection term.

Let $d(Q,p)$ be the conservative signed distance function defined in Section 3.3. This distance is positive, if the point p is outside the quadric Q, negative if inside, and zero if $Q(p) = 0$. It is conservative, because $|d(Q,p)|$ is a lower bound of the minimal distance from p to the quadric surface. This can be extended to a conservative signed distance to the whole CSG surface:

$$d(S,p) = \min_i \max_j d(Q_{ij},p) \tag{19}$$



Distance-to-csg-surface using min and max can be inexact, even if distance-to-primitive is exact.

Note that, in general, this distance is only a conservative approximation, even if the individual $d(Q_{ij},p)$ were exact. While this function can accurately classify points as "inside", "outside", or "on surface", it is not particularly efficient for our purposes. When generating samplings for a quadric Q_{ij}, we already know that the samples lie on the surface of Q_{ij}. They can still lie inside the CSG object (e.g. if inside another unioned primitive) or outside of it (e.g. if cut away by an intersection). However, we gain no further information if the sample actually belongs to the CSG surface. $d(S,p)$ is zero and we have no idea if $d(S,p+\varepsilon)$ is zero or not. This would make the subdivision scheme of Section 4.2 inefficient, as we would have to subdivide surface regions until the parts can be covered by a single splat each.

However, given the CSG tree in normal form, we can design a quadric-relative conservative signed distance function $d_{Q_{ab}}(S,p)$ that can classify large regions of a quadric surface as "samples must be kept" and "samples must be discarded" and requires subdivision only in regions where different quadrics interact. Consider the "point of view" of a quadric Q_{ab}, i.e. the b-th intersection of the a-th union of the DNF. If we generate a sample on Q_{ab}, it only belongs to the CSG surface if it is not inside any other intersection term (and thus inside the CSG object, cf. Figure 10 (c,d)) and if it is not outside any other intersection term of its own intersection term (and thus outside the CSG object, cf. Figure 10 (a,b)). The distance to the other intersection terms is

$$-\min_{i\neq a} \max_j d(Q_{ij},p). \tag{20}$$

It is negative, because we discard positive values and want to discard p if it is inside any other intersection term. The distance to the other intersections of the same intersection term is simply

$$\max_{j\neq b} d(Q_{aj},p). \tag{21}$$

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

Combining these yields our desired distance function:

$$d_{Q_{ab}}(S,p) = \max\left(-\min_{i\neq a}\max_{j} d(Q_{ij},p),\ \max_{j\neq b} d(Q_{aj},p)\right) \quad (22)$$

This function has our desired properties:

- It is positive if p does not belong to the CSG surface and no point of Q_{ab} within $|d_{Q_{ab}}(S,p)|$ belongs to it either.
- It is negative if p belongs to the surface and all points of Q_{ab} within $|d_{Q_{ab}}(S,p)|$ also belong to it.

With this function, our adaptive sampling of Section 4.2 can quickly skip over discarded regions and fill completely covered ones. Note that while $d_{Q_{ab}}(S,p)$ appears to have different meanings depending on its sign, it has a clear interpretation in terms of classification: The sign of $d_{Q_{ab}}(S,p)$ encodes if p belongs to the CSG surface or not, while $|d_{Q_{ab}}(S,p)|$ is a radius in which all points of Q_{ab} have the same classification as p.

6. Rendering

Each generated sample lies on the actual CSG surface. Furthermore, the adaptive sampling is designed for a splat-based rendering: Our approach guarantees that for each CSG surface point, at least one sample is generated within the splat radius r and on the same primitive. Thus, the following method yields a pixel-perfect rendering of the CSG surface (cf. Figure 11):

1. For each sample s on quadric Q_{ab}, render a screen-aligned quad of size $2r \times 2r$ (world-space).
2. In the fragment shader, compute the intersection p of the view ray with Q_{ab} (Equation 5), favoring the intersection closer to s and discarding those farther away than r from s.
3. Evaluate $Q_{aj}(p)$ for all quadrics $Q_{aj}, j \neq b$ that belong to the same intersection term as Q_{ab}. Discard the fragment if any $Q_{aj}(p)$ is positive. Only Q_{aj} that are "close" to the sample need to be considered, i.e. where $|d(Q_{aj},s)| < r$.

Step (1) ensures that the fragment shader is executed for each fragment that potentially belongs to a sphere of radius r around the sample s. Then, Step (2) performs a *local quadric ray-tracing* to compute the exact surface point. Finally, Step (3) implements a *local quadric clipping* to make sure that no point outside of the CSG surface is visible. This is the only potentially expensive step in the rendering, but is very cheap in practice as it can be aggressively optimized: The clipping only needs to clip against quadrics of the same intersection term as those would result in visible fragments outside the CSG surface. Some rendered points might be inside the CSG surface, but as our rendering is watertight, this is not visible. We furthermore reduce the set of required tests by only considering quadrics that might intersect with the (s,r)-sphere. In practice, most samples actually need zero clipping. Close to CSG surface edges we need one clipping quadric, close to corners two. This number can increase in the vicinity of small features and thin regions but is still low in general. The per-quadric clipping test is $p^T Qp > 0$, which is cheap.

In our implementation, we make sure to minimize the amount of redundant data. Apart from the splats, we use two additional buffers:

1. A *quadric buffer* that contains an entry for each unique quadric: matrix coefficients (10 floats as the matrix is symmetric) and a color (for visualization purposes).
2. A *clip indirection buffer* that contains indices for quadrics used in the local clipping.

The samples are rendered as an instanced quad. The per-instance data is the 3D sample position, the index of the quadric that this sample belongs to and two integers that indicate the quadrics used in the clipping: a start index and a count into the *clip indirection buffer*. When assembling the splat data on the CPU, the clipping data is deduplicated, i.e. each unique set of clipping quadrics is only stored once in the *clip indirection buffer* and used by many samples.

For accurate shading and interaction with other renderings, we compute the surface normal using Equation 4 and adjust the depth using `gl_FragDepth`. This usually disables the early depth test, but as the updated depth value is always behind the rendered geometry, a conservative depth specification can be used. Our rendering integrates smoothly into a traditional rasterization pipeline. For that purpose, it can be thought of as a special decal or particle rendering technique. All typical local shading techniques, including physically-based shading, can be used due to accurate depth and normals.

6.1. Overdraw Reduction

The described rendering strategy results in the correct rendering, but is unnecessarily conservative: The screen-aligned quad is used to cover a sphere around the sample to capture the quadric surface. However, most of the time the quadric is smooth and the splat small, thus almost flat. Especially when viewing from a shallow angle, this leads to extreme overdraw. Figure 13 shows the overdraw and the effect of the following optimizations.

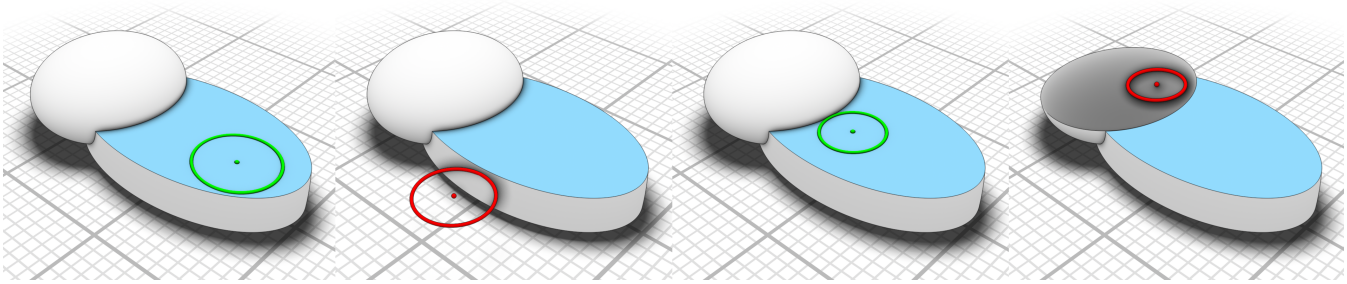
Instead of a screen-aligned quad, we can use our distortion estimate (cf. Section 3.4) to construct a local surface-aligned box around the splat surface. The first box direction is given by the surface normal $n(x,y)$ (Equation 14), the other two are arbitrary tangent directions. The extent in tangent direction is given by the splat radius r, the extent in normal direction is

$$r \cdot \left(\max \delta(x \pm r, y \pm r) - \min \delta(x \pm r, y \pm r)\right). \quad (23)$$

When smaller than an epsilon, we can safely render the splat as a single surface-aligned quad, otherwise we render the box. If we are only interested in the outer appearance, we can omit the box side in negative normal direction and enable back-face culling. Figure 12 shows an example of these splat bounds, while Figure 13 (b) and (c) show the drastic overdraw reduction achieved. We derive this bound in the supplemental material.

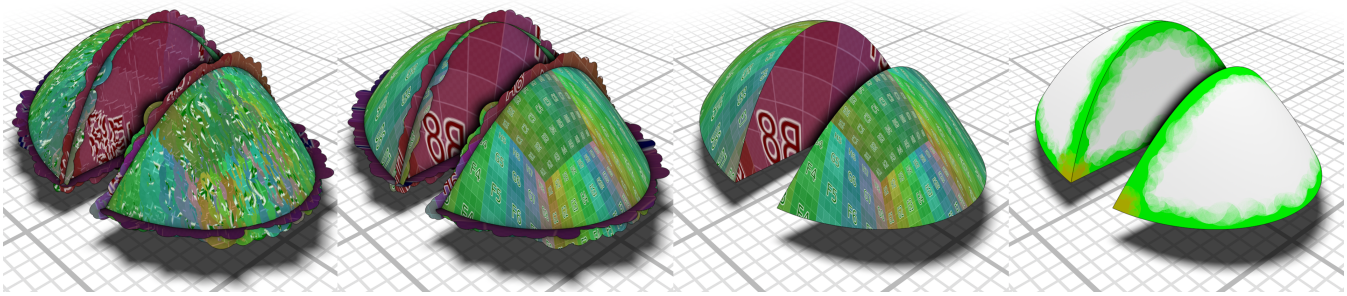
Replacing the $\sqrt{3}$ estimate in the subdivision and distance estimate by the distortion measure from Section 3.4 also results in less superfluous samples and thus less overdraw (Figure 13 (d)). Finally, close to interaction between quadrics, many subdivisions are needed and small regions occur that are similarly sized to splats or smaller. In those cases, we can reduce the radius of the emitted sample without increasing the number of samples and without any visual change, but nevertheless reducing overdraw (Figure 13 (e)).

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d)

Figure 10: *Our quadric-relative conservative signed distance function makes it possible to quickly classify large regions in our adaptive sampling. In this example, the blue quadric surface is being sampled and the function value is shown as a color-coded radius around the sample position. If the function is negative (green), all samples within the radius are guaranteed to lie on the CSG surface. If it is positive (red), those samples can be discarded instead. Due to our normal form, the sign change, or decision border, happens either when being cut away by a quadric within the same intersection (a,b) or by being inside a different term of the root union (c,d). The displayed distance uses our conservative distance-to-quadric from Section 3.3 which, while not exact, is still practical.*



(a) (b) (c) (d)

Figure 11: *Effect of our rendering on an intersection of three quadric with non-negligible curvature. An environment map reflection is added to improve the legibility of surface smoothness. (a) shows a classical splat rendering of our generated surface samples. In (b), we perform* local quadric ray-tracing *to project the splat onto the quadric surface. For accurate reconstruction of features, we add* local quadric clipping *in (c), which results in a hole-free pixel-perfect visualization of the CSG surface. The number of quadrics that must be clipped against is theoretically unbounded, but due to our aggressive optimization, it is quite low in practice. This can be seen in (d), where the number of clipping quadrics is color-coded: white means 0, green is 1, orange is 2, the maximum in this example.*

Taken together, these optimizations lead to a significant reduction in overdraw and a 10× decrease of render cost. On the CPU side, they actually lead to a slightly faster sampling process as the more accurate distortion estimate reduces the required subdivision depth and the number of generated samples.

7. Evaluation

7.1. Characteristics and Performance

All benchmarks were performed on a 3.60 GHz Intel Core i9-9900K (4.8 GHz single core turbo) and an NVIDIA RTX 2080 Ti. Unless otherwise noted, all timings are on a single CPU thread. All algorithms were implemented in C++ and OpenGL, compiled with Clang 7 using the flags `-O3` and `-march=native`.

The runtime performance of our method has two components. First, samples have to be generated for a given CSG model. This is a pre-processing step and the samples are view-independent. Afterwards, the model can be rendered from any view using the splat-based renderer of Section 6. As the sampling and the rendering are two separate contributions, there are evaluated individually.

Table 1 summarizes the key performance metrics that determine the sampling throughput. Especially all basic formulas needed for the subdivision and the covering can be evaluated multiple hundreds of millions of times per second per CPU core. In practice, this translates to a few ten million samples per second on simple CSG models and a few million per second on more complex ones. The number of samples depends on the CSG surface area and the user-provided sample radius, i.e. the splat size. We can always guarantee correct rendering, even for low numbers of samples. However, large splats usually lead to more overdraw and more "wasted" splat area, typically reducing the rendering performance. In our experience, a few hundred thousand samples provide a good tradeoff for the models we tested. This means, that CSG models can be edited in near real-time or at least interactively.

Note that the system described here does not include a spatial acceleration structure on the CSG-tree level. This is left as fu-

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

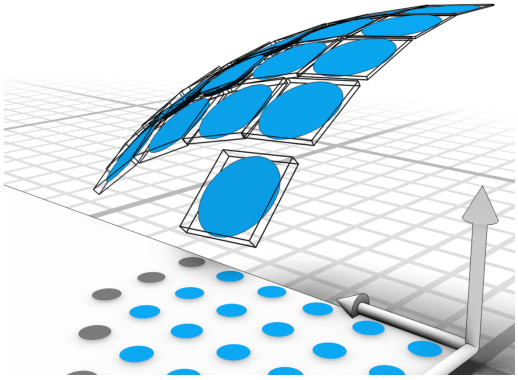


Figure 12: *Rendering with surface-aligned* flat *splats is not correct a priori, as the surface curvature might lead to holes or missed silhouette pixels. Instead, we derive surface-aligned splat bounding boxes based on local distortion bounds. These bound the* curved *splats. Rasterizing these bounding boxes and performing local quadric ray-tracing guarantees a hole-free result and a correct silhouette. In this figure, splats are not tight for illustration purposes.*

ture work and beneficial if objects with many separate or loosely-interacting parts are modeled. We do not foresee any problem adding it to the evaluation of Equation 22.

While we did not implement and benchmark a multi-core version, it is rather straightforward to implement and we expect near-linear speed-up. At least each basic region (cf. Section 3.2) can be subdivided and sampled in parallel. If this is too coarse, a work-stealing method based on the subdivision trees is appropriate.

The rendering is splat-based and can be characterized with two metrics: per-fragment cost of a single splat and the overdraw factor.

For each splat fragment, we perform local ray-tracing against a single quadric and a variable number of quadric evaluations for the clipping (cf. Figure 11). Most splats require no clipping, at the intersection of two quadrics samples need one clipping quadric, at corners typically two. In general, all quadrics of the same intersection term are collected within the splat radius, except for the one that the splat belongs to. Thus, structures smaller than the splat size can show a higher number of clipping quadrics. On our high-end graphics card, a single screen-filling (1920×1080) splat costs 0.10 ms with no clipping, and 0.12 ms with 5 clipping quadrics. An interesting perspective is that the splats can be seen as a kind of localized "CSG cache". Instead of evaluating the complete CSG tree for each view ray, the splats act as a spatial acceleration (rays are only generated for splat fragments) and a computation cache (rays have access to a pruned CSG tree via the local clipping).

Hole-free splat-based renderers tend to have non-negligible overdraw. Our technique has several optimizations to reduce overdraw (cf. Section 6.1, Figure 13, and Figure 14). In our experiments, we still have about 3–4$\times$ overdraw, i.e. on average, each pixel is covered by 3–4 fragment shader invocations. Complex and layered models can have higher overdraw, though most models require less than 1 ms to render on Full HD. As our rendering is really close to raster-based particle or decal rendering, traditional techniques like frustum, occlusion, or hierarchical z-buffer (HiZ) culling can be applied. This would lead to even more predictable frame times and the impact of high depth complexities would be minimized. Close-up viewing of the CSG model can currently be roughly twice as expensive as indicated in Table 2, as the close-up surface is effectively a second screen-filling layer. While still quite fast, proper occlusion culling would eliminate this overhead.

Note that our rendering is currently optimized for opaque rendering from the outside. To correctly clip the inside, other terms of the root union must be considered as well, leading to more complex clipping code in the shader. The impact on the rendering performance would probably be low, as, similar to the current clipping, only nearby quadrics would need to be added to the clipping.

Table 2 presents a few examples with varying splat sizes and the resulting statistics and characteristics. The number of distance queries, i.e. evaluations of $d_{Q_{ab}}(S, p)$, is relatively high, as we need many subdivisions where quadrics interact (cf. Figure 9). A sampling-plane-local acceleration structure to share distance queries between different branches of the subdivision tree might help, but incur non-negligible overhead themselves. We leave this optimization for future work. The number of clip indices, i.e. the size of the *clip indirection buffer* (cf. Section 6), is low enough that we expect that quadric buffer and clip indirection buffer are always in cache. After sampling, our rendering is extremely cheap with less than 1 ms for all models. This is roughly proportional to the number of fragments and we measured about 4$\times$ the cost when rendering in 4K.

7.2. Comparison

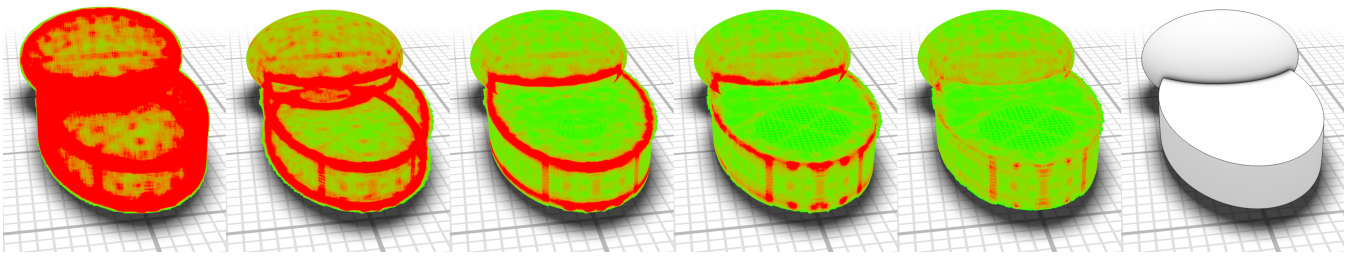
Our sampling-based CSG represents a new approach to CSG computation and visualization, with its own unique set of trade-offs.

Explicit computations are typically significantly more expensive than our sampling and often require linear approximations, e.g. triangle meshes or BSP trees. However, after the computation, rendering tends to become trivial. In contrast, pure rendering methods like ray-tracing or depth-peeling-based approaches often require little to no preprocessing but tend to be more expensive than our approach. Table 2 contains a comparison with a raytracing approach: for each intersection term of Equation 18, all quadric intersections are collected in a local buffer and sorted by depth. Counting number of times the quadrics are entered and left finds the correct CSG surface. The result is the closest surface of all terms of the root union. Except for very simple cases, our rendering is faster and relatively independent of the CSG complexity.

Most rendering methods require spatial acceleration structures or bounded primitives to be efficient. Our primary focus is on objects with complex interaction of curved quadrics, where these optimizations are less applicable. We leave adding a spatial acceleration structure for evaluating the distance function to future work and expect this straightforward to integrate.

CSG of curved surfaces is often approximated by isosurface extraction of implicit functions. Even with feature-preserving methods, these approximations tend to be poor in the presence of thin structures. In contrast, our method provides strong guarantees for

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d) (e) (f)

Figure 13: *When rendering the samples as view-aligned splats covering a sphere, overdraw becomes an issue (a). Multiple optimizations lead to a drastic reduction of wasted bandwidth: First, we use surface-aligned boxes and splats (b). This enables effective back-face culling (c). We also derive an efficient lower bound for the splat distortion, leading to less sample overlap (d). Finally, splat size can be reduced in highly subdivided regions without increasing the number of samples, resulting in less overdraw where multiple quadrics intersect (e). The CSG surface is shown in (f) and while all variants produce the same visual results, (e) is more than* 10 *times faster than (a). Overdraw is color coded, green is 1 and red is 30+.*

operation	**throughput**	**timing**
evaluation		
quadric value $Q(x)$	315 000 000 / s	3.17 ns / op
quadric normal $N(x)$	220 000 000 / s	4.54 ns / op
sampling plane heightfield value $h(x,y)$	1 000 000 000 / s	1.00 ns / op
heightfield distortion $\delta(x,y)$	625 000 000 / s	1.60 ns / op
heightfield min distortion $\min_R \delta(x,y)$	150 000 000 / s	6.70 ns / op
quadric-line intersection	53 000 000 / s	18.87 ns / op
conservative signed distance-to-quadric $d(x_0)$	130 000 000 / s	7.69 ns / op
sampling		
single quadric, no subdivision	135 000 000 / s	7.41 ns / sample
single quadric, with subdivision	40–70 000 000 / s	14–25 ns / sample
simple CSG surface (Figure 9)	10–50 000 000 / s	20–100 ns / sample
complex CSG surface (Figure 1)	0.5–2 000 000 / s	500–2000 ns / sample

Table 1: *Typical performance numbers of our method on a single* 4.8 GHz *CPU core. While many optimizations, especially for large, complex CSG models, are still untapped, the current performance is already enough for interactive modification of the CSG object. Note that our samples cover the whole CSG surface and do not require recomputation if the camera view changes. When sampling from a CSG surface, the throughput also strongly depends on the splat size: a lower size results in fewer samples and less absolute time, but the per-quadric overhead factors in more significantly.*

the sampling and the rendering. All samples are guaranteed to lie on the CSG surface and for each point on the CSG surface, there is at least one sample on the same quadric within the user-provided splat radius. Together with the splat-local quadric raytracing and clipping, we guarantee hole-free pixel-perfect rendering. The sample guarantee ensures that our method is also useful in a non-rendering context, e.g. for collision detection. Figure 15 shows an object with small, curved features with sharp tips. Even though the splat size is large in comparison, our method finds, samples, and renders those features correctly.

8. Limitations and Future Work

Our presented method uses quadric surfaces as primitives. While this already includes boxes, spheres, ellipsoids, cones, cylinders, and affine transformations thereof, many important curved surfaces cannot be exactly represented by quadric surfaces. Thus, a straightforward avenue for future research would include extending our system for other primitives types, especially freeform surfaces. Generating samples from these surfaces is usually not a problem and the challenge is to find fast and useful conservative signed distance functions.

A potentially more serious limitation is our conversion to disjunctive normal form. Depending on the input CSG graph, this can lead to exponentially many terms in the worst case. While we did not encounter that in our tests, it might still be worthwhile to explore working on the CSG tree directly. As argued in Section 5, the tree form (in contrast to a graph) is necessary as each primitive surface point should correspond to at most one point on the CSG surface. However, the normal form might not be necessary if efficient "quadric-relative conservative signed distance functions" can be formulated that respect the CSG tree. Similarly, the local quadric clipping need to be tree-aware and could probably not stay in its current, highly-efficient form.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

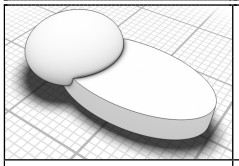
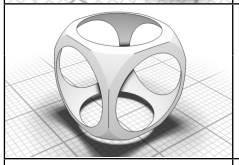
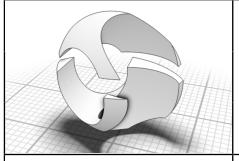
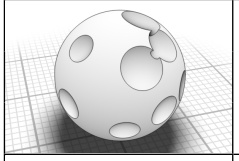
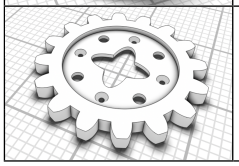
CSG object	quadrics	splat size	samples	dis. queries	#clip	sampling time	render time ours	render time raytracing
	3	large	5454	22372	2	1.51 ms	0.34 ms	0.36 ms
		medium	23452	47336	2	3.39 ms	0.23 ms	
		small	249014	100398	2	8.51 ms	0.27 ms	
	5	large	8112	103296	52	7.63 ms	0.27 ms	1.14 ms
		medium	35904	215040	23	16.45 ms	0.18 ms	
		small	329856	576048	17	66.87 ms	0.26 ms	
	5	large	9465	52578	52	4.38 ms	0.31 ms	0.87 ms
		medium	27882	103908	31	8.71 ms	0.28 ms	
		small	205075	254592	25	23.47 ms	0.26 ms	
	21	large	7134	30802	78	6.26 ms	0.45 ms	0.72 ms
		medium	31940	85744	37	18.01 ms	0.39 ms	
		small	104972	171958	31	37.91 ms	0.33 ms	
	31	large	28432	196298	51	36.63 ms	0.96 ms	4.43 ms
		medium	91895	325705	34	62.23 ms	0.87 ms	
		small	184533	415847	29	83.36 ms	0.69 ms	
	80	large	10176	194024	299	94.47 ms	0.32 ms	9.15 ms
		medium	141200	558582	244	292.35 ms	0.34 ms	
		small	655727	1182464	236	623.92 ms	0.47 ms	

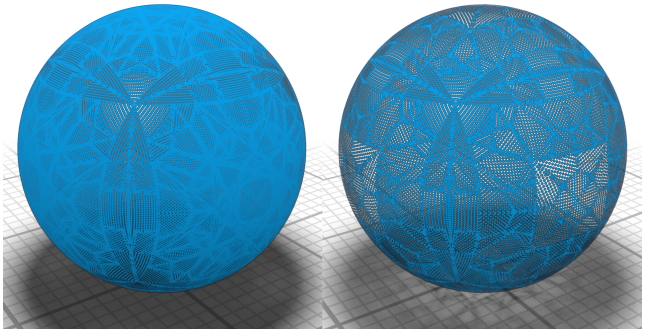
Table 2: *Examples of varying complexity with three different splat sizes and the resulting number of samples, distance queries, clip indices, and performance statistics. Sampling is a view-independent pre-processing step done on the CPU (single core,* 4.8 GHz*), while our splat-based rendering is done on the GPU (NVIDIA RTX 2080 Ti). The timings are given for* 1920×1080 *from a view where the objects are approximately screen-filling. We compare our rendering with a fragment-shader global raytracing.*

Finally, there are still many potential optimizations untapped. We expect a near-linear speed-up when utilizing more cores as our method can be easily parallelized at the level of basic regions or via a work-stealing approach. In an interactive modeling setting, a partial update would be useful and is probably easily realizable by computing regions that might have been invalidated and only recompute the affected sample. Additionally, the CSG tree itself would benefit from an optimization pass that tries to simplify superfluous operations, such as intersections that result in zero surface, differences that do not change the input, or unions where subtree is a subset of the other. For models with many loosely interacting quadrics, a spatial acceleration structure and pre-culling on the CSG tree would be appropriate.

9. Conclusion

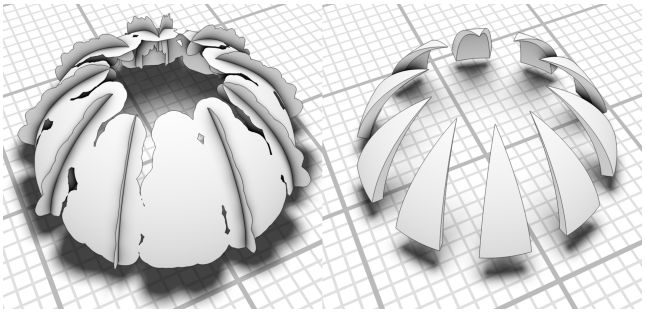
In summary, we present a new method for working with objects defined by CSG graphs, in particular where the primitives are quadric surfaces. With our approach, it is possible to create point samples from the CSG surface without computing an explicit surface representation. We derive an efficient conservative distance-to-quadric-surface function and, together with a low-distortion parametrization of the quadric surface, we can formulate an adaptive subdivision-based sampling method that can quickly classify large regions and only subdivides close to where multiple quadrics intersect. The generated samples satisfy a strong guarantee: given a user-provided splat size r, each point on the CSG surface has at least one sample within radius r that lies on the same quadric. This leads to a highly efficient watertight rendering of the CSG surface using splats with local quadric ray-tracing and clipping. The result is a sampled representation of the CSG surface that can be used for pixel-perfect

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b)

Figure 14: *Using the distortion estimate* $\min_R \delta(x,y)$ *results in a more even distribution of splats across the quadric surface. Without it, the worst-case distortion of* $\sqrt{3}$ *has to be assumed, leading to a higher density of samples when the surface points towards the quadric principal axes, i.e. has a low distortion. The images show splats with* 50% *radius, (a) assumes worst-case distortion, (b) uses our distortion estimate. The basic regions are slightly subdivided and higher densities can be observed at the seams due to our conservative covering. In this example, (b) can cover the same surface with* 36% *fewer samples.*



(a) (b)

Figure 15: *An intersection of eight quadrics that demonstrates that our approach finds and preserves sharp features, even those significantly smaller than the splat radius. Left shows splats without clipping, right with clipping.*

rendering without the usual discretization artifacts associated with point clouds. Due to the strong guarantee, our samples can also be used for some geometric operations, such as collision detection or distance computation. The sampling process achieves a few million samples per second per CPU core, which is fast enough to allow interactive modification of the CSG object. Rendering typically costs less than 1 ms and is largely independent of the model complexity.

Acknowledgements

The authors thank Aaron Grabowy for help with prototyping and fruitful discussions. This project was funded by the European Regional Development Fund within the "HDV-Mess" project under the funding code EFRE-0500038.

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.

References

[AS14] ANDREWS J., SEQUIN C.: Type-constrained direct fitting of quadric surfaces. *Computer-Aided Design and Applications 11* (01 2014). 3

[BF09] BERNSTEIN G., FUSSELL D.: Fast, exact, linear Booleans. In *Proceedings of the Symposium on Geometry Processing* (Goslar, DEU, 2009), SGP '09, Eurographics Association, p. 1269–1278. 3

[BGF15] BARKI H., GUENNEBAUD G., FOUFOU S.: Exact, robust, and efficient regularized Booleans on general 3D meshes. *Computers and Mathematics with Applications 70*, 6 (2015), 1235–1254. 3

[CK10] CAMPEN M., KOBBELT L.: Exact and robust (self-)intersections for polygonal meshes. *Comput. Graph. Forum 29* (05 2010), 397–406. 3

[CLSA20] CHERCHI G., LIVESU M., SCATENI R., ATTENE M.: Fast and robust mesh arrangements using floating-point arithmetic. *ACM Trans. Graph. 39*, 6 (Nov. 2020). 3

[CWJ02] CHANGHE TU, WENPING WANG, JIAYE WANG: Classifying the nonsingular intersection curve of two quadric surfaces. In *Geometric Modeling and Processing. Theory and Applications. GMP 2002. Proceedings* (2002), pp. 23–32. 3

[CWM*14] CHOI Y.-K., WANG W., MOURRAIN B., TU C., JIA X., SUN F.: Continuous collision detection for composite quadric models. *Graphical Models 76*, 5 (2014), 566–579. Geometric Modeling and Processing 2014. 3

[dALJ*15] DE ARAÚJO B. R., LOPES D. S., JEPP P., JORGE J. A., WYVILL B.: A survey on implicit surface polygonization. *ACM Comput. Surv. 47*, 4 (May 2015). 3

[DFR17] DOUZE M., FRANCO J., RAFFIN B.: QuickCSG: Fast arbitrary Boolean combinations of N solids. *CoRR abs/1706.01558* (2017). arXiv:1706.01558. 3

[DLLP08] DUPONT L., LAZARD D., LAZARD S., PETITJEAN S.: Near-optimal parameterization of the intersection of quadrics: I. the generic algorithm. *Journal of Symbolic Computation 43*, 3 (2008), 168–191. 3

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. 3

[GMTF89] GOLDFEATHER J., MONAR S., TURK G., FUCHS H.: Near real-time CSG rendering using tree normalization and geometric pruning. *IEEE Computer Graphics and Applications 9*, 3 (1989), 20–28. 3, 9

[Gol17] GOLUS B.: Normal mapping for a triplanar shader. https://medium.com/@bgolus/normal-mapping-for-a-triplanar-shader-10bf39dca05a, 2017. [Online; accessed 25-May-2021]. 7

[HKM07] HACHENBERGER P., KETTNER L., MEHLHORN K.: Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. *Computational Geometry 38*, 1 (2007), 64–99. Special Issue on CGAL. 3

[HR05] HABLE J., ROSSIGNAC J.: Blister: GPU-based rendering of Boolean combinations of free-form triangulated shapes. *ACM Trans. Graph. 24* (2005), 1024–1031. 3

[HR07] HABLE J., ROSSIGNAC J.: CST: Constructive solid trimming for rendering BReps and CSG. *IEEE Transactions on Visualization and Computer Graphics 13* (2007). 3

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Trans. Graph. 21*, 3 (July 2002), 339–346. 3

[KB89] KALRA D., BARR A. H.: Guaranteed ray intersections with implicit surfaces. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1989), SIGGRAPH '89, Association for Computing Machinery, p. 297–306. 3

[KBSS01] KOBBELT L. P., BOTSCH M., SCHWANECKE U., SEIDEL H.-P.: Feature sensitive surface extraction from volume data. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, Association for Computing Machinery, p. 57–66. 3

[Kle92] KLEIJ R. V.: Efficient display of quadric CSG models. *Computers in Industry 19* (1992), 201–211. 3

[Lot14] LOTT III G. K.: Direct orthogonal distance to quadratic surfaces in 3D. *IEEE Transactions on Pattern Analysis and Machine Intelligence 36*, 9 (2014), 1888–1892. 3, 4

[LTB19] LEGRAND H., THIERY J.-M., BOUBEKEUR T.: Filtered quadrics for high-speed geometry smoothing and clustering. *Computer Graphics Forum 38*, 1 (2019), 663–677. 3

[MCTB11] MAULE M., COMBA J. L., TORCHELSEN R. P., BASTOS R.: A survey of raster-based transparency techniques. *Computers & Graphics 35*, 6 (2011), 1023–1034. 3

[MDG*17] MOSTAJABODAVEH S., DIETRICH A., GIERLINGER T., MICHEL F., STORK A.: CSG ray tracing revisited: Interactive rendering of massive models made of non-planar higher order primitives. In *VISIGRAPP* (2017). 3

[MES03] MARTÍNEZ D., ESTRADA-SARLABOUS J.: On the distance from a point to a quadric surface. *Revista Investigación Operacional 24* (01 2003). 3, 4

[MT13] MEI G., TIPPER J. C.: Simple and robust Boolean operations for triangulated surfaces. *CoRR abs/1308.4434* (2013). arXiv:1308.4434. 3

[NAT90] NAYLOR B., AMANATIDES J., THIBAULT W.: Merging BSP trees yields polyhedral set operations. *SIGGRAPH Comput. Graph. 24*, 4 (Sept. 1990), 115–124. 3

[NWTK21] NEHRING-WIRXEL J., TRETTNER P., KOBBELT L.: Fast exact Booleans for iterated CSG using octree-embedded BSPs. *Computer-Aided Design 135* (2021), 103015. 3

[PCK10] PAVIC D., CAMPEN M., KOBBELT L.: Hybrid Booleans. *Comput. Graph. Forum 29* (03 2010), 75–87. 3

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM Trans. Graph. 22*, 3 (July 2003), 641–650. 3

[Ros11] ROSSIGNAC J.: Ordered Boolean list (OBL): Reducing the footprint for evaluating Boolean expressions. *IEEE Transactions on Visualization and Computer Graphics 17*, 9 (Sept. 2011), 1337–1351. 3

[Rot82] ROTH S. D.: Ray casting for modeling solids. *Comput. Graph. Image Process. 18* (1982), 109–144. 3

[RVD06] ROMEIRO F., VELHO L., DE FIGUEIREDO L. H.: Hardware-assisted rendering of CSG models. In *2006 19th Brazilian Symposium on Computer Graphics and Image Processing* (2006), pp. 139–146. 3

[RVd08] ROMEIRO F., VELHO L., DE FIGUEIREDO L. H.: Scalable GPU rendering of CSG models. *Computers & Graphics 32*, 5 (2008), 526–539. 3

[SB16] SCHMIDT R. M., BROCHU T.: Adaptive mesh Booleans. *CoRR abs/1605.01760* (2016). arXiv:1605.01760. 3

[SJNJ19] SEYB D., JACOBSON A., NOWROUZEZAHRAI D., JAROSZ W.: Non-linear sphere tracing for rendering deformed signed distance fields. *ACM Trans. Graph. 38*, 6 (Nov. 2019). 3

[SLJ98] STEWART N., LEACH G., JOHN S.: An improved Z-buffer CSG rendering algorithm. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (New York, NY, USA, 1998), HWWS '98, Association for Computing Machinery, p. 25–30. 3

[SLL*18] SHENG B., LIU B., LI P., FU H., MA L., WU E.: Accelerated robust Boolean operations based on hybrid representations. *Computer Aided Geometric Design 62* (2018), 133–153. 3

[TGB13] THIERY J.-M., GUY E., BOUBEKEUR T.: Sphere-Meshes: Shape approximation using spherical quadric error metrics. *ACM Trans. Graph. 32*, 6 (Nov. 2013). 3

[TK20] TRETTNER P., KOBBELT L.: Fast and robust QEF minimization using probabilistic quadrics. *Computer Graphics Forum* (2020). 3

[TWMW09] TU C., WANG W., MOURRAIN B., WANG J.: Using signature sequences to classify intersection curves of two quadrics. *Computer Aided Geometric Design 26*, 3 (2009), 317–335. 3

[UBT17] ULYANOV D., BOGOLEPOV D., TURLAPOV V.: Interactive vizualization of constructive solid geometry scenes on graphic processors. *Program. Comput. Softw. 43*, 4 (July 2017), 258–267. 3

[Wan11] WANG C. C. L.: Approximate Boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (2011), 836–849. 3

[WGT04] WICKE M., GROSS M., TESCHNER M.: CSG tree rendering for point-sampled objects. In *Computer Graphics and Applications, Pacific Conference on* (Los Alamitos, CA, USA, oct 2004), IEEE Computer Society, pp. 160–168. 3

[WLC10] WANG C., LEUNG Y.-S., CHEN Y.: Solid modeling of polyhedral objects by layered depth-normal images on the GPU. *Computer-Aided Design 42* (06 2010), 535–544. 3

[YLW06] YAN D., LIU Y., WANG W.: Quadric surface extraction by variational shape approximation. vol. 4077, pp. 73–86. 3

[YWLY12] YAN D.-M., WANG W., LIU Y., YANG Z.: Variational mesh segmentation via quadric surface fitting. *Computer-Aided Design 44*, 11 (2012), 1072–1082. 3

[ZCL18] ZANNI C., CLAUX F., LEFEBVRE S.: HCSG: Hashing for real-time CSG modeling. *Proc. ACM Comput. Graph. Interact. Tech. 1*, 1 (July 2018). 3

[ZGZJ16] ZHOU Q., GRINSPUN E., ZORIN D., JACOBSON A.: Mesh arrangements for solid geometry. *ACM Trans. Graph. 35*, 4 (July 2016). 3

© 2021 The Author(s)
Computer Graphics Forum © 2021 The Eurographics Association and John Wiley & Sons Ltd.