Acta Cybernetica **25** (2021) 171–185.

Quadric Tracing: A Geometric Method for Accelerated Sphere Tracing of Implicit Surfaces*

Csaba Bálint[ab] and Mátyás Kiglics[ac]

Abstract

Sphere tracing is a common raytracing technique used for rendering implicit surfaces defined by a signed distance function (SDF). However, these distance functions are often expensive to compute, prohibiting several real-time applications despite recent efforts to accelerate them. This paper presents quadric tracing, a method to precompute an augmented distance field that significantly accelerates rendering. This novel method supports two configurations: (i) accelerating raytracing without losing precision, so the original SDF is still needed; (ii) entirely replacing the SDF and tracing an interpolated surface. Quadric tracing can offer 20% to 100% speedup in rendering static scenes and thereby amortizes the slowdown caused by the complexity of the geometry.

Keywords: computer graphics, sphere tracing, signed distance function

1 Introduction

Most implicit surface representations require conversion to a triangle list for efficient, real-time visualization. Distance-based representations are such an exception where sphere tracing [8] enables raytracing for the whole scene within milliseconds, using the GPU. The representation supports a range of set operations, such as union, intersection, subtraction, and real-time offsets [9]. This allows any CSG tree to be constructed from a large set of primitives, set operations, and transformations [6]; however, the function evaluation time and convergence slow drastically with scene complexity. This paper aims to amortize the scene complexity dependence of raytracing an implicit surface defined by a signed distance function and thereby greatly accelerate render times.

*EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Csaba Bálint was also supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology.

[a]Eötvös Loránd University, Budapest, Hungary

[b]E-mail: csabix@inf.elte.hu, ORCID: 0000-0002-5609-6449

[c]E-mail: kiglics@caesar.elte.hu, ORCID: 0000-0003-4957-7495

DOI: 10.14232/actacyb.290007

For this, we generate a cache structure that allows the scene to be traversed with rays more quickly, yet it can slow down near the surface for the utmost accuracy. This structure and the accelerating algorithms must possess the following three properties to improve on existing methods:

- The structure must be concise since three-dimensional data can quickly fill up the available GPU memory.
- Query operations have to be efficient for significant render time reduction during raytracing.
- The preprocessing step should be fast enough to accelerate raytracing, even when called in every frame for dynamic scenes.

Our solution relies on special quadrics, that is, second-degree algebraic surfaces. We define these quadrics pivoted around a given point and an arbitrary direction in space. Each of the enclosed quadratic regions we generate will either contain the entirety of the scene geometry or none of it, so we call these bounding and unbounding quadrics, respectively. Our preprocessing step consists of generating quadric parameters for a 3D grid of values, while the rendering step accelerates raytracing by intersecting the rays with these quadrics first.

This paper focuses on presenting the quadric tracing algorithm and applying it to accelerate sphere tracing utilizing a uniform grid of quadric parameters.

Previous work Since the first appearance of the sphere tracing algorithm [8], several variants of accelerations and enhancements have been released. For clarity, we refer to the sphere tracing method published in [10] as the relaxed sphere tracing algorithm, and we call enhanced sphere tracing the algorithm presented in [3]. The relaxed sphere tracing [10] takes larger steps determined by a scalar, and if the radius is too large, it reverts to the basic sphere tracing size. The more recently published enhanced sphere tracing [3] uses the radii calculated in the two previous steps to efficiently approximate planar surfaces. These methods mainly attempt to increase the size of the steps taken on the ray during rendering; hence, the number of SDF evaluations are minimized. The algorithm applications include light visibility computation such as soft shadows [5], ambient occlusion [18], and global illumination. Various utilizations are also known, like room impulse response estimation [11], or deep learning based implicit signed distance functions [13]. Most of the algorithms above can be improved by quadric tracing.

2 Preliminaries

An $f : \mathbb{R}^3 \to \mathbb{R}$ function is a signed distance function (SDF) if it is continuous and

$$|f(\boldsymbol{p})| = d(\boldsymbol{p}, \{f = 0\}) \quad (\boldsymbol{p} \in \mathbb{R}^3) \,,$$

as stated in [4, 15]. Where $d(\boldsymbol{p}, \{f = 0\}) = \inf\{\|\boldsymbol{p} - \boldsymbol{x}\|_2 : \boldsymbol{x} \in \{f = 0\}\}$ is the Euclidean distance from the $\{f = 0\} = \{\boldsymbol{x} \in \mathbb{R}^3 : f(\boldsymbol{x}) = 0\}$ surface. The $\{f <$

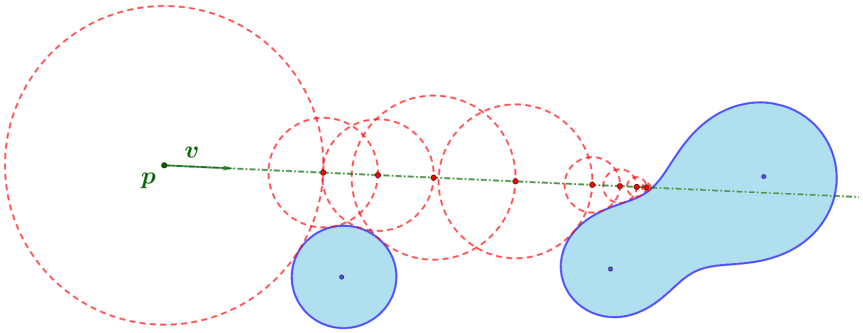


Figure 1: The sphere tracing algorithm takes distance sized steps along the ray. These distance sized steps visualized here in 2D by the red unbounding circles surrounding each signed distance function evaluation point. This image is reused from [4] with the permission of the authors.

$0\} = \{\boldsymbol{p} \in \mathbb{R}^3 : f(\boldsymbol{p}) < 0\}$ region is the inside, while the $\{f > 0\}$ region is considered the outside of the represented geometry. Set operations can be performed on objects defined by SDFs by taking the point-wise minimum or maximum of the operand functions for union or intersections, respectively [8].

Various sphere tracing algorithms exist for surface visualization. These raytracing techniques often start a ray through each pixel of the virtual camera and march along the ray, taking distance sized steps [8]. This is because, for any $\boldsymbol{p} \in \mathbb{R}^3$ point, there are no surface points within $f(\boldsymbol{p})$ distance: this is called the unbounding sphere. Hence, sphere tracing is often visualized as a series of unbounding spheres or circles along a ray as in Figure 1. When the point-to-surface distance becomes negligible, the surface is reached.

However, all sphere tracing algorithms slow down near the surface regardless of the direction taken [10, 3, 5]. The only exception is when the derivative of f is known, and the geometry is convex, in which case huge steps can be taken [8]. This is because, instead of an unbounding sphere, we can draw a separating plane with normal $\nabla f(\boldsymbol{p})$ and surface point $\boldsymbol{p} - f(\boldsymbol{p}) \cdot \nabla f(\boldsymbol{p})$. Note that if $f : \mathbb{R}^3 \to \mathbb{R}$ is an exact signed distance function, then if the derivative exists at $\boldsymbol{p} \in \mathbb{R}^3$, it is of unit length: $\|\nabla f(\boldsymbol{p})\|_2 = 1$. This can often produce long steps, even when the evaluation point is close to the surface. This method can be extended to the union of concave objects, but the average step size will rapidly decrease, and function evaluation time will increase.

Nonetheless, the surface is often concave, or the derivative might be undefined or unknown. For this reason, we generalize the unbounding sphere and separating plane approach to unbounding quadrics [12] for any SDF. Our method consists of two steps. During precomputation, we store distance values in a regular grid. For each cell, the normal vector $\nabla f(\boldsymbol{p})$ is stored along with a $k \in [-1, 1]$ parameter describing the shape of the quadric. Therefore, the quadric is parameterized with a

3D direction vector and a scalar value. During the rendering step, *quadric tracing* intersects the ray with the precomputed quadric to accelerate tracing convergence.

3 Conic sections of $k \in [-1, 1]$

We chose to use the revolution of conic sections for the quadratic proxy surfaces such that a single scalar value $k \in [-1, 1]$ we can encode a large variety of surfaces around the axis defined by $\nabla f(\boldsymbol{p})$. This section details how the 2D conics are constructed to generate both bounding and unbounding regions. Let the y-axis denote the axis of revolution such that the symmetric curve going through the origin has the following form:

$$A(k) \cdot x^2 + B(k) \cdot y^2 + C(k) \cdot y = 0 \qquad (A, B, C : [-1, 1] \to \mathbb{R}). \tag{1}$$

The coefficients are functions of the $k \in [-1, 1]$ parameter, and these control the shape, eccentricity, and curvature of the conic [17]. We found the following three functions to change the conic section in the desired, continuous way from a circle centered at $(0, -0.5)$ to another around $(0, 0.5)$ as k increased from -1 to 1.

$$A := A(k) := k^2 \ , \ B := B(k) := 2\left(|k| - \frac{1}{2}\right) \ , \ C := C(k) := -k \tag{2}$$

This choice allows the conic sections seen on Figure 2 to turn inside out at $k = 0$, and describe both bounded $|k| > \frac{1}{2}$ and unbounded $|k| \leq \frac{1}{2}$ regions visualized. For brevity, we omit the function notation.

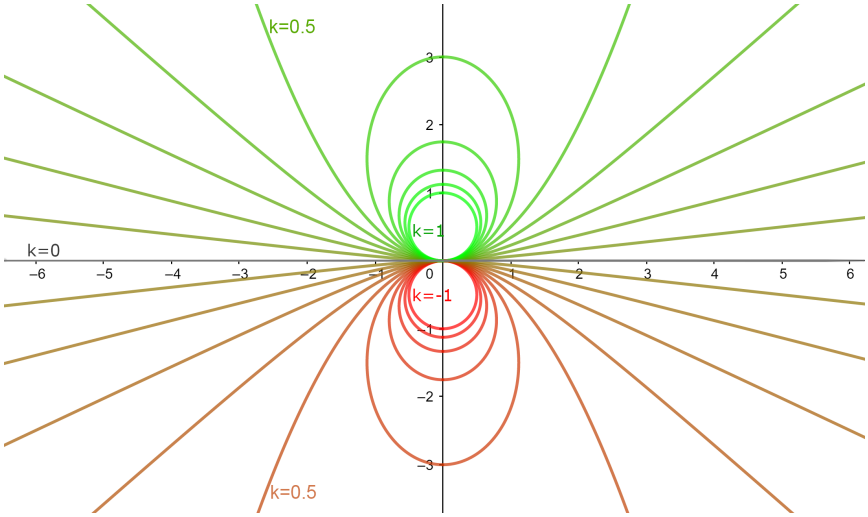


Figure 2: Conic sections with varying $k \in [-1, 1]$. For $k = \pm 1$, the conic section is a circle; for $k = \pm 0.5$, a parabola; for $k \in (0.5, 1)$ or $k \in (-1, -0.5)$, an ellipse; and for $k \in (0, 0.5)$ or $k \in (-0.5, 0)$, it is a branch of a byperbola.

We obtain the polar parametrization of these conics by intersecting them with $t \in [-\pi, \pi)$ angled rays from the origin. The distance from the origin to the intersection point will be a function of this angle, $r(t)$; so by substituting the polar coordinates into the implicit form in (1), we can solve for this $r(t)$:

$$A \cdot (r(t) \cdot \cos t)^2 + B \cdot (r(t) \cdot \sin t)^2 + C \cdot r(t) \sin t = 0 \; .$$

Assuming that $r(t) \neq 0$ yields the polar parametrization $\boldsymbol{s} : [-\pi, \pi) \to \mathbb{R}^2$ of the conic section:

$$r(t) = \frac{-C \cdot \sin t}{A \cdot \cos^2 t + B \cdot \sin^2 t} \implies \boldsymbol{s}(t) := \begin{bmatrix} \cos t \cdot r(t) \\ \sin t \cdot r(t) \end{bmatrix} \qquad \big(t \in [-\pi, \pi)\big) \tag{3}$$

For values of $k \in \left(-\frac{1}{2}, \frac{1}{2}\right)$, the $\boldsymbol{s}(t)$ describes a hyperbola with an unwanted branch. Let $L := L(k) \in [-\pi, \pi)$ denote the value where $r(t)$ has a singularity. Thus, we can restrict $\boldsymbol{s}(t)$ to the $[-L, L)$ interval where

$$L := L(k) := \begin{cases} \frac{\pi}{2}, & \text{if } A(k) \cdot B(k) \geq 0, \\ \arctan \sqrt{\frac{-A}{B}}, & \text{otherwise} \end{cases} \qquad (k \in [-1, 1]) \; .$$

Note that for all of the equations above, the heuristic $A(k), B(k)$, and $C(k)$ functions may readily be redefined if needed. We have experimented with several sets of functions. Simplicity, numerical stability, and continuity in terms of k were the deciding factors. Even though the parametric form has a singularity at $k = 0$, the algorithms in this paper are based on the implicit equation, extending to the $k = 0$ line or plane.

4 Unbounding quadrics

We parameterize quadrics of revolution by rotating $\boldsymbol{s}(t) = [\boldsymbol{s}_1(t), \boldsymbol{s}_2(t)]^T$ from (3) around the vertical axis:

$$P(u, v) := \begin{bmatrix} \cos(v) \cdot \boldsymbol{s}_1(u) \\ \sin(v) \cdot \boldsymbol{s}_1(u) \\ \boldsymbol{s}_2(u) \end{bmatrix} = r(u) \cdot \begin{bmatrix} \cos v \cdot \cos u \\ \sin v \cdot \cos u \\ \sin u \end{bmatrix} \quad (u \in [0, L(k)), v \in [0, 2\pi)).$$

These quadrics can be seen on Figure 3. Similarly, the implicit equation becomes

$$A \cdot (x^2 + y^2) + B \cdot z^2 + C \cdot z = 0.$$

Applying the above, we can calculate the intersection of a ray and the quadric surface. The ray is given by a point $\boldsymbol{p}_0 = (x_0, y_0, z_0) \in \mathbb{R}^3$ and a vector $\boldsymbol{v} = (\boldsymbol{v}_x, \boldsymbol{v}_y, \boldsymbol{v}_z) \in \mathbb{R}^3, \|\boldsymbol{v}\| = 1$, so we can substitute any $\boldsymbol{p}_0 + t \cdot \boldsymbol{v}$ $(t > 0)$ point on the ray and solve the resulting quadratic equation with the coefficients in the $at^2 + bt + c = 0$ equation:

$$\begin{aligned} a =& A \cdot \boldsymbol{v}_x^2 + A \cdot \boldsymbol{v}_y^2 + B \cdot \boldsymbol{v}_z^2 \\ b =& 2A \cdot x_0 \boldsymbol{v}_x + 2A \cdot y_0 \boldsymbol{v}_y + 2B \cdot z_0 \boldsymbol{v}_z + C \boldsymbol{v}_z \\ c =& A \cdot (x_0^2 + y_0^2) + B \cdot z_0^2 + C \cdot z_0 \end{aligned}$$

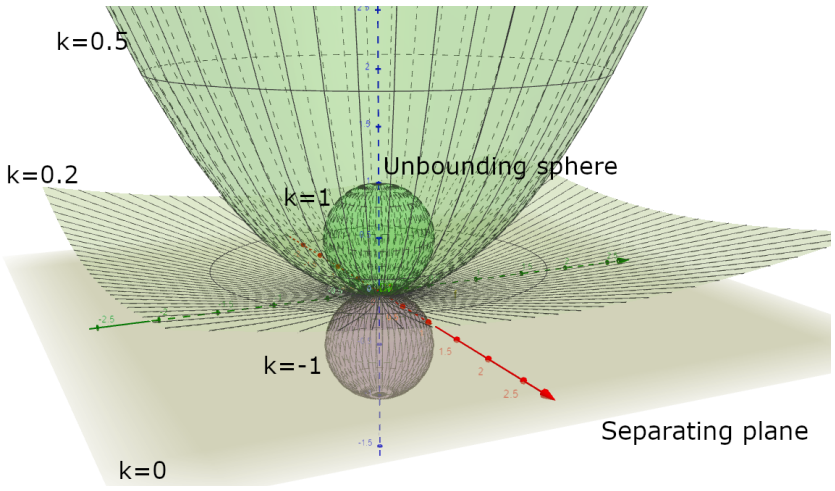


Figure 3: Surfaces of the unbounding quadrics for $k \in [-1, 1]$ separate the space into two regions. The region that contains the $(0, 0, 1)$ point must contain the entirety of the surface once the quadric is transposed and rotated into the scene.[16]

If there are real roots, let them be $t_1 \leq t_2$ and let $I := [t_1, t_2]$. If there are no real roots, let $I := [t_1, t_2] := [-\infty, +\infty]$ [14]. However, if $0 \neq |k| < \frac{1}{2}$, then the solution corresponding to the unwanted branch of the hyperbola needs to be omitted:

$$I := [t_1, t_2] := \begin{cases} [-\infty, t_2] & \text{if } (z_0 + t_1 \boldsymbol{v}_z) \cdot k < 0 \\ [t_1, +\infty] & \text{if } (z_0 + t_2 \boldsymbol{v}_z) \cdot k < 0 \end{cases}$$

If none of the conditions apply, I left as defined before. Then, the first intersection between the ray and the quadric may be computed by evaluating the following four conditional assignments in order:

$$\begin{array}{lll} t := t_2 & \text{if} & t_1 = -\infty \\ t := t_1 & \text{if} & 0 < t_2 < +\infty \\ t := \infty & \text{if} & t_1 < 0 \\ t := 0 & \text{if} & \text{otherwise} \end{array} \tag{4}$$

The resulting $t \geq 0$ is the intersection location along the $\boldsymbol{p}_0 + t \cdot \boldsymbol{v}$ ray.

5 Preprocessing

The accelerator data structure consists of a single grid that stores the distance values and the quadrics of revolutions in four 32 bit floating-point values. We multiply the normalized axis direction $\boldsymbol{n} \in \mathbb{R}^3$ ($\|\boldsymbol{n}\|_2 = 1$) with $2 + k$ to store the quadric using 96 bits while the final 32 bits are reserved for the signed distance

function values $d \in \mathbb{R}$. The quadrics are defined relative to each grid point $\boldsymbol{p}_c \in \mathbb{R}^3$ which becomes the origin in Figure 3. The quadric is scaled and rotated such that the axis of revolution points towards is $\boldsymbol{n}$.

During the preprocessing steps, the regular grid is populated with values. The axis we store is the $-\frac{\nabla f(\boldsymbol{p})}{\|\nabla f(\boldsymbol{p})\|}$ gradient vector because it mostly points towards the surface and proves to be a good heuristics. The following four steps detail the computation of the $k \in [-1, 1]$ values. These are executed for every cell in parallel using the GPU:

1. Starting from the origin of the quadric $\boldsymbol{p}_c$, we shoot rays in uniformly sampled directions $\boldsymbol{v}_i$ $(i = 1, \ldots, N)$.
2. Each $\boldsymbol{p}_c + t\boldsymbol{v}_i$ ray is then sphere traced until the surface or the bounding box is reached.
3. For each ray-surface intersection $(x_i, y_i, z_i) := \boldsymbol{p}_c + t_i\boldsymbol{v}_i$, we compute the $k_i \in [-1, 1]$ value that defines the unique quadric of revolution that touches the surface at that intersection point.
4. To obtain the unbounding quadric within the cell, we need the narrowest candidate unbounding quadric region from all rays; therefore, $k := \min\{k_i \mid i \in \{1, \ldots, N\}\}$.

The third step warrants more explanation. Note that we only need to operate in the plane defined by $\boldsymbol{v}_i$ and the $\boldsymbol{n}$ normal vector because the quadric is symmetric around $\boldsymbol{n}$. Let $\boldsymbol{q} = (\boldsymbol{q}_x, \boldsymbol{q}_y) \in \mathbb{R}^2$ be the rotated projection into this plane of the (x_i, y_i, z_i) intersection point where $\boldsymbol{n}$ corresponds to the y-axis in 2D. We can solve the 2D implicit equation in (1) of the conic for the k parameter after substituting (2):

$$k^2\boldsymbol{q}_x^2 + 2\left(|k| - \frac{1}{2}\right)\boldsymbol{q}_y^2 - k\boldsymbol{q}_y = 0$$

Depending on the sign of k, the entirety of the conic section lies in the upper or lower half planes, that is: $\operatorname{sgn}\boldsymbol{q}_y = \operatorname{sgn} k$. Substituting $|k| = k \operatorname{sgn} k$, and rearranging yields the quadratic equation

$$\boldsymbol{q}_x^2 \cdot k^2 + (2 \operatorname{sgn}\boldsymbol{q}_y \cdot \boldsymbol{q}_y^2 - \boldsymbol{q}_y) \cdot k - \boldsymbol{q}_y^2 = 0\ .$$

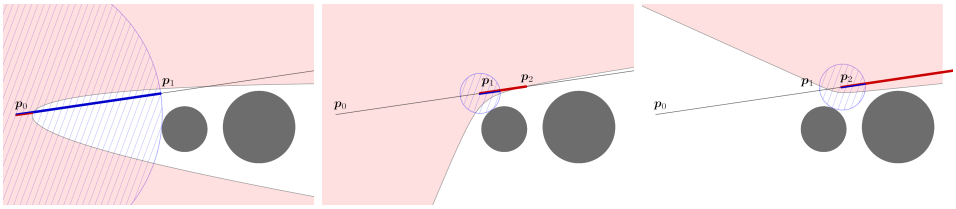
The smaller solution for k is the desired parameter k_i of the quadric in the third step.

6 Quadric tracing

The benefit of quadric tracing is that it takes much larger steps along the ray, as illustrated in Figure 4. The quadrics are stored in grid cells, so for each evaluation point along the ray, the closest quadric is queried and intersected as seen in Section 4. This often yields a step size that skips several grid cells and still does not skip ray surface intersections.

However, sometimes the quadric does not contain the whole cell that it is assigned to, so the acceleration is zero or negligible for some positions. In this case, we take advantage of the stored distance values to advance the iteration by the larger of the two proposed step size. When the computed distance value dips below a certain threshold, our quadric tracing iteration stops. The remaining iterations are then used for the enhanced sphere tracing method [3] that converges quickly close to the surface.

The C++ and OpenGL implementation was developed using the Dragonfly framework [2, 1]. The preprocessing step, the sphere tracing methods, and quadric tracing are implemented on the GPU using compute shaders and appropriate memory barriers. The CSG trees are stored as structures on the CPU, but the shader code for the SDF can be readily generated on-the-fly. Thus, upon the change of parameters or desired algorithms, the implementation generated the SDF and recompiled its shaders to preprocess and render the new surface.



(a) A sphere tracing step was taken because it was larger

(b) Quadric tracing step is now greater then the distance

(c) Quadric step to infinity terminates the raytracing

Figure 4: Consecutive quadric tracing of two circles in 2D. The preprocessing step created the pink unbounding regions to accelerate raytracing whenever the quadric step is larger (b, c). Sometimes, the sphere tracing step is larger because the quadrics are confined to a grid (a). However, the ray does not intersect the quadric in the last case, meaning the algorithm halts in the third iteration.

7 Results

Our implementation featured and compared classic sphere tracing, relaxed sphere tracing [10], enhanced sphere tracing [3], quadric tracing. For the method we call relaxed sphere tracing from [10], the recommended 1.6 step size increase was used. The enhanced sphere tracing had to be slowed down to accommodate smooth concave surfaces, so the recommended 0.95 step size reduction was applied as detailed in [3]. During our experiments, we implemented nine test scenes presented in Figure 5. These test models were also used in [7] for their measurements.

Measuring errors for sphere tracing algorithms can be problematic because there are hit rays that intersect the surface, and there are miss rays that do not. In both cases, most algorithms cannot ever reach the desired value, only converge towards it. Thus, we have a distance-to-surface error tolerance threshold that changes

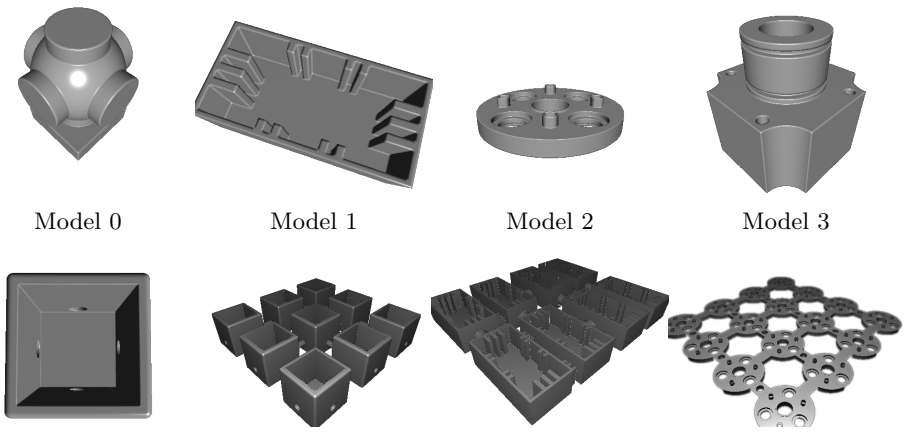


Figure 5: Performance measurements were based on these test scenes from [7] for which we have our own implementation.

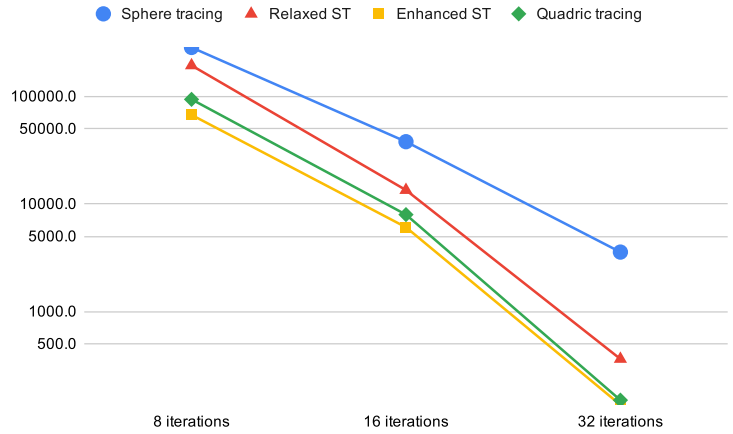
along the ray, effectively, a cone trace. This means that the absolute error can be arbitrarily large and still be acceptable. Also, if the algorithm takes larger steps, it tends to overstep the error threshold by a larger amount making the error smaller compared to another method with a smaller step size.

For our measurements, we generated a $16 \times 16 \times 16$ quadric field by shooting 70×70 rays for each cell. The preprocessing took around three to four times as long as producing a single frame.

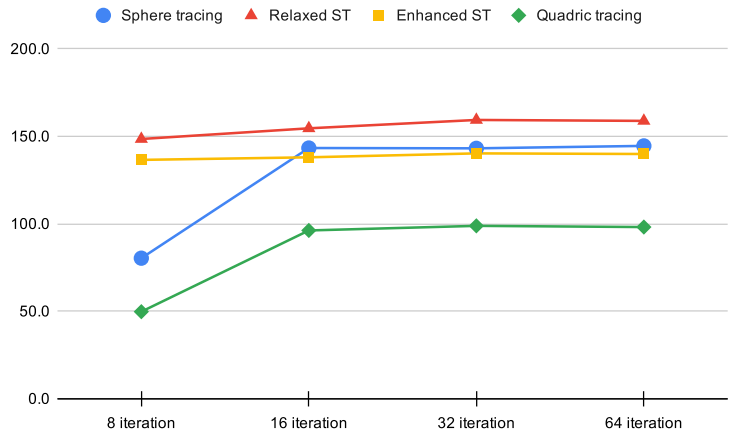
Data for ground truth were produced by sphere tracing the scene for a thousand iterations. This causes the error of relaxed, enhanced, and quadric tracing methods to plateau for high iterations, while the sphere tracing will be exact compared to itself, clearly seen above iteration 64 on Figure 6a.

We made several error metrics, such as the absolute error of divergent and convergent rays, the ratio of rays that did not converge, and the number of SDF evaluations. For the first one, we compared rays that missed or converged to the surface in the ground truth data but were still converging in the test case. Error values could not be aggregated across test scenes easily; thus, we provide in-depth data for the run with 32 iterations on Table 1. Table 1 presents average frame render times and the measured sum of absolute raytracing errors for each model. Quadric tracing performed better for more complex scenes, and such a case is presented in Figure 6. We also plotted the number of rays that did not converge before a given iteration number on Figure 8a.

Our measurements also confirmed the results of [3]: enhanced sphere tracing decreases the error at a higher rate than the relaxed or the original method. Enhanced sphere tracing is most efficient at proximity to the surface, while our quadric tracing accelerates the raytracing through the vast distances between the objects,



(a) Error on a logarithmic scale



(b) Render time in milliseconds

Figure 6: Measured error and render time values as a function of the iteration for Model 7 test scene. Quadric tracing has a slightly higher error for a given iteration count but yields much higher performance across.

as exemplified by Figure 6. The average render time improvement is similar, as demonstrated by Figure 7. Quadric tracing becomes much faster than other methods as the function evaluation time increases because the cost of memory read operations are unaffected by scene complexity. Signed distance function evaluations are decreased by 20% on average when rendering with quadric tracing, as seen in Figure 8a.

Table 1: Relative error and render time comparison of the novel quadric tracing compared to the state of the art methods, each taking 32 iterations. The relative values are divided by the error of the basic sphere tracing, taking the same number of iterations. The enhanced method is slightly more accurate but takes longer to execute for complex scenes.

	Relative error w.r.t. sphere tracing			**Render time in ms**			
	relaxed	enhanced	quadric	basic	relaxed	enhanced	quadric
model 0	0.0945	**0.0362**	0.0488	8	8	**8**	8
model 1	0.2920	**0.1227**	**0.1227**	15	14	**10**	12
model 2	0.1259	0.0347	**0.0327**	18	18	**14**	16
model 3	0.0463	**0.0034**	**0.0034**	13	13	**10**	11
model 4	**0.0003**	**0.0003**	**0.0003**	8	**8**	8	9
model 5	0.1883	**0.0879**	0.0982	71	70	59	**40**
model 6	0.0842	**0.0142**	0.0152	263	276	230	**162**
model 7	0.0574	**0.0101**	0.0137	616	695	626	**363**
model 8	0.0054	**0.0003**	**0.0003**	288	330	293	**262**

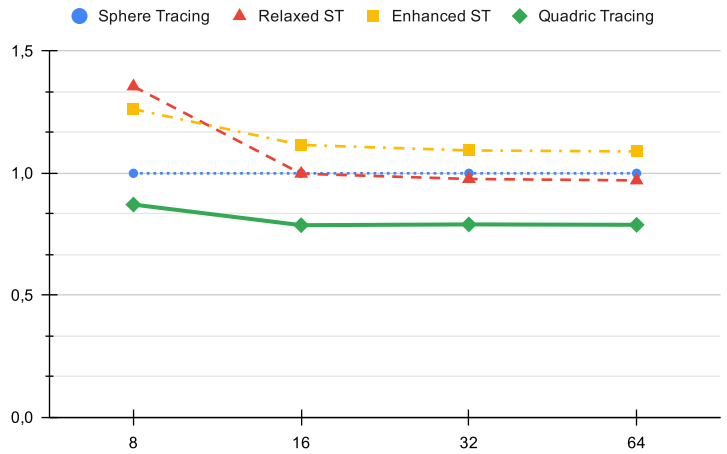
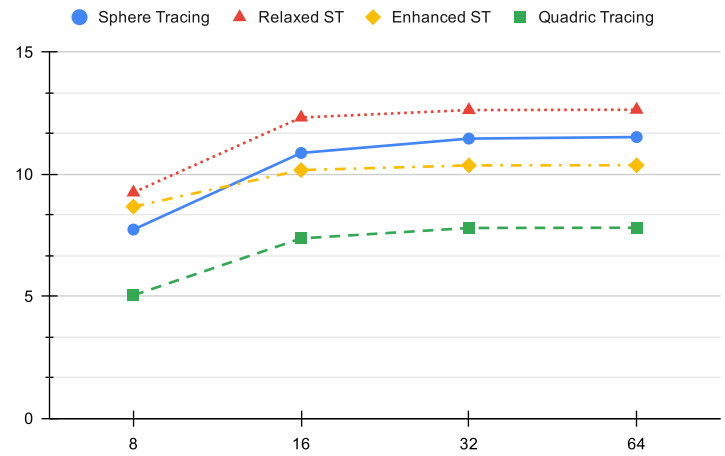
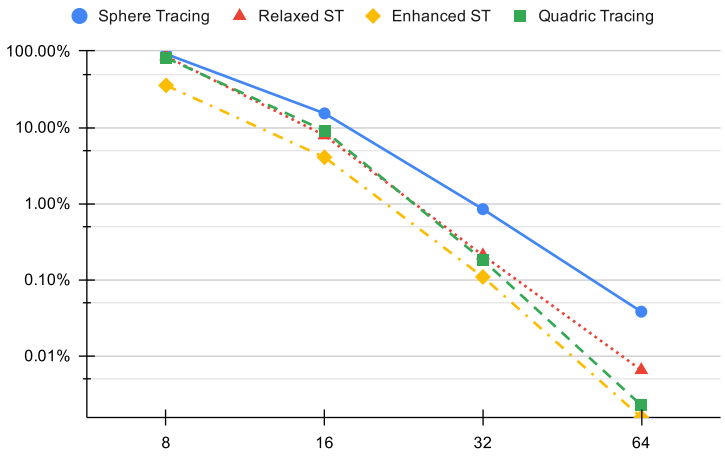


Figure 7: Average relative render time with respect to sphere tracing. Each runtime is divided by that of sphere tracing and then the results are averaged. The time overhead of the enhanced sphere tracing at the end of quadric tracing is included.



(a) Average SDF evaluations per ray



(b) Percentage of rays that did not converge

Figure 8: Number of function evaluations per pixel, and the number of rays that neither missed or hit the surface yet. About 20% of the expensive signed distance function evaluations can be avoided with quadric tracing.

8 Conclusion

This paper presented a novel algorithm to accelerate the raytracing of implicit surfaces by fitting special quadratic volumes from grid points to the surface and raytracing the precomputed proxy geometry instead. Thus, instead of a potentially expensive function evaluation, quadric tracing mostly reads from memory when rendering the scene. Even though our method can visualize the surface from the precomputed data, the acceleration structure is coarse and would seriously limit the resolution and remove surface features. Hence, we opted to continue sphere tracing on the exact signed distance values after our quadric tracing iteration. Therefore, the surface remains unchanged while a significant amount of function evaluations were avoided.

Implementation of the quadric tracing idea necessitated several difficulties to be solved. First, the unbounding volumes had to be compactly stored, efficiently computed, queried, and intersected with, for which revolutions of conic sections were designed with specific coefficient functions. Second, both the parametric and implicit representations were needed in 2D and 3D to write efficient implementations for these on the GPU. This paper focused more on the rendering part rather than the quadric generation because raytracing efficiency determines the competitiveness of quadric tracing. Finally, we implemented all of the presented methods in C++ and OpenGL, utilizing the massive parallelization of the GPU for preprocessing, sphere tracing, and rendering.

Our results show that quadric tracing can efficiently mitigate the slowdown caused by large CSG trees. In such cases, the method is up to two times faster then enhanced sphere tracing [3] whilst being slightly less precise, as shown in Figure 6. On average, quadric tracing was 40% faster than the current state of the art method.

However, quadric tracing requires the accelerator data structure to be computed and populated, taking from a few milliseconds to seconds to complete. Therefore, every time the scene changes, every quadric in every cell needs to be updated, which renders quadric tracing impractical for dynamic scenes. Although intuition suggests otherwise, executing the preprocessing step and the quadric tracing in every frame can be faster than the enhanced method, as our early experiments suggested. However, this was not the focus of this paper, so further research is required with a suitable data structure that can be updated efficiently.

References

[1] Bálint, Csaba and Bán, Róbert. Dragonfly: A C++17 OpenGL framework. 7th Winter School of PhD Students in Informatics and Mathematics, 2020. `http://www.doszmito.hu/wsps7.pdf`.

[2] Bálint, Csaba and Bán, Róbert. Dragonfly: A high level low overhead OpenGL framework. The 11th International Conference on Applied Informatics, 2020.

[3] Bálint, Csaba and Valasek, Gábor. Accelerating Sphere Tracing. In Diamanti, Olga and Vaxman, Amir, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018. DOI: 10.2312/egs.20181037.

[4] Bálint, Csaba, Valasek, Gábor, and Gergó, Lajos. Operations on signed distance functions. *Acta Cybernetica*, 24(1):17–28, May 2019. DOI: 10.14232/actacyb.24.1.2019.3.

[5] Bán, Róbert, Bálint, Csaba, and Valasek, Gábor. Area Lights in Signed Distance Function Scenes. In Cignoni, Paolo and Miguel, Eder, editors, *Eurographics 2019 - Short Papers*. The Eurographics Association, 2019. DOI: 10.2312/egs.20191021.

[6] Foley, James David. *Constructive Solid Geometry*. In *Computer Graphics: Principles and Practice*, pages 533–558. Addison-Wesley Professional, 1990.

[7] Friedrich, Markus, Roch, Christoph, Feld, Sebastian, Hahn, Carsten, and Fayolle, Pierre-Alain. A flexible pipeline for the optimization of CSG trees. *Computer Science Research Notes*, 3001, 2020. DOI: 10.24132/csrn.2020.3001.10.

[8] Hart, John. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12, 1995. DOI: 10.1007/s003710050084.

[9] Íñigo Quílez. Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. In NVScene, 2008. https://uploads.gamedev.net/monthly_2017_07/rwwtt_pdf.dde5c198ccab31d95a41093666ffaad1.

[10] Keinert, Benjamin, Schäfer, Henry, Korndörfer, Johann, Ganse, Urs, and Stamminger, Marc. Enhanced Sphere Tracing. In Giachetti, Andrea, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014. DOI: 10.2312/stag.20141233.

[11] Lechner, Patrik. Room impulse response estimation using signed distance functions. In *DAFx-2020 - Vienna*. 23rd International Conference on Digital Audio Effects, September 2020.

[12] Levy, Silvio. Geometry formulas and facts. In *30th Edition of CRC Standard Mathematical Tables and Formulas*. CRC Press, 1995.

[13] Liu, Shaohui, Zhang, Yinda, Peng, Songyou, Shi, Boxin, Pollefeys, Marc, and Cui, Zhaopeng. DIST: rendering deep implicit signed distance function with differentiable sphere tracing. *CoRR*, abs/1911.13225, 2019. http://arxiv.org/abs/1911.13225.

[14] Oden, J. Tinsley and Demkowicz, Leszek. Linear algebra. In *Applied Functional Analysis (3 ed.)*. Chapman and Hall/CRC, 2018.

[15] Osher, Stanley and Fedkiw, Ronald. Signed distance functions. In *Level Set Methods and Dynamic Implicit Surfaces*, pages 17–22. Springer, 2003.

[16] Rodrigues, Olinde. Des lois géometriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacement considérées indépendant des causes qui peuvent les produire. *Journal de Mathématiques Pures et Appliquées*, 5:380–440, 1840.

[17] Thomas, George B. and Finney, Ross L. *Curves in the Plane.* In *Calculus and Analytic Geometry (5th ed.).* Addison-Wesley, 1979.

[18] Wright, Daniel. Dynamic occlusion with signed distance fields. In *Advances in Real-Time Rendering in Games.* Epic Games (Unreal Engine), SIGGRAPH course, 2015.