Accelerating Sphere Tracing

Csaba Bálint[†] and Gábor Valasek[‡]

Eötvös Loránd University (Hungary)

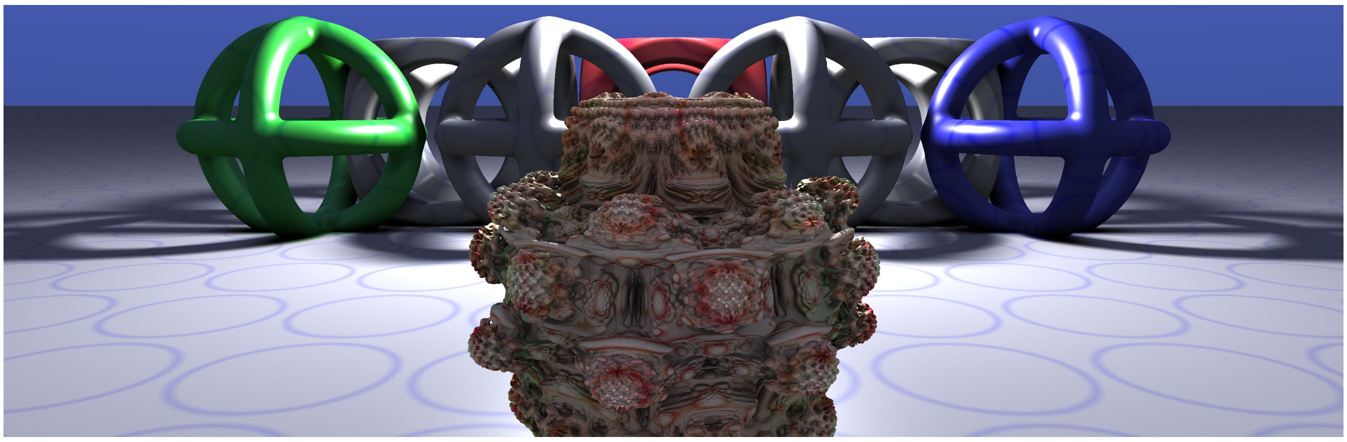

Figure 1: *Real-time sphere tracing with soft shadows and ambient occlusion. Runtime = 47ms, resolution: 3440x1440, GPU: Nvidia 1080Ti.*

Abstract

This paper presents two performance improvements on sphere tracing. First, a sphere tracing variant designed to take optimal step sizes near planar surfaces is proposed. We demonstrate how relaxation is used to make this method applicable to sphere tracing arbitrary geometries and compare its performance to classical (by Hart) and relaxed (Keinert et al.) sphere tracing in rendering various scenes. The method is also general in the sense that it can be applied in any scenario that requires the computation of ray-surface intersections. Our second contribution is a multi-resolution rendering strategy that can be used with any sphere tracing variant. By starting from a lower resolution and gradually increasing it, render times can be reduced.

CCS Concepts

•***Computing methodologies*** $\rightarrow$ *Rendering; Ray tracing;*

1 Introduction

Sphere tracing, introduced by Hart in [Har94] and first applied in [JCH89], is an iterative algorithm to compute the intersection of a ray with a surface defined by a signed distance function (SDF) or any of its lower estimates. Its ease of use and efficiency made it an ideal choice for real-time rendering of SDF representations. Keinert et al. [KSK*14] provided several improvements on the basic sphere tracing algorithm in terms of both performance and quality.

We propose a modification to the sphere tracing algorithm in Section 3 based on the observation that converging to approximately planar entities is one of the most expensive situations for existing sphere tracing techniques. To mitigate this, our *enhanced sphere tracing algorithm* makes an optimal step along the ray assuming the surface is locally flat. This allows faster convergence in such configurations. We also show how a step-size relaxation, similar to that of [KSK*14], makes this algorithm applicable to sphere tracing not only planar but arbitrary geometries, including fractals.

Our second contribution, presented in Section 4, is a multi-resolution based rendering strategy. Singed distance functions allow us to split a single ray into multiple others without penetrating the traced surface. Using this, we empirically show that gradually changing resolution can improve the performance of all algorithms.

† csabix.balint@gmail.com
‡ valasek@inf.elte.hu

DOI: 10.2312/egs.20181037

We discuss the results of our performance tests in Section 5, first comparing our proposed enhanced sphere tracing algorithm to that of Hart and Keinert et al., then we discuss our results on how resolution change can increase rendering speeds.

2 Sphere tracing

Sphere tracing starts from the origin of a ray and marches iteratively until it approaches the surface within a prescribed threshold. The step size in each iteration is the distance of the current point along the ray from the surface. This distance defines a so-called *unbounding sphere* about the current position within which it is guaranteed that the volume has no points.

Let $d(\boldsymbol{p}, A)$ denote the distance of point $\boldsymbol{p}$ from surface $A \subset \mathbb{R}^3$. The implicit mapping $f : \mathbb{R}^3 \to \mathbb{R}$ is a **distance function** if

$$\forall \boldsymbol{p} \ : \ f(\boldsymbol{p}) = d(\boldsymbol{p}, \{f \equiv 0\}) .$$

Furthermore, if $\{f \equiv 0\}$ defines the boundary of a volume, f is a **signed distance function** if $f(\boldsymbol{p}) = d(\boldsymbol{p}, \{f \equiv 0\})$ for every $\boldsymbol{p}$ outside the volume and $f(\boldsymbol{p}) = -d(\boldsymbol{p}, \{f \equiv 0\})$ if $\boldsymbol{p}$ is inside.

Although closed-form SDF representation of arbitrary geometries is infeasible, Hart noted in [Har94] that it suffices to have a lower bound on the real distance for efficient rendering.

More precisely, we say that the continuous function $f : \mathbb{R}^3 \to \mathbb{R}$ is a **signed distance function estimation (SDFE)**, if and only if there exists a $q : \mathbb{R}^3 \to [1, K)$ bounded function ($K \in \mathbb{R}$), such that $f \cdot q$ is a (signed) distance function.

Therefore, it is clear that **basic sphere tracing** in Algorithm 1 does not jump over an intersection.

```
In  : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, |\boldsymbol{v}| = 1$ ray, $f : \mathbb{R}^3 \to \mathbb{R}$ SDF estimate
Out: $t \in [0, +\infty)$ distance traveled along the ray
$t := 0; \quad i := 0;$
for $i < i_{max}$ and $f(\boldsymbol{p} + t \cdot \boldsymbol{v})$ not too small; $i := i + 1$; do
    $t := t + f(\boldsymbol{p} + t \cdot \boldsymbol{v})$
end
```

Algorithm 1: *Basic sphere tracing adapted from [Har94].*

Keinert et al. [KSK*14] noted that the step size of $f(\boldsymbol{p} + t \cdot \boldsymbol{v})$ is often too conservative and the SDF estimate value can be scaled up most of the times. Nevertheless, if the unbounding spheres of two consecutive steps are disjoint, one has to revert to a basic sphere tracing step size, so Algorithm 2 does not skip intersections.

```
In  : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3$ ray, $|\mathbf{v}|$=1, SDFE $f$, relaxation parameter $\omega$
Out: $t \in [0, +\infty)$ distance traveled along the ray
$i := 0; \quad r_i := 0; \quad r_{i+1} := +\infty \quad t := 0;$
for $i < i_{max}$ and $r_{i+1}$ not too small; $i := i + 1$ do
    $d_i := r_i + \omega \cdot r_i;$
    $r_{i+1} := f(\boldsymbol{p} + (t + d_i) \cdot \boldsymbol{v});$
    if $d_i > r_i + r_{i+1}$ then
        $d_i := r_i;$
        $r_{i+1} := f(\boldsymbol{p} + (t + d_i) \cdot \boldsymbol{v});$
    end
    $t := t + d_i; \quad r_i := r_{i+1};$
end
```

Algorithm 2: *Relaxed sphere tracing adapted from [KSK*14].*

They also proposed several modifications aimed at improving image quality, but for the purposes of our paper, we refer to Algorithm 2 as **relaxed sphere tracing**.

3 Enhanced sphere tracing

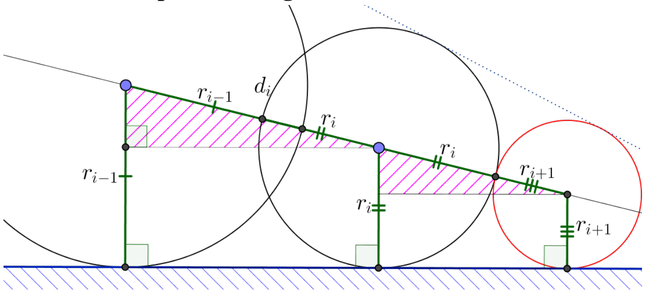

Figure 2: *Geometric construction of the enhanced sphere tracing step to optimally approximate planar surfaces.*

In this section, we propose an **enhanced sphere tracing algorithm** that approximates planar surfaces optimally. This is achieved by selecting a new point along the ray such that the previous and the new unbounding spheres are tangential. To compute the next point of sampling, the radius of this new, tangential unbounding sphere is needed. Instead of using the SDF estimate and root finding to compute it, we infer it from the previous two distance values (i.e., unbounding radii), as depicted in Fig. 2; the radius of the next sphere, r_{i+1} is the unknown. From the similarity of the striped triangles, one gets the following equation and solving it gives r_{i+1}:

$$\begin{aligned} d_i \cdot (r_i - r_{i+1}) &= (r_i + r_{i+1}) \cdot (r_{i-1} - r_i) \\ r_{i+1} &= r_i \cdot \frac{d_i - r_{i-1} + r_i}{d_i + r_{i-1} - r_i} \end{aligned} \tag{1}$$

```
In  : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3$ ray, $|\mathbf{v}|$=1, SDFE $f$, relaxation parameter $\omega$
Out: $t \in [0, +\infty)$ distance traveled along the ray
$r_{i-1} := 0; \quad r_i := 0; \quad r_{i+1} := +\infty;$
$d_i := 0; \quad t := 0;$
for $i := 0$; $i < i_{max}$ and $r_{i+1}$ not too small; $i := i + 1$ do
    $d_i := r_i + \omega \cdot r_i \cdot \frac{d_i - r_{i-1} + r_i}{d_i + r_{i-1} - r_i};$
    $r_{i+1} := f(\boldsymbol{p} + (t + d_i)\boldsymbol{v});$
    if $d_i > r_i + r_{i+1}$ then
        $d_i := r_i;$
        $r_{i+1} := f(\boldsymbol{p} + (t + d_i)\boldsymbol{v});$
    end
    $t := t + d_i;$
    $r_{i-1} := r_i; \quad r_i := r_{i+1};$
end
```

Algorithm 3: *Our enhanced sphere tracing algorithm.*

If the surface is not planar, the actual SDF estimate value may differ from our inference. This poses a problem if the current and the previous unbounding spheres are disjoint since that might be an indication of a missed intersection point. In such cases, Algorithm 3 falls back to a standard sphere trace step.

When the surface is locally convex, the unbounding spheres overlap, so no correction is needed. Resorting back to the standard sphere tracing is only required when the surface is locally concave. However, relaxing this method by scaling down the inferred new

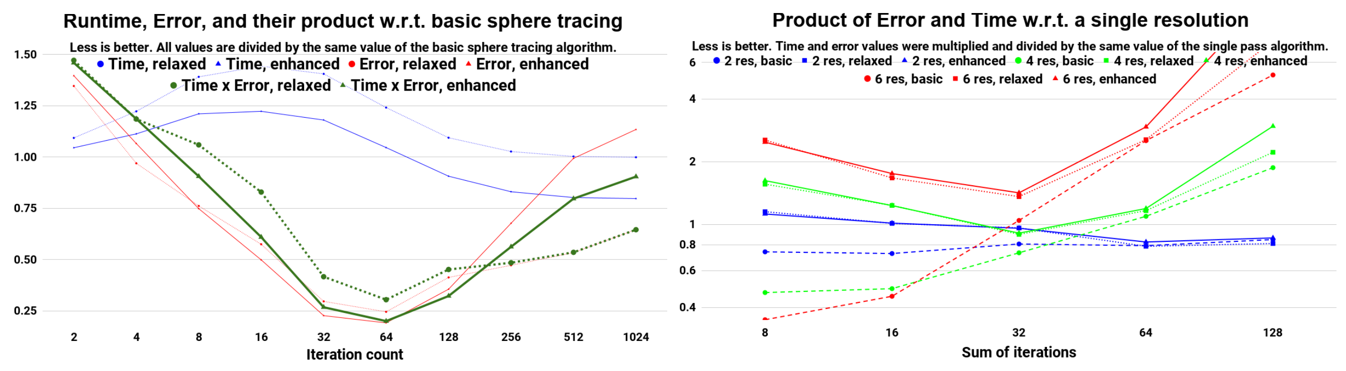

Figure 3: *Runtime and error statistics of the benchmarked algorithms, averaged over the 3 test scenes. On the left, we compare relaxed and enhanced sphere tracing using basic sphere tracing as a baseline for both runtime and error. On the right, we compare multiple resolution rendering strategies with a single resolution baseline. The horizontal axes display how many iterations the algorithms made in total.*

radius decreases the occurrence of such fallbacks for concave geometries. In general, this allows a faster traversal of segments close to the surface where basic sphere tracing would otherwise spend a considerable amount of time. Also, since our inference is a linear extrapolation of the new SDF value based on the previous two observations, step sizes increase rapidly as the ray is getting farther away from a smooth surface. Therefore, when the ray misses an object, enhanced sphere tracing makes it exit the scene faster.

4 Iteratively increasing resolution

The principal disadvantage of SDF representations is that function evaluation times grow rapidly with the increase of scene complexity. There are practical solutions to handle more complex scenes through space partitioning and bounding volumes [Har94,KSK*14]; however, the slowdown is more severe compared to the current triangle list based game engines, and often the above methods are inefficient, for example, when rendering 3D fractals.

To solve this, we implemented the sphere tracing algorithms in an interactive manner. Rendering tasks are grouped into passes, and during the rendering of a single frame, multiple passes are executed. The amount of rendering passes is determined on-the-fly using runtime statistics. Screen-sized textures are used to store the state of rendering for each pixel, such as the distance traveled along the ray. Other algorithms, like ambient occlusion and soft shadows, are implemented iteratively as well. In the next frame, the computation can be continued incrementally as long as the camera or the scene has not changed. This provides both a high-quality rendering when the camera and the scene are stationary and real-time responsiveness when they are not.

Even though the implementation is massively parallel, the resolution at which the scene is rendered is a significant factor in the final render-time. Thus, the application was equipped with the ability to change resolution between passes dynamically. Moreover, varying resolution between passes often yields shorter rendering times while achieving the same final resolution and perceptually equivalent image quality.

When rendering at a given resolution, cone-tracing is used to support resolution increase. Cone-tracing ensures that no surface point lies inside the cone. Thus, for any smaller new cone starting from where the previous cone has stopped, it is guaranteed that no intersection is skipped. The terminating condition for the pixel cone, assuming rectangular pixels, becomes

$$f(\boldsymbol{p}+t\boldsymbol{v}) < \frac{\sqrt{2}\cdot t}{\sqrt{width^2+height^2}}\,, \tag{2}$$

where $width \cdot height$ is the total number of pixels. This allows an arbitrary increase in resolution because the starting position for each new pixel is always the closest pixel in the previous resolution.

Remark 1 Eq. 2 can be incorporated into the SDFE f, so when tracing this modified function, the algorithms do not overstep the nonzero boundary. Let $\alpha := \frac{\sqrt{2}}{\sqrt{width^2+height^2}}$, then we can inspect $g(t) := f(\boldsymbol{p}+t\boldsymbol{v}) - \alpha \cdot t$. Let $\operatorname{Lip} f \leq 1$ denote the Lipschitz constant [EEJ04] of function f. Since f is an SDFE,

$$\operatorname{Lip} g \;\leq\; \operatorname{Lip} f + \operatorname{Lip}(t \to \alpha \cdot t) \;\leq\; 1+\alpha\,.$$

Therefore, the following one-dimensional SDFE can be used to approximate the distance that can be traveled safely:

$$F(t) = \frac{f(\boldsymbol{p}+t\boldsymbol{v})}{1+\alpha} - \frac{\alpha}{1+\alpha}\cdot t\,.$$

Also, there is no need for any special condition for termination. This transformation efficiently offsets the surface by a value that increases with distance.

5 Test results

All sphere tracing algorithms were implemented in GLSL using a C++ test application (`cg.elte.hu/~csabix/publications/EG2018/`). The implementation used two sets of canvas-sized textures and ping-ponging to render from one to the other. Smaller

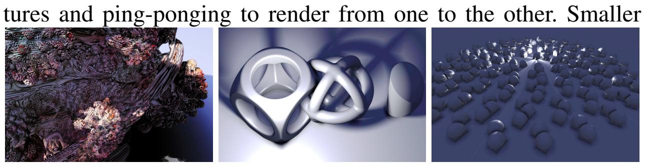

Figure 4: *Scenes used for render-time and error measurements.*

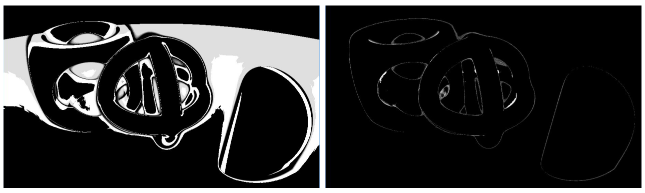
(a) *Relaxed sphere tracing* **(b)** *Enhanced sphere tracing*

Figure 5: *Ratio of oversteps between 17-24 iterations. White pixels indicate that the algorithm had to step back at every iteration.*

resolutions used a sub-rectangle inside the textures. Upon resolution change, nearest neighbor sampling was used for depth fetches, while bilinear interpolation was used for color lookups.

Render pass scheduling and time measurements for benchmarking used asynchronous GPU time queries for optimal performance and measurement consistency. In both cases, values were smoothed in time, and in the latter case, measurements were only taken after a warm-up period. For error measurements, we created a ground truth depth map of the scene using 10000 iterations of basic sphere tracing. The camera was oriented such that no ray escaped to infinity to provide a practically exact base of comparison. At each iteration, the error was measured as the squared length of the difference vector between the depth values of the ground truth and the particular algorithm.

We used the three test scenes shown in Fig. 4 for our performance tests. The first one is the Mandelbulb fractal. The second scene is a simple composition of tori, spheres, boxes, and cylinders. The third is a village with many planar surfaces implemented such that the SDF evaluation time is high. The primary indicator of performance was the execution time required to reach a prescribed error threshold w.r.t. the ground truth. The performance of enhanced sphere tracing was decisively better than that of the relaxed and basic sphere tracing algorithm, see left of Fig. 3. We used the product of error and execution time as a metric to describe an error-decrease velocity. Even though the construction of a single enhanced sphere trace step is more expensive than that of the other two sphere tracing variants, its faster convergence rate makes it outperform basic sphere tracing after 64 iterations and relaxed sphere tracing on every iteration count. Since the enhanced algorithm approximates smooth surfaces optimally, it can take longer steps and rarely branches the execution by falling back to the standard sphere trace step, see Fig. 5.

The Mandelbulb fractal was used to test performance against non-smooth surfaces. Both the relaxed and enhanced algorithms performed slightly better than the basic algorithm, but they were on par with each other within the margin of error. The combined result of the error values and render times, the green lines, indicate a consistent advantage of enhanced sphere tracing over the relaxed algorithm, up to 1.5 times better. The relative errors start to increase gradually because the relaxed and enhanced sphere tracing algorithms approximated the surface within threshold already, while standard sphere tracing was only beginning to catch up.

On the right of Fig. 3, we compare three possible render-pass schedules with variable resolutions for each of the three algorithms averaged over the three scenes, all with respect to the single-pass, single-resolution test-case. When two resolutions were used, the first one was $\frac{1}{2}$ of the final. For four resolutions, it is $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, and then the full resolution was rendered. Similarly, for six resolutions, we used the resolution ratios of $\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}, 1$.

The iterations in each render-pass sum up to the iteration count on the X-axis. However, we found that just by subdividing the iterations equally into the render-passes resulted in inferior performance. For the two render pass test case, we put $\frac{1}{8}$ of the iterations into the first pass, and the rest to the last. For the four render pass schedule, the pattern $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{5}{8}$ was used; and for the six resolution changes, we divided the iterations into passes with the pattern $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{3}{8}$.

Increasing the resolution performs differently under different iteration counts and algorithms. Using higher iteration counts, the enhanced algorithm performs better if the resolution changes less. Basic sphere tracing performs better using fewer iterations. This means that many good rendering strategies are available, some provide faster results, some better quality, and some work well with fractals. For example, the performance of enhanced sphere tracing can be increased by up to 20% using two resolutions instead of only one.

6 Conclusions

This paper presented two strategies to improve sphere tracing performance: one by altering the sphere tracing algorithm itself and the other by the manner in which it is applied.

The first approach was validated by comparing our enhanced sphere tracing variant to that of Hart and Keinert et. al. We have determined that when taking into account how fast the algorithm approximates the surface within an error threshold, our novel method can perform up to 50% better.

The second strategy, gradually increasing resolution as sphere tracing progresses, offers a possible improvement in the case of all algorithms. Nevertheless, the scale of its effect depends on the algorithm in question. Generally, we conclude that algorithms that converge slower can expect a smaller gain from the application of our multi-resolution strategy. Selecting an optimal resolution change strategy is subject to further research.

Acknowledgement The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00001).

References

[EEJ04] ERIKSSON K., ESTEP D., JOHNSON C.: *Applied Mathematics: Body and Soul*. No. 978-3-540-00890-3. Springer-Verlag, 2004. Volume 1: Derivatives and Geometry in IR3. 3

[Har94] HART J. C.: Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12* (1994), 527–545. 1, 2, 3

[JCH89] J. C. HART D. J. SANDIN L. H. K.: Ray tracing deterministic 3-d fractals. *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH* (1989), 289–296. 1

[KSK*14] KEINERT B., SCHÄFER H., KORNDÖRFER J., GANSE U., STAMMINGER M.: Enhanced Sphere Tracing. In *Smart Tools and Apps for Graphics* (2014), The Eurographics Association. 1, 2, 3