Annales Mathematicae et Informaticae
***48** (2018) pp. 5–13*
http://ami.uni-eszterhazy.hu

Interactive Rendering Framework for Distance Function Representations

Csaba Bálint, Gábor Valasek

Eötvös Loránd University
csabix.balint@gmail.com
valasek@inf.elte.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Sphere tracing, introduced by Hart in [5], is an efficient method to find ray-surface intersections, provided the surface is represented by a signed distance function (SDF) or a lower estimate of it.

This paper presents an interactive rendering framework for visualising exact and estimate SDF representations. We demonstrate the performance of the system by visualising 3D fractals and its modularity by rendering algebraic and meta surfaces. In addition, we discuss SDF estimation of algebraic surfaces.

Keywords: Computer Graphics, Signed Distance Functions, Real-time Rendering

MSC: 65D18, 68U05

1. Introduction

Rendering surfaces represented by signed distance functions (SDF) has not been in the spotlight of computer graphics research. Even though fractals have been a focus of much interest on on-line forums, literature on rendering a more general representation of surfaces, namely direct visualisation of SDFs, is scarce; the latest advancement in the field is the contribution of Keinert et. al. in [6] (2014).

A general SDF rendering engine has a far greater flexibility than incremental image synthesis based systems; even ray-tracers of practice are limited to a fixed set of surface approximations. In an SDF based rendering engine, CSG[1]-models, 3D

fractals, algebraic surfaces, and meta-surfaces can all be rendered directly without any pre-processing. This means that the surfaces appear in a considerably higher quality than any pre-processed polygon approximation.

However, the main disadvantage of using SDFs is the lower rendering speed compared to incremental image synthesis based rendering engines. Additionally, traditional ray-tracers and game engines both use the same set of primitives (usually polygons) which does not include SDFs. This paper focuses on the representation and rendering of SDFs, with emphasis on the case of algebraic surfaces.

Previous work The algorithm for rendering SDFs known as sphere-tracing was first investigated by Hart in [5] (1994). It is an iterative ray-tracing algorithm, illustrated in Figure 1. This algorithm has been commonly used for the past two decades for rendering SDFs, most notably fractals [3, 4, 7, 9].

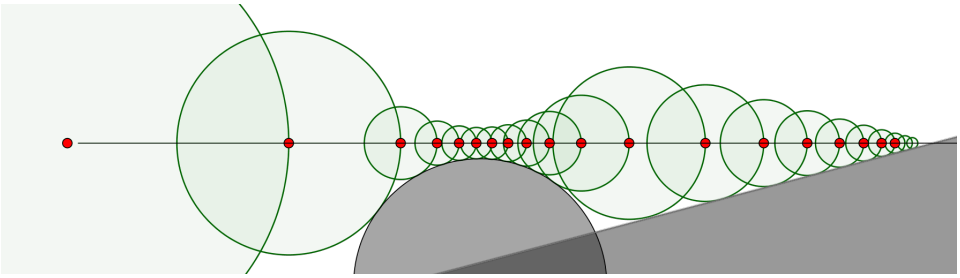


Figure 1: Illustration of the sphere-tracing algorithm in 2D.
At every point (red dot) along the ray, the distance to the surface is estimated, in this case, to the union of a half-plane and a circle. This distance defines a sphere (green circles) in which there are no intersections between the ray and the surface. Thus, sphere-tracing travels this distance along the ray to get the next estimate of the intersection point.

Following a short overview of singed distance functions, we introduce an algorithm for algebraic surface visualization. Approximating the surface normal is common problem, for witch a novel method is presented in Section 5.

2. Signed Distance Functions

In this section, definitions and notations are introduced for future reference. Definition 2.2 is from Hart's original work in [5].

Definition 2.1 (Distance to set)**.** Let (X, d) be a metric space, $x \in X$, and $A \subset X$. Then let $d(x, A) := \inf_{a \in A} d(x, a)$ (where $\inf \emptyset := +\infty$).

Definition 2.2 ((Signed) Distance Function)**.** The $f : \mathbb{R}^n \to \mathbb{R}$ function is an exact *(singed) distance function*, or (S)DF, if for any $\boldsymbol{p} \in \mathbb{R}^n$:

[1] CSG, Constructive Solid Geometry: A tree-like representation of the scene using primitive objects as leaves, set operations as nodes, and transformations as edges. [2]

$$f(\boldsymbol{p}) = d\left(\boldsymbol{p}, f^{-1}(0)\right) \quad \left(\text{or} \quad f(\boldsymbol{p}) = \begin{cases} d(\boldsymbol{p}, \text{bound}(D)) & \text{if} \quad \boldsymbol{p} \in D \\ -d(\boldsymbol{p}, \text{bound}(D)) & \text{if} \quad \boldsymbol{p} \notin D \end{cases} \right). \tag{2.1}$$

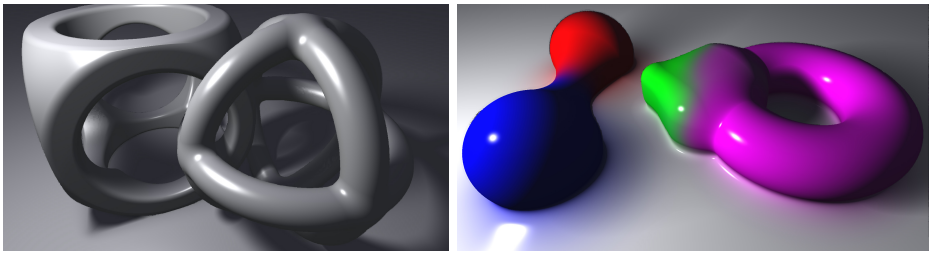
Where $\text{bound}(D) = \overline{D} \setminus \text{int}(D)$ denotes the boundary of a set.

Definition 2.3 (Distance Function Estimate)**.** The $f : \mathbb{R}^n \to \mathbb{R}$ function is a (signed) distance function estimation if and only if there exists a $q : \mathbb{R}^n \to [1, K)$ bounded ($K \in \mathbb{R}$) function, such that $f \cdot q$ is a (singed) distance function.

Remark 2.4. Besides the SDF being an upper bound to the estimate, Definition 2.3 provides a lower bound for the estimate, so sphere-tracing algorithms still converge.

The following theorem by Hart [5] describes how SDFs representing objects can be combined to create more complex geometries using CSG-like constructions. Figure 2a shows an application of the polynomial soft-min/max versions of set operations to various geometries.

Theorem 2.5 (Set operations)**.** *Let $f, g \in \mathbb{R}^n \to \mathbb{R}$ be (S)DF. Then* ***(i)*** $\{f \equiv 0\} \cup \{g \equiv 0\} = \{\min(f, g) \equiv 0\}$*,* ***(ii)*** $\{f \equiv 0\} \cap \{g \equiv 0\} = \{\max(f, g) \equiv 0\}$*,* ***(iii)*** $\{f \equiv 0\} \setminus \{g \equiv 0\} = \{\max(f, -g) \equiv 0\}$*. Additionally, the* $\min(f, g)$ *and* $\max(f, g)$ *are (singed) distance function estimates.*



(a) Soft-min/max using 3 tori, 3 cylinders, 2 spheres, 1 cube, and 1 plane (b) Meta-surface of 2 spheres, 1 cube, 1 torus, and 1 plane

Figure 2: Demonstration of the CSG model capabilities using our rendering engine

Moreover, by using different blending functions between primitive geometries one can achieve the look of different phenomena, like water [8]. Figure 2b shows meta surfaces rendered in our system.

3. Algebraic surface estimation

Let us now consider the problem of estimating SDFs to algebraic surfaces of the form $f(x, y, z) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} x^i y^j z^k \quad (a_{ijk} \in \mathbb{R})$. To construct an SDF from this, we have to use the following theorem [5]

Theorem 3.1. *If $f \in \mathbb{R}^n \to \mathbb{R}$ is a (S)DF, it is Lipschitz continuous, and* $\operatorname{Lip} f = 1$.

Therefore, for any Lipschitz continuous f function $\frac{f}{\operatorname{Lip} f}$ is a signed distance function estimate. Although algebraic surfaces are not Lipschitz continuous over $\mathbb{R}^3$, they become Lipschitz over any finite bounded subset of space. In this case, if the distance from a given point to the surface is r, the estimation would be $f(\boldsymbol{p})/\operatorname{Lip} f|_{S_r(\boldsymbol{p})} = f(\boldsymbol{p})/\operatorname{Lip}_{S_r(\boldsymbol{p})} f$, where $S_r(\boldsymbol{p})$ is the sphere with centre $\boldsymbol{p}$ and radius $r > 0$. This provides the following fixpoint-iteration

$$F(r, \boldsymbol{p}) = \frac{f(\boldsymbol{p})}{\operatorname{Lip}_{S_r(\boldsymbol{p})} f} = r. \tag{3.1}$$

Iterating F on its first argument, r, results in an estimation of the distance function.

Usually, we have to calculate the distance in a certain direction, for example along a ray. Let $\boldsymbol{s}(t) := \boldsymbol{p} + t \cdot \boldsymbol{v}$. We must calculate the Lipschitz constant of the following on a given $S_r(\boldsymbol{p})$ set.

$$f(\boldsymbol{s}(t)) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} (p_x + tv_x)^i (p_y + tv_y)^j (p_z + tv_z)^k \tag{3.2}$$

The substitution method for calculating $\operatorname{Lip}_{[-r,r]} f \circ \boldsymbol{s}$ of (3.2) is treating this expression as a $f \circ \boldsymbol{s} \in \mathbb{R}[t, p_x, p_y, p_z, v_x, v_y, v_z]$ seven variable polynomial. Multiplying out, then ordering the terms, we get $N \leq n_i + n_j + n_k + 1$ number of monomials in t. Let $P_n(\boldsymbol{p}, \boldsymbol{v}) \cdot t^n$ denote the nth monomial.

Therefore, $\operatorname{Lip}_{t \in [-r,r]} (P_n(\boldsymbol{p}, \boldsymbol{v}) \cdot t^n) \leq n \cdot r^{n-1} |P_n(\boldsymbol{p}, \boldsymbol{v})|$ is the estimate of the Lipschitz constant of the nth monomial[2], where r is from (3.1), and the sum of these is the upper-estimate of the Lipschitz constant of f.

The problem with this approach is that in practice, we have to be able to make symbolic calculations within the engine and generate GPU code based on the algebraic surface given.

4. Taylor-series method

Our method is based on the fact that a Taylor expansion of a polynomial is itself. To calculate $P_n(\boldsymbol{p}, \boldsymbol{v})$ first we note that $P_n = \frac{1}{n!}(f \circ s)^{(n)}(0)$. Now, let us find an efficient way to compute the nth derivative of $f \circ \boldsymbol{s}$. Let

$$g_{ijk}(t) := (p_x + tv_x)^i (p_y + tv_y)^j (p_z + tv_z)^k \quad (t \in [-r, r]) , \tag{4.1}$$

so $P_n = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} \frac{g_{ijk}^{(n)}(0)}{n!}$. Let $h_{ijk}(t) := \frac{iv_x}{p_x + tv_x} + \frac{jv_y}{p_y + tv_y} + \frac{kv_z}{p_z + tv_z}$.

Note that $g'_{ijk} = g_{ijk} \cdot h_{ijk}$, so $g_{ijk}^{(n+1)} = \sum_{m=0}^{n} \binom{n}{m} g_{ijk}^{(m)} h_{ijk}^{(n-m)}$, where

[2] On estimating Lipschitz constants: [1]

$$h_{ijk}^{(n)}(t) = (-1)^n \cdot n! \left[i \left(\frac{p_x}{v_x} + t \right)^{-n-1} + j \left(\frac{p_y}{v_y} + t \right)^{-n-1} + k \left(\frac{p_z}{v_z} + t \right)^{-n-1} \right]. \tag{4.2}$$

Thus, $h_{ijk}^{(n)}$, $g_{ijk}^{(n)}$, and P_n can all be computed, and so is the following approximation:

$$\underset{[-r,r]}{\text{Lip}}\,(f \circ \boldsymbol{s}) = \underset{t\in[-r,r]}{\text{Lip}} \left(\sum_{n=0}^{n_i+n_j+n_k} P_n \cdot t^n \right) \leq \sum_{n=1}^{N} n \cdot r^{n-1} |P_n| \,. \tag{4.3}$$

Finally, repeating the (3.1) iteration gives us the distance estimate.

Algorithm 1: Calculating P_n

In : $\boldsymbol{p}, \boldsymbol{v}$ and f in sparce-form: $f(x,y,z) = \sum_{l=0}^{L} A_l \cdot x^{I_l} y^{J_l} z^{K_l}$ where $A \in \mathbb{R}^L$; $I, J, K \in \mathbb{N}^L$

Out : $P \in \mathbb{R}^N$ is for P_n coefficients.

Temp : $G, H \in \mathbb{R}^N$ for g_{ijk} and h_{ijk}.

$P := (f(\boldsymbol{p}), 0, 0, \ldots, 0)$;
for $l = 0 \,..\, L-1$ **do**
  $G := \left(p_x^{I_l} \cdot p_y^{J_l} \cdot p_z^{K_l}, 0, 0, \ldots, 0 \right)$;
  for $n = 1 \,..\, N-1$ **do**
    $H_{n-1} := -(-1)^n (n-1)! \cdot \left(I_l \frac{v_x^n}{p_x^n} + J_l \frac{v_y^n}{p_y^n} + K_l \frac{v_z^n}{p_z^n} \right)$;
    for $m = 0 \,..\, n-1$ **do**
      $G_n := G_n + \binom{n-1}{m} \cdot G_m H_{n-m-1}$;
    $P_n := P_n + \frac{1}{n!} A_l \cdot G_n$;

Algorithm 2: Fix-point iteration

In : The ray $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3$, and $P \in \mathbb{R}^N$ coefficient vector from Algorithm 1 is given. For better convergence, a $\lambda \in (0, 1]$ relaxation constant, and $r_0 > 0$ starting radius, eg. from linear approximation, is also given.

Out : $r > 0$ distance that can be travelled along the $\boldsymbol{s}(t) = \boldsymbol{p} + t \cdot \boldsymbol{v}$ $(t > 0)$ ray.

Temp : The Lipschitz constants will be calculated in $Lip > 0$ variable.

$r := r_0$;
for $i = 0 \,..\, iters$ **do**
  $Lip := 0$;
  for $n = 1 \,..\, N$ **do**
    $Lip := Lip + n \cdot r^{n-1} \, |P_n|$;
  $r := r \cdot (1 - \lambda) + \frac{f(\boldsymbol{p})}{Lip} \cdot \lambda$;

Figure 3: Novel algorithms for algebraic surface visualization.
First, using equations (4.1)–(4.3), the P_n coefficients of the Taylor expansion of $f \circ \boldsymbol{s}$ are calculated. Second, the fix-point iteration in (3.1) is used to find the right step size for the sphere-tracing algorithm.

Implementing this approach is easier as it does not require symbolic expressions and complex code generation, see the algorithms on Figure 3. Figure 4 summarises our results. The algorithm can be stopped at any derivative thus achieving quadratic complexity in the number of derivatives and linear in the number of terms.

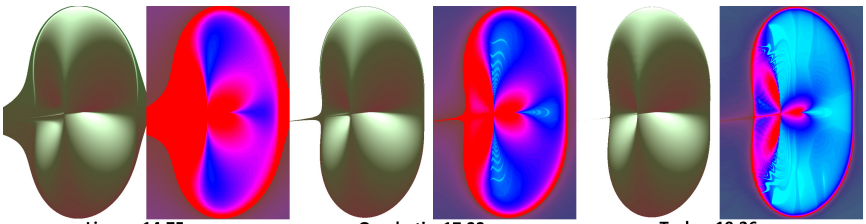


Figure 4: Comparison of SDF estimations with capped amount of steps along a ray.
The algebraic surface $f_{K_1,K_2}(x,y,z) = (x^2+y^2+z^2)(K_1x^2+K_2y^2) - 2z(x^2+y^2) = 0$ has a singular line that makes it hard to visualize from this angle. The Taylor method converges closer to the surface in less steps in about the same time as the traditional linear and quadratic SDF approximations. The light-blue means it only takes one step, and in the red region it takes 70.

5. Normal estimation

The surface normal at $\boldsymbol{p} \in \{f \equiv 0\}$ is defined as the unit vector $\mathbf{norm}(\boldsymbol{p}) = \frac{\nabla f(\boldsymbol{p})}{\|\nabla f(\boldsymbol{p})\|_2}$, for any surface defined by the differentiable implicit function $f \in \mathbb{R}^3 \to \mathbb{R}$.

In this section, we focus on calculating the normal numerically. The one-sided (forward or backward) difference method gives an error of $\mathcal{O}(\epsilon)$ for the derivative. A more accurate method is the symmetric difference:

$$\nabla f(x,y,z) = \frac{1}{2\epsilon} \cdot \begin{bmatrix} f(x+\epsilon,y,z) - f(x-\epsilon,y,z) \\ f(x,y+\epsilon,z) - f(x,y-\epsilon,z) \\ f(x,y,z+\epsilon) - f(x,y,z-\epsilon) \end{bmatrix} + \mathcal{O}\left(\epsilon^2\right) . \tag{5.1}$$

The idea of our approach is to take the following vectors (or stencil)

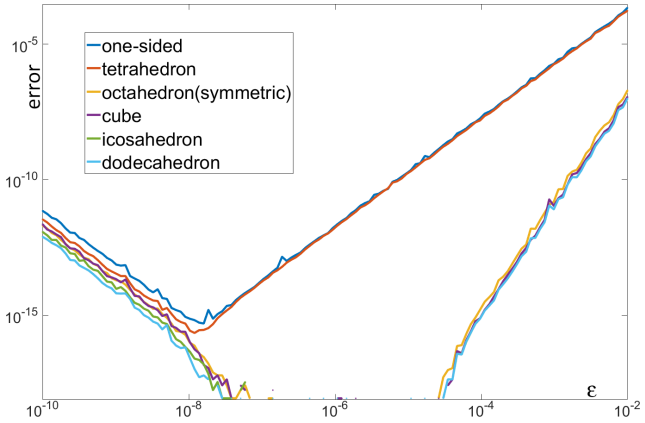
$$\boldsymbol{v}_1 := [+1,\ 0,\ 0]^\top,\ \boldsymbol{v}_3 := [0,+1,\ 0]^\top,\ \boldsymbol{v}_5 := [0,\ 0,+1]^\top,$$

$$\boldsymbol{v}_2 := [-1,\ 0,\ 0]^\top,\ \boldsymbol{v}_4 := [0,-1,\ 0]^\top,\ \boldsymbol{v}_6 := [0,\ 0,-1]^\top,$$

then (5.1) is equivalent to $\nabla f(\boldsymbol{p}) = \frac{1}{2\epsilon} \sum_{i=1}^{6} f(\boldsymbol{p} + \epsilon \cdot \boldsymbol{v}_i) \cdot \boldsymbol{v}_i$, thus we define

$$\mathbf{norm}(\boldsymbol{p}) = \frac{1}{Z} \sum_{i=1}^{6} f(\boldsymbol{p} + \epsilon \cdot \boldsymbol{v}_i) \cdot \boldsymbol{v}_i \ . \tag{5.2}$$

where $Z \in \mathbb{R}$ is the normalising constant. This means that the samples of the function are taken at the vertices of an octahedron.



(a) Error in relation to ϵ. Line breaks when the angle between the analytic and the numeric estimation is zero.

	mean	median
one-sided	119	562
tetrah.	113	843
octah.	*1.0*	*1.0*
cube	0.6	0.7
icosah.	0.5	0.1
dodecah.	0.5	0.1

	std	max
one-sided	382	306
tetrah.	265	313
octah.	*1.0*	*1.0*
cube	0.5	0.6
icosah.	0.4	0.3
dodecah.	0.4	0.3

(b) Relative error to symmetric difference for $\epsilon = 0.01$

Figure 5: Error of normal estimators measured in cosine distance[3].
Our tetrahedron kernel performs slightly better than the one-sided approach and results in a marginally lower mean error with lower variance. Cube, icosahedron and dodecahedron kernels also slightly outperformed the symmetric difference, but they also take considerably more samples.

According to our measurements, an optimal stencil vector set would consist of equal length vectors that fill the space evenly, so the best kernels in these cases

[3] $\text{cosine\_distance}(a,b) = 1 - \text{cosine\_similarity}(a,b) = 1 - \cos(\theta) = 1 - \frac{a \cdot b}{\|a\|_2 \cdot \|b\|_2}$

consisted of vertices of platonic solids. Taking every second vertex of a cube gives us the fastest kernel, the tetrahedron:

$$\boldsymbol{v}_1 := [+1, +1, +1]^\top, \boldsymbol{v}_2 := [+1, -1, -1]^\top, \boldsymbol{v}_3 := [-1, +1, -1]^\top, \boldsymbol{v}_4 := [-1, -1, +1]^\top .$$

This is as fast as the first-order divided difference, but it gave empirically better results as shown on Figure 5. Other platonic solids were also investigated.

Potentially, even higher accuracy can be achieved by sampling surface of the unit sphere with a sequence of low discrepancy, like a Halton sequence. However, this is usually not needed, because the length of the SDF estimate's gradient is usually close to one. Moreover, the normal is needed for calculating lighting effects, and small errors are not visible.

The implementation supports multiple normal calculation algorithms. The tetrahedron kernel proved to be faster than the first-order divided difference one. Symmetric or octahedron kernels introduced barely visible differences in quality along hard edges.

6. Implementation

Figure 6: Mandelbulb fractal displayed with our rendering engine.
Maximum quality was reached after 156.2ms render time. Shadows were at maximum quality after 436.2ms. GPU utilisation was 97-99%. GPU: NVidia 640M (480 GFLOPS).

The rendering engine supports operating in a progressive mode, which means when the camera is not moving, the image quality continues to increase. Therefore, the engine is optimised for static scenes. The C++ and OpenGL implementation is highly efficient achieving near 100% GPU utilisation and provides several features.

Firstly, swapping algorithms between passes was a free operation due to the OpenGL subroutines running on the GPU. This and the algorithms inter-compa-

tibility can be used for a short statistical training to determine the best schedule of algorithms for a given scene.

Secondly, a CSG model creator was also implemented. The user can either write the program computing the SDF directly or build the CSG tree from primitives and other program codes both using a built-in graphical interface.

Finally, the shader programs, including subroutines, were generated on-the-fly, thus the code for the scene geometry is embedded into the code running on the GPU. This greatly reduced both the distance function evaluation times and memory consumption.

7. Summary

This paper presented a direct signed distance function visualisation framework and its application to rendering algebraic surfaces.

We proposed a local signed distance function estimation method to such surfaces and investigated the precision of various surface normal estimation heuristics. We benchmarked the performance of the system by rendering complex scenes incorporating CSG elements, meta-surfaces, and the Mandelbulb fractal.

The framework proved to be highly efficient. In addition, it is highly modular, and outperformed current fractal-viewers [3, 4, 7, 9] in both quality and speed.

References

[1] Kenneth Eriksson, Donald Estep, and Claes Johnson. *Applied Mathematics: Body and Soul*. Number 978-3-540-00890-3. Springer-Verlag Berlin Heidelberg, 1 edition, 2004. Volume 1: Derivatives and Geometry in IR3.

[2] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design (3rd Ed.): A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[3] Paul Geisler, Daniel White, Paul Nylander, Tom Lowe, David Makin, Buddhi, Joy Leys, Knighty, and Jan Kadlec. Online fractal viewer: FractalLab. http://hirnsohle.de/test/fractalLab/, 2010.

[4] Matthew Haggett. Mandelbulb 3D (MB3D) fractal rendering software. `http://mandelbulb.com/2014/mandelbulb-3d-mb3d-fractal-rendering-software/`, 2014.

[5] John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1994.

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

[7] Krzysztof Marczak. Mandelbuilder - 3D fractal explorer. `http://www.mandelbulber.com/`, 2010.

[8] László Szécsi and Dávid Illés. Real-time metaball ray casting with fragment lists. In Carlos Andújar and Enrico Puppo, editors, *Eurographics (Short Papers)*, pages 93–96. Eurographics Association, 2012.

[9] Íñigo Quílez. Mandelbulb.
`http://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm`,
2009. (Mandelbulb in real time).