STAG: Smart Tools and Apps in computer Graphics (2016)
Giovanni Pintore and Filippo Stanco (Editors)

Kernel–Reflection Sequences

László Szécsi[1] Zoltán Bendefy[2] Ágota Kacsó[1]

[1]Budapest University of Technology and Economics
[2]NNG Llc.

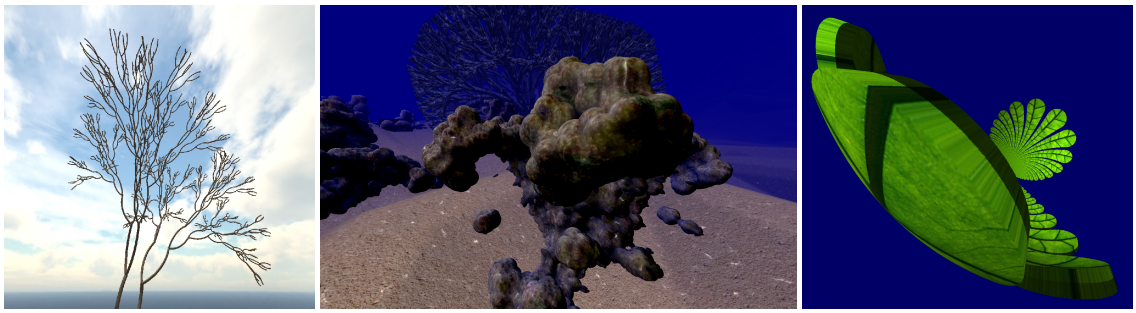

Figure 1: *Scenes modeled with Kernel–Reflection Sequences and ray-traced in real time*

Abstract

Complex geometries, like those of plants, rocks, terrain, or even clouds are challenging to model in a way that allows for real-time rendering but does not make concessions in terms of visible detail. In this paper we propose a procedural modeling approach, called KRS, or kernel–reflection sequences*, inspired by iterated function systems. The model is composed of kernel geometries defined by signed distance functions, and reflection transformations that multiply them. We show that a global distance function can be evaluated over this structure without recursion, allowing for the implementation of real-time sphere tracing on parallel hardware. We also show how the algorithm readily delivers continuous level-of-detail and minification filtering. We propose several techniques to enhance modeling freedom and avoid conspicuous symmetries. Most importantly, we extend sphere tracing to conformally transformed geometries. We also propose a GPU load balancing scheme for best utilization of computing power. To prove that the model can be used to realize various natural phenomena in uncompromising detail and extents, without obvious clues of symmetry, we render aquatic and terrestrial surface formations and vegetation in real-time.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Fractals

1. Introduction

Geometries occurring in nature typically feature intricate detail and great extents at the same time. They pose challenges throughout both the modeling and the rendering pipeline, from content creation to final visualization. The usual approach of modeling triangle meshes and rendering them through incremental rasterization fails at both stages. First, every leaf in a forest cannot be modeled manually. Creation has to be scripted, thus already requiring a procedural description of the geometry, and a way of expanding that description to a visualizable representation. Second, the resulting geometry can be too complex to be rendered in real-time by triangle rasterization. That gives emergence to level of detail techniques, which all need to address the issue of transition between different triangle mesh representations.

Ray tracing point clouds or procedural models could be a strong alternative to triangle rasterization, but it requires recursive traversal of hierarchical space subdivision structures. On GPUs, this results in divergent control flow and data access, meaning suboptimal utilization of available computing power. Thus, in this paper, we aim to propose a geometrical construct that can be ray-traced without conditional branching and divergent data access, and explore its capabilities for modeling.

DOI: 10.2312/stag.20161364

Our proposed solution uses sphere tracing on geometries composed of reflected primitive instances, resulting in a formally limited, but practically versatile procedural modeling approach. We show that distance functions for the defined models can be evaluated without recursion, and thus ray tracing of the procedural geometry can be done in real-time. As the computational complexity is superior to rasterization, rendering vastly outperforms incremental rasterization for sufficiently complex geometries.

We also extend sphere tracing to conformally mapped geometries, allowing our modeling approach to produce non-symmetric features.

2. Related work

IFSs, or *iterated function systems* [Hut79] describe a piece of self-similar geometry as the union of several transformed instances of itself, by specifying the set of transformations used. IFSs are capable of generating geometries of fractal dimensions, occasionally resembling natural phenomena, but usually with an easily discernable pattern. Our approach both restricts IFS to avoid recursion in visualization, and extends them to accommodate a wider class of natural features.

CSG, or *constructive solid geometry* defines geometries as results of regular set operations on primitives. These primitives are often given as implicit surfaces. Self-similar, natural geometries can be generated by introducing circles in the construction graph, leading to cyclic object-instantiation graphs, or CSG-PL-Systems [GT96]. Efficient ray tracing can be performed by generating bounding objects for self-similar components [TG79]. Our KRS can be considered a special case of CSG-PL-Systems, and thus it supports the expansion of the procedural model by primitive instantiation. Such algorithms for L-systems [PL90] and split grammars have been discussed by Sowers [SMM*08]. This is an option for visualizing the models we propose in this paper, but our main goal is to render images efficiently without geometry instantiation, and without an explicit choice of level of detail.

F-rep [PASS95] defines objects as sets of points for which a function is non-negative. It supports set operations and recursion. A special case is that of signed distance functions, where the function gives the exact distance from the object surface. In our solution, we use such a representation for kernel objects, which are the atomic building blocks of our models.

Sphere tracing was proposed by Hart [Har96]. It is an iterative technique for ray intersection against a geometry for which a distance function is known. This distance function must return a tight underestimation for the distance of a point and the ray-traced geometry, or, in other words, the radius of an *unbounding sphere* [HS89] centered at the point. The algorithm progresses by repeatedly advancing a point along the ray with this distance, to the surface of the unbounding sphere, eventually converging to the point of intersection. Keinert et al. proposed several enhancements to sphere tracing [KSK*14], including *over-relaxation*, the idea of which we exploit in this paper for rendering symmetric geometries under conformal transformations.

This paper extends on preliminarily results published by Szécsi [Szé14].

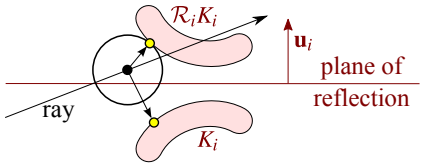

Figure 2: *The distance of the closest point and its reflected image.*

3. Kernel–reflection sequences

Let us consider a simple piece of geometry (the first *kernel*) repeatedly doubled by a sequence of planar reflections. After every doubling, we add a new piece of geometry (a different kernel), which also undergoes all the remaining reflections. This is what we call a *kernel–reflection sequence* or *KRS*. If all kernels are defined by their distance functions, the distance function of the entire KRS can be evaluated efficiently by a fixed operation count, logarithmic time algorithm. This allows fast GPU sphere tracing.

For the formal discussion, let us first consider a single kernel K_0 and a finite sequence of reflection operators $\mathcal{R}_i$, with $i \in \{0,\dots,n-1\}$. The sequence of *expansion level* geometries is as simple as:

$$A_{i+1} = A_i \cup \mathcal{R}_i A_i, \qquad A_0 = K_0.$$

For sphere tracing, the kernel set must be defined by the signed distance function $k_0(\mathbf{x})$, which gives the geometric distance between point $\mathbf{x}$ and kernel set K_0. For tracing the KRS, we need an underestimation $l_{i+1}(\mathbf{x})$ for the distance between point $\mathbf{x}$ and A_{i+1}. If A_i is entirely on the negative side of the mirror plane or $\mathcal{R}_i$ (figure 2), then the geometry will be composed of two disjoint parts on the two sides, which are reflected images of each other. A simple test can tell which part is at less distance to point $\mathbf{x}$. We just need to determine if the point is behind the mirror plane, or in front of it. The closest point of the geometry must be on the same side, because any candidate point on the other side would have a reflected image that is even closer. Note that if A_i is not entirely on the negative side, but intersected by the mirror plane, we can still apply the same formula. Doing so is equivalent to discarding the part of the geometry on the positive side. We obtain an underestimation for the distance to the mirrored geometry, which is still suitable for sphere tracing.

This construction allows only for reflection-multiplied instances of the same, unscaled kernel geometry. While this might be enough to model some natural phenomena, hierarchies (like branches of a tree) and non-symmetric parts are not covered. In order to remedy this, let us add a new kernel set K_i at each iteration. These are defined by a sequence of possibly all different $k_i(\mathbf{x})$ distance functions, where $i \in \{0,\dots,n\}$. These also form a finite sequence, where L_{i+1} is recursively defined as:

$$L_{i+1} = K_{i+1} \cup L_i \cup \mathcal{R}_i L_i, \qquad L_0 = K_0.$$

Thus, an expansion level consists of two symmetric instances of the previous level, and an additional kernel set (figure 3).

Formally, a KRS is an ordered pair of two finite sequences, one consisting of kernel sets, and another of reflection operators:

$$\mathrm{KRS} = (K_0,\dots,K_n;\mathcal{R}_0,\dots,\mathcal{R}_{n-1}).$$

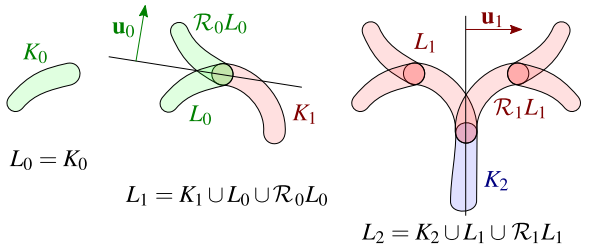

Figure 3: *A KRS expansion level is composed of reflected instances of the previous level, and a new kernel.*

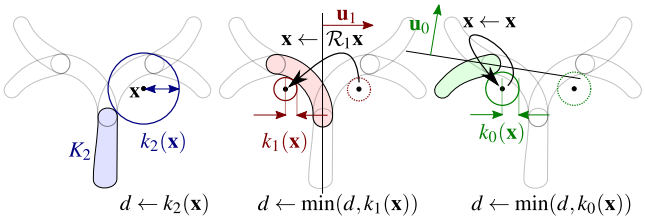

Figure 4: *The three iterations required to find distance to a three-level KRS. The distance d gathers the minimum of the kernel-space distances. The world-space point $\mathbf{x}$ is mapped to the next kernel space either through the identity or a reflection. Colored circles show the unbounding sphere with respect to the corresponding kernel. Dashed circles are their world space equivalents.*

The distance function for a KRS is:

$$l_{i+1}(\mathbf{x}) = \min\left(k_{i+1}(\mathbf{x}), l_i(\mathbf{x}), l_i(\mathcal{R}_i\mathbf{x})\right), \qquad l_0(\mathbf{x}) = k_0(\mathbf{x}).$$

We do not have to compute all the terms to evaluate the formula. As L_i and $\mathcal{R}_iL_i$ are known to be mirrored images, we can decide which distance is going to be smaller by finding on which side of the mirror plane $\mathbf{x}$ is.

$$l_{i+1}(\mathbf{x}) = \begin{cases} \min(k_{i+1}(\mathbf{x}), l_i(\mathbf{x})) & \text{if } \mathbf{u}_i \cdot \mathbf{x} - w_i \le 0, \\ \min(k_{i+1}(\mathbf{x}), l_i(\mathcal{R}_i\mathbf{x})) & \text{if } \mathbf{u}_i \cdot \mathbf{x} - w_i > 0, \end{cases}$$

where $\mathbf{u}_i \cdot \mathbf{x} - w_i = 0$ is the equation of the reflection plane. The iterative algorithm to evaluate the distance is given in Algorithm 1, and Figure 4 illustrates the process.

Algorithm 1 Returns distance to the KRS at $\mathbf{x}_w$.

```
1: function DISTANCE(x_w)
2:     x ← x_w                          ▷ point in world space
3:     d ← k_n(x)                       ▷ distance to top level kernel
4:     for i ← n − 1 downto 0 do        ▷ for all levels
5:         if x·u_i − w_i > 0 then      ▷ if behind mirror
6:             x ← R_i x                ▷ transform to next level
7:         d ← min(d, k_i(x))           ▷ keep dist to nearest kernel
8:     return d
```

Figure 5: *A school of fish modeled using unaligned mirrors (left) reveals less symmetry then with aligned mirrors (right).*

4. Ray-tracing

KRSs can be considered as unions of finite realizations of modified IFSs, where the transformations are limited to identities and reflections, but different iterations can use different transformations. These are not contractions, and the sequence is divergent, but for finite recursion depth this is only a matter of a scaling factor.

Thus, there are multiple options for the GPU visualization of a KRS, as practically any IFS visualization method could be generalized. Most prominently, all kernel instance transformations can be computed, and then kernel geometries instanced with those for rendering. However, the main motivation behind the construction of KRSs is that they can be ray-traced with an iterative, non-recursive algorithm, requiring only a few registers of read-write memory, independent of the number of levels. Thus, in this paper, we focus on visualization with ray-tracing, finding the intersection points with sphere tracing.

Sphere tracing advances iteratively, evaluating the distance function, which, in our case, is performed as an iteration itself. The search is terminated when the ray has passed through the scene or when the computed distance falls below an error threshold level. This value is inversely proportional to the camera depth, and is set to assure that the final unbounding sphere, projected onto the viewport, is smaller than a pixel. A higher threshold level makes, in practice, the kernel geometries appear thicker, as points in close proximity are considered to be members. Therefore, geometries at a large distance merge into smoother formations, loosing sub-pixel details. This, combined with the smooth shading and texturing techniques we describe in sections 8.1 and 8.2, eliminates aliasing and achieves automatic, continuous level-of-detail.

5. Techniques to hide symmetry

There are two features of a KRS that strain its credibility as a natural occurrence: symmetry might be visible and identical motifs are repeating. One countermeasure is that non-symmetric kernel elements are added on every iteration. Planes of reflection should be selected randomly enough so that the eye can only be near to a few of them at the same time. If the eye is not near to the mirror plane, the symmetry of the 3D object is not perceived in a 2D image. The undesirable symmetry effects are eliminated on the global scale. Figure 5 offers a comparison.

Fighting symmetry on the local scale is more challenging. There is always a viewpoint from where two subsets are visibly the reflected images of each other. This can only be handled if the symmetry is indeed broken.

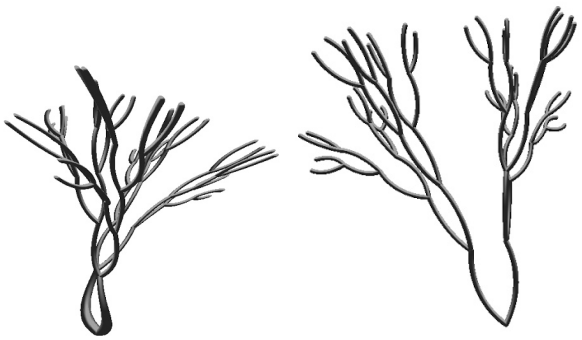

Figure 6: *A tree under twisting distortion (left) avoids the conspicuous symmetries of isometric construction (right).*

5.1. Combination of multiple KRSs

Where a single KRS cannot produce the desired effect of natural disorder, the union of multiple KRSs can. When, in a forest, there are three completely different trees between the two that are mirror images of each other, symmetry is undetected. Sphere tracing multiple KRSs can be efficiently implemented by maintaining the free distance along the ray for all components, always choosing the minimal one, and advancing it with the radius of the unbounding sphere of the respective component. The search ends when the step size for the minimal component meets the error threshold. Compared to the single KRS case, there is a slight overhead of maintaining a small sorted index of free distances for minimum selection. Otherwise, performance does not depend on the number of KRS components, but rather on the number of sphere tracing steps required. Tracing a single KRS or a compound KRS of similar complexity takes similar effort.

Procedural or projective texturing can also be considered as a way of combining features that exhibit periodicity at different frequencies. We detail techniques in section 8.2.

5.2. Distance distortion

The Lipschitz constant of KRS transformation functions is unity, resulting in exact values for the distance evaluation. By sacrificing some performance, we can handle any Lipschitz transformation of the KRS geometry as described by Hart [Har96]. The resulting geometry does not have to be symmetric any more (figure 6).

6. Sphere tracing conformally transformed geometry

Conformal transformations, which include geometric inversion, map spheres and planes to spheres and planes. If symmetric geometry is subjected to such a transformation, mirror planes are mapped to spheres, eliminating symmetry. An unbounding sphere of the symmetric geometry is mapped to an unbounding sphere of the transformed geometry. This makes it possible to find unbounding spheres for a conformally transformed KRS, even if different KRS levels are subject to different transformations. For that reason, sphere tracing geometries defined by signed distance functions under conformal transformations are of interest.

We call the space in which the signed distance function is defined the *kernel space*. The conformally distorted space in which ray tracing must be performed is the *world space*. An unbounding sphere containing a world-space point can be found by transforming the point into kernel space, and then transforming the unbounding sphere back. Note that the center of the world-space sphere is not necessarily at the original world-space point, but sphere tracing could still be performed by advancing the current point on the ray to the point of intersection with the unbounding sphere. For the sake of completeness, Appendix B provides the formula for ray–sphere intersection within the mathematical framework of conformation transformations we introduce in section 6.2.

This approach, however, does not work with the KRS scheme if we introduce a conformal transformation after every reflection. A point can only be transformed to kernel space if the mirror tests can be evaluated. As the transformed world space point is no longer the center of the unbounding sphere, it cannot be used in the mirror-side test. The actual center of the unbounding sphere, however, depends on its radius, which cannot be obtained from the kernel distance function without the kernel space position. This can be resolved if we use the idea of *sphere tracing with over-relaxation* [KSK*14], speculating on the unbounding sphere radius.

6.1. Binary sphere tracing

Given a world space *probing* sphere, we can verify if it is an unbounding sphere simply by transforming it to kernel space, by always taking the mirror test decision on the sphere center (which is known now), and checking the radius against the distance. Note that this relaxes the requirement on kernel geometries that they have to be defined by distance functions, as it is enough to merely support a binary intersection test with a sphere.

The sphere tracing process becomes a trial-and-error search finding unbounding spheres, not entirely unlike numerical root finding methods employing *binary search*. Therefore, we call this algorithm *binary sphere tracing*. It can be seen as an extreme version of sphere tracing with *over-relaxation* [KSK*14], with no option to revert to classical sphere tracing near surfaces, and without the possibility to accept an unbounding sphere that is larger than what we speculated on. These restrictions are necessary for admitting conformally transformed geometry.

Algorithm 2 Binary sphere tracing.

```
1: function BINARYTRACE(ray origin τ, direction ω)
2:     s ← 1                                          ▷ initial probe radius
3:     while s > ε do                                 ▷ until converged
4:         if ISUNBOUNDING(τ + ω·s, s) then           ▷ probe sphere
5:             s ← s · ξ_bonus                        ▷ be bolder
6:             τ ← τ + ω·2s                           ▷ advance with diameter
7:         else
8:             s ← s · ξ_penalty                      ▷ try smaller probe
       return τ
```

Algorithm 2 formalizes the process. Note that the probing sphere does not have to be centered around the last known free ray position anymore, it only has to touch it. Function ISUNBOUNDING returns

true if the probe, given by its center and radius, is indeed an unbounding sphere. In this case, the ray origin is advanced by the sphere diameter, and the radius for the next probe is increased by factor ξ_{bonus}, in order to transverse empty areas faster. Should the probe fail, the radius is reduced by factor ξ_{penalty}, and a new iteration can start. When the radius decreases below an error threshold, the process is converged and an intersection is found. We found that the algorithm performance is not strongly dependant on the values of ξ_{bonus} and ξ_{penalty}. We used 2.0 and 0.3, respectively.

In general, we can expect more iterations until convergence than regular sphere tracing, as suboptimal unbounding spheres and rejected probes must occur. This is somewhat mitigated by the fact that successful probes advance the ray by twice their distance. In order to implement function IsUnbounding for non-symmetric KRSs, we need a formalism for conformal transformations (section 6.2), and the formula for inverse transforming spheres (section 6.3).

6.2. Quaternion Möbius transformations

Conformal transformations of the three-dimensional space can be formalized as *quaternion Möbius transformations* (quaternionic, fractional, linear transformations) generated by translations, rotations, dilations, reflections and geometric inversions [BG09]. We use the usual embedding of $\mathbb{R}^3$ into quaternions as pure imaginary quaternions, with zero real part. We denote the set of quaternions with $\mathbb{H}$, and the set of purely imaginary quaternions with $\mathbb{H}_0$. As Bisi and Gentili [BG09] show, quaternionic fractional linear transformations form a group homomorphic to that of 2×2 quaternionic matrices with the mapping

$$\mathbf{H}=\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \mapsto \quad \mathcal{L}_{\mathbf{H}}(q)=(aq+b)(cq+d)^{-1}, \qquad (1)$$

where $q,a,b,c,d\in\mathbb{H}$, and $\mathcal{L}_{\mathbf{H}}(q)$ is the transformation represented by matrix $\mathbf{H}$. Möbius transformations are a special class of quaternionic fractional linear transformations. The composition of two Möbius transformations is also a Möbius transformation, the coefficients of which are obtainable as a matrix product. The matrices representing elemental transformations are given in Table 1. With these, any conformal transformation can be built.

6.3. Inverse transforming spheres

The implicit equation of a sphere or plane using quaternions has the form

$$\alpha(q\bar{q})+\beta q+\bar{q}\bar{\beta}+\gamma=0, \qquad (2)$$

where $q\in\mathbb{H}$ is the free variable, and coefficients are $\alpha,\gamma\in\mathbb{R}$ and $\beta\in\mathbb{H}_0$. For a sphere of radius $\rho\in\mathbb{R}$, centered at $\kappa\in\mathbb{H}_0$, the coefficients can be found as $\alpha=1$, $\beta=\kappa$, and $\gamma=|\kappa|^2-\rho^2$.

Let α, β, and γ be the coefficients of a sphere obtained by the transformation $\mathcal{L}_{\mathbf{H}}$. We need to find the coefficients α', β', and γ' of the original sphere. As derived in Appendix A, the formulas are:

$$\alpha'=\alpha a\bar{a}+2\,\mathrm{Re}\,\bar{c}\beta a+\gamma c\bar{c},$$

$$\beta'=\alpha\bar{b}a+\bar{d}\beta a+\bar{b}\bar{\beta}c+\gamma\bar{d}c,$$

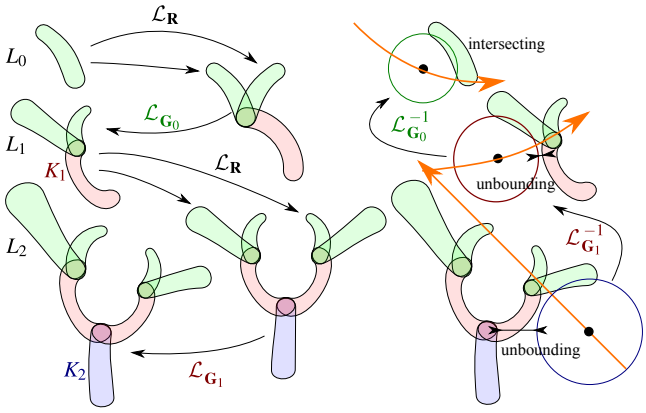

Figure 7: *MöKRS construction by conformally distorting expansion levels (left), and free distance evaluation by repeatedly transforming a probing sphere (right).*

$$\gamma'=\alpha b\bar{b}+2\,\mathrm{Re}\,\bar{d}\beta b+\gamma d\bar{d}.$$

With these, it is possible to transform a world-space sphere to kernel space, then verify if it is unbounding against the kernel distance function.

7. Möbius kernel–reflection sequences

We would like to eliminate symmetries on all levels of a KRS. Thus, we redefine an expansion level to be the *conformally transformed* union of a kernel set and reflected instances of the previous expansion level. Thus, we specify a Möbius transformation $\mathcal{L}_{\mathbf{G}_i}$ for every KRS expansion level, changing the iteration formula to

$$L_{i+1}=\mathcal{L}_{\mathbf{G}_i}\left(K_{i+1}\cup L_i\cup\mathcal{L}_{\mathbf{R}}(L_i)\right).$$

The reflection $\mathcal{R}_i$ has been replaced with $\mathcal{L}_{\mathbf{R}}$, a Möbius transformation with the matrix

$$\mathbf{R}=\begin{bmatrix} -\mathrm{i} & 0 \\ 0 & \mathrm{i} \end{bmatrix},$$

which is the reflection across the $x=0$ plane. This is without loss of generality, as any translation or rotation can be included in the Möbius transformation of the previous expansion level. We call this construct the *Möbius kernel–reflection sequence* or MöKRS. Figure 7 shows both its construction and free distance evaluation for sphere tracing.

To ray-trace this geometry, speculative sphere tracing for conformally mapped geometries is used, as formalized in Algorithm 3. For every expansion level, the current ray point is transformed with $\mathcal{L}_{\mathbf{G}_i}$ to the space of symmetry, and then to kernel space with either the identity or $\mathcal{L}_{\mathbf{R}}$, as dictated by the mirror side test. Function TransformSphere evaluates the formula detailed in Appendix A.

identity	dilation	translation	rotation	reflection across $x=0$	inversion
$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}s & 0\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}1 & v\\ 0 & 1\end{bmatrix}$	$\begin{bmatrix}r & 0\\ 0 & r\end{bmatrix}$	$\begin{bmatrix}-\mathrm{i} & 0\\ 0 & \mathrm{i}\end{bmatrix}$	$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
$\mathcal{L}(q)=q$	$\mathcal{L}(q)=sq$	$\mathcal{L}(q)=q+v$	$\mathcal{L}(q)=rqr^{-1}$	$\mathcal{L}(q)=-\mathrm{i}q\mathrm{i}$	$\mathcal{L}(q)=q^{-1}$
	$s\in\mathbb{R}$	$v\in\mathbb{H}_0$	$r\in\mathbb{H},\lvert r\rvert=1$		

Table 1: *Elementary Möbius transformations.*

Algorithm 3 Check is sphere is unbounding for MöKRS.

```
1:  function IsUnbounding(center κ_w, radius ρ_w)
2:      α ← 1                                          ▷ sphere ...
3:      β ← κ_w                                        ▷ ... equation ...
4:      γ ← |β|^2 − ρ_w^2                              ▷ ... coeffs
5:      for i ← n − 1 downto 0 do                      ▷ for all levels
6:          α, β, γ ← TransformSphere(α, β, γ, G_i)
7:          if Re(iβ) < 0 then                         ▷ x < 0
8:              β ← L_R(β)                             ▷ reflect across x = 0
9:          κ ← β/γ                                    ▷ kernel space sphere center
10:         ρ ← √(|κ|^2 − γ)                           ▷ kernel space sphere radius
11:         if k_i(κ) < ρ ∨ α < 0 then                 ▷ intersecting or inside-out
12:             return false                           ▷ not unbounding
13:     return true                                    ▷ unbounding
```

8. Combination with other real-time techniques

KRS ray-casting can only be a viable alternative to procedurally generated triangle mesh geometries if it supports all the incremental image synthesis techniques contributing to realism.

8.1. Local shading

KRS kernels are solids with surface normals defined by the gradient of their distance functions. The gradient can be used as a shading normal for surface points obtained by sphere tracing. Lighting computation can either be performed in world space or kernel space. In the former case, the transformation matrix for the complete chain must be maintained throughout the iteration, so that we can transform kernel gradients to world space. The formula for transforming the kernel gradient to world space can be found in Appendix C.

When kernels and reflections are not selected so that kernel separation is upheld, there is likely to be a visible discontinuity of surface normals where the plane of reflection intersects the mirrored geometry. These can be smoothed by interpolating between the kernel-to-world-space matrices on the two sides of a mirror, when the shaded point is found near a mirror plane during the iteration. This technique allows the generation of smooth surfaces from a kernel as simple as an infinite plane. Figure 8 compares flat and smooth shading of surfaces composed of simple planar and spherical kernels. The shading algorithm runs only once, after sphere tracing has found the intersection point.

8.2. Texturing

The kernel solids are usually simple objects (e.g. torus segments) that lend themselves to easy u,v parametrization. There are two cases when this solution is not feasible: first, if we use kernels where such a parametrization is not trivial; second, if the kernels are simplistic, like infinite planes, where even the smallest details of the KRS geometry are determined by the reflection transformations. In this second case, texturing all kernel instances with the same coordinates would produce an extremely repetitive pattern, emphasizing symmetries undesirably. Procedural 3D or triplanar [Gei07] texturing is applicable with convincing results (figure 9). Procedural geometry and procedural or wrapped textures combine to eliminate the observable repetitiveness of each other.

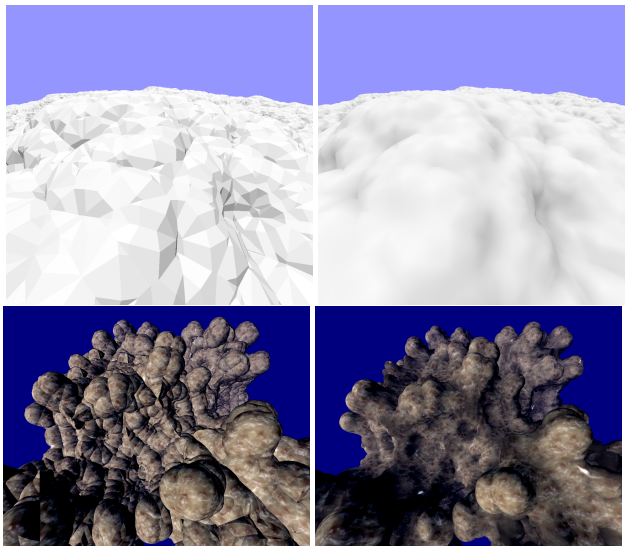

Figure 8: *Rock composed of planar and spherical kernels with flat and smooth shading.*

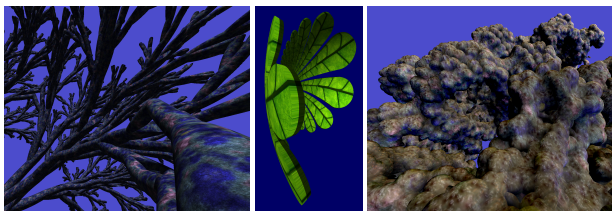

Figure 9: *Texturing with kernel parametrization on coral branches, leaves, and triplanar projection on rock.*

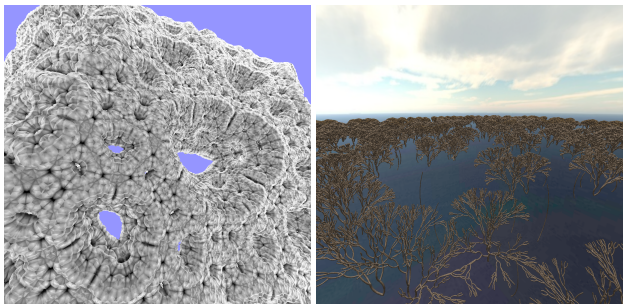

Figure 10: *Sponge with ambient occlusion, and a groove of trees.*

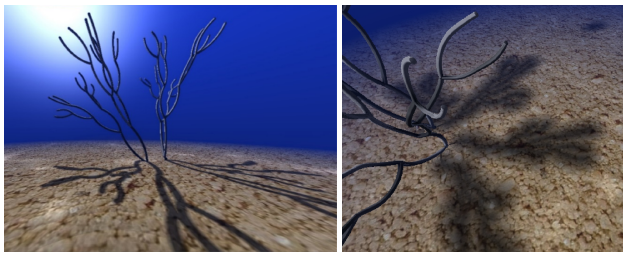

Figure 11: *Harder and softer shadows rendered using the minimum unbounding sphere radii encountered when sphere tracing shadow rays.*

8.3. Depth composition

A ray-cast KRS can easily be integrated into scenes rendered incrementally. The depth buffer can be used for early termination of rays. The ray casting shader can also output depth if it is necessary, e.g. if transparent geometry is to be rendered afterwards, or if shading is deferred.

8.4. Collision and destructibility

Based on the distance function and surface normal computation, a KRS can be integrated into any collision detection and response scheme.

A KRS is not locally controllable. However, it is possible to create *empty zones* by specifying unbounding spheres over subsets. The regions of these spheres should be skipped during sphere tracing. The subset of the KRS within these empty zones can be substituted with instances of kernel geometry rendered incrementally. These solid instances can then be subjects of any kind of physical simulation. E.g. when a branch of a tree in a forest should break, the complete tree is covered with an empty sphere, and an identical tree of rigid body branches and leaves is built. This can then be manipulated independently.

8.5. Global shading

Beyond sheer triangle throughput, KRS geometry has another key advantage over incrementally rendered geometry. Not only can sphere tracing be performed for eye rays, but also for secondary rays. Shadows, reflections, ambient occlusion (figure 10), or any global illumination techniques based on ray tracing can be implemented. An additional benefit of evaluating shadow rays with ray tracing is that convincing, albeit non-physical, soft shadows can be rendered by deriving an occlusion value from the radius of the smallest unbounding sphere when tracing the shadow ray (figure 11).

9. Modeling

KRSs appear to be limited at what can be modeled with them. At an iteration count and scale large enough for the individual kernels not to be distinguishable, the geometry tends to resemble the 3D equivalent of a Lévy C-curve [Lev59]. However, a forest from the air, a cloud in the sky, a hilly landscape, or a battered rock look exactly like that, from far enough. In this section we are going to present a few examples of application.

9.1. Coral, terrain and rock

Those natural features that are traditionally well modeled by IFS are good candidates for KRS. The variation in scale that is lost because we only use isometries is compensated by additional kernels and, in the MöKRS case, by conformal distortions. When modeling these geometries, the choice of kernel sets is also less important: spheres or infinite planes are sufficient. The separation of kernels is neither desirable nor always possible. Even the smallest details are defined by the transformations. Smooth shading and procedural or triplanar texturing greatly enhance the visual quality.

9.2. Tree and forest

A tree is a classic hierarchical structure, where the kernel sets added at each iteration become crucial. The first few kernels and reflections define the leaf geometry and the thinnest twig. Then, every new expansion level doubles this geometry, with a reflection placed so that the two main branches start from the same point. A new, thicker branch that ends at the junction is added as the next kernel. For kernels acting as branches, we used *toroidal capsules*, composed of a torus segment and two capping spheres. The coefficients of the distance functions are computed from intuitive modeling parameters. First, forking positions along one route from the trunk to a twig end must be given. Branches along this route are the kernels. Then, a control point on every such branch can be moved to set curvature. Two forking positions and a control point define the generating circle of the torus. Branch width gives the section radius. Planes of reflection must be placed at forking positions, with editable normals. A forest can be generated by adding more reflections (identical to those of the terrain, if it is also present) with empty extra kernel sets.

10. GPU scheduling

While distance evaluation is a fully unconditional iteration process, the number of steps it takes for sphere tracing to converge varies heavily for different rays. If all threads would evaluate an entire ray, thread block execution time would be determined by the most expensive ray. Therefore, evaluation of rays must be broken into *tasks*.

A task evaluates just a few sphere tracing iterations, and stores the point of intersection, if converged. If not converged yet, it inserts the remaining ray segment into a task buffer for the next turn. The process is repeated until all rays are converged, and then shading for all intersection points is performed in a deferred manner. Shadows are added easily by creating shadow ray tasks.

This scheme also works well with compound KRS models described in section 5.1, where component models are likely to use different code for kernel distance evaluation, thus requiring different GPU programs. We maintain multiple task buffers, one for every component type. We also maintain the per-component free distances for every ray, and a small sorted index for faster minimum selection. Before every turn, for every ray, the component with the minimal free distance is chosen, and either a task is inserted into the respective buffer, or, if the ray has already converged, the point of intersection is stored for shading. Then, one after the other, all task buffers are processed. When a task is finished, the free distance and the index are updated.

For primary rays, the coherence can be exploited by performing the first few steps of sphere tracing at a lower resolution. The current point on the ray can only be advanced as far as the frustum of the low-res pixel is entirely covered by the unbounding sphere.

11. Results

We measured performance on an nVidia GeForce Titan X video card, on full-screen 1920×1200 resolution. Frame rates heavily depend on the camera view, with significantly shorter frame times when there are empty zones in the image. We compared the following cases:

1. single KRS, sphere traced in the pixel shader of a full-screen quad,
2. single KRS, binary sphere traced in the pixel shader of a full-screen quad,
3. single MöKRS, binary sphere traced in the pixel shader of a full-screen quad,
4. single KRS, sphere traced with compute shaders, with low-resolution acceleration phase,
5. compound of eight KRSs, using different models, sphere traced with compute shaders, with low-resolution acceleration phase and GPU load balancing.

Figure 12 plots the shader execution times versus the number of expansion levels considered in the KRS models. The most important observation is that even though the geometry expands exponentially, the rendering times only increase linearly. Direct sphere tracing of linear KRS geometries is able to maintain 15 FPS up to 28 expansion levels, which corresponds to half a billion toroidal primitives, which could be rendered by rasterizing hundreds of billions of triangles.

Binary sphere tracing performed poorly in comparison, as approximately twice as many iterations were required, and a mere 14 expansion levels fit into 15 FPS. This is not only due to the trial-and-error nature of binary tracing, but also the ray termination error threshold has to be set much lower to avoid artifacts. Sphere tracing a continuous distance function does not introduce surface discontinuities even with a high error threshold, it only makes surfaces

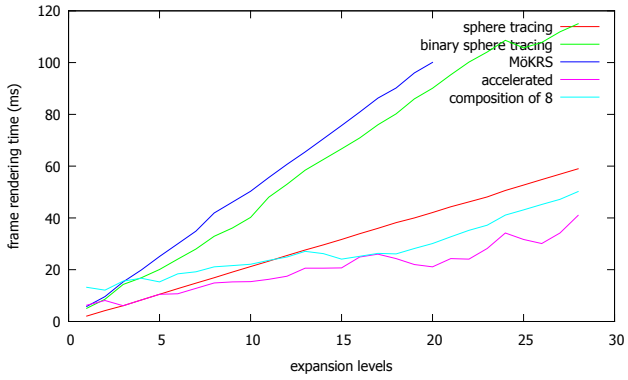

Figure 12: *Shader execution times versus the number of expansion levels.*

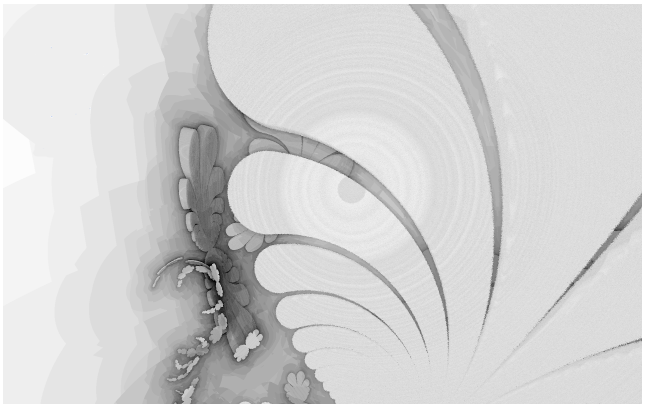

Figure 13: *Number of iterations plotted for a palm leaf MöKRS, traced straightforwardly without low-res optimization or GPU scheduling, with binary sphere tracing. White and black correspond to* 0 *and* 70 *iterations, respectively.*

appear slightly inflated. Conversely, binary tracing produces discontinuities where unbounding sphere decisions diverge. Figure 13 depicts the distribution of iterations steps in image space.

Möbius transformations do not impose significant further penalty, but as they require binary sphere tracing, rendering them is much more demanding than linear KRSs. So while it is true that MöKRS systems offer more freedom in modeling and still scale linearly with expansion levels, with today's top hardware they are not yet fully competitive with rasterization in terms of triangle throughput. We believe this will inevitably change in a few generations of graphics accelerators, as MöKRS rendering performance will increase exponentially against rasterization throughput.

Our low-resolution acceleration phase offers a speedup of 30-50% over straightforward per-pixel ray tracing. Rendering a compound KRS of eight components has negligible overhead compared to rendering a single KRS of similar kernel count. However, our implementation relies on a highly optimized component sorting rou-

tine which packs three-bit component indices and component termination flags into a single 32-bit integer register. Thus, combining a higher number of KRS models may be more expensive. However, just eight components are completely sufficient to create a coral reef scene with seabed terrain, rock formations, coral, and fish.

None of our measurements included shadows, ambient occlusion, or Lipschitz distortion. We could render soft shadows at quarter-resolution for 25% of the cost of the primary ray casting. Ambient occlusion is equivalent in cost to computing one additional expansion level. Cost impact of Lipschitz distortions may vary, the same considerations as for original sphere tracing apply.

12. Conclusions

KRS rendering can be used together with incremental image synthesis in real-time applications. It is capable of modeling a wide range of natural geometries, without the possibility of local control, but with fine details and large extent at the same time. Features that could only be represented by billions of triangles, like grasslands or forests, can be rendered in real time. Thus, KRS can add the desired natural richness of detail to virtual worlds without the need for customized level-of-detail techniques.

Animation of KRS geometries is left for future work. While animation of the model parameters is unlikely to produce credible motion, subtle changes to low-index transformations might be acceptable. Time-dependent Lipschitz transformations appear more promising.

We did not do extensive analysis on the choice of constants ξ_{penalty} and ξ_{bonus} in the binary sphere tracing algorithm. It may be interesting to explore what factors could influence where their optimum is. While we proposed an intutive editing scheme for tree-like branching models, implementing similar methods for other varieties of KRSs requires more work.

Acknowledgments

This work has been supported by OTKA PD-104710 and OTKA K-104476.

References

[BG09] BISI C., GENTILI G.: Möbius transformations and the Poincaré distance in the quaternionic setting. *Indiana Univ. Math. J. 58* (2009), 2729–2764. 5

[Dza12] DZAGNIDZE O.: On the differentiability of quaternion functions. *arXiv preprint arXiv:1203.5619* (2012). 10

[Gei07] GEISS R.: *Generating complex terrains using the GPU*. Addison-Wesley Professional, 2007, ch. 1, pp. 7–37. 6

[GT96] GERVAUTZ M., TRAXLER C.: Representation and realistic rendering of natural phenomena with cyclic CSG graphs. *The Visual Computer 12*, 2 (1996), 62–74. 2

[Har96] HART J.: Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10 (1996), 527–545. 2, 4

[HS89] HART J., SANDIN D.: Louis H Kauffman t. Ray Tracing Deterministic 3D Fractals. *Computer Graphics 23*, 3 (1989). 2

[Hut79] HUTCHINSON J. E.: *Fractals and self similarity*. University of Melbourne.[Department of Mathematics], 1979. 2

[KSK*14] KEINERT B., SCHÄFER H., KORNDÖRFER J., GANSE U., STAMMINGER M.: Enhanced Sphere Tracing. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference* (2014), Giachetti A., (Ed.), The Eurographics Association. doi:10.2312/stag.20141233. 2, 4

[Lev59] LEVY E.: Complex-curve fitting. *Automatic Control, IRE Transactions on*, 1 (1959), 37–43. 7

[PASS95] PASKO A., ADZHIEV V., SOURIN A., SAVCHENKO V.: Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer 11*, 8 (1995), 429–446. 2

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer-Verlag New York, Inc. New York, NY, USA, 1990. 2

[SMM*08] SOWERS B., MENZIES T., MCGRAW T., ROSS A., MORGANTOWN W.: *Increasing the Performance and Realism of Procedurally Generated Buildings*. ProQuest, 2008. 2

[Szé14] SZÉCSI L.: A geometry model for logarithmic-time rendering. In *Seventh Hungarian Conference on Computer Graphics and Geometry* (2014), Szirmay-Kalos L., Renner G., (Eds.), John von Neumann Computer Society. 2

[TG79] TRAXLER C., GERVAUTZ M.: Efficient ray tracing of complex natural scenes. *Proceedings Fractals 97* (1979). 2

Appendices

Appendix A: Inverse Möbius transformation of spheres

Let α, β, and γ be the coefficients of a sphere obtained by the transformation $\mathcal{L}$, and let us find the coefficients α', β', and γ' of the original sphere.

From Equation 2 it is true that

$$\alpha\left(\mathcal{L}(q)\overline{\mathcal{L}(q)}\right)+\beta\mathcal{L}(q)+\overline{\mathcal{L}(q)}\bar{\beta}+\gamma=0,$$

and substituting the definition of the Möbius transformation (Equation 1) we get

$$\begin{aligned}&\alpha\left((aq+b)(cq+d)^{-1}\overline{(aq+b)(cq+d)^{-1}}\right)\\&+\beta(aq+b)(cq+d)^{-1}\\&+\overline{(aq+b)(cq+d)^{-1}}\bar{\beta}\\&+\gamma=0.\end{aligned}$$

Using the identities $\overline{pq}=\bar{q}\bar{p}$ and $q^{-1}=\frac{\bar{q}}{q\bar{q}}$, we can write

$$\begin{aligned}&\alpha\frac{(aq+b)(cq+d)\overline{(cq+d)}\,\overline{(aq+b)}}{(cq+d)\overline{(cq+d)}}\\&+\frac{\beta(aq+b)\overline{(cq+d)}}{(cq+d)\overline{(cq+d)}}\\&+\frac{(cq+d)\overline{(aq+b)}\bar{\beta}}{(cq+d)\overline{(cq+d)}}\\&+\gamma=0.\end{aligned}$$

Now let us multiply with $\overline{(cq+d)}$ from the left and $(cq+d)$ from the right. We get

$$\begin{aligned}&\alpha(aq+b)\overline{(aq+b)}\\&+\overline{(cq+d)}\beta(aq+b)\\&+\overline{(aq+b)}\bar{\beta}(cq+d)\\&+\gamma\overline{(cq+d)}(cq+d)=0.\end{aligned}$$

Further expansion gives

$$\begin{aligned}&\alpha\left(a\bar{a}q\bar{q}+\bar{b}aq+\bar{q}\bar{a}b+b\bar{b}\right)\\+&\left(\bar{q}\bar{c}\beta aq+\bar{q}\bar{c}\beta b+\bar{d}\beta aq+\bar{d}\beta b\right)\\+&\left(\bar{q}\bar{a}\bar{\beta}cq+\bar{q}\bar{a}\bar{\beta}c+\bar{b}\bar{\beta}cq+\bar{b}\bar{\beta}d\right)\\+&\gamma\left(c\bar{c}q\bar{q}+\bar{d}cq+\bar{q}\bar{c}d+d\bar{d}\right)=0.\end{aligned}$$

Let us notice that $\bar{d}\beta b+\bar{b}\bar{\beta}d=2\,\mathrm{Re}\,\bar{d}\beta b$, and similarly $\bar{q}\bar{c}\beta aq+\bar{q}\bar{a}\bar{\beta}cq=q\bar{q}2\,\mathrm{Re}\,\bar{c}\beta a$. Then we can rearrange to get

$$\begin{aligned}&q\bar{q}\left(\alpha a\bar{a}+2\,\mathrm{Re}\,\bar{c}\beta a+\gamma c\bar{c}\right)\\+&\left(\alpha\bar{b}a+\bar{d}\beta a+\bar{b}\bar{\beta}c+\gamma\bar{d}c\right)q\\+&\bar{q}\left(\alpha\bar{a}b+\bar{c}\beta b+\bar{a}\bar{\beta}d+\gamma\bar{c}d\right)\\+&\left(\alpha b\bar{b}+2\,\mathrm{Re}\,\bar{d}\beta b+\gamma d\bar{d}\right)=0,\end{aligned}$$

which is the equation of the original sphere. Thus, the coefficients are:

$$\begin{aligned}\alpha'&=\alpha a\bar{a}+2\,\mathrm{Re}\,\bar{c}\beta a+\gamma c\bar{c},\\\beta'&=\alpha\bar{b}a+\bar{d}\beta a+\bar{b}\bar{\beta}c+\gamma\bar{d}c,\\\gamma'&=\alpha b\bar{b}+2\,\mathrm{Re}\,\bar{d}\beta b+\gamma d\bar{d}.\end{aligned}$$

Appendix B: Ray–sphere intersection

Let the equation of the sphere or plane be:

$$\alpha_w(q\bar{q})+\beta_w q+\bar{q}\bar{\beta}_w+\gamma_w=0,$$

and the equation of the ray embedded in quaternions is:

$$q(t)=\tau+\omega t,$$

with $\tau,\omega\in\mathbb{H}_0$ being the ray origin and direction, and $t\in\mathbb{R}$ the ray parameter. Substituting the ray equation in the sphere equation gives:

$$\begin{aligned}&\alpha_w\tau\bar{\tau}+\alpha_w t\tau\bar{\omega}+\alpha_w t\bar{\tau}\omega+\alpha_w t^2\omega\bar{\omega}\\+&\beta_w\tau+t\beta_w\omega\\+&\bar{\tau}\bar{\beta}_w+t\bar{\omega}\bar{\beta}_w\\+&\gamma_w=0.\end{aligned}$$

This can be simplified to:

$$\begin{aligned}&\alpha_w\tau\bar{\tau}+2\alpha_w t\,\mathrm{Re}\,\tau\bar{\omega}+\alpha_w t^2\\+&2\,\mathrm{Re}\,\beta_w\tau+2t\,\mathrm{Re}\,\beta_w\omega\\+&\gamma_w=0.\end{aligned}$$

This can be arranged into a quadratic equation for t:

$$\begin{aligned}&\alpha_w\,t^2\\+&\left(2\alpha_w\,\mathrm{Re}\,\tau\bar{\omega}+2\,\mathrm{Re}\,\beta_w\omega\right)t\\+&\alpha_w\tau\bar{\tau}+2\,\mathrm{Re}\,\beta_w\tau+\gamma_w\quad=0.\end{aligned}$$

This can be solved for t to obtain the ray parameters of the intersections.

Appendix C: Möbius transformation of gradients

Let $g\in\mathbb{H}_0$ be the gradient at point q of a distance field, and $\mathcal{L}$ a M obius transformation. Let us find gradient g_w of the transformed distance field at point $q_w=\mathcal{L}(q)$. The gradient g_w can be formulated as differential change of q_w in response to a differential change of q along g.

$$g_w=\left.\frac{\mathrm{d}\mathcal{L}(q+g\varepsilon)}{\mathrm{d}\varepsilon}\right|_{\varepsilon=0}.$$

There exist several definitions of differentiation over quaternions [Dza12]. In our simple case, however, it is sufficient to state that non-commutative linearity, the product rule, and the chain rule apply, with the reciprocal rule being

$$\frac{\mathrm{d}f^{-1}(x)}{\mathrm{d}x}=-f^{-1}(x)\frac{\mathrm{d}f(x)}{\mathrm{d}x}f^{-1}(x).$$

Let us expand $\mathcal{L}$ as defined in Equation 1:

$$g_w=\left.\frac{\mathrm{d}\left[a(q+g\varepsilon)+b\right]\left[c(q+g\varepsilon)+d\right]^{-1}}{\mathrm{d}\varepsilon}\right|_{\varepsilon=0}.$$

We now apply the product rule, and immediately substitute $\varepsilon=0$ into the derivatives. We obtain:

$$g_w=ag(cq+d)^{-1}+(aq+b)\left.\frac{\mathrm{d}\left[c(q+g\varepsilon)+d\right]^{-1}}{\mathrm{d}\varepsilon}\right|_{\varepsilon=0}.$$

With the reciprocal rule this becomes:

$$g_w=ag(cq+d)^{-1}-(aq+b)(cq+d)^{-1}cg(cq+d)^{-1}.$$

Now $g(cq+d)^{-1}$ can be factored to the right:

$$g_w=\left[a-(aq+b)(cq+d)^{-1}c\right]g(cq+d)^{-1}.$$

If both q and q_w are known, then this can be written in a simpler form:

$$g_w=(a-q_w c)g(cq+d)^{-1}. \tag{3}$$

Note that normalization is required to get a unit length pure quaternion.

If $\mathcal{L}$ transforms from kernel space to world space, and kernel-space position q has been obtained by inverse transforming world-space position q_w (or a small sphere that contains it), then the world-space gradient g_w can be obtained from the gradient g defined by kernel geometry using the formula 3.