Adv. Appl. Clifford Algebras (2021) 31:16
© The Author(s) 2021
0188-7009/020001-33
published online February 11, 2021
https://doi.org/10.1007/s00006-021-01117-8

Advances in
Applied Clifford Algebras

Exploring Novel Surface Representations via an Experimental Ray-Tracer in CGA

Hugo Hadfield*, Sushant Achawal, Joan Lasenby,
Anthony Lasenby and Benjamin Young

Abstract. Conformal Geometric Algebra (CGA) provides a unified representation of both geometric primitives and conformal transformations, and as such holds significant promise in the field of computer graphics. In this paper we implement a simple ray tracer in CGA with a Blinn–Phong lighting model, before putting it to use to examine ray intersections with surfaces generated from the direct interpolation of geometric primitives. General surfaces formed from these interpolations are rendered using analytic normals. In addition, special cases of point-pair interpolation, which might find use in graphics applications, are described and rendered. A closed form expression is found for the derivative of the square root of a scalar plus 4-vector element with respect to a scalar parameter. This square root derivative is used to construct an expression for the derivative of a pure-grade multivector projected to the blade manifold. The blade manifold projection provides an analytical method for finding the normal line to the interpolated surfaces and its use is shown in lighting calculations for the ray tracer and in generating vertex normals for exporting the evolved surfaces as polygonal meshes.

Mathematics Subject Classification. Primary 99Z99, Secondary 00A00.

Keywords. Conformal geometric algebra, Ray-tracing, Direct object interpolation, Surface curvature, Mesh geometry.

1. Introduction

Tubular and ribbon surfaces have wide interest in fields such as neuronal modelling and streamline visualisation. The need to represent vast networks

This article is part of the ENGAGE 2019 Topical Collection on Geometric Algebra for Computing, Graphics and Engineering edited by Linwang Yuan (EiC), Werner Benger, Dietmar Hildenbrand, and Eckhard Hitzer.

*Corresponding author.

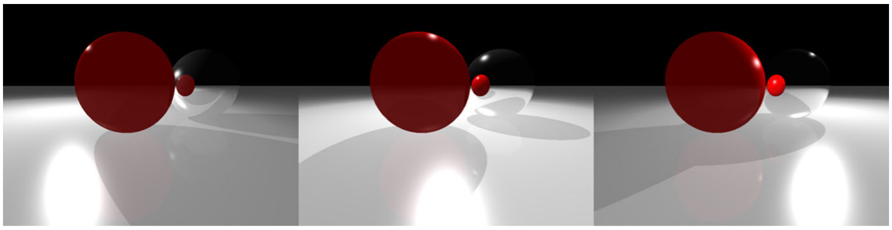


Figure 1. Three images rendered with varying lighting positions. These demonstrate the lighting model, multiple light source capability and recursive tracing of rays for reflections

of tubular data efficiently and render these surfaces in a visually pleasing way has led to a range of different parametric representations, fitting methods and rendering techniques [2,23,24]. Conformal Geometric Algebra (CGA) encodes circles and line-segments, as well as planes, spheres, infinite lines and the geometric transformations between them, as natural elements of an algebra [13,16,22]. Given its representational power for curved surfaces and simple encoding of complicated operations, CGA appears to hold great promise in the field of Computer Graphics. Indeed several ray-tracers/path-tracers/sphere-marchers using CGA have been implemented in the past [5,6,10,11,13,19,25]. More recently the design of more intricate surfaces has been investigated with rotors [8] and direct-interpolation of geometric primitives [15]. In this paper we will press some of these techniques into use to describe tubes and ribbons as well as to develop the techniques required to render them. Figure 1 shows an example of output from the CGA ray-tracer we describe in this paper.

2. Conformal Geometric Algebra, CGA

The ray-tracer used in this paper is constructed using CGA and all algebraic expressions given will be in terms of elements of this algebra. CGA adds two more basis vectors, e and $\bar{e}$, to the original basis vectors of 3D Euclidean space, giving a complete basis for the 5D space with the following signature: $e_1^2 = e_2^2 = e_3^2 = e^2 = 1$ and $\bar{e}^2 = -1$. These extra basis vectors are used to define two null vectors: $n_\infty = e + \bar{e} \equiv n$ and $n_0 = \frac{\bar{e}-e}{2} \equiv -\frac{\bar{n}}{2}$—note that the $(n, \bar{n})$ notation was that originally used when Hestenes first introduced this model in [17]. The mapping from a 3D vector, x, to its corresponding CGA vector, X, is given by:

$$X = F(x) = \frac{1}{2}\left(x^2 n + 2x - \bar{n}\right) \equiv \frac{1}{2}x^2 n_\infty + x + n_0. \quad (1)$$

All vectors formed from such a mapping are null. CGA is chosen for the construction of the ray-tracer since we seek neat expressions for describing intersections, reflections and lighting models, made possible in CGA since

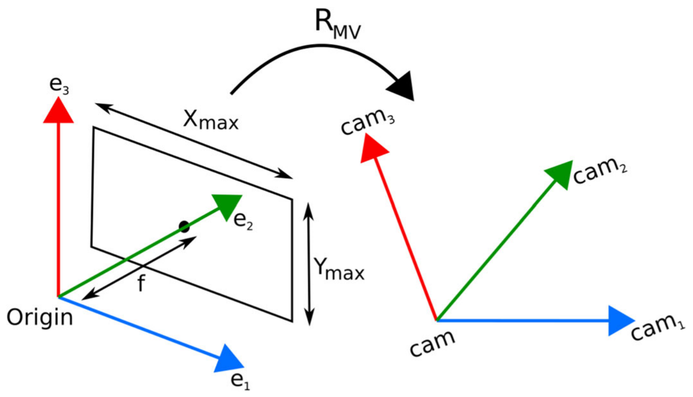


Figure 2. The camera is defined by a focal length, a transformation from the origin, and bounds on the image plane

rays and scene objects are both elements of the algebra. More background on CGA can be found in [13,16,22].

3. Camera Model and Ray Casting

A pinhole camera model is used with the geometry shown in Fig. 2. It is defined by a rotor R_{MV} (where MV indicates *model view*) incorporating rotation and translation that takes the camera from the origin to its pose in space, a focal length f and two bounds $x_{\max}$ and $y_{\max}$ on the size of the image plane.

We take $(i, j) = (0, 0)$ to be at the bottom left hand corner of the image. For an image of width w and height h, the world coordinates of the point P_{ij} at the centre of pixel (i, j) are given by:

$$\begin{aligned} W_{i,j} &= F\left(fe_2 - \frac{x_{\max}}{2}(1-(2i/w))e_1 - \frac{y_{\max}}{2}(1-(2j/h))e_3\right), \\ P_{ij} &= R_{MV} W_{i,j} \tilde{R}_{MV}. \end{aligned} \tag{2}$$

We then generate the ray from the camera centre, L_{ij}, that passes through P_{ij}, via the expression

$$L_{ij} = X_0 \wedge P_{ij} \wedge n_\infty$$

where X_0 is the origin transformed by the model-view rotor R_{MV} to the position of the camera.

4. Ray Geometries for Basic Objects

Initially we will start with some basic objects representable as blades in CGA. The ray-tracer will thus initially concentrate on rendering planes, spheres and circles/discs, an example of which is shown in Figs. 1 and 3.

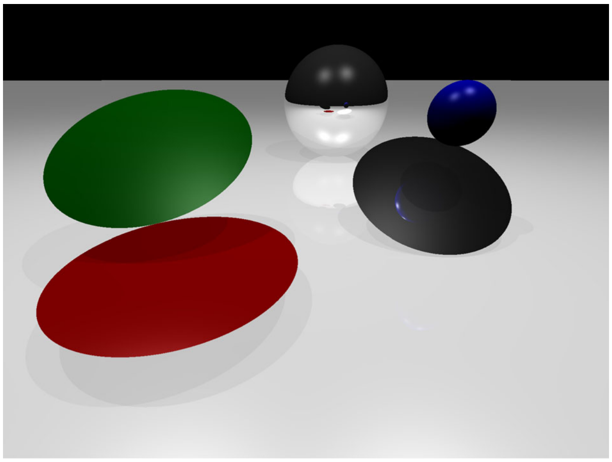


Figure 3. An image from the ray-tracer containing examples of disks, spheres and planes

4.1. Ray-Object Intersections

In order to compute intersections between blades, the *meet* operator ($\vee$) is used. We will, for the proposes of this paper, always take the meet with respect to the full 5D space rather than to the join of the blades. Thus, if X is an r-grade blade, Y is an s-grade blade, and the number of basis vectors in the algebra is n, then :

$$X \vee Y = \langle XY \rangle_{2n-r-s} I_5, \tag{3}$$

where $\langle Z \rangle_m$ indicates the m-grade component of the multivector Z, and I_5 represents the 5D pseudoscalar of the algebra [22].

4.1.1. Planes. A plane is a 4-blade and a ray is a 3-blade so the meet gives a 2-blade. If the meet itself is 0, the line lies in the plane. If the meet squared is 0, there is no finite intersection. Otherwise, the intersection point, X, of a line L with a plane Φ, satisfies the following: $L \vee \Phi = \lambda X \wedge n_\infty$ [22], where λ is a scalar. When extracting the 3D intersection point x, we need to account for the sign and magnitude of the line in our extraction, we can do this via the constant of proportionality λ:

$$L \vee \Phi = \lambda X \wedge n_\infty = \lambda x \wedge n_\infty - \lambda n_\infty \wedge n_0. \tag{4}$$

Therefore, x can be extracted from the $e_i e$ and $e_i \bar{e}$ coefficients (for $i \in \{1, 2, 3\}$) by dividing by λ, the $e\bar{e}$ coefficient.

4.1.2. Spheres. Spheres are also 4-blades and so once again, taking the meet with a ray gives a 2-blade, F. With spheres, there can be zero, one or two points of intersection corresponding to the cases where $F^2 < 0$, $F^2 = 0$ and $F^2 > 0$ respectively.

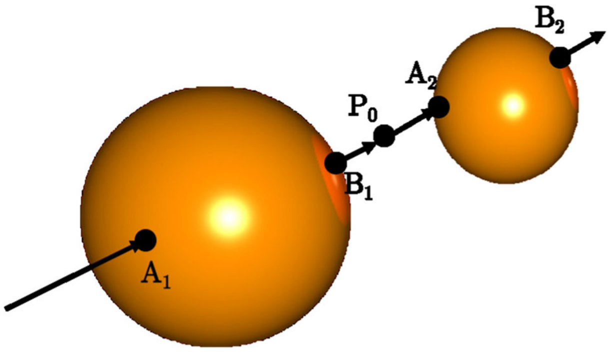


FIGURE 4. Image showing the positions of intersection points with spheres

If $F = A \wedge B$ (with A and B null vectors) and $F^2 \geq 0$, the points can be extracted from the point pair/blade, F, by the following formula [22]:

$$\sigma_a A = \left(1 - \frac{F}{\sqrt{-F\tilde{F}}}\right)(F \cdot n_\infty), \quad \sigma_b B = \left(1 + \frac{F}{\sqrt{-F\tilde{F}}}\right)(F \cdot n_\infty),$$
$$A = \frac{-\sigma_a A}{(\sigma_a A) \cdot n_\infty}, \qquad B = \frac{-\sigma_b B}{(\sigma_b B) \cdot n_\infty}, \tag{5}$$

where σ_a, σ_b are scalar constants. If we define $F = A \wedge B$ with F oriented in the same direction as our ray L, then A is the point closest to the origin of the ray, P_0, as long as our sphere is 'in front of' the ray source. To ensure the alignment we can pre-normalise our sphere S via the following expression:

$$S \longrightarrow -\frac{S}{S^* \cdot n_\infty}. \tag{6}$$

For any given sphere, its dual can square to a positive or negative number; however, by carrying out the normalisation in Eq. (6), we ensure that all spheres, S, satisfy $S^* \cdot n_\infty = -1$. If the meet of a ray L and a normalised sphere S, is then formed from $L \vee S$, as in Eq. (3), the resulting bivector will be ordered as $A \wedge B$ where A is the point that the ray hits first in its orientation.

In Fig. 4, the meet of the ray (direction as shown) from a point P_0 with the smaller sphere, would result in the point pair $A_2 \wedge B_2$. For the larger sphere in Fig. 4, the point pair resulting from the meet will be $A_1 \wedge B_1$. We extract the points from the point pair and form the distance between these points and P_0 (via taking the inner product). We then see that for the smaller sphere the distance of the first point is less than that of the second point, whereas for the larger sphere, the distance of the first point is larger than that of the second point—which will therefore lead us to label the larger sphere as being 'behind' the point P_0. This allows us to perform bounces only with spheres that are in front of the ray origin point P_0.

4.1.3. Circles/Discs. Circles are 3-blades and so the meet with a ray gives a 1-vector, Y.

- If Y itself is zero the ray (or line) lies in the plane of the circle and either does not intersect the circle or intersects the circle in one or two points.
- If $Y^2 < 0$, the ray does not lie in the plane of the circle and passes through the circle disc but does not intersect.
- If $Y^2 > 0$ the line does not lie in the plane of the circle and passes outside the circle disc without intersecting.
- If $Y^2 = 0$ (and $Y \neq 0$) the ray intersects the circumference of the circle.

Figure 5 shows an example of each case along with a geometric interpretation of the form of the meet. If $Y \neq 0$ and there is an intersection, the plane containing the circle is formed by taking the wedge product between the circle and n_∞, $C \wedge n_\infty$, and the intersection point is then extracted from the ray and this plane.

If $Y = 0$, so that the ray lies in the plane of the circle, we need to work in 2D, so that our 'meet' will result from taking the 2-part of the geometric product and dualising (with respect to the plane of the circle) to give a bivector. If the bivector has negative square there are two intersections at points A and B, so the bivector is $A \wedge B$. If the bivector has positive square, there is no intersection. If the bivector squares to zero there is one intersection at A, and the bivector is $a \wedge n_0$, where $A = F(a)$. In both cases, the intersection points are easily extracted.

4.2. Extracting Normals and Reflecting Rays

Extracting the normal to the surface of an object at a ray intersection point, X, and the reflection of that ray at X, are two fundamental building blocks in our ray tracer.

For a plane Φ which intersects with a ray, we can compute the reflection L' of an incident ray L (we assume Φ and L have been normalised such that $\Phi^2 = -1$, $L^2 = 1$) with the plane by simple sandwiching: $L' = \Phi L \Phi$. The resulting line is oriented correctly, passes through the intersection point and $L'^2 = 1$. For the case of a sphere S, we use the following formula from [22]:

$$L' \propto -(X \cdot (SLS)) \wedge n_\infty, \tag{7}$$

where X is the first point of intersection. Here, SLS is an example of an *inversion*, where the incoming ray/line, L, is inverted in the sphere to give a *circle* which passes through the two points of intersection and the origin of the sphere (note we only have a meaningful reflection if there are two points of intersection). The tangent line to this circle at the first point of intersection, X, is the reflected ray, L'. Figure 6 illustrates this geometrical construction. Note that one can also form the tangent plane at X and reflect L in this plane; this is performed by the following formula:

$$L' \propto \Phi_X L \Phi_X, \quad \Phi_X = (X \cdot S) \wedge n_\infty. \tag{8}$$

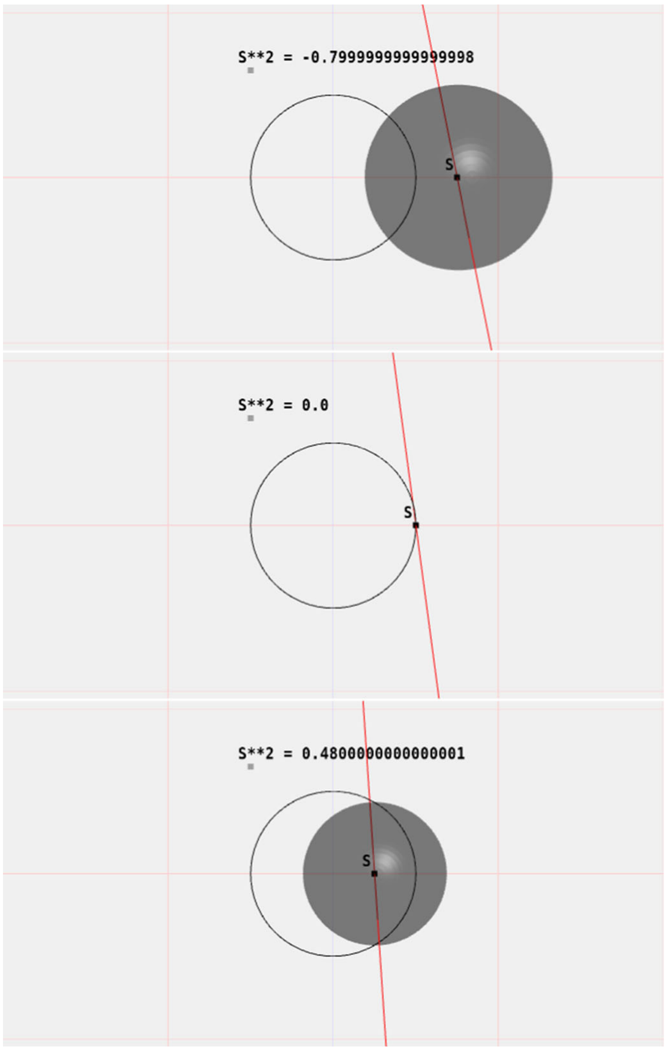


FIGURE 5. The 5D meet of a ray, L (in red) and a circle C (black) results in a vector Y whose dual is the sphere S (grey), the properties of this sphere vary with the relative positioning of C and L. In the top case the ray passes outside of the circle, S passes orthogonally through both the circle and the ray and squares to a negative number as we would expect from a standard CGA sphere. In the middle case the ray hits the perimeter of the circle and the meet squares to zero, in this case the dual sphere is the special case of zero radius, it represents the intersection point itself. In the bottom case the ray passes inside the circle. Here the sphere squares to a positive scalar implying it is now an imaginary sphere and in fact the circle passes through the sphere's antipodal points

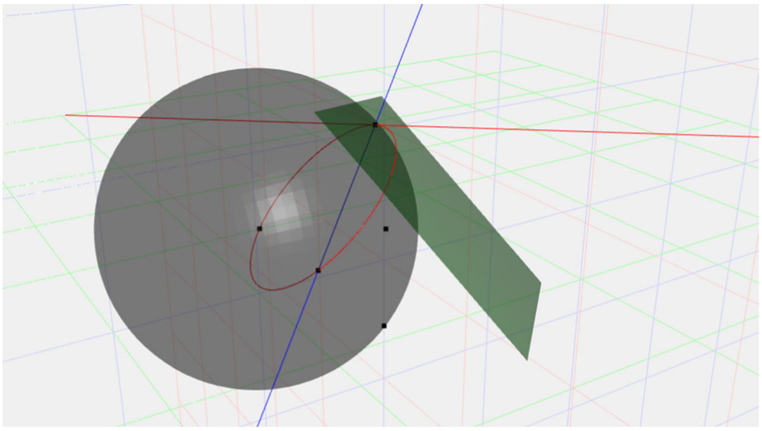


Figure 6. A ray in blue hits a sphere. The tangent plane at the point of impact is shown in green. The inversion of the ray in the sphere produces the red circle. The reflection of the ray in the tangent plane gives the red line. The reflected ray is also the tangent to the red circle at the point of impact

For a circle/disc C, we first form the plane $C \wedge n_\infty = \Phi$ in which it lies. If the ray intersects the disc (see Sect. 4.1), the reflected ray can then be found using the same formula as for the plane reflection case: $L' \propto (C \wedge n_\infty)\, L\, (C \wedge n_\infty)$. Note that these expressions specifically give the (correctly oriented) reflected ray that passes through the point of intersection of the incident ray and the object, rather than a parallel ray at the origin.

We end this section with two very useful constructions which we will put to use later in the paper. Firstly, consider the reflected ray, L', and the incident ray, L, both normalised such that they square to 1. The normal line to the surface, N, can be simply found:

$$N \propto (L' - L).$$

The tangent line, T, (in the plane containing incident and reflected rays) can similarly be found from the sum

$$L_T \propto L' + L.$$

Figure 7 shows a graphical example of these constructions for the reflection of a ray in a sphere.

5. Ray Tracing Evolved Circles

We will now turn to an interesting class of surface that arises from the direct interpolation of CGA circles [15], examples of which are shown in Fig. 8. In order to generate such a surface, a direct interpolation is first performed between two boundary circles, C_1 and C_2 both of which are normalised such

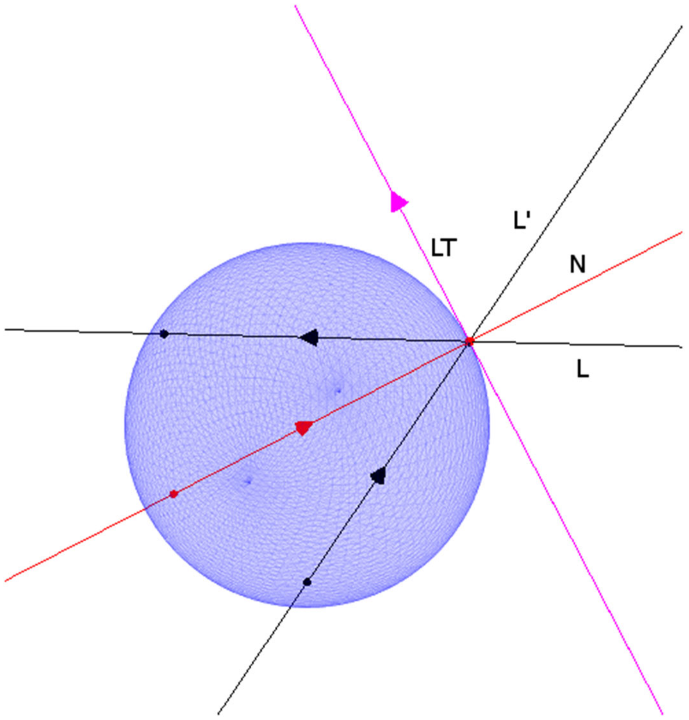


FIGURE 7. An incident ray L (black) hits a sphere from the right hand side of the figure. The reflected ray L' (also black) scatters to the top right corner of the figure. For L and L' normalised such that $L^2, L'^2 = 1$, the normal N (red) to the surface is proportional to $L' - L$. The tangent line L_T (pink) in the plane containing the incident and reflected rays can be found from $L_T \propto L' + L$

that $C_1^2 = C_2^2 = 1$. Our interpolation is of the form:

$$C_\alpha^{'} = \alpha C_1 + (1-\alpha)C_2, \tag{9}$$

where we take α moving between 0 and 1, which moves us from C_2 to C_1. The result of this interpolation is not itself a valid circle and needs to be 'projected' onto a blade via multiplication by a *projector*, which we shall call $\mathcal{S}$. This projector has *only* scalar and 4-vector parts and its construction is detailed in [15] and outlined in the following.

Consider a quantity $\Sigma = \langle\Sigma\rangle_0 + \langle\Sigma\rangle_4$. We then define the quantity $[[\Sigma]] = \sqrt{\langle\Sigma\rangle_0^2 - \langle\Sigma\rangle_4^2}$, and with this the principal square root [14] of the scalar + 4-vector, Σ, can be found as:

$$\sqrt{\Sigma} = \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} = \frac{\langle\Sigma\rangle_0 + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} + \frac{\langle\Sigma\rangle_4}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}. \tag{10}$$

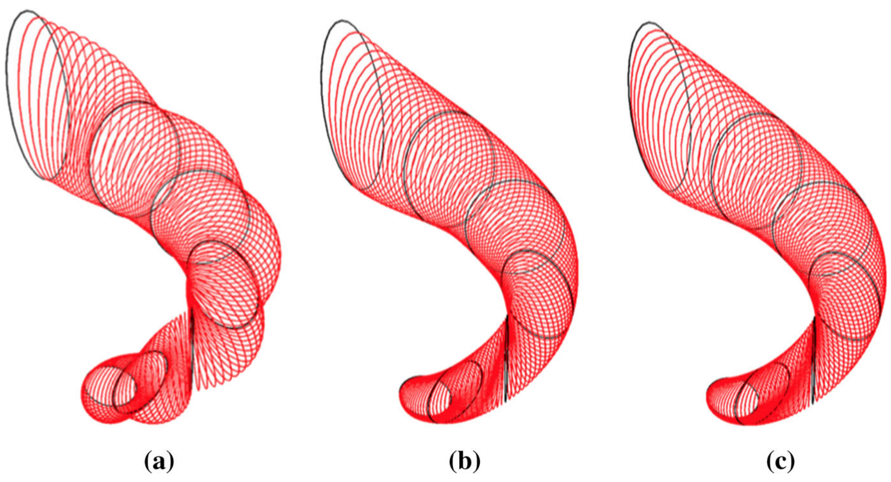


FIGURE 8. Polynomial interpolation through circular control objects. **a** Linear, **b** quadratic, **c** cubic

With this square root we can then form:

$$k\mathcal{S}^- = \sqrt{-C'_\alpha \tilde{C}'_\alpha}, \tag{11}$$

where $\mathcal{S}^-$ is $\mathcal{S}$ with the sign of the 4-vector part reversed and $\frac{1}{k} = \mathcal{S}^-\mathcal{S}$. We then construct $k\mathcal{S}$ by reversing the sign of the 4-vector part, ($k\mathcal{S} = \langle k\mathcal{S}^-\rangle_0 - \langle k\mathcal{S}^-\rangle_4$), and use this to produce the following expression for the projector $\mathcal{S}$ and interpolated circle C_α:

$$C_\alpha = \frac{k\mathcal{S}}{(k\mathcal{S})(k\mathcal{S}^-)} C^{'}_\alpha \equiv \mathcal{S} C^{'}_\alpha, \quad \alpha \in [0,1]. \tag{12}$$

Given that these surfaces may find genuine applications in computer graphics and CAD, it is desirable to explore their properties with respect to the ray tracing framework. Specifically, for a given ray and scene object, the geometric constructions of interest for lighting models are the point of intersection between a ray and a surface, and the surface normal at that specific intersection point.

In order to render this surface, we first show how to extract the intersection point with a given ray and then how to construct the surface normal at this point.

5.1. Intersection Point of Ray and Interpolated Surface

We saw earlier that the intersection of a ray with a circle produces the 1-vector Y. If $Y = 0$ the ray lies in the plane of the circle and if $Y \neq 0$ and $Y^2 = 0$ there is one intersection. Therefore (in the case where the meet is not zero) to find the intersection point between our interpolated surface and a ray L, we need to find a value of α for which:

$$(C_\alpha \vee L)^2 = 0 \implies \langle C_\alpha L\rangle_4^2 = 0.$$

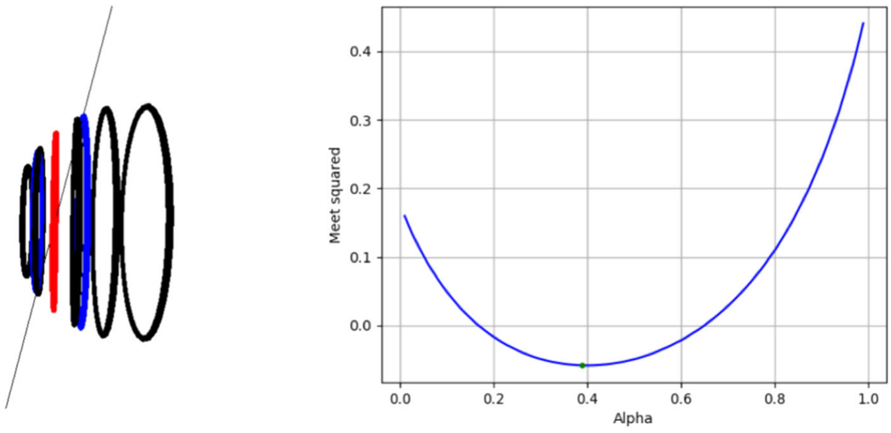


FIGURE 9. Left: an image showing an example interpolated surface and a ray passing through it, the circles in blue show the circles which have a meet squared of 0 with the incident ray, the red circle shows where the meet squared is minimised. Right: a plot showing the value of the meet squared as a function of α for this case

The system must also be tested for the case of $Y = 0$; if an α exists such that $(C_\alpha \vee L) = 0$, the ray may intersect C_α once, twice or not at all.

Figure 9 provides a simple visual illustration of one example of the shape of this curve as a function of α. While this example shown in the figure is particularly smooth, experiments indicate that in the general case this function is not well approximated by low order polynomials.

5.1.1. Non-linear Intersection Point Finder. As it is in general difficult to extract a closed form expression for the solution to $(C_\alpha \vee L)^2 = 0$, it is necessary to design an iterative algorithm to find the roots of the equation. Our implemented algorithm works as follows:

1. Check for intersection with a sphere enclosing the entire surface
2. Calculate the value of $(C_\alpha \vee L)^2$ at N intermediate values of α
3. Record where $(C_\alpha \vee L)^2$ changes sign between successive evaluated values of α
4. Locally approximate $(C_\alpha \vee L)^2$ as a quadratic equation in the region of the sign change and solve to get the value of α at the intersection point

Computing the intermediate objects in the surface can be done once per scene and reused for all rays calculated in that scene. To generate the enclosing sphere again we reuse the intermediary objects, in the following way:

1. Given C_1 and C_2, form intermediate circles, C_α, and then calculate the bounding sphere which is given by $S_\alpha = I_5 C_\alpha (C_\alpha \wedge n_\infty)$
2. Construct a sphere that contains all intermediate circle bounding spheres by successive application of a two sphere bounding algorithm

Again the enclosing sphere of the object can be calculated once per scene and used for all rays. Any two sphere bounding algorithm can be used, here we chose the algorithm from [18] which is summarised as follows:

1. Ensure both spheres S_1, S_2 are normalised according to Eq. (6)
2. Construct the line L joining the centres of both spheres $L = (S_1 I_5) \wedge (S_2 I_5) \wedge n_\infty$
3. Intersect the line with the first sphere to produce a point pair $L \vee S_1 = F_1 \propto A_1 \wedge B_1$ and extract A_1 using Eq. (5)
4. Intersect the line with the second sphere to produce a point pair $L \vee S_2 = F_2 \propto A_2 \wedge B_2$ and extract B_2 using Eq. (5)
5. Check if $B_2 \cdot (S_1 I_5) > 0$, if so S_1 encloses S_2 and so S_1 is the bounding sphere
6. Check if $A_1 \cdot (S_2 I_5) > 0$, if so S_2 encloses S_1 and so S_2 is the bounding sphere
7. If neither original sphere is enclosed by the other, the new bounding sphere is given by $\frac{1}{2}(A_1 + B_2) I_5$

This iterative algorithm for the most part performs perfectly satisfactorily. When compared against specially constructed test cases for which the intersection points are known it produces negligible error. The main downside to this solution is that it is not mathematically guaranteed to give correct results especially in the case of small numbers of intermediary objects. In practice we can pre-compute large numbers of intermediary objects before rendering, allowing us to get good approximations to the function of interest. Having said that, the more intermediate objects that are created, the more computationally expensive the process is, as the root finder has to evaluate our function at each one for each ray.

5.1.2. 'Closed Form' Solution for the Intersection of a Ray and an Evolved Circle Surface. In this section we will use C'_α to be the interpolated circle; however we emphasise that the process outlined here will also hold for the intersection of rays with other evolved objects. The intersection of the ray, L, and the surface, $\mathcal{S}C'_\alpha$ (see Eq. (12)) occurs when:

$$(L \vee [\mathcal{S}C'_\alpha])^2 = 0.$$

Writing $\Sigma = -C'_\alpha \tilde{C}'_\alpha$, this can be rewritten as:

$$\left(\frac{L \vee \left[\langle \sqrt{\Sigma} \rangle_0 C'_\alpha - \langle \sqrt{\Sigma} \rangle_4 C'_\alpha \right]}{\left(\langle \sqrt{\Sigma} \rangle_0 - \langle \sqrt{\Sigma} \rangle_4 \right) \sqrt{\Sigma}} \right)^2 = 0. \tag{13}$$

The denominator of this expression is never infinite (other than in the uninteresting case of $\Sigma = 0$) and so does not contribute roots. Thus we can write:

$$\left(L \vee [\langle \sqrt{\Sigma} \rangle_0 C'_\alpha] - L \vee [\langle \sqrt{\Sigma} \rangle_4 C'_\alpha] \right)^2 = 0.$$

Now expanding $\sqrt{\Sigma}$ as:

$$\sqrt{\Sigma} = \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle \Sigma \rangle_0 + [[\Sigma]]}}$$

means we can write:

$$\left(L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} \right\rangle_0 C'_\alpha\right] - L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} \right\rangle_4 C'_\alpha\right]\right)^2 = 0.$$

Again the denominator of the square root function is simply a scalar which is never infinite, thus it contributes no roots and we can write:

$$\left(L \vee [\langle \Sigma + [[\Sigma]]\rangle_0 C'_\alpha] - L \vee [\langle \Sigma + [[\Sigma]]\rangle_4 C'_\alpha]\right)^2 = 0.$$

The quantity $[[\Sigma]]$ is a scalar and so distributing the grade selection operators gives us:

$$\left(L \vee [\langle \Sigma\rangle_0 C'_\alpha] + [[\Sigma]] L \vee C'_\alpha - L \vee [\langle \Sigma\rangle_4 C'_\alpha]\right)^2 = 0.$$

Expanding this leads to:

$$\begin{aligned} 0 &= [L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)]^2 + [[\Sigma]]^2 (L \vee C'_\alpha)^2 \\ &\quad + [[\Sigma]] \{(L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha))(L \vee C'_\alpha) \\ &\quad + (L \vee C'_\alpha)(L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha))\}. \end{aligned} \tag{14}$$

To make progress on solving this we recall that $\Sigma = -C'_\alpha \tilde{C}'_\alpha$ and that for our linear interpolation of circles we have defined C'_α as:

$$C'_\alpha = \alpha C_1 + (1-\alpha)C_2 = \alpha(C_1 - C_2) + C_2.$$

As C'_α has terms in α of order 1 we would expect Σ to have terms of order 2. Continuing on this train of thought one might suspect that it is possible to re-write Eq. (14) as a simple polynomial in α. However, it is easy to see that this is not possible due to $[[\Sigma]]$, which is a scalar polynomial in α enclosed entirely in a square root:

$$[[\Sigma]] = \sqrt{\langle\Sigma\rangle_0^2 - \langle\Sigma\rangle_4^2}.$$

Thus in order to solve Eq. (14) we need to rearrange:

$$\begin{aligned} &[L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)]^2 + [[\Sigma]]^2 (L \vee C'_\alpha)^2 \\ &\quad = -[[\Sigma]] \{L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)\} (L \vee C'_\alpha) \\ &\quad + (L \vee C'_\alpha) \{L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)\}. \end{aligned} \tag{15}$$

We can then square both sides of the equation, eliminating the square root in the process:

$$\begin{aligned} &\left([L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)]^2 + [[\Sigma]]^2 (L \vee C'_\alpha)^2\right)^2 \\ &\quad = [[\Sigma]]^2 \left[\{L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)\} (L \vee C'_\alpha)\right. \\ &\quad \left. + (L \vee C'_\alpha) \{L \vee (\langle\Sigma\rangle_0 C'_\alpha) - L \vee (\langle\Sigma\rangle_4 C'_\alpha)\}\right]^2. \end{aligned} \tag{16}$$

Expanding this out will give a polynomial in α—it turns out that this is a scalar polynomial due to the fact that $\langle\Sigma\rangle_4 C'_\alpha$ has only trivector components. This polynomial can then be solved with any numerical polynomial solver such as finding the eigenvalues of the companion matrix [20].

For this case of linear evolution of circles we will get a polynomial of order 12, implying 12 potential roots. In reality 6 of these roots are extraneous, generated by the process of squaring to handle the square root term in

$[[\Sigma]]$. Some of the 6 remaining roots may be imaginary, some may be outside of the range $0 \leq \alpha \leq 1$ and some will be spurious roots corresponding to $\mathcal{S} = 0$. To filter out the valid roots we simply take all roots between 0 and 1 and evaluate $(L \vee [\mathcal{S}C'_\alpha])^2$ at these positions, selecting the roots for which $|(L \vee [\mathcal{S}C'_\alpha])^2| < \epsilon$ for some small ϵ threshold where $\epsilon > 0$ (in our experiments $\epsilon = 10^{-6}$ works satisfactorily).

An interesting point to note here is that we could extend this intersection finding method to C' being higher order functions of α, so long as $(L \vee [\mathcal{S}C'_\alpha])^2 = 0$. Generating such higher order splines through geometric primitives is described in Sect. 8.

5.1.3. A Comment on Rendering Speed. For this paper our raytracer was implemented in Python with the Clifford Library [1]. It is simply an investigative tool used as a framework in which to conduct basic research into the shapes and properties of surfaces as well as the algorithms used to render them. Of course in a production computer graphics environment, a higher performance language such as C/C++/GLSL would be required and the trade off of accuracy for speed with regard to the number of intermediary objects would need to be closely analysed. Such an analysis would require very careful benchmarking and comparison across multiple modern computer architectures and as such is beyond the scope of this paper.

5.2. Analytic Form for Normals

Given the α for which the ray intersects the surface, we have both the interpolated circle, C_α, and the point of intersection X. Using the result from [22], which is also used in Eq. (7), we extract a tangential line L_C in the plane of the circle at X:

$$L_C = (X \cdot C_\alpha) \wedge n_\infty. \tag{17}$$

We would now like an analytic form for the tangent to the surface corresponding to evolving the surface through an increment of α, postulating that this will be orthogonal to L_C: some future work remains to understand how these two tangent vectors are related to the directions of principal curvature. Clearly $\frac{dC(\alpha)}{d\alpha} \equiv \dot{C}_\alpha$ will be a key quantity in deriving this additional tangent vector. A first observation is that the circle and its derivative will be orthogonal to one another, i.e. $\dot{C} \cdot C = 0$, and that the geometric product is minus itself under reversion, i.e. $\dot{C}\,C = -C\,\dot{C}$ (note that here, and in what follows, we will drop the α subscript on C). This follows from the fact that $C^2 = C \cdot C = 1$ (our circles are all normalised), so that:

$$\frac{d}{d\alpha}(C \cdot C) = C \cdot \dot{C} + \dot{C} \cdot C = 0, \quad \frac{d}{d\alpha}(C\,C) = C\,\dot{C} + \dot{C}\,C = 0. \tag{18}$$

Since $C \cdot \dot{C} = -C \cdot \dot{C}$ and they are both scalars, this tells us $\dot{C} \cdot C = 0$. Using the fact that $\tilde{C} = -C$, we see that $C\,\dot{C} = -C\,\dot{C} = -(C\,\dot{C})\tilde{}$. As there are no 6-vector parts, this indicates that the product can only have bivector parts (this is a standard construct in many areas, the most obvious being rigid body dynamics [12]). Let us call this bivector, Ω_C:

$$\Omega_C = C\dot{C}. \tag{19}$$

Using the analogy with rigid body dynamics, we think of this bivector as the *angular velocity bivector* of the circles as they evolve under the parameter α. We note here that a similar construction would be possible for the other main objects that we use in CGA, since they are all normalised to 1 or 0. The null vectors representing points, X, have a constant 'length' due to normalisation, so as with the circles, we can differentiate wrt α to see that X and $\dot{X}$ are orthogonal, i.e. $X \cdot \dot{X} = 0$. If we were to define the 'velocity', $\dot{X}$ to be the inner product with the angular velocity bivector given in Eq. (19):

$$\dot{X} = X \cdot \Omega_C = X \cdot (C\dot{C}), \tag{20}$$

the condition $X \cdot \dot{X} = 0$ is satisfied since $X \cdot (X \cdot B) = (X \wedge X) \cdot B = 0$. Thus, given an X on the surface, lying on a circle with parameter α, the $\dot{X}$ defined above will preserve its length and is, we claim, the tangential direction required. In order to show this, the first thing we must do is establish that if we evolve X according to this rule, generating a quantity we call $X(\alpha)$, then $X(\alpha)$ should lie on C_α, for all α, i.e.,

$$X(\alpha) \wedge C_\alpha = 0.$$

Differentiating this (and again dropping the subscript α for clarity) and using $\dot{X} = X \cdot (C\dot{C})$, gives

$$\begin{aligned} &\dot{X} \wedge C + X \wedge \dot{C} = 0 \\ &\quad \implies \dot{X} \wedge C \equiv \Big(X \cdot (C\dot{C})\Big) \wedge C = -X \wedge \dot{C}. \end{aligned}$$

We now expand this expression using standard expansion results $(a \cdot (A_r \wedge B_s) = (a \cdot A_r) \wedge B_s + (-1)^r A_r \wedge (a \cdot B_s))$:

$$\begin{aligned} \Big(X \cdot (C\dot{C})\Big) \wedge C &= X \cdot \Big((C\dot{C}) \wedge C\Big) - (C\dot{C}) \wedge (X \cdot C) \\ &= -\frac{1}{2}\langle C\dot{C}(XC + CX)\rangle_4 \\ &= \frac{1}{2}\langle \dot{C}C(XC + CX)\rangle_4. \end{aligned} \tag{21}$$

The first term on the right hand side of the first line of this expansion is zero as $(C\dot{C}) \wedge C = \langle C\dot{C}C\rangle_5 = \langle -C^2\dot{C}\rangle_5 = \langle -\dot{C}\rangle_5 = 0$. Since X lies on C and so $X \wedge C = 0$, we see that $XC = CX$ which means that

$$\frac{1}{2}\langle \dot{C}C(XC + CX)\rangle_4 = \langle \dot{C}C^2X\rangle_4 = \langle \dot{C}X\rangle_4 = \dot{C} \wedge X = -X \wedge \dot{C}$$

giving $\dot{X} \wedge C = -X \wedge \dot{C}$ as required, so the proposed evolution is compatible with the constraint.

If we therefore assume that $\dot{X}$ is the direction we want, we can calculate the tangent line in this direction via:

$$L_T = \dot{X} \wedge X \wedge n_\infty. \tag{22}$$

The fact that lines L_C and L_T are perpendicular can be verified by showing that the quantity $L_T L_C$ has only a bivector part (see [22] for a discussion of when intersecting lines are orthogonal—if two lines L_1 and L_2

intersect at a point, then $\langle L_1 L_2\rangle_4 = 0$. In addition, if they are orthogonal, $\langle L_1 L_2\rangle_0 = 0$). If this is the case, $L_T L_C$ will reverse to minus itself.

To show this, we need to consider the reverse of $(\dot{X}\wedge X\wedge n_\infty)((X\cdot C)\wedge n_\infty)$. We will need the facts that that $XC = CX$, $\dot{X}C = -C\dot{X}$, $C\dot{C} = -\dot{C}C$, $X\dot{X} = -\dot{X}X$ and $\tilde{C} = -C$. We have shown all of these identities earlier in this section. We also need an additional fact, which is that $\dot{X}$ anticommutes with C. To see this we use another standard result $(a\wedge(A_r\cdot B_s) = (a\cdot A_r)\cdot B_s + (-1)^r A_r\cdot(a\wedge B_s))$:

$$\dot{X}\cdot C = \Big(X\cdot(C\dot{C})\Big)\cdot C = X\wedge\Big((C\dot{C})\cdot C\Big) - (C\dot{C})\cdot(X\wedge C). \tag{23}$$

The first term on the RHS of this equation is zero as $(C\dot{C})\cdot C = \langle C\dot{C}C\rangle_1 = \langle -C^2\dot{C}\rangle_1 = 0$, and the second term on the RHS is also zero as X lies on C so $X\wedge C = 0$. Thus $\dot{X}\cdot C = 0$ and $\dot{X}$ therefore anticommutes with C as required.

We are now in a position to expand out $(\dot{X}\wedge X\wedge n_\infty)((X\cdot C)\wedge n_\infty)$:

$$\begin{aligned}
&(\dot{X}\wedge X\wedge n_\infty)((X\cdot C)\wedge n_\infty)\\
&\quad = \Big((\dot{X}X)\wedge n_\infty\Big)((XC)\wedge n_\infty)\\
&\quad = \frac{1}{4}\left[\Big(\dot{X}Xn_\infty + n_\infty\dot{X}C\Big)(XCn_\infty + n_\infty XC)\right]\\
&\quad = \frac{1}{4}\left[\dot{X}Xn_\infty XCn_\infty + n_\infty\dot{X}Xn_\infty XC\right].
\end{aligned} \tag{24}$$

In the above we have used the facts that $\dot{X}\wedge X = \dot{X}X$, $XC = X\cdot C$ and $X^2 = 0$. Note that the term $Xn_\infty X$ in the final line of Eq. (24) can be written as $2(X\cdot n_\infty)X$ (from the standard reflection formula and the fact that $C^2 = 0$). Reversing the final line of Eq. (24) and using the commutation and anticommutation relations discussed, it is easy to show that the reverse of $L_T L_C$ is indeed minus itself, implying it has only a bivector part, as required, meaning the lines are orthogonal. Note that this result relies crucially on the fact that $\dot{X}$ and C anticommute, which is a good indication that $\dot{X}$ lies in the right direction.

Given these two orthogonal tangent lines L_C and L_T, we can construct the plane tangent to the surface at X by computing the join of the two lines. Or, we can bypass the plane entirely and compute the surface normal line directly as:

$$N = \langle L_T L_C\rangle_2 I_5. \tag{25}$$

6. Calculating the Derivative of the Object Manifold Projection

To calculate $\dot{C}$ we must differentiate the projection onto the blade manifold of our interpolated object with respect to our evolution parameter α. We will continue to work with circles but note that the process works with the general case where C'_α is any pure-grade multivector which is a function of a scalar parameter α. Let the projection of C'_α onto the blade manifold be given by:

$$C_\alpha = \mathcal{S}C'_\alpha$$

where $\mathcal{S}$ is our blade projector. The differential of this with respect to α is given by:

$$\frac{\partial C_\alpha}{\partial \alpha} = \frac{\partial \mathcal{S}}{\partial \alpha}C'_\alpha + \mathcal{S}\frac{\partial C'_\alpha}{\partial \alpha}. \tag{26}$$

Thus, any closed form expression for the derivative on the manifold will first require a closed form for the derivative of the projector $\frac{\partial \mathcal{S}}{\partial \alpha}$. Recall from Eq. (13) that we can write the projector $\mathcal{S}$ as a function of $\sqrt{\Sigma}$, where $\Sigma = -C'_\alpha \tilde{C}_\alpha$, and so

$$C_\alpha = \frac{\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4}{\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}}C'_\alpha \equiv SC'_\alpha. \tag{27}$$

Thus to find an expression for $\frac{\partial \mathcal{S}}{\partial \alpha}$ we will first need one for $\frac{\partial\sqrt{\Sigma}}{\partial \alpha}$.

6.1. Closed Form Derivative of the Square Root Operation

The closed form for the derivative of the principal square root function can be found by repeated application of the chain and product rules:

$$\frac{\partial[[\Sigma]]}{\partial \alpha} = \frac{\langle\frac{\partial\Sigma}{\partial\alpha}\rangle_0\langle\Sigma\rangle_0 + \langle\Sigma\rangle_0\langle\frac{\partial\Sigma}{\partial\alpha}\rangle_0 - \langle\frac{\partial\Sigma}{\partial\alpha}\rangle_4\langle\Sigma\rangle_4 - \langle\Sigma\rangle_4\langle\frac{\partial\Sigma}{\partial\alpha}\rangle_4}{2[[\Sigma]]}, \tag{28}$$

where we are using the fact that $\left\langle\frac{\partial\Sigma}{\partial\alpha}\right\rangle_g = \frac{\partial\langle\Sigma\rangle_g}{\partial\alpha}$.

$$\begin{aligned}
\frac{\partial}{\partial\alpha}\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right) &= \frac{-1}{2\sqrt{2}}(\langle\Sigma\rangle_0 + [[\Sigma]])^{-\frac{3}{2}}\left(\langle\frac{\partial\Sigma}{\partial\alpha}\rangle_0 + \frac{\partial[[\Sigma]]}{\partial\alpha}\right),\\
\frac{\partial\sqrt{\Sigma}}{\partial\alpha} &= \left(\frac{\partial\Sigma}{\partial\alpha} + \frac{\partial[[\Sigma]]}{\partial\alpha}\right)\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right)\\
&\quad + (\Sigma + [[\Sigma]])\frac{\partial}{\partial\alpha}\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right).
\end{aligned} \tag{29}$$

Thus the derivative of $\sqrt{\Sigma}$ is a function only of Σ and $\frac{\partial\Sigma}{\partial\alpha}$ which in turn can be written in terms of C'_α and $\frac{\partial C'_\alpha}{\partial\alpha}$:

$$\Sigma = -C'_\alpha\tilde{C}'_\alpha, \qquad \frac{\partial\Sigma}{\partial\alpha} = -\frac{\partial C'_\alpha}{\partial\alpha}\tilde{C}'_\alpha - C'_\alpha\frac{\partial\tilde{C}'_\alpha}{\partial\alpha}.$$

6.2. Closed Form Derivative of the Projector

With our square root derivative in place we can proceed to finding the derivative of the projector $\mathcal{S}$. Recall, $\mathcal{S}$ is given by:

$$\mathcal{S} = \frac{\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4}{\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}}.$$

We can again differentiate this with repeated applications of the chain and product rule:

$$\frac{\partial\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)}{\partial\alpha} = \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4,$$

$$\frac{\partial\left(\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1}\right)}{\partial\alpha}$$
$$= \left[\left(\left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4\right)\sqrt{\Sigma} + \left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right]$$
$$* \left[-\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-2}$$

and so finally we have our closed form expression for the projector derivative:

$$\frac{\partial\mathcal{S}}{\partial\alpha} = \left(\left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4\right)\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1}$$
$$+ \left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\frac{\partial\left(\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1}\right)}{\partial\alpha}. \qquad (30)$$

Now consider C'_α to be an interpolated circle of the form $C'_\alpha = C'_\alpha = \alpha C_1 + (1-\alpha)C_2$. The derivative of this with respect to α is a constant:

$$\frac{\partial C'_\alpha}{\partial\alpha} = C_1 - C_2.$$

This derivative is the final piece required for Eq. (26), giving us a completely closed form for $\frac{\partial C_\alpha}{\partial\alpha} \equiv \dot{C}$.

An important point to note here is that this blade projection derivative is grade-agnostic and so can be used for objects other than just evolved circles.

7. Ray Tracing Evolved Point Pairs

We will return to the actual ray tracing of circles later (see Figs. 16, 17), but first we turn our attention to point pairs. Due to the mathematical similarities between circles and point-pairs in CGA [13], as well as the practical desire to represent ribbon-like surfaces, we can apply similar ray-tracing methods to surfaces formed from the interpolation of point-pair bivectors representing line segments. If P_1 and P_2 are point-pairs which represent a line segment, we form a surface via:

$$P_\alpha = \mathcal{S}P'_\alpha = \mathcal{S}(\alpha P_1 + (1-\alpha)P_2),$$

where again, $\mathcal{S}$ is a scalar plus 4-vector which maps the interpolated bivector onto a 2-blade. Figure 10 gives examples of such surfaces.

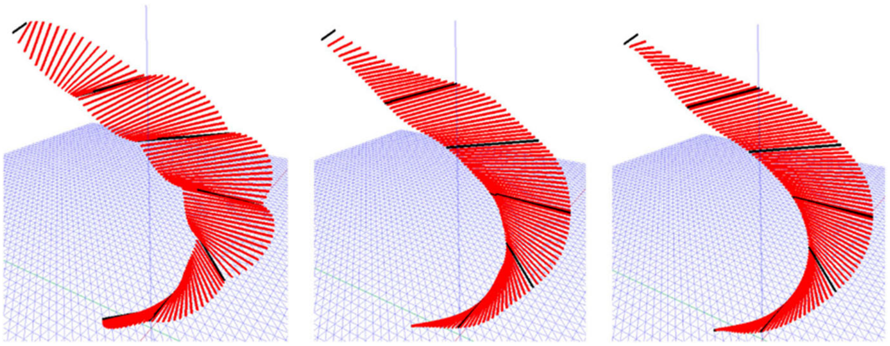


FIGURE 10. Polynomial interpolation through point-pair control objects. From left to right: linear, quadratic, cubic

7.1. Closed Form Solution for the Intersection of a Ray and an Evolved Point-Pair Surface

To find the intersection point of a ray and these surfaces we again form the meet of a ray L and the form of the evolved point-pair P_α. The result is a scalar quantity that can be written as:

$$L \vee P_\alpha \equiv (L^* \wedge P_\alpha^*) I_5.$$

In the case that the ray and the line that passes through both of the points in the point-pair in the surface (also known as the carrier line) intersect, this will give an answer of zero:

$$L \vee P_\alpha = 0.$$

As with the evolved circle surfaces we will attempt to construct this as a simple polynomial in α. We start with:

$$L \vee [\mathcal{S} P'_\alpha] = 0.$$

Expressing $\mathcal{S}$ in terms of Σ where as before, $\Sigma = -P'_\alpha \tilde{P}'_\alpha$, gives

$$L \vee \left[\frac{\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4}{\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}} P'_\alpha \right] = 0.$$

As before, the denominator cannot usefully be zero, giving:

$$\begin{gathered}
L \vee [\langle\sqrt{\Sigma}\rangle_0 P'_\alpha] - L \vee [\langle\sqrt{\Sigma}\rangle_4 P'_\alpha] = 0, \\
L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} \right\rangle_0 P'_\alpha \right] - L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} \right\rangle_4 P'_\alpha \right] = 0, \\
\left(L \vee [\langle\Sigma\rangle_0 P'_\alpha] + [[\Sigma]] L \vee P'_\alpha - L \vee [\langle\Sigma\rangle_4 P'_\alpha]\right)^2 = 0.
\end{gathered}$$

As again, the denominator is not zero.We now take the term containing $[[\Sigma]]$ (a scalar), to the RHS;

$$L \vee [\langle\Sigma\rangle_0 P'_\alpha] - L \vee [\langle\Sigma\rangle_4 P'_\alpha] = -[[\Sigma]] L \vee P'_\alpha.$$

Squaring then gives us:

$$(L \vee [\langle\Sigma\rangle_0 P'_\alpha] - L \vee [\langle\Sigma\rangle_4 P'_\alpha])^2 = [[\Sigma]]^2 (L \vee P'_\alpha)^2.$$

allowing us to form a simple scalar polynomial in α (we can see that this produces a scalar equation since $\langle\Sigma\rangle_4 P'_\alpha$ has only bivector parts):

$$(L \vee [\langle\Sigma\rangle_0 P'_\alpha] - L \vee [\langle\Sigma\rangle_4 P'_\alpha])^2 - [[\Sigma]]^2 (L \vee P'_\alpha)^2 = 0, \tag{31}$$

which can again be solved with a fast numerical polynomial solver.

This intersection equation for linearly interpolated point pairs is of order 6, implying there are up to 6 potential hitting points. Again the same process can be used to filter the roots as was done for the roots of the circle intersection equation.

7.2. Bounding Sphere and Normal Calculation

We saw earlier that the meet will be zero if the ray hits anywhere along the carrier line of the point-pair $L_C = P \wedge n_\infty$. Assuming the carrier line and ray do meet, the point of intersection can be extracted via the method outlined in the last section of [15]. Given that the carrier line of P_α (for some α) and the ray intersect at a point X, we can then check if the intersection point is within the bounding sphere $S = P_1 \wedge P_2$ of the surface by ensuring:

$$S^* \cdot X = [(P_1 \wedge P_2) I_5] \cdot X > 0.$$

Since the endpoints of all interpolated point-pairs will lie on the surface of S (see [15]), the above condition ensures there is an intersection with the line segment and not just the carrier line of the point-pair. To find the normal to the point-pair surface we can simply use exactly the same argument, and in fact the same code, as we did before for the evolved circles case but this time extracting L_C as:

$$L_C = P \wedge n_\infty.$$

Figure 11 shows an example of rendering a surface composed of interpolated point-pairs.

7.3. Special Cases of Evolved Point-Pairs

A special case of the evolved point-pairs occurs when they are co-planar and form chords of a circle, Fig. 12 shows two examples of this. As proved in [15] this special case results in the 4-vector part of the projector becoming zero implying the interpolation requires no re-projection back to the object manifold, i.e.,

$$P_\alpha = \alpha P_1 + (1-\alpha) P_2.$$

In this case the intersection of the carrier line can be found by looking for a point at which:

$$P_\alpha \vee L = \alpha P_1 \vee L + (1-\alpha) P_2 \vee L = 0.$$

Re-arranging gives an expression for α:

$$\alpha = \frac{P_2 \vee L}{P_2 \vee L - P_1 \vee L}.$$

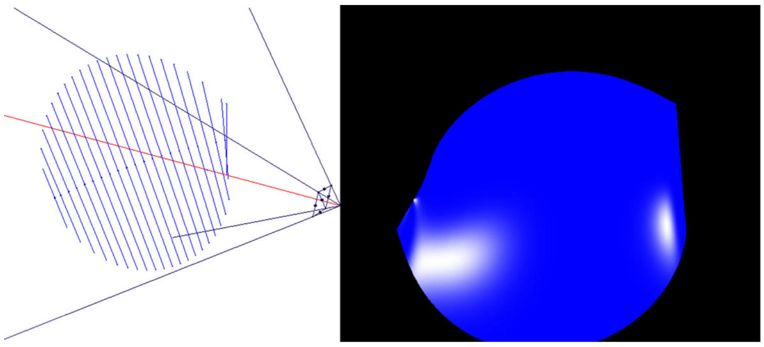


FIGURE 11. Ray tracing evolved point-pairs. Left: the scene to be rendered, in blue is a representation of the point-pair surface to be rendered, the camera frustum is shown in black, the camera axis is shown in red. Right: the resultant rendered surface of interpolated point-pairs

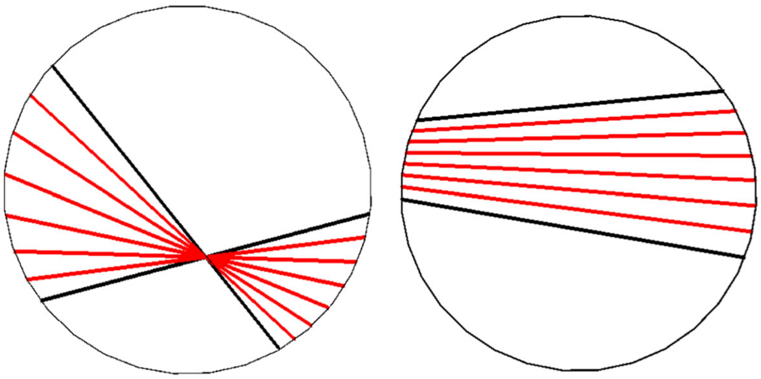


FIGURE 12. Two examples of point-pair interpolations for which all intermediary objects are blades without requiring projection to the object manifold

If the α derived from the above expression is between 0 and 1 then there is an intersection of the ray with the carrier line.

The disappearing 4-vector part of the projector, which is proportional to $P_1 \wedge P_2$, allows the ray-tracer to detect these cases reduces the computational expense of a ray-surface intersection considerably.

7.4. Triangular Facets from Evolved Point-Pairs

The intersection of co-circular point-pairs also allows us to examine the intersection of rays with triangular facets. Consider a ray L and a set of three points A, B, C which together form a triangular facet. First we will form a set of normalised point pairs:

$$P_1 = \frac{A \wedge C}{|A \wedge C|}, \quad P_2 = \frac{A \wedge B}{|A \wedge B|}, \quad P_3 = \frac{C \wedge B}{|C \wedge B|}.$$

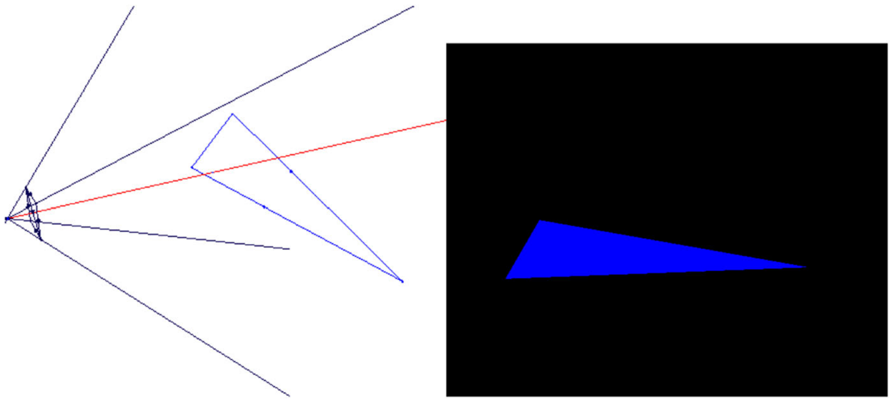


FIGURE 13. Ray tracing a triangular facet

We can then check if the ray intersects the facet by computing two scalar quantities

$$\alpha = \frac{P_2 \vee L}{P_2 \vee L - P_1 \vee L}, \qquad \beta = \frac{P_3 \vee L}{P_3 \vee L - P_2 \vee L}.$$

If both α and β are between 0 and 1 then the ray hits the facet. Figure 13 shows an example of rendering a triangular facet using this technique. It is of course possible to combine together multiple triangular facets and thus make meshes. Note that the line-facet intersection problem is not new in CGA, an alternative solution is already known via a reciprocal frame construction equivalent to barycentric coordinates and is well demonstrated in the raytracers of [13,22].

8. Bézier Curves and Hermite Splines through Geometric Primitives

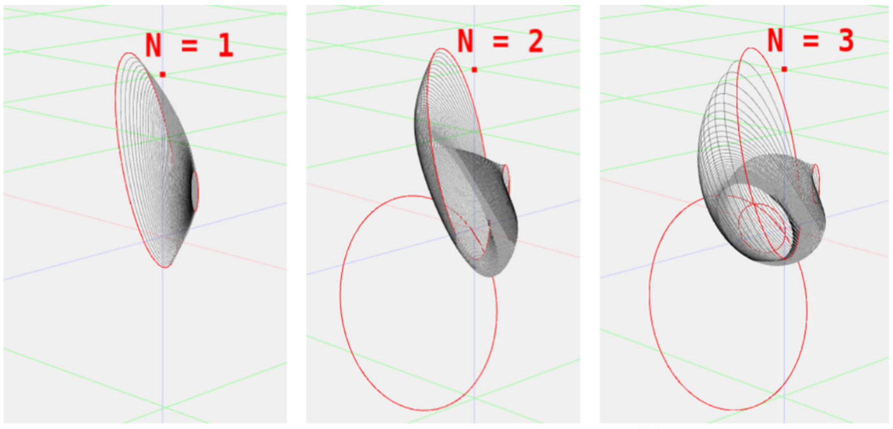
So far we have restricted our mathematics to linear interpolation of objects but have hinted that higher order interpolations are possible. A commonly used family of higher order interpolating curves are the Bézier curves [3], which in the cubic case and with specific first order endpoint conditions are known as Hermite curves.

8.1. Linear Interpolation as a Linear Bézier Curve

The simplest form of Bézier curve is simply a linear interpolation between two vectors. If we replace the vectors with k-blades and couple with the projection to the blade manifold we have the exact same linear interpolation, although this time with α going in the other direction. Adopting a notation of C_0 as the first object and C_1 as the second:

$$C'_\alpha = (1-\alpha)C_0 + \alpha C_1, \quad C_\alpha = \mathcal{S}C'_\alpha.$$

In Sects. 5.2 and 6, our analysis to extract surface normals was based on having an expression for the derivative of the pure grade multivector as a



(a) Order 1 **(b)** Order 2 **(c)** Order 3

FIGURE 14. Projected multivector Bézier curves of progressively higher order. The surfaces are shown in grey while the control objects are shown in red

function of α. For the case of the linear interpolation the solution is constant:

$$\frac{\partial C'_\alpha}{\partial \alpha} = C_1 - C_0.$$

8.2. Quadratic Bézier Curve

With three multivectors we can specify a quadratic function of α:

$$C'_\alpha = (1-\alpha)^2 C_0 + 2(1-\alpha)\alpha C_1 + \alpha^2 C_2.$$

This is known as a quadratic Bézier curve. Again we can take derivatives:

$$\frac{\partial C'_\alpha}{\partial \alpha} = 2(1-\alpha)(C_1 - C_0) + 2\alpha(C_2 - C_1).$$

8.3. Cubic Bézier Curves

With four control multivectors we get the the most commonly used form of Bézier curve, the cubic Bézier curve:

$$C'_\alpha = (1-\alpha)^3 C_0 + 3(1-\alpha)^2\alpha C_1 + 3(1-\alpha)\alpha^2 C_2 + \alpha^3 C_3.$$

Again we can take derivatives allowing us to extract surface normals:

$$\frac{\partial C'_\alpha}{\partial \alpha} = 3(1-\alpha)^2(C_1 - C_0) + 6(1-\alpha)\alpha(C_2 - C_1) + 3\alpha^2(C_3 - C_2).$$

Figure 14 shows examples of orders 1, 2 and 3 Bézier interpolation through circles, along with the control objects used to generate the surface.

8.4. *N*th Order Bézier Curve

More generally we can say that an Nth order multivector Bézier curve is of the form

$$C'_\alpha = \sum_{i=0}^{N} b_{i,N} C_i,$$

where

$$b_{i,N} = \begin{cases} \binom{N}{i}\alpha^i(1-\alpha)^{N-i}, & 0 \le i \le N, \\ 0, & \text{otherwise,} \end{cases}$$

are known as the Bernstein polynomials. The derivative of our Nth order Bézier curve is:

$$\frac{\partial C'_\alpha}{\partial \alpha} = \sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} C_i \quad \text{where} \quad \frac{\partial b_{i,N}}{\partial \alpha} = N(b_{i-1,N-1} - b_{i,N-1}).$$

If we re-arrange our coefficients the Bézier curve derivative can also be written in the form:

$$\frac{\partial C'_\alpha}{\partial \alpha} = N \sum_{i=0}^{N-1} b_{i,N-1}(C_{i+1} - C_i).$$

8.5. Rational Bézier Curves

A **rational** Bézier curve adds weights w_i to the polynomials allowing them to represent a broader class of curves:

$$C'_\alpha = \frac{\sum_{i=0}^{N} b_{i,N} C_i w_i}{\sum_{i=0}^{N} b_{i,N} w_i}.$$

Again, a closed form for their derivatives with respect to α can be calculated:

$$\begin{aligned} \frac{\partial C'_\alpha}{\partial \alpha} = \frac{1}{[\sum_{i=0}^{N} b_{i,N} w_i]^2} \Bigg(& \left[\sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} C_i w_i\right] \left[\sum_{i=0}^{N} b_{i,N} w_i\right] \\ & - \left[\sum_{i=0}^{N} b_{i,N} C_i w_i\right] \left[\sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} w_i\right] \Bigg). \end{aligned} \tag{32}$$

Thus we can additionally represent projected multivector rational Bézier curves and calculate analytic normals to the evolved surfaces formed.

8.6. Hermite Cubic Curves and Splines

Hermite cubic curves are another common form of interpolating curve. They are defined by control points, C_i, (where we use the notation C for points as we will see shortly that these can be replaced by objects) and associated tangent vectors, V_i, at each end of the curve, for $\alpha \in [0, 1]$:

$$C'_\alpha = (2\alpha^3 - 3\alpha^2 + 1)C_0 + (\alpha^3 - 2\alpha^2 + \alpha)V_0 + (-2\alpha^3 + 3\alpha^2)C_1 + (\alpha^3 - \alpha^2)V_1.$$

The derivative of the curve is:

$$\frac{\partial C'_\alpha}{\partial \alpha} = (6\alpha^2 - 6\alpha)C_0 + (3\alpha^2 - 4\alpha + 1)V_0 + (-6\alpha^2 + 6\alpha)C_1 + (3\alpha^2 - 2\alpha)V_1.$$

Cubic Hermite curves can be converted to cubic Bezier curves and vice-versa. As with Bezier curves, putting multivectors and multivector derivatives instead of the control points and tangents will give us a multivector valued curve.

A very common use of Hermite curves is in the construction of Hermite splines; these are piece-wise constructions in which multiple Hermite curves are placed end to end, sharing tangent vectors and control points at each endpoint. By constructing a curve in this way, a C1 continuous piece-wise curve is designed that passes through the control points exactly.

When moving the spline generation process to the multivector domain we must check whether the blade projection introduces problems with C1 continuity on the manifold. To check C1 continuity we need to evaluate the curve derivative either side of a junction between curves in the spline. Consider the form of the derivative:

$$\frac{\partial C_\alpha}{\partial \alpha} = \frac{\partial \mathcal{S}}{\partial \alpha} C'_\alpha + \mathcal{S} \frac{\partial C'_\alpha}{\partial \alpha}. \tag{33}$$

Let us now evaluate this at the endpoint of the nth curve in a piece-wise spline where $\alpha = 1$ and of curve $(n+1)$ where $\alpha = 0$. First we note that on both curves at these points, $\mathcal{S} = 1$ because the curve passes through a blade control object which requires no projection. Additionally we see that by definition of the Hermite spline at that point, the derivative in pure-grade space is shared across both curves as is the control point:

$$\frac{\partial C'_\alpha}{\partial \alpha} = V_{n,n+1}, \quad C'_\alpha = C_{n,n+1},$$

where $V_{n,n+1}$ and $C_{n,n+1}$ are the derivative (or 'tangent') and control objects respectively of the curve that are shared between segments n and $n+1$.

Thus for the derivative to evaluate to the same on either side of the boundary, we only need to check that $\frac{\partial \mathcal{S}}{\partial \alpha}$ is the same either side of the boundary. Considering the equations in Sect. 6.1 we can see that $\frac{\partial \mathcal{S}}{\partial \alpha}$ is a function only of C' and $\frac{\partial C'}{\partial \alpha}$ which are both constant across the junction. Thus the curve is C1 continuous on the manifold as required. With assurances that the spline is continuous across the boundaries we are free to chose any means of generating tangents in the pure-grade space that we like.

One such mechanism for generating tangents for Hermite splines comes from Kochanek and Bartels [21]. The Kochanek–Bartels (KB) spline is an interpolating spline with three scalar design parameters t, b, c known as tension, bias and continuity respectively. For given control objects C_i, C_{i+1} the corresponding tangents V_i, V_{i+1} can be calculated using the control objects in the spline C_{i-1} and C_{i+2} which lie previous to, and after, the curve in the order of the spline:

$$\begin{aligned} V_i &= \frac{(1-t)(1+b)(1+c)}{2}(C_i - C_{i-1}) + \frac{(1-t)(1-b)(1-c)}{2}(C_{i+1} - C_i), \\ V_{i+1} &= \frac{(1-t)(1+b)(1-c)}{2}(C_{i+1} - C_i) \\ &\quad + \frac{(1-t)(1-b)(1+c)}{2}(C_{i+2} - C_{i+1}). \end{aligned}$$

Setting all three scalar parameters to a value of 0 produces the commonly used Catmull–Rom spline [7]. Figure 15 shows an example of a KB spline of multivector geometric primitives.

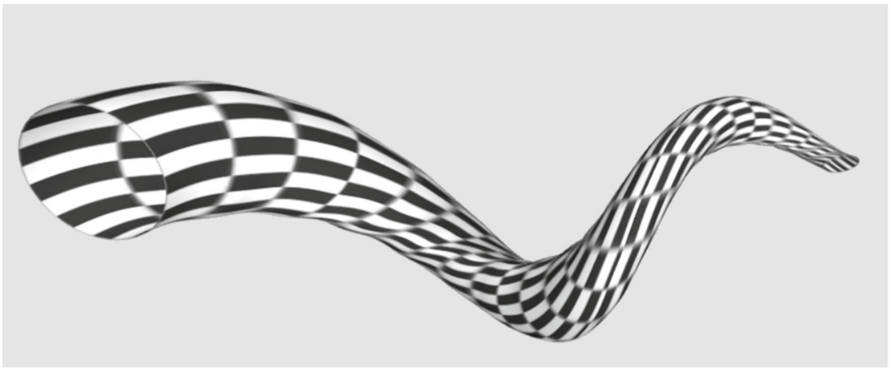


FIGURE 15. A Kochanek–Bartels spline of evolved circles meshed, textured, and rendered with smooth shading

9. Blinn–Phong Lighting Model

To complete our discussion of the ray-tracer we need to describe its lighting model. We employ the Blinn–Phong lighting model, this simple model is well studied in the graphics literature, for more in depth information see [4]. Under this model the intensity value for each pixel for colour channel λ is given by the following expression:

$$\begin{aligned} I_\lambda &= c_\lambda k_a + k_r I_{r\lambda} \\ &\quad + \sum_i S_i f_{\text{att}} I_{pi\lambda} \left(c_\lambda k_d \left(L_i \cdot N\right) + k_s \left(N \cdot H\right)^q\right), \quad \forall \lambda \in \{R, G, B\} \end{aligned} \tag{34}$$

Note that traditionally the terms N, L_i and H are standard 3D vectors. Since the language of our ray-tracer is CGA, these terms in Eq. (34) actually represent trivector lines which pass through the ray-object intersection point. As all these lines are normalised such that $L^2 = 1$ the inner products between them will simply give the cosine of the angles between them as in the traditional 3D vector case [12]. Other terms will be described in the following subsections.

The *normal line* is required across multiple terms and is crucial to the lighting model. This is given in CGA as $N = \frac{L' - L}{\sqrt{(L'-L)^2}}$ where L' is the normalised reflected ray and L is the normalised incident ray. Note that L' is necessarily calculated for tracing reflected rays. The line L_i specifying the direction to the ith point light source is given by forming the wedge product between the point of intersection of the incident ray and the object, the point position of the ith light source and n_∞ and again normalising the result such $L_i^2 = 1$. The normalised half way line H can be found by $H = \frac{L_i - V}{\sqrt{(L_i - V)^2}}$, where V is the normalised line from the intersection point to the viewer.

9.1. Diffuse Term

The diffuse term is given by:

$$I_d = I_{pi\lambda} c_\lambda k_d \left(L_i \cdot N\right), \quad \forall \lambda \in \{R, G, B\}. \tag{35}$$

The line L_i again represents the normalised line pointing towards specifying the the ith light source. The intensity of channel λ of the ith light source is labelled $I_{pi\lambda}$ where the p subscript denotes that it is a point light source. $I_{pi\lambda}$ along with the material channel reflection coefficient c_λ (which effectively controls the colour of the material) and k_d the diffuse property of an object material, are all scalar parameters of the model.

9.2. Specular Term

$$I_s = I_{pi\lambda} k_s \left(N \cdot H\right)^q , \quad \forall \lambda \in \{R, G, B\}. \tag{36}$$

This term requires the computation of the CGA line that corresponds to the half-way vector, **H**, defined as the vector halfway between the vectors to the viewer, **V**, and the light source, so that $H \propto L_i - V$. The normalised half way line H can be found by $H = \frac{L_i - V}{\sqrt{(L_i - V)^2}}$, where V is the normalised line from the intersection point to the viewer. k_s and q are scalar parameters of the object material.

9.2.1. Depth Attenuation. Depth attenuation is given by an inverse quadratic: $f_{\text{att}} = 1/\left(a + bd + cd^2\right)$ where d is the distance from the origin of a ray to the intersection point and a, b, c are scalar parameters of the lighting model. For the examples presented here we use $a = 0.02$, $b = 0$ and $c = 0.002$.

9.2.2. Shadow Attenuation. In order to determine whether an object is in shadow, the ray cast from the point of intersection to the light source is used for intersection tests with objects in the scene. If this yields an intersection, the shadow attenuation constant, S_i, is used, otherwise it is omitted.

10. Examples of Ray Tracing Simple Objects and Evolved Surfaces

Putting together the material from previous sections we can now raytrace both simple objects and evolved surfaces. Figure 3 shows an example of simple objects, spheres, planes and disks being rendered. Figure 16 shows an example of an evolved surface being rendered on its own. The class of surfaces that are able to be generated with the interpolation of circles is large and Fig. 17 shows a more unusual surface being rendered in a scene with a sphere and a plane.

11. Meshing Evolved Surfaces

Most graphics pipelines in modern computers use triangular meshes with some form of interpolation of vertex normals for approximating the look of curved surfaces. In light of this it is clearly desirable to be able to convert from an explicitly parameterised evolved surface to a mesh approximation of that surface.

To produce a mesh approximation we first need to generate a set of points that are in some sense evenly spaced and lie on the surface itself. To

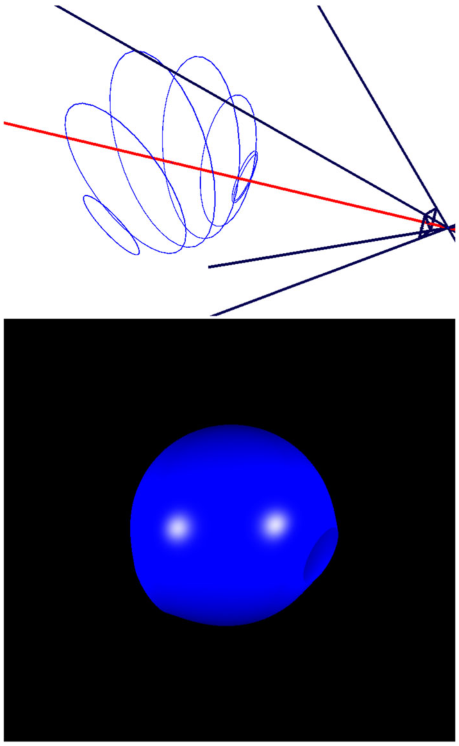


Figure 16. Left: a scene composed of only an evolved surface in blue and a camera. Right: the rendering of the scene from the camera

do this we will begin by producing a set of evenly spaced points on the first object C_1 and then transform these points along a small step in α to give a second set of points. Continuing in this way we can cover the surface entirely. An appropriate transformation for this task needs to preserve the relative spacing of the points on the objects in order to produce a good quality mesh. TRS (Translation Rotation Scaling) rotors have this property and can map circles to circles, spheres to spheres and point pairs to point pairs (these quantities are sometimes known as *rounds*). A TRS rotor that takes one object C_1, to another, C_2, can be calculated with the following process:

- Calculate T_1 and T_2 the translation rotors that bring C_1 and C_2 respectively to the origin
- Apply T_1 and T_2 to C_1 and C_2 respectively, bringing them to the origin and producing C_1' and C_2'
- Calculate the rotation rotor R_{12} between the blades $C_1' \wedge n_\infty$ and $C_2' \wedge n_\infty$ (if C_1 and C_2 are spheres then we do not need a rotation rotor so set $R_{12} = 1$)
- Calculate the difference in scale between the objects by extracting their relative sizes

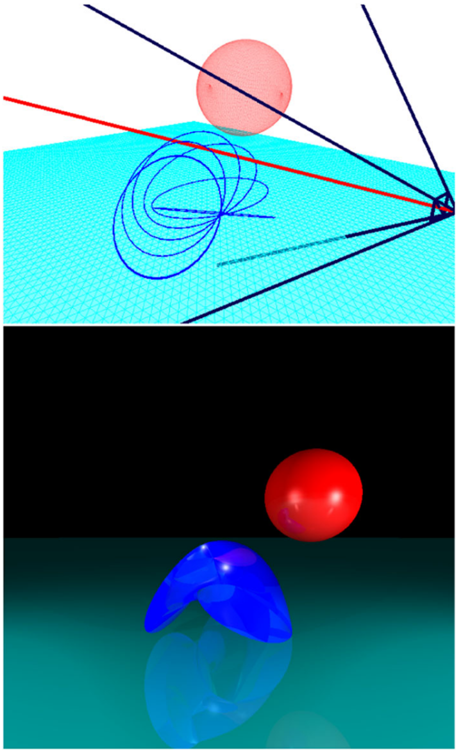


Figure 17. Left: a scene composed of a ground plane in cyan, an evolved surface in blue and a sphere in red. Right: the rendering of the scene from the camera

- Use the scale to generate a dilation rotor D_{12} that scales C'_1 to the same size as C'_2
- Compose the final TRS rotor Z_{12} that takes C_1 to C_2 as:

$$Z_{12} = \tilde{T}_2 D_{12} R_{12} T_1.$$

Armed with our transformation, we now simply need to generate a set of starting points on the first object. First consider the case of evolved circles. We can produce a set of N evenly spaced points on the unit circle in the e_1, e_2 plane by N successive rotations about the origin of a point X lying initially at $X_0 = F(e_1)$ yielding X_n for $n \in 0, \ldots, N$ i.e., for a fixed rotor R_θ:

$$X_n = (R_\theta)^n X_0 (\tilde{R}_\theta)^n$$

where θ is chosen so that $N + 1$ uniformly spaced points cover the whole circle. With the TRS rotor Z_{01} that maps from the unit circle at the origin to the first object C_1 it is possible to transform our points to the first object:

$$U_{n1} = Z_{01} X_n \tilde{Z}_{01}.$$

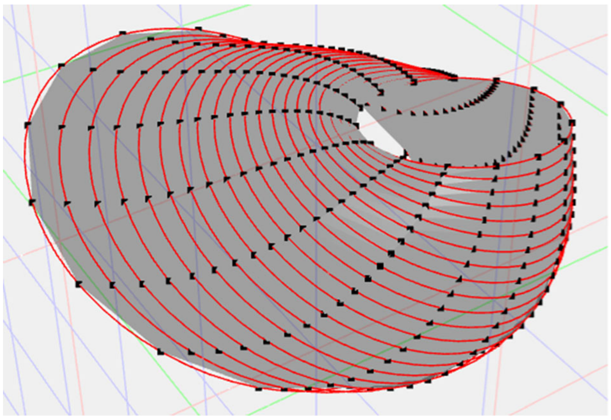


Figure 18. A linear interpolation surface of evolved circles meshed and rendered with flat shading

Our process then becomes one of stepping sequentially through α from 1 to 0 in small increments of $\delta\alpha$ and transforming our points along the way. We will use Z_α to refer to the TRS rotor that maps C_α to $C_{\alpha-\delta\alpha}$ and so can write the nth point at α as:

$$U_{n(\alpha-\delta\alpha)} = Z_\alpha U_{n\alpha} \tilde{Z}_\alpha.$$

By doing this we have effectively constructed a mapping from a coordinate system in the 2D plane of α and n to the surface manifold. This is useful as it lets us generate the mesh in the 2D plane of α and n and map the vertex positions directly to 3D.

Modern graphics engines allow users to write shaders that interpolate vertex normals in smart ways, giving the illusion of curved surfaces over flat facets. In our ray tracing experiments we have already identified how to calculate the normal to the evolved surface at any point on the surface provided that α is known at the point. The vertex normals are calculated using the formulae in the above sections. While Fig. 18 shows a surface of evolved circles meshed and rendered using flat shading with ganja.js [9], Fig. 15 shows a tubular KB spline surface, meshed, textured, and shaded with a smooth vertex normal interpolation scheme.

12. Summary and Conclusions

In this paper we have outlined the basic workings of a CGA ray tracer that can render geometric primitives as well as more advanced interpolated surfaces defined by two circles or point-pairs and an *evolution* parameter, α. Integral to ray-tracing these evolved surfaces is the derivation of analytic intersection points and normal vectors.

Acknowledgements

Hugo Hadfield acknowledges support from the EPSRC and ARM via ICASE award UKRI project number 1949701. Sushant Achawal acknowledges the Royal Academy of Engineering for funding the attendance of the CGI 2019 conference in Calgary via the Engineering Leaders Scholarship.

References

[1] Arsenovic, A., Hadfield, H., Wieser, E., Kern, R., The Pygae Team.: Version v1.3.0dev2. https://www.github.com/pygae/clifford. https://doi.org/10.5281/zenodo.3724677. Zenodo (March 23 2020)

[2] Bauer, U., Polthier, K.: Parametric reconstruction of bent tube surfaces. In: 2007 International Conference on Cyberworlds (CW-07), pp. 465–474. IEEE (2007)

[3] Bezier, P.: The Mathematical Basis of the UNISURF CAD System. Elsevier, Amsterdam (1986)

[4] Blinn, J.: Models of light reflection for computer synthesized pictures. ACM SIGGRAPH Comput. Graph. **11**(2), 192–198 (1977)

[5] Breuils, S., Nozick, V., Fuchs, L.: Garamon: a geometric algebra library generator. Adv. Appl. Clifford Algebras **29**, 69 (2019). https://doi.org/10.1007/s00006-019-0987-7

[6] Breuils, S., Nozick, V., Sugimoto, A., Hitzer, E.: Quadric conformal geometric algebra of $R^{9,6}$. Adv. Appl. Clifford Algebras **28**, 35 (2018). https://doi.org/10.1007/s00006-018-0851-1

[7] Catmull, E., Rom, R.: A class of local interpolating splines. In: Barnhill, R.E., Riesenfeld, R.F. (eds.) Computer Aided Geometric Design, pp. 317–326. Academic Press, London (1974)

[8] Colapinto, P.: Composing surfaces with conformal rotors. Adv. Appl. Clifford Algebras **27**, 453–474 (2017). https://doi.org/10.1007/s00006-016-0677-7

[9] De Keninck, S.: ganja.js https://github.com/enkimute/ganja.js (2017)

[10] De Keninck, S.: Non-parametric realtime rendering of subspace objects in arbitrary geometric algebras. In: Gavrilova, M., Chang, J., Thalmann, N.M.,

Hitzer, E., Ishikawa, H. (eds.) Advances in Computer Graphics, pp. 549–555. Springer International Publishing, Cham (2019)

[11] Deul, C., Burger, M., Hildenbrand, D., Koch, A.: Raytracing point clouds using geometric algebra. **9** (2009)

[12] Doran, C., Lasenby, A.: Geometric Algebra for Physicists, 1st edn. Cambridge University Press, Cambridge (2003)

[13] Dorst, L., Fontijne, D., Mann, S.: Geometric Algebra for Computer Science: An Object-oriented Approach to Geometry, 1st edn. Elsevier; Morgan Kaufmann, Amsterdam (2007)

[14] Dorst, L., Valkenburg, R.: Square root and logarithm of rotors in 3D conformal geometric algebra using polar decomposition. In: Dorst, L., Lasenby, J. (eds.) Guide to Geometric Algebra in Practice. Springer, London (2011). https://doi.org/10.1007/978-0-85729-811-9_5

[15] Hadfield, H., Lasenby, J.: Direct linear interpolation of geometric objects in conformal geometric algebra. Adv. Appl. Clifford Algebras **29**, 85 (2019). https://doi.org/10.1007/s00006-019-1003-y

[16] Hestenes, D.: Old wine in new bottles: a new algebraic framework for computational geometry. In: Corrochano, E.B., Sobczyk, G. (eds.) Geometric Algebra with Applications in Science and Engineering. Birkhäuser, Boston (2001)

[17] Hestenes, D., Sobczyk, G.: Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics, 2nd edn. Reidel, Dordrecht (1987)

[18] Hildenbrand, D., Hitzer, E.: Analysis of point clouds—using conformal geometric algebra. In: Proceedings of the Third International Conference on Computer Graphics Theory and Applications (2008)

[19] Hildenbrand, D.: Geometric computing in computer graphics using conformal geometric algebra. Comput. Graph. **29**(5), 802–810 (2005)

[20] Horn, R.A., Johnson, C.R.: Matrix Analysis, pp. 146–147. Cambridge University Press, Cambridge (1999)

[21] Kochanek, D.H.U., Bartels, R.H.: Interpolating splines with local tension, continuity, and bias control. SIGGRAPH Comput. Graph. **18**, 33–41 (1984)

[22] Lasenby, A., Lasenby, J., Wareham, R.: A Covariant Approach to Geometry Using Geometric Algebra, CUED Technical Report CUED/F-INFENG/TR-483. University Engineering Department, Cambridge (2004)

[23] Peternell, M., Pottmann, H.: Computing rational parametrizations of canal surfaces. J. Symb. Comput. **23**, 255–266 (1997)

[24] Petrovic, V., Fallon, J., Kuester, F.: Visualizing whole-brain DTI tractography with GPU-based tuboids and LoD management. IEEE Trans. Vis. Comput. Graph. **13**, 1488–1495 (2007)

[25] Wareham, R.J., Lasenby, J.: Generating fractals using geometric algebra. Adv. Appl. Clifford Algebras **21**, 647–659 (2011). https://doi.org/10.1007/s00006-010-0265-1

Hugo Hadfield, Sushant Achawal and Joan Lasenby
Department of Engineering
Cambridge University
Cambridge
UK
e-mail: hh409@cam.ac.uk

Sushant Achawal
e-mail: sushachawal@gmail.com

Joan Lasenby
e-mail: jl221@cam.ac.uk

Anthony Lasenby
Department of Physics
Cambridge University
Cambridge
UK
e-mail: a.n.lasenby@mrao.cam.ac.uk

Benjamin Young
Department of Applied Mathematics and Theoretical Physics
Cambridge University
Cambridge
UK
e-mail: bay22@cam.ac.uk

Received: October 31, 2019.
Accepted: January 6, 2021.