mathematics

MDPI

Article

Parametric Surfaces for Elliptic and Hyperbolic Geometries

László Szirmay-Kalos *, András Fridvalszky, László Szécsi and Márton Vaitkus

Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Műegyetem rkp. 3, 1111 Budapest, Hungary; fridvalszky@iit.bme.hu (A.F.); szecsi@iit.bme.hu (L.S.); vaitkus@iit.bme.hu (M.V.)

* Correspondence: szirmay@iit.bme.hu; Tel.: +36-20-468-4978

Abstract

Background/Objectives: In computer graphics, virtual worlds are constructed and visualized through algorithmic processes. These environments are typically populated with objects defined by mathematical models, traditionally based on Euclidean geometry. However, there is increasing interest in exploring non-Euclidean geometries, which require adaptations of the modeling techniques used in Euclidean spaces. **Methods**: This paper focuses on defining parametric curves and surfaces within elliptic and hyperbolic geometries. We explore free-form splines interpreted as hierarchical motions along geodesics. Translation, rotation, and ruling are managed through supplementary curves to generate surfaces. We also discuss how to compute normal vectors, which are essential for animation and lighting. The rendering approach we adopt aligns with physical principles, assuming that light follows geodesic paths. **Results**: We extend the Kochanek–Bartels spline to both elliptic and hyperbolic geometries using a sequence of geodesic-based interpolations. Simple recursive formulas are introduced for derivative calculations. With well-defined translation and rotation in these curved spaces, we demonstrate the creation of ruled, extruded, and rotational surfaces. These results are showcased through a virtual reality application designed to navigate and visualize non-Euclidean spaces.

Keywords: splines; parametric curves and surfaces; hyperbolic geometry; elliptic geometry; transformations

MSC: 51M09; 53A05; 65D17; 68U07

check for updates

Academic Editors: Attila Fazekas and Miklos Hoffmann

Received: 5 September 2025
Revised: 15 October 2025
Accepted: 23 October 2025
Published: 25 October 2025

Citation: Szirmay-Kalos, L.; Fridvalszky, A.; Szécsi, L.; Vaitkus, M. Parametric Surfaces for Elliptic and Hyperbolic Geometries. *Mathematics* **2025**, *13*, 3403. https://doi.org/10.3390/math13213403

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Modeling in computer graphics constructs virtual worlds by defining object surfaces through mathematical equations. Two common approaches are *implicit* and *parametric* formulations. Implicit equations of the form $f(\mathbf{r}) = 0$ describe the conditions a point $\mathbf{r}$ must satisfy to lie on a surface, often involving constraints such as the distance, angle, or orthogonality. Implicit forms are particularly useful in ray tracing: by substituting the ray's parametric equation into the surface equation, one obtains a scalar equation for the ray parameter.

Parametric modeling, in contrast, describes motion or shape directly. A parametric curve assigns a position $\mathbf{r}(t)$ to each parameter t, while a parametric surface $\mathbf{r}(u, v)$ extends this idea to two dimensions, generated by sweeping or interpolating curves. Many widely used graphics tools rely on such parametric constructions, with surfaces defined through extrusion, rotation, ruling, or interpolation of control points.

While Euclidean geometry underlies most existing modeling techniques, many applications benefit from extending them to non-Euclidean geometries. A common but limited strategy is to first design models in Euclidean space, tessellate the surfaces to small triangles, and then project vertices of the triangles into the curved space using central or stereographic projection, or with the exponential map [1]. However, because Euclidean and curved spaces differ in Gaussian curvature, any projection inevitably distorts either distances or angles, often breaking the modeler's spatial intuition. A more natural approach is to define models directly within the target non-Euclidean geometry (Figure 1).

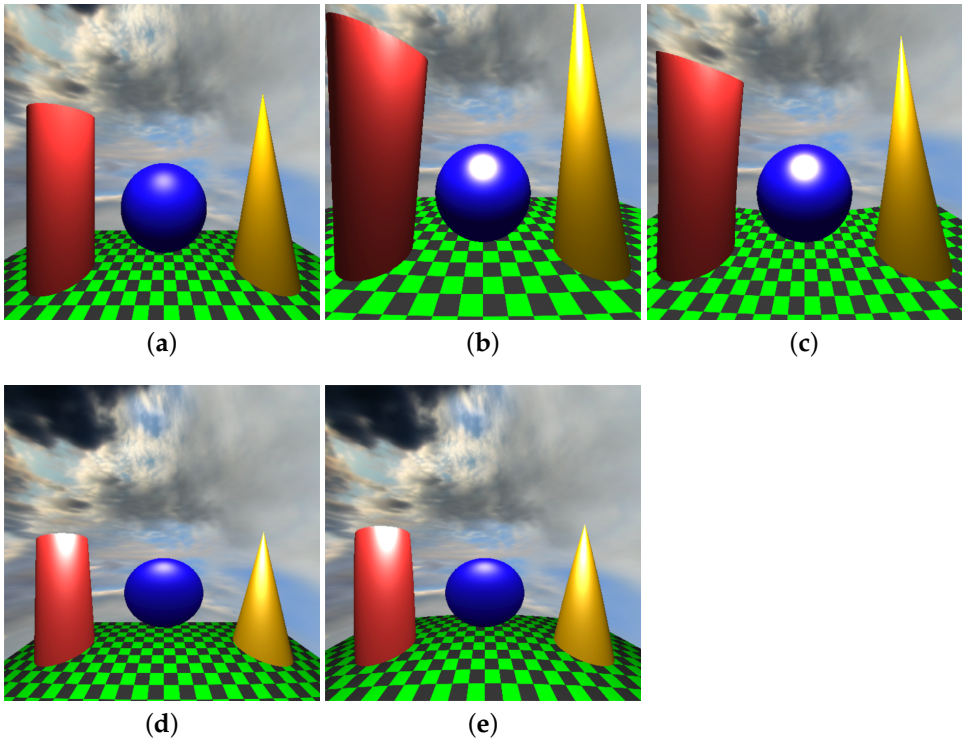


Figure 1. Modeling the scene directly in curved spaces versus porting the scene from Euclidean geometry with the exponential map. The exponential map makes those points of Euclidean and non-Euclidean spaces correspond to each other that are at the same direction and the same distance from the geometry origin of the spaces. Note that the base rectangle is supposed to be bounded by lines, but they are made curved by porting, whereas in our direct modeling approach, they are correctly represented as geodesics of the space. (**a**) Euclidean reference. (**b**) Modeled in elliptic space. (**c**) Ported to elliptic space. (**d**) Modeled in hyperbolic space. (**e**) Ported to hyperbolic space.

1.1. Previous Work

Curves can be specified by their parametric equation or by interpolation or approximation of *control points*. In Euclidean space, this is based on the center of mass analogy. Every control point $\mathbf{p}_i$ gets parameter dependent mass $m_i(t)$, and the point of the curve associated with the given parameter t is the center of mass of the resulting mechanical system:

$$\mathbf{r}(t) = \frac{\sum_i \mathbf{p}_i m_i(t)}{\sum_i m_i(t)}.$$

Unfortunately, this does not work in curved spaces, where, even if the control points are in the space of the geometry, their linear combination $\mathbf{r}(t)$ may be out.

There are several ways to generalize parametric spline curves to non-Euclidean spaces—see the recent survey of Mancinelli and Puppo [2]. The center of mass can be

given another interpretation [3,4] as the point from which the sum of the weighted squared distances to the control points is minimal:

$$\mathbf{r}(t) = \operatorname{argmin}_{\mathbf{r}} \sum_i d^2(\mathbf{r}, \mathbf{p}_i) m_i(t),$$

where $d(\mathbf{p}, \mathbf{q})$ is the distance of points $\mathbf{p}$ and $\mathbf{q}$, and the search is constrained to the space of the geometry. In non-Euclidean geometry, finding this optimum requires numerical optimization, which complicates the application of automatic differentiation techniques. The same is true for alternative generalizations of spline curves in terms of functional minimization [5–8] or recursive subdivision [9,10]. Yet, efficient and accurate derivatives are crucial for shading, animation, and differentiable rendering.

As non-Euclidean geometries also incorporate the concept of lines as *geodesic curves* [11], spline evaluation can be made geometry-independent if its construction is traced back to a sequence of interpolations along geodesics. Famous examples of parametric curve definition based on linear interpolations are the de Casteljau construction of the Bézier curve [12], the de Boor construction of the B-spline [13], the Neville's algorithm for Lagrange polynomial interpolation, or the Barry–Goldman method [14] for evaluating Catmull–Rom splines [15]—see the monograph [16] for a unified treatment based on the concept of blossoming [17].

The earliest attempts at extending linear interpolation to curved spaces focused on curves in two- or three-dimensional spheres (i.e., unit vectors or quaternions). Shoemake proposed including spherical linear interpolation (SLERP) [18] and spherical cubic interpolation (SQUAD) [19], which have become widely used in the field of computer animation and inspired many practical algorithms [20–22]—see ([23], Ch. 25) and [24] for overviews. The concept of blossoming has also been generalized to spherical geometry in various ways [25,26]. Schaefer and Goldman [27] have introduced variants of Bézier, Lagrange, and B-spline curves, as well as Catmull–Rom splines defined over spheres of arbitrary dimensions. Geier has proposed generalizations of Catmull–Rom and Kochanek–Bartels [28] splines to quaternions (as foundation for the Python 3.13 package *splines 0.3.3* (https://pypi.org/project/splines/, accessed on 25 July 2025)).

The aforementioned parametric curve constructions are specific to spherical geometries. De Casteljau's algorithm has also been successfully extended to general Riemannian manifolds [29–34]; however, the practical examples in these works are usually restricted to Lie groups, such as SO(3) or SE(3). We are aware of only a few publications that consider parametric curves in the hyperbolic plane [31,35] and none that do so in hyperbolic 3-space.

Rendering in curved spaces poses further challenges. Since light follows geodesics, visibility computations in ray tracing must generalize intersection tests to geodesic paths [36–38]. Object–space algorithms can be adapted by exploiting Klein's model, which maps geodesics to straight lines while preserving the visibility order [39,40].

1.2. Contributions

The purpose of this paper is to extend efficient Euclidean modeling and rendering techniques to constant-curvature non-Euclidean spaces. Our contributions are as follows:

- A generalization of the Kochanek–Bartels spline (in special cases also known as the Overhauser or Catmull–Rom spline) to curved geometries, including elliptic and hyperbolic spaces. The resulting curve interpolates control points, offers local control, and is defined entirely through motions along geodesics. In contrast to Geier's approach that provides a piecewise-Bézier formulation, our solution is based on the blending of quadratic interpolants.
- An extension of translational, rotational, and ruled parametric surfaces to curved spaces.

- Methods for computing spline tangents and surface normals in non-Euclidean geometries, supporting applications in shading, animation, and differentiable rendering.

The remainder of the paper is organized as follows. Section 1.3 reviews the *embedding space model* of Euclidean, elliptic, and hyperbolic geometries. In Section 2.1, we present our spline construction for curved spaces, while Section 2.2 focuses on the computation of tangent vectors. Section 2.4 derives formulas for pure translations and rotations, which are then applied in Section 2.5 to generate extruded, rotated and ruled parametric surfaces. Section 2.6 discusses efficient GPU-based rendering of these objects. The results are reported in Section 3, and the paper concludes with final remarks in Section 4.

1.3. The Embedding Space Model of Geometries and Notations

Euclidean, elliptic, and hyperbolic geometries can be described within a unified framework by embedding them into a higher-dimensional space [41,42]. Specifically, 3D geometries are realized as subsets of a 4D *embedding space*, where each point is represented by a vector $\mathbf{v} = (\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z, \mathbf{v}_w)$ with four coordinates (see Table 1 for notations). Figure 2 illustrates the simplified case of 2D geometries embedded in 3D space.

Table 1. Notations used in this paper.

Notation	Interpretation
$\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l}$	Basis vectors in the 4D embedding space
$\mathbf{p}, \mathbf{q}, \mathbf{r}, \mathbf{v}$	Points or vectors in the 4D embedding space
$\mathbf{n}$	Normal vector of a hyperplane in 4D
$\mathbf{e}$	Eye position in 4D embedding space
$\mathbf{g}$	Geometry origin: $(0, 0, 0, 1)$
$\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z, \mathbf{p}_w$	Coordinates of $\mathbf{p}$
$\mathbf{n}_{xyz}$	3D vector formed by the first three coordinates of 4D vector $\mathbf{n}$
$\mathbf{q}_{xyz0}$	4D vector where the w coordinate is replaced by zero
$\mathcal{L}$	1 for Euclidean and elliptic geometries and -1 for hyperbolic
$\langle \mathbf{a}, \mathbf{b} \rangle$	4D Euclidean or Lorentzian dot product depending on $\mathcal{L}$
$\mathbf{a}_{xyz} \cdot \mathbf{b}_{xyz}$	3D Euclidean dot product
$\mathbf{a}_{xyz} \times \mathbf{b}_{xyz}$	3D cross product
$S(\alpha)$	$\sin(\alpha)$ or $\sinh(\alpha)$ depending on $\mathcal{L}$
$C(\alpha)$	$\cos(\alpha)$ or $\cosh(\alpha)$ depending on $\mathcal{L}$

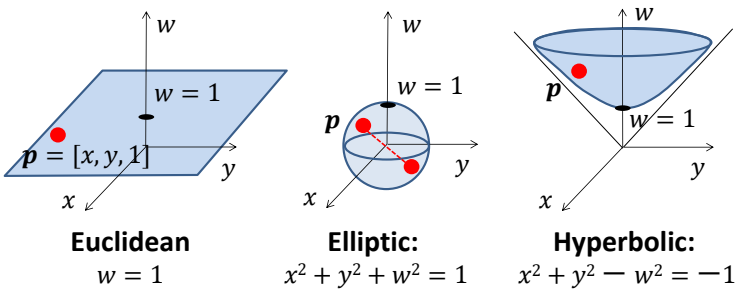


Figure 2. Embedding of the 2D Euclidean, elliptic, and hyperbolic geometries into 3D space. Points of Euclidean geometry lie on the plane $w = 1$, points of elliptic geometry lie on the unit sphere $x^2 + y^2 + w^2 = 1$, and points of hyperbolic geometry correspond to the upper sheet of the two-sheeted hyperboloid $x^2 + y^2 - w^2 = -1$.

The metric in the embedding space is defined by the *dot product* $\langle \mathbf{a}, \mathbf{b} \rangle$. For Euclidean and elliptic geometries, this is the standard Euclidean dot product; however, hyperbolic

geometry requires a Minkowski embedding with the Lorentzian dot product. To unify notation, we introduce the *curvature sign* $\mathcal{L}$, defined as

$$\mathcal{L} = \begin{cases} 1, & \text{Euclidean or elliptic geometry,} \\ -1, & \text{hyperbolic geometry.} \end{cases}$$

With this, the general dot product becomes

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}_x \mathbf{b}_x + \mathbf{a}_y \mathbf{b}_y + \mathbf{a}_z \mathbf{b}_z + \mathcal{L}\, \mathbf{a}_w \mathbf{b}_w. \tag{1}$$

For Euclidean 3D vectors, we also use the conventional dot symbol:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}_{xyz} \cdot \mathbf{b}_{xyz} + \mathcal{L}\, \mathbf{a}_w \mathbf{b}_w.$$

The embedding space is equipped with four basis vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l}$, that are normalized and mutually orthogonal:

$$\langle \mathbf{i}, \mathbf{i} \rangle = \langle \mathbf{j}, \mathbf{j} \rangle = \langle \mathbf{k}, \mathbf{k} \rangle = 1, \quad \langle \mathbf{l}, \mathbf{l} \rangle = \mathcal{L}, \tag{2}$$

$$\langle \mathbf{i}, \mathbf{j} \rangle = \langle \mathbf{j}, \mathbf{k} \rangle = \langle \mathbf{k}, \mathbf{l} \rangle = \langle \mathbf{l}, \mathbf{i} \rangle = 0. \tag{3}$$

1.3.1. Points

Points $\mathbf{p} = (\mathbf{p}_x, \mathbf{p}_y, \mathbf{p}_z, \mathbf{p}_w)$ of elliptic and hyperbolic geometries lie on their respective hyperspheres:

$$\langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L} \quad \Longleftrightarrow \quad \mathbf{p}_x^2 + \mathbf{p}_y^2 + \mathbf{p}_z^2 + \mathcal{L}\mathbf{p}_w^2 = \mathcal{L}. \tag{4}$$

In elliptic geometry, antipodal points $\mathbf{p}$ and $-\mathbf{p}$ are identical, i.e., a point corresponds to a diameter of the hypersphere. In hyperbolic geometry, only the upper sheet is considered, requiring $\mathbf{p}_w > 0$.

The point of embedding coordinates $(0, 0, 0, 1)$ lies in all three geometries (Euclidean, elliptic, hyperbolic). It is called the *geometry origin* and is denoted by $\mathbf{g}$.

1.3.2. Vectors

A *vector* $\mathbf{v}$ based at a point $\mathbf{p}$ must not point out of the space of the geometry. This requires $\mathbf{v}$ to lie in the 3D tangent hyperplane of the hypersphere at $\mathbf{p}$:

$$\langle \mathbf{p}, \mathbf{v} \rangle = 0. \tag{5}$$

1.3.3. Lines and Distances

Geodesics in elliptic and hyperbolic geometries arise from large circles that are intersections of the hypersphere with planes through the origin. Consider a sphere of radius R centered at the origin, a starting point $\mathbf{p}$, and an initial unit tangent vector $\mathbf{v}$. The parametric equation of the resulting great circle is given by

$$\mathbf{c}(t) = \mathbf{p} \cos(t/R) + \mathbf{v} R \sin(t/R), \tag{6}$$

which can be interpreted as a unit speed motion, since $\|\dot{\mathbf{c}}(t)\| = \sqrt{\langle \dot{\mathbf{c}}(t), \dot{\mathbf{c}}(t) \rangle} = 1$.

From the equation of a general main circle, the line of the elliptic geometry can be obtained by considering a unit radius sphere ($R = 1$):

$$\mathbf{r}_{\text{elliptic}}(t) = \mathbf{p} \cos(t) + \mathbf{v} \sin(t). \tag{7}$$

The distance d between points $\mathbf{p}$ and $\mathbf{q}$ is the travel time along the geodesic:

$$\mathbf{q} = \mathbf{p}\cos(d) + \mathbf{v}\sin(d) \quad \Longrightarrow \quad d = \cos^{-1}(|\langle \mathbf{p}, \mathbf{q} \rangle|).$$

The absolute value operation is needed because a point can be represented by both $\mathbf{p}$ and $-\mathbf{p}$. To resolve this ambiguity, we require that for points of a single object, the scalar product must be positive. When the distance from the camera is needed by rendering, objects, where the dot product with the eye position is negative, are removed by clipping, but the scene is rendered two times, once with the original points $\mathbf{p}$ and once with $-\mathbf{p}$.

The geodesic segment between $\mathbf{p}$ and $\mathbf{q}$ at distance d can also be written as a spherical interpolation (SLERP) [18]:

$$\mathbf{r}_{\text{elliptic}}(\tau) = \mathbf{p}\,\frac{\sin((1-\tau)d)}{\sin(d)} + \mathbf{q}\,\frac{\sin(\tau d)}{\sin(d)}, \qquad \tau \in [0,1], \tag{8}$$

which yields a uniform motion of speed d if τ is assumed to be the time.

In hyperbolic geometry, the geodesic takes the form

$$\mathbf{r}_{\text{hyperbolic}}(t) = \mathbf{p}\cosh(t) + \mathbf{v}\sinh(t), \tag{9}$$

corresponding to a sphere of imaginary radius $R = \sqrt{-1}$. The distance d between points $\mathbf{p}$ and $\mathbf{q}$ is

$$\mathbf{q} = \mathbf{p}\cosh(d) + \mathbf{v}\sinh(d) \quad \Longrightarrow \quad d = \cosh^{-1}(-\langle \mathbf{p}, \mathbf{q} \rangle).$$

A geodesic segment can be expressed as a hyperbolic interpolation [43]:

$$\mathbf{r}_{\text{hyperbolic}}(\tau) = \mathbf{p}\,\frac{\sinh((1-\tau)d)}{\sinh(d)} + \mathbf{q}\,\frac{\sinh(\tau d)}{\sinh(d)}, \qquad \tau \in [0,1]. \tag{10}$$

Both spherical and hyperbolic interpolations work for $d = 0$ as well, i.e., when $\mathbf{p} = \mathbf{q}$. For small d values, they can be approximated by the Euclidean version:

$$\mathbf{r}_{\text{elliptic}}(\tau) \approx \mathbf{r}_{\text{hyperbolic}}(\tau) \approx \mathbf{p}(1-\tau) + \mathbf{q}\tau.$$

2. Methods

This section introduces our approach for defining spline curves in curved spaces and generating surfaces through curve transformations. We employ a unified notation that simultaneously handles elliptic and hyperbolic geometries: functions $C(\alpha)$ and $S(\alpha)$ denote $\cos(\alpha)$ and $\sin(\alpha)$ in the elliptic case and $\cosh(\alpha)$ and $\sinh(\alpha)$ in the hyperbolic case.

The *uniform interpolation* between vectors $\mathbf{a}$ and $\mathbf{b}$ that are associated with points at distance d is defined as

$$I(\mathbf{a}, \mathbf{b}, d, \tau) = \frac{\mathbf{a}\,S((1-\tau)d) + \mathbf{b}\,S(\tau d)}{S(d)}.$$

The *line segment* with endpoints $\mathbf{p}$ and $\mathbf{q}$ is obtained by interpolating the location along their geodesic:

$$L(\mathbf{p}, \mathbf{q}, \tau) = I(\mathbf{p}, \mathbf{q}, d(\mathbf{p}, \mathbf{q}), \tau). \tag{11}$$

2.1. Geometry-Independent Interpolating Spline

We construct a smooth curve $\mathbf{r}(t)$ that interpolates a sequence of control points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ at knot values $t_1, \ldots, t_n$, respectively. Both the control points and the resulting curve lie in the underlying geometry:

$$\langle \mathbf{p}_i, \mathbf{p}_i \rangle = \langle \mathbf{r}(t), \mathbf{r}(t) \rangle = \mathcal{L}.$$

The construction relies only on the constant-speed motion $L(\mathbf{p}, \mathbf{q}, \tau)$ along the geodesic defined by $\mathbf{p}$ at $\tau = 0$ and $\mathbf{q}$ at $\tau = 1$. Thus, by substituting the appropriate line equation, the same algorithm applies in Euclidean, elliptic, and hyperbolic geometries.

Consider three control points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ to be interpolated at t_1, t_2, t_3 (Figure 3). Let d_{12} and d_{23} denote the distances between $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_2, \mathbf{p}_3$, respectively. We introduce two auxiliary points: $\mathbf{p}'_1$ on the extension of $\mathbf{p}_1 \to \mathbf{p}_2$, located at distance $\tau'_1 d_{12}$ from $\mathbf{p}_1$, and $\mathbf{p}'_3$ on the extension of $\mathbf{p}_3 \to \mathbf{p}_2$, located at distance $-\tau'_3 d_{23}$ from $\mathbf{p}_3$.

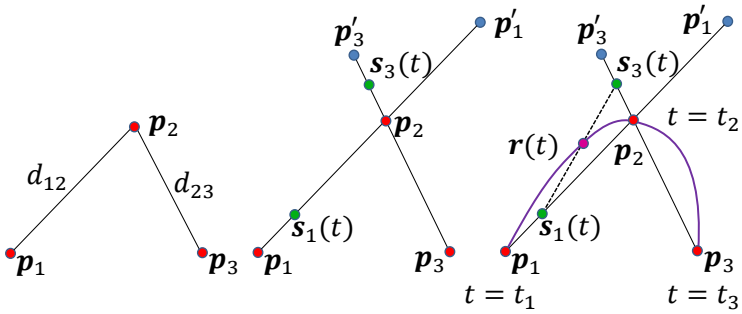


Figure 3. Spline construction for three control points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ as a composition of three motions. Motion $\mathbf{s}_1(t)$ goes from $\mathbf{p}_1$ to $\mathbf{p}'_1$, motion $\mathbf{s}_3(t)$ goes from $\mathbf{p}'_3$ to $\mathbf{p}_3$, and the spline $\mathbf{r}(t)$ is a motion between $\mathbf{s}_1(t)$ and $\mathbf{s}_3(t)$.

Define motions $\mathbf{s}_1(\tau_1) = L(\mathbf{p}_1, \mathbf{p}_2, \tau_1)$, extending toward $\mathbf{p}'_1$, and $\mathbf{s}_3(\tau_3) = L(\mathbf{p}_2, \mathbf{p}_3, \tau_3)$, extending toward $\mathbf{p}'_3$. Finally, let the spline of the three points be the motion interpolating between $\mathbf{s}_1(\tau_1)$ and $\mathbf{s}_3(\tau_3)$:

$$\mathbf{r}(t) = L(\mathbf{s}_1(\tau_1(t)), \mathbf{s}_3(\tau_3(t)), \tau_2(t)).$$

Interpolation conditions are satisfied when

$$\begin{aligned} \mathbf{r}(t_1) &= \mathbf{p}_1 \quad \text{if } \tau_1(t_1) = 0,\ \tau_2(t_1) = 0, \\ \mathbf{r}(t_2) &= \mathbf{p}_2 \quad \text{if } \tau_1(t_2) = 1,\ \tau_3(t_2) = 0, \\ \mathbf{r}(t_3) &= \mathbf{p}_3 \quad \text{if } \tau_2(t_3) = 1,\ \tau_3(t_3) = 1. \end{aligned}$$

A convenient choice of the line parameters is

$$\tau_1(t) = \left(\frac{t - t_1}{t_2 - t_1}\right)^{\gamma}, \quad \tau_2(t) = \left(\frac{t - t_1}{t_3 - t_1}\right)^{\beta}, \quad \tau_3(t) = 1 - \left(\frac{t_3 - t}{t_3 - t_2}\right)^{\gamma}, \tag{12}$$

yielding auxiliary distances

$$\tau'_1 = \left(\frac{t_3 - t_1}{t_2 - t_1}\right)^{\gamma}, \quad \tau'_3 = 1 - \tau'_1. \tag{13}$$

Here, γ is the *tension* parameter, controlling the curvature near control points, while β is the *bias* parameter, shifting the curve to anticipate or overshoot a turn.

For $\gamma = \beta = 1$ in Euclidean space, the parametrization reduces to

$$\tau_1(t) = \frac{t - t_1}{t_2 - t_1}, \quad \tau_2(t) = \frac{t - t_1}{t_3 - t_1}, \quad \tau_3(t) = \frac{t - t_2}{t_3 - t_2},$$

which yields Neville's construction of the Lagrange interpolating curve:

$$\mathbf{r}(t) = \mathbf{p}_1 \frac{t-t_2}{t_1-t_2}\frac{t-t_3}{t_1-t_3} + \mathbf{p}_2 \frac{t-t_1}{t_2-t_1}\frac{t-t_3}{t_2-t_3} + \mathbf{p}_3 \frac{t-t_1}{t_3-t_1}\frac{t-t_2}{t_3-t_2}.$$

As with classical Lagrange interpolation, this construction suffers from oscillations and poor local control for larger sets of control points.

To address these issues, we adapt Overhauser's approach [44]: the proposed interpolation scheme is applied only to local triplets, e.g., $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ and then $\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$, etc. The final curve is obtained by merging the overlapping segments using geodesic blending (Figure 4a):

$$\mathbf{r}(t) = L\left(\mathbf{r}_i(t), \mathbf{r}_{i+1}(t), \frac{t-t_i}{t_{i+1}-t_i}\right) \quad \text{if } t \in [t_i, t_{i+1}]. \tag{14}$$

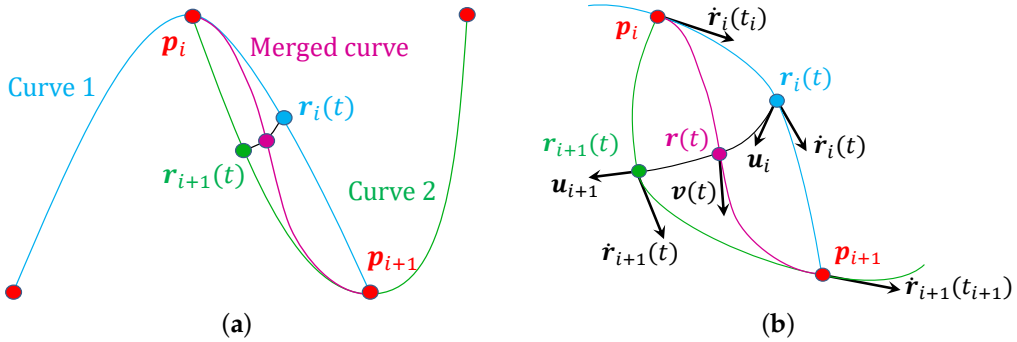


Figure 4. Overhauser style blending of two spline segments $\mathbf{r}_i(t)$ and $\mathbf{r}_{i+1}(t)$ sharing two control points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. (**a**) Blending Curve 1 defined by $\mathbf{r}_i(t)$ and Curve 2 defined by $\mathbf{r}_{i+1}(t)$. (**b**) Zoom in for continuity analysis. $\dot{\mathbf{r}}_i(t)$ and $\dot{\mathbf{r}}_{i+1}(t)$ are the velocities, and $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ are the unit tangents at the endpoints of the blending geodesic.

2.2. Spline Tangent Calculation

In applications such as path animation based on the Frenet frame or illumination computations involving surface normals, it is necessary to evaluate the tangent vector of the spline at specific points. This tangent is obtained as the derivative of the spline with respect to its parameter. Since our spline is constructed as a hierarchy of constant-speed motions along geodesics, we propose a recursive algorithm for tangent computation.

At each step, the inputs are the geodesic endpoints $\mathbf{p}_1$ and $\mathbf{p}_2$ and their velocities $\mathbf{v}_1$ and $\mathbf{v}_2$ (Figure 5). In addition, the distance d between the endpoints and the unit direction vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ of the geodesic at the endpoints can be expressed as

$$d = C^{-1}(\mathcal{L}\langle \mathbf{p}_1, \mathbf{p}_2 \rangle), \quad \mathbf{u}_1 = \frac{\mathbf{p}_2 - \mathbf{p}_1 C(d)}{S(d)}, \quad \mathbf{u}_2 = \frac{\mathbf{p}_2 C(d) - \mathbf{p}_1}{S(d)}.$$

The derivative of

$$\mathbf{r}(t) = L(\mathbf{p}_1(t), \mathbf{p}_2(t), \tau(t))$$

yields the velocity $\mathbf{v}(t)$ at point $\mathbf{r}(t)$ on the geodesic. It consists of three terms: (i) the motion of the endpoints, (ii) the change ointhe interpolation parameter τ, and (iii) the variation in the geodesic length. The formula for the velocity is

$$\mathbf{v}(t) = I(\mathbf{v}_1, \mathbf{v}_2, d, \tau) + I(\mathbf{u}_1, \mathbf{u}_2, d, \tau)\,(\dot{\tau} d + \tau \dot{d}) - \frac{S(\tau d)\,\dot{d}}{S(d)}\,\mathbf{u}_2, \tag{15}$$

where the derivative of the geodesic distance d between $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ is

$$\dot{d} = -\frac{\langle \mathbf{p}_1, \mathbf{v}_2 \rangle + \langle \mathbf{v}_1, \mathbf{p}_2 \rangle}{S(d)}. \tag{16}$$

The proof is given in Appendix A.

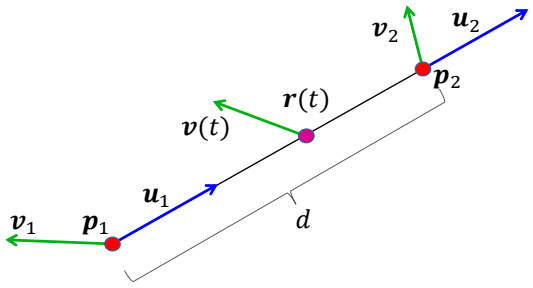


Figure 5. One step of the derivative calculation. Inputs are endpoint positions $\mathbf{p}_1$, $\mathbf{p}_2$, endpoint velocities $\mathbf{v}_1$, $\mathbf{v}_2$, and the derived quantities including distance d and unit direction vectors $\mathbf{u}_1$ and $\mathbf{u}_2$. Outputs are the position $\mathbf{r}(t)$ and velocity $\mathbf{v}(t)$ at time t.

2.3. Continuity Analysis

In between the control points, the spline of Equation (14) is continuously differentiable by an arbitrary number of times. To examine the smoothness at the control points, let us take the derivative expressed by Equation (15) of the spline defined in Equation (14), where $\mathbf{v}_1 = \dot{\mathbf{r}}_i(t)$ is the derivative of segment i at parameter point t, and $\mathbf{v}_2 = \dot{\mathbf{r}}_{i+1}(t)$ is the derivative of segment $i+1$ (Figure 4b). To prove continuous differentiability, i.e., C^1 continuity, we need to show that the derivative of $\mathbf{r}(t)$ goes to the derivative of $\mathbf{r}_i(t)$ and the derivative of $\mathbf{r}_{i+1}(t)$ when t converges to t_i and t_{i+1}, respectively. As segment i and segment $i+1$ share control points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$, the distance d between $\mathbf{r}_i(t)$ and $\mathbf{r}_{i+1}(t)$ converges to zero when t goes to t_i or t_{i+1}. Substituting these into Equation (15), we obtain

$$\lim_{t \to t_i} \mathbf{v}(t) = \lim_{\tau \to 0} \left(I(\dot{\mathbf{r}}_i(t), \dot{\mathbf{r}}_{i+1}(t), d, \tau) + I(\mathbf{u}_i, \mathbf{u}_{i+1}, d, \tau) \left(\dot{\tau} d + \tau \dot{d}\right) - \frac{S(\tau d)\, \dot{d}}{S(d)} \mathbf{u}_2 \right) = \dot{\mathbf{r}}_i(t_i), \tag{17}$$

since $I(\mathbf{a}, \mathbf{b}, d, \tau) = \mathbf{a}$ and $S(\tau d) = 0$ if $\tau = 0$, and $\dot{\tau} d + \tau \dot{d} = 0$ if both τ and d are zero. Similarly, when t goes to t_{i+1}, τ becomes 1 and d goes to zero; thus, we get

$$\lim_{t \to t_{i+1}} \mathbf{v}(t) = \dot{\mathbf{r}}_{i+1}(t_{i+1}) + \lim_{\tau \to 1} \left(\mathbf{u}_{i+1} (\dot{\tau} d + \tau \dot{d} - \dot{d}) = \dot{\mathbf{r}}_{i+1}(t_{i+1}), \right.$$

since $I(\mathbf{a}, \mathbf{b}, d, \tau) = \mathbf{b}$ and $S(\tau d)/S(d) = 1$ if $\tau = 1$, and $\dot{\tau} d + \tau \dot{d} - \dot{d} = 0$ if $\tau = 1$ and $d = 0$. This means that the derivative of the merged spline at the control points depends just on that segment that has this control point in the middle of its three control points. Thus, the spline has continuous velocity, i.e., belongs to the C^1 class.

2.4. Transformations and Isometries

Our goal is to express transformations in non-Euclidean spaces in terms of explicit formulas, ideally as 4×4 matrix multiplications. Such transformations must map the geometry onto itself; that is, the image of any point must remain within the same space. Since isometries preserve the dot product, they also preserve membership in the geometry, as point $\mathbf{p}$ belongs to the space precisely when $\langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L}$. Thus, enforcing dot product preservation guarantees that transformed points remain valid. In practice, this condition is satisfied if the transformed basis vectors form an orthonormal frame, i.e., they satisfy Equations (2) and (3).

In Euclidean geometry, translations and rotations can be expressed as compositions of two planar reflections [40]. If the reflecting planes are parallel, the result is a translation; if they intersect, the result is a rotation around their line of intersection (Figure 6).

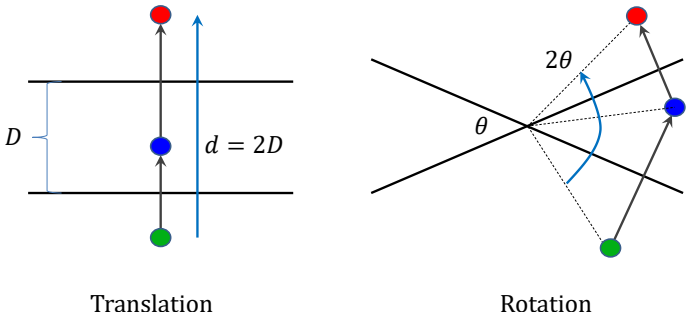


Figure 6. Translation and rotation constructed from two planar reflections. If the planes are parallel and are at distance D, then the two reflections make a translation by distance $d = 2D$. If the two planes are intersecting and have angle θ, then the two reflections are equivalent to a rotation by angle 2θ.

In elliptic and spherical geometries, parallels do not exist, and hence, true translations are absent. By contrast, in hyperbolic geometry there are many parallels, making the translation as reflections on parallels ambiguous. Nevertheless, both translations and rotations are essential for intuitive modeling. We therefore introduce the notions of *pure rotation* and *pure translation*. A pure rotation fixes the geometry origin, while a pure translation moves the origin to a new point. Moreover, translations must implement *parallel transport*, preserving direction as closely as possible.

By spherical symmetry, reflection on a plane of the geometry corresponds to reflection on a hyperplane of the embedding space (Figure 7). Such a hyperplane must pass through the origin of the embedding space to map the curved space onto itself and can be written as

$$\langle \mathbf{n}, \mathbf{r} \rangle = 0,$$

where $\mathbf{n}$ is its normal.

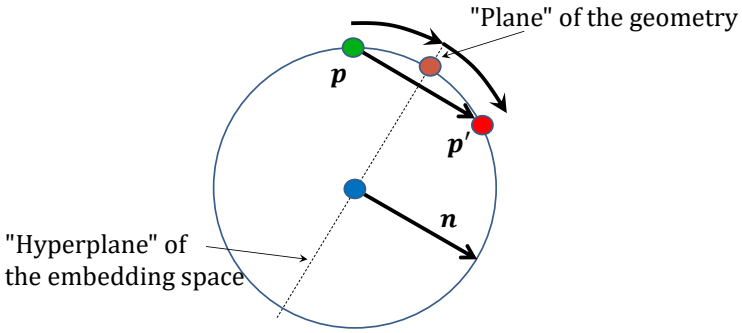


Figure 7. Reflection of $\mathbf{p}$ on a plane of the geometry corresponds to a reflection on a hyperplane of normal $\mathbf{n}$ in the embedding space (shown in a lower-dimensional example, where the plane is a point, and the hyperplane of the embedding space is a line).

Reflecting point $\mathbf{p}$ across the hyperplane with normal $\mathbf{n}^{(1)}$ yields

$$\mathbf{p}' = \mathbf{p} - 2\frac{\langle \mathbf{p}, \mathbf{n}^{(1)} \rangle}{\langle \mathbf{n}^{(1)}, \mathbf{n}^{(1)} \rangle}\mathbf{n}^{(1)}. \tag{18}$$

A second reflection, with normal $\mathbf{n}^{(2)}$, gives

$$\begin{aligned}\mathbf{p}'' &= \mathbf{p}' - 2\frac{\langle \mathbf{p}', \mathbf{n}^{(2)} \rangle}{\langle \mathbf{n}^{(2)}, \mathbf{n}^{(2)} \rangle}\mathbf{n}^{(2)} \\ &= \mathbf{p} - 2\frac{\langle \mathbf{p}, \mathbf{n}^{(1)} \rangle}{\langle \mathbf{n}^{(1)}, \mathbf{n}^{(1)} \rangle}\mathbf{n}^{(1)} - 2\frac{\langle \mathbf{p}, \mathbf{n}^{(2)} \rangle}{\langle \mathbf{n}^{(2)}, \mathbf{n}^{(2)} \rangle}\mathbf{n}^{(2)} + 4\frac{\langle \mathbf{n}^{(1)}, \mathbf{n}^{(2)} \rangle \langle \mathbf{n}^{(1)}, \mathbf{p} \rangle}{\langle \mathbf{n}^{(1)}, \mathbf{n}^{(1)} \rangle \langle \mathbf{n}^{(2)}, \mathbf{n}^{(2)} \rangle}\mathbf{n}^{(2)}. \quad (19)\end{aligned}$$

Since $\mathbf{p}''$ depends linearly on $\mathbf{p}$, transformations defined by reflections can be implemented with 4×4 matrices.

2.4.1. Translation

To construct a pure translation, we compose two reflections so that the origin $\mathbf{g}$ is mapped to $\mathbf{q}$ (Figure 8). The first reflection plane passes through $\mathbf{g}$, and the second lies midway between $\mathbf{g}$ and $\mathbf{q}$. To enforce parallel transport, the normals of both planes, together with $\mathbf{g}$ and $\mathbf{q}$, must be coplanar. A convenient choice is

$$\begin{aligned}\mathbf{n}^{(1)} &= \mathbf{q} - \mathbf{g}\mathbf{q}_w = [\mathbf{q}_x, \mathbf{q}_y, \mathbf{q}_z, 0] = \mathbf{q}_{xyz0}, \\ \mathbf{n}^{(2)} &= \mathbf{q} - \mathbf{g} = [\mathbf{q}_x, \mathbf{q}_y, \mathbf{q}_z, \mathbf{q}_w - 1]. \end{aligned} \quad (20)$$

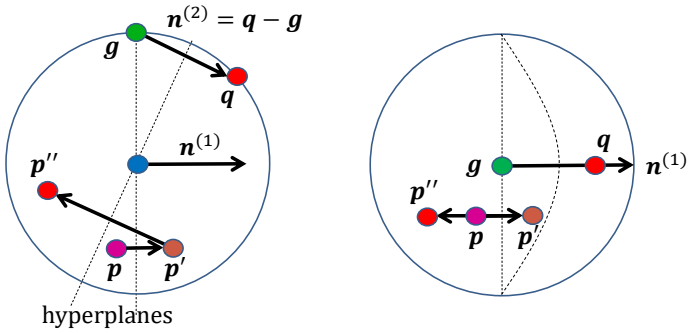


Figure 8. Translation of $\mathbf{p}$ to $\mathbf{p}''$ using two reflections, first on the hyperplane of normal $\mathbf{n}^{(1)}$ to get $\mathbf{p}'$ and then on the hyperplane of normal $\mathbf{n}^{(2)}$ to obtain $\mathbf{p}''$. The first reflection does not affect geometry origin $\mathbf{g}$, but the second takes it to $\mathbf{q}$.

Substituting into Equation (19) yields the following closed-form expression for a pure translation (see Appendix B for proof):

$$T_{\mathbf{q}}(\mathbf{p}) = \mathbf{p} + (\mathbf{q} - \mathbf{g})\mathbf{p}_w - (\mathbf{q} + \mathbf{g})\frac{\mathbf{p}_{xyz} \cdot \mathbf{q}_{xyz}}{\mathcal{L}(1 + \mathbf{q}_w)}. \quad (21)$$

2.4.2. Rotation

A pure rotation is also an isometry, but unlike a translation, it fixes the geometry origin $\mathbf{g} = (0, 0, 0, 1)$ (Figure 9). A reflection preserves the geometry origin if its w coordinate is zero; thus, we require $\mathbf{n}^{(1)}_w = \mathbf{n}^{(2)}_w = 0$. For simplicity, assume the normals have unit length. Then, Equation (19) reduces to

$$\mathbf{R}_{\mathbf{a},\phi}(\mathbf{p}) = \mathbf{p} - 2\left(\mathbf{p}_{xyz} \cdot \mathbf{n}^{(1)}_{xyz}\right)\mathbf{n}^{(1)} - 2\left(\mathbf{p}_{xyz} \cdot \mathbf{n}^{(2)}_{xyz}\right)\mathbf{n}^{(2)} + 4\left(\mathbf{n}^{(1)}_{xyz} \cdot \mathbf{n}^{(2)}_{xyz}\right)\left(\mathbf{p}_{xyz} \cdot \mathbf{n}^{(1)}_{xyz}\right)\mathbf{n}^{(2)}. \quad (22)$$

Here, the axis is given by

$$\mathbf{a}_{xyz} = \mathbf{n}^{(1)}_{xyz} \times \mathbf{n}^{(2)}_{xyz}, \qquad \mathbf{a}_w = 0,$$

and the rotation angle is $\phi = 2\theta$, where θ is the angle between the two normals.

Note that $\mathbf{p}_w$ remains unchanged since the w coordinates of the normal vectors are zero, and the xyz-coordinates follow Euclidean operations. Thus, in this setting the Rodrigues rotation formula applies not only in Euclidean space but also in hyperbolic and elliptic geometries.

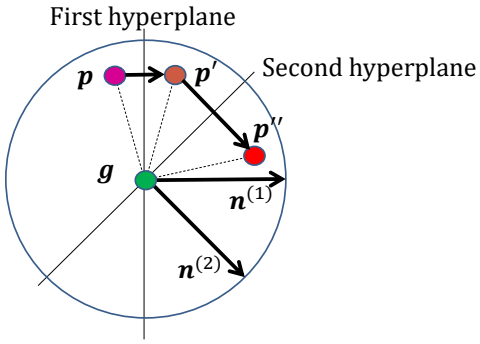


Figure 9. Rotation constructed as a double reflection, first reflecting $\mathbf{p}$ on the hyperplane of normal $\mathbf{n}^{(1)}$ to get $\mathbf{p}'$, and then reflecting $\mathbf{p}'$ on the hyperplane of normal $\mathbf{n}^{(2)}$ to obtain $\mathbf{p}''$ (top view). As geometry origin $\mathbf{g}$ is in both reflection hyperplanes, it is a fixed point of the transformation.

2.5. Parametric Surfaces

Surfaces are two-dimensional manifolds, which means that two independent motions are required to define them. The first is a *profile curve* $\mathbf{p}(u)$, describing the position of a point along the curve as a function of parameter u. The second is a transformation that moves the profile curve to sweep the surface. Two common transformations are translation (also called extrusion) and rotation. Alternatively, ruling generates a surface by prescribing two synchronized motions that define the endpoints of a moving line segment.

2.5.1. Extruded Surfaces

Translation is controlled by an additional *spine curve* $\mathbf{s}(v)$ parameterized by v, which translates the geometry origin to point $\mathbf{s}(v)$. Substituting the current spine point into the target $\mathbf{q}$ in Equation (21), the extruded surface point as a function of parameters u and v is given by

$$\mathbf{r}(u,v) = T_{\mathbf{s}(v)}(\mathbf{p}(u)) = \mathbf{p}(u) + (\mathbf{s}(v) - \mathbf{g})\mathbf{p}_w(u) - (\mathbf{s}(v) + \mathbf{g})\frac{\mathbf{p}_{xyz}(u) \cdot \mathbf{s}_{xyz}(v)}{\mathcal{L}(1 + \mathbf{s}_w(v))}. \tag{23}$$

2.5.2. Rotational Surfaces

A rotational surface is generated by rotating the profile curve by an angle v around a unit-length axis $\mathbf{a}$ passing through the geometry origin. Using Rodrigues' formula, we obtain

$$\begin{aligned} \mathbf{r}_{xyz}(u,v) &= \mathbf{p}_{xyz}(u)\cos(v) + \mathbf{a}_{xyz}\big(\mathbf{p}_{xyz}(u) \cdot \mathbf{a}_{xyz}\big)(1 - \cos(v)) + \mathbf{a}_{xyz} \times \mathbf{p}_{xyz}(u)\sin(v), \\ \mathbf{r}_w(u,v) &= \mathbf{p}_w(u). \end{aligned} \tag{24}$$

2.5.3. Ruled Surfaces

A ruled surface is formed as the locus of points swept by a moving line segment. It is specified by two endpoint curves, $\mathbf{r}_1(u)$ and $\mathbf{r}_2(u)$, defined as splines with common parameter u. Parameter $v \in [0,1]$ then interpolates along the line segment, yielding

$$\mathbf{r}(u,v) = L(\mathbf{r}_1(u), \mathbf{r}_2(u), v). \tag{25}$$

2.5.4. Tessellation

For rendering with the GPU pipeline, surfaces must be approximated by triangles. When triangles are sufficiently small, the curved patch can be approximately represented as the convex combination of its three vertices in the 4D embedding space. Approximating a parametric surface with a triangular mesh is called *tessellation*, which amounts to subdividing the (u, v) parameter space into triangles. For each vertex of parameters (u, v), the parametric function $\mathbf{r}(u, v)$ provides the location in the 4D embedding space.

2.5.5. Surface Normals

Rendering requires the calculation of the reflected radiance of surfaces visible in pixels. Since the reflected radiance depends on the surface normal vector $\mathbf{N}$, tessellation must also address normal computation. The normal vector at $\mathbf{r}(u, v)$ must lie in the tangent space of the geometry to avoid pointing out of the manifold, i.e., $\langle \mathbf{r}, \mathbf{N} \rangle = 0$ must hold. Furthermore, $\mathbf{N}$ must be orthogonal to the surface tangents $\partial \mathbf{r}/\partial u$ and $\partial \mathbf{r}/\partial v$. Thus, the 4D normal $\mathbf{N}$ is perpendicular to $\partial \mathbf{r}/\partial u$, $\partial \mathbf{r}/\partial v$, and $\mathbf{r}$, and can be expressed using the 4D analogue of the 3D cross product [45]:

$$\mathbf{N}(u,v) = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ \mathbf{r}_x & \mathbf{r}_y & \mathbf{r}_z & \mathcal{L}\mathbf{r}_w \\ \partial \mathbf{r}_x/\partial u & \partial \mathbf{r}_y/\partial u & \partial \mathbf{r}_z/\partial u & \mathcal{L}\partial \mathbf{r}_w/\partial u \\ \partial \mathbf{r}_x/\partial v & \partial \mathbf{r}_y/\partial v & \partial \mathbf{r}_z/\partial v & \mathcal{L}\partial \mathbf{r}_w/\partial v \end{vmatrix}. \tag{26}$$

2.6. Rendering

To generate an image of the virtual world, a camera must be introduced into the geometric space. Conceptually, a camera consists of an *eye position* $\mathbf{e}$, a set of pixels, and a rule that associates each pixel with the direction of a line containing points that are potentially visible through the pixel center. This construction follows from the fact that light travels along geodesics between the visible surfaces and the eye. The global orientation of the camera is specified by three orthogonal unit vectors in the tangent space of the eye position: the *right direction* $\mathbf{i}'$, the *up direction* $\mathbf{j}'$, and the *negative view direction* $\mathbf{k}'$. Pixels are identified by *Normalized Device Coordinates* (NDC) $(x_{\text{ndc}}, y_{\text{ndc}}) \in [-1, 1]^2$, with the range boundaries corresponding to the edges of the image: $y_{\text{ndc}} = -1$ is the bottom of the image, $y_{\text{ndc}} = 1$ is the top, $x_{\text{ndc}} = -1$ is the left boundary, and $x_{\text{ndc}} = 1$ is the right boundary.

The mapping between directions and pixels simplifies considerably if the camera has *canonical settings*: the eye position is at the geometry origin $\mathbf{g}$, $\mathbf{i}' = \mathbf{i}$, $\mathbf{j}' = \mathbf{j}$, and $\mathbf{k}' = \mathbf{k}$. To achieve this, we apply an isometric transformation (translation and rotation) to both the scene and the camera. Because isometries act simultaneously on objects and the camera, the resulting image remains unchanged.

This transformation can be expressed as a multiplication with the following matrix [1]:

$$\mathbf{V} = \begin{bmatrix} \mathbf{i}'_x & \mathbf{j}'_x & \mathbf{k}'_x & \mathcal{L}\mathbf{e}_x \\ \mathbf{i}'_y & \mathbf{j}'_y & \mathbf{k}'_y & \mathcal{L}\mathbf{e}_y \\ \mathbf{i}'_z & \mathbf{j}'_z & \mathbf{k}'_z & \mathcal{L}\mathbf{e}_z \\ \mathcal{L}\mathbf{i}'_w & \mathcal{L}\mathbf{j}'_w & \mathcal{L}\mathbf{k}'_w & \mathbf{e}_w \end{bmatrix}. \tag{27}$$

If the camera is small, we can rely on the analogy of Euclidean geometry (Figure 10). To associate pixel coordinates with ray direction $\mathbf{v}$, we set the x and y components of $\mathbf{v}$ to the pixel NDCs scaled by factors s_x and s_y. The scaling should make sure that the z-component of the ray direction is equal to -1. With relative parameters s_x and s_y we can

control the camera field-of-view and aspect ratio settings. Since the eye lies at the geometry origin, ray directions belong to its tangent space where $w = 0$:

$$\mathbf{v}(x_{\text{ndc}}, y_{\text{ndc}}) = (s_x x_{\text{ndc}}, s_y y_{\text{ndc}}, -1, 0). \tag{28}$$

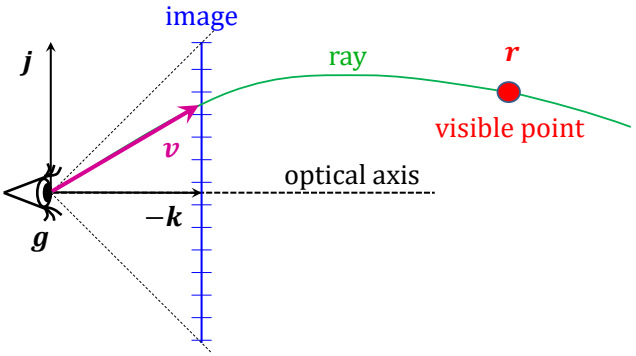


Figure 10. Canonical camera setup: The eye is in geometry origin **g**, the right direction is **i**, the up direction is **j**, and the viewing direction is $-\mathbf{k}$, where the negative sign ensures that the frame is right-handed. The start of the ray is the eye. The current pixel determines the initial direction vector **v** of the ray that contains the potentially visible points **r**.

Using the geodesic equation, the visibility ray parameterized by distance d from the eye being in **g** is

$$\mathbf{ray}(d) = \mathbf{g}C(d) + \mathbf{v}^0(x_{\text{ndc}}, y_{\text{ndc}})S(d),$$

where $\mathbf{v}^0 = \mathbf{v}/||\mathbf{v}||$ is the normalized version of the ray direction. Taking into account that $\mathbf{v}_z = -1$, the inverse operation is

$$\mathbf{v} = -\frac{\mathbf{v}^0}{\mathbf{v}^0_z}. \tag{29}$$

If objects are defined implicitly by equations of the form

$$f(\mathbf{r}) = 0 \text{ and } \langle \mathbf{r}, \mathbf{r} \rangle = \mathcal{L},$$

then their ray intersections can be determined by solving the scalar equation $f(\mathbf{ray}(d)) = 0$ for d. Among all positive solutions across all objects, the smallest one determines the visible surface and thus the pixel color.

In object-space rendering, the focus is shifted from the pixels to object points, and the task is to determine the pixel in which object point **r** appears. The unit direction of the ray containing **r** is

$$\mathbf{v}^0 = \frac{\mathbf{r} - \mathbf{g}C(d)}{S(d)} \implies \mathbf{v}^0_{xyz} = \frac{\mathbf{r}_{xyz}}{S(d)}, \tag{30}$$

where the distance d from the eye in **g** satisfies

$$C(d) = \mathcal{L}\langle \mathbf{r}, \mathbf{g} \rangle = \mathcal{L}^2 \mathbf{r}_w = \mathbf{r}_w, \tag{31}$$

since $\mathbf{g} = (0, 0, 0, 1)$.

The GPU expects the vertex shader to output points in homogeneous coordinates (X, Y, Z, W), where the $(X/W, Y/W)$ pair equals the pixel's normalized device coordinates, and $Z/W = \mathcal{Z}(d)$ is an increasing function of distance d, which is used for visibility determination. Thus, the point should be submitted in the following homogeneous form:

$$
\begin{aligned}
(x_{ndc}W, y_{ndc}W, \mathcal{Z}(d)W, W) &= \left(\frac{\mathbf{v}_x}{s_x}W, \frac{\mathbf{v}_y}{s_y}W, \mathcal{Z}(d)W, W\right) \\
&= \left(-\frac{\mathbf{v}_x^0}{s_x\mathbf{v}_z^0}W, -\frac{\mathbf{v}_y^0}{s_y\mathbf{v}_z^0}W, \mathcal{Z}(d)W, W\right) \\
&= \left(-\frac{\mathbf{r}_x}{s_x\mathbf{r}_z}W, -\frac{\mathbf{r}_y}{s_y\mathbf{r}_z}W, \mathcal{Z}(d)W, W\right),
\end{aligned} \tag{32}
$$

where we first applied Equation (28), then Equation (29), and finally Equation (30).

Setting W to $-\mathbf{r}_z$, the homogeneous coordinates have the following form:

$$
\mathbf{P}(\mathbf{r}) = \left(\frac{\mathbf{r}_x}{s_x}, \frac{\mathbf{r}_y}{s_y}, -\mathcal{Z}(d)\mathbf{r}_z, -\mathbf{r}_z\right). \tag{33}
$$

The clipping stage of the GPU retains points satisfying $-W \leq X, Y, Z \leq W$ and $W > 0$. Inequality $W > 0$ means that $W = -\mathbf{r}_z = -\mathbf{v}_z^0 S(d)$ is positive, which requires $S(d)$ to be positive. This is always the case in hyperbolic geometry, but in elliptic geometry, $S(d)$ is positive only in half of the main circle. Objects that are closer in the opposite direction are clipped away. To resolve this problem, the scene is rendered two times in elliptic geometry: once with the original points $\mathbf{r}$, and once with negated points $-\mathbf{r}$ [39].

If W is positive, then inequalities $-W \leq X, Y \leq W$ are satisfied by the construction in Equation (32), since the NDCs are between -1 and 1.

For coordinate $Z = \mathcal{Z}(d)W$, function $\mathcal{Z}(d)$ should be set to get the points of the view frustum to pass the clipping test. Let us search Z as a linear function of $\mathbf{r}_z$ and $\mathbf{r}_w$:

$$
Z = a\mathbf{r}_z + b\mathbf{r}_w, \tag{34}
$$

where parameters a and b are determined from the distance range of the view frustum. Using Equation (30) on the optical axis,

$$
W = -\mathbf{r}_z = -\mathbf{v}_z^0 S(d) = S(d),
$$

since on the optical axis $\mathbf{v} = (0, 0, -1, 0)$; thus, $\mathbf{v}_z^0 = -1$. Substituting this and Equation (31) into Equation (34), we get

$$
Z = -aS(d) + bC(d).
$$

On the optical axis, the visibility ray enters the visible frustum at $d_{\min}$ and leaves it at $d_{\max}$. The entry point must be on the front clipping plane of equation $Z = -W$, with the exit point on the back clipping plane of equation $Z = W$, which establishes a system of linear equations for unknowns a and b:

$$
\begin{aligned}
-aS(d_{\min}) + bC(d_{\min}) &= -S(d_{\min}), \\
-aS(d_{\max}) + bC(d_{\max}) &= S(d_{\max}).
\end{aligned}
$$

Solving this, the a and b parameters are

$$
a = -\frac{S(d_{\min} + d_{\max})}{S(d_{\max} - d_{\min})}, \qquad b = -\frac{2S(d_{\min})S(d_{\max})}{S(d_{\max} - d_{\min})}.
$$

3. Results

In order to demonstrate the discussed algorithms, first we compare the 2D spline curves in Euclidean, elliptic, and hyperbolic planes. To visualize the curved spaces, they are projected onto the Euclidean plane with central projection (Klein's model) and stereographic projection (Poincaré's model). In the case of hyperbolic geometry, the whole plane is

projected to a circle that is drawn in gray. For elliptic and Euclidean geometries, the projection is infinitely large; so, only a part of the projected plane is visible here. The Klein model shows geodesics as Euclidean lines, while the Poincaré model preserves angles but presents geodesics as circular arcs. Figure 11 shows the proposed spline with unit bias and tension for all three considered geometries and for both central and stereographic projections. The tangents are also depicted to show the results of the derivative calculation.

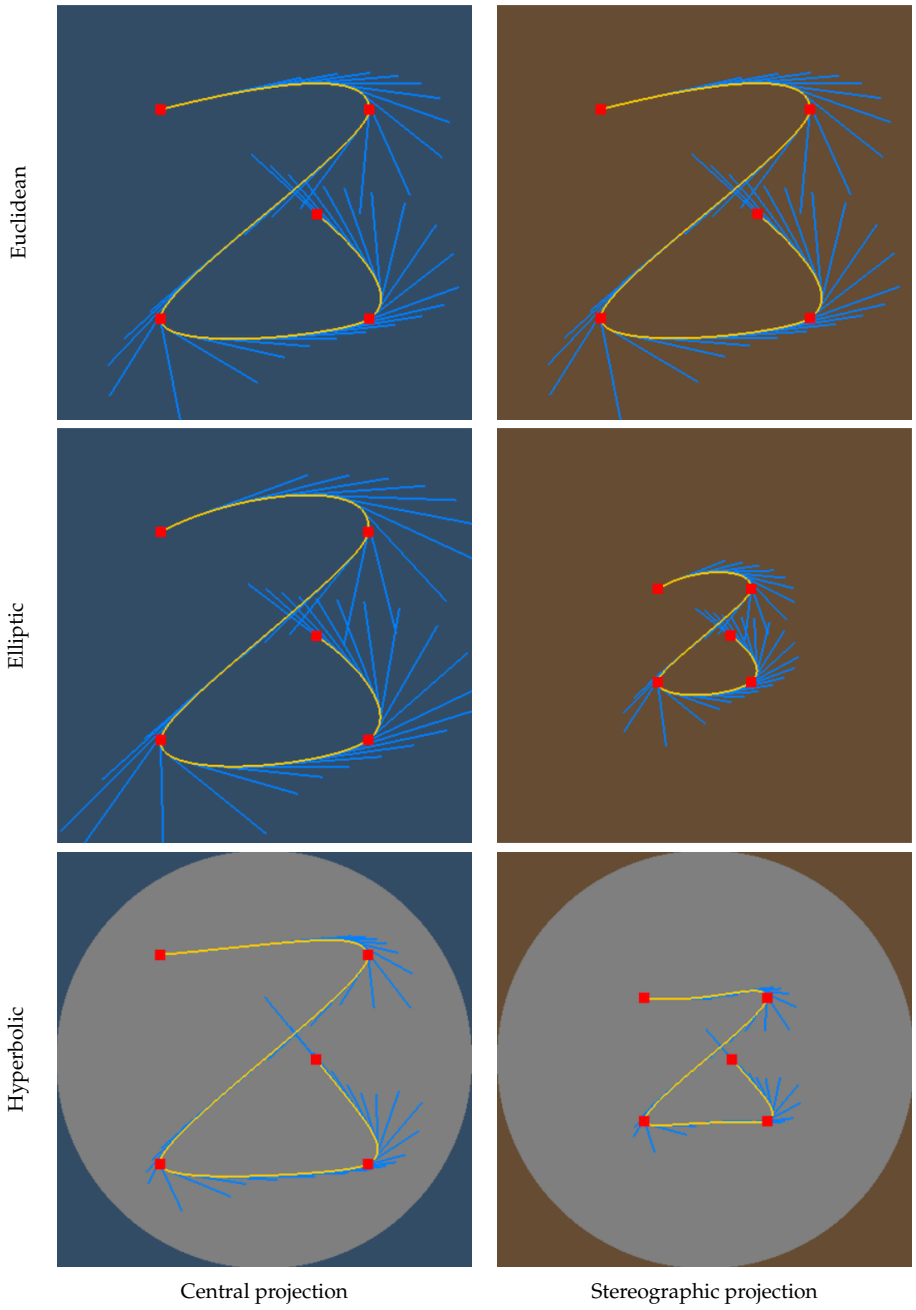
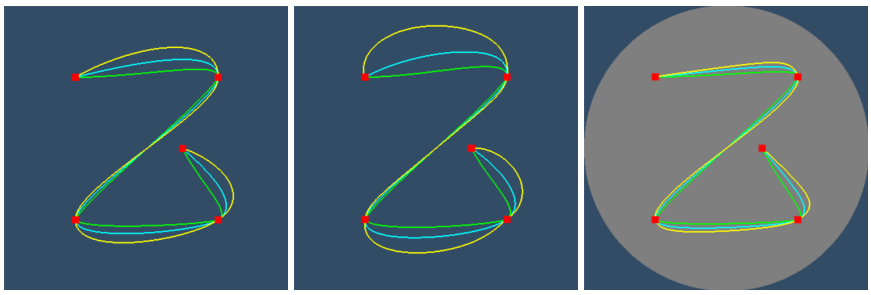


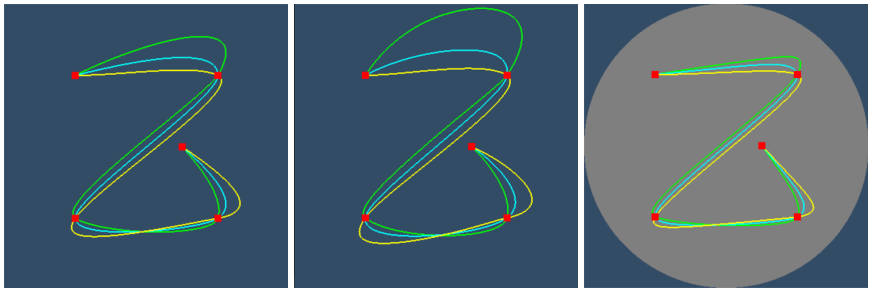
Figure 11. Our spline with yellow color in Euclidean (**top**), elliptic (**middle**), and hyperbolic (**bottom**) geometries. Left: central projection (Klein's model). Right: stereographic projection (Poincaré's model). Cyan line segments indicate tangents. Red squares are the control points.

Figures 12 and 13 examine the effects of the tension and bias parameters. If tension γ is increased, then the curvature in the control points is reduced, but it is increased farther from the control points. If bias β is increased, the previous control point affects more the tangent in the current control point than the control point after the current one.



(**a**) (**b**) (**c**)

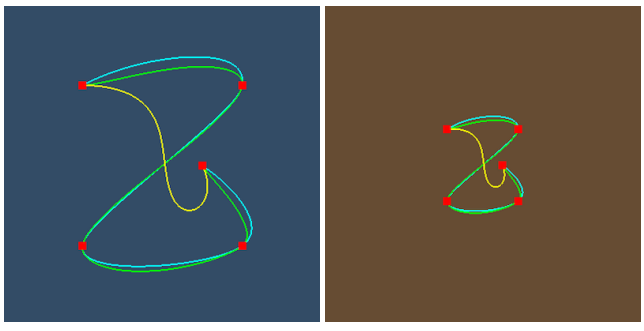
Figure 12. The effect of tension parameter γ on the spline rendered with central projection (Klein's model). Green, cyan, and yellow splines correspond to $\gamma = 0.5$, $\gamma = 1$, and $\gamma = 1.5$, respectively. (**a**) Euclidean. (**b**) Elliptic. (**c**) Hyperbolic.



(**a**) (**b**) (**c**)

Figure 13. The effect of bias parameter β on the spline rendered with central projection (Klein's model). Green, cyan, and yellow splines correspond to $\beta = 0.5$, $\beta = 1$, and $\beta = 2$, respectively. (**a**) Euclidean. (**b**) Elliptic. (**c**) Hyperbolic.

Figure 14 compares our proposed spherical spline to *Spherical Quadrangle Interpolation* (SQUAD) [19] and to the direct adaptation of de Casteljau's algorithm [34] to the sphere. As the SQUAD works for quaternions, i.e., for the sphere in 4D, the z coordinate of the SQUAD solution is fixed to zero. To make the different solutions similar, we set the bias and tension parameters to one and assigned the natural numbers to the knot values since unlike our solution, de Casteljau's method and the SQUAD both assume uniform parametrization. To find a spline point for a given parameter value, our method requires 1.1 µs computation time on a Intel i7 2.6GHz processor (Intel Corporation, Santa Clara, CA, USA), SQUAD 0.6 µs, and de Casteljau's method 1.8 µs. This corresponds to the fact that our method needs seven geodesic interpolations per segment, SQUAD uses three, and de Casteljau's approach $N(N-1)/2$ evaluations, where N is the number of control points.



(**a**) (**b**)

Figure 14. Comparison of our proposed spline (cyan) to SQUAD (green) and to the application of the de Casteljau's method (yellow) to define curves on the spherical surface. (**a**) Central projection. (**b**) Stereographic projection.

The 3D results are shown in a scene where objects are above a textured rectangle modeled as a ruled surface of two lines (Figures 15–17). The stage rectangle and the objects above it are enclosed in a sky sphere, which is generated as a rotational surface. The eye is put to the geometry origin $(0, 0, 0, 1)$, the up direction of the camera is $(0, 0, 1, 0)$, while the view direction is $(0, 1, 0, 0)$, which are transformed to the canonical frame by the view transformation.

Note that the edges of the rectangular stage show up as Euclidean lines, although they are geodesics of curved spaces. The explanation is as follows: Before arriving at the coordinate system of the canonical camera, isometric transformations are applied that map geodesics to geodesics. A geodesic is the combination of two points (Equations (8) and (10)); thus, it is in the plane defined by the two combined points and the origin of the embedding space. Projection transformation $\mathbf{P}(\mathbf{r})$ is a homogeneous linear transformation that maps planes to planes. Finally, when the GPU executes the homogeneous division and obtains the points in 3D Cartesian coordinates, the intersection of the transformed plane and the plane of $W = 1$ is determined, which is a Euclidean line.

To allow the comparison of scenes in different geometries, we tried to make them as similar as possible. The control points defining the spline surfaces are at the same distance and direction from the geometry origin in every examined geometric space.

Figure 1 also follows this setting, where classical surfaces like the cone, cylinder, and the sphere are modeled as rotational surfaces. Note that in hyperbolic geometry, visibility rays diverge faster than in Euclidean geometry; therefore, objects in the hyperbolic space look smaller. In elliptic geometry, however, visibility rays diverge first, and then, they converge; thus both close and far away, objects look large. Using the analogy of the Earth, if the eye position is at the North Pole, then objects close to the North and South Poles look large, while objects close to the equator look small.

Figure 15 is a spline curve approximating a dolphin contour in 3D. The rows correspond to different tension and bias settings and the columns to the three compared geometries. Unlike Figures 11–14, the control points have been placed in the 3D curved spaces, or, in other words, in the 4D embedding space, making sure that the control point also belongs to the hyperbolic Euclidean and elliptic geometry. The proposed construction guarantees that all points of the the resulting splines belong to the respective space. The spline is drawn with different tension and bias parameters to demonstrate their effect in the 3D case. As in 2D, with smaller tension parameters γ, the curvature of the spline grows at the control points. With larger bias, we can emphasize the effect of the previous control point in the tangent of the curve at the current control point (control points are specified in clock-wise order in this figure).

Figure 16 presents extruded surfaces with their profile and the spine curves. Rows present the profile curve, the spine curve, and the final extruded surface obtained by sweeping the profile curve along the spine curve. Figure 17 displays the rotated surfaces together with their profile curves. The axis of rotation is $(0, 0, 1, 0)$ in all examples.

The profile and the spine curves originally start close to the geometry origin, where the profile curve is extruded or rotated to generate the parametric surface. The parametric surface is finally translated to its position to separate the objects; then, the view transformation is applied to take the camera to its canonical settings.

When parametric surfaces are rendered, the normal vector of the surface is determined from the curve tangents according to Equation (26). When objects are transformed, the normal vectors are also updated. During illumination computation, we applied the diffuse-specular Phong–Blinn model depending on the transformed normal, illumination, and

viewing directions. The unit viewing direction **V** and the unit illumination direction **L** at point **r** are obtained from the equation of the geodesics:

$$\mathbf{V} = \frac{\mathbf{r} - \mathbf{e}C(d_c)}{S(d_c)}, \quad \mathbf{L} = \frac{\mathbf{r} - \mathbf{s}C(d_l)}{S(d_l)},$$

where **e** is the eye position, d_c is the distance to the camera, **s** is the position of the light source, and d_l is the distance to the illuminated point.

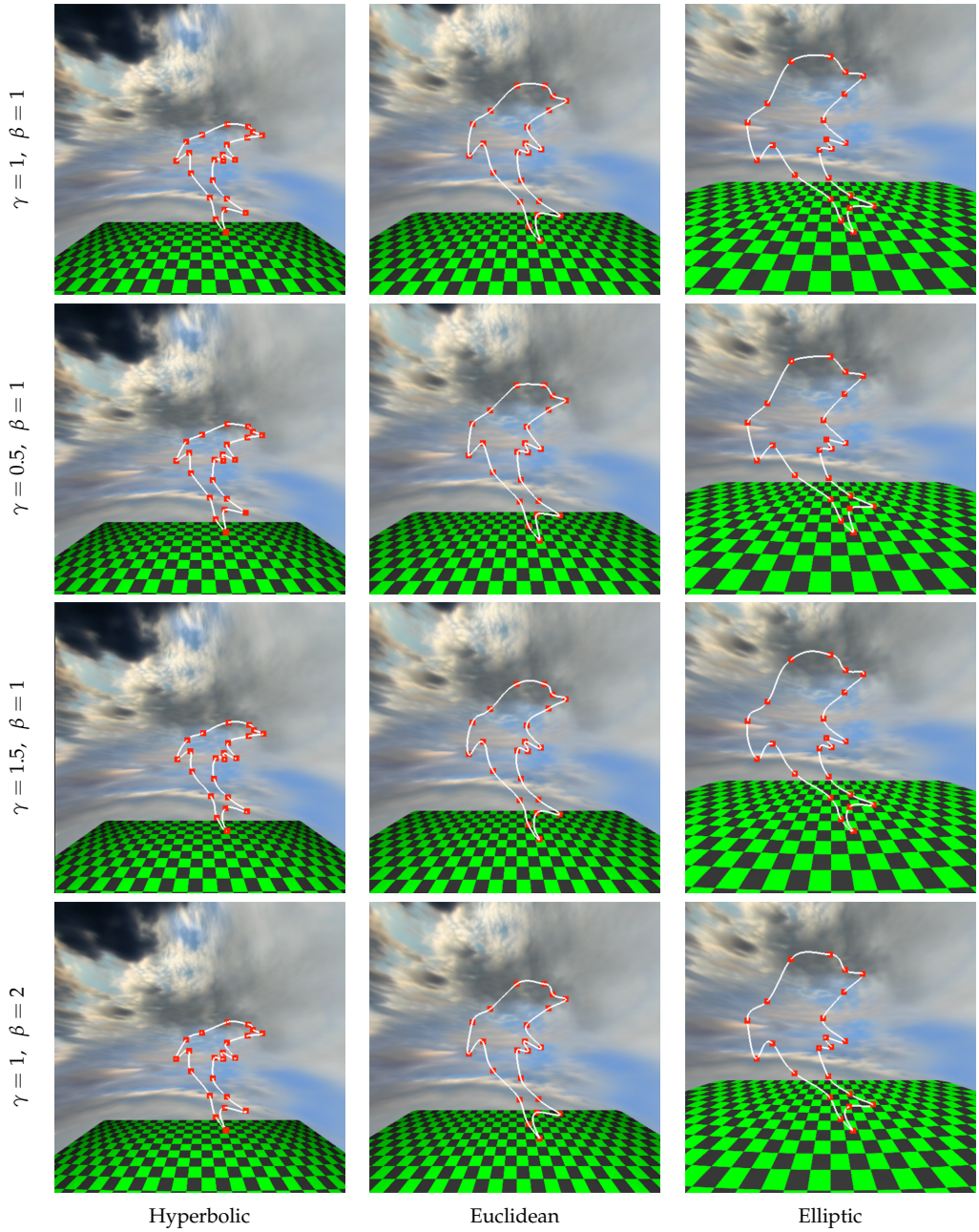


Figure 15. Spline curve depicting the contour of a dolphin in the three model geometries of the columns. Rows correspond to different tension γ and bias β settings.

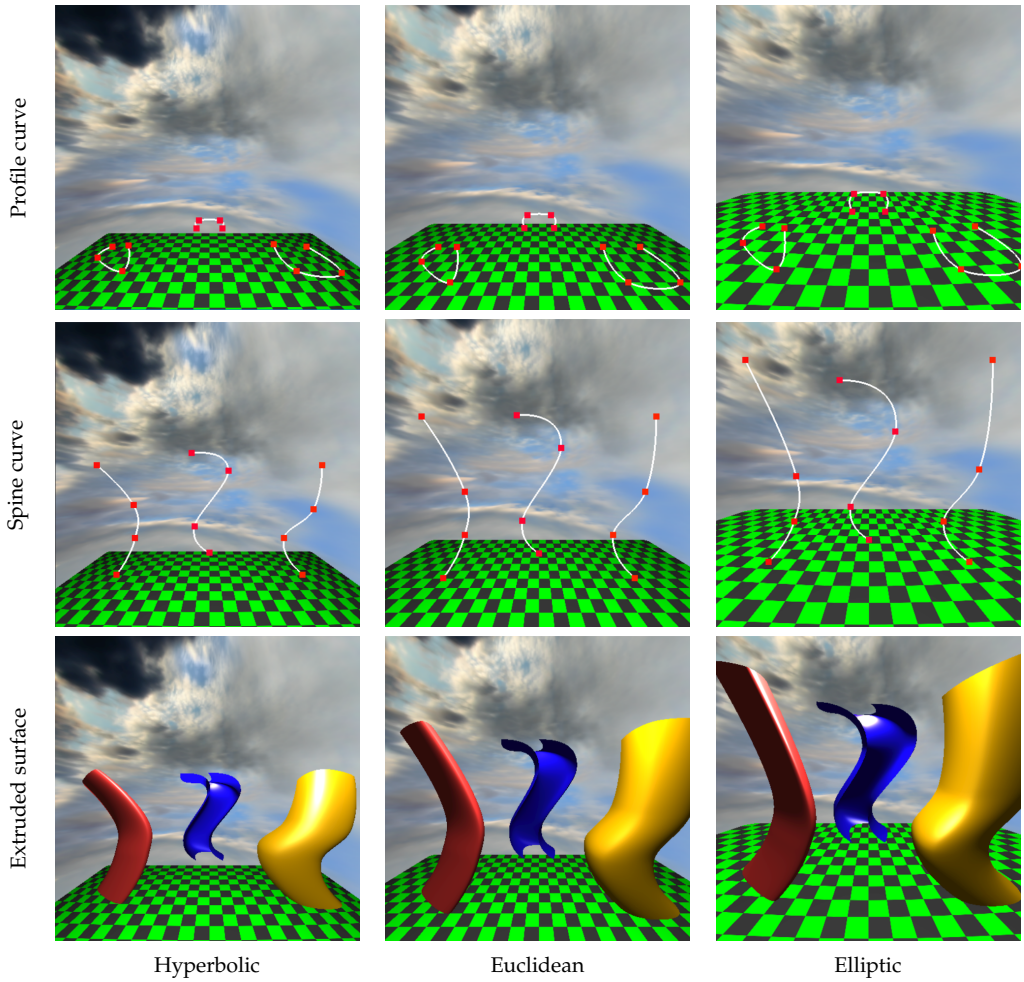


Figure 16. Extruded surfaces.

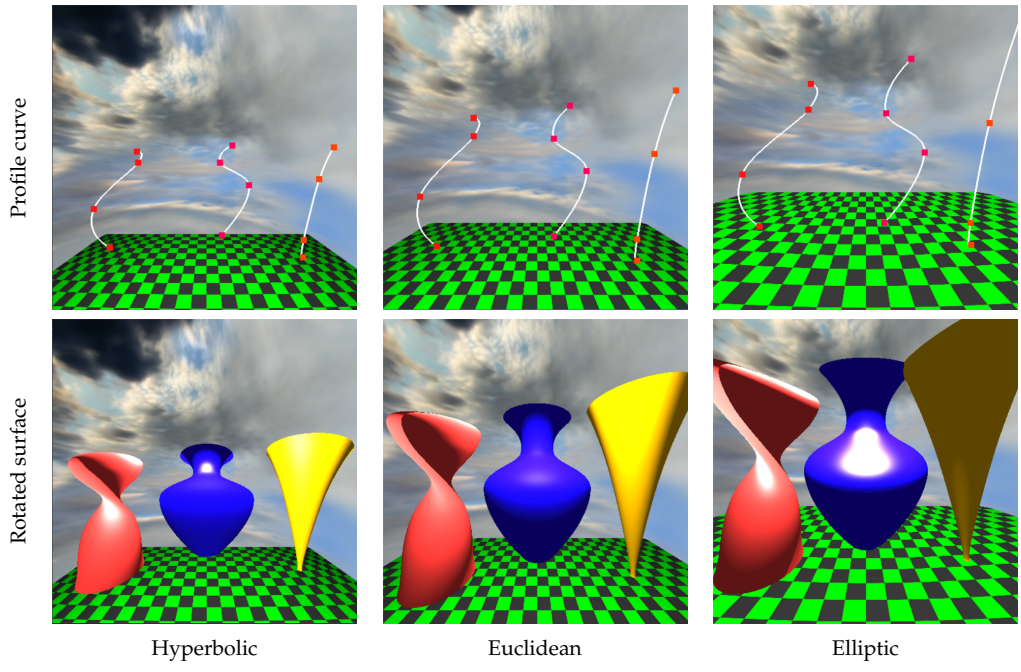


Figure 17. Rotated surfaces. The profile curves are originally close to the geometry origin, where they are rotated and then translated their final position.

4. Conclusions

We have presented a unified framework for constructing and rendering parametric curves and surfaces in constant-curvature geometries. Building on the embedding space model, we developed spline representations suitable for Euclidean, elliptic, and hyperbolic spaces, together with a recursive method for tangent calculation. Using these tools, we derived explicit formulas for extruded, rotational, and ruled surfaces, and addressed their tessellation and normal vector computation in the 4D embedding space.

On the rendering side, we demonstrated how cameras can be consistently defined via geodesic visibility lines and showed that view transformations reduce to matrix multiplications in the embedding space. This allowed us to describe both ray-based and object-based rendering methods and to integrate them with the incremental GPU pipeline through appropriate transformations. The resulting framework provides a foundation for efficiently modeling and visualizing curved geometries in real time.

Extending our techniques beyond constant-curvature spaces would require overcoming several challenges. Geodesic interpolation has no closed-form expression in general manifolds; so, extending spline curves to these spaces must involve numerical approximations. For special cases (e.g. matrix Lie groups, homogeneous spaces, etc.), the exponential and logarithmic maps—and thus geodesic interpolation—can still be computed via relatively inexpensive linear algebra. General manifolds have almost no global isometries; so, strict analogues of extruded and rotational surfaces can only exist in highly symmetric cases and, even then, only in distinguished directions. Nevertheless, manifold-based analogues of procedural surface definitions might be worthy of investigation, using, e.g., parallel transport or curve developments. Investigating alternatives to ray tracing for the visualization of Thurston geometries [46] is another promising avenue for related future research.

Author Contributions: Conceptualization, L.S.-K.; methodology, L.S.-K. and M.V.; software, A.F. and L.S.; validation, A.F.; formal analysis, L.S.-K. and M.V.; investigation, L.S.-K., A.F., L.S. and M.V.; writing—original draft preparation, L.S.-K.; writing—review and editing, A.F., L.S. and M.V.; visualization, A.F. and L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This project has been supported by the National Research, Development and Innovation Office, Hungary (OTKA K-145970).

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: GPUs used in this project are donations of NVIDIA.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GPU	Graphics Processing Unit
NDC	Normalized Device Coordinates
SLERP	Spherical Linear Interpolation
SQUAD	Spherical QUADrangle interpolation

Appendix A. Proof of Equation (15)

Let us compute the time derivative of position $\mathbf{r}(t)$ selected by parameter τ on the line segment L:

$$\mathbf{r}(t) = L(\mathbf{p}_1(t), \mathbf{p}_2(t), \tau(t)) = I(\mathbf{p}_1(t), \mathbf{p}_2(t), d(t), \tau(t)).$$

The time derivative is denoted by Newton's dot notation:

$$\begin{aligned}
\dot{\mathbf{r}}(t) &= I(\dot{\mathbf{p}}_1(t), \dot{\mathbf{p}}_2(t), d(t), \tau(t)) \\
&+ \frac{\mathbf{p}_1 C((1-\tau)d)(\dot{d} - (\dot{\tau d}))S(d) + \mathbf{p}_2 C(\tau d)(\dot{\tau d})S(d)}{S^2(d)} \\
&- \frac{\mathbf{p}_1 S((1-\tau)d)C(d) + \mathbf{p}_2 S(\tau d)C(d)}{S^2(d)}\dot{d} \\
&= I(\dot{\mathbf{p}}_1(t), \dot{\mathbf{p}}_2(t), d(t), \tau(t)) && \text{(A1)} \\
&+ \frac{\mathbf{p}_1\big(C((1-\tau)d)S(d) - S((1-\tau)d)C(d)\big) - \mathbf{p}_2 S(\tau d)C(d)}{S^2(d)}\dot{d} && \text{(A2)} \\
&+ \frac{\mathbf{p}_2 C(\tau d) - \mathbf{p}_1 C((1-\tau)d)}{S(d)}(\dot{\tau d}). && \text{(A3)}
\end{aligned}$$

The first term (Equation (A1)) is simply the interpolation of the endpoint velocities, $I(\mathbf{v}_1, \mathbf{v}_2, d, \tau)$. In the second term (Equation (A2)), we have collected the factors of the derivative of distance $\dot{d}$ between the endpoints, and in the third term (Equation (A3)), we collected the factors of $(\dot{\tau d})$.

Appendix A.1. Factor of $\dot{d}$ (Equation (A2))

Applying the addition rule for sin or sinh,

$$S(d)C((1-\tau)d) - C(d)S((1-\tau)d) = S(d - (1-\tau)d) = S(\tau d),$$

we obtain

$$\frac{\mathbf{p}_1(C((1-\tau)d)S(d) - S((1-\tau)d)C(d)) - \mathbf{p}_2 S(\tau d)C(d)}{S^2(d)} = \frac{S(\tau d)}{S(d)} \cdot \frac{\mathbf{p}_1 - \mathbf{p}_2 C(d)}{S(d)}.$$

This can be further rewritten as

$$\frac{S(\tau d)}{S(d)} \cdot \frac{\mathbf{p}_1 - \mathbf{p}_2 C(d)}{S(d)} = -\frac{S(\tau d)}{S(d)}\,\mathbf{u}_2,$$

using the equation of the line (Equations (7) and (9)),

$$\mathbf{p}_1 = \mathbf{p}_2 C(d) - \mathbf{u}_2 S(d).$$

Appendix A.2. Factor of $(\dot{\tau d})$ (Equation (A3))

The point at distance τd from $\mathbf{p}_1$ on the line segment can be expressed both from the start and from the end points with their unit direction vectors:

$$\mathbf{p}_2 C(\tau d) - \mathbf{u}_2 S(\tau d) = \mathbf{p}_1 C((1-\tau)d) + \mathbf{u}_1 S((1-\tau)d),$$

which rearranges to

$$\mathbf{p}_2 C(\tau d) - \mathbf{p}_1 C((1-\tau)d) = \mathbf{u}_1 S((1-\tau)d) + \mathbf{u}_2 S(\tau d).$$

Thus, the factor of $(\dot{\tau d})$ can be written as

$$\frac{\mathbf{p}_2 C(\tau d) - \mathbf{p}_1 C((1-\tau)d)}{S(d)} = I(\mathbf{u}_1, \mathbf{u}_2, d, \tau(t)).$$

Appendix A.3. Derivative of the Distance: $\dot{d}$

Now consider the derivative of the distance with respect to t. The distance d between points $\mathbf{p}_1$ and $\mathbf{p}_2$ satisfies

$$C(d) = \mathcal{L}\langle \mathbf{p}_1, \mathbf{p}_2 \rangle.$$

Differentiating both sides gives

$$-\mathcal{L}S(d)\dot{d} = \mathcal{L}(\langle \mathbf{v}_1, \mathbf{p}_2 \rangle + \langle \mathbf{p}_1, \mathbf{v}_2 \rangle).$$

Solving it for $\dot{d}$ yields Equation (16).

Appendix B. Proof of Equation (21)

Let us substitute Equation (20) into the general formula of Equation (19). As the first three coordinates of $\mathbf{n}^{(1)}$ and $\mathbf{n}^{(2)}$ are equal and the fourth coordinate of $\mathbf{n}^{(1)}$ is zero, we can use the following simplifying identity:

$$\langle \mathbf{n}^{(1)}, \mathbf{n}^{(2)} \rangle = \langle \mathbf{n}^{(1)}, \mathbf{n}^{(1)} \rangle = \mathbf{q}_{xyz} \cdot \mathbf{q}_{xyz};$$

thus, the transformed point after two reflections is

$$\begin{aligned} \mathbf{p}'' &= \mathbf{p} - 2\frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathbf{q}_{xyz} \cdot \mathbf{q}_{xyz}} \mathbf{q}_{xyz0} - 2\frac{\langle \mathbf{p}, \mathbf{q} - \mathbf{g} \rangle}{\langle \mathbf{q} - \mathbf{g}, \mathbf{q} - \mathbf{g} \rangle}(\mathbf{q} - \mathbf{g}) + 4\frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\langle \mathbf{q} - \mathbf{g}, \mathbf{q} - \mathbf{g} \rangle}(\mathbf{q} - \mathbf{g}) \\ &= \mathbf{p} - 2\frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathbf{q}_{xyz} \cdot \mathbf{q}_{xyz}} \mathbf{q}_{xyz0} + 2\frac{\langle \mathbf{p}, 2\mathbf{q}_{xyz0} - \mathbf{q} + \mathbf{g} \rangle}{\langle \mathbf{q} - \mathbf{g}, \mathbf{q} - \mathbf{g} \rangle}(\mathbf{q} - \mathbf{g}). \end{aligned}$$

Rearranging Equation (4) for $\mathbf{q}$ that is in the geometry, we obtain

$$\mathbf{q}_{xyz} \cdot \mathbf{q}_{xyz} = \mathcal{L}(1 - \mathbf{q}_w^2).$$

Scaler product $\langle \mathbf{q} - \mathbf{g}, \mathbf{q} - \mathbf{g} \rangle$ is expressed using the definition of $\mathbf{g} = (0, 0, 0, 1)$ and $\langle \mathbf{q}, \mathbf{q} \rangle = \mathcal{L}$:

$$\langle \mathbf{q} - \mathbf{g}, \mathbf{q} - \mathbf{g} \rangle = 2\mathcal{L}(1 - \mathbf{q}_w).$$

Vector $2\mathbf{q}_{xyz0} - \mathbf{q} + \mathbf{g}$ can be obtained as

$$2\mathbf{q}_{xyz0} - \mathbf{q} + \mathbf{g} = \mathbf{q}_{xyz0} + \mathbf{g}(1 - \mathbf{q}_w).$$

With these, the double reflection formula can be further simplified:

$$\begin{aligned} \mathbf{p}'' &= \mathbf{p} - 2\frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathcal{L}(1 - \mathbf{q}_w^2)} \mathbf{q}_{xyz0} + \frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} + \mathbf{g}(1 - \mathbf{q}_w) \rangle}{\mathcal{L}(1 - \mathbf{q}_w)}(\mathbf{q} - \mathbf{g}) \\ &= \mathbf{p} - \frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathcal{L}(1 - \mathbf{q}_w)^2}(2\mathbf{q}_{xyz0} - (1 + \mathbf{q}_w)(\mathbf{q} - \mathbf{g})) + \frac{\langle \mathbf{p}, \mathbf{g} \rangle}{\mathcal{L}}(\mathbf{q} - \mathbf{g}) \\ &= \mathbf{p} - \frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathcal{L}(1 - \mathbf{q}_w^2)}(1 - \mathbf{q}_w)(\mathbf{q} + \mathbf{g}) + \mathbf{p}_w(\mathbf{q} - \mathbf{g}) \\ &= \mathbf{p} - \frac{\langle \mathbf{p}, \mathbf{q}_{xyz0} \rangle}{\mathcal{L}(1 + \mathbf{q}_w)}(\mathbf{q} + \mathbf{g}) + \mathbf{p}_w(\mathbf{q} - \mathbf{g}). \end{aligned}$$

References

1. Szirmay-Kalos, L.; Magdics, M. Adapting Game Engines to Curved Spaces. *Vis. Comput.* **2021**, *38*, 4383–4395. [CrossRef]
2. Mancinelli, C.; Puppo, E. Splines on manifolds: A survey. *Comput. Aided Geom. Des.* **2024**, *112*, 102349. [CrossRef]
3. Karcher, H. Riemannian center of mass and mollifier smoothing. *Commun. Pure Appl. Math.* **1977**, *30*, 509–541. [CrossRef]
4. Afsari, B. Means and Averaging on Riemannian Manifolds. Ph.D. Thesis, University of Maryland, College Park, MD, USA, 2009.
5. Gabriel, S.A.; Kajiya, J.T. Spline interpolation in curved manifolds. In *SIGGRAPH '85, Course Notes for "State of the Art in Image Synthesis"*; ACM: New York, NY, USA, 1985.
6. Noakes, L.; Heinzinger, G.; Paden, B. Cubic splines on curved spaces. *IMA J. Math. Control Inf.* **1989**, *6*, 465–473. [CrossRef]
7. Hofer, M.; Pottmann, H. Energy-minimizing splines in manifolds. In *ACM SIGGRAPH 2004 Papers*; ACM: New York, NY, USA, 2004; pp. 284–293. [CrossRef]
8. Camarinha, M.; Silva Leite, F.; Crouch, P.E. High-order splines on Riemannian manifolds. *Proc. Steklov Inst. Math.* **2023**, *321*, 158–178. [CrossRef]
9. Wallner, J. Geometric subdivision and multiscale transforms. In *Handbook of Variational Methods for Nonlinear Geometric Data*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 121–152. [CrossRef]
10. Mancinelli, C.; Nazzaro, G.; Pellacini, F.; Puppo, E. b/Surf: Interactive Bézier Splines on Surface Meshes. *IEEE Trans. Vis. Comput. Graph.* **2023**, *29*, 3419–3435. [CrossRef]
11. Crane, K.; Livesu, M.; Puppo, E.; Qin, Y. A survey of algorithms for geodesic paths and distances. *arXiv* **2020**, arXiv:2007.10430. [CrossRef]
12. Boehm, W.; Müller, A. On de Casteljau's algorithm. *Comput. Aided Geom. Des.* **1999**, *16*, 587–605. [CrossRef]
13. De Boor, C. *A Practical Guide to Splines*; Springer: Berlin/Heidelberg, Germany, 1978; Volume 27.
14. Barry, P.J.; Goldman, R.N. A recursive evaluation algorithm for a class of Catmull-Rom splines. *ACM SIGGRAPH Comput. Graph.* **1988**, *22*, 199–204. [CrossRef]
15. Catmull, E.; Rom, R. A class of local interpolating splines. In *Computer Aided Geometric Design*; Elsevier: Amsterdam, The Netherlands, 1974; pp. 317–326. [CrossRef]
16. Goldman, R. *Pyramid Algorithms: A Dynamic Programming Approach to Curves and Surfaces for Geometric Modeling*; Elsevier: Amsterdam, The Netherlands, 2002.
17. Ramshaw, L. Blossoms are polar forms. *Comput. Aided Geom. Des.* **1989**, *6*, 323–358. [CrossRef]
18. Shoemake, K. Animating rotation with quaternion curves. In Proceedings of the SIGGRAPH '85: 12th Annual Conference on Computer Graphics and Interactive Technique; ACM Press: New York, NY, USA, 1985; pp. 245–254. [CrossRef]
19. Dam, E.B.; Koch, M.; Lillholm, M. *Quaternions, Interpolation and Animation*; Technical Report DIKU-TR-98/5; Department of Computer Science, University of Copenhagen: Copenhagen, Denmark, 1998.
20. Kim, M.J.; Kim, M.S.; Shin, S.Y. A general construction scheme for unit quaternion curves with simple high order derivatives. In Proceedings of the SIGGRAPH '95: 22nd Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 6–11 August 1995; ACM Press: New York, NY, USA, 1995; pp. 369–376. [CrossRef]
21. Buss, S.R.; Fillmore, J.P. Spherical averages and applications to spherical splines and interpolation. *ACM Trans. Graph.* **2001**, *20*, 95–126. [CrossRef]
22. Popiel, T.; Noakes, L. C^2 spherical Bézier splines. *Comput. Aided Geom. Des.* **2006**, *23*, 261–275. [CrossRef]
23. Hanson, A.J. *Visualizing Quaternions*; Morgan Kaufmann: Burlington, MA, USA, 2006.
24. Geier, M. *Splines in Euclidean Space and Beyond*; Zenodo: Geneva, Switzerland, 2025. [CrossRef]
25. Lin, A.; Walker, M. CAGD techniques for differentiable manifolds. In *Algorithms for Approximation IV*; Levesley, J., Anderson, I., Mason, J., Eds.; Huddlersfield University: Huddersfield, UK, 2001; pp. 36–43.
26. Rockwood, A. Splossoms: Spherical Blossoms. In *Advances in Computer Graphics*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 200–209. [CrossRef]
27. Schaefer, S.; Goldman, R. Freeform curves on spheres of arbitrary dimension. In Proceedings of the Pacific Graphics, Macao, China, 12–14 October 2005; pp. 160–162.
28. Kochanek, D.H.; Bartels, R.H. Interpolating splines with local tension, continuity, and bias control. *ACM SIGGRAPH Comput. Graph.* **1984**, *18*, 33–41. [CrossRef]
29. Park, F.; Ravani, B. Bézier curves on Riemannian manifolds and Lie groups with kinematics applications. *J. Mech. Des.* **1995**, *117*, 36–40. [CrossRef]
30. Crouch, P.; Kun, G.; Leite, F.S. The De Casteljau Algorithm on Lie Groups and Spheres. *J. Dyn. Control Syst.* **1999**, *5*, 397–429. [CrossRef]
31. Popiel, T.; Noakes, L. Bézier curves and C^2 interpolation in Riemannian manifolds. *J. Approx. Theory* **2007**, *148*, 111–127. [CrossRef]
32. Nava-Yazdani, E.; Polthier, K. De Casteljau's algorithm on manifolds. *Comput. Aided Geom. Des.* **2013**, *30*, 722–732. [CrossRef]

33. Bogfjellmo, G.; Modin, K.; Verdier, O. A Numerical Algorithm for C^2-Splines on Symmetric Spaces. *SIAM J. Numer. Anal.* **2018**, *56*, 2623–2647. [CrossRef]
34. Hanik, M.; Nava-Yazdani, E.; von Tycowicz, C. De Casteljau's algorithm in geometric data analysis: Theory and application. *Comput. Aided Geom. Des.* **2024**, *110*, 102288. [CrossRef]
35. Ahanchaou, T.; Ikemakhen, A. Hyperbolic interpolatory geometric subdivision schemes. *J. Comput. Appl. Math.* **2022**, *399*, 113716. [CrossRef]
36. Berger, P.; Laier, A.; Velho, L. An image-space algorithm for immersive views in 3-manifolds and orbifolds. *Vis. Comput.* **2015**, *31*, 93–104. [CrossRef]
37. Velho, L.; Silva, V.d.; Novello, T. Immersive Visualization of the Classical Non-Euclidean Spaces using Real-Time Ray Tracing in VR. In Proceedings of the Graphics Interface, Toronto, ON, Canada, 28–29 May 2020; pp. 423–430. [CrossRef]
38. Hart, J.C. Sphere Tracing: Simple Robust Antialiased Rendering of Distance-Based Implicit Surfaces. In *SIGGRAPH 93 Course Notes: Modeling, Visualizing, and Animating Implicit Surfaces*; ACM Press: New York, NY, USA, 1993; pp. 1–11.
39. Weeks, J. Real-time rendering in curved spaces. *IEEE Comput. Graph. Appl.* **2002**, *22*, 90–99. [CrossRef]
40. Gunn, C. Advances in Metric-neutral Visualization. In Proceedings of the 2nd International Workshop on Computer Graphics, Computer Vision and Mathematics, GraVisMa, Brno, Czech Republic, 7–10 September 2010; pp. 17–26.
41. Martelli, B. An Introduction to Geometric Topology. *arXiv* **2016**, arXiv:1610.02592.
42. Loustau, B. Hyperbolic geometry. *arXiv* **2020**, arXiv:2003.11180. [PubMed]
43. Ghadami, R.; Rahebi, J.; Yayli, Y. Linear interpolation in Minkowski space. *Int. J. Pure Appl. Math.* **2012**, *77*, 469–484.
44. Overhauser, A.W. *Analytic Definition of Curves and Surfaces by Parabolic Blending*; Technical Report SL 68–40; Mathematical and Theoretical Sciences Department, Scientific Laboratory, Ford Motor Company: Dearborn, MI, USA, 1968. [CrossRef]
45. Blinn, J. *Jim Blinn's Corner: Notation, Notation, Notation*; Morgan Kaufmann Publishers Inc.: Burlington, MA, USA, 2002.
46. Novello, T.; da Silva, V.; Velho, L. Visualization of Nil, Sol, and $SL_2(R)$ geometries. *Comput. Graph.* **2020**, *91*, 219–231. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.