



Adapting Game Engines to Curved Spaces

László Szirmay-Kalos¹ · Milán Magdics¹

Accepted: 7 September 2021 / Published online: 11 October 2021
© The Author(s) 2021

Abstract

Curved spaces are very un-intuitive to our eyes trained on Euclidean geometry. Games provide an interesting way to explore these strange worlds. Games are written with the help of modeling tools and game engines based on Euclidean geometry. This paper addresses the problem of adapting 3D game engines to the rules of curved spaces. We consider the conversion of Euclidean objects, geometric calculations, transformation pipeline, lighting and physical simulation. Finally, we identify where existing game engines should be modified.

Keywords Hyperbolic geometry · Elliptic geometry · Transformations · Illumination

1 Introduction

Euclidean, elliptic and hyperbolic geometries share all but the parallel axiom, i.e. for a given line they postulate exactly one, none, and more than one non-intersecting line passing through a given point, respectively. The difference in the parallel axiom has significant consequences; thus, each of these geometries describes a different world. Games offer the experience of strange worlds; thus, the adaptation of game engines to non-Euclidean geometries provides an interesting way of exploring and understanding these geometries.

A game has to support the following main tasks:

- Loading of the modeled objects into the game world.
- Simulating game objects to determine their state including their translation and rotation in each frame.
- Animating game objects and the avatar's camera by applying the computed transformations to vertices.
- Rendering the game objects determining the visibility and the radiance of their surfaces and projecting them onto the screen.

These tasks are solved in game engines, e.g. Unity3D [4], implementing formulas assuming Euclidean geometry. The objective of this paper is to convert graphics engines developed for Euclidean geometry to virtual worlds defined by elliptic or hyperbolic geometry. (Initial concepts related to the transformations in elliptic geometry were discussed in our previous paper [25].) The contributions are:

- A general framework and simple formulas with proofs to set up the transformation matrices according to the rules of elliptic and hyperbolic geometries.
- A method of converting game objects and worlds from Euclidean to non-Euclidean geometries.
- Modification of the laws of light propagation, illumination, and dynamics for curved spaces.
- Review of the possibilities of existing game engine adaptation.

We assume unit curvature spaces, i.e. set the unit of length to the constant curvature of the space. However, it does not mean that the change of the curvature cannot be demonstrated since the same effect can be achieved by scaling Euclidean objects and locations before conversion.

The structure of the paper is as follows: Section 2 surveys the previous work. In Sect. 3, we summarize the *embedding space model* of Euclidean, elliptic, hyperbolic and projective geometries. Our contributions start with Sect. 4 that presents formulas to build up the transformation matrices of typical operations in the animation and rendering pipeline. Section 5 discusses how to convert objects obtained in the framework

✉ László Szirmay-Kalos
szirmay@iit.bme.hu

Milán Magdics
magdics@iit.bme.hu

¹ Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary

of Euclidean geometry into non-Euclidean spaces and introduces our solution based on the exponential map. Section 6 discusses the adaptation of illumination and physics. In Sect. 7, we examine the possibilities of game engine integration and show the shader programs of one particular solution. Finally, we demonstrate the results.

2 Previous work

Rendering non-Euclidean geometries is a topic in mathematics, cartography and art [9,14,21]. Visibility determination in 3D spaces can be based on ray tracing, i.e. image space rendering, or object space approaches. In ray tracing, only the ray definition and the intersection calculation should be altered according to the shortest path, i.e. the geodesics of the actual geometry [2,27]. If ray marching or its improved versions like, e.g. sphere tracing, is used instead, non-isotropic Thurston geometries [6,20] or even geometries with varying curvature can also be rendered [8]. Visualizing special and general relativity or black holes are good examples for these approaches [30,31]. Ray tracing can also take care of teleporting at fundamental domain boundaries [2].

Object space rendering algorithms can also be adapted to elliptic or hyperbolic geometry. Visibility computation is based on the “smaller than” relation of distances on a ray, but the exact distance values are not needed; thus, the space can be distorted before this operation if the distortion preserves lines and the order of distances along the line. The projection of the *Klein’s model* meets these requirements; thus, classical clipping and visibility algorithms are appropriate also for hyperbolic and elliptic geometries [10,28]. Other Thurston geometries can also be considered [13], but rays may become curved.

Modeling and animation of objects and the camera involve geometric calculations like the determination of distance, angle or direction, finding orthogonal directions, and the determination of the transformation matrices. In Euclidean geometry, translation makes a transformed line parallel with the original line, but this would be impossible in non-Euclidean geometry because of the curvature of the space.

This means that even the identification of the translation is a problem in non-Euclidean geometry. Rigid motions preserving a sphere belong to the rotation group $O(4)$, and rigid motions of the hyperbolic space are the Lorentz group $O(3, 1)$. Thus, subsets of these transformations must be re-interpreted as translations when the concepts of the classical rendering pipeline are extended for these geometries. Weeks [28] answered this problem only if the translation is done along the coordinate axes. Gunn [22] proposed to execute translation as two planar reflections. Hart et al [11], on the other hand, considered only infinitesimal translations. The transformation matrix of infinitesimal translations can be

generated using the Lie theory [17]. Infinitesimal translations may be enough in a fly-through animation or physical animation, but placing the objects in their initial state requires non-infinitesimal translations.

The first object space 3D graphics system visualizing spherical and hyperbolic spaces was *Geomview* [1], which inspired other systems like *jReality* [5]. The tiling of the hyperbolic plane was also exploited in 2D games like *HyperRogue* [12]. Note that virtual reality interactions pose additional challenges since the user’s real body parts in Euclidean space, the avatar’s body parts in non-Euclidean space together with their perceived view become inconsistent and require compensation [29]. In our paper, we do not consider this problem and assume that the avatar is represented by a virtual pin-hole camera.

3 The embedding space model of geometries

To examine different geometries analytically in a unified framework, we can take an outsider’s view [15,16]. It means that the real space is looked at from a space of one more dimension, i.e. we consider the 3D geometries as subsets of the 4D *embedding space*. The embedding space is associated with a *dot product* and four orthogonal basis vectors $\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l}$, and its elements are characterized by four-element row vectors $\mathbf{v} = (x, y, z, w)$ of the four coordinates.

For 3D Euclidean geometry or 3D elliptic geometry, the dot product in the 4D embedding space is the conventional Euclidean dot product

$$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle_E = x_1x_2 + y_1y_2 + z_1z_2 + w_1w_2. \quad (1)$$

However, hyperbolic geometry requires that the embedding space is of Minkowski type with the Lorentzian definition of the dot product

$$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle_L = x_1x_2 + y_1y_2 + z_1z_2 - w_1w_2. \quad (2)$$

To handle the different cases uniformly, we introduce *curvature sign* \mathcal{L} that is 1 in case of Euclidean and elliptic geometry, and -1 in hyperbolic geometry. With this, the general dot product is

$$\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = x_1x_2 + y_1y_2 + z_1z_2 + \mathcal{L}w_1w_2. \quad (3)$$

The basis vectors satisfy the condition of normalization:

$$\langle \mathbf{i}, \mathbf{i} \rangle = \langle \mathbf{j}, \mathbf{j} \rangle = \langle \mathbf{k}, \mathbf{k} \rangle = 1 \quad \text{and} \quad \langle \mathbf{l}, \mathbf{l} \rangle = \mathcal{L}, \quad (4)$$

and the condition of orthogonality:

$$\langle \mathbf{i}, \mathbf{j} \rangle = \langle \mathbf{j}, \mathbf{k} \rangle = \langle \mathbf{k}, \mathbf{l} \rangle = \langle \mathbf{l}, \mathbf{i} \rangle = 0. \quad (5)$$

Vectors starting at a point represent directions in which one can tangentially pass through that point, i.e. vectors are in the *tangent space* associated with the point.

3.1 Euclidean geometry

Points $\mathbf{p} = (p_x, p_y, p_z, p_w)$ of the Euclidean 3D space are identified by equation $p_w = 1$. As vectors are directions between two points, *vectors* \mathbf{v} of the Euclidean space have the property of $v_w = 0$. Different points share the same tangent space and thus the same set of possible vectors.

A *line* is a uniform motion of start \mathbf{p} and velocity \mathbf{v} :

$$\mathbf{r}_E(t) = \mathbf{p} + \mathbf{v}t, \tag{6}$$

or a combination of two points \mathbf{p} and \mathbf{q} :

$$\mathbf{r}_E(t) = \mathbf{p}(1 - t) + \mathbf{q}t. \tag{7}$$

The *plane* is also a linear structure but unlike the line, it is two-dimensional and thus is spanned by two vectors \mathbf{a} and \mathbf{b} :

$$\mathbf{r}_E(t_1, t_2) = \mathbf{p} + \mathbf{a}t_1 + \mathbf{b}t_2. \tag{8}$$

A plane is also the union of all lines passing through \mathbf{p} and having a direction vector that is the combination of the two spanning vectors \mathbf{a} and \mathbf{b} .

3.2 Elliptic geometry

Point \mathbf{p} in the 4D embedding space belongs to the 3D elliptic geometry if its coordinates satisfy the equation of the unit hyper-sphere:

$$p_x^2 + p_y^2 + p_z^2 + p_w^2 = 1 \text{ or } \langle \mathbf{p}, \mathbf{p} \rangle_E = 1. \tag{9}$$

Points of the elliptic geometry are *diameters* of the unit hyper-sphere; thus, antipodal points (p_x, p_y, p_z, p_w) and $(-p_x, -p_y, -p_z, -p_w)$ are considered to be the same. Treating points as diameters is needed to keep the validity of the axiom, stating that “two distinct points unambiguously define a line” in elliptic geometry as well. Geodesics on the sphere are the main circles, which would intersect each other twice. However, declaring the antipodal points identical, two lines intersect each other in a single point. Elliptic space is topologically equivalent to the projective space, but it has the metric of the sphere.

Vectors \mathbf{v} are directions that must be in the 3D tangent hyperplane called the tangent space of the hyper-sphere at their start. Position vectors of points on an arbitrary dimensional sphere centered in the origin are orthogonal to the

tangent space; thus, vectors should satisfy

$$\langle \mathbf{p}, \mathbf{v} \rangle_E = 0. \tag{10}$$

The *line* is the unit speed uniform motion starting at \mathbf{p} and of initial unit direction vector \mathbf{v} :

$$\mathbf{r}_S(t) = \mathbf{p} \cos(t) + \mathbf{v} \sin(t). \tag{11}$$

Distance d between two points is the time needed to travel from one to the other with unit speed uniform motion:

$$\mathbf{q} = \mathbf{p} \cos(d) + \mathbf{v} \sin(d) \implies d = \cos^{-1}(|\langle \mathbf{p}, \mathbf{q} \rangle_E|). \tag{12}$$

The absolute value makes the distances to the two equivalent antipodal points equal.

The line can also be expressed as the spherical combination of two points \mathbf{p} and \mathbf{q} being at distance d [24]:

$$\mathbf{r}_S(t) = \mathbf{p} \frac{\sin((1-t)d)}{\sin(d)} + \mathbf{q} \frac{\sin(td)}{\sin(d)}. \tag{13}$$

Similarly to Euclidean geometry, the *plane* is defined by a point \mathbf{p} and two spanning vectors \mathbf{a} and \mathbf{b} in the tangent space of point \mathbf{p} . The set of points of the plane is the union of all lines passing through \mathbf{p} and having a direction vector that is a combination of the two spanning vectors \mathbf{a} and \mathbf{b} .

3.3 Hyperbolic geometry

Points \mathbf{p} of the hyperbolic space satisfy equation

$$p_x^2 + p_y^2 + p_z^2 - p_w^2 = -1, \text{ and } p_w > 0, \tag{14}$$

which could be interpreted as the upper part of a two-sheet hyperboloid if the embedding space were Euclidean. However, hyperbolic geometry is embedded in Minkowski space equipped with the Lorentzian dot product; thus, the equation looks identical to the equation of a hyper-sphere of imaginary radius $\sqrt{-1}$:

$$\langle \mathbf{p}, \mathbf{p} \rangle_L = -1.$$

Again, *vectors* must be in the tangent space of their starting point, defined by equation $\langle \mathbf{p}, \mathbf{v} \rangle_L = 0$, which is the consequence that the hyperbolic space is also a hyper-sphere similarly to elliptic geometry, but in Minkowski embedding space.

In hyperbolic geometry, the *line*, i.e. the unit speed uniform motion, starting at \mathbf{p} and of initial direction \mathbf{v} is

$$\mathbf{r}_H(t) = \mathbf{p} \cosh(t) + \mathbf{v} \sinh(t). \tag{15}$$

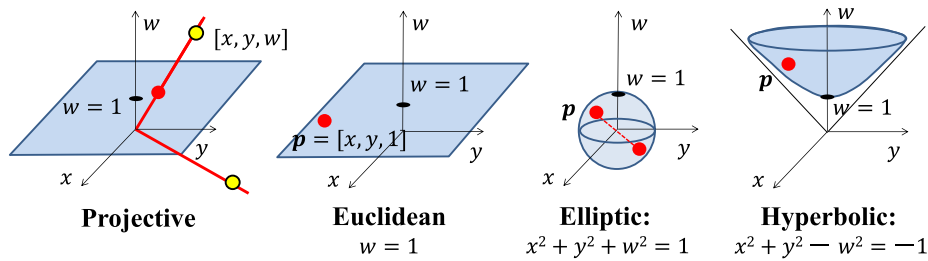


Fig. 1 Embedding the 2D projective, Euclidean, elliptic and hyperbolic geometries in the 3D world where a point has coordinates (x, y, w) . Points of the Euclidean geometry are on a plane of $w = 1$, points of the elliptic geometry are on the sphere of equation $x^2 + y^2 + w^2 = 1$, and points of the hyperbolic geometry belong to the upper part of the two sheet hyperboloid of equation $x^2 + y^2 - w^2 = -1$; thus, they are

2D manifolds of the 3D embedding space. 2D projective geometry is similar to the Euclidean geometry but establishes a different correspondence between 3D embedding space and 2D projective space, which is responsible for dimensional reduction. In projective geometry, lines of $(\lambda x, \lambda y, \lambda w)$ correspond to points in projective geometry.

The *distance* d between two points is the time needed to travel from one to the other with unit speed uniform motion:

$$\mathbf{q} = \mathbf{p} \cosh(d) + \mathbf{v} \sinh(d) \implies d = \cosh^{-1}(-\langle \mathbf{p}, \mathbf{q} \rangle_L). \tag{16}$$

The *line* is again the combination of two points \mathbf{p} and \mathbf{q} being at distance d , but in hyperbolic geometry weights are defined by the hyperbolic sine function [7]:

$$\mathbf{r}_H(t) = \mathbf{p} \frac{\sinh((1-t)d)}{\sinh(d)} + \mathbf{q} \frac{\sinh(td)}{\sinh(d)}. \tag{17}$$

The former definition of the plane remains valid, i.e. the *plane* is the set of lines passing through \mathbf{p} and having a direction vector that is a combination of the two spanning vectors \mathbf{a} and \mathbf{b} being in the tangent space of \mathbf{p} .

3.4 Projective geometry

In the geometries discussed so far, their 3D subsets of the 4D embedding space are identified by equations. When embedding projective geometry, a different strategy can be applied to reduce dimension. Namely, elements of the 3D projective geometry are associated with higher-dimensional structures in the embedding space.

Points of the 3D projective space are represented by lines crossing the origin in embedding space, i.e. we consider all 4D points $(\lambda x, \lambda y, \lambda z, \lambda w)$ with arbitrary $\lambda \neq 0$ equivalent.

A *line* of the projective space is a plane crossing the origin in the embedding space.

Embedding space lines would intersect an origin centered sphere at two antipodal points, making projective geometry topologically similar to elliptic geometry.

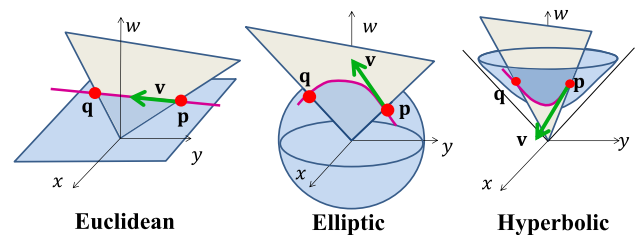


Fig. 2 A line defined by points \mathbf{p} and \mathbf{q} is on the embedding plane defined by \mathbf{p}, \mathbf{q} and the embedding origin in all geometries. Note that lines of different geometries look the same from embedding origin; thus, points of the line in one geometry can be projected onto the line of another geometry

3.5 Common ground

Figure 1 shows the 2D projective, Euclidean, elliptic and hyperbolic geometries, all embedded in the 3D embedding space. Note that this figure is for analogy only since we investigate 3D geometries embedded in the 4D embedding space, which would be hard to visualize.

The point of embedding coordinates $(0, 0, 0, 1)$ is included in all three geometries, which is called the *geometry origin* and denoted by \mathbf{g} . This point is the origin of the world coordinate system in all geometries. The point of coordinates $(0, 0, 0, 0)$, on the other hand, is called the *embedding origin*. This point is outside of the considered geometries.

Note that in all geometries, the points of a line are expressed as a combination of points \mathbf{p} and \mathbf{q} ; thus, the line is in the plane spanned by these two vectors (Fig. 2). Additionally, lines are in their respective geometries; thus, they are the intersections of their respective geometries and the plane defined by points \mathbf{p}, \mathbf{q} and the embedding origin. It means that lines in the three geometries are projectively equivalent assuming the embedding origin to be the center of projection. When visibility is evaluated, they can be projected onto the $w = 1$ plane representing Euclidean space, so this task can be solved uniformly here.

We have seen that a plane can be interpreted in all three geometries as the set of lines passing through \mathbf{p} and having a direction vector that is a combination of the two spanning vectors \mathbf{a} and \mathbf{b} being in the tangent space of \mathbf{p} . Based on this definition, we can establish an equation for the plane that is valid in all geometries. Let us consider a line of the plane

$$\mathbf{r}(\tau_1, \tau_2) = \mathbf{p} + \mathbf{v}(\tau_1)\tau_2$$

where $\mathbf{v}(\tau_1) = \tau_1\mathbf{a} + (1 - \tau_1)\mathbf{b}$ is the combination of \mathbf{a} and \mathbf{b} .

Suppose we have a normal vector \mathbf{n} in the embedding space that is perpendicular to vectors \mathbf{a} , \mathbf{b} and \mathbf{p} . The scalar product of perpendicular vectors is zero; thus, multiplying \mathbf{r} with normal \mathbf{n} , we get

$$\langle \mathbf{r}, \mathbf{n} \rangle = \langle \mathbf{p}, \mathbf{n} \rangle + \langle \mathbf{a}, \mathbf{n} \rangle \tau_1 \tau_2 + \langle \mathbf{b}, \mathbf{n} \rangle (1 - \tau_1) \tau_2 = 0.$$

Thus, $\langle \mathbf{r}, \mathbf{n} \rangle = 0$ implicit equation describes a plane in all geometries. Note that this equation can be satisfied also by those embedding space points that are not part of the given geometry. So, the points of the plane are the intersection of the solution of $\langle \mathbf{r}, \mathbf{n} \rangle = 0$ and the solution of the equation of points in a particular geometry.

To establish the equation of a plane crossing point \mathbf{p} and spanned by vectors \mathbf{a} and \mathbf{b} being in the tangent space of \mathbf{p} , we need to find a 4D vector \mathbf{n} that is perpendicular to these three vectors \mathbf{a} , \mathbf{b} and \mathbf{p} . The same problem needs to be solved when we are looking for a vector that is perpendicular to two other vectors \mathbf{a} and \mathbf{b} that belong to the tangent space of point \mathbf{p} .

For this, we can use a 4D analog of the 3D cross product [3]:

$$\mathbf{n} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} & \mathbf{l} \\ p_x & p_y & p_z & \mathcal{L}p_w \\ a_x & a_y & a_z & \mathcal{L}a_w \\ b_x & b_y & b_z & \mathcal{L}b_w \end{vmatrix}. \tag{18}$$

To prove that vector \mathbf{n} is indeed perpendicular to the three operands, let us consider the dot product of \mathbf{n} and an arbitrary vector \mathbf{v} :

$$\langle \mathbf{n}, \mathbf{v} \rangle = \begin{vmatrix} v_x & v_y & v_z & \mathcal{L}v_w \\ p_x & p_y & p_z & \mathcal{L}p_w \\ a_x & a_y & a_z & \mathcal{L}a_w \\ b_x & b_y & b_z & \mathcal{L}b_w \end{vmatrix}.$$

If \mathbf{v} were equal to any of \mathbf{p} , \mathbf{a} , \mathbf{b} (or their linear combination), then according to the properties of determinants, this determinant would be zero, which indicates orthogonality since two vectors are orthogonal if their dot product is zero.

4 Transformations and isometries

Our goal is to express transformations of the non-Euclidean space as 4×4 matrix multiplications. These matrix multiplications must map the geometry onto itself, i.e. the image of a point of the geometry should also belong to the same geometry. Isometric transformations preserve the dot product and consequently distances and angles. The points of the non-Euclidean spaces are also identified by a dot product, namely, point \mathbf{p} is in the geometry if $\langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L}$. It means that the preservation of the scalar product automatically guarantees that points of the geometry are mapped to the same space. It is easy to show that the scalar product is preserved if the transformed basis vectors also form an orthogonal and normalized frame, i.e. to satisfy Eqs. 4 and 5.

4.1 Translation

In Euclidean geometry, translating an object means adding the translation vector to the position vector of the vertices. However, in non-Euclidean geometry, the allowed vectors depend on the point; thus, this interpretation cannot be used anymore. Points before and after the transformation should satisfy $\langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L}$, so they are of the same “distance” from the embedding origin, i.e. elliptic and hyperbolic geometries have a spherical structure, where mathematically only rotations and reflections exist. However, location is associated with translation in our Euclidean mind, so we should separate a class of rotations and identify them as translations.

To identify the appropriate class of isometries that can be called translations in non-Euclidean geometry, we consider its intuitive properties. The translation

- is an isometry preserving the dot product and the orientation,
- is defined by point $\mathbf{q} = (q_x, q_y, q_z, q_w)$ to which the geometry origin $\mathbf{g} = (0, 0, 0, 1)$ is translated, and
- should execute *parallel transport*, i.e. keep the original and transformed directions parallel as much as possible.

Concerning the requirement of isometry and orientation preservation, we are left with even number of reflections. In order to distinguish non-Euclidean translation from rotation, we can say that translations modify the geometry origin and keep the directions as parallel as possible while rotations preserve the geometry origin. We follow the idea of [10] that expresses translation as appropriately chosen reflections and derive simple formula with proofs for the game engine implementation. Reflections can be executed on planes and lines crossing the embedding origin, i.e. on vectors. We use here the latter option since it allows a simpler derivation of the same matrix. The reflection of point \mathbf{p} on vector \mathbf{m} is:

$$\mathbf{p}' = 2 \frac{\langle \mathbf{p}, \mathbf{m} \rangle}{\langle \mathbf{m}, \mathbf{m} \rangle} \mathbf{m} - \mathbf{p}. \tag{19}$$

To prove that it is indeed a reflection, we note that \mathbf{p}' is in the plane spanned by \mathbf{m} and \mathbf{p} and show that the points of the geometry are mapped to the same geometry and the distance of \mathbf{m} and \mathbf{p} equals the distance of \mathbf{m} and \mathbf{p}' . Point \mathbf{p} is in the geometry if $\langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L}$, so is the transformed point:

$$\begin{aligned} \langle \mathbf{p}', \mathbf{p}' \rangle &= 4 \frac{\langle \mathbf{p}, \mathbf{m} \rangle^2}{\langle \mathbf{m}, \mathbf{m} \rangle^2} \langle \mathbf{m}, \mathbf{m} \rangle - 4 \frac{\langle \mathbf{p}, \mathbf{m} \rangle}{\langle \mathbf{m}, \mathbf{m} \rangle} \langle \mathbf{p}, \mathbf{m} \rangle + \langle \mathbf{p}, \mathbf{p} \rangle \\ &= \langle \mathbf{p}, \mathbf{p} \rangle = \mathcal{L}. \end{aligned} \tag{20}$$

As the distance is derived from the dot product, the equality of dot products is shown:

$$\langle \mathbf{p}', \mathbf{m} \rangle = 2 \frac{\langle \mathbf{p}, \mathbf{m} \rangle}{\langle \mathbf{m}, \mathbf{m} \rangle} \langle \mathbf{m}, \mathbf{m} \rangle - \langle \mathbf{p}, \mathbf{m} \rangle = \langle \mathbf{p}, \mathbf{m} \rangle. \tag{21}$$

To guarantee that translation executes parallel transport, we use two reflection vectors that are in the plane spanned by geometry origin \mathbf{g} and target point \mathbf{q} and expect that after two reflections the geometry origin is mapped to \mathbf{q} . We could use any two vectors meeting these requirements. A convenient choice is to select the first reflector vector to be the geometry origin $\mathbf{m}_1 = \mathbf{g}$ and the second reflector to be the halfway vector between the geometry origin and target point \mathbf{q} , i.e. $\mathbf{m}_2 = \mathbf{g} + \mathbf{q}$. An arbitrary point \mathbf{p} is mapped by the first reflection as

$$\mathbf{p}' = 2 \frac{\langle \mathbf{p}, \mathbf{g} \rangle}{\langle \mathbf{g}, \mathbf{g} \rangle} \mathbf{g} - \mathbf{p} = 2p_w \mathbf{g} - \mathbf{p}. \tag{22}$$

The result of the second reflection is:

$$\mathbf{p}'' = 2 \frac{\langle \mathbf{p}', \mathbf{g} + \mathbf{q} \rangle}{\langle \mathbf{g} + \mathbf{q}, \mathbf{g} + \mathbf{q} \rangle} (\mathbf{g} + \mathbf{q}) - \mathbf{p}'. \tag{23}$$

Substituting Eq. 22 and exploiting that \mathbf{q} is in the geometry, i.e. $\langle \mathbf{q}, \mathbf{q} \rangle = \mathcal{L}$, we obtain

$$\mathbf{p}'' = \mathbf{p} + 2p_w \mathbf{q} - \frac{\mathcal{L} \langle \mathbf{p}, \mathbf{q} \rangle + p_w}{q_w + 1} (\mathbf{q} + \mathbf{g}). \tag{24}$$

Evaluating this formula for the four basis vectors, the rows of the transformation matrix of the translation to point \mathbf{q} can be expressed:

$$\mathbf{T}(\mathbf{q}) = \begin{bmatrix} 1 - \mathcal{L} \frac{q_x q_x}{1+q_w} & -\mathcal{L} \frac{q_x q_y}{1+q_w} & -\mathcal{L} \frac{q_x q_z}{1+q_w} & -\mathcal{L} q_x \\ -\mathcal{L} \frac{q_y q_x}{1+q_w} & 1 - \mathcal{L} \frac{q_y q_y}{1+q_w} & -\mathcal{L} \frac{q_y q_z}{1+q_w} & -\mathcal{L} q_y \\ -\mathcal{L} \frac{q_z q_x}{1+q_w} & -\mathcal{L} \frac{q_z q_y}{1+q_w} & 1 - \mathcal{L} \frac{q_z q_z}{1+q_w} & -\mathcal{L} q_z \\ q_x & q_y & q_z & q_w \end{bmatrix}. \tag{25}$$

The last row of the matrix is the target point \mathbf{q} ; thus, this matrix indeed translates the geometry origin to this point. This matrix generates an isometry since its row vectors are orthogonal to each other and of unit length. We can see that the matrix executes parallel transport since vector $(q_x, q_y, q_z, 0)$ pointing into the direction of \mathbf{q} from the geometry origin remains parallel with the geodesic between the geometry origin and target point \mathbf{q} during the translation.

4.2 Rotation

Rotation is also an isometry, but unlike translation, it keeps the geometry origin in $(0, 0, 0, 1)$. From this, the fourth row of the transformation matrix should also be $(0, 0, 0, 1)$. In case of isometries $\langle \mathbf{l}', \mathbf{i}' \rangle = \langle \mathbf{l}', \mathbf{j}' \rangle = \langle \mathbf{l}', \mathbf{k}' \rangle = 0$, which means that the fourth coordinates of \mathbf{i}' , \mathbf{j}' , \mathbf{k}' basis vectors are zero. So, the structure of the transformation matrix in non-Euclidean geometry is identical to that of the Euclidean geometry.

4.3 The view matrix

The camera is defined by eye position \mathbf{e} and three orthogonal unit vectors in the tangent space of the eye, right direction \mathbf{i}' , up direction \mathbf{j}' , and negative view direction \mathbf{k}' . The orthogonality can be enforced by Eq. 18. Then, the view matrix can be expressed as:

$$\mathbf{V} = \begin{bmatrix} i'_x & j'_x & k'_x & \mathcal{L}e_x, \\ i'_y & j'_y & k'_y & \mathcal{L}e_y, \\ i'_z & j'_z & k'_z & \mathcal{L}e_z, \\ \mathcal{L}i'_w & \mathcal{L}j'_w & \mathcal{L}k'_w & e_w \end{bmatrix} \tag{26}$$

To prove that this matrix meets the requirements of the view matrix, we look at the transformation of the eye position \mathbf{e} and the camera basis vectors. The eye position is transformed as:

$$\begin{aligned} \mathbf{e} \cdot \mathbf{V} &= (\langle \mathbf{e}, \mathbf{i}' \rangle, \langle \mathbf{e}, \mathbf{j}' \rangle, \langle \mathbf{e}, \mathbf{k}' \rangle, \langle \mathbf{e}, \mathcal{L}\mathbf{e} \rangle) \\ &= (0, 0, 0, \mathcal{L}^2) = (0, 0, 0, 1) = \mathbf{g}. \end{aligned} \tag{27}$$

since \mathbf{i}' , \mathbf{j}' , \mathbf{k}' are in the tangent space of eye position \mathbf{e} , thus their scalar product with the eye position is zero, and the eye position is in the curved space identified by $\langle \mathbf{e}, \mathbf{e} \rangle = \mathcal{L}$.

The transformation of the right direction \mathbf{i}' is:

$$\mathbf{i}' \cdot \mathbf{V} = (\langle \mathbf{i}', \mathbf{i}' \rangle, \langle \mathbf{i}', \mathbf{j}' \rangle, \langle \mathbf{i}', \mathbf{k}' \rangle, \langle \mathbf{i}', \mathcal{L}\mathbf{e} \rangle) = (1, 0, 0, 0) = \mathbf{i}.$$

The transformation of view direction and up direction can be examined in the same way.

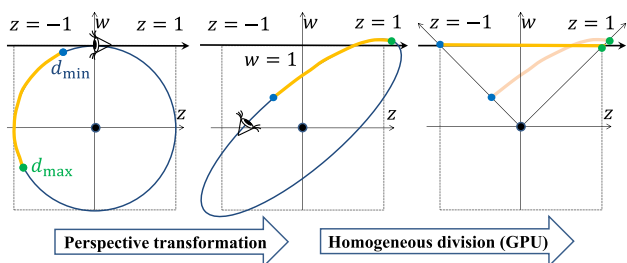


Fig. 3 Perspective transformation maps the hyper-sphere to hyper-ellipsoid preparing the object for GPU clipping and projection, which interprets embedding coordinates as homogeneous coordinates

4.4 The perspective transformation matrix

After the view transformation, the camera is at the geometry origin, looks at the $-z$ direction, and its right direction is axis x and its up direction is axis y . Using OpenGL, the GPU assumes that the vertex shader outputs the point in homogeneous coordinates and the viewing rays are parallel with axis z . A point is inside the frustum if inequalities $-w \leq x, y, z \leq w$ are satisfied. Thus, the perspective transformation should map the selected frustum to the domain defined by the clipping inequalities.

With a linear transformation of the 4D embedding space, the 3D hyper-sphere of elliptic and hyperbolic geometries is transformed to a 3D hyper-ellipsoid (Fig. 3). The frustum is a pyramid defined by inequalities $s_x z \leq x \leq -s_x z$ and $s_y z \leq y \leq -s_y z$, where s_x and s_y depend on the field of view. The pyramid is truncated by minimum distance d_{min} and by maximum distance d_{max} on the optical axis. In Euclidean geometry, these entry and exit points on the optical axis are $(0, 0, -d_{min}, 1)$ and $(0, 0, -d_{max}, 1)$, respectively. In elliptic geometry, the corresponding points are $(0, 0, -\sin(d_{min}), \cos(d_{min}))$ and $(0, 0, -\sin(d_{max}), \cos(d_{max}))$. In hyperbolic geometry, these points are at $(0, 0, -\sinh(d_{min}), \cosh(d_{min}))$ and $(0, 0, -\sinh(d_{max}), \cosh(d_{max}))$. The eye position should be mapped to the ideal point $(0, 0, \lambda, 0)$ of axis z , the entry point to the front clipping plane defined by $-w' = z'$ and the exit point to the back clipping plane of equation $w' = z'$. From these, the perspective transformation is:

$$\mathbf{P} = \begin{bmatrix} 1/s_x & 0 & 0 & 0, \\ 0 & 1/s_y & 0 & 0, \\ 0 & 0 & \alpha & -1, \\ 0 & 0 & \beta & 0 \end{bmatrix} \tag{28}$$

In Euclidean geometry, the α and β parameters are:

$$\alpha_E = -\frac{d_{min} + d_{max}}{d_{max} - d_{min}}, \quad \beta_E = -\frac{2d_{min}d_{max}}{d_{max} - d_{min}}.$$

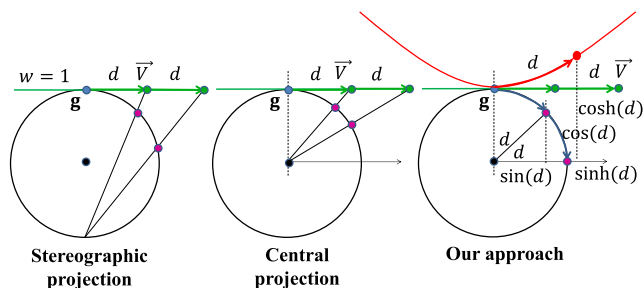


Fig. 4 Transporting objects from Euclidean to curved spaces. Stereographic projection preserves angles and circles, central projection lines, but both of them strongly distort distances. Our selected approach is the exponential map of the geometry origin [26], which preserves direction V and distance d from geometry origin g

In hyperbolic geometry, a similar result can be obtained:

$$\alpha_H = -\frac{\sinh(d_{min} + d_{max})}{\sinh(d_{max} - d_{min})}, \quad \beta_H = -\frac{2 \sinh(d_{min}) \sinh(d_{max})}{\sinh(d_{max} - d_{min})}.$$

In elliptic geometry, the hyperbolic sine is replaced by the normal sine function:

$$\alpha_S = -\frac{\sin(d_{min} + d_{max})}{\sin(d_{max} - d_{min})}, \quad \beta_S = -\frac{2 \sin(d_{min}) \sin(d_{max})}{\sin(d_{max} - d_{min})}.$$

5 Porting objects from Euclidean to non-Euclidean geometry

The virtual world is usually created with modeling tools following the rules of Euclidean geometry, and outputting vertices in Cartesian coordinates. The models and points need to be “transported” to the curved space with minimal distortions. We assume that our objects are triangle meshes. So, we need to consider only the porting of points representing vertices and the vectors, e.g. normals, associated with them.

As the elliptic and hyperbolic geometries have nonzero curvature, any correspondence with Euclidean geometry necessarily introduces distortions, such as change of distance, angle, or type of geometric primitives. There are many possibilities to establish a correspondence between Euclidean and curved spaces (Fig. 4). As in games, objects are typically defined in modeling space, i.e. close to the origin, and transformed to the actual position by translations, we prefer a mapping where the distortion diminishes close to the geometry origin. Stereographic projection, also called the Beltrami–Poincaré model in hyperbolic geometry, preserves angles and circles, but the distortion of distances grows rapidly far from the geometry origin. Central projection, also called the Beltrami–Klein model in hyperbolic geometry, preserves lines, but distorts both angles and distances even stronger than stereographic projection.

We need a projection that introduces less distortions and looks more natural. Instead of preserving either lines or angles, we wish to have a compromise which is acceptable for appropriately tessellated meshes. A mapping meeting this requirement is based on the recognition that it is worth preserving the distance and the direction of the point from the geometry origin. Such parametrization is called the *exponential map* of the geometry origin in differential geometry. If a point has Cartesian coordinates $\mathbf{P} = [X, Y, Z]$ in Euclidean geometry, it means that the point is at distance $d = \sqrt{X^2 + Y^2 + Z^2}$ from the geometry origin and its direction is defined by unit vector \mathbf{P}/d . The natural pair of this Euclidean point in elliptic space is the point

$$\mathbf{p} = \mathcal{P}(\mathbf{P}) = (\mathbf{P}/d \sin(d), \cos(d)) \quad (29)$$

that is also in direction \mathbf{P}/d and at the same distance d from the geometry origin. Replacing the sine and cosine functions with their hyperbolic counterparts, we get the formula valid in hyperbolic geometry. Note that the tangent of all three geometries is identical at the geometry origin; thus, this direction can be ported from one to the other without any modification.

As sine and cosine are periodic functions, this process will map multiple points onto the same spherical points. If the order of objects needs to be preserved, scaling can make sure that only a single period is covered.

5.1 Porting vectors

Let us consider vector $(\mathbf{V}, 0)$ in Euclidean space starting at point $(\mathbf{P}, 1)$. For example, this vector can be the shading normal of the surface at \mathbf{P} . If point $(\mathbf{P}, 1)$ were the geometry origin, then the Euclidean, elliptic, or hyperbolic space vectors would be the same. The vector should follow the point with parallel transport, i.e. with minimum change, which is provided by the developed matrix of translation (Eq. 25). This means that vectors can be transported from Euclidean to curved space by translating them to their start $\mathcal{P}(\mathbf{P})$:

$$\mathbf{v} = (\mathbf{V}, 0) \cdot \mathbf{T}(\mathcal{P}(\mathbf{P})). \quad (30)$$

6 Physical simulation

Physical simulation calculates the interaction of two meeting objects or the effect of a field on an object. Laws describing only local properties, i.e. depending on the point and directions in the tangent space of the point, are invariant to the chosen geometry with the exception of the calculation of angles between directions, which can be obtained as the inverse cosine of the geometry dependent dot product. The

underlying geometry can change physical laws via the calculation of directions and their angles, and the dependence of fields on distances.

Distance d and direction \mathbf{v} of source \mathbf{q} from affected point \mathbf{p} are expressed differently in the three geometries:

$$\begin{aligned} d_E &= \sqrt{\langle \mathbf{p} - \mathbf{q}, \mathbf{p} - \mathbf{q} \rangle_E}, & \mathbf{v}_E &= \frac{\mathbf{q} - \mathbf{p}}{d_E}, \\ d_S &= \cos^{-1}(|\langle \mathbf{p}, \mathbf{q} \rangle_E|), & \mathbf{v}_S &= \frac{\mathbf{q} - \mathbf{p} \cos(d_S)}{\sin(d_S)}, \\ d_H &= \cosh^{-1}(-\langle \mathbf{p}, \mathbf{q} \rangle_L), & \mathbf{v}_H &= \frac{\mathbf{q} - \mathbf{p} \cosh(d_H)}{\sinh(d_H)}. \end{aligned} \quad (31)$$

As we can see in the right of Fig. 4, points being at distance d from the geometry origin are on a circle in 2D or on a sphere in 3D, where the radii are d , $\sin(d)$ and $\sinh(d)$ in Euclidean, elliptic and hyperbolic geometries, respectively. Thus, the strengths of a field generated by a point source at distance d in 3D Euclidean, elliptic and hyperbolic geometries are inversely proportional to the following *attenuation* factors:

$$a_E(d) = d^2, \quad a_S(d) = \sin^2(d), \quad a_H(d) = \sinh^2(d).$$

Attenuation factors can also be expressed from the two points \mathbf{q} and \mathbf{p} :

$$a_{S,H}(\mathbf{q}, \mathbf{p}) = |1 - \langle \mathbf{q}, \mathbf{p} \rangle|^2. \quad (32)$$

6.1 Illumination

Local light reflection depends on local properties associated with angles of directions starting at this point. The calculation of the reflected radiance from the illumination direction, surface normal, view direction, the BRDF and the incident radiance is the same in all geometries with the exception of the angle calculation. Euclidean and elliptic geometries require the 4D Euclidean dot product, hyperbolic geometry and the Lorentzian product. In general, the Riemannian metric assigns a bi-linear dot product for each point [19]. Unlike light reflection, *ambient occlusion* depends on the curvature, so it must be re-calculated according to the rules of the geometry.

The radiance of a point light source at distance d falls with attenuation $a(d)$ differently in the three geometries. Note that there is no directional light source in non-Euclidean spaces.

In the case of area light sources or indirect illumination computation, the source surface is decomposed to infinitesimal areas and their contributions are integrated. A light source of infinitesimal area ΔA and isotropic radiance L^e emits power

$$\Delta \Phi = L^e \Delta A \cos(\theta) \Delta \omega \quad (33)$$

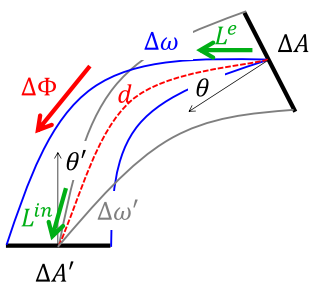


Fig. 5 Radiance remains constant along a geodesic: the incident radiance L^{in} at the receiver surface $\Delta A'$ is equal to the emitted radiance L^e of the source surface ΔA

in small solid angle $\Delta\omega$, where θ is the angle between the direction of emission and the surface normal.

Suppose that in this solid angle $\Delta\omega$ a light receiver surface is visible at distance d (Fig. 5). The area of the receiver surface is $\Delta A'$, the angle between the illumination direction and the surface normal is θ' , and the solid angle in which source ΔA is visible from the receiver surface is $\Delta\omega'$.

The relationship of the distance and the solid angle is a non-local property of the geometry. The solid angles are determined by the projected areas:

$$\Delta\omega = \frac{\Delta A' \cos(\theta')}{a(d)}, \quad \Delta\omega' = \frac{\Delta A \cos(\theta)}{a(d)}. \tag{34}$$

The incident radiance of the receiver surface is:

$$L^{in} = \frac{\Delta\Phi}{\Delta A' \cos(\theta') \Delta\omega'} = L^e \frac{\Delta A \cos(\theta) \Delta\omega}{\Delta A' \cos(\theta') \Delta\omega'} = L^e. \tag{35}$$

This means that the two attenuation factors cancel out, making the radiance of a light beam originating at a surface constant along the ray also in non-Euclidean geometries, which means that the adaptation of the indirect illumination calculation requires only the modification of the angle computation. For occlusion calculation, the curved path between the source and the receiver can be processed similarly to the visibility from the camera, having placed the virtual eye position in the light source.

6.2 Animation

In games, animation can be specified by different techniques. In key-frame, script, motion-capture, spline, etc. animation, the motion is prescribed typically in Euclidean space and can be modulated with the user controlled speed, environment properties like terrain elevation, and collision detection. Terrain elevation and collision detection can be solved in Euclidean space, resulting in the point of interaction, which can be converted to non-Euclidean space. Based on the calculated position, translation matrices should be altered according to the formulas of Section 4.

In dynamic simulation, however, computed forces update velocity and position vectors. The magnitude and direction of the force due to a field can depend on the distance and the position of the source. Direction and distance are computed with Eq. 31. Physical forces can depend only on the position and the velocity. Thus, the resulting force acting upon a point is automatically in the tangent space of the point, so is the velocity which is the time integral of acceleration caused by the force. In numerical simulation, it can happen that points do not fit into the geometry and vectors into the tangent space because of numerical errors. This should be solved by projecting points \mathbf{p} and vectors \mathbf{v} back to the geometry after updating them:

$$\mathbf{p}^* = \frac{\mathbf{P}}{\sqrt{|\langle \mathbf{p}, \mathbf{p} \rangle|}}, \quad \mathbf{v}^* = \mathbf{v} - \mathcal{L}(\mathbf{p}, \mathbf{v})\mathbf{p}. \tag{36}$$

If the affected object is small with respect to the curvature, then the local space is approximately Euclidean, so we can use the rigid body dynamics of Euclidean geometry, which typically decomposes the motion to a translational motion of the center of mass and a rotational motion around the center of mass, and the linear and angular momenta are updated according to the total force and torque taking into account the total mass and rotational inertia.

For large objects, rigid body dynamics should be adapted to the curved space. In Euclidean space, the *center of mass* of a system of points $\mathbf{q}_1, \dots, \mathbf{q}_n$ with associated masses m_1, \dots, m_n is defined as:

$$\mathbf{c} = \frac{\sum_i m_i \mathbf{q}_i}{\sum_i m_i}, \tag{37}$$

or alternatively, as the point where the following function $I_E(\mathbf{c})$ takes its minimum:

$$I_E(\mathbf{c}) = \sum_i d^2(\mathbf{q}_i, \mathbf{c})m_i = \sum_i a_E(\mathbf{q}_i, \mathbf{c})m_i. \tag{38}$$

Salvai [23] has shown that this definition can be kept in hyperbolic geometry as well, and the point defined in this way keeps some of the important properties of the center of mass. For example, there are three independent force-free rotations around this point. A similar argument works for spherical geometry as well, so we generalize Eq. 38 for the center of mass calculation. The center of mass in elliptic and hyperbolic geometries is the point \mathbf{c} that minimizes

$$I_{S,H}(\mathbf{c}) = \sum_i a_{S,H}(\mathbf{q}_i, \mathbf{c})m_i \tag{39}$$

with the constraint that this point must be in the geometry, i.e. must satisfy $\langle \mathbf{c}, \mathbf{c} \rangle = \mathcal{L}$. Plugging in the geometry-dependent

attenuation factors (Eq. 32), we have to find the critical point of $\sum_i m_i \langle \mathbf{q}_i, \mathbf{c} \rangle^2$.

Unfortunately, as Zitterbarth [32] pointed out, there is no guarantee that under a force-free motion the center of mass moves along a geodesic, so computing the acceleration of the center of mass from the total force is only an approximation. Note also that forces attacking the body at points different from the center of mass cannot be used directly since the force must be in the tangent space of the point of action. Forces can be translated to the center of mass by parallel transport, i.e. according to Eq. 30.

7 Game adaptation and results

There are three distinct parts of a game that are affected by the geometry of the game universe: definition of objects, physics simulation including animation and illumination, and transformations. Object definition can be adapted according to Sect. 5. This conversion can happen when the models are loaded from files or uploaded to the GPU by the engine and also when vertex information is used in physics simulation or in the shaders during rendering. The last option is less efficient since it executes the conversion whenever vertex data are used, but requires minor modifications, and allows the interactive modification of the curvature.

The transformation matrices can be set according to the results of Sect. 4. This operation often requires the calculation of a vector that is orthogonal to two other vectors. Examples include Gram–Schmidt orthogonalization finding the coordinate frame, e.g. for the camera transformation matrix, the Frenet frame, or for the modeling transformation of a billboard. In non-Euclidean geometry, the orthogonal vector can be obtained with the evaluation of the determinant of Eq. 18, which also includes the point where this computation is executed. Note that there are no similarity transformations in non-Euclidean spaces, so scaling cannot be mimicked with matrices, but must be the part of object conversion (Eq. 29).

Similarly to vertex data, the modified computation of matrices can be inserted in the game engine where it calculates the matrices or also where it uses them for transformation. In the second case, the vertex shader can modify the matrices calculated by the Euclidean game engine before the actual transformation takes place. If the game does not apply dynamics simulation and illumination is calculated in the GPU shaders only, all changes can be concentrated in the vertex and fragment shaders.

The following simplified vertex shader can handle a single light source and assumes that vertices $vtxPos$ and normals $vtxNorm$ are defined by Cartesian coordinates in Euclidean space, the modeling transformation is rigid, i.e. is a combination of rotation R and translation T . The rotation matrix can be obtained with the Rodriguez formula, the translation matrix with Eq. 25. View matrix V and projection matrix P must be

calculated according to Eqs. 26 and 28. After the projection transformation, objects are in the normalized device space as required by the GPU. So clipping, projection and visibility determination can be executed uniformly with the fixed function pipeline elements. In this implementation, these matrices are passed as uniform variables, but it is also possible for the shaders to extract this information from the matrices assuming Euclidean geometry. The scaling factor is $scale$. The light source and the eye are defined with embedding coordinates by uniform variables $wLightPos$ and $wEyePos$.

For the sake of simplicity, we ignored texturing and shadow mapping. Parameter $curv$ takes values $-1, 0$ and 1 for hyperbolic, Euclidean and elliptic geometries, respectively.

Considering diameters as “points” in elliptic geometry means that an object is visible in the location mirrored at the embedding origin as well. This effect is produced by rendering every object twice, once with the original coordinates and once with negated ones. Parameter $anti$ is 1 for hyperbolic and Euclidean geometry and also for the first rendering pass of elliptic geometry. It only takes value -1 in the second rendering pass of elliptic geometry.

The vertex shader computes the normal vector $wNormal$, view vector $wView$ and illumination vector $wLight$ in the vertex according to the rules of the given geometry. The lengths of the view and illumination vectors encode the square roots of the attenuation factors, which can be used in illumination computations.

```
uniform mat4 R, T, V, P; // Rotate, Translate, View, Project
uniform float scale; // scaling factor
uniform vec4 wLightPos, wEyePos; // light source and eye
uniform float curv; // curvature: -1, 0, 1
uniform float anti; // -1 in the 2nd pass of elliptic

layout(location = 0) in vec3 vtxPos; // Euclidean vertex
layout(location = 1) in vec3 vtxNorm; // Euclidean normal
out vec4 wNormal, wView, wLight; // transformed

vec4 port(vec3 ePoint) { // port from Euclidean geometry
    vec3 p = ePoint * scale; // scaling happens here
    float d = length(p); // distance from geometry origin
    if (d < 0.0001 || curv == 0) return vec4(p, 1);
    if (curv > 0) return vec4(p/d * sin(d), cos(d));
    if (curv < 0) return vec4(p/d * sinh(d), cosh(d));
}

void main() {
    vec4 wPos = anti * port(vtxPos) * R * T; // location
    gl_Position = wPos * V * P;
    wNormal = anti * vec4(vtxNorm, 0) * R * T;
    wView = direction(wEyePos, wPos);
    wLight = direction(wLightPos, wPos);
}
```

The fragment shader differs from a standard solution only in the dot product calculation:

```
uniform vec4 La, Le, ka, kd, ks; // Light & material
uniform float shininess;
uniform float curv; // curvature: -1, 0, 1
in vec4 wNormal, wView, wLight;
out vec4 fragmentColor; // Computed radiance

void main() {
    vec4 N = normalize(wNormal);
    vec4 V = normalize(wView);
    float atten = dot(wLight, wLight); // attenuation
    vec4 L = normalize(wLight);
    vec4 H = normalize(L + V);
    float cost = max(dotProduct(N, L), 0);
    float cosd = max(dotProduct(N, H), 0);
    float mat = kd * cost + ks * pow(cosd, shininess);
    fragmentColor = ka * La + mat * Le / atten;
}
```

Both the vertex and the fragment shaders call the custom *dotProduct* function that calculates the dot product taking into account whether the 4D embedding space is Euclidean or of Minkowski type:

```
float dotProduct(vec4 u, vec4 v) {
    return dot(u, v) - ((curv < 0) ? 2 * u.w * v.w : 0);
}
```

The computation of the direction vector also depends on the geometry encoded by the *curv* parameter. The length of the returned direction vector is the square root of the attenuation factor between the two points, which can be used in lighting calculations.

```
vec4 direction(vec4 to, vec4 from) {
    float dp = (curv != 0) ? dotProduct(from, to) : 1;
    return to - from * dp;
}
```

To demonstrate the results, we implemented three games (Figs. 6, 7 and 8). The “Lego” and the “Museum” do not use dynamic simulation, so they concentrated all non-Euclidean calculations in the vertex and fragment shaders leaving other parts of the game engine unaffected. The “Fight in space” controlled the spaceships’ positions with dynamic simulation taking into account the geometry-dependent gravitational field of the planets. The spaceships’ rotations are determined with path animation calculating the Frenet frames.

8 Conclusion

This paper proposed a minimally invasive approach to modify a game from Euclidean to elliptic or hyperbolic geometry. We investigated the adaptation of the geometric calculations, object definitions, transformation matrices and the physical simulation.

The implementation that can easily be plugged into UnityEngine3D [4] games is available at

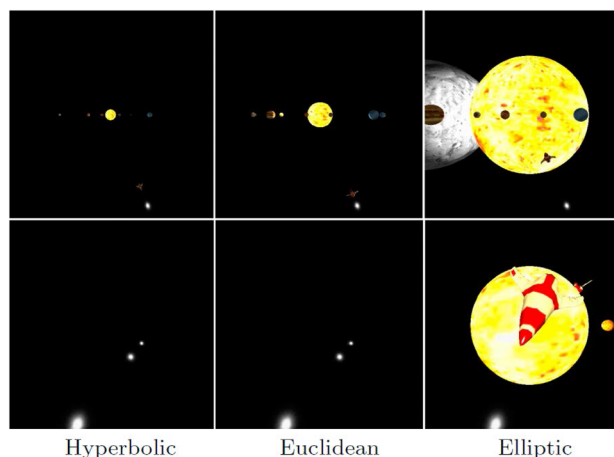


Fig. 6 “Fight in space” game in hyperbolic (left) Euclidean (middle), and elliptic (right) spaces when looking forward (upper row) and backward (lower row). Note that in elliptic spaces objects at distance close to π have similar perceived size as objects being at distance close to zero. When turning back, we can see the same objects in reverse order. As the space is curved, in the upper images we can see the bottom of the spaceship in elliptic geometry, while its top is visible in hyperbolic space

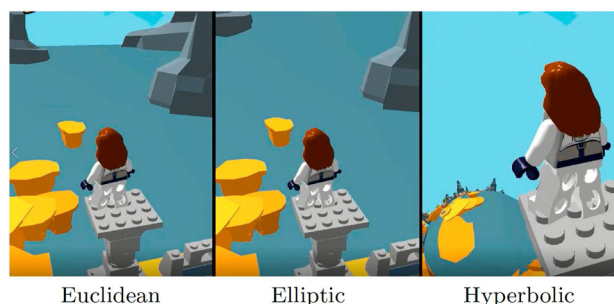


Fig. 7 “Lego game” in Euclidean, elliptic and hyperbolic spaces. Video is attached as supplementary material

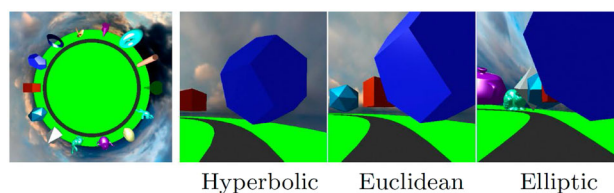


Fig. 8 “Museum” top view and rendering in hyperbolic, Euclidean and elliptic spaces. Elliptic geometry is the most unnatural as approaching objects first become smaller than they grow back.

<https://github.com/mmagdics/noneuclideanunity>.

In future work, we plan to extend the engine to other Thurston geometries using their projective interpretation [18].

Supplementary Information The online version contains supplementary material available at <https://doi.org/10.1007/s00371-021-02303-2>.

Acknowledgements This project has been supported by OTKA K-124124.

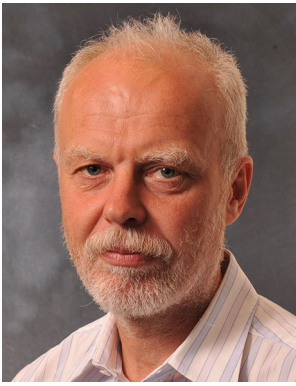
Funding Open access funding provided by Budapest University of Technology and Economics.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Amenta, N., Levy, S., Munzner, T., Phillips, M.: Geomview: a system for geometric visualization. In: Proceedings of the Eleventh Annual Symposium on Computational Geometry, SCG '95, pp. 412–413, 1995. <https://doi.org/10.1145/220279.220327>
- Berger, P., Laier, A., Velho, L.: An image-space algorithm for immersive views in 3-manifolds and orbifolds. *Vis. Comput.* **31**, 93–104 (2015). <https://doi.org/10.1007/s00371-013-0913-2>
- Blinn, J.: Jim Blinn's Corner: Notation, Notation. Morgan Kaufmann Publishers Inc., Burlington (2002)
- Bond, J.G.: Introduction to Game Design, Prototyping, and Development. Addison-Wesley, Boston (2014)
- Brinkmann, P., Gunn, C., Weissmann, S.: Jreality—interactive audiovisual applications across virtual environments. In: Proceedings of the 2010 IEEE Symposium on 3D User Interfaces, 3DUI '10, pp. 123–124 (2010)
- Coulon, R., Matsumoto, E.A., Segerman, H., Trettel, S. J.: Ray-marching Thurston geometries. <https://arxiv.org/abs/2010.15801> (2020)
- Ghadami, R., Rahebi, J., Yayli, Y.: Linear interpolation in Minkowski space. *Int. J. Pure Appl. Math.* **77**, 01 (2012)
- Gröller, E.: Nonlinear ray tracing: visualizing strange worlds. *Vis. Comput.* **11**(5), 263–274 (1995). <https://doi.org/10.1007/BF01901044>
- Guimaraes, F., Mello, V., Velho, L.: Geometry independent game encapsulation for non-Euclidean geometries. In: Proceedings of SIBGRAPI (2015)
- Gunn, C.: Advances in metric-neutral visualization. In: *GraVisMa*, pp. 17–26 (2010)
- Hart, V., Hawksley, A., Matsumoto, E. A., Segerman, H.: Non-euclidean virtual reality I: Explorations of H^3 (2017). <https://arxiv.org/abs/1702.04004>
- Kopczyński, E., Celińska, D., Čtrnáct, M.: Hyperrogue: playing with hyperbolic geometry. In: Swart, D., Séquin, C. H., Fenyvesi, K. (eds.) *Proceedings of bridges 2017: mathematics, art, music, architecture, education, culture*, pp. 9–16 (2017)
- Kopczyński, E., Celinska-Kopczyńska, D.: Real-time visualization in non-isotropic geometries (2020). <https://arxiv.org/abs/2002.09533>
- Lamping, J., Rao, R.: The hyperbolic browser: a focus+context technique for visualizing large hierarchies. *J. Vis. Lang. Comput.* **7**(1), 33–55 (1996). <https://doi.org/10.1006/jvlc.1996.0003>
- Loustau, B.: Hyperbolic geometry (2020). <https://arxiv.org/abs/2003.11180>
- Martelli, B.: An introduction to geometric topology (2016). <https://arxiv.org/abs/1610.02592>
- McCaleb, R.A., North, C.: Smooth, efficient, and interruptible zooming and panning. *IEEE Trans. Vis. Comput. Gr.* **25**(2), 1421–1434 (2019). <https://doi.org/10.1109/TVCG.2018.2800013>
- Molnár, E.: The projective interpretation of the eight 3-dimensional homogeneous geometries. *Beiträge Algebra Geom.* **38**(2), 261–288 (1997)
- Novello, T., da Silva, V., Velho, L.: Global illumination of non-Euclidean spaces. *Comput. Gr.* **93**, 61–70 (2020). <https://doi.org/10.1016/j.cag.2020.09.014>
- Novello, T., da Silva, V., Velho, L.: Visualization of nil, sol, and SL2(R) geometries. *Comput. Gr.* **91**, 219–231 (2020). <https://doi.org/10.1016/j.cag.2020.07.016>
- Osudin, D., Child, C., He, Y.-H.: Rendering non-Euclidean space in real-time using spherical and hyperbolic trigonometry. *Comput. Sci. - ICCS 2019*, 543–550 (2019). https://doi.org/10.1007/978-3-030-22750-0_49
- Phillips, M., Gunn, C.: Visualizing hyperbolic space: unusual uses of 4x4 matrices. In: *Proceedings of the 1992 Symposium on Interactive 3D Graphics, I3D '92*, pp. 209–214 (1992). <https://doi.org/10.1145/147156.147206>
- Salvai, M.: On the dynamics of a rigid body in the hyperbolic space. *J. Geom. Phys.* **36**(1), 126–139 (2000). [https://doi.org/10.1016/S0393-0440\(00\)00017-6](https://doi.org/10.1016/S0393-0440(00)00017-6)
- Shoemake, K.: Animating rotation with quaternion curves. *Comput. Gr.* **16**(3), 157–166 (1985)
- Szirmay-Kalos, L., Magdics, M.: Gaming in Elliptic geometry. In: Theisel H., Wimmer M. (eds), *Eurographics 2021 - Short Papers* (2021). <https://doi.org/10.2312/egs.20211010>
- Thielhelm, H., Vais, A., Wolter, F.-E.: Geodesic bifurcation on smooth surfaces. *Vis. Comput.* **31**(2), 187–204 (2015). <https://doi.org/10.1007/s00371-014-1041-3>
- Velho, L., Silva, V., Novello, T.: Immersive visualization of the classical non-Euclidean spaces using real-time ray tracing in VR. In: *Proceedings of Graphics Interface 2020, GI 2020*, pp. 423–430 (2020). <https://doi.org/10.20380/GI2020.42>
- Weeks, J.: Real-time rendering in curved spaces. *IEEE Comput. Gr. Appl.* **22**(6), 90–99 (2002)
- Weeks, J.: Body coherence in curved-space virtual reality games. *Comput. Gr.* **97**, 28–41 (2021). <https://doi.org/10.1016/j.cag.2021.04.002>
- Weiskopf, D., Borchers, M., Ertl, T., Falk, M., Fechtig, O., Frank, R., Grave, F., King, A., Kraus, U., Muller, T., Nollert, H., Mendez, I.R., Ruder, H., Schafhitzel, T., Schar, S., Zahn, C., Zatloukal, M.: Explanatory and illustrative visualization of special and general relativity. *IEEE Trans. Vis. Comput. Gr.* **12**(4), 522–534 (2006). <https://doi.org/10.1109/TVCG.2006.69>
- Weiskopf, D.: Visualization of four-dimensional spacetimes. PhD thesis, University of Tübingen, Germany (2001)
- Zitterbarth, J.: Some remarks on the motion of a rigid body in a space of constant curvature without external forces. *Demonstratio Math.* **24**(3–4), 465–494 (1991). <https://doi.org/10.1515/dema-1991-3-407>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



László Szirmay-Kalos was graduated from the Budapest University of Technology in 1987, received PhD and Doctor of Science degree from the Hungarian Academy of Science in 1991 and in 2001, respectively. He is currently a full professor of computer graphics at Budapest University of Technology. His research interests include rendering, Monte Carlo methods and medical imaging. He is the fellow of Eurographics. His web-page is. [https://www.iit.bme.hu/users/dr-szirmay-kalos-1%C3%](https://www.iit.bme.hu/users/dr-szirmay-kalos-1%C3%A1szl%C3%B3?language=en)

[A1szl%C3%B3?language=en](https://www.iit.bme.hu/users/dr-szirmay-kalos-1%C3%A1szl%C3%B3?language=en)



Milán Magdics was graduated from the Eötvös Lóránd Science University and received PhD from the Budapest University of Technology in 2014 in medical imaging. He is now involved in Monte Carlo methods, medical imaging and computational physics and works as an associate professor. He is member of Eurographics. His web page is. <https://www.iit.bme.hu/users/dr-magdics-mil%C3%A1n?language=en>