Rendering Non-Euclidean Space in Real-Time Using Spherical and Hyperbolic Trigonometry

Daniil Osudin(✉), Chris Child, and Yang-Hui He

City, University of London, London, UK
daniil.osudin.1@city.ac.uk

Abstract. We introduce a method of calculating and rendering shapes in a non-Euclidean 2D space in real-time using hyperbolic and spherical trigonometry. We record the objects' parameters in a polar coordinate system and use azimuthal equidistant projection to render the space onto the screen. We discuss the complexity of this method, renderings produced, limitations and possible applications of the created software as well as potential future developments.

Keywords: non-Euclidean geometry · Spherical trigonometry · Hyperbolic trigonometry · Azimuthal equidistant projection · Polar coordinate system · Real-time

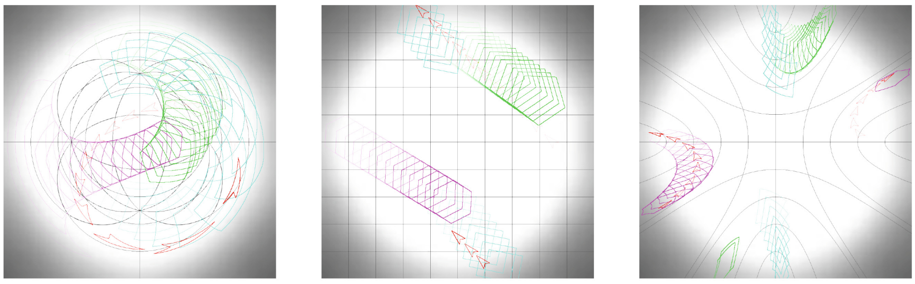


Fig. 1. Time-lapse images of multiple objects moving through spherical (a), planar (b) and hyperbolic (c) 2D space calculated and rendered by the described software

J. M. F. Rodrigues et al. (Eds.): ICCS 2019, LNCS 11540, pp. 543–550, 2019.
https://doi.org/10.1007/978-3-030-22750-0_49

1 Introduction

Non-Euclidean geometry is a field that studies any space that arises from changing Euclid's fifth postulate [1] or changing the metric requirement. In spherical geometry, Fig. 2(a), all geodesics (shortest paths in a non-planar space) intersect: don't preserve the distance and appear to 'bend' towards each other. In Hyperbolic geometry, Fig. 2(c), each line has an infinite number of parallel lines, as they appear to 'bend' away from each other.

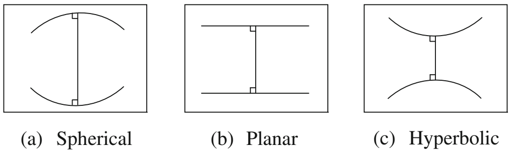


Fig. 2. Comparison of parallel lines in the 2D spaces

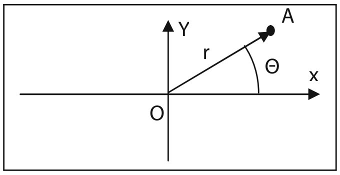


Fig. 3. Point A with polar coordinates r and θ

We present a method for calculating the object's position and its vertices in polar coordinates using spherical [2] or hyperbolic trigonometry [3,4]. A polar coordinate system of the form (r, θ) is used in this model for all calculations instead of Cartesian coordinates. The centre of the of the screen is taken as a reference point $O(0, 0)$ for the distance coordinate, r, while eastbound is the reference direction for the bearing coordinate, θ. This allows the same coordinates to be used irrespective of the correct curvature. In order to render the curved space onto a flat 2D screen, we are using azimuthal equidistant projection. By definition, distances and bearing from the centre of the projection are preserved. This works well with Polar coordinates, projection is intuitive and can be used with no changes for both spherical and hyperbolic 2D spaces (Fig. 3).

2 Method

The calculations are split into two parts: movement of the objects and rendering of the shapes. The screen (rendering space) is limited to a circle of an arbitrary size. When the object's centre moves past the circumference of the circle, it is repositioned to the antipodal point on the circle with the velocity preserved. This is implemented in order to keep the objects in the visible area on the screen.

Shape has a list of position vectors for each vertex in local coordinates with object's position being the reference point and reference direction is taken as the reverse of its position vector (Fig. 4).

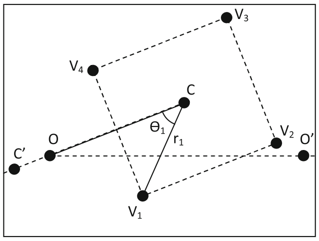


Fig. 4. O $(0, 0)$, reference point; C (r_c, θ_c), position and local reference point; V$_x$ (r_x, θ_x), vertices; OO', reference direction; CC', local reference direction

2.1 Rendering the Shape

Let $K \in [-1, 1] \subset \Re$ s.t. $K = 0 \Rightarrow$ Euclidean Geometry;

$K > 0 \Rightarrow$ Spherical Geometry, $r = \frac{1}{\sqrt{K}}$

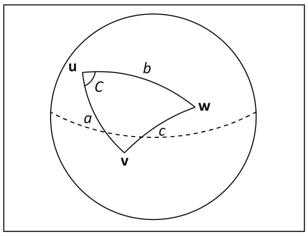


Fig. 5. Spherical triangle

Theorem 1. *For a sphere of radius* $\boldsymbol{r}$ *and hence Gaussian curvature* $K = \frac{1}{r^2}$ *and a spherical triangle on its surface described by points* $\boldsymbol{u}$, $\boldsymbol{v}$ *and* $\boldsymbol{w}$, *connected by great circles that form the edges* a, b, c *(interpreted as subtended angles) and an angle* C (Fig. 5), *the spherical law of cosines [5] states:*

$$\cos\frac{c}{r} = \cos\frac{a}{r}\cos\frac{b}{r} + \sin\frac{a}{r}\sin\frac{b}{r}\cos C \tag{1}$$

$K < 0 \Rightarrow$ Hyperbolic Geometry, $k = -\frac{1}{\sqrt{K}}$

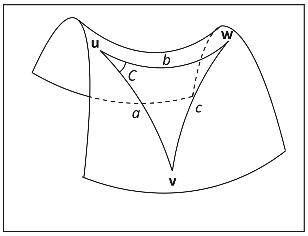


Fig. 6. Hyperbolic triangle

Theorem 2. *For a hyperbolic plane with Gaussian Curvature* $K = -\frac{1}{k^2}$ *and a hyperbolic triangle on its surface described by points* $\boldsymbol{u}$, $\boldsymbol{v}$ *and* $\boldsymbol{w}$, *connected by geodesics that form the edges* a, b *and* c, *as well as an angle* C (Fig. 6), *the hyperbolic law of cosines [6] states:*

$$\cosh\frac{c}{k} = \cosh\frac{a}{k}\cosh\frac{b}{k} - \sinh\frac{a}{k}\sinh\frac{b}{k}\cos C \tag{2}$$

Note: To simplify the equations below, all lengths are assumed to have been divided by r or k depending on the value of K.

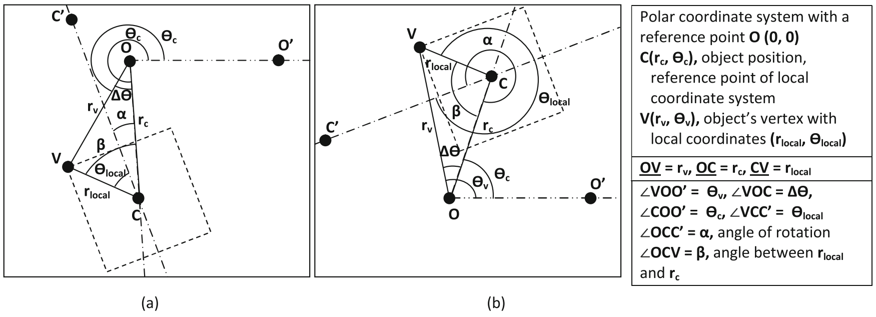

C'
O
O'
θc
ΔΘ
rv
α
rc
β
V
Θlocal
rlocal
C
(a)
V
α
rlocal
C
Θlocal
β
rv
rc
C'
ΔΘ
θv
Θc
O
O'
(b)
Polar coordinate system with a reference point O (0, 0)
C(rc, Θc), object position, reference point of local coordinate system
V(rv, Θv), object's vertex with local coordinates (rlocal, Θlocal)
OV = rv, OC = rc, CV = rlocal
∠VOO' = Θv, ∠VOC = ΔΘ,
∠COO' = Θc, ∠VCC' = Θlocal
∠OCC' = α, angle of rotation
∠OCV = β, angle between rlocal and rc

Fig. 7. Finding the θ and r coordinates of an object's vertices through a hyperbolic/spherical triangle OCV; Case (a): $\theta_{local} + \alpha < \pi$; case (b): $\theta_{local} + \alpha > \pi$

Corollary 1. *Given:* $\boldsymbol{O}(0,0)$, $\boldsymbol{C}(r_c,\theta_c)$, $\boldsymbol{V}(r_v,\theta_v)$, $\underline{\boldsymbol{OC}} = r_c$, $\underline{\boldsymbol{CV}} = r_{local}$, $\angle\boldsymbol{COO'} = \theta_c$, $\angle\boldsymbol{OCC'} = \alpha$, $\angle\boldsymbol{VCC'} = \theta_{local}$

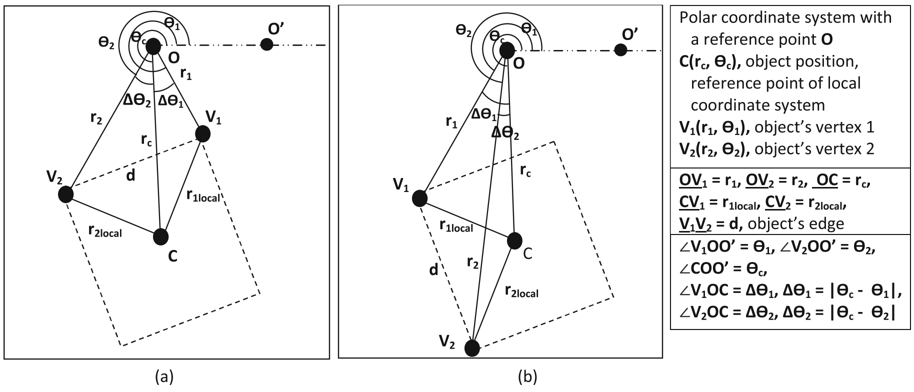
Find: r_v, θ_v = ?

If $K > 0$, *then:* *If* $K < 0$, *then:*

$$\cos r_v = \cos r_c \cos r_{local} + \sin r_c \sin r_{local} \cos\beta \qquad \cosh r_v = \cosh r_c \cosh r_{local} - \sinh r_c \sinh r_{local} \cos\beta \tag{3}$$

$$\cos\varDelta\theta_v = \frac{\cos r_{local} - \cos r_c \cos r_v}{\sin r_c sin r_v} \qquad \cos\varDelta\theta_v = \frac{\cosh r_c \cosh r_v - \cosh r_{local}}{\sinh r_c \sinh r_v} \tag{4}$$

In order to find r_v, *first find* $\beta = \alpha + \theta_{local}$; *if* $\Pi < \beta < 2\Pi$, *use the explementary angle instead to determine to which side of* $\underline{OC}$ *the triangle lies. Depending on that* $\varDelta\theta$ *is then added to or subtracted from* θ_c *to find* θ_v *(Fig. 7).*



Polar coordinate system with a reference point **O**
C(rc, Θc), object position, reference point of local coordinate system
V1(r1, Θ1), object's vertex 1
V2(r2, Θ2), object's vertex 2

$\underline{OV_1}$ = r1, $\underline{OV_2}$ = r2, $\underline{OC}$ = rc,
$\underline{CV_1}$ = r1local, $\underline{CV_2}$ = r2local,
$\underline{V_1V_2}$ = d, object's edge

∠V1OO' = Θ1, ∠V2OO' = Θ2,
∠COO' = Θc,
∠V1OC = ΔΘ1, ΔΘ1 = |Θc - Θ1|,
∠V2OC = ΔΘ2, ΔΘ2 = |Θc - Θ2|

Fig. 8. Finding the length of edge d and the angle $\varDelta\theta$. Case (a), $\varDelta\theta_1$ and $\varDelta\theta_2$ diverge, so $\varDelta\theta$ is the sum; case (b), angles converge, so $\varDelta\theta$ is the absolute value of the difference.

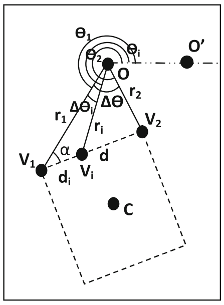
Corollary 2. *Given:* $\boldsymbol{O}(0,0)$, $\boldsymbol{C}(r_c,\theta_c)$, $\boldsymbol{V}_1(r_1,\theta_1)$, $\boldsymbol{V}_2(r_2,\theta_2)$, $\underline{\boldsymbol{OC}} = r_c$, $\overline{\boldsymbol{OV}_1} = r_1$, $\underline{\boldsymbol{OV}_2} = r_2$, $\underline{\boldsymbol{CV}_1} = r_{1local}$, $\underline{\boldsymbol{CV}_2} = r_{2local}$, $\angle\boldsymbol{COO'} = \theta_c$, $\angle\boldsymbol{V}_1\boldsymbol{OO'} = \theta_1$, $\angle\boldsymbol{V}_2\boldsymbol{OO'} = \theta_2$ (Fig. 8)

Find: d, $\varDelta\theta$ = ?

If angles converge, $\varDelta\theta = \|\varDelta\theta_1 - \varDelta\theta_2\|$; *if angles diverge,* $\varDelta\theta = \|\varDelta\theta_1\| + \|\varDelta\theta_2\|$

If $K > 0$, *then:* *If* $K < 0$, *then:*

$$\cos d = \cos r_1 \cos r_2 + \sin r_1 \sin r_2 \cos\varDelta\theta \qquad \cosh d = \cosh r_1 \cosh r_2 - \sinh r_1 \sinh r_2 \cos\varDelta\theta \tag{5}$$



Polar coordinate system with a reference point **O** **V₁(r₁, ϴ₁), V₂(r₂, ϴ₂),** vertices **Vᵢ(rᵢ, ϴᵢ),** intermediate point
OV₁ = r₁, OV₂ = r₂, OVᵢ = rᵢ, V₁V₂ = d, object's edge **V₁Vᵢ = dᵢ,** section of the edge
∠V₁OO' = ϴ₁, ∠V₂OO' = ϴ₂, ∠VᵢOO' = ϴᵢ, ∠OV₁Vᵢ = α, ∠V₁OV₂ = Δϴ, angle between **V₁** and **V₂** **∠V₁OVᵢ = Δϴᵢ,** angle between **V₁ and Vᵢ**

Fig. 9. Finding intermediate points in order to render the edge.

Note: distance d is divided into a number of equal parts in order to find the distance d_i for each of the points on the edge V_1V_2. The number of segments depends on the object tesselation variable.

Corollary 3. *Given:* $\boldsymbol{O}(0,0)$, $\boldsymbol{V}_1(r_1, \theta_1)$, $\boldsymbol{V}_2(r_2,\theta_2)$, $\boldsymbol{V}_i(r_i,\theta_i)$, $\underline{\boldsymbol{OV}_1} = r_1$, $\underline{\boldsymbol{OV}_2} = r_2$, $\underline{\boldsymbol{V}_1\boldsymbol{V}_2} = \boldsymbol{d}$, $\underline{\boldsymbol{V}_1\boldsymbol{V}_i} = d_i$, $\angle \boldsymbol{V}_1\boldsymbol{OO}' = \theta_1$, $\angle \boldsymbol{V}_2\boldsymbol{OO}' = \theta_2$, $\angle \boldsymbol{V}_1\boldsymbol{OV}_2 = \Delta\theta$

Find: r_i, θ_i = ?

If $K > 0$, *then:*

$$\cos\alpha = \frac{\cos r_2 - \cos r_1 \cos d}{\sin r_1 \sin d}$$

$$\cos r_i = \cos r_1 \cos d_i + \sin r_1 \sin d_i \cos\alpha$$

$$\cos \Delta\theta_i = \frac{\cos d_i - \cos r_1 \cos r_i}{\sin r_1 \sin r_i}$$

If $K < 0$, *then:*

$$\cos\alpha = \frac{\cosh r_1 \cosh d - \cosh r_2}{\sinh r_1 \sinh d} \tag{6}$$

$$\cosh r_i = \cosh r_1 \cosh d_i - \sinh r_1 \sinh d_i \cos\alpha \tag{7}$$

$$\cos \Delta\theta_i = \frac{\cosh r_1 \cosh r_i - \cosh d_i}{\sinh r_1 \sinh r_i} \tag{8}$$

α is calculated to find the angle opposite r_i. Then r_i and subsequently $\Delta\theta_i$ can be found using the cosine rule (illustrated on Fig. 9*). Then to find actual coordinates of the point V_i, r_i should be multiplied by r or k depending on the value of K; $\Delta\theta_i$ should be added to or subtracted from angle $\theta 1$, depending on the direction of the edge d, determined previously.*

2.2 Updating Object Position

Corollary 4. *Given:* $\boldsymbol{O}(0,0)$, $\boldsymbol{C}_{t0}(r_{t0},\theta_{t0})$, $\boldsymbol{C}_{t1}(r_{t1},\theta_{t1})$, $\underline{\boldsymbol{OC}_{t0}} = r_{t0}$, $\underline{\boldsymbol{C}_{t0}\boldsymbol{C}_{t1}} = r_p$, $\angle \boldsymbol{C}_{t0}\boldsymbol{OO}' = \theta_{t0}$, $\angle \boldsymbol{OC}_{t0}\boldsymbol{C}'' = \gamma_{t0}$, $\angle \boldsymbol{OC}_{t0}\boldsymbol{C}'_{t0} = \beta_{t0}$

Find: r_{t1}, θ_{t1}, γ_{t1}, β_{t1} = ?

γ_{t0} *should be* 0 *to* π, *if calculated value is* $\gamma_{t0} > \pi$, *take the explemntary angle. This indicates the movement direction with respect to the reference point* (Fig. 10).

Let $\angle OC_{t1}C_{t0} = \gamma'_{t1}$

If $K > 0$, *then: If* $K < 0$, *then:*

$$\cos r_{t1} = \cos r_{t0} \cos r_p + \sin r_{t0} \sin r_p \cos\alpha \qquad \cosh r_{t1} = \cosh r_{t0} \cosh r_p + \sinh r_{t0} \sinh r_p \cos\alpha \tag{9}$$

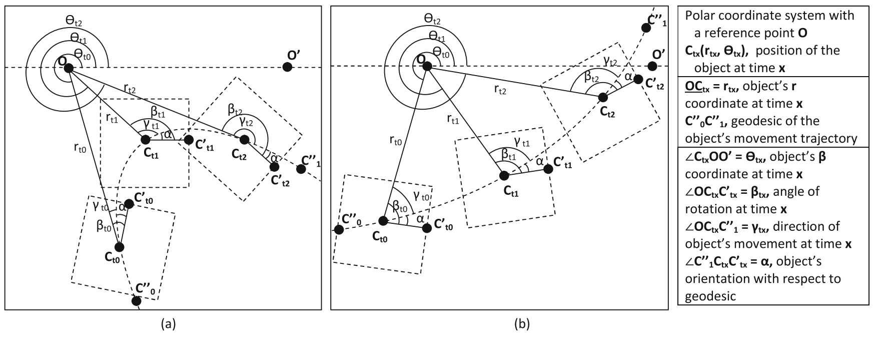

Θt2
Θt1
Θt0
O
O'
rt2
rt1
rt0
βt1
γt1
α
Ct1
C't1
βt2
γt2
Ct2
C't2
C''1
γt0
βt0
C't0
Ct0
C''0
(a)
(b)
Polar coordinate system with a reference point O
Ctx(rtx, Θtx), position of the object at time x
OCtx = rtx, object's r coordinate at time x
C''0C''1, geodesic of the object's movement trajectory
∠CtxOO' = Θtx, object's β coordinate at time x
∠OCtxC'tx = βtx, angle of rotation at time x
∠OCtxC''1 = γtx, direction of object's movement at time x
∠C''1CtxC'tx = α, object's orientation with respect to geodesic

Fig. 10. Movement of the object along a hyperbolic in Spherical (a) and Hyperbolic (b) space. Orientation with respect to the geodesic is kept the same (angle α is constant) if the object is not rotating.

$$\cos \varDelta\theta = \frac{\cos r_p - \cos r_{t0} \cos r_{t1}}{\sin r_{t0} \sin r_{t1}} \qquad \cos \varDelta\theta = \frac{\cosh r_{t0} \cosh r_{t1} - \cosh r_p}{\sinh r_{t0} \sinh r_{t1}} \tag{10}$$

$$\cos \gamma'_{t1} = \frac{\cos r_{t0} - \cos r_p \cos r_{t1}}{\sin r_p \sin r_{t1}} \qquad \cos \gamma'_{t1} = \frac{\cosh r_p \cosh r_{t1} - \cosh r_{t0}}{\sinh r_p \sinh r_{t1}} \tag{11}$$

$\alpha = \beta_{t0} - \gamma_{t0}$. α is the difference between rotation direction and the geodesic of movement ($C"_0C"_1$), it does not change if the object is not rotating. Hence, $\beta_{t1} = \gamma_{t1} + \alpha$. Because γ'_{t1} and γ_{t1} are supplementary angles, $\gamma_{t1} = \varPi - \gamma'_{t1}$.

To find the θ coordinate, either subtract or add $\varDelta\theta$ to the θ_c depending on whether the angle α or its explementary angle is used for this calculation.

3 Results

3.1 Implementation

Using the method described above and OpenGL, we created a software capable of calculating the objects and rendering the vector graphics in a non-Euclidean space with constant curvature in the range of $-1 \leq K \leq 1$. Figure 1 shows the time-lapses of multiple objects in spherical (a), planar (b) and hyperbolic (c) geometries. They show movement through different geodesics at $K = 1$, $K = 0$ and $K = -1$ respectively. Starting positions as well as shape definitions of each object are the same across all time-lapses (grid-lines have been created and rendered as separate objects). The software can calculate the object moving in arbitrary direction with arbitrary speed as well as starting from arbitrary position in the space.

Curvature of the world can be modified in real-time using keyboard inputs in a similar manner to controlling the object's acceleration and orientation. Another feature is the cut-off of the world at a distance of N pixels. This can be seen in the hyperbolic and planar time-lapse images. While these spaces should be infinite, we chose to limit them in order to keep objects within the boundaries of the screen (non-shaded area). We created a video [7] displaying the implementation.

3.2 Complexity Analysis

Positions of each vertex need to be calculated, requiring $O(v)$ time, where v is the number of vertices. Subsequently, intermediate points have to be computed, requiring $O(i)$ time to find all of the points on a single edge, where i is the level of tessellation. Complexity to render the world with s number of shapes is therefore $O(s * v * i)$. The best case would be equal to $O(n)$ complexity, if two of the terms are negligibly small. The worst case can be approximated to $O(n^3)$ if all terms were comparably large. Spatial complexity for shape rendering is only $O(v * i)$ as previous shape's data is rewritten to store the next shape's data. So either $O(n)$ in the best case or $O(n^2)$ in the worst case.

Only one movement calculation per object is required and the previous position record is overwritten, both spatial and time complexity is $O(n)$, where n is the number of objects in the world.

Trigonometric and hyperbolic functions in the calculations are slower to compute than simpler operations, hence additional cost (implementation dependent). For example the AGM iteration [8] method is faster than the previously common Taylor series method.

4 Discussion

Implementation does not affect performance up to a certain number of objects or tessellation amount. The focus was on implementing the method correctly and having it work continuously under any curvature in the range $-1 \leq K \leq 1$.

The next step in the project's development is improving the execution time using parallelised calculations. Subsequent calculation of the points creates a bottleneck, which can be solved by performing some calculations directly on the GPU. Other approaches are considered as well, including lookup tables to speed up trigonometric calculations, for example, Frank Rochet's implementation [9]; or finding intermediate points from a geodesic equation.

Potential applications for the software include education about non-Euclidean geometry (more intuitive than standard projections: Poincare disk and Upper Half-Plane models); cartography [10] (the engine could be modified to efficiently convert data into different projections); ecology [11] and climatology [12] (modelling dynamic systems); Astrophysics (modelling systems of cosmological objects and gravitational fields) and video games (game engine for a real-time continuous non Euclidean space, unlike HyperRogue [13], which uses step by step implementation).

References

1. Heath, T.L.: Euclid's Elements. Dover, New York (1956). (translated)
2. Todhunter, I.: Spherical Trigonometry For the use of colleges and schools. Project Gutenberg License (1886). (republished 12 November 2006)
3. Carslaw, H.S.: The Elements of Non-Euclidean Plane Geometry and Trigonometry. Longmans, Green and Co., London (1916)
4. Traver, T.: Trigonometry in the hyperbolic plane (2014). Accessed December 2017, Manuscript
5. Gellert, W., Gottwald, S., Hellwich, M., Kästner, H., Küstner, H.: The VNR Concise Encyclopedia of Mathematics, 2nd edn. Van Nostrand Reinhold, New York (1989). ch. 12
6. Gray, J.: Non-Euclidean geometry–a re-interpretation. Historia Mathematica **6**, 236–258 (1979)
7. Osudin, D., Child, C., He, Y.-H.: Rendering non-Euclidean space in realtime using spherical and hyperbolic trigonometry (2019). https://youtu.be/A1ZCFh5qfNg
8. Brent, R.P.: Multiple-precision zero-finding methods and the complexity of elementary function evaluation (2010). http://arxiv.org/abs/1004.3412v2. Accessed 26 Aug 2018
9. Rochet, F.: Fast trigonometry functions using lookup tables (2004). http://www.flipcode.com/archives/Fast_Trigonometry_Functions_Using_Lookup_Tables.shtml. Accessed 30 Aug 2018
10. Gartner, G., Huang, H.: Recent research developments in modern cartography in Europe, Issue 1: EuroCarto 2015 (2015)
11. Sutherland, C., Fuller, A.K., Royle, J.A.: Modelling non-Euclidean movement and landscape connectivity in highly structured ecological networks. Methods Ecol. Evol. **6**, 169–177 (2015)
12. Frei, C.: Interpolation of temperature in a mountainous region using nonlinear profiles and non-Euclidean distances. Int. J. Climatol. **34**, 1585–1605 (2013). https://doi.org/10.1002/joc.3786
13. Zeno Rogue Games: Hyperrogue (2019). http://roguetemple.com/z/hyper/