Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 24, 2021

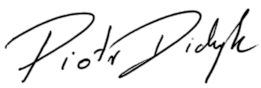
Rendering a 3D hyperbolic space

Giovagnini Alessio

Abstract

Computer graphics make extensive use of 4x4 matrices to compute linear transformation, but all the operations are made to simulate the behaviour of geometry in a euclidean space. Other kinds of non-euclidean geometry exists, each with interesting and mind-bending visual effects. In this project, I decided to focus on hyperbolic geometry in particular. To start I implemented special 4x4 matrices [3] that represents transformations in hyperbolic space. Then I created a web application that renders a simple 3D hyperbolic world in real time using WebGl and JavaScript. The application also allows to visualise the world from both an outsider perspective and a first person view where it is possible to move and look around giving the impression of walking inside the hyperbolic space. I will briefly introduce this special geometry and present the transformations used in the project. Hyperbolic geometry has interesting properties that pose unusual challenges in computer graphics. In this report I will propose my solutions for different problems, for example how to correctly compute normals transformation in the shaders and explain a simple way to freely move inside the world using mapping functions. I will also explain shadow computation and the problems that the hyperbolic world pose for the geometry near the border of the sphere. This non-euclidean geometry produce some very peculiar visual effects, I will show different examples and explain how I obtained them.

Advisor
Prof. Didyk Piotr

Advisor's approval (Prof. Didyk Piotr):  Date: 24.06.2021

Contents

1 Introduction

I was first introduced to hyperbolic geometry when I watched the trailer of the game Hyperbolica, the game is still in development and will be released, hopefully, later this year. The premise of the game is to let the player explore and play in a non-euclidean space where classical geometry is replaced by unique and puzzling effects. I was very interested in this different geometry so I searched for more information and found an interesting paper [3] on hyperbolic transformation that dates back to December 1991. I then decided to focus my bachelor project on hyperbolic geometry using this paper as a base.

1.1 Classical geometry

Euclidean geometry is the mathematical system described by Euclid of Alexandria (300 BC). In his textbook "The Elements", five postulates are stated:

1. Any two points can be joined with a straight line.
2. A straight segment can be extended indefinitely in a straight line.
3. A circle can be drawn by having a centre an a distance.
4. All right angles are congruent.
5. The parallel postulate: If two lines intersect a third line and the sum of the inners angles formed are less than two right angles, then the two lines will intersect eventually.

For more than two thousands years no other geometry was conceived since it appeared so intuitive and it represented reality so well. Today other non-euclidean geometries are known to exist, physical space has been discovered to be curved and that classical euclidean geometry is only an approximation of it in relative short distance.

1.2 What is hyperbolic geometry?

Hyperbolic geometry [1] is a non-euclidean geometry obtained by modifying the fifth postulate previously stated to:

- For any line L and point P not on L, in the plane that contains both L and P there are two distinct lines (f, g) through P that do not intersect L.

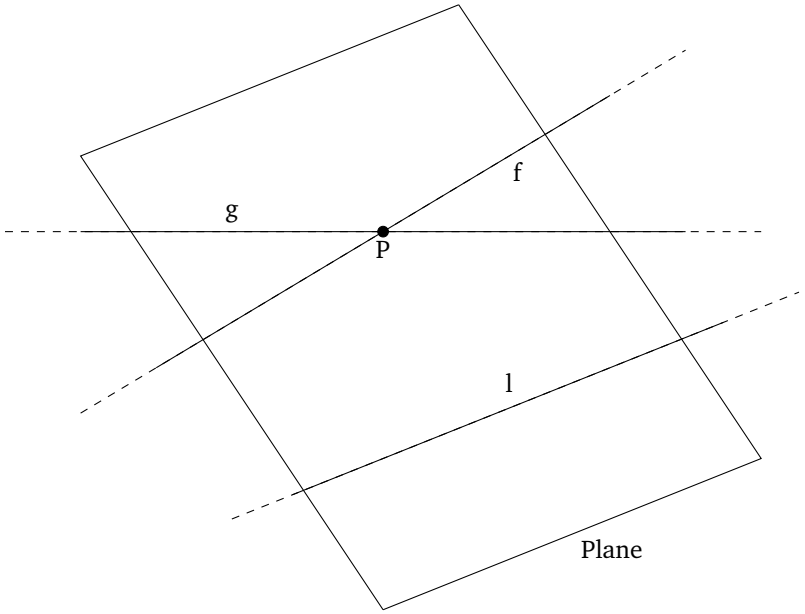


Figure 1. Lines g and f are not parallel to L but at the same time they will not intersect L even if they all lay on the same plane.

Figure 1 show three lines (l, g, and f) and a point P laying on a plane, g and g intersect P and are not parallel to l, in Euclidean geometry if they were straight lines they will eventually intersect with l but in hyperbolic geometry this is not the case. The notion that were given for granted in Euclidean space are not valid anymore for the hyperbolic space. Another example is found in triangles, classically the sum of all angles is exactly two right angles (180°), but

in hyperbolic geometry, the sum of a triangle is less than two right angles ($< 180^{\circ}$).
Many formulas also differ. The circumference of a circle with ray r is greater than $2\pi r$ in hyperbolic geometry.
The formula for the circumference in hyperbolic geometry is:

$$2\pi R \cdot sinh(\frac{r}{R})$$

and the area formula is:

$$2\pi R^2(cosh\frac{r}{R} - 1)$$

With $R = \frac{1}{\sqrt{-K}}$, and where K is a constant that represents the curvature of the space, in hyperbolic geometry the curvature is negative, how it is possible that the space is curved? Actually, the space we live in is curved too, the phenomena is more apparent close to object of considerable mass, like a star or a black hole.

Figure 2 and 3 are a render of the same scene, the left picture has been rendered using normal Euclidean geometry while the right picture has been obtained using hyperbolic operations. The most noticeable difference is the house, its roof is not parallel to the terrain unlike in the Euclidean render. Another difference is the size of the geometry, objects appear smaller in figure 3.

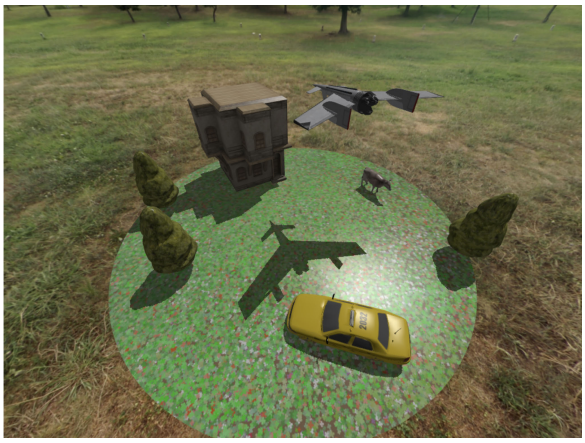


Figure 2. A view of a scene rendered using classical euclidean transformations.

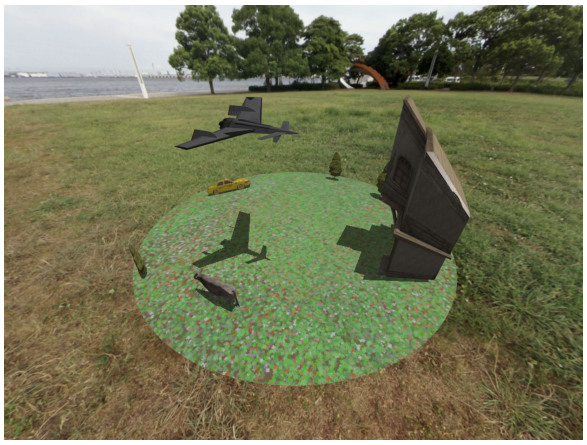


Figure 3. The same scene but rendered using hyperbolic transformations.

1.3 Models of hyperbolic geometry

Different models of hyperbolic geometry exist, all satisfying the postulates, I will list the more common and briefly describe them.

1.3.1 Beltrami-klein model

This is the model that the project use, in this model straight lines are preserved but angles are distorted, meaning that a cube can appear as trapezoidal prism near the border of the 3D hyperbolic world.
Figures 4 and 5 show a cube before and after a hyperbolic translation is applied. The resulting geometry is a trapezoid, the blue face is smaller than the opposite face, for this reason the other adjacent faces are visible.

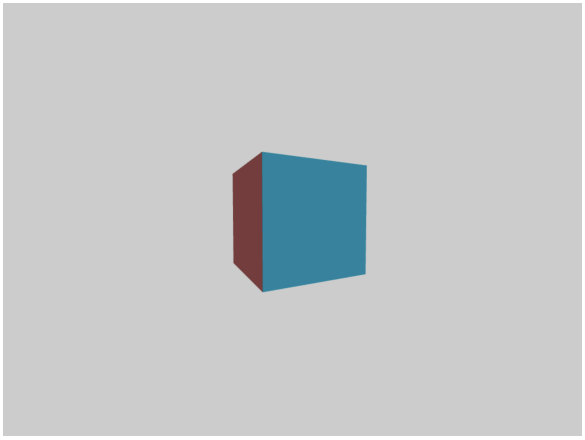


Figure 4. Before the translation the cube is completely normal.

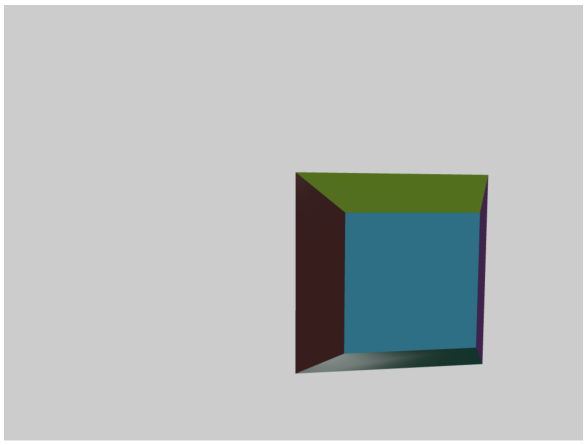


Figure 5. After the translation the cube appear as a trapezoid to an observer situated outside the hyperbolic world 2.1.

In two dimensions this model is a disk and can be obtained with an orthogonal projection of the lower hemisphere of a ball onto a plane below it.

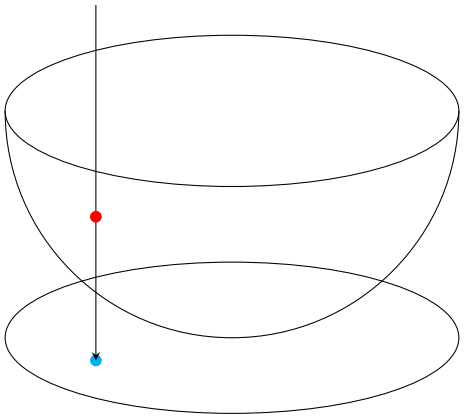


Figure 6. The red point on the hemisphere is projected on the plane.

Figure 6 shows how the projection works. The hemisphere surface contain points, lines and any other two dimensions geometry. The surface is then projected using an orthogonal projection. The result is a flat disk that represent a two dimension hyperbolic disk, the geometry is the same as the surface of the hemisphere but is distorted and appears to shrink.
This model, as the project will show, can be extended in three dimensions.

1.3.2 Poincaré disk model

In this model lines are distorted and become curved, while angles are preserved. The plane model can be obtained by projecting the hemisphere onto the plane by the point situated at the top of the ball.

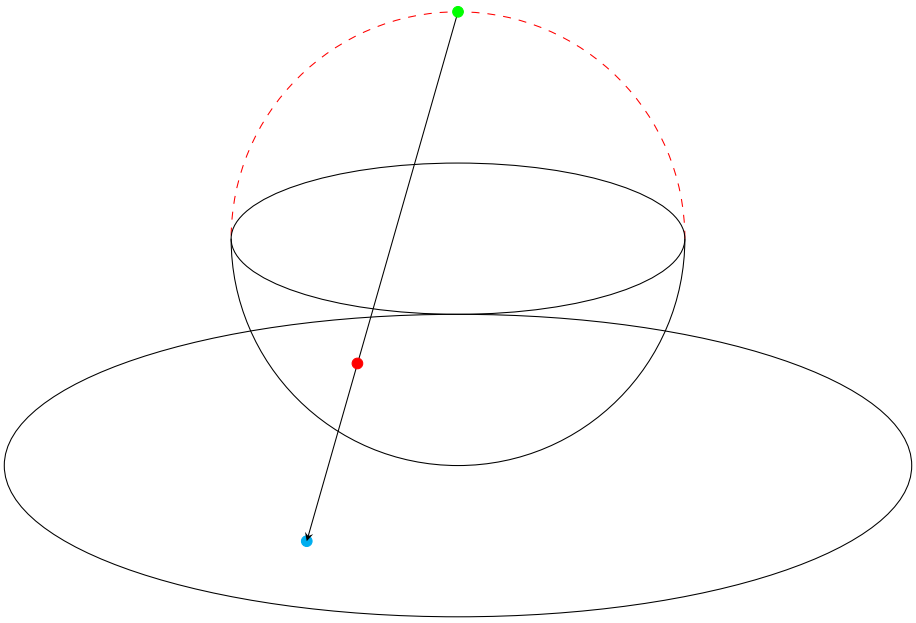


Figure 7. The projection start at the green point and the red point on the hemisphere is projected onto the plane as the blue point.

Figure 7 show a similar concept as figure 6. The difference is that the projection start from a single point on the top hemisphere.

1.3.3 Hyperboloid model

This model is another representation of hyperbolic geometry. Is also called the Minkowski model and is related to the Poincaré disk model and the Beltrami-klein model.

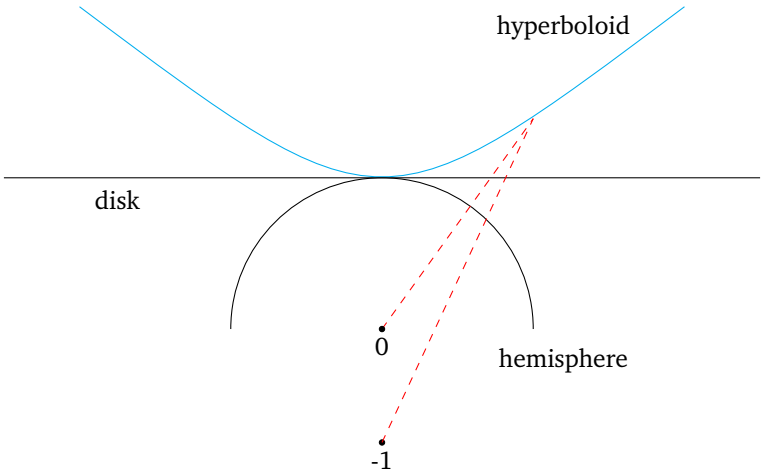


Figure 8. sideview of an hyperboloid and a hemisphere, the dashed lines show the relation between the two in the intersection with the disk.

Figure 8 shows a sideview of a hemisphere, a plane and an hyperboloid, this model is related to the Poincaré disk model. The dashed line from the point -1 to the hyperboloid is the relation between the two models since it intersects the disk at the same point.

1.4 Overview

The paper "Visualizing Hyperbolic Space: Unusual Uses of 4x4 Matrices" [3] presents a simple and straightforward way to compute hyperbolic transformation matrices. I will recapitulate the formulas to compute hyperbolic transformation in the following sections.
Then I will Present the mains problems and challenges I found in Hyperbolic geometry and the solution used, finally

I will also show some interesting visual results obtained.

2 Transformation in hyperbolic space using 4x4 matrices

4x4 matrices are used in computer graphics to compute linear transformations, the five types of transformations are: translation, rotation, scale, reflection and shear. Different transformations can be combined together by multiplying different matrices. For example let T be a translation matrix, R a rotation matrix, S a scaling matrix and p a vector that represent a point, then:

$$T \cdot R \cdot S \cdot p$$

means that the point p is first scaled by S, then rotated by R and finally translated by T. The reason that transformations are represented in matrix form is that GPUs have implemented hardware operations to compute matrices and vectors calculations very efficiently. If hyperbolic transformation are represented by matrices then it is possible to take advantage of the GPU and render hyperbolic geometry as efficiently as Euclidean geometry.

Points in space can be represented by 3-dimensional coordinates (x, y, z). These are called the affine coordinate of the point. In computer graphics points are represented by homogeneous coordinates, vectors of 4-dimensions:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x & y & z & w \end{bmatrix}^T$$

To transform affine coordinates to homogeneous coordinates a fourth element w is added. 1 is added for points while 0 is added if the vector represents a direction. To convert back from homogeneous to affine coordinate divide every element by w, given that $w \neq 0$, otherwise the last element is simply removed.

$$\begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix}$$

In the rendering pipeline points and directions are first converted into homogeneous coordinates, then all the necessary transformations happen, and finally, the result is converted back into affine coordinates. The last step is especially essential in hyperbolic geometry so that every point stays inside the hyperbolic world.

2.1 Hyperbolic space

Before talking about hyperbolic transformation I want to illustrate a property of the hyperbolic model used in the project. In this case the hyperbolic world is enclosed in a sphere with a ray of length equal to 1 and with the centre at the origin. Even if the boundary of the sphere is limited, the volume contained inside is infinite. It might be hard to imagine that a sphere of finite size contain infinite space so I will present a simple example. A person starts at the origin of the world and chooses a direction. Then, the person starts walking toward the border taking a step after another. From his perspective nothing strange is happening. It would be like walking in a normal Euclidean space, but an outsider will quickly notice that each step taken by that person is shorter than the previous, meaning that despite the speed or the time the person walks in the same direction, he will never reach the border.
Figure 9 show the person walking in the hyperbolic space represented by the sphere. The arrows are the steps and the closer they are to the border of the sphere the shorter they become.

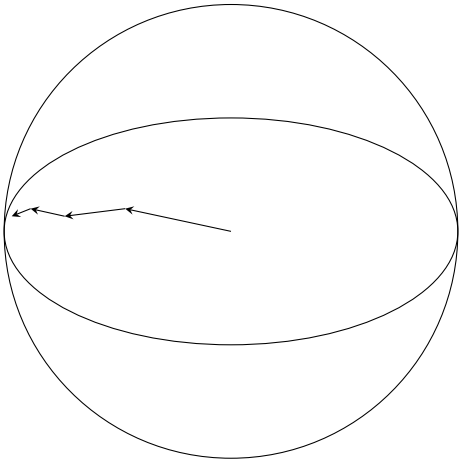


Figure 9. A representation of a person walking in the hyperbolic sphere.

2.2 Conserving the unit sphere

The hyperbolic space is enclosed in a unit sphere. Fortunately all the transformation matrices that will be presented below preserve the unit sphere, meaning that if a point inside the sphere is transformed by an hyperbolic transformation the resulting point will still lay inside the sphere after being converter in affine coordinates. All the points will satisfy the following condition: $\sqrt{x^2+y^2+z^2} < 1$. This property guarantee that the geometry will never lay outside the hyperbolic sphere after a transformations.

Figure 10 shows a sphere made of wire that represent the hyperbolic space, after an hyperbolic translation the cube appears distorted, but it will stay inside the sphere as figure 13 show, figure 11 and 12 are intermediate steps in the transformation.
To generate the hyperbolic transformations matrices it is also important to use points that are inside the unit sphere, if this is not the case and the point is situated outside the hyperbolic world then the computation will fail, for example in the midpoint we will obtain a zero under the squared root.

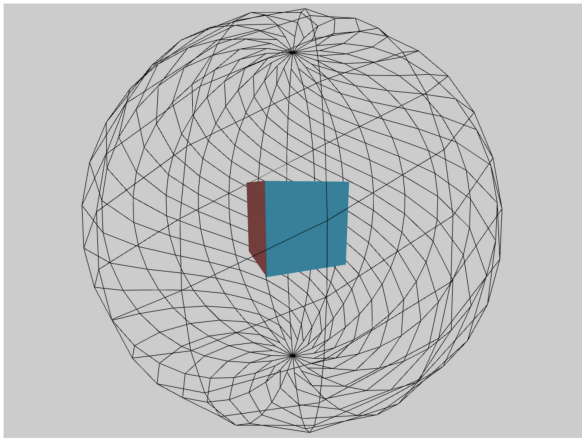


Figure 10. The cube is at the centre of the world and appear normal.

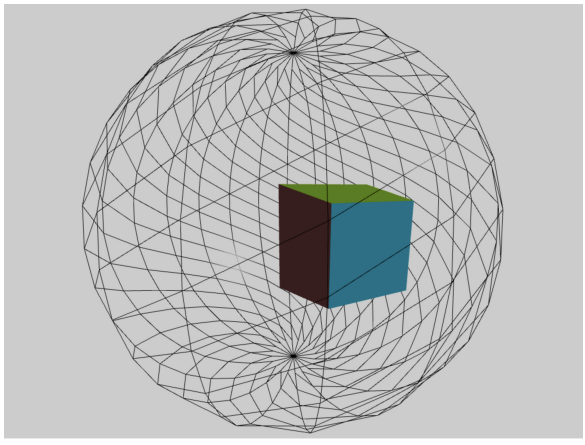


Figure 11. The cube is translated and start to appear distorted.

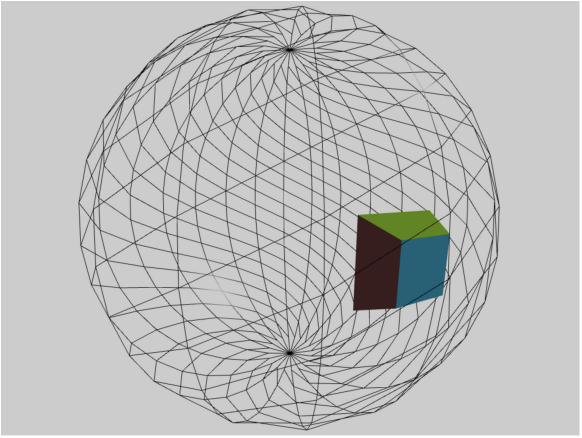


Figure 12. The distortion continue as the cube move towards the border.

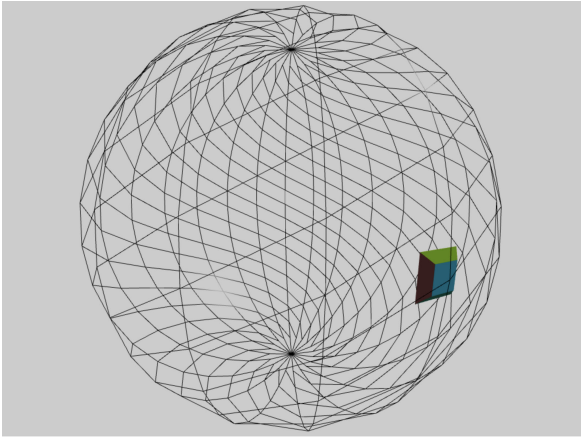


Figure 13. The cube appear also smaller after the translation.

2.3 Hyperbolic inner product

In hyperbolic geometry the inner product between homogeneous vector is calculated differently than in classical Euclidean geometry. Normally the fourth element of the vector is not considered since it usually take the value of 0 or 1, but in non-euclidean geometry the last element can assume different values and in the calculation of the inner product it affect the result.
Let $\mathbf{a}$ and $\mathbf{b}$ be two vectors $\in \mathbf{R}^4$ that represent 3D vector in homogeneous coordinates. Then, the classic inner product is defined as:

$$\langle \mathbf{a}, \mathbf{b} \rangle = a_1 b_1 + a_2 b_2 + a_3 b_3$$

meanwhile the spherical inner product is:

$$\langle \mathbf{a}, \mathbf{b} \rangle_s = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4$$

and the hyperbolic inner product is:

$$\langle \mathbf{a}, \mathbf{b} \rangle_h = a_1 b_1 + a_2 b_2 + a_3 b_3 - a_4 b_4$$

$\langle \cdot, \cdot \rangle_h$ is called the Minkowski inner product and it also can be obtained as: $\langle \cdot, \cdot \rangle_h = \mathbf{a}^T I^{3,1} \mathbf{b}$ with:

$$I^{3,1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

2.4 Reflection

The Reflection matrix is the simplest to compute and it uses the Minkowski inner product.
Let $\mathbf{p}$ be a vector $\in \mathbf{R}^4$ that represent a 3D point in homogeneous coordinates. We can use a 4x4 matrix to compute reflection of any point in the hyperbolic space around the point $\mathbf{p}$ as:

$$\mathbf{r}_h(\mathbf{p}) = I - 2\mathbf{p}\mathbf{p}^T I^{3,1} / \langle \mathbf{p}, \mathbf{p} \rangle_h$$

with I being the identity matrix.
Any point transformed using the matrix $r_h(\mathbf{p})$ will be reflected around the point $\mathbf{p}$ inside the hyperbolic space.

2.5 Translation

A classical translation is computed with a single vector that is added to the transformed point. In hyperbolic space a translation is defined by a starting point and an ending point.
Let $\mathbf{a}$ and $\mathbf{b}$ be two vectors $\in \mathbf{R}^4$ that represent 3D points in homogeneous coordinates, and let $\mathbf{m}$ be the be a vector $\in \mathbf{R}^4$ representing the hyperbolic midpoint between $\mathbf{a}$ and $\mathbf{b}$ and defined as:

$$\mathbf{m} = \mathbf{a}\sqrt{\langle \mathbf{b}, \mathbf{b} \rangle_h \langle \mathbf{a}, \mathbf{b} \rangle_h} + \mathbf{b}\sqrt{\langle \mathbf{a}, \mathbf{a} \rangle_h \langle \mathbf{a}, \mathbf{b} \rangle_h}$$

We can then obtain the 4x4 matrix to define hyperbolic translation from point $\mathbf{a}$ to point $\mathbf{b}$ using the hyperbolic reflection of the midpoint of $\mathbf{a}$ and $\mathbf{b}$ and the hyperbolic reflection of $\mathbf{a}$ as:

$$\mathbf{T}_h(\mathbf{a},\mathbf{b}) = \mathbf{r}_h(\mathbf{m}) \cdot \mathbf{r}_h(\mathbf{a})$$

2.6 Rotation around a point

Rotation in hyperbolic space around an axis l is the same as Euclidean rotation around the same axis. To compute the rotation it is sufficient to translate l with a hyperbolic translation such that it passes around the origin. Then perform a classical rotation around l, and finally translate back l to the original position using the inverse of the first hyperbolic translation.
Let l be a line passing around a and b. In this case a and b are affine coordinate. Also let θ be the angle that we want to rotate and l_0 be the closest point to the origin of l.

$$l_0 = \frac{a \cdot (a-b)}{(a-b)\cdot(a-b)} b + \frac{b \cdot (b-a)}{(b-a)\cdot(b-a)} a$$

Now the hyperbolic rotation is defined as:

$$\mathbf{R}_h(l,\theta) = (\mathbf{T}_h(l_0, origin))^{-1} \cdot \mathbf{R}_{euc}(u,\theta) \cdot \mathbf{T}_h(l_0, origin)$$

where $\mathbf{R}_{euc}(u,\theta)$ is a Euclidean rotation of angle θ, and u is a unit vector in direction of l. The first hyperbolic translation brings l_0 to the origin, while the second translation is the inverse of the first one, so that l_0 is brought back to its original position.

2.7 Lorentz matrix

There is another way to compute translation in hyperbolic space [4], using Lorentz matrices we can achieve translations through the principals axis x, y and z.

$$X = \begin{bmatrix} cosh(d) & 0 & 0 & sinh(d) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ sinh(d) & 0 & 0 & cosh(d) \end{bmatrix}, \quad Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cosh(d) & 0 & sinh(d) \\ 0 & 0 & 1 & 0 \\ 0 & sinh(d) & 0 & cosh(d) \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cosh(d) & sinh(d) \\ 0 & 0 & sinh(d) & cosh(d) \end{bmatrix}$$

Matrix X is a hyperbolic translation along the x axis, d is the distance of translation along the same axis and d $\in \mathbb{R}$, matrices Y and Z accomplish the same but along the y and z axis respectively.
This translation is not very useful since it translates points only along principal axis. Hyperbolic translation in section 2.5 can achieve the same result and in all directions. I implemented Lorentz matrices in the project but I didn't find any situation where they could bring an advantage over the other translations.

2.8 Property of hyperbolic transformations

The conservation of the unit sphere is an extremely noticeable property but is not the only big difference from classical geometry. Hyperbolic translation also contain components of rotation and scaling unlike their euclidean counterpart. If an object is translated towards the border the object will appear distorted and squished to an external observer.

In Euclidean geometry, if we walk left and then up we would reach the same destination as if we walked up and left, so given two Euclidean translation matrices T_1, T_2 and $T_1 \neq T_2$, the following is valid: $T_1 \cdot T_2 = T_2 \cdot T_1$. For hyperbolic translation this is not always true.
To demonstrate this property I will provide a simple example.
Let $T_{Lx}(2)$ and $T_{Ly}(2)$ be two Lorentz matrices with an hyperbolic translation of distance 2 in the x direction and y direction respectively.
Then let $T_{xy} = T_{Lx}(2) \cdot T_{Ly}(2)$ and $T_{yx} = T_{Ly}(2) \cdot T_{Lx}(2)$.
Now let p_o be a point at the origin of the world in homogeneous coordinate.

$$p = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Then compute the transformations:

$$p_{xy} = T_{xy} \cdot p_o, \qquad p_{yx} = T_{yx} \cdot p_o$$

Finally, we can compare p_{xy} with p_{yx}:

$$p_{xy} = \begin{bmatrix} 13.64 \\ 3.62 \\ 0 \\ 14.15 \end{bmatrix} \qquad p_{yx} = \begin{bmatrix} 3.62 \\ 13.64 \\ 0 \\ 14.15 \end{bmatrix}$$

Even without normalising them it is easy to notice a difference. The position of the first and second element are swapped meaning that the two points are in different positions. Once normalised we obtain the following points:

$$p'_{xy} = \begin{bmatrix} 0.96 \\ 0.26 \\ 0 \end{bmatrix} \qquad p'_{yx} = \begin{bmatrix} 0.26 \\ 0.96 \\ 0 \end{bmatrix}$$

It is also easy to notice that both vectors have a length less than 1, meaning that the two points lay inside the hyperbolic sphere.
Figure 14 show the hyperbolic sphere, at the centre are situated two cubes that perfectly overlap so that they appear as one. In figure 15 the two cubes have been transformed using T_{xy} and T_{yx} respectively. After the transformation the positions do not coincide anymore.

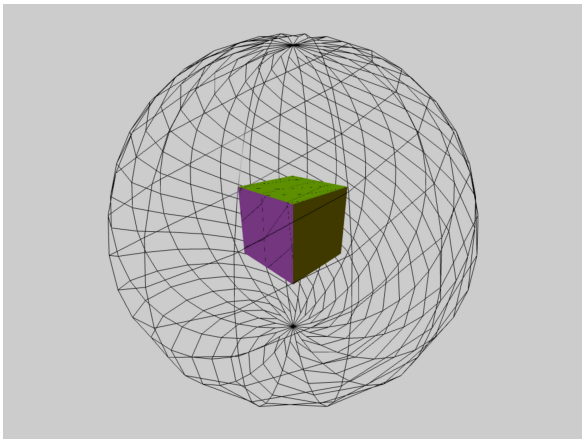


Figure 14. Two cubes before the transformation perfectly overlaps at the origin. The sphere represent the limit of the hyperbolic world.

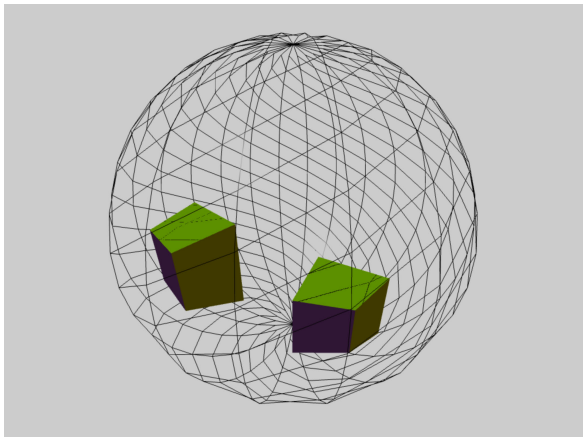


Figure 15. After being transformed by T_{xy} and T_{yx} respectively the two cubes end in different places.

Moving inside the hyperbolic sphere can be achieved using hyperbolic translation, this motion cause some very interesting visual effects, for example figure 17 show a view inside the hyperbolic world of a cube situated at the origin, the wireframe of the sphere appears distorted and the two pole appear in front looking directly at the camera. The lines that connected the north with the south pole now appear distorted and are very reminiscent of magnetic fields. Figure 16 is the same cube but viewed from outside the sphere.

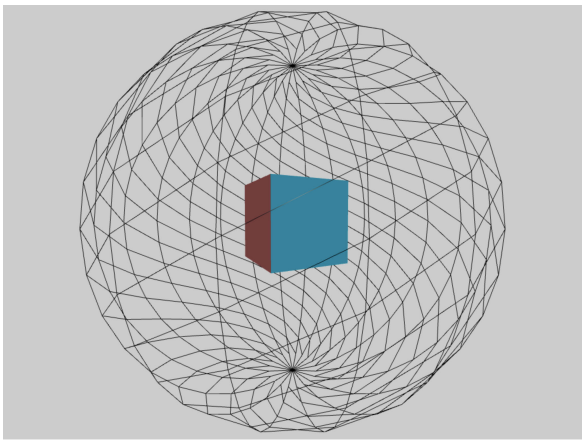


Figure 16. View of a cube inside the hyperbolic space from an outside perspective.

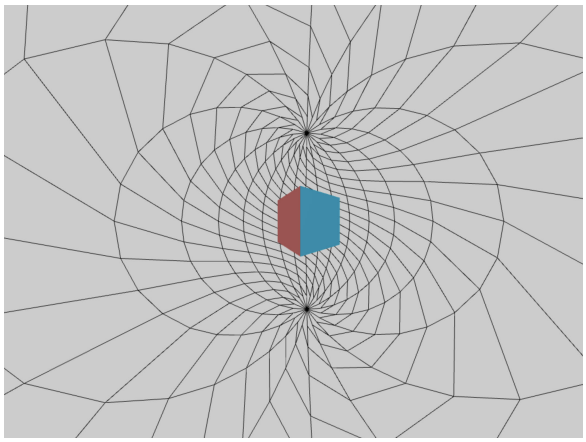


Figure 17. View of the same cube but from inside the hyperbolic sphere.

3 Implementation

The program uses standard HTML5 elements as interface while the logic in the background is written in JavaScript. All the hyperbolic matrices are implemented in JavaScript using the glMatrix framework as base. The 3D rendering is performed using WebGL2.

It might seem a strange choice to use JavaScript since its performance is not as fast as compiled languages like C++, but for the scope of the project javascript is more than enough, and since the project is centred around matrices computation the most heavy calculation are performed by WebGL. So in the end the performance depends mostly on the GPU and not on the choice of the programming language.

3.1 Frameworks libraries and API

3.1.1 glMatrix

This project has as an objective the implementation of hyperbolic matrices and the visualisation of a 3D hyperbolic space. GlMatrix is a JavaScript library designed to perform matrix and vector operations fast and efficiently. It includes a vast set of functions to create and manipulate vectors and matrices of two, three and four dimensions. Everything is made keeping in mind the need of WebGL so all the matrices and vectors can be passed to the shaders easily.
In the hyperbolic implementation, glMatrix is used to compute operation such as multiplication, insertion, scaling and rotation on matrices.

3.1.2 WebGL2

WebGl 2.0 is a JavaScript API to render 3D graphics on any compatible web browser. It is based on OpenGL ES 3.0 and the shaders are written in GLSL (OpenGL Shading Language). Shaders are compiled into GPU code and calculations are performed on hardware making them very efficient.

3.2 Moving inside the hyperbolic space

The classical way of moving in a 3D environment can be achieved in a straightforward way. Two 3D vectors are necessary, one to store the position of the camera, called vector $\vec{p}$, and one to store the direction the camera is looking at called vector $\vec{d}$. To move in the same direction as $\vec{d}$, it is sufficient to add $\vec{d}$ to $\vec{p}$ and save this vector as the new position. The length of the step is decided by the length of $\vec{d}$.
In a Euclidean space, such procedure does not lead to any problems since the world is infinite; however, in the hyperbolic world the same notion is not valid.

3.2.1 Problem with the unit sphere

After a limited number of steps, using the euclidean method, the point $\vec{p}$ will end up outside the world no matter how small the length of $\vec{d}$ is. Having a point outside the unit sphere is first of all undesired, but it also causes problems in the computation of hyperbolic matrices. During the computation of the hyperbolic midpoint we will obtain the value 0 under the square root resulting in the program returning "NaN, or not a number".
The solution I implemented is to make smaller and smaller steps the closer we get to the border. To achieve this effect I used two mapping functions.

3.2.2 Mapping the position

I decided that the camera should only move along a 2D plane to give the impression of "walking" in the hyperbolic space. This also reduces the number of coordinates of the camera since the one representing the height is fixed.
Even with only two coordinates to represent the camera the problem persists. It is still possible to move outside the world and obtain wrong computation. The position of the camera is kept by a vector $\vec{p} = [x, y]$ with $x, y \in \mathbb{R}$, the position is not limited inside a unitary disk but can be as large as JavaScript precision allow. Since these coordinates likely lay outside the unitary disk, they are not directly used to compute the view of the scene. My solution is simple, yet effective. First I use the sigmoid function from figure 18 on both x and y coordinates of the camera to bring them in the range from -1 to 1. Then I apply a second mapping function that guarantees that the position is inside the hyperbolic disk.

The first function map x in a range between -1 and 1. The parameter k with $k > 0$ decide how fast the curve grow, the function tend towards 1 if the arguments are positive and tend toward -1 if the arguments are negative.

$$f(x) = \frac{x}{|x|} \cdot (1 - e^{-k \cdot |x|}) \qquad \begin{cases} \lim_{x \to +\infty} f(x) = 1 \\ \lim_{x \to -\infty} f(x) = -1 \end{cases}$$

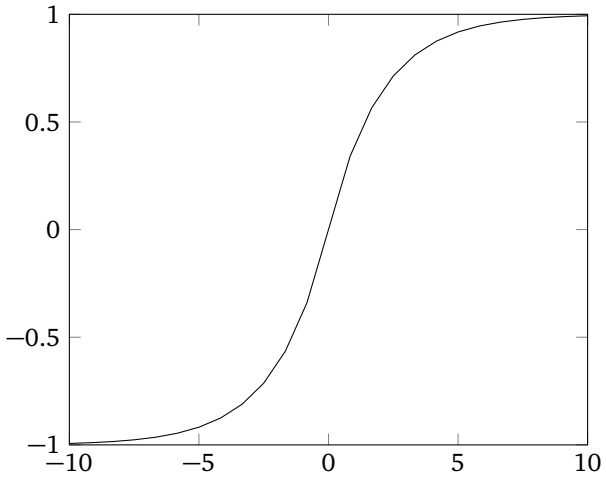


Figure 18. Plot of the sigmoid function.

The first step of the mapping algorithm is to map point (x, y) to (f(x), f(y)). Basically, the sigmoid function f(x) maps an infinite plane in a square of length 2 with the top right coordinate being (1,1) and the bottom left coordinate (-1,-1). This is not enough because there are still points that will fall outside the hyperbolic circle.
It also possible to use the hyperbolic tangent function to achieve the same result, but with this function I have more control on the curve by choosing a different parameter k. Another method to choose the speed of movement of the camera is to simply manipulate the distance of the step during the update of the euclidean position.

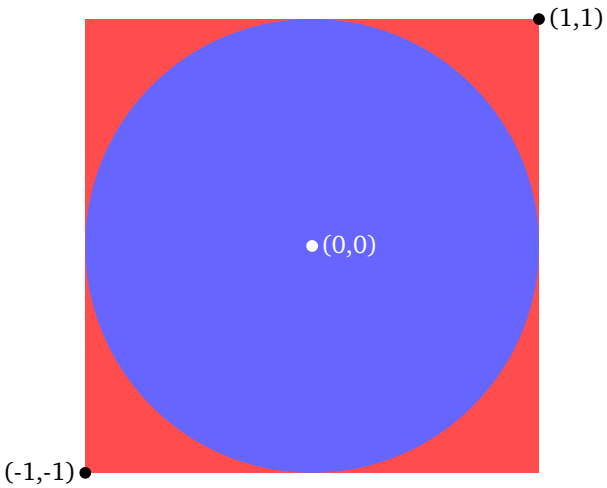


Figure 19. The red square is the mapped plane while the blue circle is the limit of the hyperbolic world.

Another function is needed to map the square into the disk. I choose to use Elliptical grid mapping [2].

$$x' = x\sqrt{1-\frac{y^2}{2}} \qquad y' = y\sqrt{1-\frac{x^2}{2}}$$

x and y are the coordinates on the square in figure 20, while x' and y' are the mapped coordinates on the disk in figure 21.

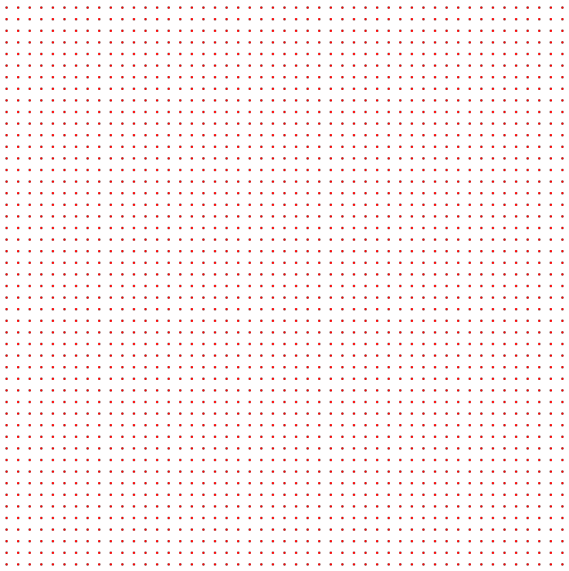


Figure 20. points in the square

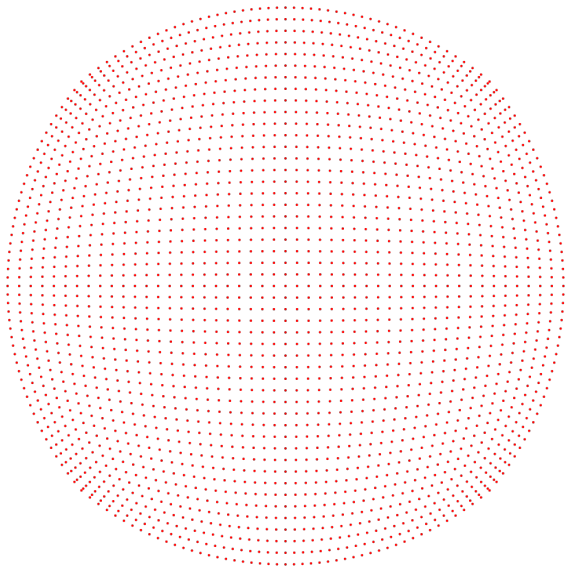


Figure 21. mapped points in the disk

After the coordinates are mapped, I use them in the calculation of the view position, but I never store them. When the camera moves, the program updates the Euclidean coordinates x and y and recomputes the mapping to the new real position.

3.2.3 Method in action

I will provide a simple example to show how the algorithm works. To reduce the number of calculation the camera will move only on the x axis in the positive direction so that only one variable x is needed and y will always be zero. The length of each step is then set to 1, so at every step the position is recomputed as $x = x + 1$.
Now let $\boldsymbol{x}$ be the real position of the camera saved in the system and let $\boldsymbol{x'}$ be its mapped position. To compute the mapped value the parameter k is set to 1. At the beginning $\boldsymbol{x} = 0$ so $\boldsymbol{x'} = 0$ as well. Then, after a step, $\boldsymbol{x} = 1$, and after being mapped, $\boldsymbol{x'} = 0.632$. The second step will update $\boldsymbol{x}$ to 2 and the resulting $\boldsymbol{x'}$ will be 0.865. After another step, $\boldsymbol{x} = 3$ and $\boldsymbol{x'} = 0.95$. This procedure can be repeated and $\boldsymbol{x'}$ will get closer and closer to 1. In the program, only the value of $\boldsymbol{x}$ is saved and updated, while $\boldsymbol{x'}$ is recalculated every frame and is used to computed the view.

3.2.4 Disadvantage

Two problem arise with this method. The first is that it is impossible to map a square into a disk without introducing distortion, as the figure 21 show. There are four areas where a distortion is introduced. These areas correspond to the four corners of the original square. In these regions, the movement might feel strange due to the fact that points are closer between each other than in the rest of the space. In the regions where points are more regularly distributed the movement will feel as natural as moving in the Euclidean space.
The second big problem is introduced by the same nature of hyperbolic space. Even if is contained in a sphere of radius one, and in theory is infinite, in practice computers have a limited precision for floating point numbers. This mean that the world is limited by the precision used by the program. JavaScript represents floating points numbers with a double precision format, 64 bits are allocated for every number. When the position will reach a distance close to 1 from the origin the value will overflow and the mapped position will lay outside the disc causing problems in the matrix computation. Even if finite, the world is still quite large and does not really pose a critical problem for this particular project that aims to give a simpler demonstration of 3D hyperbolic geometry rather that create a detailed and large 3D world.

3.3 View, projection and model matrix

The model matrix can be composed by different transformations and is used to map points from local coordinates to the world space. Basically it can be used to move objects in the space. WebGL camera is always looking towards the negative direction of the z axis so the view matrix maps the world space to the camera space and move objects we wish to visualise in front of the camera. The projection matrix projects the camera space to the view space. A perspective projection is applied to give a 3D perception of the scene, while an orthogonal projection can be used in

shadow map computation.
A point p is transformed applying the following sequence of matrices:

$$p' = Projection \cdot View \cdot Model \cdot p$$

Projection matrices in hyperbolic space work the same as in Euclidean space. GlMatrix already provides functions to compute them and can be immediately used in hyperbolic geometry without changing anything. I used two different view matrices in the project. One transform the world such that the camera looks at the origin of the world from a position outside the hyperbolic world, this version is the same as a Euclidean matrix and glMatrix provides a function even for this. The second view matrix I used gives a view inside the world and is made by combining three transformation:

$$M_{view} = R_X R_Y T_H$$

R_X is a Euclidean rotation on the x axis, R_Y is a Euclidean rotation on the y axis, and T_H is a hyperbolic translation from the origin of the world to the point were the camera is situated. Model matrices in hyperbolic geometry can also be a combination of different transformation just as in Euclidean geometry. We can combine even Euclidean and hyperbolic transformation. Most model matrices that I used are hyperbolic transformation described in section 2.

3.4 Shaders

Shaders are programs written in OpenGL Shading Language. Unlike JavaScript, they need to be compiled into hardware instruction for the GPU. The primary use of a shader is to execute some stages of the rendering pipeline. In my case I used a vertex shader to transform vertices of geometry, normals and light direction and a fragment shader to compute the correct colour output. Hyperbolic geometry requires some small modification while computing directions like normals and light direction, while for computing position no changes are required. Directional vectors are represented by a four element vector with the last element being 0, $d = (d_x, d_y, d_z, 0)$. In Euclidean geometry after a direction is transformed, it is sufficient to remove the last element 0 and then normalise the vector to obtain the correct result. In hyperbolic geometry, a normalisation is needed before removing the fourth element since unlike classical geometry the last element could be different. After this normalisation the vector can be cut and normalised again. Classical geometry:

1) Remove last element 0.
2) Normalise the vector.

Hyperbolic geometry:

1) Normalise the four dimension vector.
2) Remove last element.
3) Normalise the vector again.

This change must be applied only for the computation of direction, for position WebGL automatically transform homogeneous coordinate to affine coordinate so the position will lay inside the hyperbolic sphere and no additional changes are needed.

3.5 ShadowMap

Shadow mapping is a technique to compute shadows in a scene. The concept is quite simple: first render the scene from the point of view of the light source and save the depth of the scene in a texture, then render the scene from the camera prospective. During the second rendering the program will compare the depth of vertices with the texture previously created, and it will decide if a vertex is in in shadow or exposed to light.
In a hyperbolic space it works the same. The only difference is that the entire world is enclosed in a unit sphere. This means that the space to take into consideration is relatively small. The point of view of the camera while rendering the shadow texture is at a distance slightly larger than one, while the projection matrix is an orthogonal projection to simulate directional lighting from the sun.

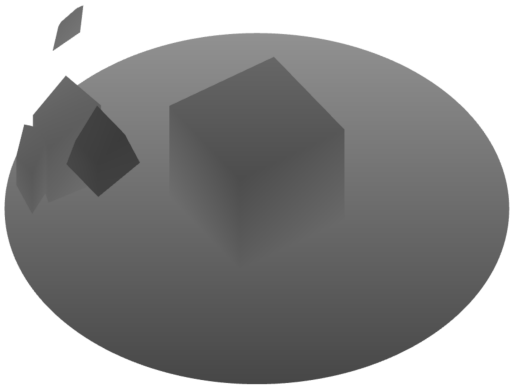


Figure 22. An example of a shadowMap texture, the darker the colour the closer the object is to the camera.

Figure 22 shows the hyperbolic world from the point of view of the light, in this example few objects are present, five cubes and one disk. The closest an object is to the camera the darker is. The background is white since the distance is "infinity".
This method for computing shadow works well if the objects are close to the centre therefore without noticeable distortions. Objects that are closer to the border will appear distorted and shrink, so the shadow will also be smaller than the real size of the object.
Figure 23 shows a view of the hyperbolic world from the outside. The distortion of the taxi inside the red circle is very noticeable. On the other hand figure 24 shows the same taxi but from inside the world, the object appear normal but the distortion of the shadow is still present since when the shadow map was computed the taxi was distorted. Since the shadowMap is saved as a texture it has a limited resolution, meaning that smaller objects might be rendered with an imprecise shadow. The taxi in figure 24 not only has a smaller shadow due to the distortion, but the shadow itself contains artefacts caused by the limited resolution of the shadow texture.

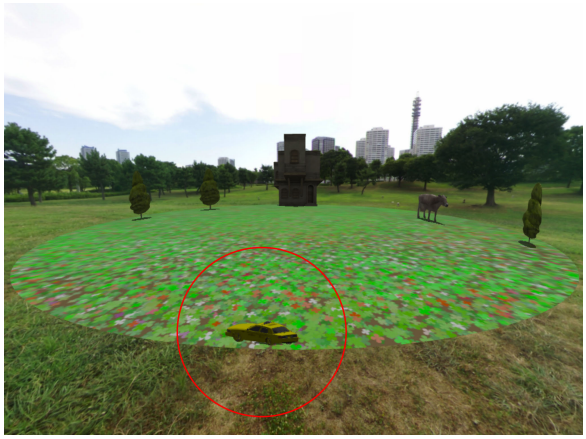


Figure 23. The taxi is distorted and shrink at the border of the world.

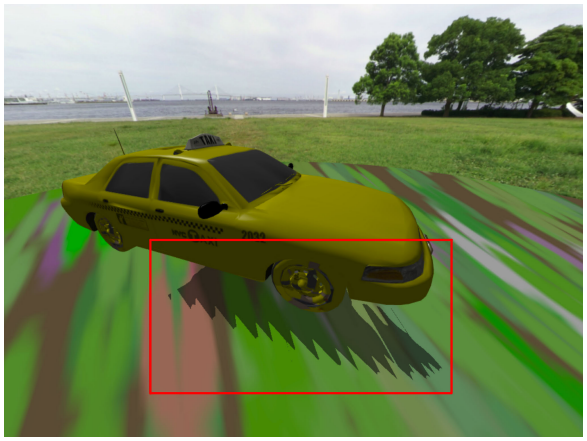


Figure 24. The shadow is distorted while the taxi is not.

4 Visual result

Hyperbolic translations are defined by a starting point and an end point unlike classical transformations that are defined by a single vector. This property causes objects to be warped depending on their relative position to the points that define the translation. Figure 25 is a sideview of a house where no translation was applied, figure 26 is the same house with an hyperbolic translation applied, the red arrow represent the translation and the two green lines show the distortion caused to the roof and the bottom of the house. The starting and ending points of the translation were at half the height of the house. Figure 27 has a hyperbolic translation applied too, but in this case the starting and ending point were at ground level. The translation is represented by the red arrow. In this case only the roof is distorted while the bottom remain unchanged since it overlapped the translation. This show that the

further away a vertex is to the translation the more distorted it will become.

Figure 25. No hyperbolic transformation applied.

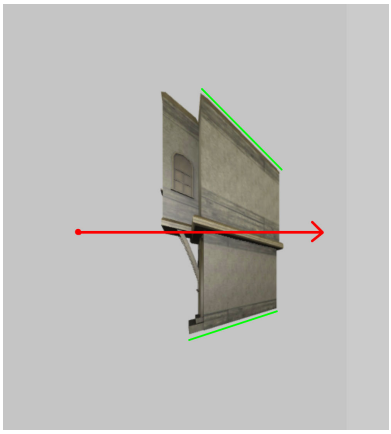


Figure 26. Hyperbolic translation along the red arrow.

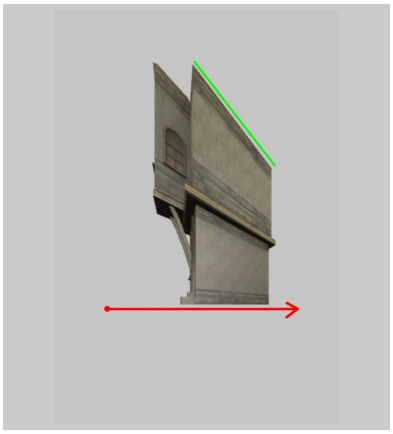


Figure 27. hyperbolic translation along the red arrow.

Other interesting effects are given by hyperbolic rotations, figures 28, 29 and 30 show a rotation of a spaceship around a point, the trajectory is still a perfect circle but the ship is distorted while travelling closer to the sphere limit and becomes normal again while it return toward the centre of the world.

Figure 28. Spaceship rotating around a point using a hyperbolic rotation.

Figure 29. Same view without the scene.

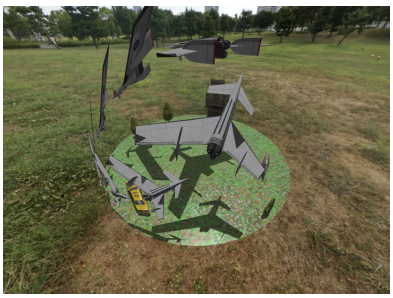


Figure 30. Another view of the same scene.

The last effect I want to present is the distortion introduced while moving further away from objects. Figure 31 shows a house and since it is close it looks normal. Figure 32 is the same exact view but I walked further back, notice how I did not move the position of the camera vertically or horizontally but the world still looks as if it curving. This is because the curvature actually exists, the house is not just getting further but also is being warped by the curvature of the space.

Figure 31. Close view of a house in hyperbolic space.

Figure 32. The same house but view further away.

Screenshots cannot properly convey all the visual effect of hyperbolic geometry. For this reason I prepared a simple scene as demo that can be visualised from the browser. The demo is hosted by GitHub Pages and can be found on this link: https://alessiogiovagnini.github.io/.
WebGl is not compatible with every web browser, and in some case the website might not works. I suggest using Google Chrome or Firefox.

5 Future works

Many other kinds of non-euclidean geometry exist, each with different visual effects. It would be interesting to visualise spherical geometry and compare it with hyperbolic geometry since both are made from a curved space but with different curvatures in opposite directions. Shadow computation should also be explored more in depth to find a solution for the distortion created at the edge of the world.

References

[1] *Main Wikipedia article on hyperbolic geometry.*

[2] M. Lambers. *Mapping between Sphere, Disc, and Square*. University of Siegen, Germany, 2016.

[3] M. Phillips and C. Gunn. *Visualizing Hyperbolic Space: Unusual Uses of 4x4 Matrices*. The National Science and Technology Research Center for Computation and Visualisation of Geometric Structures, 1991.

[4] J. Weeks. *Real-Time Rendering in Curved Spaces*. 2002.