Ray tracing in Non-euclidean spaces

BACKGROUND

Ray tracing is a popular algorithm used in computer graphics to simulate realistic lighting of scenes. It is based on the idea of tracing a ray of light along a scene, calculating its intersection with objects and bouncing it off in a different direction. Normally we use this algorithm in the usual Euclidean space as it represents the real world. However, ray tracing can also be implemented in other (non-Euclidean) spaces to produce interesting visual results. This will be the main task of this project.

Ray tracing algorithm. For the sake of simplicity we will represent our scene with surfaces given by the equations of the form

$$f(x, y, z) = 0. \tag{1}$$

Example 1. Such surfaces are:

(1) Planes given by $ax + by + cz = d$, where $a, b, c, d \in \mathbb{R}$.

(2) Spheres given by $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$, where $(x_0, y_0, z_0) \in \mathbb{R}^3$ is the center of the sphere and $r > 0$ is the diameter. ■

To trace a light ray we choose a starting point $T_0(x_0, y_0, z_0) \in \mathbb{R}^3$ and a direction $\vec{v} = (a, b, c) \in \mathbb{R}^3$. A ray is then represented with the following system of equations:

$$\begin{aligned} x(t) &= x_0 + at, \\ y(t) &= y_0 + bt, \\ z(t) &= z_0 + ct, \end{aligned}$$

where $t \geq 0$ is a parameter. We start with $t = 0$ and increase it continuously in small steps to simulate the movement of the ray. An intersection with an object given by the equation 1 occurs when the sign of f changes. If we find a point of intersection on an interval $[t_1, t_2]$, we can find a more precise point of intersection using the Newton method, for example. We solve the equation We solve the equation

$$g(t) := f(x_0 + at, y_0 + bt, z_0 + ct) = 0$$

with the initial approximation $t = (t_1 + t_2)/2$. The derivative of g that is needed for Newton's method is

$$\begin{aligned} g'(t) = a f_x(x_0 + at, y_0 + bt, z_0 + ct) \\ + b f_y(x_0 + at, y_0 + bt, z_0 + ct) \\ + c f_z(x_0 + at, y_0 + bt, z_0 + ct), \end{aligned}$$

where f_x, f_y and f_z are partial derivatives of f with respect to x, y and z, respectively.

Implementation of ray tracing. We define the position and viewing direction of a "camera" (the point from which we view the scene) and the positions of "pixels" distributed on a plane located at a distance d in front of the camera position. d determines the field of view. We send a ray from each pixel into the scene, and color it depending on the intersection points with objects (or with a default color if there is no intersection point). We can also add a light source to the scene, which can be represented by a single point (this type of light is called *point light*). When we get an intersection point, we send another ray from the intersection point towards the light source to see if there is another object in the way. If this is the case, the original intersection

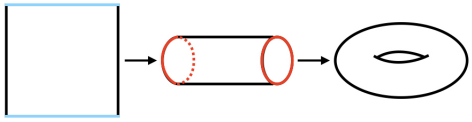

FIGURE 1. Flat torus $\mathbb{T}^2_{\mathrm{f}}$.

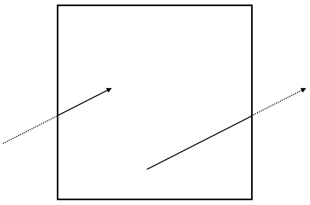

FIGURE 2. A ray in the flat torus $\mathbb{T}^2_{\mathrm{f}}$.

point is in shadow and we can color the pixel with a darker color. This creates shadows that give the scene more realism.

Ray tracing in non-Euclidean spaces.

Flat torus. Flat torus $\mathbb{T}^2_{\mathrm{f}}$ is a simple non-Euclidean space that we get if we *associate* ("glue" together) opposite edges of a unit square $[0,1] \times [0,1] \subset \mathbb{R}^2$ to get the usual torus like shown in the figure 1.

If we imagine a ray traveling in the flat torus, it would travel the same as in the Euclidean space (hence the word "flat" in the name), except it would get "mapped" to the opposite edge when reaching the border, like shown in the figure 2.

We can extend $\mathbb{T}^2_{\mathrm{f}}$ with another dimension to get the 3-dimensional flat torus $\mathbb{T}^3_{\mathrm{f}}$. Here we associate opposite faces of the unit cube instead of the edges.

2-sphere. For ray tracing on the 2-sphere $\mathbb{S}^2$ living in $\mathbb{R}^3$ we need to trace a ray along the surface of a sphere. For this we will introduce *geodesics*, which are the straightest paths between points on a surface. In general, this does not coincide with the shortest paths, however in case of the sphere it turns out that one of the two geodesics is also a shortest path. For a plane this would be a line and for a sphere it would be a curve on its surface.

Let us have a sphere in $\mathbb{R}^3$ parametrized with coordinates $(u, v) \in \mathbb{R}^2$ as

$$\begin{aligned} X &= \cos(v)\sin(u), \\ Y &= \sin(v)\sin(u), \\ Z &= \cos(u). \end{aligned}$$

We choose a starting point on the sphere $P = (u_0, v_0) \in \mathbb{R}^2$ and some direction $(v_1, v_2) \in \mathbb{R}^2$. We derive a system of differential equations (DEs) that allows us to make a small step along a geodesic on the sphere. The system of DEs is

$$\begin{aligned} \frac{d^2u}{dt^2} - \cos(u)\sin(u)\frac{dv}{dt}\frac{dv}{dt} &= 0 \\ \frac{d^2v}{dt^2} - 2\cot(u)\frac{du}{dt}\frac{dv}{dt} &= 0, \end{aligned} \tag{2}$$

where (u, v) is the current position, $\left(\frac{du}{dt}, \frac{dv}{dt}\right)$ is the direction of travel and $\left(\frac{d^2u}{dt^2}, \frac{d^2v}{dt^2}\right)$ is the "speed" of travel. If you are interested in the full derivation take a look at the videos [1, 2].

We solve the system of equations 2 with a numeric method, for example Euler's method. However, for this aim we first need to transform it into a system of four first order differential equations.

To perform ray tracing on the 2-sphere use the following steps:

(1) Transform a point from $\mathbb{R}^3$ to the uv-plane.

(2) Perform one step of Euler's method on the system 2 using the current point and the initial approximation $\left(\frac{du}{dt}, \frac{dv}{dt}\right) = (1, 0)$.[1]

(3) Transform the new point from uv-plane to $\mathbb{R}^3$.

(4) Calculate intersections as described before.

Here are some additional tips you should know:

- To achieve a true 3-dimensional rendering of the scene you should trace rays along different spheres (one sphere for each row of pixels should work well).
- A ray never leaves a certain sphere, so it might also never hit a light source even if it is not obstructed by another object. To avoid this we simplify the problem by tracing rays towards light sources in the standard Euclidean fashion.

TASK

(1) Implement the basic ray tracing algorithm. Your program should be able to render a simple scene with basic shading as described above. Any other extensions, such as reflections and soft shading, are optional, since they are not the focus of this project.

(2) Implement ray tracing for the 3-dimensional flat torus $\mathbb{T}^3_{\mathrm{f}}$.

(3) Implement ray tracing for the 2-sphere $\mathbb{S}^2$. **Suggestion:** first just try to plot one geodesic using the method described above and then move to ray tracing.

(4) Render a scene in Euclidean space and the presented non-Euclidean spaces and visually compare the results. What are the characteristics of each space? If you find any interesting behaviours report on them and try to explain them.

(5) **Optional:** implement ray tracing for some other "flat" space. One idea is the *mirrored cube*, where we take the unit cube, think of its faces as mirrors and place the scene inside of it. You can also come up with a space of your own and produce some interesting results.

REFERENCES

[1] eigenchris, *Tensor Calculus 15: Geodesics and Christoffel Symbols (extrinsic geometry)* `https://www.youtube.com/watch?v=1CuTNveXJRc`, Accessed 16. 2. 2024

[2] eigenchris, *Tensor Calculus 16: Geodesic Examples on Plane and Sphere,* `https://www.youtube.com/watch?v=8sVDceI70HM&t=470s`, Accessed 16. 2. 2024

[1] Euler's method tends to make big numerical errors when using other initial approximations. If we use an adaptive method like Dormand-Prince 5 we can choose any direction, but you don't need to do that for this project.