Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Mathematical Modelling

Group Project

Report

Authors
Kim Ana badovinac,
Katja Logar, Jernej Vivod

Lecturers and mentors
Dr. Neža Mramor Kosta,
Dr. Peter Marijan Kink

June 10, 2018

Contents

1 Problem Statement

In this group project we were tasked with implementing a simple ray tracing model capable of simulating ray refraction cause by an optical lens represented by an implicit function. We implemented the project using the MATLAB programming language and also provided a simple user interface for interacting with the simulation.

2 Screens and Rays

We started work on the project by implementing the two screens that were to provide the means to trace and visually represent the path of rays and how the change of the path changes the provided image.

Both screens were represented as a point in space corresponding to the upper left corner and two vectors spanning the parallelogram representing the screen. The rays were modelled as a parametrized line function.

The natural next step was to compute line equations each representing a ray going from a point representing a light source and through each pixel of the image on the first screen represented as a 2 dimensional matrix. This was achieved by dividing the parallelogram representing the screen into squares of equal size and getting the coordinates of the centres of each square.

This was achieved using the functions *get_ pixel_ coordinates* and its auxiliary function *get_ coordinates.* The function *get_ pixel_ coordinates* returns a cell array containing at indices pair (i, j) the coordinates of the pixel with indices (i, j). The source code for these two functions can be found in the screens folder.

This matrix of line functions represents a sort of interface between the three main modules of the program. The first module generates, with respect to light source position and first screen position, size and shape, the cell array of line functions representing the rays as they leave each pixel of the first screen. This matrix serves as the input to the second module performing the actual lensing and, modified by the second module, is also the input to the third module that then actually computes the intersection coordinates and the indices of pixels that were intersected by the rays on the second screen.

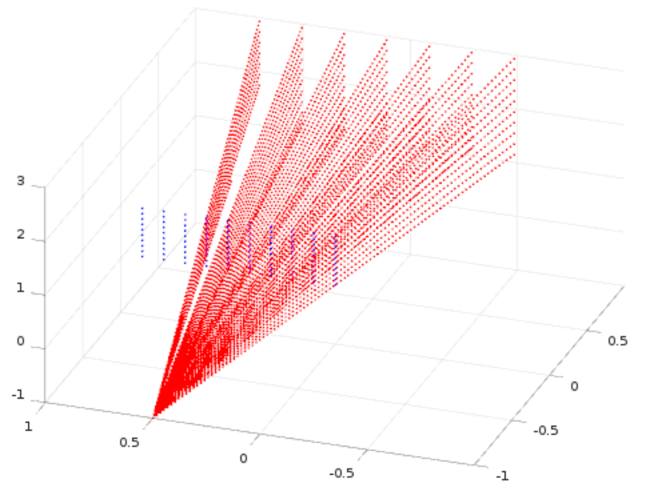

Figure 1: Visualizing the Rays Passing from the Light Source Through each Pixel on the Second Screen

3 Computing Intersections with the Second Screen

This section describes the workings of the so called third module that takes the cell array of line functions representing the rays processed by the second module (which applies lensing). This is a well known problem of computing the intersection of a line with with a plane spanned by the two vectors representing the second screen.

A point on the plane can be represented as a linear combination given by

$$P_0 + P_{01}u + P_{02}v, u, v \in R$$

Where P_{01} and P_{02} are vectors from a point on the plane to the tips of the spanning vectors u and v respectively.

The point at which the line intersects the plane s therefore described by setting the point on the line equal o the point on the plane, giving the parametric equation

$$l_a + l_{ab}t = P_0 + P_{01}u + P_{02}v$$

Where l_a is a point on the line and l_{ab} is the line's directional vector.

Rewritten, these equations can be expressed in matrix form as

$$\begin{bmatrix} l_a - P_0 \end{bmatrix} = \begin{bmatrix} -l_{ab} \; P_{01} \; P_{02} \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix}$$

This system of of linear equations can now be solved for t, u and v. The point of intersection can then be computed by plugging the solution for t back into the line equation. We can also check if the intersection actually happened on the screen, that is, the parallelogram spanned by the two vectors. This fact can be establishing by inspecting the results for the spanning vector scaling coefficients that we got by solving the system of linear equations. If they are elements of $[0, 1]$ the intersection happened on the parallelogram and hence the second screen.

For getting the actual indices of the pixels that were intersected the second screen can be divided in a similar fashion to what was done to the first screen when we computed the coordinates of each pixels. Performing a kind of inverse computation we can use similar reasoning to get the index pair (i, j) from the coordinates of intersection.

This allows us to get a mapping from pixels on the second screen to the pixels on the first screen which consequently makes it possible to assign to the pixels on the first screen the color of the pixel that was intersected by the ray corresponding to this pixel on the second screen.

4 Refraction

When light passes through media with different indices of refraction, it refracts according to Snell's law. Snell's law tells us that the ratio of the sines of the angles of incidence and refraction is equivalent to the ratio of the indices of refraction:

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{n_2}{n_1}$$

In the above equation, θ_1 in the angle of incidence (the angle between the ray of incidence and the normal to the surface), θ_2 in the angle of refraction (the angle between the ray of refraction and the normal to the surface), n_1 is the refractive index of the incoming medium and n_2 is the refractive index of the outcoming medium.

In our project we notice refraction when the ray enters the len and when it exits the len.

Refraction is implemented in function $refraction.m$. Instead of the scalar form of the law, we use Snell's law in vector form:

$$\vec{s_2} = \frac{n_1}{n_2}(\vec{n} \times (-\vec{n} \times \vec{s_1})) - \vec{n}\sqrt{1 - (\frac{n_1}{n_2})^2(\vec{n} \times \vec{s_1})(\vec{n} \times \vec{s_1})}$$

Here $\vec{s_2}$ is the vector we seek, meaning it is the refracted vector, $\vec{s_1}$ is the incident vector and $\vec{n}$ is the normal, that we compute by computing the gradient of the function of the len at the intersection.

It is important to note, that the above formula assumes, that both $\vec{n}$ and $\vec{s_1}$ are vectors with the norm equal to 1. Therefore, before we compute $\vec{s_2}$, we need to divide $\vec{n}$ and $\vec{s_1}$ by their norm.

Also, when computing the refracted ray, it is important to know, whether we are entering the len or exiting it. That is because the first part of the above equation: $\frac{n_1}{n_2}(\vec{n} \times (-\vec{n} \times \vec{s_1}))$ contains $-$ in $(-\vec{n} \times \vec{s_1})$, which takes into consideration that we should actually take the negative normal, when entering the len, since the negative normal points in the same direction (towards the center) as the incident ray. Therefore, the $-$ in $(-\vec{n} \times \vec{s_1})$ is not needed when we are exiting the len.

One more thing that we have to watch out for is total internal reflection. It can happen for rays that are crossing into a less dense medium ($n_2 < n_1$) - in our case that happens when light exits the len. In that case, if the incident angle is big enough, the light won't refract, it will reflect. That happens for angles bigger than the critical angle.

4.0.1 Alternative method for computing the refracted ray

We can compute the orthogonal projection of the incident vector to the normal of the surface:

$$\vec{p} = \frac{\vec{s_1}\vec{n}}{\vec{n}\vec{n}}\vec{n}$$

Then we compute the vector $\vec{u}$:

$$\vec{u} = \vec{s_1} - \vec{p}$$

The refracted ray is then:

$$\vec{s_2} = \vec{p} + \frac{n_1}{n_2}\vec{u}$$

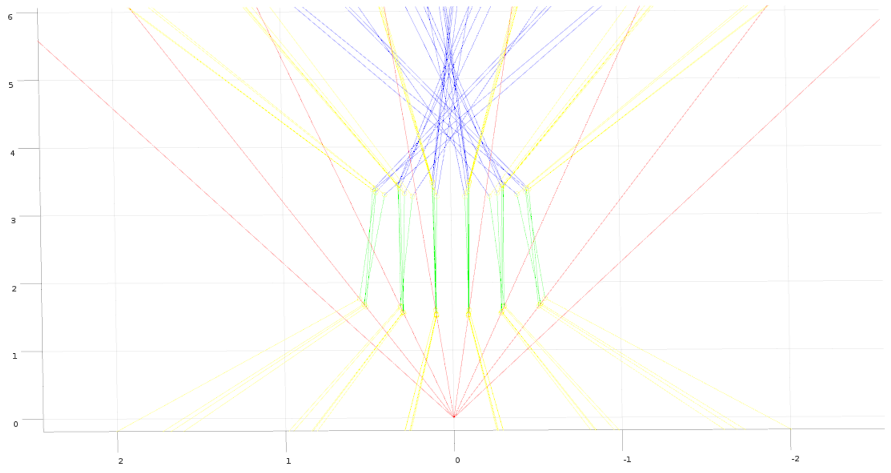

Figure 2: Visualizing the Rays entering a Spherical lens

5 Lenses

A lens is given by an implicit surface

$$F(x, y, z) <= 0$$

and some refractive index. For our project we chose 4 surfaces:

- ellipsoid,
- torus,
- heart surface,
- bifolia.

For each surface we wrote a Matlab function accepting the position and stretch vector of the surface and returning its function handle. The two codes are at the end of the report.

5.1 Ellipsoid

An ellipsoid may be obtained from a sphere with directional stretching. The standard equation is

$$F(x, y, z) = (\frac{x - x_0}{a})^2 + (\frac{y - y_0}{b})^2 + (\frac{z - z_0}{c})^2 - r^2,$$

where x_0, y_0, z_0 are the center coordinates, a, b, c are the directional stretch factors and r is the radius of the stretched sphere.

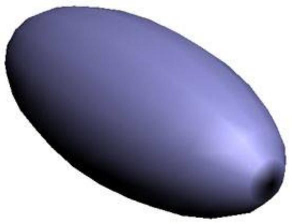

Figure 3: Ellipsoid

5.2 Torus

A torus may be generated by revolving a circle. The standard equation is

$$F(x,y,z) = (R - \sqrt{(x-x_0)^2 + (y-y_0)^2})^2 + (z-z_0)^2 - r,$$

where x_0, y_0, z_0 are the center coordinates, R is the radius from the hole in the center and r is the radius of the torus tube.

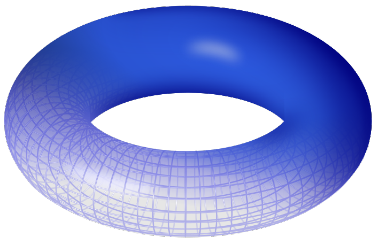

Figure 4: Torus

5.3 Heart surface

A heart surface can be given by slightly different functions. An example is

$$F(x,y,z) = (2\cdot(\frac{x-x_0}{a})^2 + 2\cdot(\frac{y-y_0}{b})^2 + (\frac{z-z_0}{c})^2 - 1)^3 - \frac{1}{10}\cdot(\frac{x-x_0}{a})^2\cdot(\frac{z-z_0}{c})^3 - (\frac{y-y_0}{b})^2\cdot(\frac{z-z_0}{c})^3,$$

where x_0, y_0, z_0 are the center coordinates and a, b, c are the directional stretch factors.

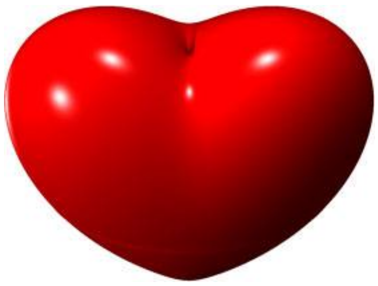

Figure 5: Heart Shape

5.4 Bifolia

A bifolia is given by a function

$$F(x,y,z) = (\frac{x-x_0}{a})^4 + (\frac{y-y_0}{b})^4 + (\frac{z-z_0}{c})^4 + 2\cdot(\frac{x-x_0}{a})^2\cdot(\frac{y-y_0}{b})^2 + 2\cdot(\frac{x-x_0}{a})^2\cdot(\frac{z-z_0}{c})^2$$
$$+2\cdot(\frac{y-y_0}{b})^2\cdot(\frac{z-z_0}{c})^2 - 3\cdot\frac{y-y_0}{b}\cdot(\frac{x-x_0}{a})^2 - 3\cdot\frac{y-y_0}{b}\cdot(\frac{z-z_0}{c})^2,$$

where x_0, y_0, z_0 are the center coordinates and a, b, c are the directional stretch factors.

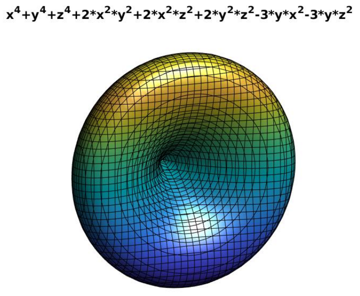

Figure 6: Bifolia

6 Intersection of a ray with an implicit surface

We want to find the possible intersection of each ray

$$r(t) = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} a \\ b \\ c \end{bmatrix} \cdot t$$

with the implicitly given lens

$$f(x, y, z) <= 0.$$

There may be zero, one or multiple intersections. Because we compute the ray's refraction and continue working with the new ray, we only need to find the next intersection. The way we find the intersection is by manually traveling on the ray and getting closer to the lens's surface and then using the Newton's method. The intersection is detected when for two values of t, the function

$$f(r(t)(1), r(t)(2), r(t)(3))$$

changes its sign which means the ray entered or exited the lens. We start approaching the possible solution from the starting value $t = 0.01$. After saving the initial sign, we increase t by 0.01 until the sign changes or t is bigger than 5. This is an interval $t \in [0.01, 5.0]$ where the lens is positioned. If we surpass this interval, we did not find the intersection. But if we stay in this interval and the sign of the function changes, we can now use the Newton's method with the initial guess for the zero $t - 0.005$.

Newton's method quickly finds a good approximation for zeroes of real-valued functions. It tells us that a better approximation for the zero is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

We repeat this calculation as many times as necessary for the desired accuracy and we get a series of approximations which quickly converge to the zero of the function. Our initial guess x_0 is the value t calculated from before. We write the value 10^{-13} for the tolerance for declaring convergence and value 100 for the maximum number of iterations. The base of the Newton's method is a for loop which iterates from 1 to maximum number of iterations. The loop assigns the previous value t, then uses the equation above to get the next value of t. If the absolute difference of these two values is smaller than the tolerance, the convergence is apparent and we found our solution.

7 Results

This section presents some images that were obtained by running the program. The program is configured to be able to produce both images with applied lensing and also plots of the light source, screens and lens configuration that produced these results.

An extended collection of the results can be found in the results folder in the the project folder.

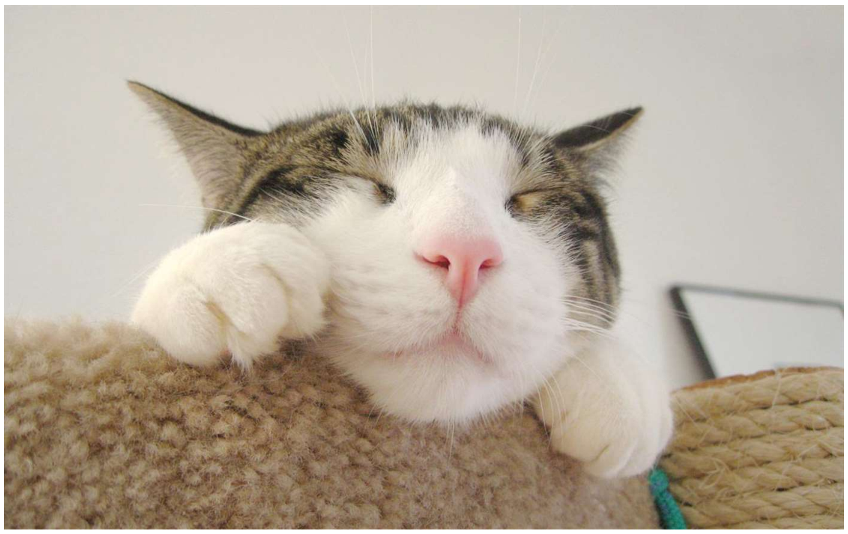
Figure 7: The Original image of a Cat used for Testing Purposes

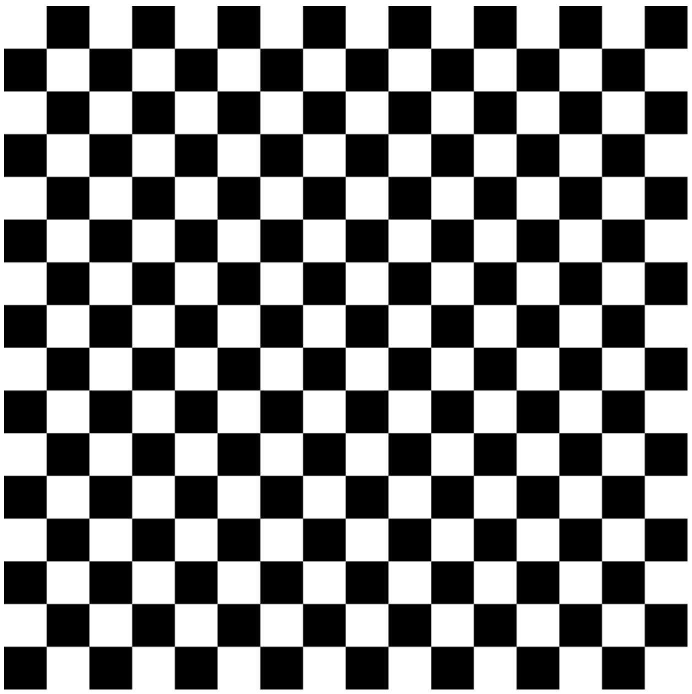
Figure 8: The Original Image of A Checkers Pattern used for Testing Purposes

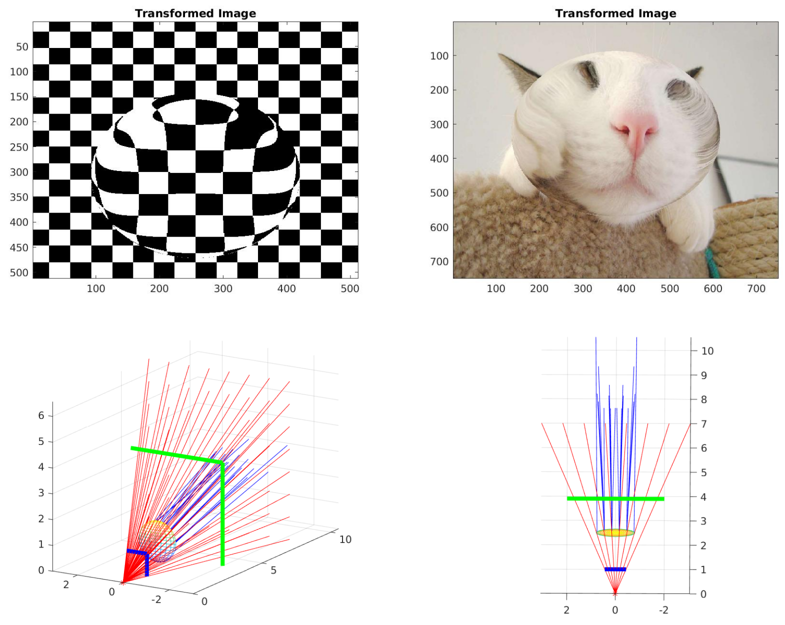

Figure 9: Applying an Ellipsoidal lens with an 0.2 Scaling factor in the x Direction

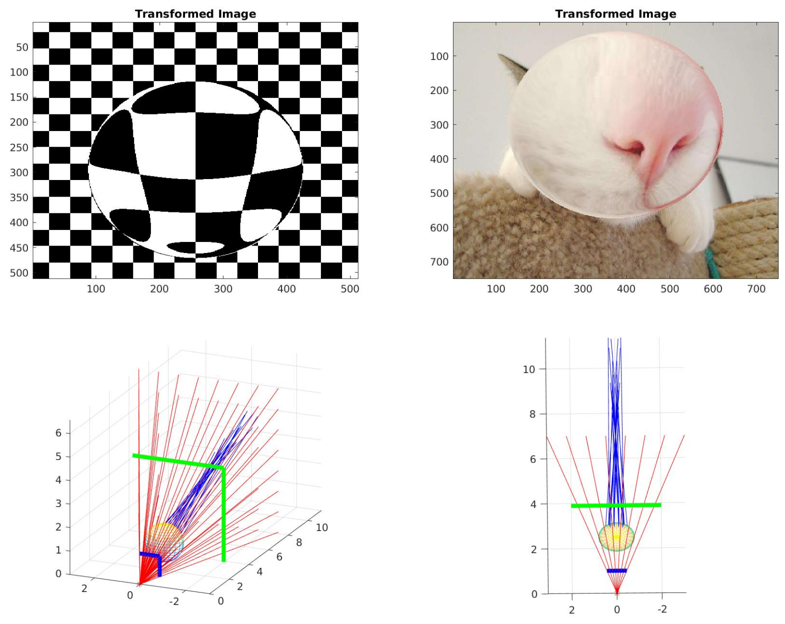

Figure 10: Applying an Ellipsoidal lens with an 0.8 Scaling factor in the x Direction

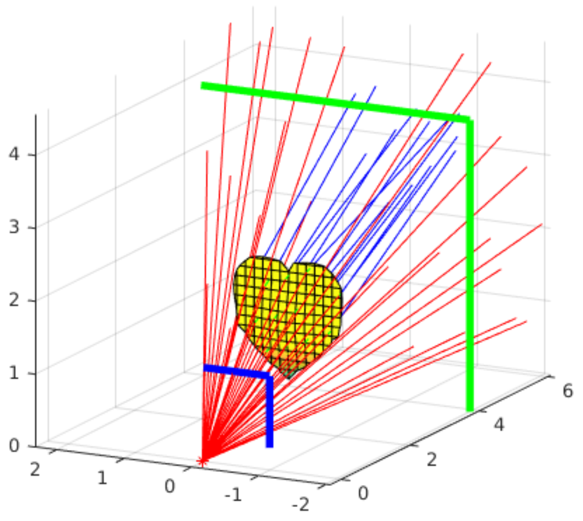

Figure 11: Visualizing the Screens and The Taubin's Heart lens

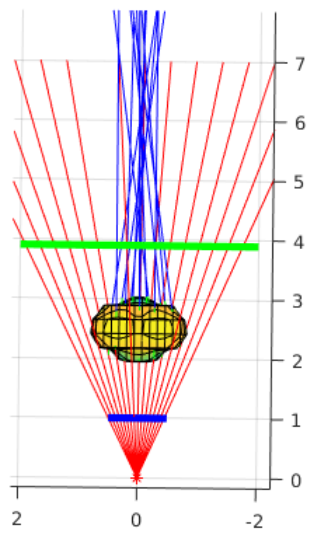

Figure 12: Top Down View of the Screens and The Taubin's Heart lens

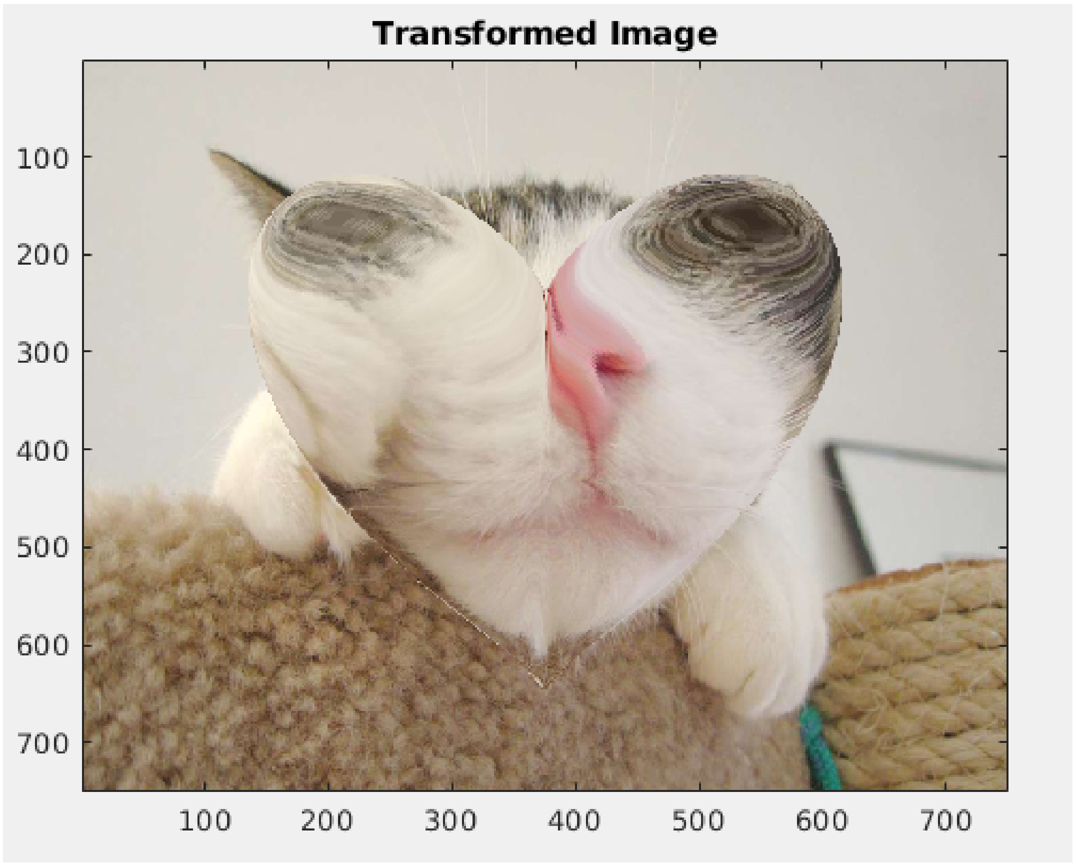

Figure 13: The Cat Image transformed through a Taubin's heart scaled by a factor 0.2 in the x direction

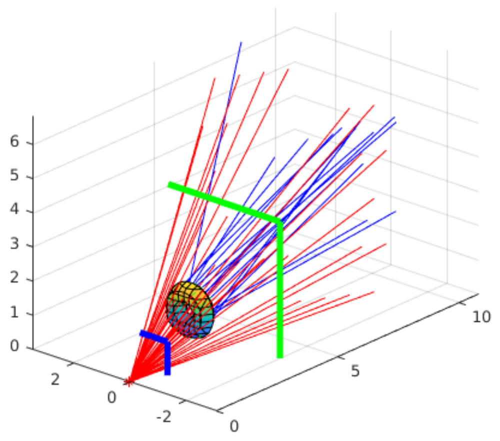

Figure 14: Visualization of the light source, screens and lens configuration

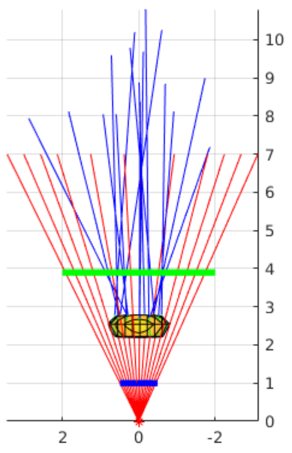

Figure 15: Top Down View of the light source, screens and lens configuration

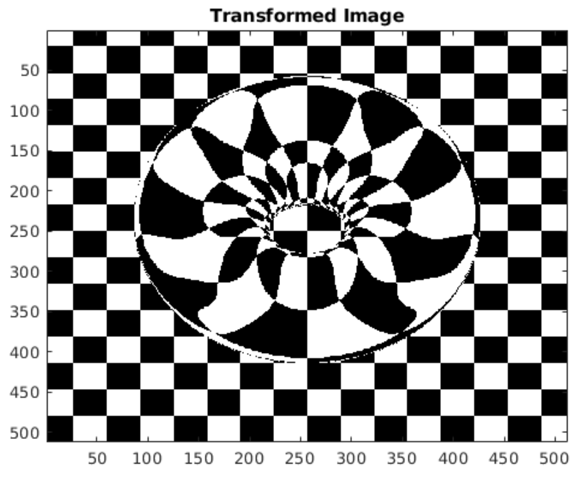

Figure 16: Applying Torical lens to the Checkers pattern

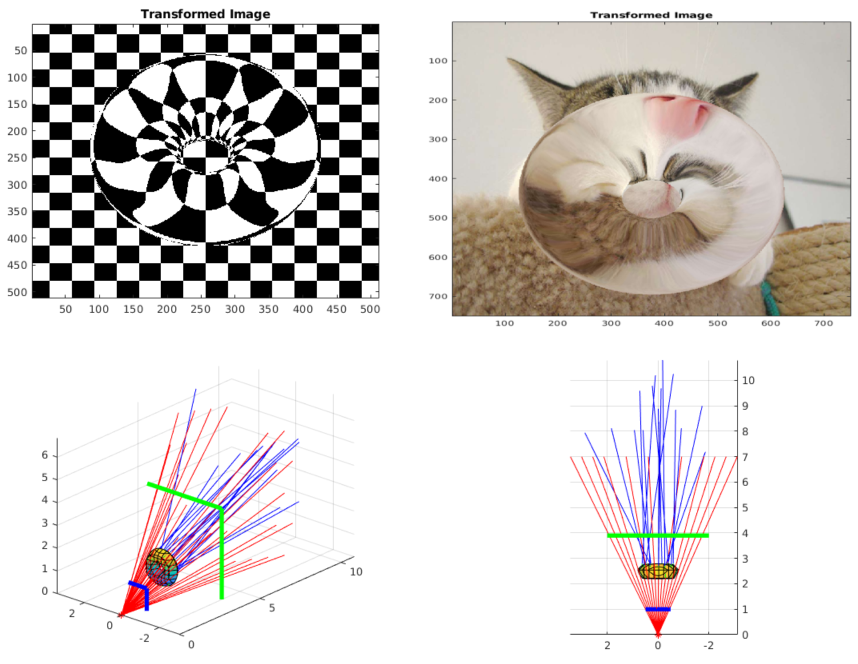

Figure 17: Results of Applying torical lens

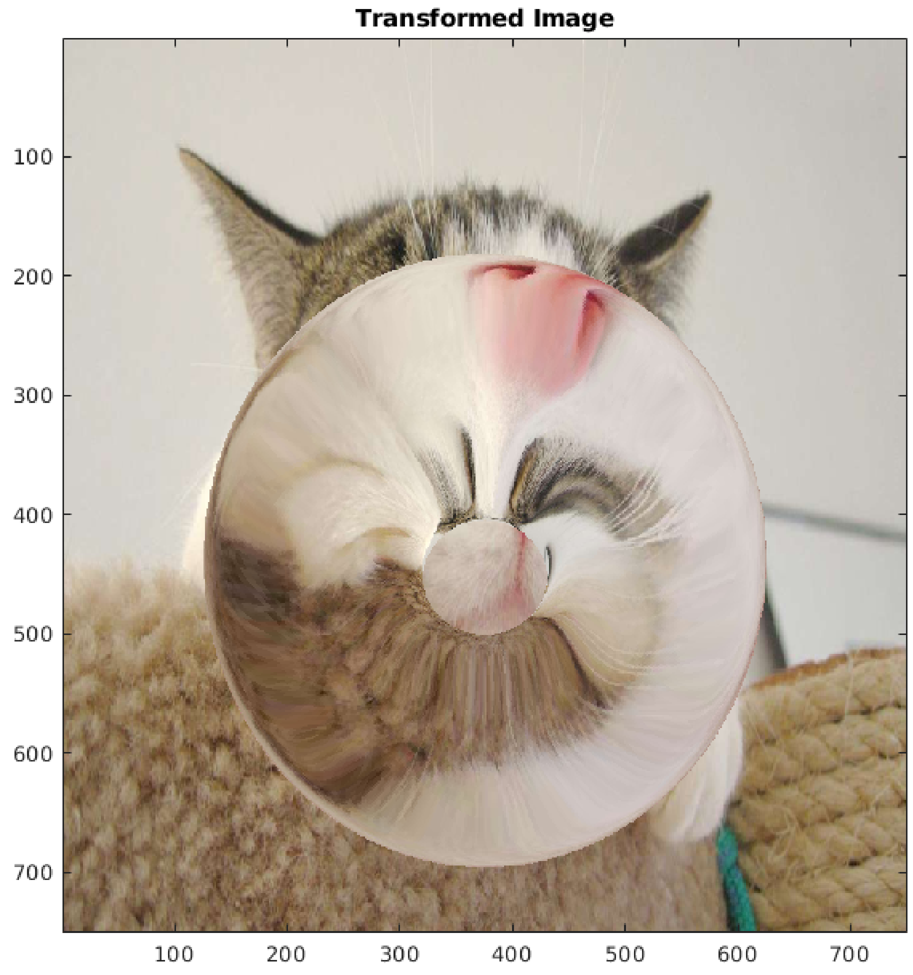

Figure 18: Image of a Cat Lensed with a torical lens

8 Sources and Workload Distriution

8.1 Workload Distribution

Kim Ana Badovinac - Lens equations and representation, intersections with lens, Newton-Raphson method, report

Katja Logar - Refraction computation, applying lensing to rays, report

Jernej Vivod - Screens, intersections with screens, animations and image transformations, user interface, report

8.2 Sources

- https://en.wikipedia.org/wiki/Refraction
- https://en.wikipedia.org/wiki/Snell
- http://www.starkeffects.com/snells-law-vector.shtml
- Zakrajsek Egon. Matematicno Modeliranje. DMFA zaloznistvo, 2004.
- Strang, Gilbert. Introduction to Linear Algebra. Wellesley-Cambridge Press, 2016.
- https://en.wikipedia.org/wiki/Lens_(optics)
- https://en.wikipedia.org/wiki/Implicit_surface
- https://en.wikipedia.org/wiki/Ellipsoid
- https://en.wikipedia.org/wiki/Torus
- http://mathworld.wolfram.com/HeartSurface.html
- https://en.wikipedia.org/wiki/Newton