university of groningen

faculty of science and engineering

Virtual Ray Tracer: Using Light to Deform Objects

Bachelor's Thesis Project
Computing Science

Author: Irina Bodola
First supervisor: Prof. Dr. Jiří Kosinka
Second supervisor: Dr. Steffen Frey

University of Groningen
4 August 2024

Abstract

Anamorphism requires that a viewer needs to have a certain position or use special devices to look at an image or object in order to see a recognisable picture.

Anamorphic art has been around for centuries but it has always been difficult to create due to its complex mathematical calculations. With the advancement of modern technology, new methods to create such art have been created. One such method uses ray tracing to build anamorphic sculptures based on an object's projection in a mirror or refraction through a transparent medium.

Virtual Ray Tracer is a tool developed to teach ray tracing. Given its usefulness to students and the intriguing method to produce art described above, this paper explores the possibility of making a tool meant to teach the way anamorphic sculptures are constructed.

Contents

1 Introduction

Anamorphosis represents the process of projecting images and objects outside of their usual proportions. When these pieces are then looked at from a specific point of view, they return to normal [7]. The concept of bending an object out of its proportions using perspective, reflection, and refraction has fascinated artists for centuries. With the evolution of modern technology, new ways to fashion such art work have been discovered. One method in particular uses ray tracing to build 3D anamorphic sculptures [17].

Ray tracing is one of the most popular rendering algorithms in Computer Graphics. It takes a 3D scene composed of many objects and lights and creates an array of pixels that translates to a 2D picture [15]. Due to the nature of this algorithm, it can be difficult to teach and explain how it works. Since most of the algorithm deals with 3D spaces, helping someone visualise it in 2D is challenging. Hence, students and scientists at the University of Groningen developed an application with the purpose of making the learning process easier. This application is called Virtual Ray Tracer [20, 19]. From this point onwards we will refer to Virtual Ray Tracer as VRT.

The aim of this paper is to build a new level in VRT that explains how anamorphic sculptures can be built using ray tracing. The paper written by Louis Pratt, Andrew Johnston, and Nico Pietroni [17], which was mentioned above, focuses on catoptric anamorphosis. Therefore, the level will focus on the use of mirrors and the objects reflection in them to build anamorphic sculptures.

In this paper, we explore the method of using ray tracing as a means to deform objects based on their reflections in different types of mirrors. Then, we plan on building our own framework to implement it. At last, we will create a new game level in a version of VRT that uses real-time ray tracing to present the method.

This project aims to help Computer Graphics students, art and technology enthusiasts learn about anamorphic sculptures. With the help of the tool, other people should be able to build their own frameworks that deform objects using ray tracing.

In Section 2 we explore background and related work information about anamorphism and VRT. Section 3 describes the main requirement of this project. In Section 4 we explain our approach to construct anamorphic art and build a level to explain it. Section 5 discusses the results of our implementation. A conclusion of the project is drawn in Section 6. Possible future work is touched upon in Section 7.

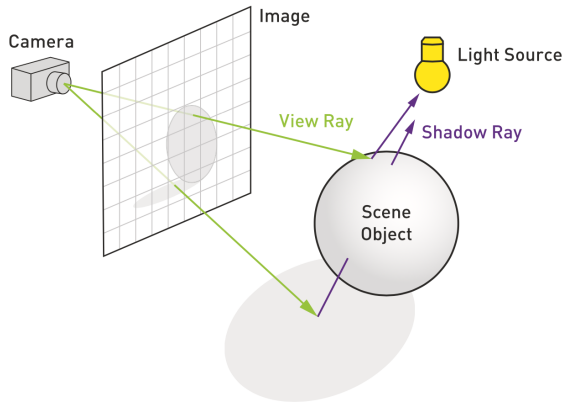
2 Background and Related Work

Before diving into the requirements and implementation details of this project, let us have a look at some background information and related work. We will first have a look at the definition of anamorphosis. Next we will talk about ray tracing and tools used to visualise this algorithm. Afterwards, we will talk about anamorphosis and its applications in art.

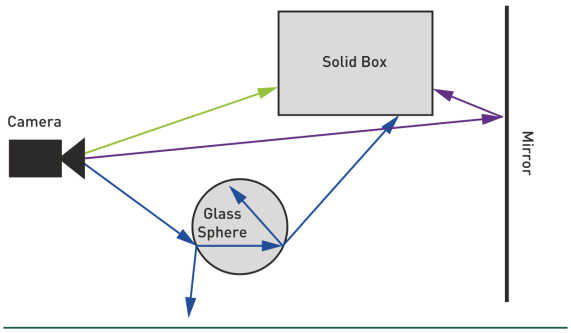
2.1 Ray Tracing

Ray tracing is one of the most popular algorithms in the field of Computer Graphics. To define what it does, we first have to take a look at the definition of ray casting.

Ray casting is the process of shooting a ray from a camera, through a pixel on a screen, and into the scene beyond. The screen is positioned between the camera and the scene. The procedure is meant to determine if the ray going through a pixel intersects any object in the scene. Sometimes, a second ray may be send from the point of intersection of a ray with an object towards a light source. This is done with the purpose of determining how that point is illuminated [13]. An illustration of this can be found in figure 1a.



(a) Ray casting [13]



(b) Ray tracing [13]

Figure 1: Ray casting and ray tracing explained. Images adopted from [13]

Ray tracing takes a 3D model of a scene and renders a 2D digital image based on it. It uses the ray casting process recursively by bouncing the ray multiple times around the scene starting from the initial point of intersection in the scene. It gathers information about light, reflected and refracted rays, and keeps track of them in a recursion tree. To determine the final colour of the original ray's corresponding pixel, the tree is evaluated by climbing it from the bottom along the pixel's designated chain [13]. An image describing the bounce of the ray from object to object in the scene can be found in Figure 1b.

2.2 Ray Tracing Visualisation Tools

Ray tracing is a complex algorithm that makes use of three dimensional descriptions of scenes. Because of this, it can be difficult to explain how it works without visual aids.

Unfortunately, 2D images are not always enough to help a student fully grasp the concept of ray tracing. This is why, researchers at the University of Groningen have come up with *Virtual Ray Tracer* (VRT) [20]. VRT is an educational tool developed by C.S. van Wezel and W.A. Verschoore de la Houssaije, two former Computing Science Bachelor students at the University of Groningen, in collaboration with the Scientific Visualisation and Computer Graphics (SVCG) research group.

Since its initial version published in 2022, a second version of the application has been developed. VRT2.0 [19] is a gamified version of the original tool. This has been done in order to make the application more engaging.

In a typical level in VRT, a user can move around the scene, see the scene from the perspective of a camera that can be moved about, and change the colour, position, rotation, and scale of an object. Another useful feature is that in front of the camera used in ray tracing the scene, there is a pixel grid whose size can be modified. The rays going from the camera pass through every pixel of the grid and change colour depending on type of interaction in the scene. An individual ray can be seen by selecting a pixel in the preview screen. It also shows a preview panel of what the image constructed by the ray tracer looks like. A screenshot of the typical VRT scene can be seen in Figure 2.

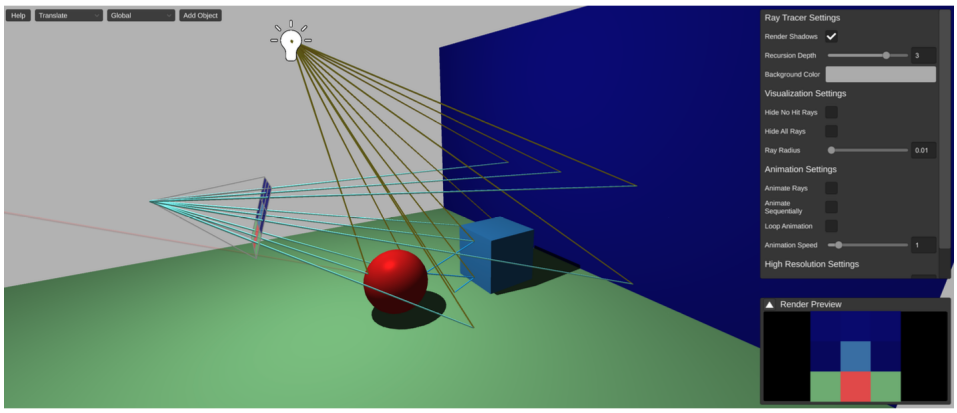


Figure 2: A scene in VRT [20].

Another tool worth mentioning is the *Ray Tracing Visualization Toolkit* (rtVTK) [12]. It works similarly to VRT. The two downsides of rtVTK are the fact that it is not beginner friendly nor is it user friendly. VRT guides the student through every step of the ray tracing algorithm, while rtVTK assumes some previous knowledge. Furthermore, VRT has a web version making it easily accessible to anyone, while rtVTK needs to be installed and a ray-based renderer needs to exist on the machine of the user.

2.3 Anamorphic Art

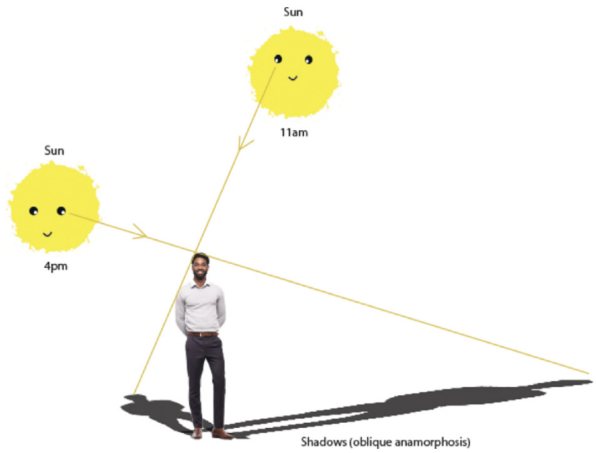
Anamorphosis means to produce a distortion of an object which upon glance will look nothing like its original form. It requires a viewer to either have a specific point of view, to use special devices, such as mirrors, glasses of water, or other similar optic media, or a combination of both, in order to see a distinguishable image of the object that was deformed. The concept

of anamorphism has been around for a long time, however anamorphic art has only started to become popular in the sixteenth century [7, 11].

We distinguish between two main types of anamorphosis. Namely, *perspective (oblique)* and *mirror (catoptric)* anamorphosis.

- Perspective anamorphosis requires the viewer to occupy a specific point of view in order to see a recognisable image or object [18]. It deforms an object such that distances and angles are modified, however lines are not. An example that illustrates perspective anamorphosis is the shadow of a person on the ground at different times of the day, as it can be seen in Figure 3a. A more modern approach is anamorphic 3D chalk art. Artists draw on the street with chalk and certain perspectives give the effect of the art appearing 3D [8].

- In the case of mirror anamorphosis, the viewer needs a mirror in order to see a recognisable image [9]. The most common type of mirror used is cylindrical. This type of anamorphosis was primarily used to make 2D art, as seen in Figure 4a, until more recent years when artists started building 3D sculptures centred around a mirror, exemplified in Figure 4b.



(a) Shadows illustrating anamorphism [17]

(b) Anamorphic chalk art [8]

Figure 3: Examples of oblique (perspective) anamorphism.

The focus of this project is mirror anamorphosis. More specifically, how to construct sculptures using mirror anamorphosis. Some researchers have tried to simulate building such sculptures by using software. In a recent paper, a method that uses ray tracing to construct such sculptures is used [17]. Some of the sculptures they created have been 3D printed and displayed at different art festivals. One of their sculptures can be seen in Figure 5.

(a) 2D illustration using mirror anamorphosis [16]

(b) 3D anamorphic sculpture using a mirror [14]

Figure 4: Examples of catoptric (mirror) anamorphsim.

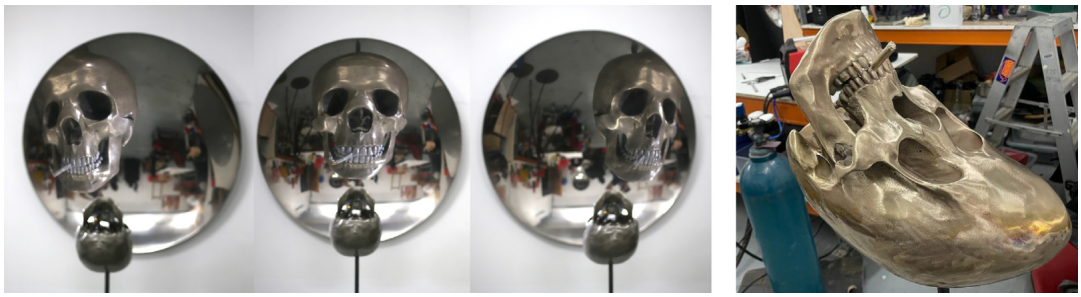
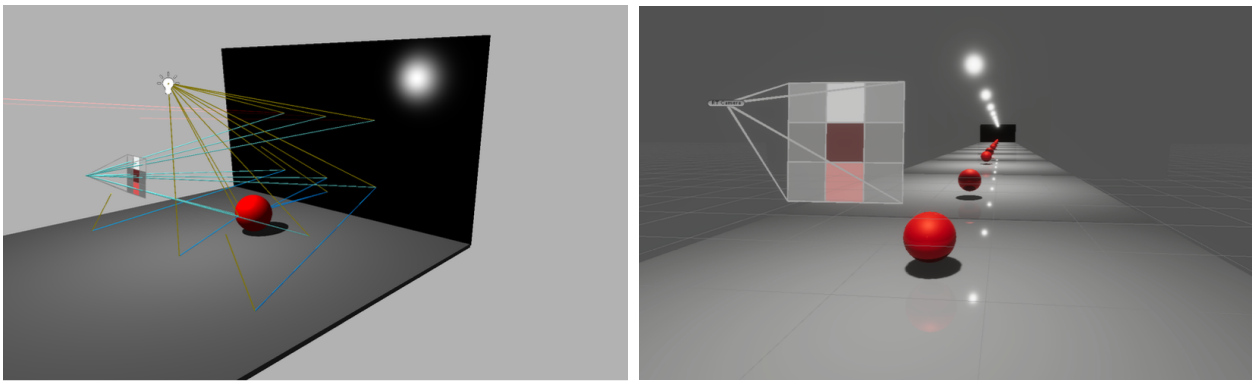


Figure 5: Anamorphic skull sculpture deformed using ray tracing [17].

3 Requirements

The goal of this project is to build a level (or multiple levels) in VRT that explain how anamorphic sculptures are created. The most important part is then to be able to see the reflection of the deformed object on the mirror in the scene.

VRT was developed using the game engine Unity [6]. The original VRT uses rasterisation to render its scenes. However, it is almost impossible to get good reflections and refractions when using this algorithm. An exaple of reflection can be found in Figure 6a. For this reason, in 2023 a student at the University of Groningen implemented real-time ray tracing in VRT [10]. This version uses Unity's *High Definition Render Pipeline* to render the final scenes of the game [3]. Therefore, reflections and refractions look realistic in this version of the tool, as can be seen in Figure 6b. It is this version of the tool that we will be building upon in this project since we need real-time ray tracing in order to ascertain the accuracy of the reflections of the sculptures.



(a) Reflections in VRT [10]

(b) Reflections in VRT using HDRP [10]

Figure 6: Difference in reflection between VRT versions.

As has been explained so far, when it comes to mirror anamorphosis, the reflected rays are very important. Which is why we have the following requirements:

1. Implement the method to deform objects explained by Pratt et al. [17].
2. Add a level to VRT where mirror anamorphism is taught.
    - Define and explain anamorphism.
    - Explain how an obejct is deformed in mirror anamprphosis.
    - The reflection of the deformed object should be visible on the mirror.
    - Have a reference to the original object to compare with the reflection of the deformed one.
    - Maintain the other functionalities of VRT.

4 Methodology

This project has two main components. Namely, the framework used to deform an object and VRT. The object deformed by the framework is used to construct the levels in VRT.

4.1 Tools and Languages

The first tool we address is the framework that we used to deform the objects. Given a scene description and the mesh of an object, we feed them into a C++ framework. The result is the mesh of the deformed object.

A scene description is a JSON file [4] that contains information about the context in which we will deform an object. It contains three main elements: the position of the camera, the type, shape and, position of mirror used in the deformation, and the position of the object we want to deform. Examples of the scene descriptions we used, as well as additional information about them, can be found in Appendix A. Additionally, depending on the type of mirror present in the scene, the mirror entity will have additional details such as position, radius, rotation, and in the case of plane mirrors, the four edges at each corner of the plane.

Another file necessary for our framework is an OBJ file [5]. An object file gives the description of the mesh of an object. It gives the coordinates of the vertices of a mesh, the faces of the mesh, texture coordinates, and other details. In our case, we are only interested in the coordinates of the vertices and the faces of the objects they form. The input file is generally a model we constructed using Blender [1] and exported them as Wavefront(.obj) files. After deforming the object, the new vertex coordinates of the mesh and faces, which have been preserved, are written to a new object file. This object file will later be used to build a scene in VRT.

The C++ framework mentioned above has initially been a ray tracing framework used in the course Computer Graphics taught at the University of Groningen. Most of the framework has been erased and what remained has been repurposed to fit the requirements of this project.

As mentioned above, Blender [1] has been used to make models of objects to deform or to edit models downloaded online. It has also been build to model example scenes of what the final VRT scene would look like and to visualise the deformed object.

The last tool used for this project is the game engine Unity [6] since this is the engine originally used to build VRT.

4.2 The Framework

The framework [1] is relatively simple. Once the scene description and the object file are passed to it, it parses them and saves the relevant information. Then it loops through all of the points on the mesh, calculates their deformation, and writes all of the new displaced points to a new OBJ file. Therefore, we are interested in how two things are done, and those are how to calculate the deformation of the object and how is the intersection of the rays with the mirror calculated. Additionally, before deforming an object, if it is a large mesh,

[1] `https://github.com/irinaB11/anamorphicRayTracer`

a function that fits it in the unit cube can be called. This is necessary in order to avoid situation where the new points cannot be calculated.

We are given a camera C, a mirror, and an object mesh with vertices V, where V is any vertex on the mesh of the object.

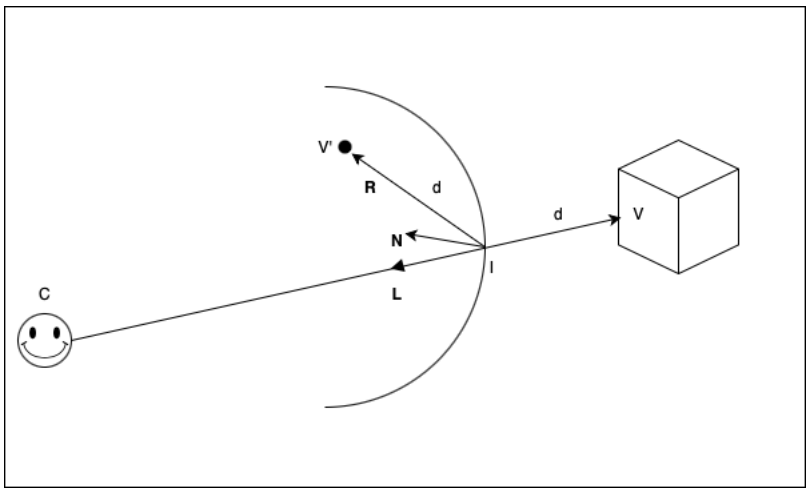


Figure 7: Process of calculating deformed point

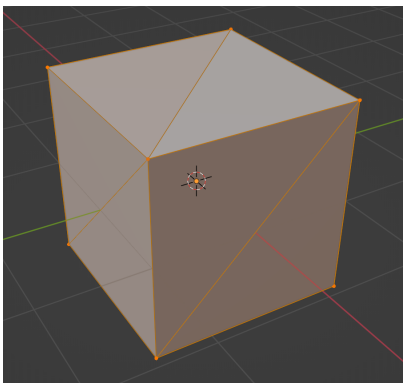
A ray shoots from the camera to a point on the object. This ray passes through a mirror. Therefore, it can have (plane, cylinder, sphere) one or two intersections points with it, depending on the type of mirror used. We are interested in the intersection point that is closest to the object. Now that we have the intersection point of the ray with the mirror, we calculate two things. That is, the normal at the intersection point and the distance d between the intersection point and the vertex of the object. Based on the incident ray and the normal we can calculate the reflection ray. Taking the distance, we move the point of the object along the reflected ray at distance d from the intersection point. An illustration of this process can be found in Figure 7. We repeat this algorithm for every vertex of the object.

Finding the point of intersection of the ray with the mirror can result in three different type of calculations based on the shape of the mirror. The type of mirrors the algorithm addresses are flat mirrors (also known as quads or plane mirrors), cylinder mirrors, and sphere mirrors.

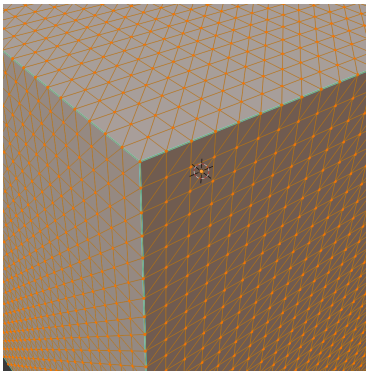
4.3 The Objects

The meshes of the objects that we choose to deform plays an import role in the shape of the resulting object. Multiple objects can have the same shape but they have meshes that look completely different from each other. Such an example can be seen in Figure 8. The mesh of the cube shown in Figure 8a consists of 8 vertices and 16 triangular faces. On the other hand, the cube in Figure 8c looks the same but has a much more complex mesh. It has 6146 vertices and 12288 triangular faces. A close up of this mesh can be seen in Figure 8b.

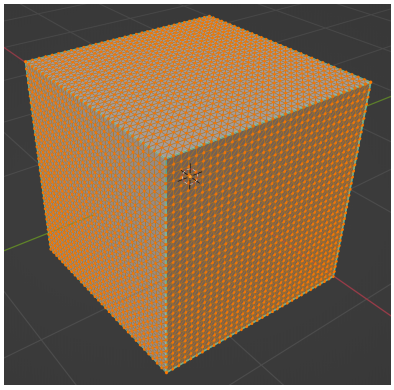
The deformations of the meshes described above based on their reflection on a cylindrical mirror can be seen in Figure 9. The edges of the deformed cube with the simpler mesh are



(a) A cube with a low number of vertices.



(b) A close up of a cube with a complex mesh.



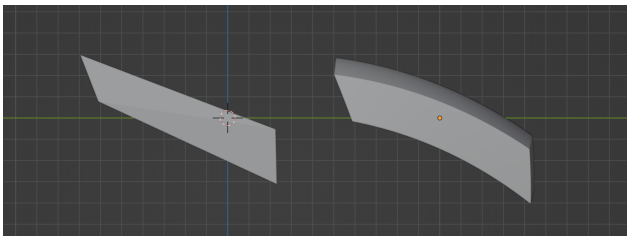
(c) A cube with a high number of vertices.

Figure 8: Difference between the same object with different mesh complexities.

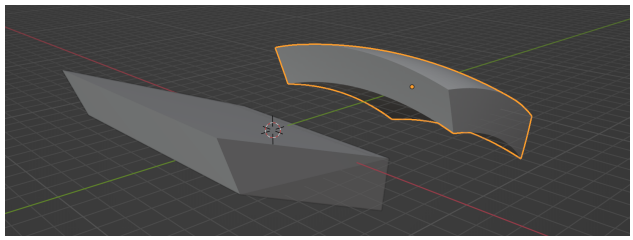
straight, while in the more complex mesh, there is a distinctive curve. This is the result of the number of vertices along the edge of a cube. If an edge is composed of only the vertices at its ends, in the resulting deformation, the edge is be a straight line. However, if a cube has multiple vertices along an edge, after deforming it, connecting the vertices results in a curved edge.

From the above example we can conclude that using objects that have a high number of vertices in their mesh will result in more accurate deformations. However, before constructing the mesh of an object or downloading one online, we have to take into consideration the limit of vertices a mesh can have in Unity. It is recommended that this limit is not surpassed.

In the implementation of our VRT scenes, we use an object shaped like a box. This box has a mesh consisting of 6146 vertices and 12288 triangular faces. This file of this object is saved in a folder in the C++ framework [2] that also performs the deformation of this object. If the object is so large that the ray going from the camera to the vertex does not intersect the mirror in between them, a function *unitize()* can be called. This function finds the bounding box the object and uniformly shrinks it so that it fits in a unit cube.



(a) Side view.



(b) Perspective view.

Figure 9: Side comparison of deformed objects that have the same shape but different meshes. The right object has a more complex mesh than the left one. Their deformation was calculated using a cylindrical mirror.

[2] `https://github.com/irinaB11/anamorphicRayTracer`

4.4 VRT Level

When we deformed the object, we had to feed the program a scene description. This scene description is based on a model we build using Blender. The model not only gives us the coordinates of the positions of the objects present in the scene but it also serves as a mock-up of what the VRT level will look like. Since we have three different types of mirror we use to deform objects, we build three models, and subsequently three levels in VRT. Images of these models can be seen in Figures 11.

Since the Ray Tracing camera (RT Camera) is an important component in this project, it has been kept in the scene as the point of view used to deform the original object. None of its functionality has been disabled. Therefore, the rays that shoot from it into the scene are visible, its position can be changed, and the number of pixels present on the grid in front of it can be tinkered with. Also, the reflection that resembles the original object the most can be seen from its perspective.

The original object has been kept in the scene in all of the levels, positioned behind the mirror, in order to have a reference to what the reflection of the deformed object should look like.

The mirrors in the levels have all been added to the scene by using prefabs already present in the project file of VRT. To these prefabs, a mirror texture that can be rendered by Unity's *HDRP* has been added. These textures were already imported into the project. While calculating the deformation of the object, we do not take into consideration the mesh of the mirror. We are only interested in the four points at the corners of the quad, in the case of plane mirrors; the radius of the sphere when we have spherical mirrors; and the radius, upper most and lower most points on the central axis of the cylinder when we have a cylindrical mirror.

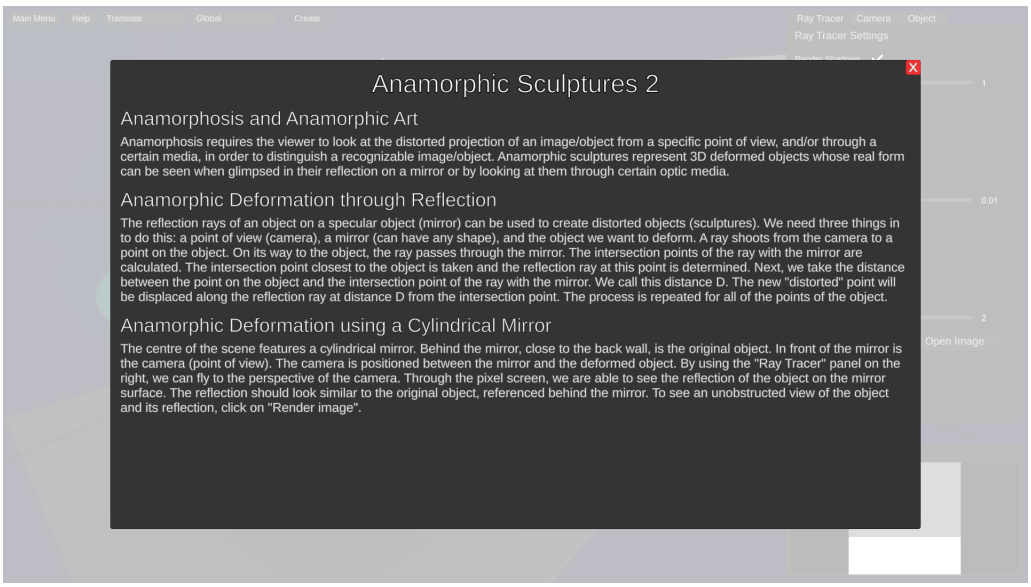
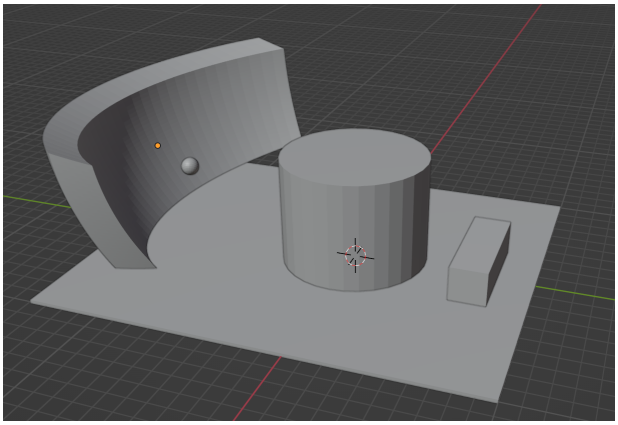
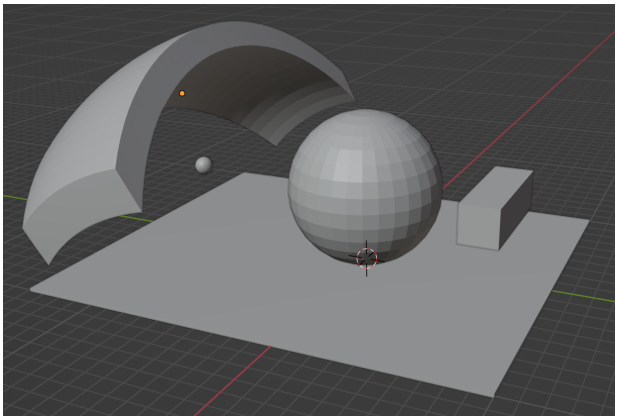


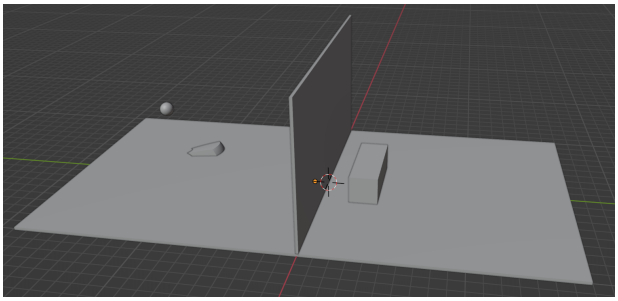
Figure 10: Panel that pops up when opening a level. It explains what happens in the scene.



(a) Scene with a cylinder mirror.



(b) Scene with a sphere mirror.



(c) Model of a scene with a plane mirror.

Figure 11: Models of scenes. In all three cases, the original object is the box on the right.

The initial idea of the project was to position the RT Camera such that the deformed object and its reflection on the mirror are visible at the same time. In the case of the flat mirror level, this was possible. However, when we used a cylindrical and spherical mirror, the reflection of the deformed object ended up being obstructed by itself. This is why in those levels, we decided to keep the RT Camera between the deformed object and the mirror. This way the deformed object is visible from the normal perspective of the game camera, and its reflection on the mirror is visible from the perspective of the RT Camera.

The first time a level is opened, the user will be met with a panel. This panel has information about what anamorphosis is, how objects are deformed in mirror anamorphosis using their reflection, and a general description of the scene and how to view the reflection of the deformed object. A sample of this can be seen in Figure 10.

Additional models depicting the deformation of objects that are not the rectangular parallelepiped used in these levels, can be found in the appendix of this thesis.

5 Results

In this section, we show the final VRT [3] levels we implemented. Additionally, we will analyse the resulting reflections of the deformed objects and discuss the limitations of the application. The laptop we used to build these levels and to test the reflections of the objects works on a Windows 11 operating system and has a NVIDIA GeForce RTX 3050 GPU.

In Figure 12, the top left image shows what the first level of anamorphic sculptures looks like after closing the introductory panel. The top right picture was taken after navigating through the scene to see the original object behind the mirror. In the bottom right picture, the perspective of the viewer was changed to match that of the RT Camera. The final image, on the bottom right, shows the reflection of the deformed object unobstructed by the pixel grid. As can be seen in the picture, the reflection of the deformed object does not exactly match the original object. After experimenting with different positions for both the camera and the object, we managed to make the reflection of the object match the frontal view of the original object almost perfectly. While we are not exactly sure why the reflection of the object does not match its original shape, we have some theories. It is possible that the right position and angle has not been found yet. Otherwise, assuming that the mathematical part of our calculations is correct and we did not exclude any possible edge cases, a possible explanation is a floating point inaccuracy. In our framework, we use type float to store and calculate the vertices of the mesh. It is possible that were we to change the type to double, the reflection would be more accurate. If, on the other hand, we made a mistake from the mathematical point of view of this project, righting the mistake should result in a more accurate reflection.

In the second level we developed in VRT, we had better luck regarding the accuracy of the reflection. Once again, as it can be seen in Figure 13, the top two images show the scene at first glance and the scene after navigating it to catch a better look at the original object. It is worth mentioning that the user can change the position of this object if they wish to have a better look at it without the obstruction of the mirror in front of it. The bottom left image shows the scene from a perspective that comprises the whole scene, slightly above every component present. From this point of view, the reflection is visible and we can see that it somewhat resembles the original object. The last image shows the reflection of the anamorphic sculpture from the perspective of the RT Camera. This is the closest reflection we have to the original object. The bottom edge of the front of the object still has a slight curve which we could not diminish further. The curvature of the bottom edge could be the result of the position of the object. While implementing the scene, we experimented with different distances from the mirror, and rotations of the object, in the pursuit of the most accurate reflection possible. It is possible that we just have not found the most optimal position yet. Another possibility, is a calculation error when we implemented the framework that deformed these objects.

[3] `https://github.com/irinaB11/Virtual-Ray-Tracer-Anamorphic-Sculptures/tree/VRT-RTX-Anamorphic-Sculptures`

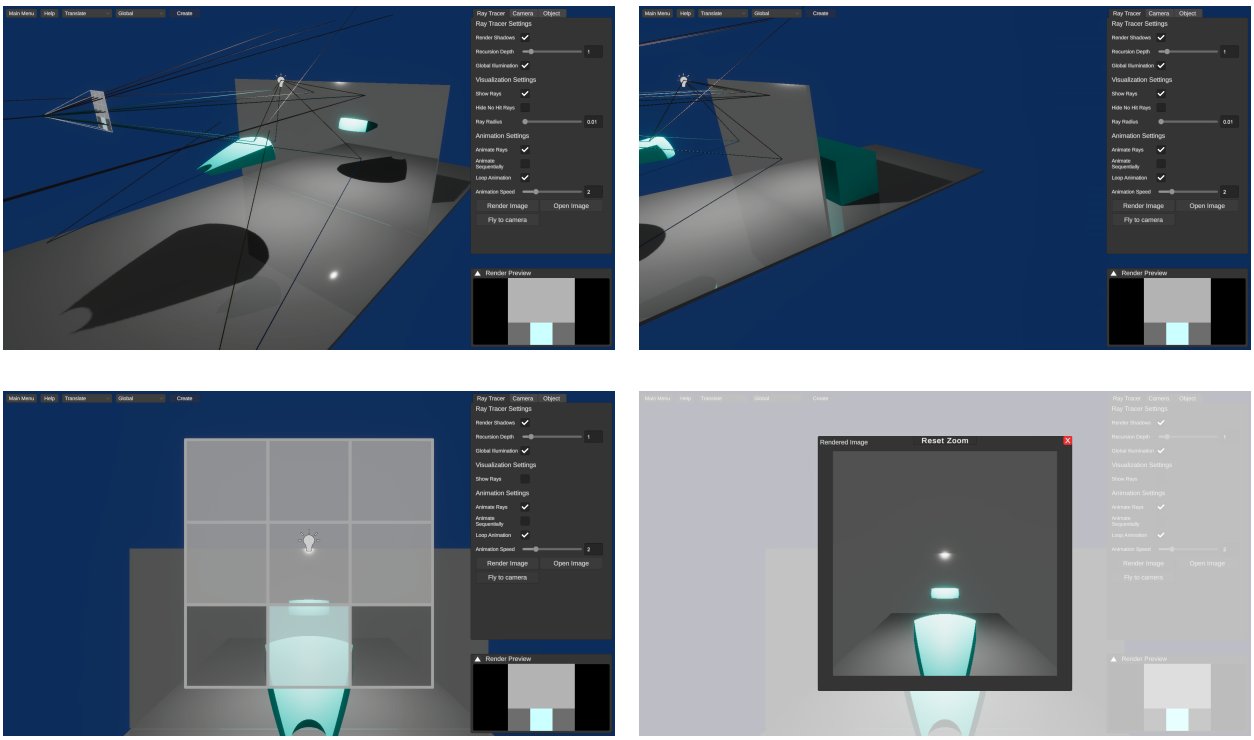


Figure 12: VRT level: Anamorphic Sculptures 1.

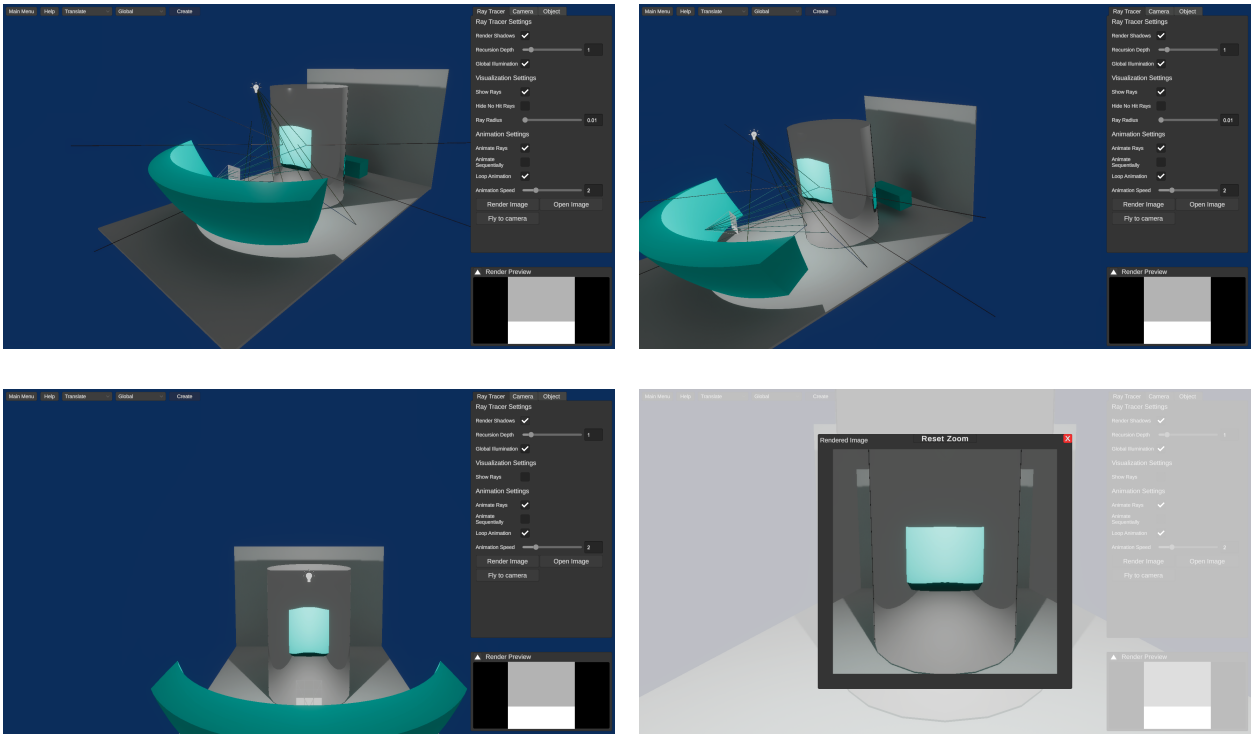


Figure 13: VRT level: Anamorphic Sculptures 2.

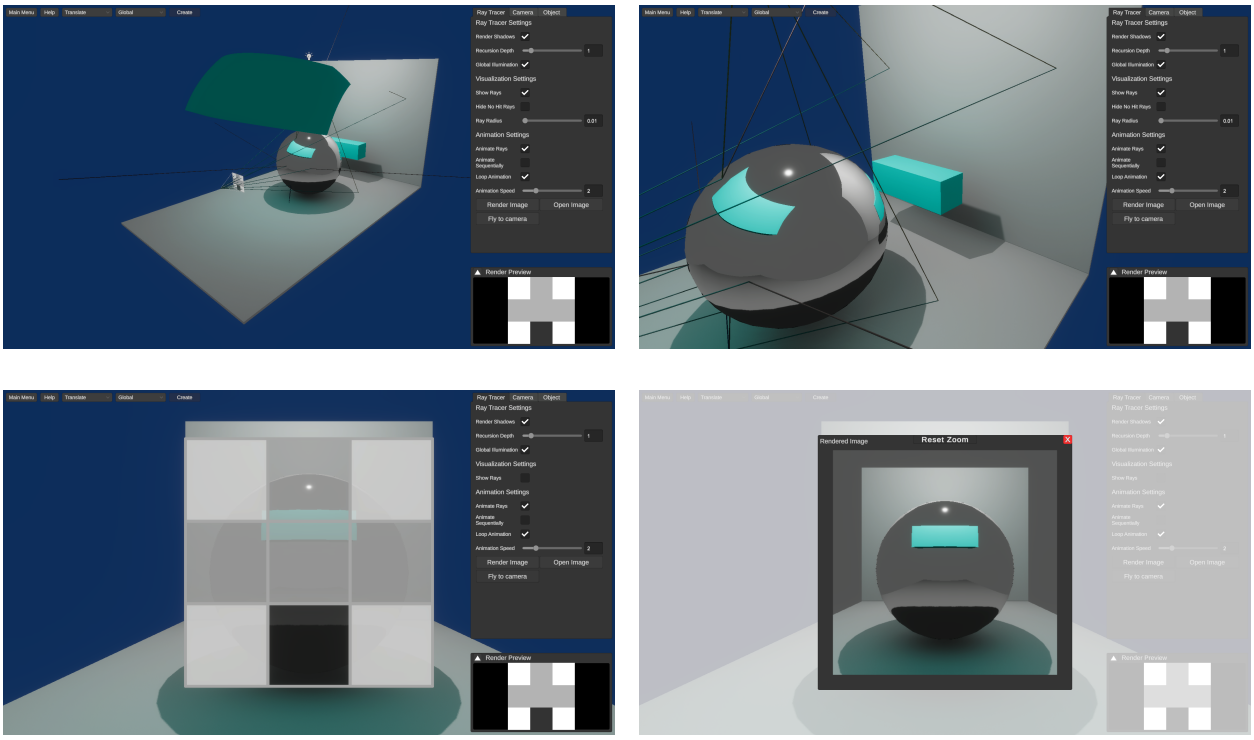


Figure 14: VRT level: Anamorphic Sculptures 3.

In the last level we implemented, we are looking at the case where we have a spherical mirror. Once again, the top two pictures show the initial scene and the scene where the perspective is oriented closer to the original object. Here we stumbled into similar problems to the cylinder level when tinkering with the perspective of the RT Camera. However, the object is positioned much higher than the rest of the objects in the scene in order to get the right reflection as viewed by the RT Camera. The bottom pictures in Figure 14 show the perspective of the RT Camera, once through the pixel grid and another time after rendering the picture. In this level, we have a reflection that perfectly matches the original object. None of the edges are curved, like it is the case in the other levels. Here, it is possible to see that the reflection of the deformed object matches the original one very well. There is also no visible curvature to its edges.

6 Conclusion

We started this project with the goal of understanding how building anamorphic sculptures using ray tracing works and building a level in VRT trying to explain the process.

In the end, we managed to build a framework that deforms objects using their reflections on three different types of mirrors. Using the meshes of the resulting objects we proceeded to build three different levels each explaining mirror anamorphosis performed with the use of each type of mirror described.

Whilst the results that we have are not the most ideal, we hope that this project sets the base for future development of this topic or similar ones in VRT.

7 Future Work

The framework that we built has several limitations. One of which is the type of mirror that can be used to deform an object. Since momentarily it is only possible to deform objects based on three types of mirror, a viable future feature would be to extent the framework to free-form mirrors.

Another possible addition would be to deform the objects based on the refraction through a transparent medium. Examples of such media may include transparent spheres, glasses of water, and others.

A third possible feature would to implement the deformation function directly in VRT. This way, the framework, which acts like a middle man in our case, would not be necessary anymore.

A fourth option, would be to gamify the level such that the explanation gets broken into smaller tutorial-like steps. This would help the user understand the process better and make the learning experience more enjoyable.

To address some of the issues we encountered when we constructed the VRT levels, we will suggest what could be changed/reviewed/experimented with.

- For both levels, *Anamorphic Sculptures 1* and *Anamorphic Sculptures 2*, tinkering with the position of the deformed object and its rotation along the x-axis might solve the problem. If this does not solve the issue, the following points might be helpful.

- In level *Anamorphic Sculptures 1* (plane mirror), the reflection does not match the original object closely. We theorised that it is either a mathematical problem or a float type issue. Therefore, we first suggest reviewing the mathematics used to calculate the ray-plane intersection. The functions mentioned are in the C++ framework in *shapes/quad.cpp*. If the mathematical equations turn out not to need any tweaks, a second suggestion we make is to change all of the float type arrays, variables, and functions to type double.

- In the second level we implemented, *Anamorphic Sculptures 2* (cylinder mirror), the reflection of the deformed object matches the shape of the original object almost perfectly. The only difference is the lower edge of the reflection is curved upward. It could be one of the two issues mentioned above for the plane mirror levels. For this reason,

we suggest to double-check the mathematical part of the code that calculates the ray-cylinder intersection. It can be found in the C++ framework in *shapes/cylinder.cpp*.

8 Acknowledgements

I want to thank my supervisors Prof. Jiří Kosinka and Dr. Steffen Frey for their help and support along this project. Their guidance has been very appreciated. I would also like to thank Tom Couperus for his help and advice, especially for his help with VRT, Unity, and other small questions I had. I also want to thank my friend Eertze van de Riet for his help in debugging the framework when I was really stuck. And thank you to all my family and friends who let me use them as rubber duckies whenever I was thinking about this project aloud.

References

[1] Blender documentation. [Accessed: 11-07-2024]. URL: https://docs.blender.org/manual/en/latest/.

[2] Free3d. [Accessed: 22-07-2024]. URL: https://free3d.com/3d-model/mummy-hand-v1--243397.html.

[3] High definition render pipeline overview. [Accessed: 11-07-2014]. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@8.0/manual/index.html.

[4] Json file. [Accessed: 11-07-2024]. URL: https://fileinfo.com/extension/json.

[5] Obj file. [Accessed: 11-07-2024]. URL: https://docs.fileformat.com/3d/obj/.

[6] Unity documentation. [Accessed: 11-07-2014]. URL: https://docs.unity3d.com/2020.3/Documentation/Manual/index.html.

[7] Jurgis Baltrušaitis. *Anamorphic art*. Chadwyck-Healey, 1977.

[8] Cecilia Mazzoli Caterina Morganti Cristiana Bartolomei, Alfonso Ippolito. From anamorphosis to vision: "3d sidewalk chalk art". *Disegnare con*, 2020. https://doi.org/10.20365/disegnarecon.24.2020.1.

[9] Francesco de Comite. A General Procedure for the Construction of Mirror Anamorphoses. In *Bridges 2010: Mathematics, Music, Art, Architecture, Culture*, pages 231–238, Pecs, Hungary, July 2010. URL: https://hal.science/hal-00861388.

[10] Tomas de Vries. Implementing real-time ray tracing in unity to increase the render quality of a ray tracing visualization tool. 2023. URL: https://fse.studenttheses.ub.rug.nl/30711/.

[11] Crannell A. Frantz, M. *Viewpoints: Mathematical perspective and fractal geometry in art*. Princeton University Press., 2011.

[12] Christiaan Gribble, Jeremy Fisher, Daniel Eby, Ed Quigley, and Gideon Ludwig. Ray tracing visualization toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '12, page 71–78, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2159616.2159628.

[13] Eric Haines and Tomas Akenine-Möller, editors. *Ray Tracing Gems*. Apress, 2019. http://raytracinggems.com.

[14] Yifat Davidoff Jonty Hurwitz. Rosso horse, 2022. URL: https://jontyhurwitz.com/horse-rosso.

[15] S Marschner and P Shirley. *Fundamentals of computer graphics (Fourth)*. Taylor & Francis Group, 2016.

[16] Kaushik Patowary. Anamorphic art by István Orosz. *Amusing Planet*, 21 April 2010. [Accessed: 11-07-2024). URL: https://www.amusingplanet.com/2010/04/anamorphic-art-by-istvan-orosz.html.

[17] Louis Pratt, Andrew Johnston, and Nico Pietroni. Bending the light: Next generation anamorphic sculptures. *Computers Graphics*, 114:210–218, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0097849323000778, doi:10.1016/j.cag.2023.05.023.

[18] Javier Sánchez-Reyes and Jesús M. Chacón. Anamorphic free-form deformation. *Computer Aided Geometric Design*, 46:30–42, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0167839616300826, doi:10.1016/j.cagd.2016.06.002.

[19] Chris S. van Wezel, Willard A. Verschoore de la Houssaije, Steffen Frey, and Jiří Kosinka. Virtual ray tracer 2.0. *Computers Graphics*, 111:89–102, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0097849323000067, doi:10.1016/j.cag.2023.01.005.

[20] Willard A. Verschoore de la Houssaije, Chris S. van Wezel, Steffen Frey, and Jiri Kosinka. Virtual Ray Tracer. In Jean-Jacques Bourdin and Eric Paquette, editors, *Eurographics 2022 - Education Papers*. The Eurographics Association, 2022. doi:10.2312/eged.20221045.

A Code, Images, Files

A.1 JSON file scene descriptions

The word "Eye" denotes the camera and the numbers following between square brackets give its position. "Objects" denotes the other two elements of the scene, the mirror and the original object. The mirror is denoted by "type":"x" where x can be a quad (flat mirror), cylinder, or sphere. Depending on the type of mirror, the following additional information will be given about it:

- "quad": the four corner points of the plane will be given as vertices "vi", where i = { 0, 1, 2, 3}.
- "cylinder": "position" refers to the point at the centre of the object, "radius" gives the value of the radius of the cylinder, "topPoint" and "bottomPoint" refer to the points in the centre of the circles that bound the cylinder at its top and bottom.
- "sphere": "position" refers to the point at the centre of the sphere, "radius" gives the value of the radius of the sphere.

flat.json

```
{
    "Eye": [2, -12, 8],
    "Objects": [
        {
            "type": "quad",
            "v0": [15, 0, 10],
            "v1": [-15, 0, 10],
            "v2": [-15, 0, 0],
            "v3": [15, 0, 0]
        },
        {
            "type": "mesh",
            "position": [2, 10, 1]
        }
    ]
}
```

cylinder.json

```
{
    "Eye": [0, -30, 15],
    "Objects": [
        {
            "type": "cylinder",
            "position": [0, 0, 10],
            "radius": 10,
```

```
            "topPoint": [0, 40, 10],
            "bottomPoint": [0, 0, 10]
        },
        {
            "type": "mesh",
            "position": [0, 30, 1]
        }
    ]
}
```

sphere.json

```
{
    "Eye": [0, -12, 5],
    "Objects": [
        {
            "type": "sphere",
            "position": [0, 0, 5],
            "radius": 5
        },
        {
            "type": "mesh",
            "position": [0, 5, 1]
        }
    ]
}
```

A.2 OBJ file

An example of what an input object file exported from Blender looks like. Ideally, we would an object that is formed out of many vertices. This input file will not have many vertices since it is meant to be an example template.

cube.obj

```
# Blender 3.3.1
# www.blender.org
mtllib cube2.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
```

```
vn -0.0000 1.0000 -0.0000
vn -0.0000 -0.0000 1.0000
vn -1.0000 -0.0000 -0.0000
vn -0.0000 -1.0000 -0.0000
vn 1.0000 -0.0000 -0.0000
vn -0.0000 -0.0000 -1.0000
vt 0.625000 0.500000
vt 0.375000 0.500000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.875000 0.500000
vt 0.625000 0.250000
vt 0.125000 0.500000
vt 0.375000 0.250000
vt 0.875000 0.750000
vt 0.625000 1.000000
vt 0.625000 0.000000
vt 0.375000 0.000000
vt 0.375000 1.000000
vt 0.125000 0.750000
s 0
usemtl Material
f 5/5/1 3/3/1 1/1/1
f 3/3/2 8/13/2 4/4/2
f 7/11/3 6/8/3 8/12/3
f 2/2/4 8/14/4 6/7/4
f 1/1/5 4/4/5 2/2/5
f 5/6/6 2/2/6 6/8/6
f 5/5/1 7/9/1 3/3/1
f 3/3/2 7/10/2 8/13/2
f 7/11/3 5/6/3 6/8/3
f 2/2/4 4/4/4 8/14/4
f 1/1/5 3/3/5 4/4/5
f 5/6/6 1/1/6 2/2/6
```

A.3 Models of different deformed objects

Figure 15 shows a model of a mummy hand deformed based on its reflection in a sphere and a cylinder mirror. If we were to create a scene in VRT for them, the deformed object's position would differ a little bit from this scene. The mesh of the object has been found online on a website that makes it possible to download the mesh for free [2]. Before we used it, we had to triangulate the mesh in Blender and fit the object into the unit cube using the C++ framework built. The mesh has 48994 vertices and 97984 triangular faces. A mesh of a simple hand, a goat, a monkey head, and other more basic shapes can be found on the GitHub repository of the C++ framework [4].

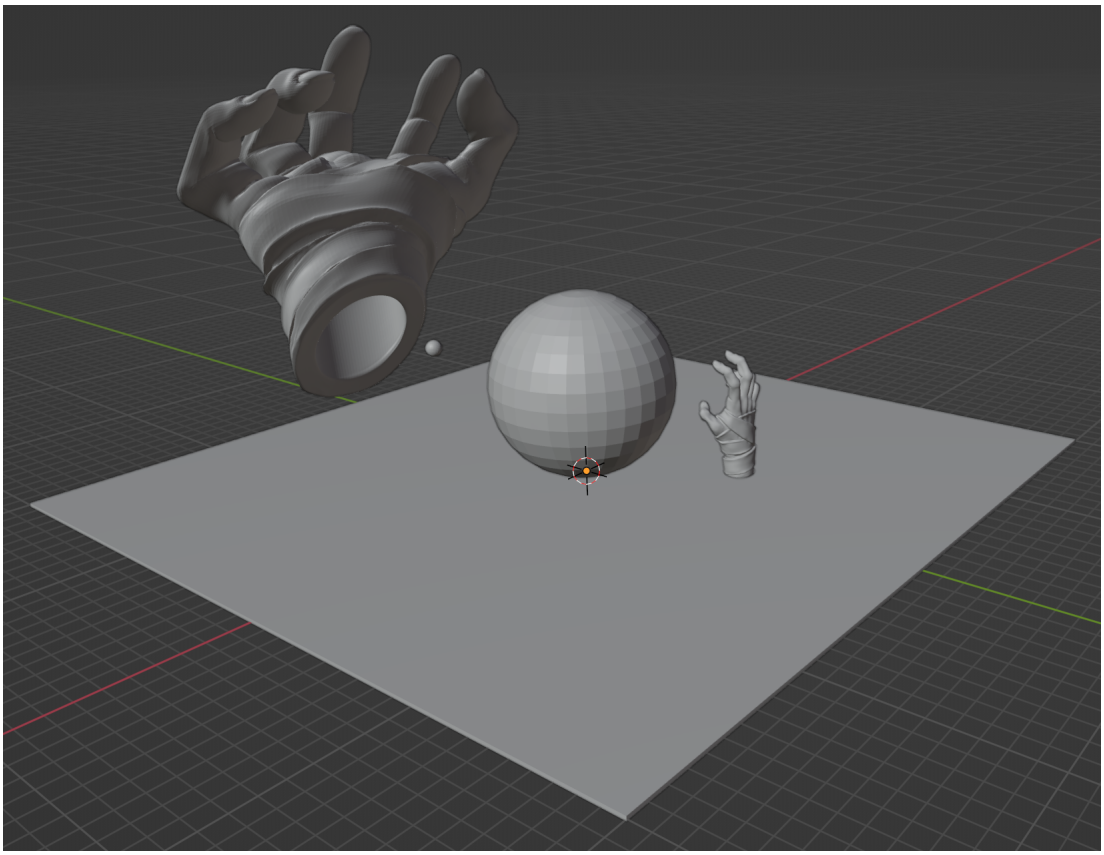


Figure 15: Deformed mummy hand.

[4] `https://github.com/irinaB11/anamorphicRayTracer`