university of groningen / faculty of science and engineering

Implementing real-time ray tracing in Unity to increase the render quality of a ray tracing visualization tool

Bachelor Thesis

July 7, 2023

Author:
Tomas de Vries

Primary supervisor:
Jiří Kosinka

Secondary supervisor:
Steffen Frey

Abstract

Ray tracing is a rendering method that is rising in popularity as it offers more realistic views compared to the other popular rendering method, rasterization.
Unity is a popular game engine that offers real-time ray tracing functionality. We have looked into the possibilities of combining this functionality with Virtual Ray Tracer (VRT), which is a tool to visualize ray tracing. By implementing real-time ray tracing features in VRT we intend to improve the quality of the tool. A small user survey was conducted to see what the users think of VRT with real-time ray tracing.

Contents

1 Introduction

In computer graphics, two major techniques are currently being used: rasterization and ray tracing. Ray tracing is more realistic in its rendering than rasterization. Ray tracing can visualize reflections, refractions and give better shadows. Figure 1 shows the difference between a scene without ray tracing and a scene with ray tracing.

Figure 1: The difference between a scene without ray tracing and with ray tracing using Nvidia RTX [1].

Ray tracing uses the concept of light in real life. We perceive color because a ray of light bounces off an object and it bounces into our eyes. It would be too computationally expensive if an application replicated this way of color perception because there are too many rays that would not hit our eyes. A common way of tracing the rays is tracing them from our eyes and letting them hit objects. If these objects are not reflective/refractive, a ray can be traced toward the light. If that ray then hits an object, it means that the hit point should be in a shadow. So the main concept of ray tracing is tracing rays of light to create a realistic rendering.

Ray tracing was not used in real-time until recently [2]. This is because it takes a lot of time to trace the rays. Rasterization is still a more popular way of visualization in real-time, however we believe that this will shift to ray tracing in the near future. That is why we have implemented ray tracing in a ray tracing visualization tool. The tool that we have implemented this in is Virtual Ray Tracer (VRT) [3]. VRT is a tool that is mostly used by RUG Computer Science students for the course Computer Graphics. It visualizes ray tracing by letting the user navigate in different scenes with objects and light sources in them. VRT allows the user to position a camera and from that viewpoint render a ray traced image. This is shown in more detail in section 2.
In this research we answered the following question: can we render the scenes of VRT using real-time ray tracing in Unity?

When implementing real-time ray tracing, performance could be a downside. As mentioned in the paragraph above, ray tracing takes more processing power than rasterization. If implementing real-time ray tracing leads to bad performance, we want to look into ways to improve this.
This can lead to the following question: How can we optimize the performance of our ray traced tool?

When optimizing the tool we might sacrifice graphics. We do not want to sacrifice too much, so we would like to find a good balance between performance and graphics. However, we will also keep in mind that performance can be different on various systems. Unity only allows advanced ray tracing features to run on systems that have a ray tracing compatible GPU [4]. So we were

not able to look at performance on machines that have a lower-end GPU. But we might still consider altering graphics settings based on performance on higher-end GPUs.

We believe that the ray traced scenes will benefit students that follow the course Computer Graphics. By ray tracing the scenes, transparent objects and reflections appear more realistic. Also, the shadows look better and we can use real-time ray traced global illumination. Most importantly, there will not be any mismatches between the view and the rendered image. We speculate that this will result in a smoother learning experience.

We want to look into the learning experience so we asked the question: Does real-time ray tracing improve the learning experience? This thesis includes a short evaluation that some Computer Science students filled in after they have tested the prototype. This gives us an insight into the usefulness of the new tool.

2 Background

This research has built upon the *Virtual Ray Tracer* (VRT) tool [3]. This tool is used to visualize ray tracing. Figure 2 shows how the rays are visualized in VRT. The user can navigate in a scene with rays, the rays are visualized as cylinders [3]. The logic behind the ray visualization will be explained in Section 3. The actual scene that the user navigates in is rendered using rasterization. This limitation is present because ray tracing is more time-consuming than rasterization [5]. Real-time ray tracing is therefore often too slow and rasterization is used instead [3]. However, that is what we have built on in this research. As mentioned in the introduction, we will extend the *Virtual Ray Tracer* tool to render the scenes using ray tracing to improve the learning experience.

2.1 Ray Tracing Visualization

One of the first implementations of an educational ray tracing visualization application was in 1999. This was Java Applets [6]. This application works similarly to VRT. It lets a user navigate a scene, in that scene it shows how rays would navigate through that scene. It is also possible to render a ray traced image of the scene from a certain perspective. Due to the application being so old, it is a bit simple. Unfortunately, it is also no longer available online. This means that it is not possible to extend the application to meet our requirements.

There is also *Ray Tracing Visualization Toolkit* (rtVTK) [7]. This tool visualizes ray tracing similar to how VRT does this. A notable limitation of this is user-friendliness. rtVTK can only be downloaded whereas VRT can be used in a browser as well. More importantly, rtVTK focuses on writing plugins. This gives more possibilities for rendering algorithms, but it is less beginner friendly since it is designed as a plugin for the user's own ray-based rendering programs. On the contrary, VRT gradually guides the user through ray tracing logic which we believe is better for learning ray tracing. Figure 3 shows how a basic scene in rtVTK looks.

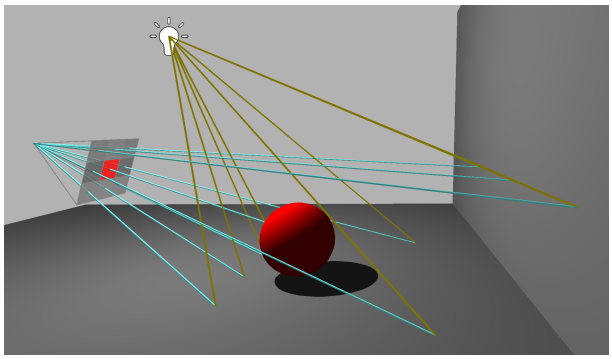

Figure 2: A basic scene in VRT

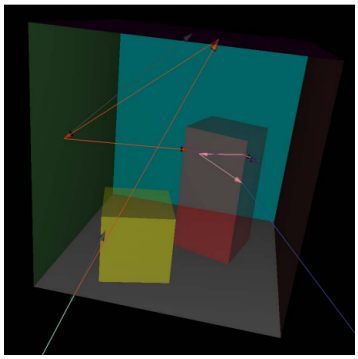

Figure 3: A basic scene in rtVTK

2.2 Unity Ray Tracing Projects

Implementing real-time ray tracing will all be done in *Unity* [4]. Unity is a game development tool that includes the possibility to integrate ray tracing [4]. Although it is fairly new there have been other Unity projects that implement real-time ray tracing. For example, the game *Syberia: The World Before* [8] has been made with Unity and implements real-time ray tracing throughout the entire game. In Figure 4 we see an example of the visuals that Unity can produce with ray tracing.

Figure 4: Screenshot of Syberia: The World Before which shows a ray traced scene

In March 2022 Unity released a new demo [9]. The demo is called 'Enemies'. Enemies shows what is possible with Unity HDRP. It was created by the official Unity team, so it truly shows the capabilities that HDRP has to offer. It shows realistic reflections as well as ray traced shadows. However, the most noticeable thing in the demo was how realistic the person was rendered. Ray tracing is still rapidly evolving and that is something that becomes clear from the demo. Figure 5 shows a screenshot from the Enemies demo.

Figure 5: Screenshot of Enemies

3 Ray Tracing concepts

As previously mentioned in the introduction, ray tracing is a concept where light rays are traced to decide which color is perceived. VRT shows how this is done. VRT divides different ray tracing concepts into different levels. The basic ray tracing level shows that rays are traced from the camera and when an object is hit a ray will be traced towards the light. Depending on whether an object is hit when tracing a ray towards the light a shadow will be rendered. This way we can not only render objects realistically but their shadows will be more realistic as well.

The next important concept is reflection. When a reflective object is hit, the ray should not yet trace toward the light. Instead, it should bounce back with an angle according to the law of reflection. This process can repeat recursively until an object is hit that is not reflective or no object is hit. There will also be a limit on how many times this process can repeat, since recursively tracing rays is computationally expensive and it is not possible to trace rays infinitely.

The next level introduces refraction. When looking through transparent objects, the light might be refracted, meaning it will not go straight. The light will instead move with an angle. This angle depends on the refractive indices of the air (or any other object it was moving through) and the object it hits.

Another important concept that is not explicitly mentioned in VRT is indirect lighting. When a red object is lit next to a white wall, the wall will have a red tint. This is because the light bounces off the red object and indirectly lights the white wall. In Computer Graphics this effect is known as Global Illumination. What happens is similar to reflections: When light hits an object it should bounce off and if it hits another object, indirect lighting should be considered.

4 Implementation

We aim to create real-time ray traced scenes that the user can navigate in. This project should be easily extensible. With the rapid evolution of ray tracing, the application must adapt to the new changes. This section explains how the ray tracing concepts can be implemented.

4.1 Program Structure

The application is implemented in Unity. Unity is a popular game engine that is widely used, well documented, and many resources online explain and discuss its various components. So the application can be easily extended.

This application includes two different rendering methods. The ray tracer and the 3D Unity application. Originally the ray tracer was used to render the preview screen as well as the rendered image. Unity was then used to render the scenes with rasterization. In this implementation, the rendered image is also rendered by the Unity application. Because Unity allows us to implement ray tracing, we can now render the image without the need for VRT's ray tracing method. So it simulates light behavior based on real-world physics principles considering factors like light intensity, color, distance, and materials' properties to calculate the final lighting results. VRT uses Phong's shading model, which has components like ambient, diffuse, specular, and shininess. Ambient decides the constant lighting regardless of position of the lights in the scene. Diffuse simulates the way light scatters when it interacts with a rough surface. Specular simulates the reflection of concentrated light. So this is responsible for highlighted parts of objects. Shininess mainly controls the size of the specular highlight. Unity offers metallic and smoothness components for materials. The metallic component is used to decide how "metal-like" a surface is. When the component is higher, the environment is reflected more and its albedo color becomes less visible [10]. The smoothness controls the "microsurface detail" controlling how light scatters. This differs a lot from VRT's model so that is also a reason to let Unity

render the final image.

4.2 Preview Screen

Now only the preview screen is rendered with the ray tracer. To let the preview screen be accurate the metallic and smoothness values have been mapped to Phong shading values. We can not directly map values from Unity to the Phong shading model, because the metallic and smoothness both control how reflective the material is in a different way. But we can come very close. As stated in the paragraph above, if an object becomes more metallic, it will reflect the environment more and its albedo color becomes less visible. When and object becomes more diffuse, the lit parts will become clearer, making its albedo color become more visible. So we mapped the metallic value to the inverse of the diffuse value. The smoothness value controls how smooth an object is. This has an effect which does not compare to the specular component, however it does work well to map this to the shininess component. By using the shininess value on six different smoothness values the following formula can be created, where x represents the smoothness value on the slider in the application:

$$77.9856 * x^{0.00135846} - 77.5856 \tag{1}$$

There is one more slider in the application: the ambient component. To recreate the ambient from Phong shading we add the color times the ambient value to the color. So we use this equation:

$$color = color + ambientValue * color \tag{2}$$

Unity allows *overflow* of color. This will result in the object being too bright and having an incorrect color. Figure 6 shows how ambient overflow can cause spheres to have an incorrect color. To prevent this from happening we store the maximum color with ambient added and divide the RGB values with it so the maximum RGB value will be 1 and the other values will be smaller with the corresponding ratio.

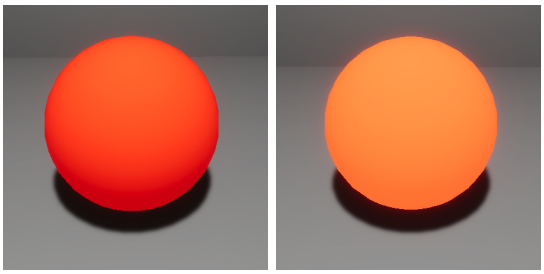

Figure 6: Ambient overflow will cause the red sphere to look slightly orange

With these Unity components mapped to the Phong shading model, the render preview screen will be similar to the scenes even though they are rendered differently. This unfortunately does not mean that the screen is perfect. Colors might be slightly off in some cases due to the different shading models. In the user study (see Section 6.1), we will see if this has affected the users. Figure 7 shows how the render preview looks in front of the scene.

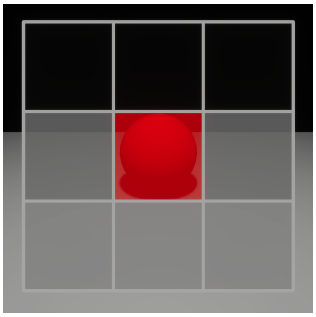

Figure 7: Render preview shows the colors of the scene on the corresponding pixels

4.3 Unity's Rendering Pipeline

To implement ray tracing, Unity developed the *High Definition Rendering Pipeline* (HDRP). This is a high-fidelity Scriptable Render Pipeline built by Unity for modern (Compute Shader compatible) platforms [4]. Currently Unity has two different pipelines, a user can choose between HDRP and URP [11]. URP stands for Universal Render Pipeline and is the older rendering pipeline. URP is currently more used because it requires less computational power and is supported on more platforms. We expect that when commercial PCs get computationally stronger HDRP will become the main rendering pipeline. The physically-based lighting techniques from HDRP should already allow for more realistic applications.

The shaders that are present in the original VRT tool are not compatible with HDRP and have therefore been updated to HDRP compatible shaders. HDRP offers a great variety of shaders for different materials [12]. We believe the default shader to be the best shader for VRT. This creates realistic materials and includes all options we would like to use. The shader should also hold all values that we use for the render preview screen. That is why we copied the HDRP/Lit shader and added all values such as ambient and diffuse to it. This way, the shader is compatible with Unity's rendering and also with VRT's rendering.

On top of that, since October 2021 Unity supports ray tracing. This feature is still in development and is not officially finished. This means that everything that is now implemented with ray tracing is a preview and can change in the future. These features are only available for ray tracing compatible GPUs. This has the disadvantage of making the application less accessible. However, we believe that due to the rapid evolution of GPUs and ray tracing it will become more accessible in a few years. In the following sections we will look at different implementations of Unity's ray tracing.

HDRP constructs an axis-aligned grid specifically for ray tracing. In each cell it maintains a list of Lights to retrieve if an intersection occurs within that cell. This is called a 'Light Cluster' [13]. The center of this cluster grid will be on the camera's position. It is impossible to make this cluster infinite, therefore we can change the Camera Cluster Range. If this value is low some parts of the scene will seem to be unlit. We therefore increased this value to a value where all objects and lights in the scene are considered properly.

4.4 Global Illumination

To recreate the effect of indirect lighting Unity uses Screen Space Global Illumination (SSGI). SSGI gathers and approximates the indirect lighting contributions from nearby surfaces visible on the screen. Because it only considers what is on screen it is not ray traced. However, in the options of SSGI it is possible to enable ray traced global illumination which completely changes the implementation of global illumination [14]. When enabling this, the renderer will evaluate indirect diffuse lighting. This means that Unity will now trace rays to all objects and bounce off them. We have included a checkbox that lets the user enable and disable global illumination. Figure 8 shows an example of global illumination. Here the blue wall is not lit from below. But

since the green floor is lit, the wall has a green tint.

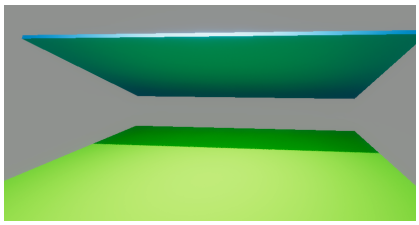

Figure 8: Example of Ray Traced Global Illumination.

4.5 Reflections

When using rasterization for rendering it is almost impossible to create good reflections. So when VRT renders a reflective plane, it will render a black plane. It will show the rays bouncing off this mirror, but ideally, we would want to see the reflections in the scene ourselves. Figure 9 shows a reflective plane in VRT.

We aimed to create a scene where reflections were always visible. Unity allows for *recursive rendering* [15]. Recursive rendering is a replacement pipeline for rendering Meshes in HDRP. Recursive rendering is used for both reflections and refractions. Recursive rendering is ray tracing done by Unity. The light rays are traced and when a reflective object is hit, the light ray will bounce off. If an object is not reflective/refractive the information of that point will be returned. To control performance, there are several options available. Most importantly, the times a ray is allowed to bounce recursively can be changed. When increasing this, the performance might take a big hit. That is why the maximum allowed number of bounces is 9 in Unity. When using VRT with Unity's ray tracing we can change this recursion depth at runtime. Figure 10 shows a scene where two reflective planes are standing opposite of each other and the number of bounces is set to 9. Rays should stop going in a straight line eventually when nothing is hit. Then it should return that there is no object hit and the background should be displayed. We can also change how long a ray is allowed to go without hitting an object. Since ray length is not easily observable we have decided to use a fixed ray length where everything in the scene will be visible when reflected. So far, we have not seen any performance issues by using these options.

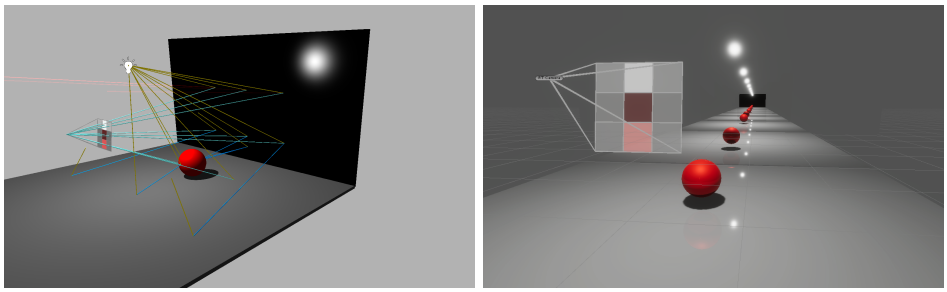

Figure 9: Reflections scene in VRT (left) and Unity HDRP (right)

4.6 Refractions

Refractions work similarly to reflections. They are also generated with recursive rendering. When a light ray is going through transparent material, it might refract. For example, light will change direction when going through the water's surface. How much light is refracted is decided with the *index of refraction*. In the properties of the material of a mesh, it is possible to change the surface type to transparent. When doing this, the refractive options become visible. Here it is possible to control the index of refraction. The refractive options also include the

possibility to choose a *Refraction Model*. This can either be a box, a sphere, or thin. Refraction models are used to control the refraction of the light rays. Using this we can create all types of transparent objects that refract light. With this, the tool can show ray traced refractions. Of course, the refractions in the scene should match the refractions from VRT's ray tracing method. Unfortunately, this will not be the case when not altering the refractive index. The refractive indices do not have to be altered for values below 1.00. But we apply this equation for refractive indices between 1.00 and 1.20, where x represent the refractive index on the slider in the application:

$$56.1237 * x^{0.0370227} - 55.1144 \tag{3}$$

By comparing the refractive index values for Unity and VRT we could see when the refractions were the same. This was used to create points. These points were used to create the above equation. Using this equation the refractive indices match the refractions from VRT. For every refractive index above 1.20 adding 0.19 matches the refractions.

Figure 11 shows the difference between a refractive sphere in VRT's and Unity's recursive rendering approach.

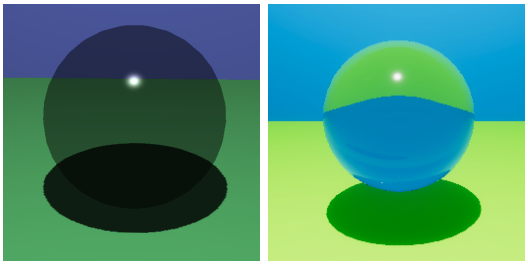

Figure 10: Refractive sphere in VRT (left) and Unity HDRP (right)

4.7 Shadows

To create fully ray traced scenes, the shadows should also be rendered using ray tracing. Unity does this by tracing rays of light from the light source. Theoretically, this would not give very smooth shadows since VRT uses *point lights*. Point lights are the simplest form of lighting [16]. The only attributes considered for point lights are position and intensity; the light shines from a certain position with a certain intensity in all directions. This is also how the lighting is evaluated in Unity. But Unity evaluates the shadowing from point lights as if the light was coming from the surface of a sphere. This can create smoother shadows. Unity offers some shadowing options [17]. In this application we make use of the *sample count*, which controls the number of rays that HDRP uses to evaluate the shadows per pixel, per frame. So if the value is higher, the shadows will be more accurate. The other option that we use is *denoise*. Denoise enables a spatio-temporal filter to remove noise from the ray traced shadows. This spatio-temporal filter denoises in space and time. The filter analyzes the neighboring pixels (in space) and analyzes the same pixel in sequential frames (in time). These options give us the desired ray traced shadows. An example of this can be seen in Figure 11.

5 Limitations

As mentioned in Section 1, ray tracing was not used in real-time until recently. This means that the developments that Unity has made are all fairly new. All ray tracing features from Unity are in a state of *preview*, meaning that they can still change in the future. Apart from that, Unity has taken some measures for performance reasons. As mentioned in Section 4.3, ray tracing features are only supported on ray tracing compatible GPUs. On top of that, Unity's web development tool *WebGL* does not support these features either. Meaning that it is not possible to deploy a web version, unlike VRT. VRT is mainly meant to use in the course Computer Graphics. So it is unfortunate that a ray tracing compatible GPU is needed. However, we do believe that in the near future ray tracing compatible GPUs will be commonly used. We believe that deploying to the web will become possible with the release of *WebGPU* [18]. WebGPU is currently under development by W3C. It will use physical GPU hardware to run high-quality graphics on the web. This will therefore be perfect for deploying VRT with real-time ray tracing.

Windows uses the Direct3D12 Graphics API while Linux uses Vulkan and Mac uses Metal. The project was developed on Windows and is able to run all functionality that is described in this thesis. The other APIs however are not able to run all ray tracing features. This means that currently, the project is not able to build on Linux and Mac, so currently Windows is the only supported platform. Again we do believe that this will shift in the future. Ray tracing is relatively new, but it is very likely that the developers from Unity want to expand these features to all platforms.

Another performance measure Unity takes is disabling ray traced shadows on objects that have recursive rendering enabled [15]. Ray traced shadows are made by tracing rays from light sources. If these rays had to bounce off the object again this would impact performance. Because of this, the shadows are then made with rasterization instead of ray tracing. This means that in every reflection/refraction, the objects are rendered using ray tracing but the shadows are rendered using rasterization. We believe that this will change in the future, but it would impact performance too much on current GPUs.

6 Evaluation

One of the important things about this application is that it is reactive. The application should immediately react when a user inputs something. With ray tracing this can be tricky, because ray tracing is a computationally expensive task. However, as said in Section 5, the ray tracing features are currently only allowed on ray tracing compatible GPUs. Therefore the program has only been tested on an RTX 3070ti and an RTX 2070. These systems were both able to run the program with stable framerates. The framerates were around 60 fps. VRT has really simple scenes especially when we compare it to something like Syberia: The world before (see Section 2.2). Unity allows for ray tracing to be used in more complex games that also seem to run well commercially, so VRT should be able to run relatively well.

6.1 User study

This application is designed to be used in the course Computer Graphics. So to get the most useful insights we have asked Computer Science students to fill in a small survey after trying both the old VRT and the real-time ray tracing version. Since real-time ray tracing is limited to high-end GPUs we only have 6 results, but this still gives us a good view of what people think about the application.

All participants of this survey have followed the course Computer graphics. Most of them have also used the old VRT, which either helped them understand ray tracing better or they already understood it well beforehand. The most important part of this application is improving the learning experience. That is why we asked the participants if they think implementing real-time ray tracing can help other people help understand ray tracing better. All participants stated that they think that it improves the application a lot. We have asked the participants about most ray traced features. This resulted in most participants saying that all features become a lot clearer. For shadows, 2 people said it only becomes a bit clearer or there is no significant difference. This is a fair assessment since shadows are visible in rasterized scenes as well, just in a lower resolution. Meaning that it might be equally clear how shadows are rendered, but in ray tracing it is rendered in higher resolution. Due to the different lighting models that are used between the scene and the render preview (see Section 4.2) someone might spot mismatches. That is why we asked the participants if they have noticed any mismatches. One person actually noticed some slight mismatches. Even though this is unfortunate, this is to be expected. And because mismatches are still possible, if this application is going to be extended, the render preview will likely be rendered using Unity's ray tracing as well. This will completely remove the possibility of mismatches. Lastly, we asked what they thought about the visuals. This is also something that has improved between the old VRT and the current VRT, so it is something that we want to check. Luckily, everyone thought the visuals were either good or very good.

7 Summary and conclusions

We have presented a real-time ray tracing variant of Virtual Ray Tracer [3], implemented with Unity's ray tracing. The application was designed to help with teaching ray tracing and to be used in the course Computer Graphics at the University of Groningen, and hopefully also beyond that.

This application has implemented a physically-based lighting model that replaces the Phong model for rendering. We believe that this model can become the new standard for VRT. This application also has real-time reflections. Similarly, it also has real-time refractions. Meaning that the user can see how ray traced reflections and refractions look at all times when navigating the scenes.

We believe that this makes the tool go a step forward. There can not be any more mismatches between the scene and the rendered image. We believe that this is good for learning ray tracing. The application was evaluated with a small user study. So we can also look at opinions of some users. The user study was conducted by only a few people due to the high specifications that are needed to run the application. The users were very positive about the application. All of them think that implementing real-time ray tracing in the scenes can increase the learning experience greatly.

As the developed application is not very accessible right now, it is not likely that it will be used in Computer Graphics 2023-2024. But this project has shown the potential of real-time ray tracing. It shows that this can likely improve the learning experience for students. So with the rapid evolution of GPUs and ray tracing it is very likely that this application or even an extension of this application can be used to help future students of Computer Graphics in a few years.

8 Future Work

In this chapter, we discuss potential improvements to the real-time ray tracing version of VRT.

Preview Screen
As explained in Section 4.2, the preview screen is currently rendered using VRT's ray tracing instead of Unity's ray tracing. This can cause mismatches so it would be beneficial to replace the ray tracing that is used to Unity's ray tracing.

Extra Explanations
The explanations at the start of every level have basically stayed the same. These explanations have not been altered to talk about real-time ray tracing. It is probably useful for users to get some extra explanation about real-time ray tracing.

Evolve With Unity's Ray Tracing
Unity's ray tracing is still evolving. This means that the features that we have implemented are likely to change. New features might also be added. When this happens it might be useful to integrate those features into VRT as well. We would like to optimally use the features that Unity provides to create visually pleasing and functional ray traced scenes.

Larger User Study
As GPUs and ray tracing evolve we would like to ask more people to use this application. This can give more insights about it. Eventually, it would be great if this application could be used for Computer Graphics and some more user input could help improve it.

9 Acknowledgements

I would like to thank my supervisors Jiří Kosinka and Steffen Frey for helping me by answering questions, providing me with some useful documentation and providing great feedback. I would also like to thank Chris van Wezel for giving a small introduction to VRT and helping me with questions about it. Lastly, I would like to thank the participants of the survey for giving me useful insights about the application.

References

[1] NVIDIA RTX : Top 5 des jeux qui utilisent bien le ray tracing (et 5 qui le font mal). https://www.omgpu.com/nvidia-rtx-top-5-des-jeux-qui-utilisent-bien-le-ray-tracing-et-5-qui-le-font-mal/. Accessed: 07-03-2023.

[2] Eric Haines and Tomas Akenine-Möller. *Ray Tracing Gems*. Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, 2019.

[3] Willard A. Verschoore de la Houssaije, Chris S. van Wezel, Steffen Frey, and Jiří Kosinka. Virtual Ray Tracer. In Jean-Jacques Bourdin and Eric Paquette, editors, *Eurographics 2022 - Education Papers*. The Eurographics Association, 2022.

[4] Getting started with ray tracing. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@16.0/manual/Ray-Tracing-Getting-Started.html. Accessed: 21-02-2023.

[5] Josef Bo. *Efficient Ray Tracing Architectures.* ProQuest LLC, 789 East Eisenhower Parkway, 2015.

[6] An interactive web-based ray tracing visualization tool. https://courses.cs.washington.edu/courses/cse457/01au/applets/seeray/thesis.pdf. Accessed: 15-06-2023.

[7] Jeremy Fisher, Daniel Eby, Edward Quigley, Gideon Ludwig, and Christiaan Gribble. Ray tracing visualization toolkit, 08 2011.

[8] Syberia: The world before. https://www.microids.com/game-syberia-the-world-before/. Accessed: 28-03-2023.

[9] Raising the bar on unity's visual quality. https://unity.com/demos/enemies. Accessed: 19-06-2023.

[10] Metallic mode: Metallic parameter. https://docs.unity3d.com/Manual/StandardShaderMaterialParameterMetallic.html. Accessed: 13-06-2023.

[11] render pipelines feature comparison. https://docs.unity3d.com/Manual/render-pipelines-feature-comparison.html. Accessed: 13-06-2023.

[12] High definition render pipeline overview. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@8.0/manual/HDRP-Features.html#Material. Accessed: 16-06-2023.

[13] Light cluster. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@10.2/manual/Ray-Tracing-Light-Cluster.html. Accessed: 19-06-2023.

[14] Using screen space global illumination. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@10.2/manual/Override-Screen-Space-GI.html. Accessed: 13-06-2023.

[15] recursive rendering. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@10.2/manual/Ray-Tracing-Recursive-Rendering.html. Accessed: 13-06-2023.

[16] Types of light. https://docs.unity3d.com/Manual/Lighting.html. Accessed: 16-06-2023.

[17] Ray-traced shadows. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@12.0/manual/Ray-Traced-Shadows.html. Accessed: 16-06-2023.

[18] Webgpu w3c working draft. https://www.w3.org/TR/webgpu/. Accessed: 15-06-2023.