RAY TRACING TEACHING TOOL

An Undergraduate Research Scholars Thesis

by

MICHAEL R. STEWART

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Shinjiro Sueda

May 2022

Major: Computer Engineering

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Michael Stewart, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Ray Tracing Educational Tool

Michael R. Stewart
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. Shinjiro Sueda
Department of Computer Science and Engineering
Texas A&M University

Ray tracing is a complex topic lacking educational and interactive tools for both classroom demonstration and student independent study. Ray tracing is a technique of mapping rays of light in each given scene as it bounces off walls and subjects, casting shadows and producing reflections that produces more real-life graphical representations. There is considerable interest in the graphics community to learn more about ray tracing to develop the desired level of photorealism in production and interactive rendering programs. The industry acceptance and resulting photorealism makes ray tracing an important concept for Computer Graphics students to learn. Ray tracing concepts are taught in introductory and advanced Computer Graphics classes around the world. The concepts associated with ray tracing are challenging to teach through slides as there are no intermediate visual output steps to aid students' understanding of the concepts. The need for an interactive instructional tool that allows for camera controls, contains presentational aids and is easily accessible to teachers and students is high. The goal of the project is to build a tool that will aid in teaching ray tracing concepts to students.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my research faculty advisor, Dr. Shinjiro Sueda for his patient guidance and support throughout the course of this research. He has been a steady mentor who was generous with his time advising me and consistently encouraged me while working on my project.

Thanks also go to my friends and colleagues in the TAMU SIGGRAPH Chapter who helped grow my interest and awareness of computer graphics and the vast future possibilities in the field. I appreciate the TAMU Computer Science department faculty and staff for making my time at Texas A&M University a wonderful and valuable experience.

Funding Sources

I did not have additional funding sources for this project.

NOMENCLATURE

ACM	Association of Computing Machinery as a Science and Profession
API	Application Programming Interface
B/CS	Bryan/College Station
Buffer	Dedicated computer memory location for storing specific data associated with a program
BVH	Bounding Volume Hierarchy is a way to organize the geometry in a scene into a Data structure
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
JSON	Java Script Object Notation
Normal	In geometry, a normal is a ray or vector that is perpendicular to a plane or object. In computer graphics, normals are direction of an object’s surface toward light.
OpenGL	Open Graphics Library
PBR	Physically Based Rendering is a computer graphics approach to render images.
Shader	A program which shades or determines the color of a surface, based on normal light direction, angle to camera, etc.
SIGGRAPH	Special Interest Group on Graphics, a subgroup of Association of Computing Machinery (ACM)
TAMU	Texas A&M University
WebGL	Website Graphics Library
Vertex	A point in a geometrical solid common to three or more sides

1. INTRODUCTION

1.1 Prevalence in Industry

Computer graphics has produced various technologies to generate images in a wide variety of applications. As computer generated images look more realistic, they are being used for scientific research and modeling as well as for entertainment value. Ray tracing is a technique of mapping every ray of light in each given scene as it bounces off walls and subjects, casting shadows and producing reflections that produces more real-life graphical representations. There is considerable interest in the graphics community to learn more about ray tracing to develop the desired level of photorealism in production and interactive rendering programs. Monsters University, a Pixar film released in 2013, was one of the first fully ray traced computer-animated films. The new ray tracing lighting system imitated the behavior of real-world light, delivered more realism and soft shadows. The advancements of computer graphics technologies have considerably eased the efforts required from the artists, which allows them to spend more time on models and other complex scenes. Pixar changed Renderman, an in-house rendering software to utilize ray tracing technologies for lighting effects. This began the industry shift towards ray tracing techniques. Further progress has been made as NVIDIA has included ray tracing capability in their Graphics Processing Unit (GPU) hardware making ray tracing possible for real-time graphics programs. Ray tracing popularity motivated NVIDIA to partner with several authors to create and publish two books, *Ray Tracing Gems 1* and *Ray Tracing Gems 2*, resulting in more than 70 pages of published information [1]. Advancements in hardware technologies have also enabled real-time ray tracing to be used in graphics rendering engines and interactive game productions.

1.2 Prevalence in Class

The industry focus and resulting photorealism makes ray tracing an important concept that Computer Graphics students need to learn. Ray tracing concepts are taught in introductory and advanced computer graphics classes around the world. The research paper *What we are teaching in Introduction to Computer Graphics* [2] surveyed 224 world-wide universities and selected the top 20 universities which had the largest impact in terms of graphics research. This paper then examined their courses for graphics concepts and teaching techniques. The paper reports that OpenGL is taught at many universities but notes its difficulty as a learning resource. They also note that 90% of the surveyed universities taught ray tracing and the graphics pipeline. This shows a clear prevalence of these topics in university courses. The concepts associated with ray tracing are challenging to demonstrate with OpenGL and/or through slides because there are no intermediate visual output steps to aid student's understanding of the concepts. As noted in the paper, *Incorporating Modern OpenGL into computer Graphics Education,* it is even harder now, with modern OpenGL, to achieve the first satisfying visual output, when compared to previous legacy versions [3].

There is a high need for an interactive instructional tool that allows for camera controls, contains presentational aids and is easily accessible to teachers and students. The goal of the project is to build a tool that aids in teaching ray tracing concepts to students. This is important because ray tracing use is growing in several areas. As computer graphics result in more photorealistic images, more scientific and entertainment applications are using computer generated graphics in wider and wider applications. While offline rendering has been in practice for many years, the real-time graphics has begun using ray tracing capabilities built into hardware, such as Graphics Processing Unit (GPU) hardware, now produced by a variety of vendors. Ray tracing concepts continue growing in popularity and demand for knowledge of the technology is increasing in importance. There is considerably more interest in ray tracing technology than in the past. Academia needs more teaching aids to help students more easily grasp the concepts.

1.3 Other Tools Available (khan + Pixar) Rayground

In the 2020 publication titled *Rayground: An Online Educational Tool for Ray Tracing*, the authors make the point that there are few specialized educational tools that exist to help students quickly learn the needed concepts [4]. Rayground.com is a website with an integrated development environment (IDE) that demonstrates ray tracing. The site also contains a paper discussing how it may be integrated into computer graphics class lessons. Rayground's IDE includes a code editor and resulting display window. This type of tool benefits students as it runs in a standard web browser and does not require specialized software licenses or installations. At this point, Rayground lacks instructional presentation aids that enable teachers or students to walk through the ray tracing process in an ordered, step-by-step process *in 3D*. A tool that makes visible the underlying steps to produce a specific image would benefit both teachers and students. Interactive engagement with the tool allows students to grasp the step-by-step process more easily and would encourage student exploration and curiosity. Note, however, that Computer Science teachers would typically not want the source code for an assigned project the student is to build to be exposed, as this tempts students to cheat and copy the code.

One current example of an interactive graphics education tool is the Khan Academy and Pixar partnership series *Pixar In A Box* [5]. In the rendering 101 section of the course, there is a video in which they show a tool that has an interactive view of rays being shot through an object.

This series is helpful in the explanation of what ray tracing is, but the authors do not make this tool available to students and it is quite simplistic visually.

1.4 Background

To better understand this thesis, it is important to have a basic understanding of computer graphics. This paper and the accompanying software program will feature computer generated images. The process of computers generating images is called rendering. A computer renders into pixels the image described by the scene. Computer-generated rendered images use three-dimensional (3D) model objects to develop a scene. These 3D objects can be rendered with lighting to describe their shape and create a realistic look, including shadowing. For the computer to understand these objects, they are described with vertices and triangles. As an example, a teapot in real life is made of trillions of particles which interact with light. The wavelengths of light that are not absorbed by the material are reflected, absorbed by our eyes, and interpreted by our brain. Today’s generation of computer technology struggles to replicate this computationally intensive process; instead, technology aims to approximate it. Computers describe objects with a series of points (vertices), then connect three of these points (vertices) together to form a triangle with a surface plane. A series of triangles arranged correctly can adequately describe an object, such as the teapot. When rendered, the image will appear to the human eye to look just like a teapot with appropriate shading.

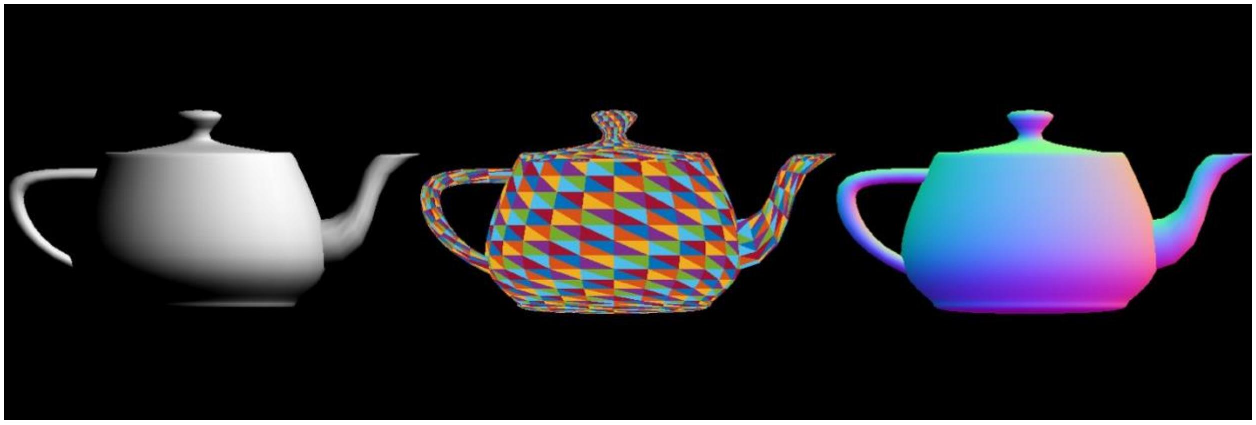


Figure 1.1: The left teapot is lit using rasterized lighting. The middle teapot illustrates the triangles used to describe the teapot. The right teapot is a rasterized view of the normal of the model.

In Figure 1.1 above, three teapots are shown. The teapots on the left and the right are to help the reader understand what they are viewing. The middle teapot image is used to illustrate the use of triangles to describe a model concept discussed in the previous paragraph. In this figure, all three teapots are rendered using a rasterizer process. The left is a black and white image of a teapot. The right most image is a teapot with the colors showing the normal direction of the surface of the teapot. This image shows the shape of the teapot which was in the shadows in the left most teapot view. The colors represent directions of the teapot's surface pointing away from the surface. Red is pointing to the right, blue is pointing to the left, green is pointing up, and purple is pointing down. The teapot image in the middle describes every triangle on the model with a different color. This illustrates how a bunch of triangles can be used to describe an object.

Next, we will discuss two methods of rendering: Rasterization and Ray Tracing.

1.4.1 Rasterization

Rasterization is a method of creating images of a 3D scene. It determines how to draw triangles as a grid of pixels. As described in *Fundamentals of Computer Graphics* [6] rasterization is a technique that can be implemented with minimal computer time and complexity. Its low computing cost allows for real time and interactive rendering, making it a popular choice for video games. Rasterization is more limited than ray tracing as it does not support light simulation. Instead, it uses a series of techniques to produce a similar effect. These limitations are what give images a “video game” look. Though ray tracing can be more realistic, it has a higher computing cost. Therefore, for more real time rendering situations, rasterization is generally preferred.

1.4.2 Ray Tracing

Ray tracing popularity motivated NVIDIA to partner with several authors to create and publish two books, *Ray Tracing Gems 1* and *Ray Tracing Gems 2* [1], resulting in more than 70 pages of published information. Advancements in hardware technologies have enabled real-time ray tracing to be used in graphics rendering engines and video game productions.

Though the process of ray tracing is different from rasterization, it is still a way of describing data used to render an image.

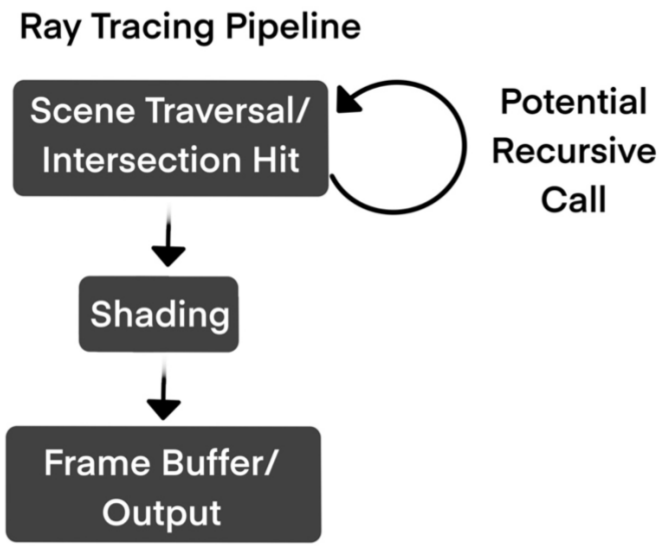


Figure 1.2: Ray Tracing Pipeline

Figure 1.2 above describes the pipeline for ray traced rendering. The purpose of my research and this thesis is to develop a tool which can better communicate this concept. Ray tracing is difficult to fully understand without visually seeing progressive steps associated with it. This explanation may illuminate why this program will be helpful to Computer Science graphics professors and students working to learn more about ray tracing.

From the camera's lens point a ray is projected through the center of a pixel and then traverses through the scene. An algorithm is used to determine if the ray intersects with a triangle within the scene. Here is an example of a formula which can be used to determine triangle intersections:

$$a(X_a + t * X_D) + b(y_A + t * y_D) + c(z_A + t * z_D) + d = 0, solve\ for\ t \qquad \text{(Eq. 1.2)}$$

The ray is checked against triangles in the scene. If there are multiple intersections (hits) in the scene, then the hits are sorted to find the closest hit to the camera. When the closest hit is detected, that hit becomes the new starting position as a ray is shot towards each light in the scene. If the ray is able to traverse all the way to the light source, without hitting any other model, then a lighting calculation is made to determine the color of the pixel, based on the angle of the ray. If a ray hits an object model on its way to the light, then no lighting calculation is made, which means the point is in shadow. This simple rule set can produce powerful results. One of the major improvements that ray tracing provides over rasterization is that shadows are calculated across the scene. Not only are shadows being cast from one object to others in the scene, but shadows are also cast from a model onto itself.

With light simulation, it is possible to assign different materials that change the behavior of the light slightly. For example, to achieve a mirror surface, send out a ray at an angle opposite to that of the incoming ray that hit on the surface of the triangle. This will then start the process of light bounce as described previously. Without much effort, you can achieve similar effects such as soft shadows and refractions.

2. METHODS

My project leverages work from my Computer Graphics class on a program developed in C++ and OpenGL. To enable the program to run on a web browser, I use JavaScript and Web Graphics Library (WebGL). JavaScript, a programming language and WebGL, a 2D/3D graphics library API for JavaScript are programming languages new to me.

2.1 Source Code Control and Web Hosting

GitHub is used for source code revision control and for web hosting the Ray Tracing Teaching Tool.

2.2 Bounding Spheres

The tool performance is improved using a bounding spheres technique. Calculating ray tracing intersections is computationally expensive, especially on models with thousands of triangles. For these cases multiple optimizations are made in this program to improve performance. For all models which are not primitive shapes, a bounding sphere is used to help improve computational performance. A bounding sphere encompasses the entire object. If the entire object is inside of the bounding sphere and if a ray hits the sphere, then the object has the chance of appearing in that ray’s path. But if the ray does not hit the sphere, the object will not be in that ray’s path, so the program can safely ignore it. It is more computationally efficient to check a bounding sphere than it is a check the thousands of triangles composing the object within the bounding sphere. If the ray does hit the bounding sphere, then the program will check all triangles on the model within the sphere for ray intersection. This greatly improves computational performance and is best demonstrated in the time spent to get an output at high resolution of scenes 6 & 7 in the program.

2.3 Background Coloring

One big difference between how the real time view rasterization and the ray tracer rendering works is in how backgrounds are created. In ray tracing the default pixel color is black and if a ray never hits any objects the pixel color remains black. In rasterization if no triangle occupies the pixel, then no data is added to the color buffer, so WebGL leaves a clear, or transparent pixel which will show whatever is underneath that web element, which will be the website background for this program. To help distinguish the real time view from the rest of the website, I use an HTML web element which colors black that section where the real time view canvas will be located. This is one of the ways rasterized renderings differ from ray traced renderings.

2.4 Primitives Representation in Rasterized Real Time View

Ray tracing can render primitive objects because it uses an intersection equation for those primitives. But in WebGL, real time rasterization triangle-based models must be used. To represent the ray traced images primitive objects, I created a triangle mesh representation that is visible in the real time view, aligning the scale, position, and rotation of these meshes with the primitive scale position and rotation in the rendered image. To make sure the triangle mesh data closely approximates the primitive ray traced object required an iterative trial and adjust development process.

2.5 Using JSON File Format For 3D Files

One major difference between C++ with OpenGL and JavaScript with WebGL is the format used to bring in the models. A common industry standard is the file extension format OBJ, commonly known as an object file format. My program uses Java Script Object Notation

(JSON), a light-weight format for storing and transporting data and for sending data to a web page. JSON is a self-describing and easy to understand format.

2.6 Static Pictures

To allow users to better understand what the ray traced view will look like, a high-resolution output is shown on the right side of the real time view in the program. This is to allow users to see what the ray tracing effect will look like without having a rasterized representation. A full 1000 by 1000 pixel image can take a long time to generate on a mobile computing device. To support quick scene switching and lower powered hardware to be able to view the full resolution render of the image displayed on the right, the image was rendered in advance and is displayed as a PNG file, instead of being rendered each time by the program when the user hits the spacebar. This is achieved by aligning an image tag object in HTML next to the WebGL rendering canvas and having the keyboard input change the assigned image.

2.7 Two Cameras

Inside of my program there are two camera views. The first camera represents the ray traced view shooting rays from the camera and is represented in the scene with the view frustum and the lines shooting through the corners of the frustum. The viewing frustum is the grid of colored squares visible on the screen. The second camera in the scene is the real time view camera that the user can control and manipulate with mouse movements. This allows a user to view the rest of the scene without disturbing the original position of the ray tracing camera. Supporting two cameras in the program requires establishing two systems: one for the real time view camera and one for the ray tracing camera.

2.8 Lines in the Real Time View

Lines in WebGL are created by specifying the starting and ending position with 3D coordinates. A program can push a set of these position points into the WebGL shaders and then specify to draw a line, instead of a triangle, and the specified points will be displayed as a line. This method is used in combination with a simpler vertex and fragment shader to produce all the lines in the scene, including the animated rays and the view frustum pixel boundary lines.

2.9 Blinking Selector

For a user to quickly recognize the path of a specific ray, it is helpful for them to identify specifically which pixel they are viewing. To achieve this effect, the program is set to display a highlight around the specific pixel selected for viewing.

I use a cosine function to alternate the colors of the highlighted square. This mechanism causes the square highlighting to switch from dark to light. Resulting in a visual indicator that should eventually be recognized by the user. If the color of the indicated square is white, then a black highlight is best. If the color of the square is dark, then a bright highlight is best for visual recognition. If the square's color is a middling brightness, then the outline is yellow, by subtracting a small amount from blue and green.

2.10 Tiled Squares on View Frustum

The colored tiles for the view frustum are created by using a square plane 3D model, made of two triangles. Using the fragment shader, I color the specific square with the color of the pixel that it represents. I then position the square model in the corresponding pixel's spot on the view frustum. This requires a simple translation along the X and Y axis. Triangles have a specific direction, noted by the normal of each face. This means that the back side of the square model would be transparent. To have the image frustum show up on both sides requires rendering another square model in the exact same position but rotated 180 degrees along the Y axis.

2.11 Animated Lines

To achieve the animated lines traversing through the scene, the animateLine() function accepts 6 arguments. The six arguments are: the ray starting and ending coordinates, start and end time, current time, the shader with which to draw the lines, and a Boolean value controlling if the line is yellow. With these input arguments, it is possible to construct animated lines. As mentioned in section 2.9, to draw a line you only need to send the start and end position to the vertex buffer. The way the line extends from the starting position is based on the current time. If the current time is before the start time, then the line will not be drawn. If the current time is greater than the end time, then the whole line will be drawn. It is only if the current time is between the start and end time that something different occurs. Using the (current time – starting time) / (end time – starting time) as the interpolation factor between start position and stop position, it is possible to get the interpolated stopping position for the line at that time. This will allow the line to grow as the time increases until it hits the end of the line where it will stay until all lines are drawn.

2.12 Ray Tracing on Top of WebGL

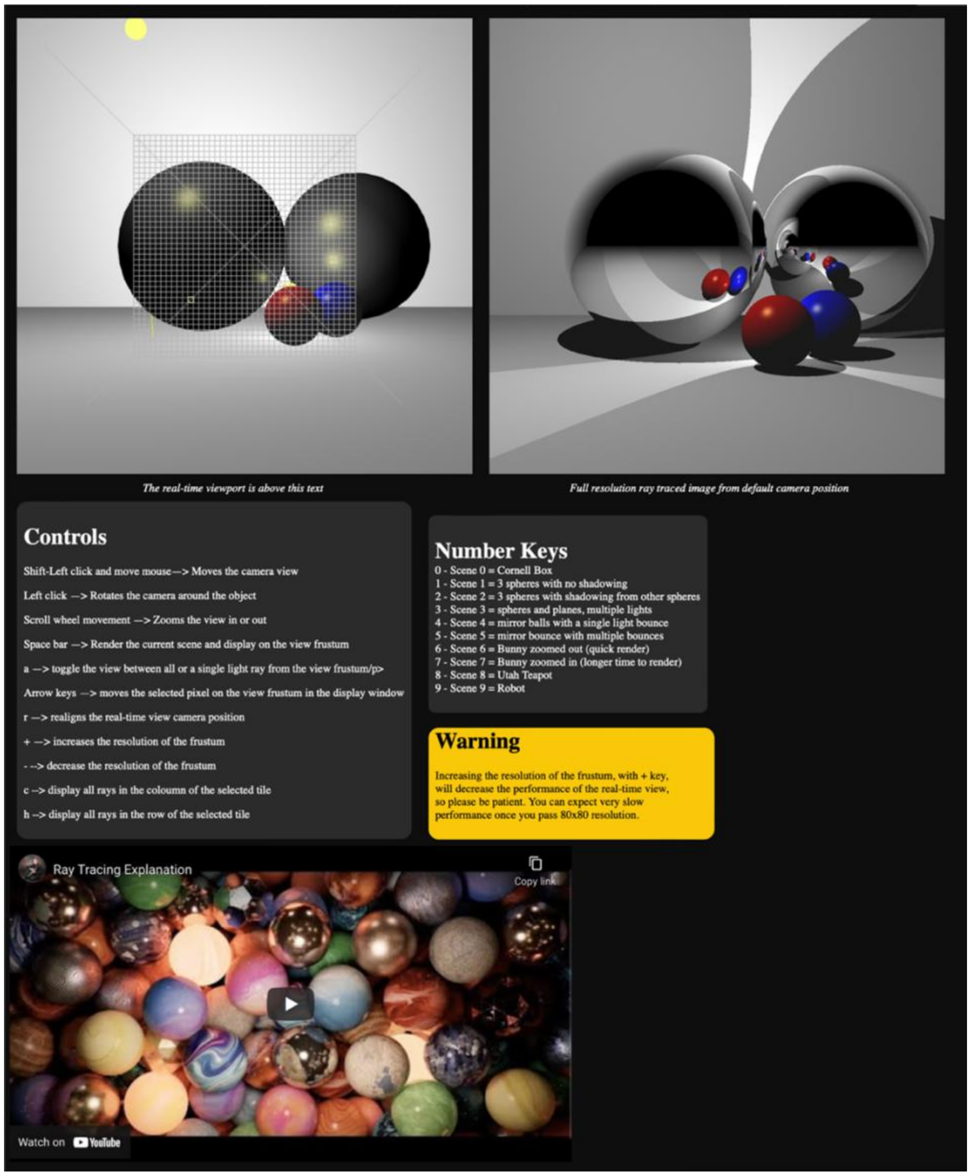
Two forms of rendering are running at the same time in this program. In addition to the WebGL real-time renderer, there is a ray tracing renderer running on top of the program. The ray tracing renderer runs every time the user presses the spacebar, changes the frustum resolution while retracing is toggled, or changes the scene while retracing is enabled. In these scenarios ray tracing is running adjacent to the WebGL renderer and outputs the results onto the view frustum-colored tiles.

2.13 Obfuscation

All JavaScript files in the Ray Tracing Teaching Tool have been obfuscated. This was achieved by combining all JavaScript files into one, then running the combined file through the website obfuscator.io. Obfuscation prevents students from reading the code and copying the logic for their assignments.

3. RESULTS

This section will discuss the software program I developed as a part of this research project. The program can be viewed at this link: https://michaelstewart2.github.io/RayTeaching/.



The real-time viewport is above this text
Full resolution ray traced image from default camera position
Ray Tracing Explanation
Copy link
Watch on YouTube

Controls

Shift-Left click and move mouse—> Moves the camera view

Left click —> Rotates the camera around the object

Scroll wheel movement —> Zooms the view in or out

Space bar —> Render the current scene and display on the view frustum

a —> toggle the view between all or a single light ray from the view frustum/p>

Arrow keys —> moves the selected pixel on the view frustum in the display window

r —> realigns the real-time view camera position

+ —> increases the resolution of the frustum

- --> decrease the resolution of the frustum

c --> display all rays in the coloumn of the selected tile

h --> display all rays in the row of the selected tile

Number Keys

0 - Scene 0 = Cornell Box
1 - Scene 1 = 3 spheres with no shadowing
2 - Scene 2 = 3 spheres with shadowing from other spheres
3 - Scene 3 = spheres and planes, multiple lights
4 - Scene 4 = mirror balls with a single light bounce
5 - Scene 5 = mirror bounce with multiple bounces
6 - Scene 6 = Bunny zoomed out (quick render)
7 - Scene 7 = Bunny zoomed in (longer time to render)
8 - Scene 8 = Utah Teapot
9 - Scene 9 = Robot

Warning

Increasing the resolution of the frustum, with + key, will decrease the performance of the real-time view, so please be patient. You can expect very slow performance once you pass 80x80 resolution.

Figure 3.1: Overview of Ray Tracing Teaching Tool Webpage

Figure 3.1 shows the landing page of my program. It can be run on any device which supports a web browser. A device with mouse and keyboard support is recommended to interact with the real time view. The top left window contains the real time view output. The top right window is the high-resolution output of the ray traced image of the current scene. When the user changes the scene, the full resolution view will automatically update. Under the first row of images is a box showing the operational controls for the program. The controls panel provides an on-screen aid describing which keys will perform which actions. On the right side of the screen is a numbers panel to explain which scene will be selected when specific number keys are pressed. The exact results of these controls will be discussed later. There is also a warning information box. The warning box is intended to help users understand what parameters in the scene might cause performance impacts. At the bottom of the display is an embedded YouTube video containing the video describing and demonstrating the program. This video provides a good summation of the program and should be a useful introduction to new users of this program.

This webpage has been designed to fit on most screen sizes. Users with a particularly large display may need to zoom in a bit, but the program output fits well on landscape and portrait-oriented displays. For smaller portrait displays the webpage will reorganize its contents to fit, placing every item in a single column. A typical screen size format looks like Figure 3.1 above.

3.1 Controls

3.1.1 Shift + Mouse Movement

When the user presses the shift key (left or right) then clicks and drags the mouse button, the position of the camera in the real time view port will be moved in the direction that the mouse has moved. This is illustrated in Figure 3.2 below.

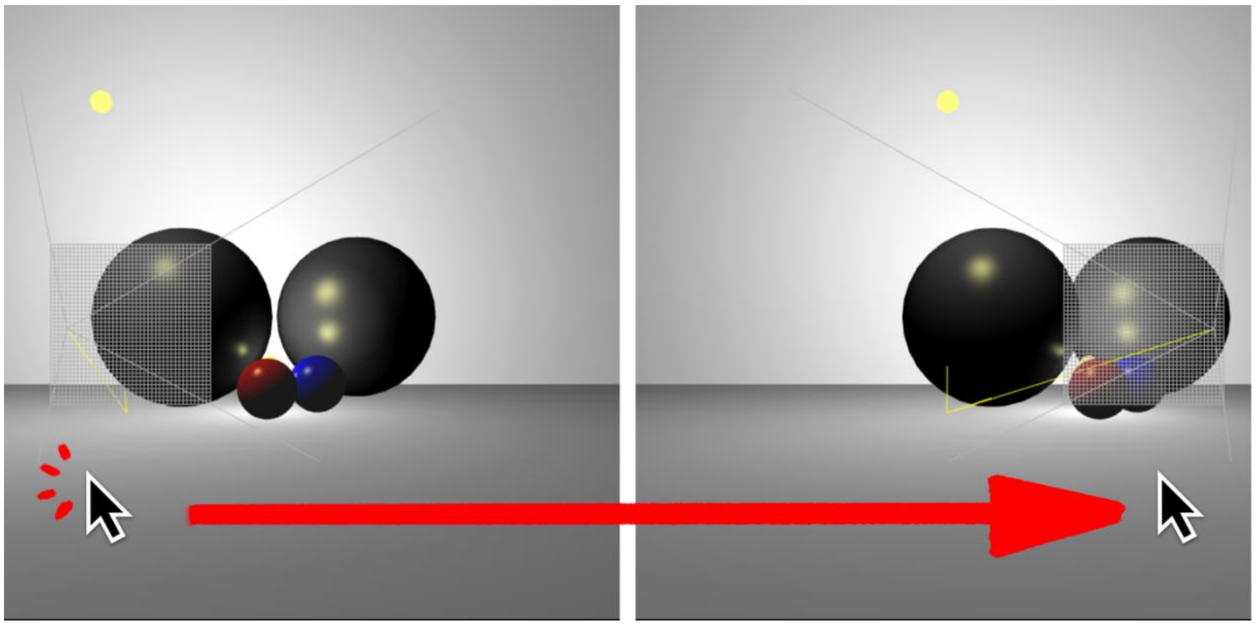


Figure 3.2: Translational Movement, Starting Camera Position (Left), Ending Camera Position (Right)

3.1.2 Left Click

Left mouse button clicking and moving the mouse allows the user to rotate the camera view around the center of the scene. Moving the mouse to the right will cause the view to rotate clockwise around the scene. Moving the mouse to the left will cause the view to rotate counterclockwise around the scene. Moving the mouse up or down rotates the view vertically. This will allow the user to be able to see a different perspective on the scene to help them understand the object's geometry. This is demonstrated in Figure 3.3 below.

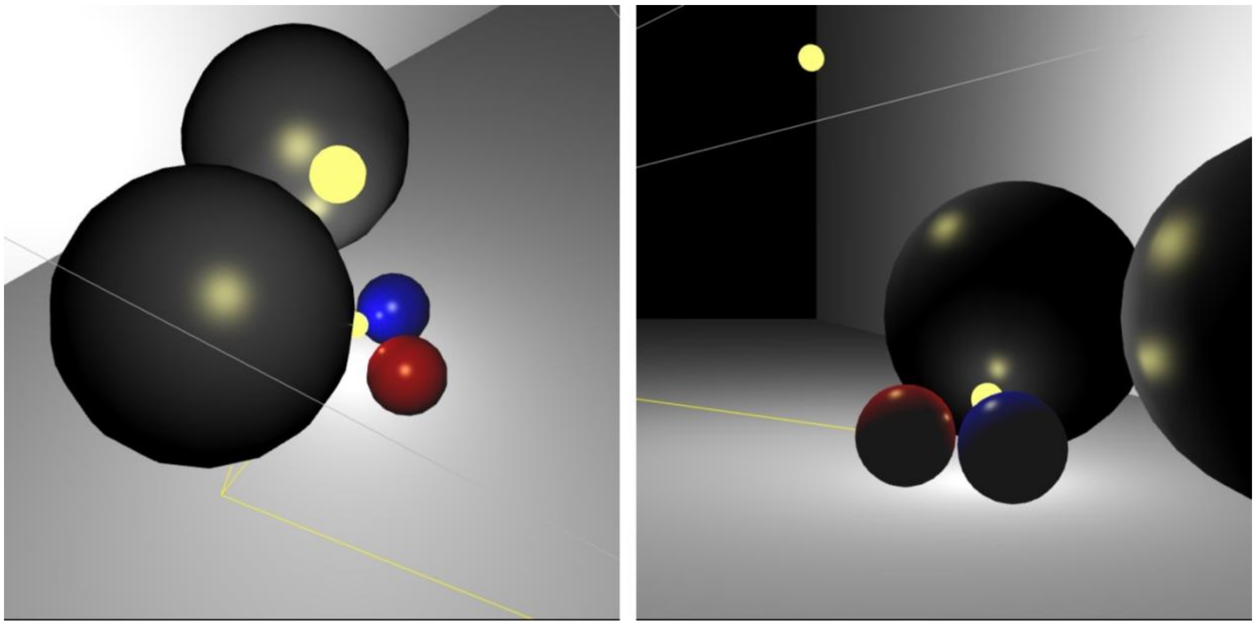


Figure 3.3: Rotational Transformation with the Mouse, Starting Camera Rotation Position (Left), Ending Camera Rotation Position (Right)

3.1.3 Scroll Wheel / Trackpad Movement

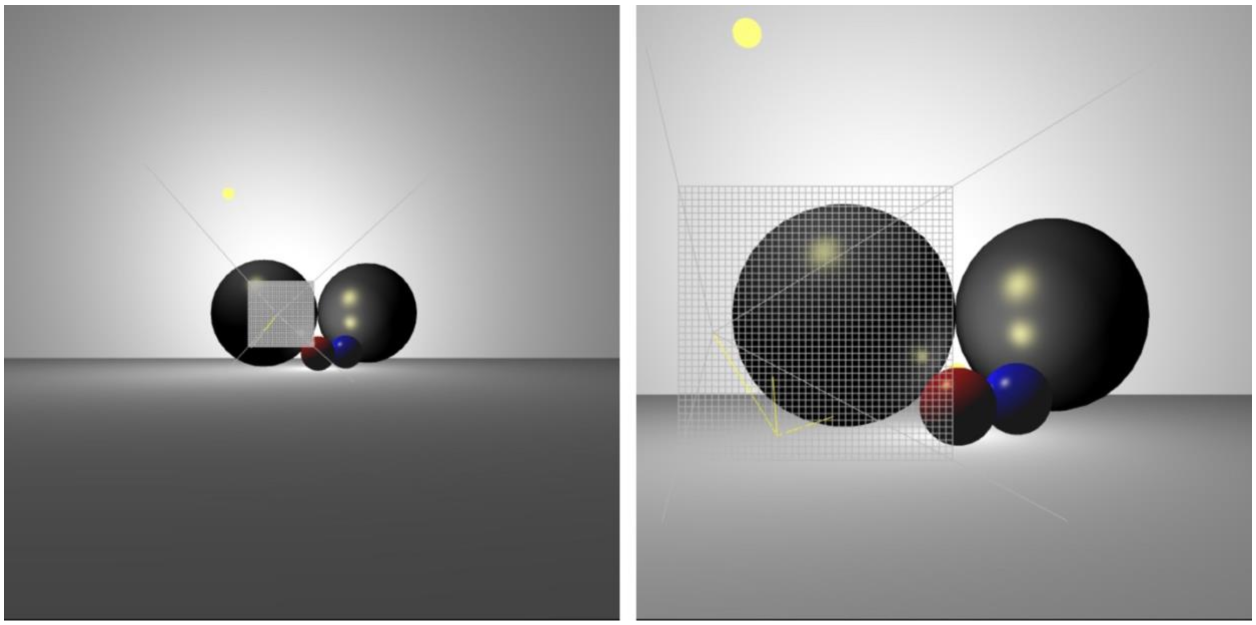


Figure 3.4: Zoom with Scroll Wheel/Trackpad Movement, Zoomed Out (Left), Zoomed In (Right)

As shown in Figure 3.4 above, the mouse's scroll wheel can be used to zoom the view in and out. This can be achieved with a computer trackpad as well by using two fingers at the same time and either pinching or sliding up and down. This will allow the user to get closer to the scene to view details, or back out to see the whole picture.

3.1.4 Space Bar Functionality

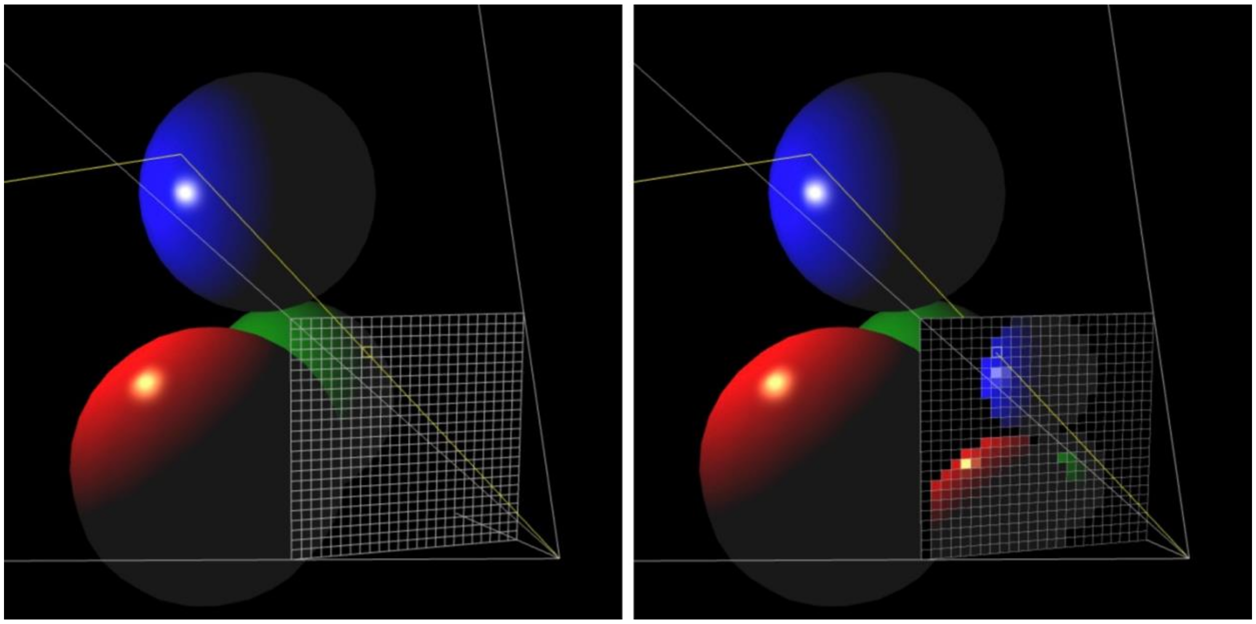


Figure 3.5: Space Bar to Render, Image Plane Empty (Left), Image Plane Filled In (Right)

Figure 3.5 demonstrates the rendering function which the spacebar activates. When the user hits the spacebar, the program will render the scene using ray tracing and display the corresponding image on the view frustum. This should aid the user in seeing how each ray shoots from the camera through the center of the pixel out and into the scene. This will help visualize why a pixel is a certain color.

3.1.5 All Rays View - The A Key Toggle Functionality

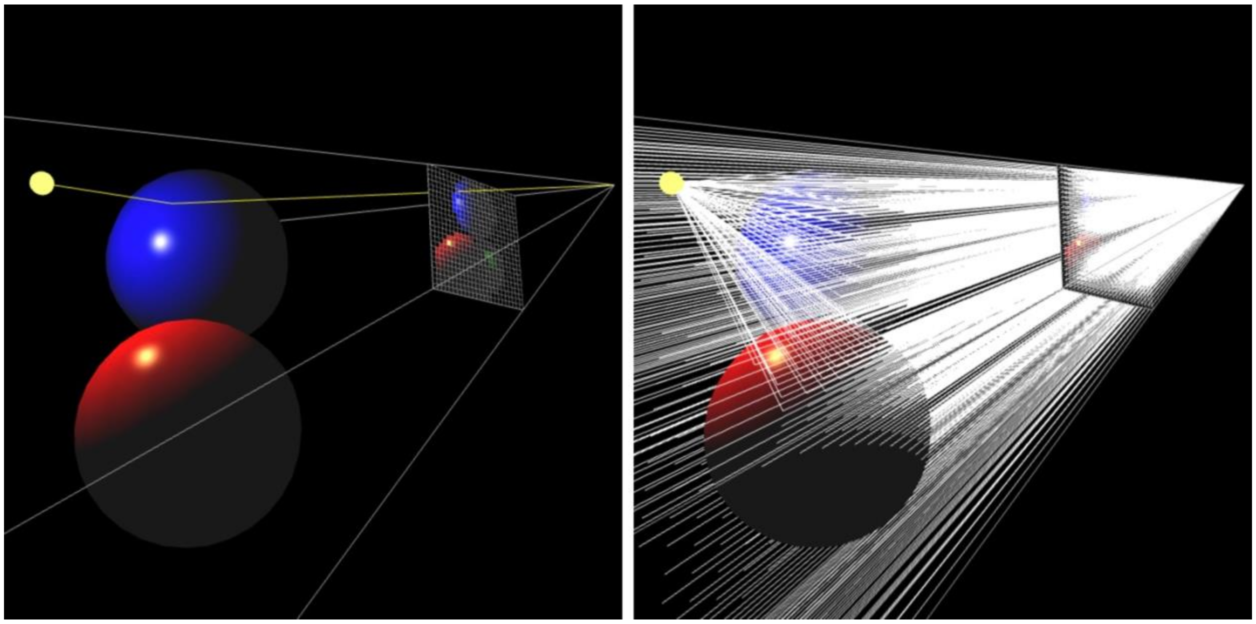


Figure 3.6: All Rays Toggled Off (Left), All Rays Toggled On (Right)

Figure 3.6 demonstrates the ‘a’ toggle key functionality. Tapping the ‘a’ key, toggles the display of all rays in the scene on and off. This can help demonstrate that the entire image is ray traced for all pixels. It also allows the user to start looking at individual rays as that ray traverses through the scene.

3.1.6 Horizontal (h) and Vertical (v) Views

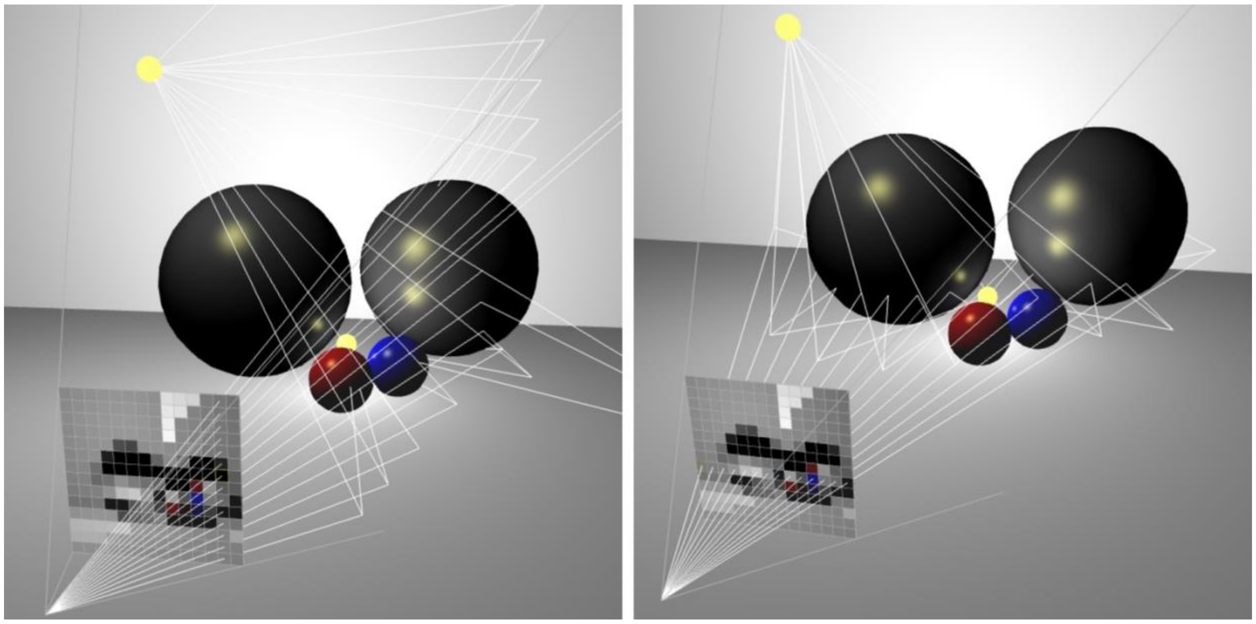


Figure 3.7: All Rays in the Column displayed (Left), All Rays in the Row Displayed (Right)

Another visual aid included in the program is the display of all rays in a column or row. This feature allows the user to see the pattern of how rays traverse through a horizontal or vertical plane of the scene and is easier to understand than the view of all (‘a’ toggle) rays in a scene, which can be too much information. Viewing a slice of the scene limits the amount of visual information and will help the user better understand the ray tracing concepts. This is demonstrated in Figure 3.7 above. Displaying the horizontal row of rays in this scene is very useful because you can see how the rays reflect off the spheres and onto the ground in the right side of Figure 3.7. This feature can be particularly helpful in understanding reflections and refractions.

3.1.7 Arrow Key Functionality

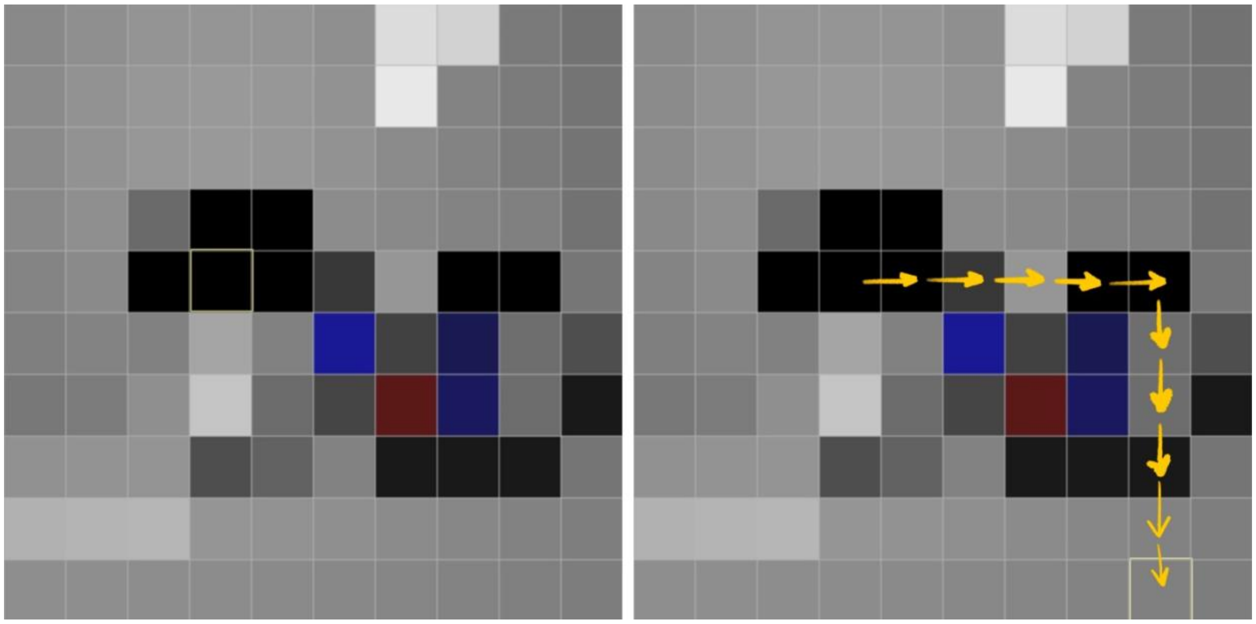


Figure: 3.8: Arrow Key Movement Driving the Cursor on the View Frustum, Cursor Starting Position (Left), Cursor Movement (Right)

The user can use the arrow keys to control the position of the cursor on the view frustum. In the right side of Figure 3.8 above, the right arrow key being struck five times and the down arrow key being struck five times moving the cursor to the right and then down. This feature allows the user to decide which ray they want to view as it moves through the scene. This feature is helpful in allowing the user to better understand how a single ray interacts with the scene. All four arrow keys are implemented and whenever the user reaches the border of the view frustum and attempts to move past the border, the highlighted position will overflow to the other side of the view frustum, and the cursor will not appear to be stuck.

3.1.8 The Reset Key Functionality

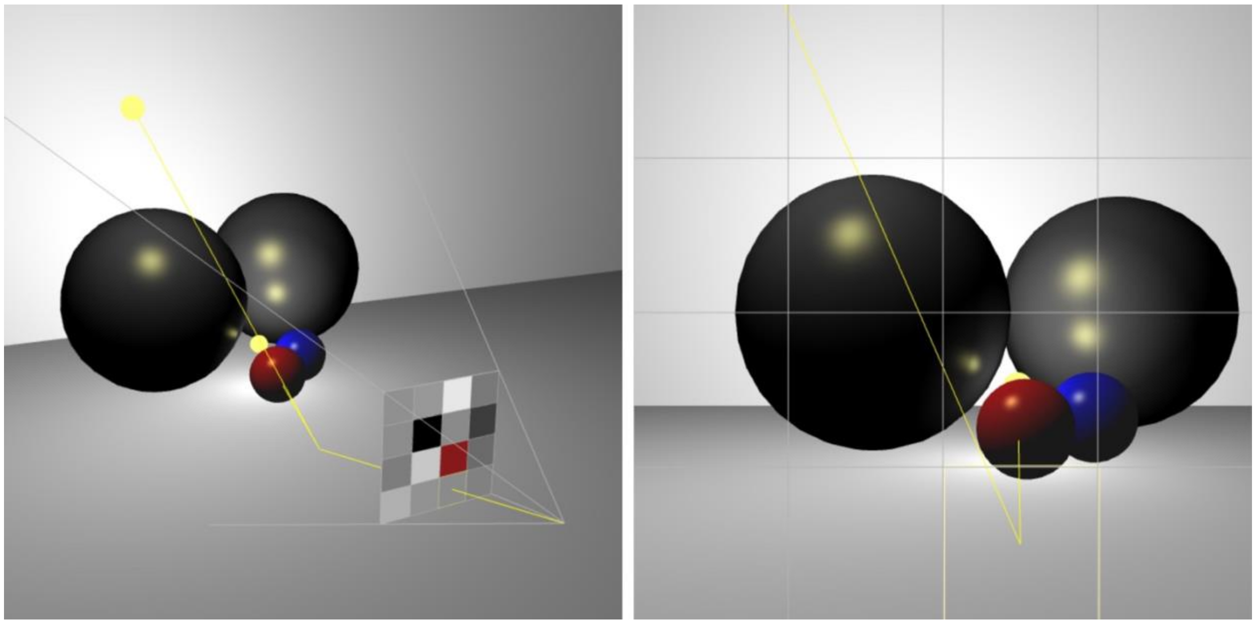


Figure 3.9: Reset Function (R Key), Camera View Away Origin Before Pressing R (Left), View Directly Behind Frustum After Pressing R (Right)

Figure 3.9 demonstrates what happens when the user hits the reset ‘r’ key. The reset function which will move the real time view camera to reposition itself in the same spot as the ray tracing view camera. The current position of the real time view camera is irrelevant when the ‘r’ key is struck, as the position and rotation of the camera will be reset. This feature is useful when you have adjusted the camera so far that it is hard to see and move around the scene. Simply reset the camera view and start over.

3.1.9 The Plus (+) and Minus (-) Key Functionality

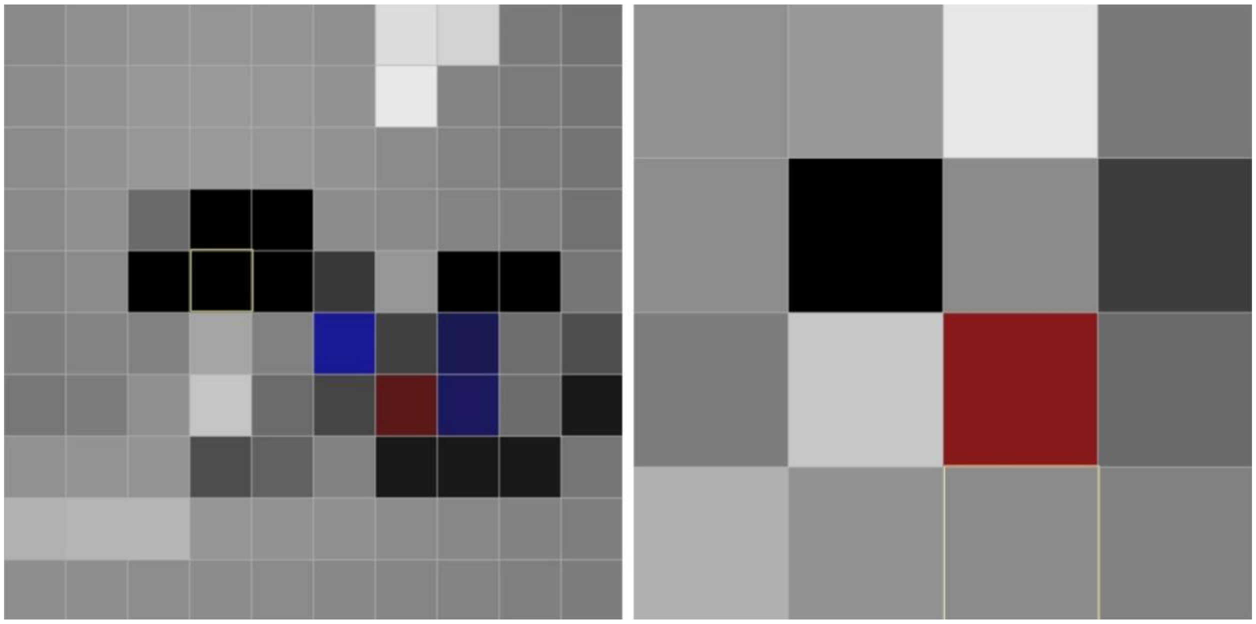


Figure 3.10: Increasing the view frustum resolution (Left), Decreasing the view frustum resolution (Right)

Sometimes the view frustum is not showing the specific ray you would like to see in the scene, or the view frustum resolution is not high enough or low enough for what you want to see. Pressing the plus (+) or minus (-) keys allows the user to increase or decrease the view frustum resolution, as shown in Figure 3.10 above. Striking the plus (+) key will allow the user to increase the resolution as seen on the left side of Figure 3.10. Increasing resolution results in more visible squares in the image. Hitting the minus (-) key will decrease the resolution of the frustum, as seen on the right side of Figure 3.10. Decreasing the resolution results in fewer visible squares in the image.

3.1.10 Usability Improvements

Using the mouse's scroll wheel will normally move the web page up and down. Using the spacebar will force the web page to move to the bottom. This would normally happen when trying to use the real time view, but I have negated the standard operation of these two input methods whenever the mouse is hovered over the real time view square to improve the user experience.

3.2 Ray Tracing Effects that can be Demonstrated

The number keys are used to switch scenes. These scenes were designed to demonstrate different features of ray tracing. The different scenes allow the user to view different aspects of ray tracing to help them better understand ray tracing concepts.

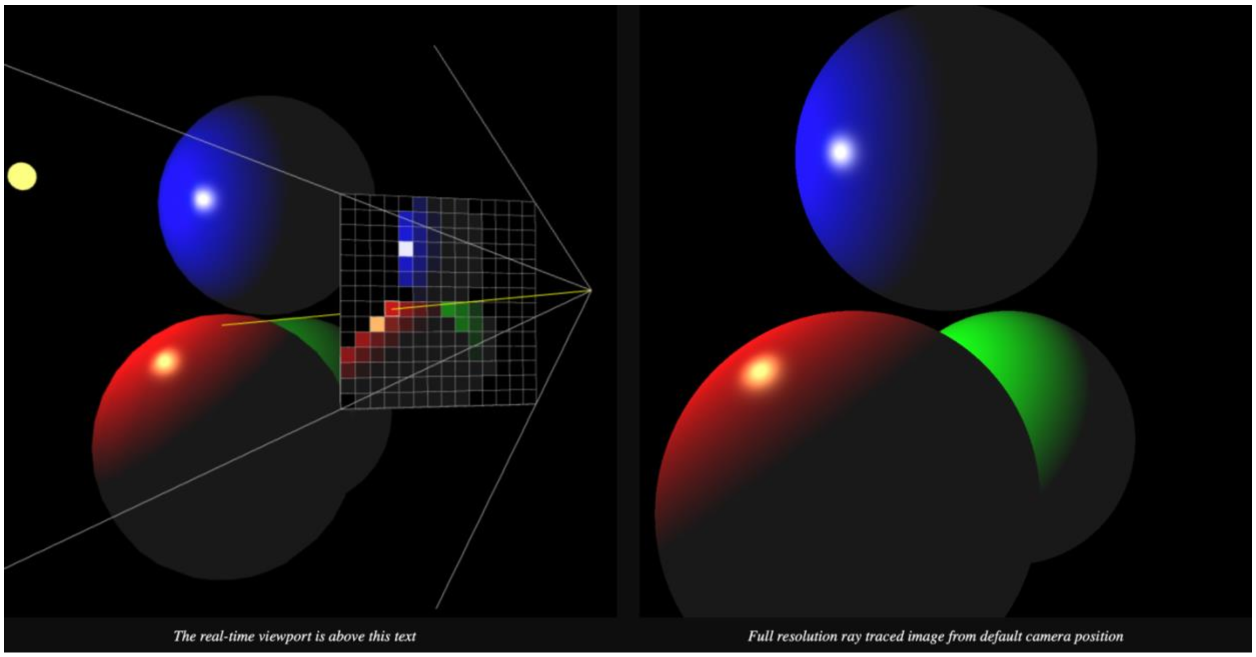

The real-time viewport is above this text
Full resolution ray traced image from default camera position

Figure 3.11: Top of Web Page Showing Scene 1, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scene 1 as shown in Figure 3.11 is designed to help the user understand how a ray traverses through a scene and gets its color. This scene will demonstrate only the color based off the shader of the objects in the scene. No secondary rays are shot from the point of intersection towards the light checking for shadows. This scene is helpful for understanding that a ray traverses through the scene for a specific pixel until it intersects with an object. That object determines the color of that pixel. This is repeated for all pixels of the image. Scene 1 also demonstrates ray intersection with spheres.

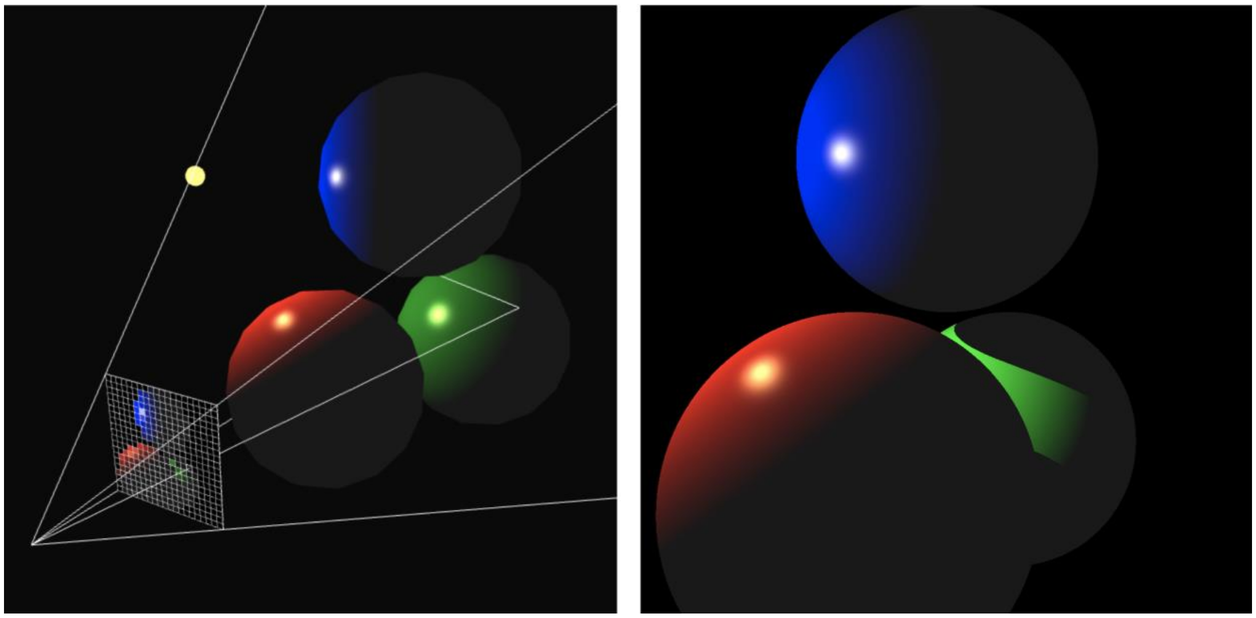


Figure 3.12: Scene 2, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scene 2, shown in Figure 3.12 above, includes the check for shadows. This is achieved by sending another ray from the closest intersection towards the light source. If the ray intersects another object on its way to the light source, that point is in shadow and is not receiving any light from the light source. This can best be seen on the green sphere in Figure 3.12 above. The top of the green sphere is in shadow because the blue sphere is blocking the light source.

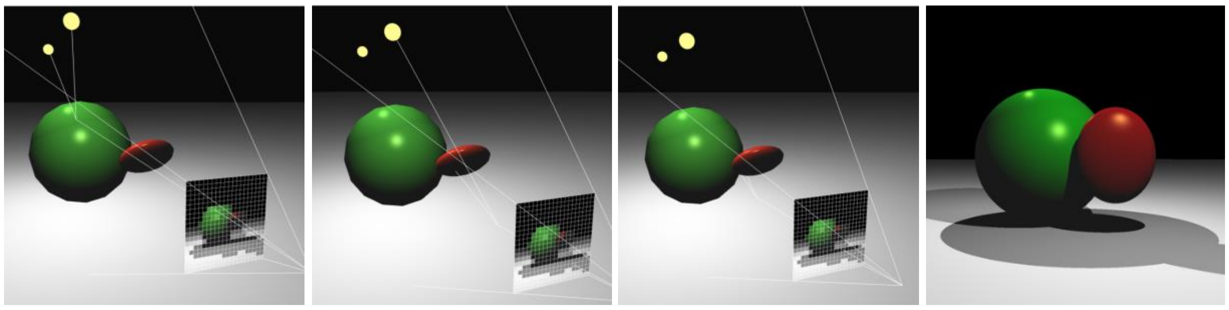


Figure 3.13: Scene 3, Light on Object From Two Lights(Far Left), One Light Shadow (Second Left), Complete Shadow (Second Right), Full Resolution Ray Traced Image (Far Right)

Scene 3, shown in Figure 3.13 above, contains multiple light sources and demonstrates several facets of ray tracing. With multiple lights in this scene, several shadow interactions are possible. The leftmost square shows what happens whenever two lights hit a point on a sphere.

The second image from the left illustrates what happens whenever a point is able to intersect with only one light as the other light is being blocked by the red ellipsoid. In this scenario only half the light in the scene can reach the shadow areas, so the shadow receives a half value. The area is in shadow for one light and is lit by the other light. The second from the right image shows what happens when neither light can be hit from the point of intersection. In this scenario, a very dark shadow appears as neither light is able to illuminate that part of the scene. The right most image shows a full resolution image of the scene. This scene also shows off ellipsoid objects and a plane object.

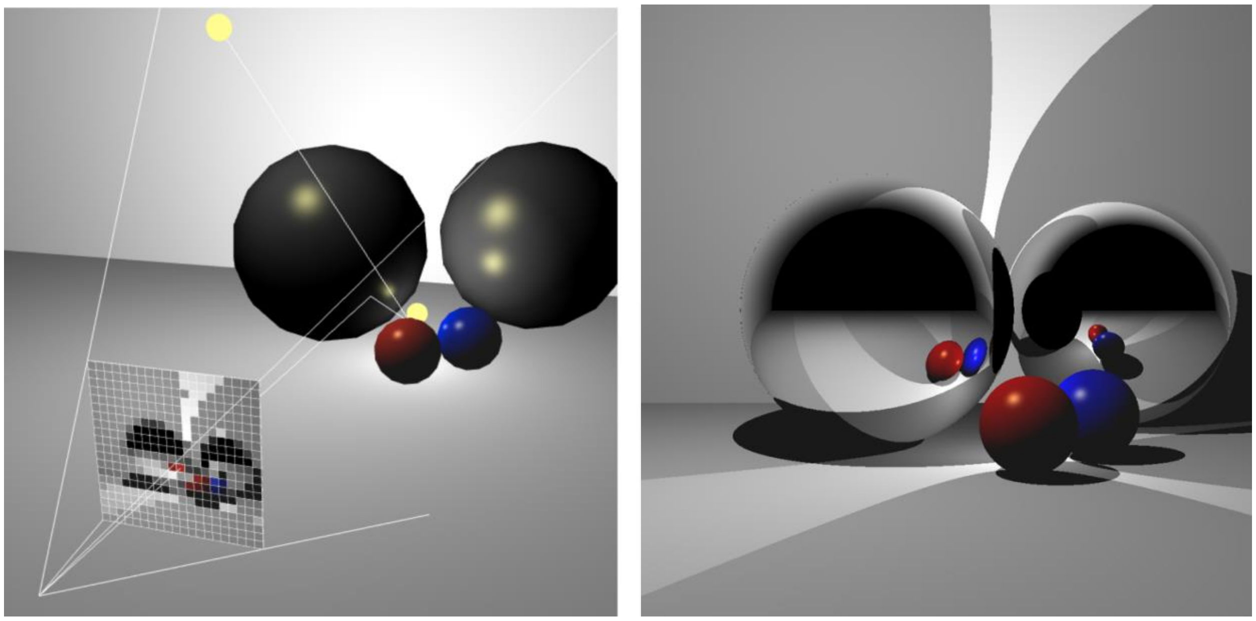


Figure 3.14: Scene 4, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scene 4, as shown in Figure 3.14 above, demonstrates reflections. In the scene you can see the scene reflected in the mirror spheres. This scene helps demonstrate how a ray traverses through the scene and intersects with a mirror object that bounces that ray out, back into the world recursively. In this scene there is only one recursive bounce of the reflection. So inside the reflection on one sphere you are not able to see the reflection of another mirror surface. This leads to the other sphere being black in the reflection of the mirror spheres. Program users will gain a better understanding by being able to see the rays traverse the scene, or horizontally cut the scene, so they can see how a row of rays bounces off the reflective surface. The parts of the reflection on the spheres in scene 4 that are pure black are due to the reflected ray not intersecting with another part of the scene. There is no wall, surface or object for them to reflect off so they are displaying no light and no color.

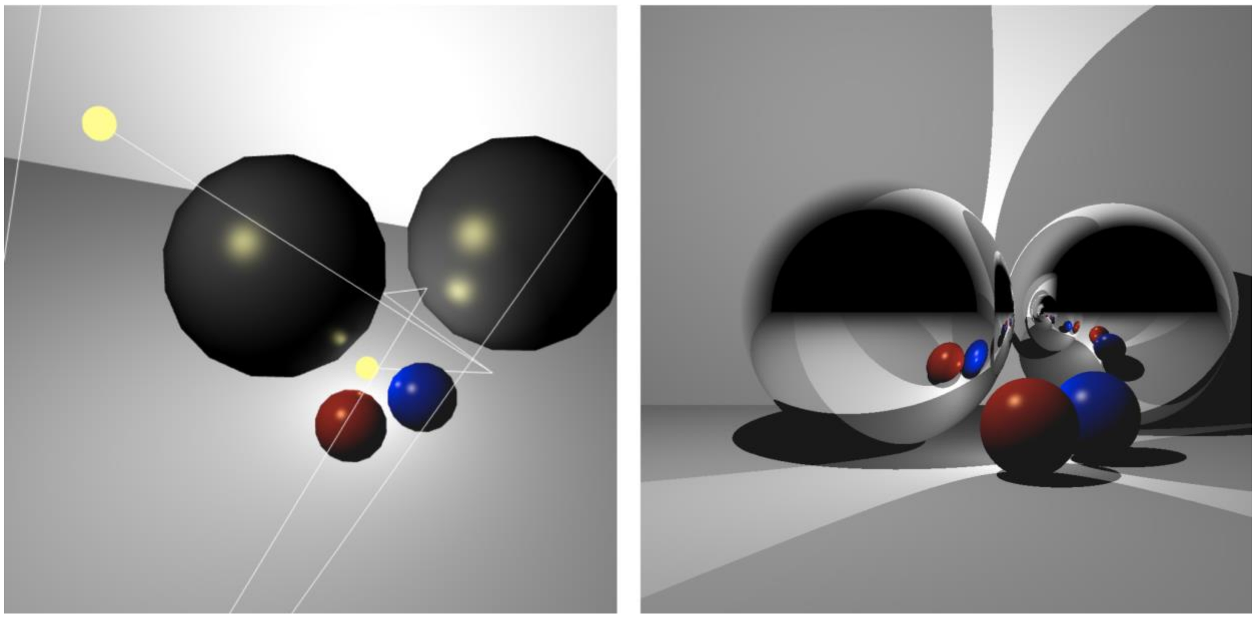


Figure 3.15: Scene 5, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scene 5, shown in Figure 3.15 above, demonstrates multiple recursive bounces. In this scene the limit on recursive bounces has been greatly increased allowing for reflections within reflections. In the right part of the image you can see multiple reflections cascading on top of each other in the right mirror surface. It is easy to see this in the real time view as you can see the line bouncing off of one sphere intersecting with another sphere and bouncing off of that sphere and then traversing into the rest of the scene, where it is able to perform the light check.

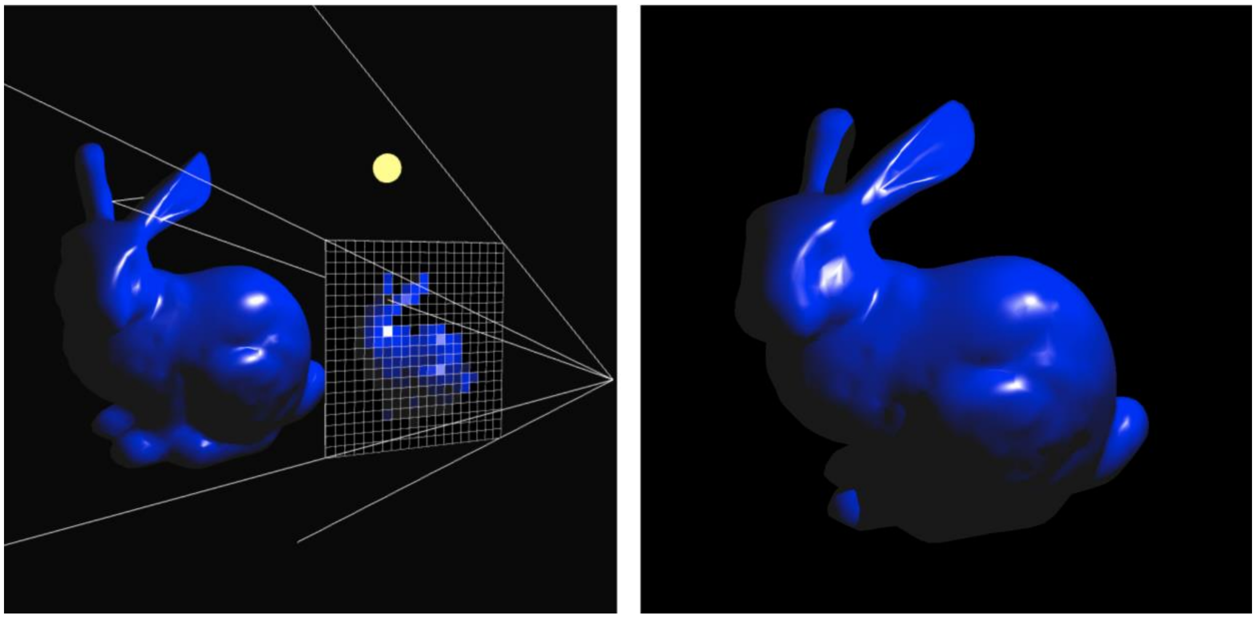


Figure 3.16: Scene 7, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scenes 6 and 7 demonstrate multiple concepts. Both display the blue bunny, but Scene 6 displays a transformed version. Scenes 6 and 7 demonstrate the performance increase of using bounding spheres. This scene shows a non-primitive object. Non primitive objects are 3D models made of thousands of triangles. Previous scenes showed basic primitive objects such as spheres, ellipsoids, planes, and cylinders. Because of the basic geometric shape of these primitive objects, a simpler hollistic intersection equation can be used which is easier for students to initially program. Bounding spheres are a technique to improve computational performance. In scene 6, the bunny only takes up a small portion of the screen allowing for the bounding spheres to result in a significant performance improvement. Using the bounding sphere approach lets the program first check for intersections with a bounding sphere before executing the more computationally intensive code to evaluate for intersection with the thousands of triangles that comprise the non-primitive bunny in scene 6 and 7. In scene 7, where the bunny object consumes a greater portion of the screen, the program will take longer to render the frame because more pixels in the frame contain the bunny. Inside of scene 7, as shown in Figure 3.16 above, shadows are cast from the bunny object onto itself. This concept is best demonstrated at the left side of the left ear of the bunny being in shadow from the ear, down the neck and to the underside of the bunny. They are in shadow as the light is being blocked by the bunny's body.

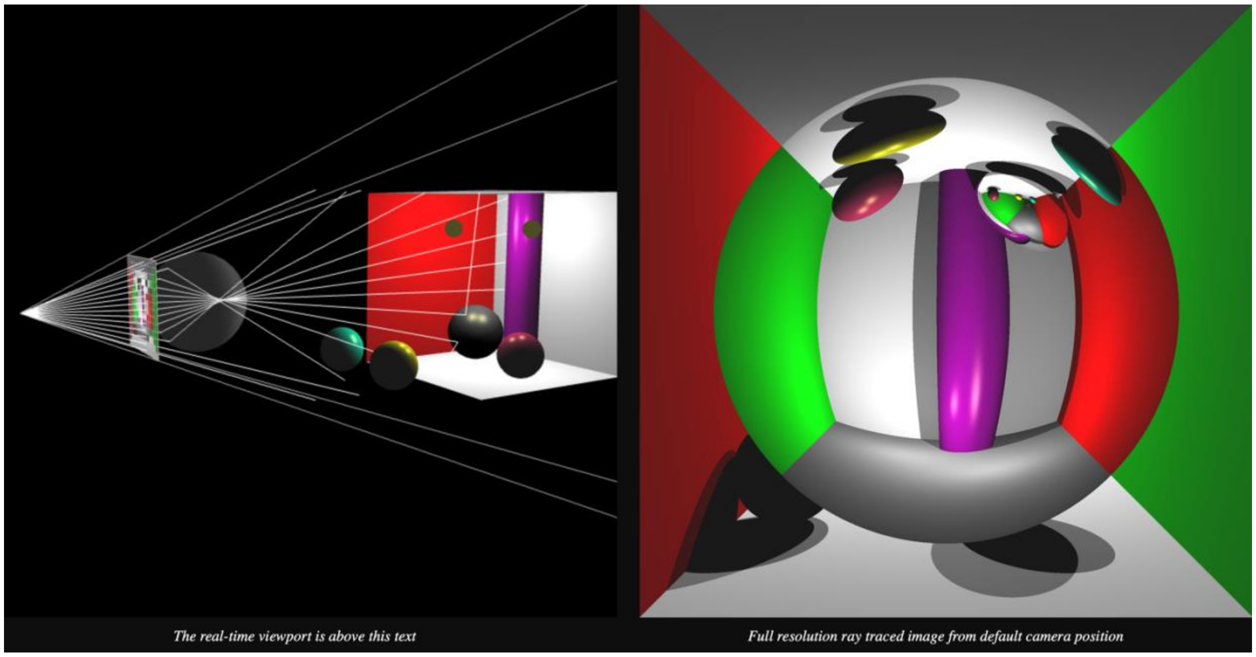

The real-time viewport is above this text
Full resolution ray traced image from default camera position

Figure 3.17: Scene 8, Real Time Viewport (Left), Full Resolution Ray Traced Image (Right)

Scene 8, as shown in Figure 3.17 above, demonstrates refractions of all of the primitive shapes used in the rest of the program. It is instructive to be able to visualize how the ray enters the refractive object. Then how the ray's path is bent, based on the shape of the object. Then the ray exits the refractive object and the ray's path continues into the rest of the world, resulting in the sphere being refractive. This additional effect is added on top of normal Blinn-Phong objects and reflective objects.

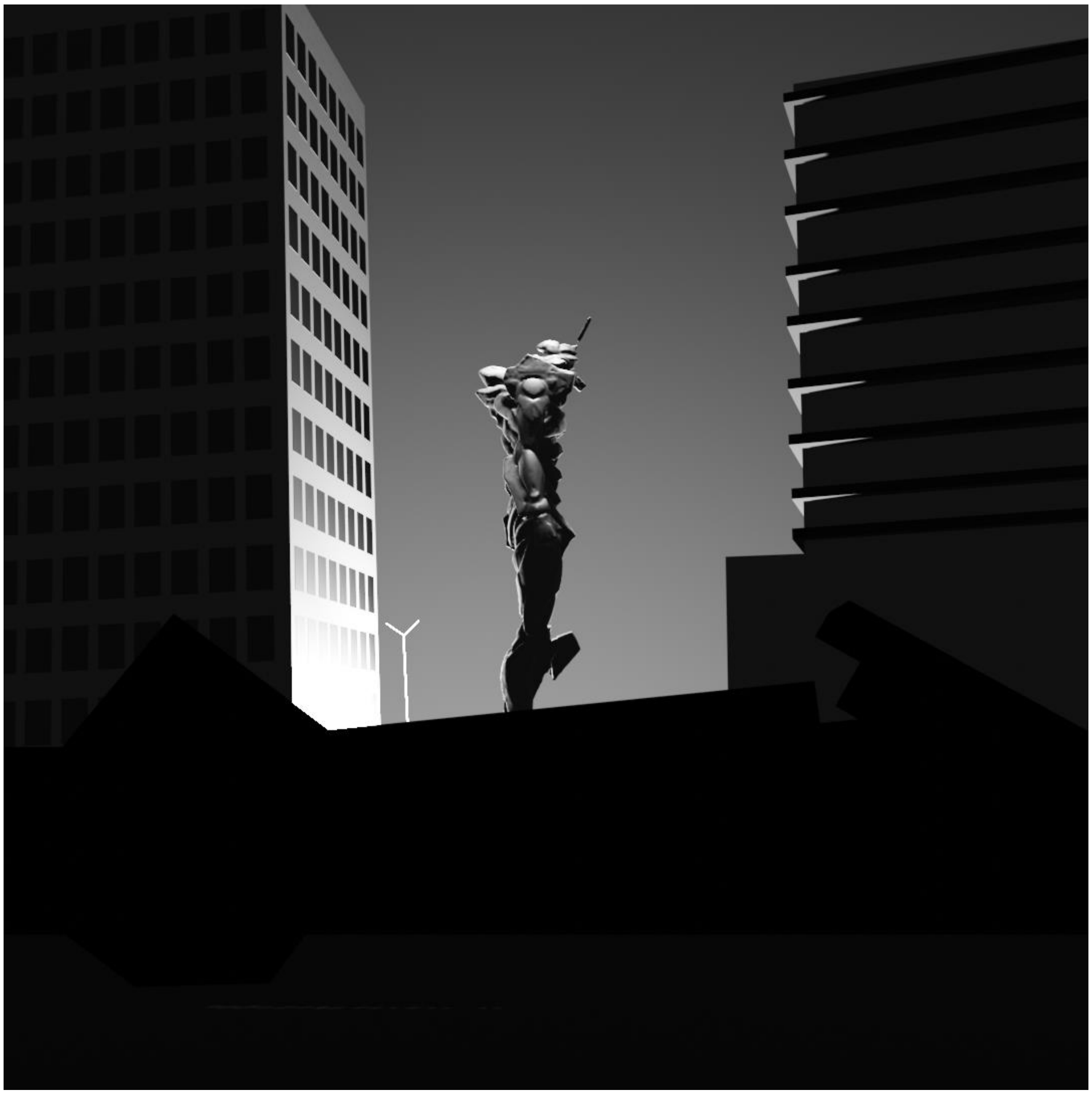


Figure 3.18: Scene 9

Scene 9, shown in Figure 3.18 above, demonstrates a more artistic scene. This scene is not necessarily for educational purposes, but for the user to have something to look at which is not a primitive shape and carries more of an artistic interest.

4. CONCLUSION

This tool can be used for future computer graphics classes and for students wanting to learn about ray tracing. The project involved learning new concepts, including a new programming language (JavaScript) and WebGL libraries. A demonstration video for the URS symposium is also included, which received high praise from both students and reviewers. The program will be immediately useful for computer graphics instructors at Texas A&M University and beyond.

4.1 Future Work

There remain a few enhancement areas which could be added in the future. As an example, techniques for real time rendering which could make the viewport look closer to the ray traced image would be a nice improvement to help users understand what is happening in the viewport scene. These techniques could include shadow maps, reflection maps and real time refraction, potentially through vertex and fragment shaders. To enhance the performance of the ray tracer, something like a Bounding Volume Hierarchy (BVH) structure could be implemented. Adding path tracing, another method of rendering, is another potential enhancement to the program. Path tracing is similar to ray tracing except at the first point of intersection in the ray's path multiple rays are shot off in random directions to gather information about the scene. Path tracing algorithms would produce even more realistic effects. In addition to this, Physically Based Rendering (PBR) and the Fresnel effect, which considers the ray's angle in the amount of reflection, could be programmed to produce more realistic results.

With the improvements in performance from techniques, such as BVH, combined it may be possible to have the full resolution ray traced image be calculated in real time and projected onto a texture. This could be displayed in another view to the right of the real time view, making it more interactive and allowing for the retracing views camera to be changed. This change to the program could also allow for tools to be implemented which would allow the user to change the position, scale, and rotation of objects in the scene and enable the user to move the lighting. These enhancements are not necessary to understand ray tracing but would allow the user to experiment more with this educational tool. This would bring it closer to a 3D editing program. A final enhancement to consider is for the user to import their own 3D files.

This educational tool was informally evaluated by some of the Texas A&M computer graphics professors. They provided feature requests and very positive feedback about the program and its usability for their classroom and student use. One professor provided feedback that this Ray Tracing Teaching tool is excellent and will be a very useful teaching aid. Another professor provided feedback that he would really appreciate an interactive tool, like this one, but enhanced with additional graphics concepts. Another professor would like to add a feature that would allow the user to customize scenes. Another feature they would like is to list the 3D coordinates of intersection points of the ray currently selected by the user.

REFERENCES

[1] Haines, E., & Akenine-M, T. (Eds.). (2019). Ray Tracing Gems Series. Ray tracing Gems series. [Cited September 11, 2021] Available: http://www.realtimerendering.com/raytracinggems/

[2] Balreira, D. G., Walter, M., & Fellner, D. W. (2017). What we are teaching in introduction to computer graphics. [Cited September 10, 2021] Available: https://diglib.eg.org/bitstream/handle/10.2312/eged20171019/001-007.pdf

[3] Reina, G., Müller, T., & Ertl, T. (2014, July). Incorporating modern OpenGL into computer Graphics Education. IEEE Xplore. [Cited September 11, 2021] Available: https://ieeexplore.ieee.org/document/6855305

[4] Vitsas, N., Gkaravelis, A., Vasilakis, A., Vardis, K., & Papaioannou, G. (2020). Rayground: An online educational tool for ray tracing. GitHub. [Cited September 11, 2022] Available: http://graphics.cs.aueb.gr/graphics/docs/papers/Rayground___EG_2020_Educational.pdf

[5] "Rendering | Pixar In A Box | Rendering," *Khan Academy*, 27-Aug-2015. [Online]. [Cited March 06, 2022] Available: https://www.khanacademy.org/computing/pixar/rendering

[6] S. Marschner, *Fundamentals of Computer Graphics*, 5th ed. CRC Press, 2021.