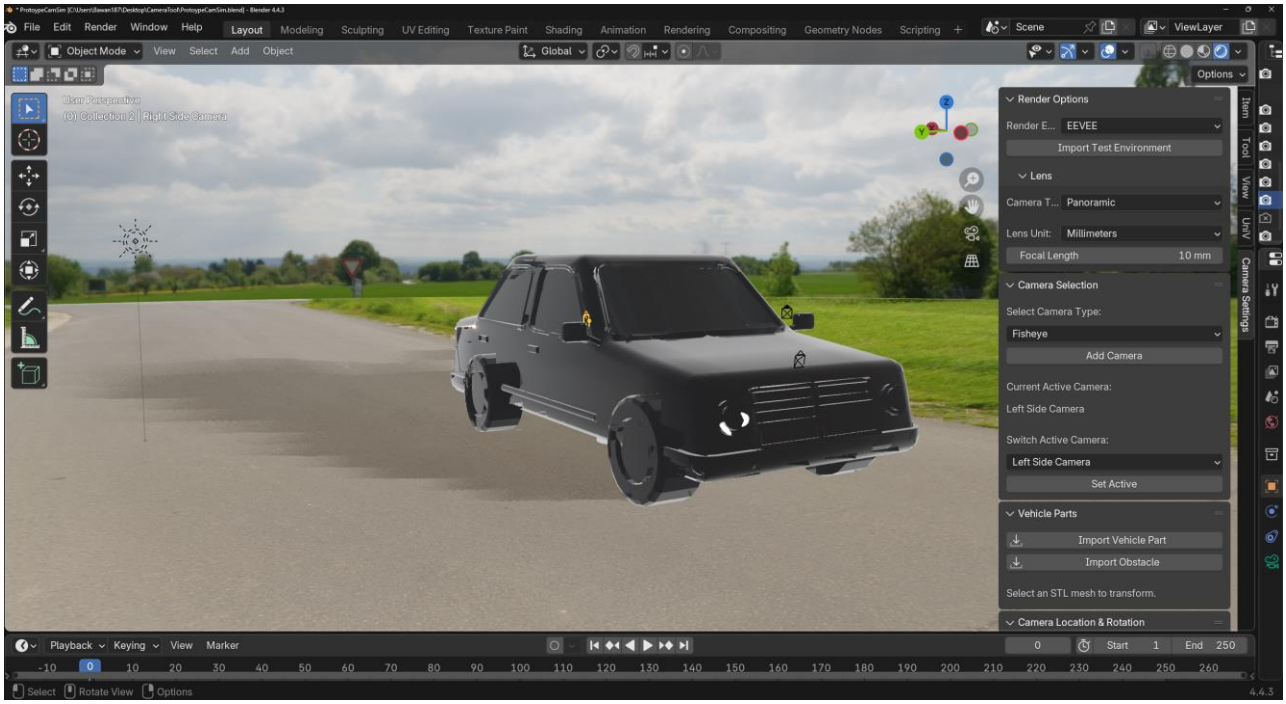
CHALMERS



Camera Simulation Tool for Automotive Applications

Using Blender in Junction with Python Scripts

Bachelor´s thesis in Electrical Engineering & Mechatronics Engineering

AMIR SMAJIC
BAWAN KAMAL MOHAMMAD

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

CONTENT PAGES

Innehållsförteckning

1 INTRODUCTION 1

1.1 BACKGROUND 1
1.2 PURPOSE 1
1.3 LIMITATIONS 2
1.4 CLARIFICATION OF RESEARCH QUESTION 2

2 THEORY 3

2.1 THE HISTORY OF BLENDER 3
2.2 NAVIGATING THE SOFTWARE 3
2.3 CAMERAS IN BLENDER 7
2.3.1 Camera types 7
2.3.2 Camera Parameters 9
2.3.3 The Python API and Blender 19
2.3.4 Add-on Development 23
2.4 ENVIRONMENT 25
2.4.1 Nodes 25
2.4.1 Object Types 26
2.4.2 CATIA 26

3 METHOD 27

3.1 CAD- PARTS 27
3.2 CREATING THE ENVIRONMENT 27
3.3 LEARNING BLENDER 28
3.4 REVIEWING PYTHON 29
3.5 IDENTIFICATION OF FUNCTIONAL REQUIREMENTS (ÄNDRA KANSKE) 29

4 RESULTS 36

5 CONCLUSION 38

5.1 SPECIFIC PROBLEMS 38
5.1.1 Further Developments 38
5.1.2 Timeline 38

6 REFERENCES 39

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

BACHELOR'S THESIS IN ELECTRICAL AND MECHATRONICS ENGINEERING

Camera Simulation Tool for Automotive Applications

Using Blender in Junction with Python Scripts

AMIR SMAJIC
BAWAN KAMAL MOHAMMAD

CHALMERS

Camera Simulation Tool for Automotive Applications

Using Blender in Junction with Python Scripts

AMIR SMAJIC
BAWAN KAMAL MOHAMMAD

Supervisor: Armin Todorovac, Polestar Performance AB
Examiner: Bertil Thomas, Electrical Engineering

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Acknowledgements

We, as two students within electrical and mechatronics engineering contacted Armin Todorovac at Polestar to discuss the possibility of writing our degree project. Within some time, this became a reality, and it was time for us to apply our knowledge and expertise from our time at Chalmers University of Technology. We would like to recognize our supervisor, Armin, for the consistent feedback regarding the requirements and expectations of the tool that we have developed. Another thank you goes out to the rest of the Polestar team for allowing us to work in their facilities and providing us with the tools to be able to execute this accordingly. We would like to thank our examiner, Bertil Thomas, for his involvement guiding us through the writing process. Finally, we would like to thank Josip Vukusic for his guidance through the application process and for his forbearance.

Amir Smajic and Bawan Kamal Mohammad, May 2025

Abstract

As technology in the automotive industry consistency evolves, and driver aids get more advanced the need for advanced driver assistance systems (ADAS) grows larger. The need for quicker and more reliable simulation tools becomes essential to further evolve ADAS. This thesis presents the design and development of a simulation tool for 3D camera position testing. The tool created using blender and blender's own Application Programming Interface (API) and blenders own node functions to create a realistic environment to improve performance in the sense that it produces a reduced gap in between simulation and actual performance. The simulation tool was created in collaboration with Polestar to create an improved simulation tool to an already existing version with the limitation of the previous tool being setup-time and a steep learning curve. By creating a personalized interface with real-time visualization before rendering set in a realistic environment and with adjustable camera settings the simulation tool. The tool enables engineers to seamlessly integrate CAD models into the simulation tool without adjusting coordinates. The finished tool was tested by Polestar's ADAS team and found the tool to be both smoother while using the tool and a more efficient setup time which minimized time consumption

Abbreviations

CAD	Computer Aided Design
API	Application Programming Interface
STL	Standard Tessellation Language
ADAS	Advanced Driver Assistance Systems
EV	Electric Vehicle
POV	Point of View
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
DOF	Depth of Field
BSDF	Bidirectional Scattering Distribution Function
OOP	Object-Oriented Programming
UI	User Interface

1 Introduction

The automotive industry is constantly undergoing changes. Nowadays, many high-end vehicle companies use ADAS technology. These systems gather important information about the surroundings of the vehicle using sensors and cameras which are mounted on or inside the vehicle.

1.1 Background

Polestar, owned by Geely Holding Group, is a company that produces EV's and is based in Gothenburg, Sweden. Many of their commercial vehicles use cameras and sensors, which need rigorous testing to ensure that they meet safety standards. This project aims at creating a tool that can be used by Polestar engineers to test and verify camera functionality by simulating a three-dimensional environment that allows for camera simulation.

The engineering team at Polestar have been faced with issues regarding previous tools that were used to achieve accurate simulation of cameras. These issues were mainly centered around efficiency and software bugs that could not be predicted. Previous 3D-testing environments could take up to hours to render, which affected their ability to quickly test different camera setups. Moreover, the software faced bugs that usually resulted in crashes that affected progress.

1.2 Purpose

To develop a 3D – test environment which can let users simulate many different types of camera scenarios. This includes POV images of what the camera is currently showing, allowing users to change specific camera parameters to replicate different setups and enabling developers to import 3D – models of vehicles or specific vehicle parts, such as front and rear bumpers.

Another important feature of the tool is to eliminate the need for prior knowledge in programming. This is considered during the design process and aims to increase efficiency in projects. Furthermore, the tool needs to be able to quickly allow for time consuming processes such as image rendering. The previous software that was used had issues regarding speed. This tool should account for fast renders and setups.

1.3 Limitations

Initially, this project was set to start in January and end in June. Along the way, there have been some timeline issues that have forced us to begin in April. This has in turn led to a small cut in the functionality of the tool that is going to be developed. Furthermore, Polestar has prompted us to not show 3D – models of their vehicles as it goes against internal policies. Description of all the tools and functions of blender will be limited to the ones used in the project with the purpose of keeping clarity and avoiding overwhelming the reader with irrelevant information.

1.4 Clarification of Research Question

- Is it possible to build this tool in Blender?
- Does the tool improve efficiency during the test and verification stage?
- Can the tool be designed with users needing no prior knowledge in programming to get started?

2 Theory

In the following chapter, an introduction to Blender's workspace is presented. However, the focus will be mainly centered around the camera customization.

2.1 The History of Blender

Blender is a free and open-source 3D modeling and animation created by Ton Roosendaal in 2002. Roosendaal at the time, was the cofounder of NeoGeo and oversaw the in-house software at the company. The software being developed at NeoGeo would later become Blender in 2002 after financial issues that resulted in Roosedaal making Blender open-sourced [1].

2.2 Navigating the Software

After installing Blender and opening, this is what can be seen.

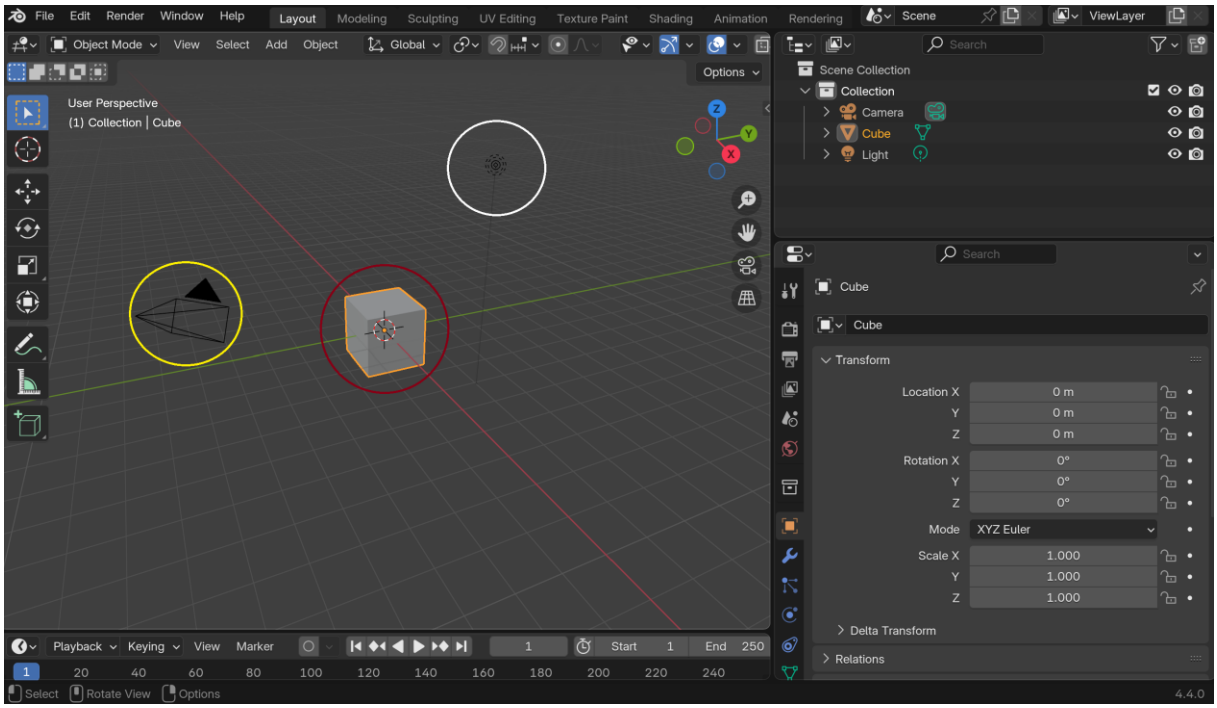


Figure 1: Blender starting screen.

Here, it can get overwhelming with the many different areas, options and buttons that the software offers. For now, the focus will be centered around the “layout” workspace. In figure 1, a large area of the screen is taken up by the 3D – viewport. Inside the 3D – viewport, three different types of objects are present.

To the left, marked in yellow, is the camera. The camera is used to render an image of the 3D – object. A render of an image is the process which turns a 3D – image into a 2D – image. Next, highlighted in red is a 3D – cube. Lastly, Blender offers lighting functionality. Highlighted in white is the point light. This object radiates light, omnidirectionally. This means that the light projected is the same in every direction. The user can view the cube in figure 1 and proceed to render the image [2].

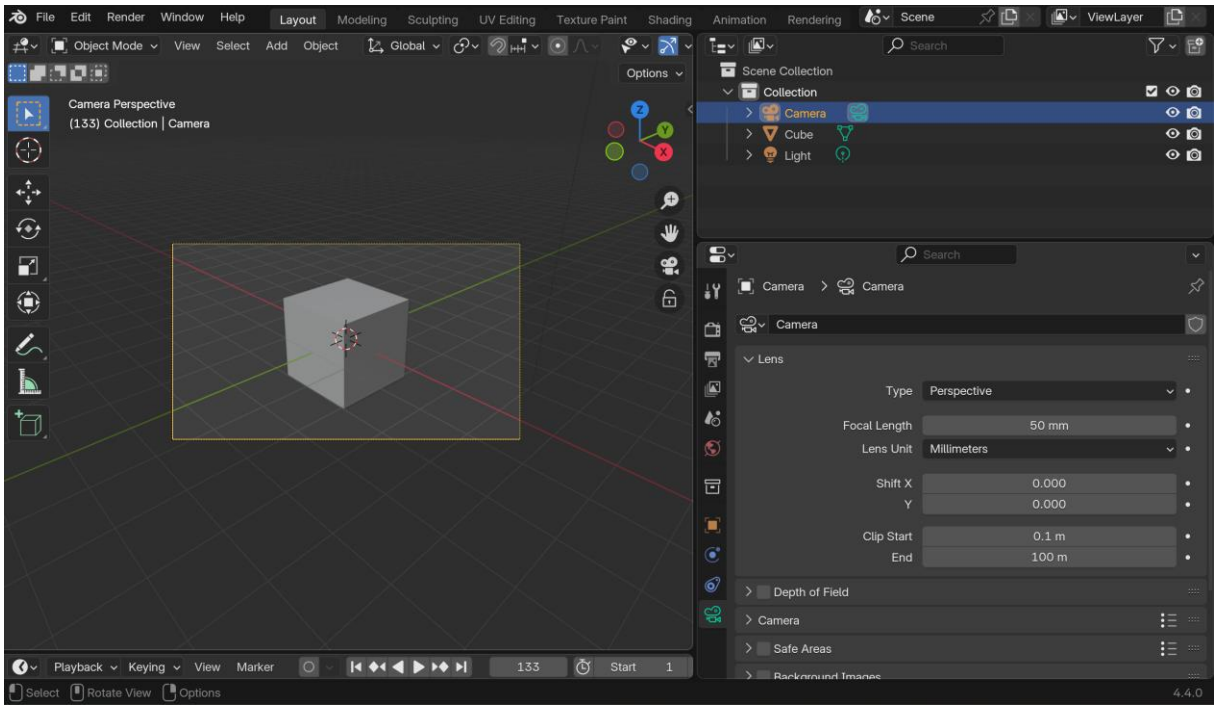


Figure 2: Camera point of view in Blender.

In figure 2, the cube is shown through the POV of the camera. The rectangular area surrounded by orange lines is what the image captures.

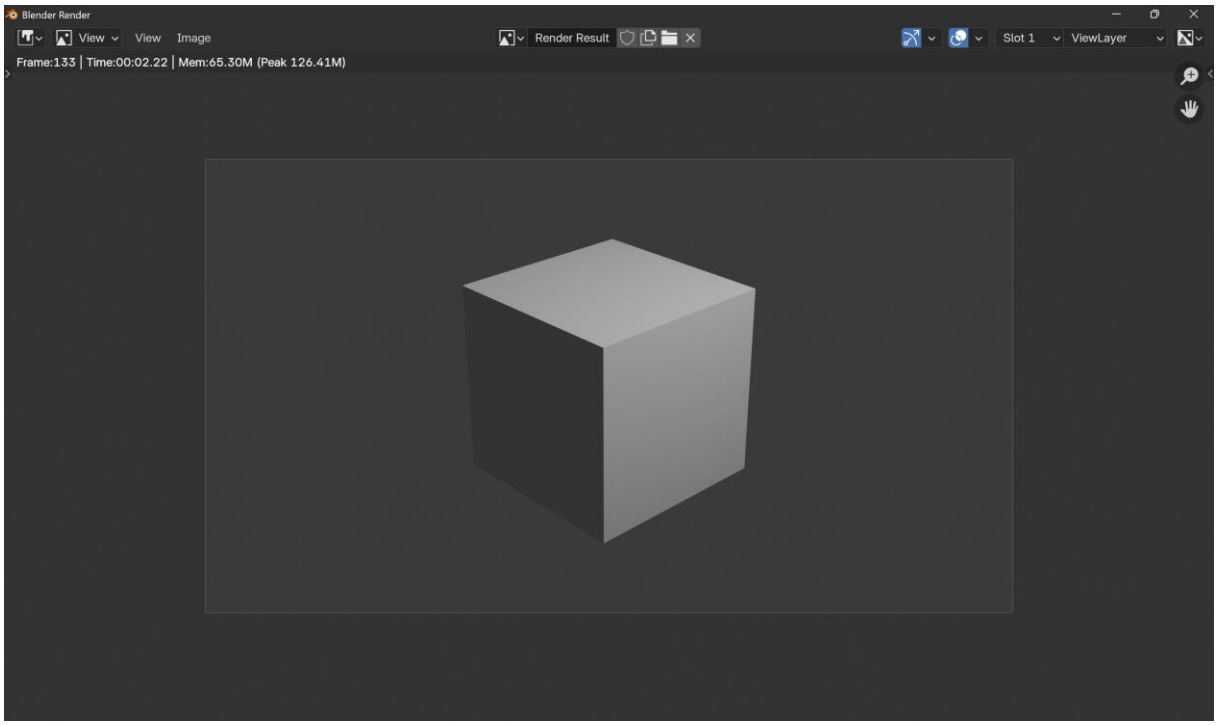


Figure 3: Rendered image of a cube.

After the render has finalized, blender has then successfully made a 3D – image into a 2D – image.

Every object that is currently in our 3D – view can move freely in the x, y and z – axis. They can also be rotated and scaled accordingly.

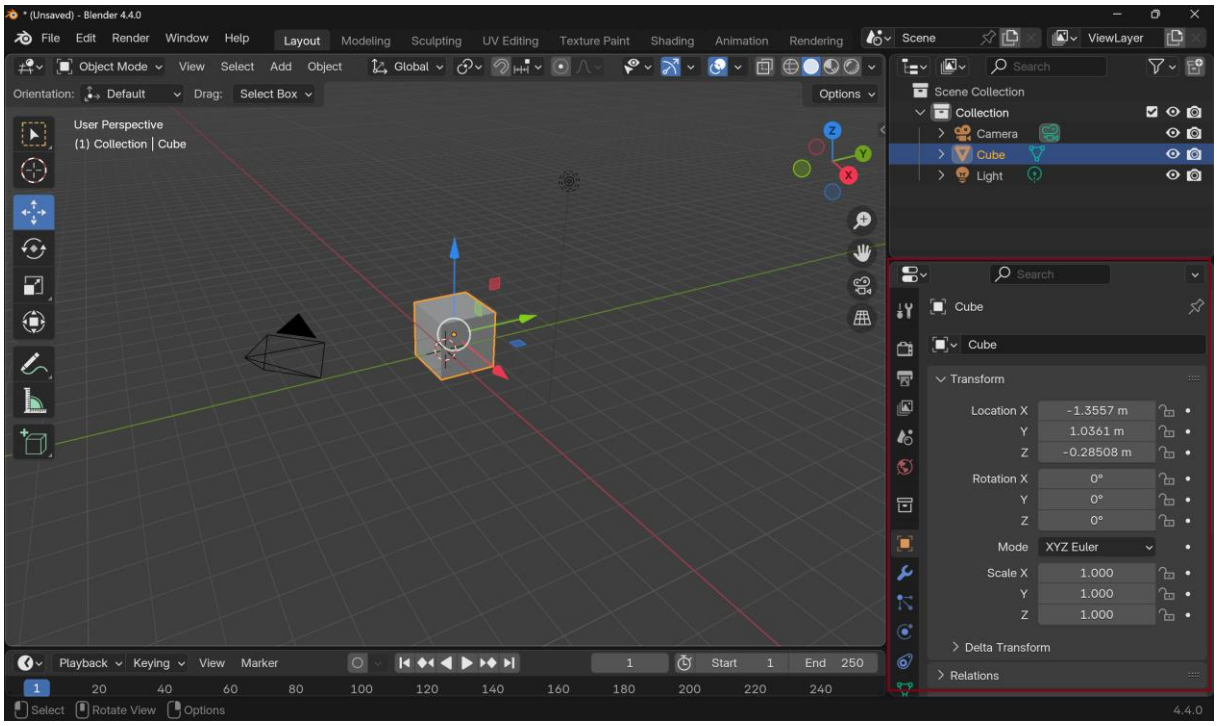


Figure 4: Object properties.

In figure 4, in the area that is highlighted in red, it shows many different settings that apply to the cube that is currently in our view. Under “Transform”, the user can change the location of the cube in either the x, y or z – axis. The same applies to the rotation and scale of the object.

2.3 Cameras in Blender

To capture an image in Blender, a camera must be present. The camera defines how much of the 3D – view that is visible.

2.3.1 Camera types

There are three main types of cameras in Blender, which each one having unique characteristics. The perspective type matches how humans view the real world. Objects appear smaller the farther away they are and larger the closer they are.

Figure 5: Perspective view of a camera.

The second type of camera is called the orthographic type. The size of an object through this view is not dependent on distance, thus it's actual size always appears. In figure 5, it can be observed that the railroad tracks appear to converge to a single point on the image. However, in an orthographic view, this is not the case.

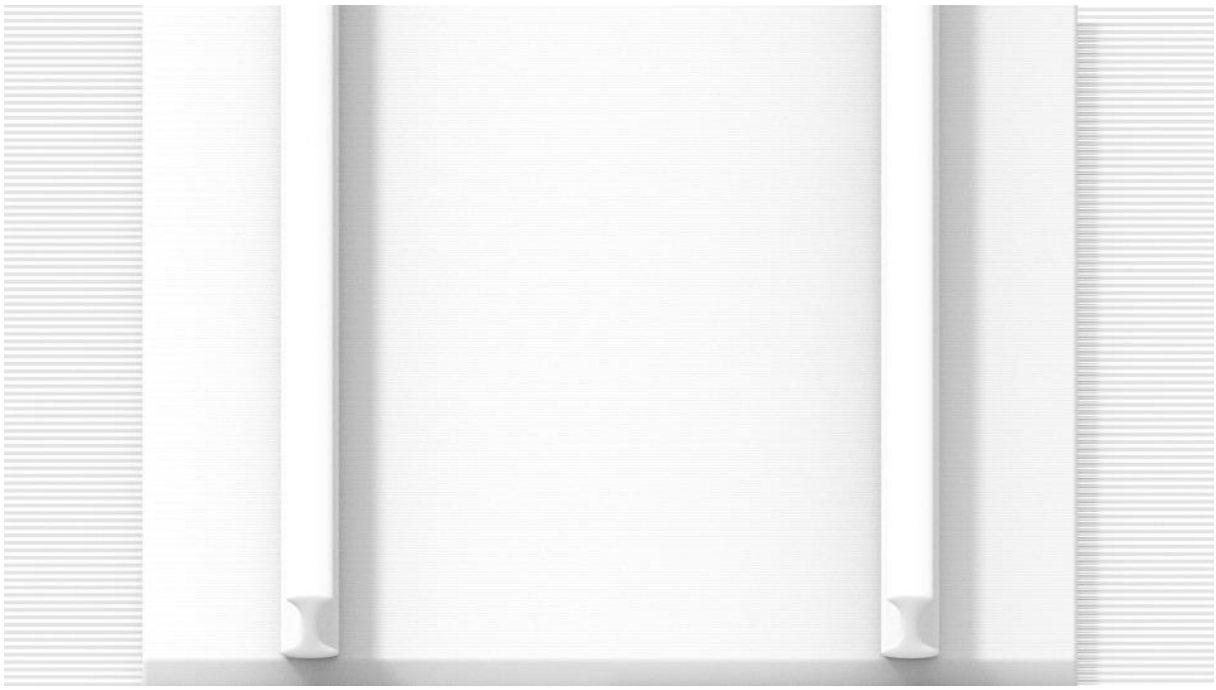


Figure 6: Orthographic view of a railroad track.

As shown in figure 6, which is the same image as depicted in figure 5, except that it is now showing an orthographic view. Here, parallel lines always appear parallel.

The last type of camera in Blender is the panoramic type. The characteristic of a panoramic type is the higher FOV associated with it. This means that it can capture more information about the environment in one single shot [2].

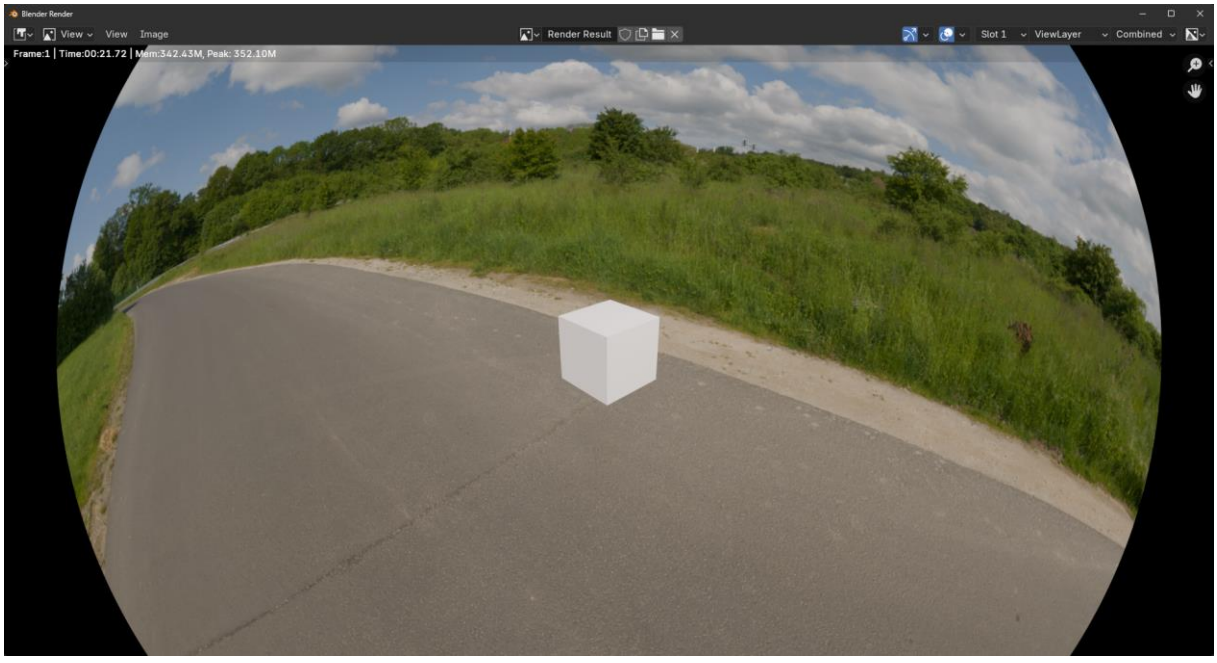


Figure 7: Panorama view of a cube on a road in Blender.

2.3.2 **Camera** Parameters

Cameras are highly customizable and can be adjusted to most applications. One important parameter is focal length. This controls the amount of zoom, which is how much of a given scene that is shown. It can be measured in terms of degrees or millimeters. A larger focal length narrows the FOV, and a lower focal length widens the FOV.

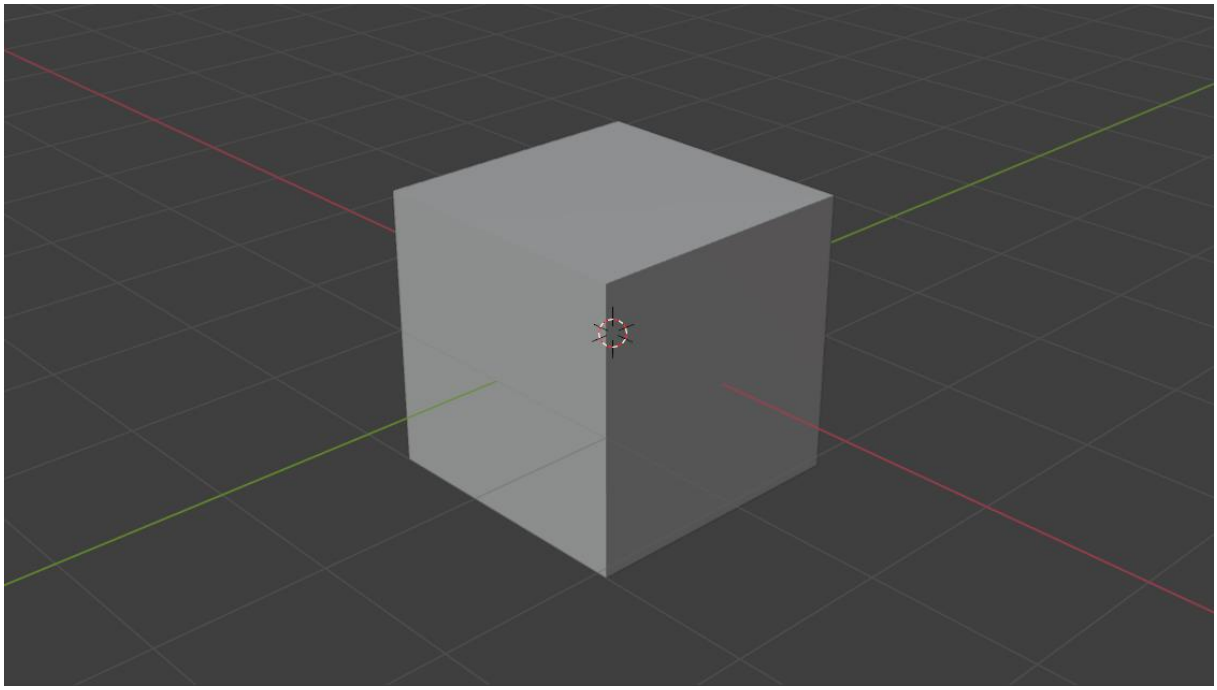


Figure 8: Camera view with a focal length of 50 mm.

In figure 8, the focal length is set to 50 mm, and the image appears to be slightly zoomed in.

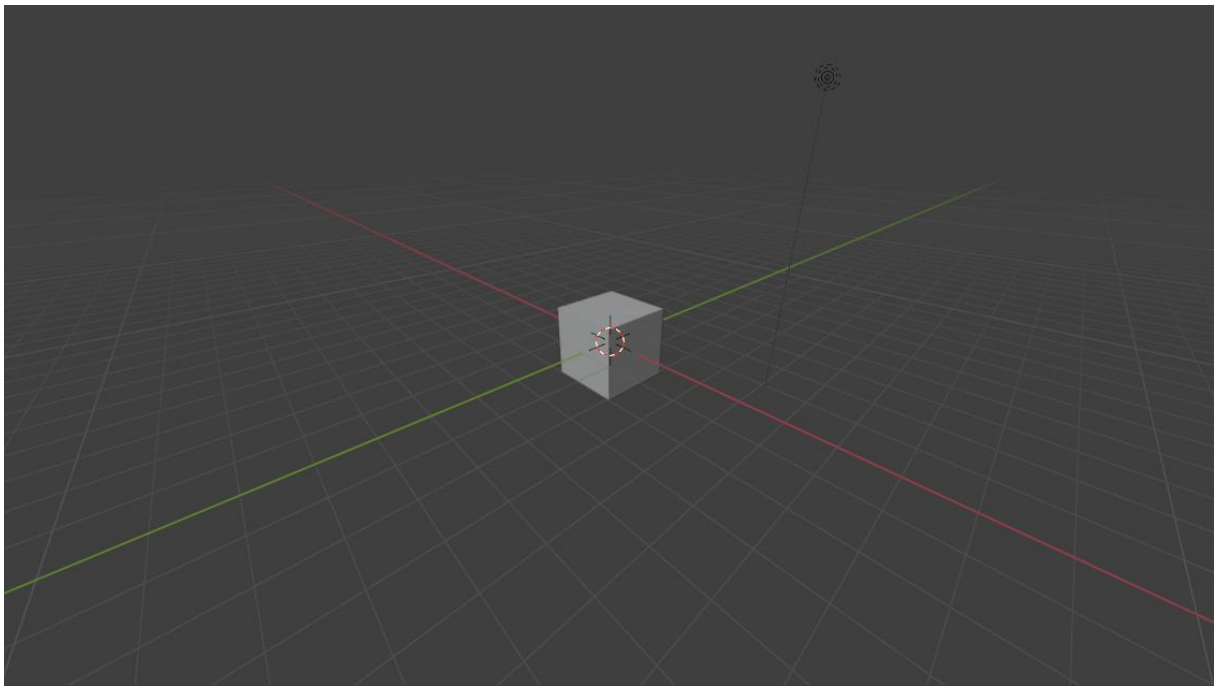


Figure 9: Camera view with a focal length of 12 mm.

In figure 9, the conditions are the same as figure 8, except that the focal length is now set to 12 mm. It can be observed that a shorter focal length broadens the FOV.
Another parameter that plays an important role in the image is the physical size of the sensor within the camera. The image sensor consists of a semiconductor chip, either a CCD or CMOS chip, which are embedded with arrays of photosensitive elements called photosites or pixels. Each pixel contains a photodiode that generates an electrical charge in response to light, with the strength of the charge corresponding to the amount of light received [3].

The sensor size has direct effects on the produced image, as a larger surface area can capture more light.

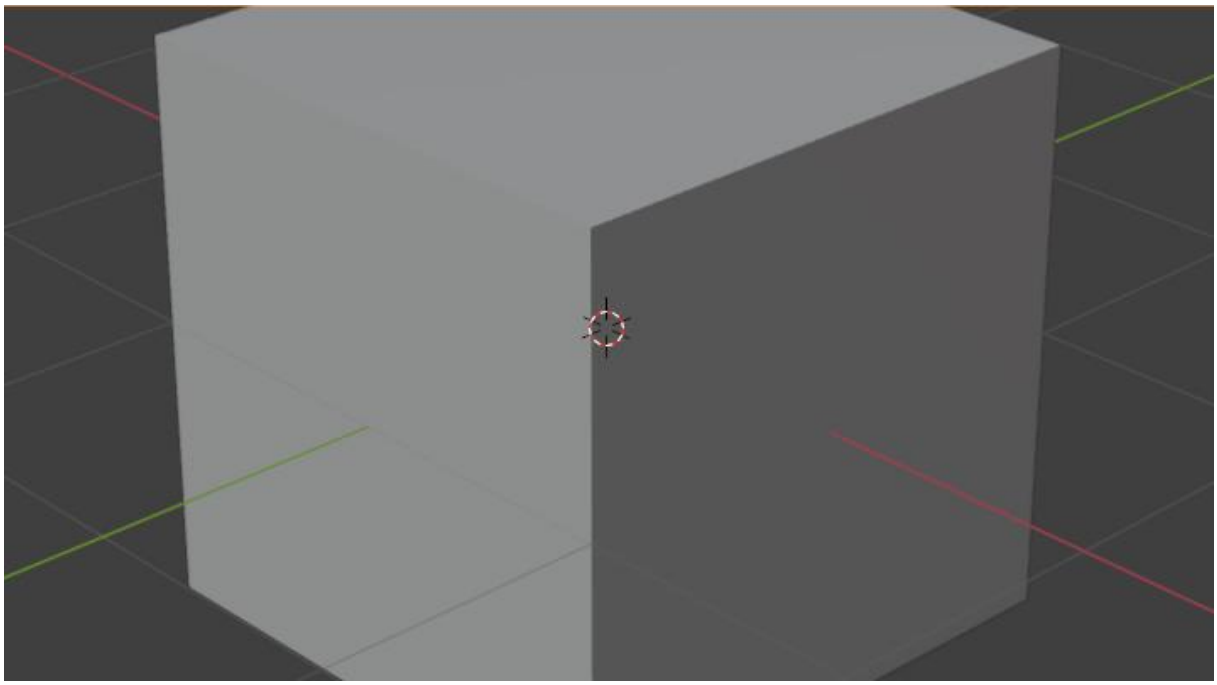


Figure 10: Camera view with a sensor width of 14 mm.

As shown in figure 10, a smaller sensor width has a direct effect on the FOV of the image.

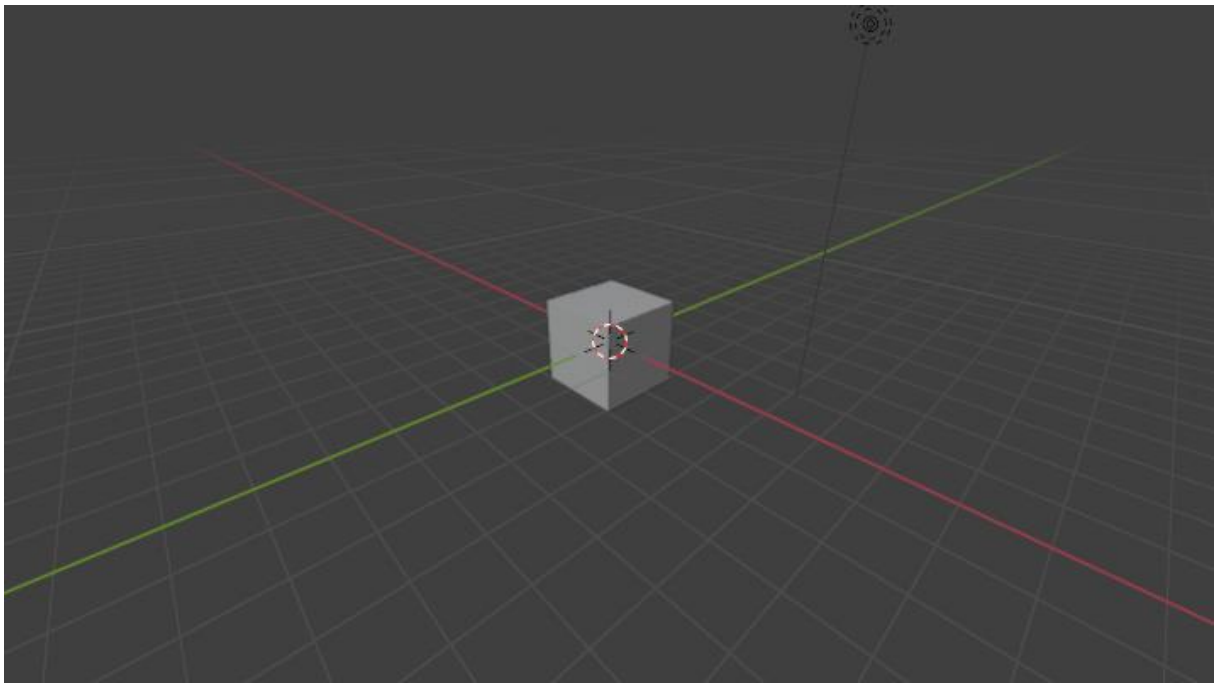


Figure 11: An image with a sensor width of 100 mm.

The picture in figure 11 shows a wider FOV as the sensor width has increased.

Real-world cameras use a lens to bend and focus incoming light onto the sensor. As a result, only objects at a specific distance appear sharp and in focus, while those closer or farther away appear blurred. This is known as DOF

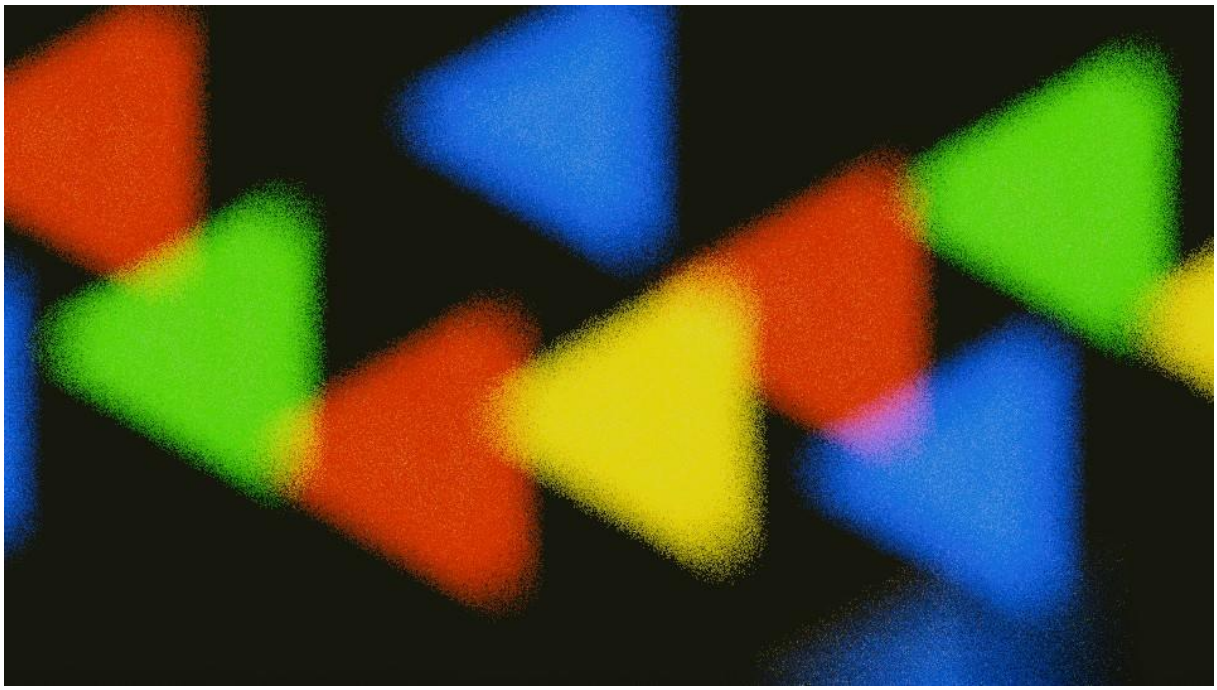


Figure 12: The so called DOF bokeh effect.

In figure 12, some of the triangles appear blurry, while others appear sharper. The area that is in focus in called the focal plane. Aperture, measured in f-stops, controls the amount of light that enters the lens. A f-stop is defined as the ratio of the focal length to the diameter of the entrance pupil.

In a compound lens, which is made up of several layers of glass elements designed to correct optical aberrations, the entrance pupil is not simply the physical aperture. Instead, it is the image of the aperture stop, as seen through the front lens elements. The size of this entrance pupil directly affects the f-stop value.
A larger entrance pupil, which corresponds to a lower f-stop value, allows more light to reach the image sensor and creates a shallower depth of field, making more of the background appear blurry. A smaller pupil, which amounts to a higher f-stop value, allows less light in, but increases the depth of field, keeping more of the image in focus [4].

Resolution is another important variable in cameras; the resolution is the number of pixels horizontally and vertically. According to [4], a pixel represents a single sample of light, intensity and color. The number of pixels on a sensor determines the resolution of the image. The higher the pixel count, the finer the detail in image will appear, as more information is able to fit inside of the image. Associated with the resolution is the pixel aspect ratio, which is the ratio of the width to height of a pixel. A ratio of 1:1 means that the pixel is square.

As stated previously, Blender offers three types of camera types. These are perspective, orthographic and panoramic. Each camera has its own specific parameters.

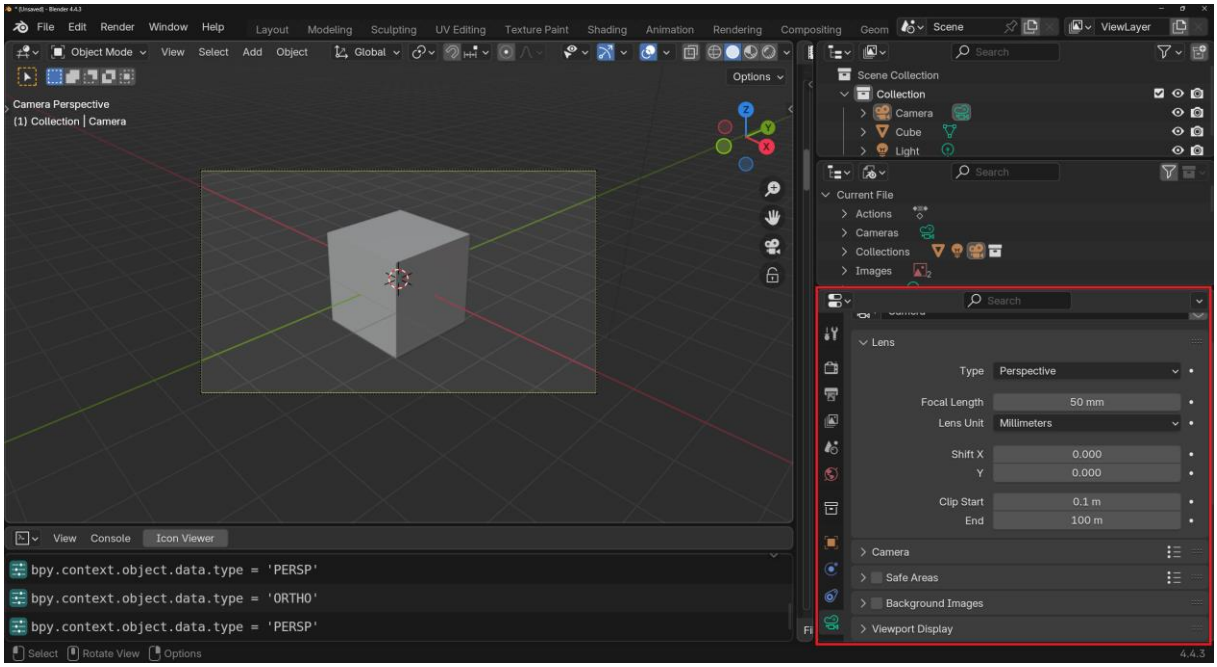


Figure 13: Highlights options for the perspective camera type.

As presented in figure 13, after selecting the perspective camera type, a couple of settings appear in the right corner. The first is focal length, which ranges from 1 millimeter up to 5 meters. The next setting allows the user to change the lens unit. The lens unit can be set to millimeters or degrees. The next setting is shift, which adjusts the vanishing points. A vanishing point is the point where parallel lines converge within a given image.

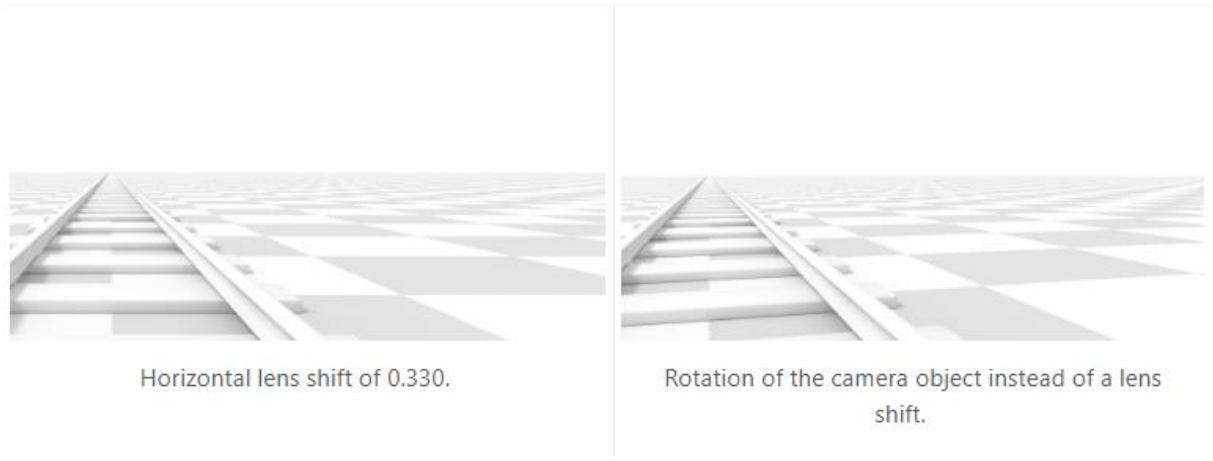


Figure 14: Difference in images when applying horizontal shift to a perspective camera compared to a rotational shift.

In figure 14, both the horizontal and vertical shift can be adjusted.

To control the visibility of objects in the image, a user can change the clip start and end, which is the interval in which objects are visible.

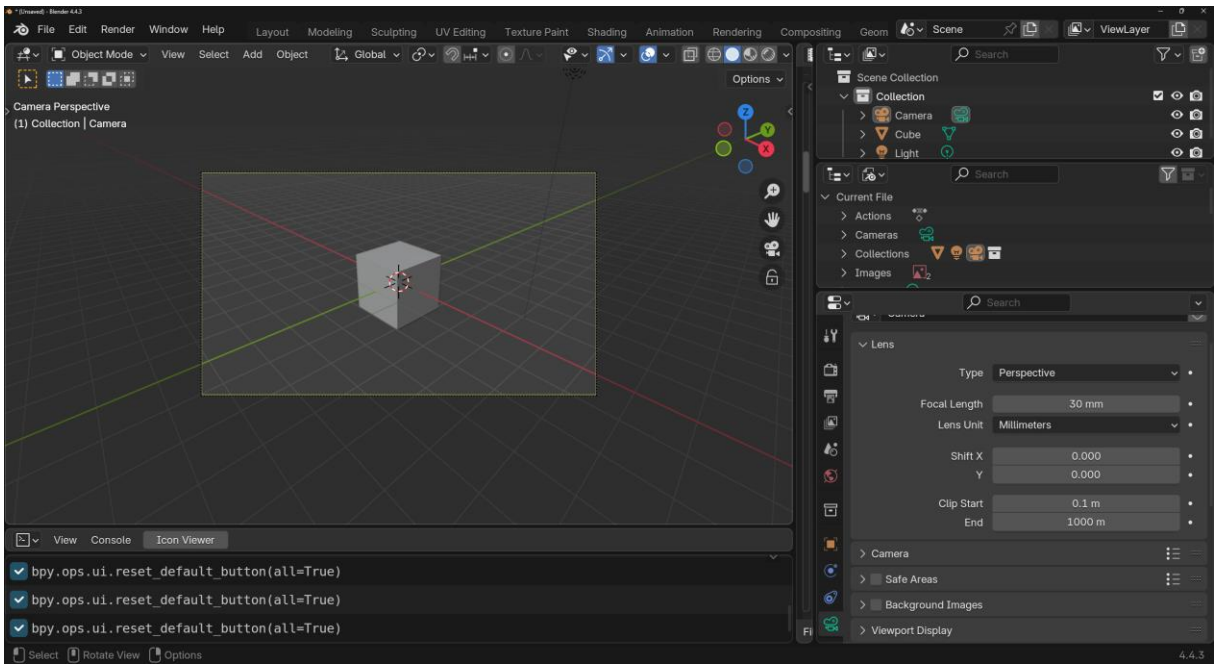


Figure 15: Clip start and end at default values.

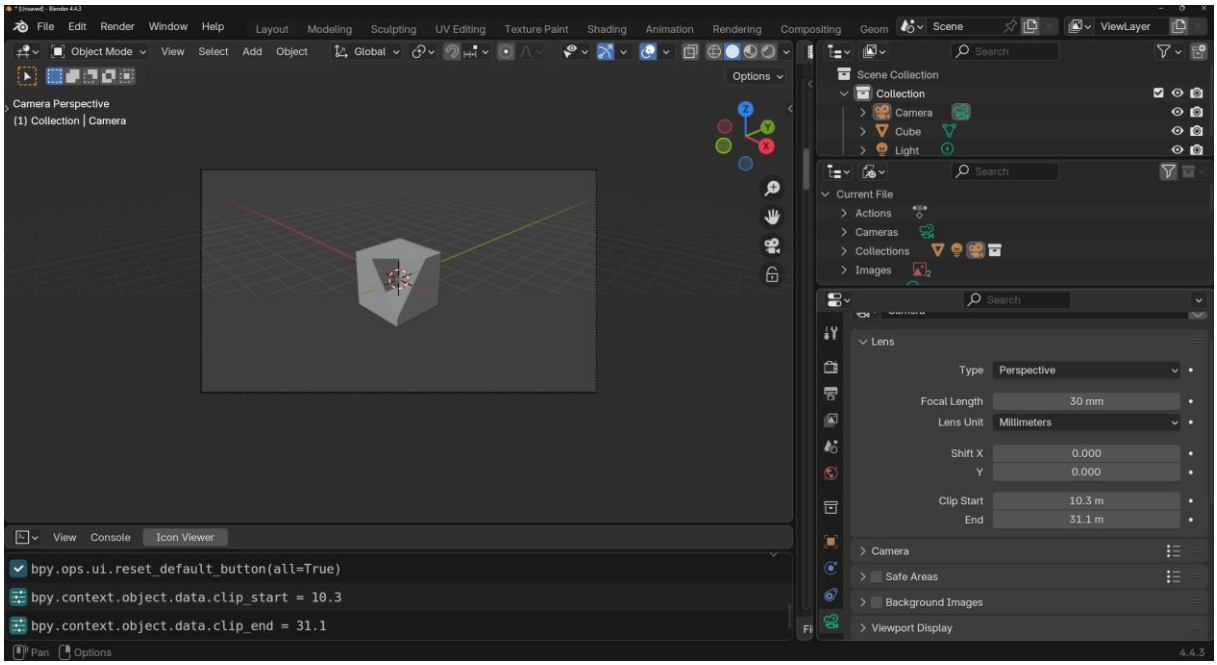


Figure 16: Clip start set to 10.3 meters and end set to 31.1 meters.

As visible in figure 16, less of the scene is now visible inside of the camera view.

After selecting the orthographic camera type, this appears in the same panel as in figure 13.

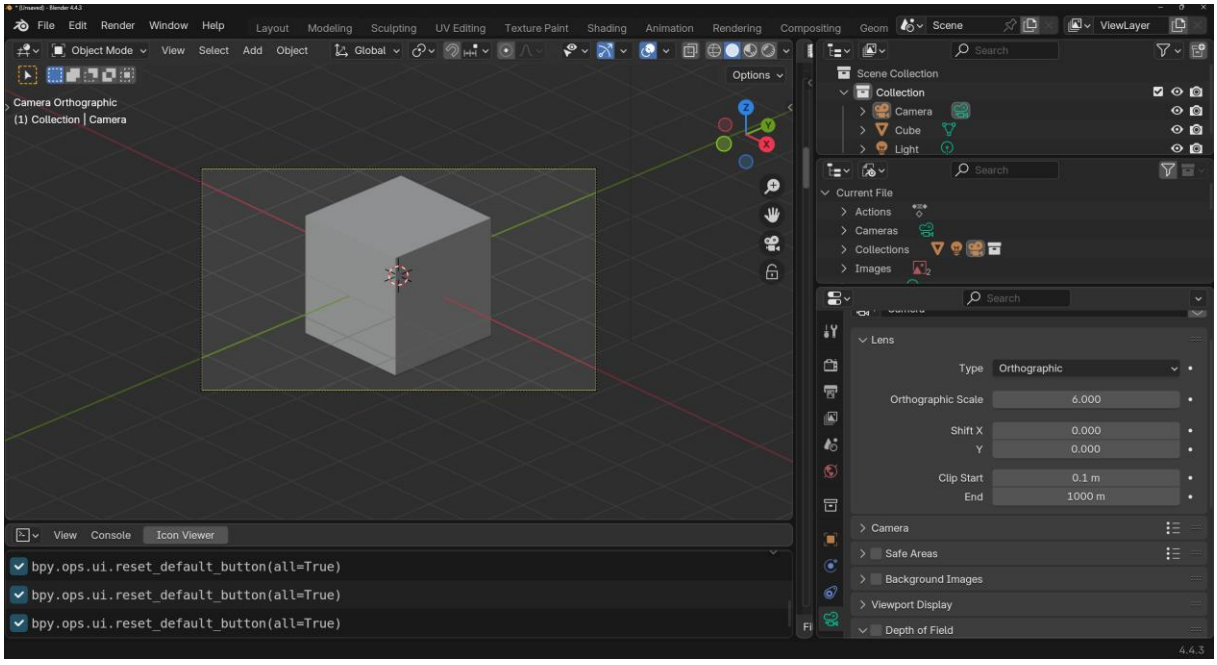


Figure 17: Camera type now set to orthographic.

The settings that are unique to the orthographic type is the orthographic scale setting. The scale is by default set to 6.

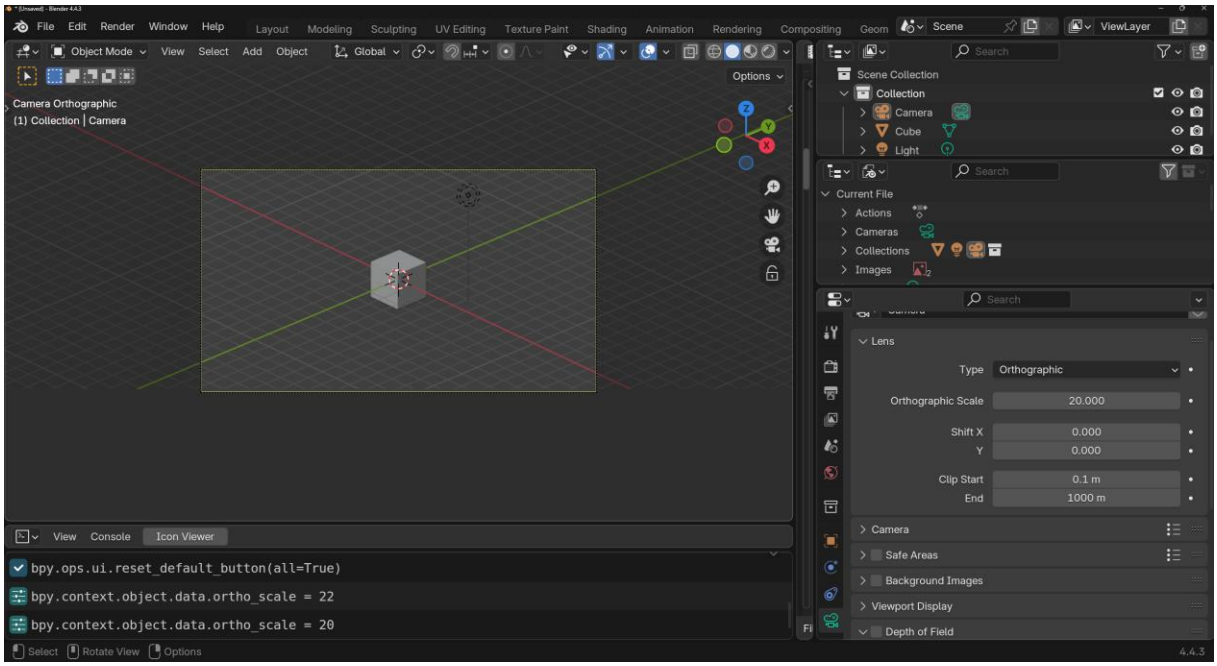


Figure 18: The camera setting orthographic scale set to 20.

Comparing figure 17 to figure 18, it can be observed that a higher scale creates a "zoom" effect.

Lastly, the panoramic type can be set.

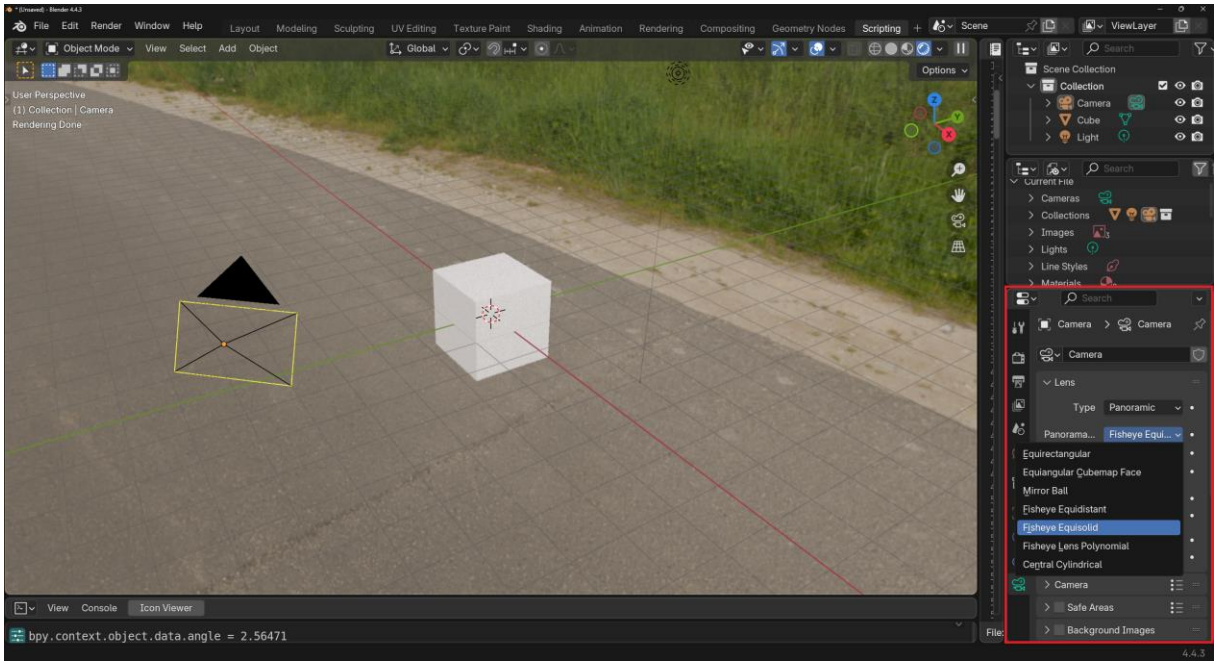


Figure 19: The drop-down menu that contains panorama types.

For panorama type cameras to be selected within Blender, the render engine must be set to cycles. Also, only some of the settings inside the panorama type will be utilized in the final tool.

For the tool, Fisheye Equisolid, Mirror Ball, Fisheye Equidistant and Central Cylindrical are of interest.

Fisheye Equisolid is a type of fisheye lens. The typical characteristics of this type of lens is a wide angle FOV with strong image distortion. This is analogous to a real camera, as it also considers the sensor dimensions.

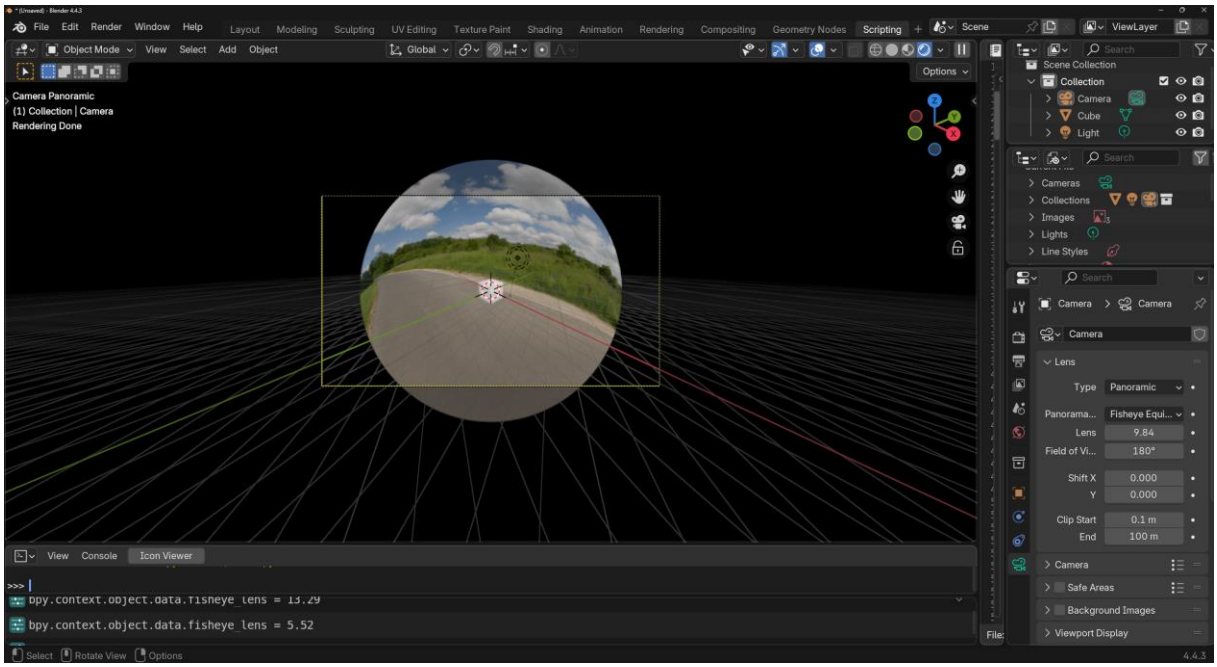


Figure 20: Panorama type set to "Fisheye Equisolid".

The important variables in this type of setup are the "Lens" and "Field of View" settings. Lens is the focal length in millimeters. Field of View is in degrees and can be set to as high as 360 degrees to capture the full environment.

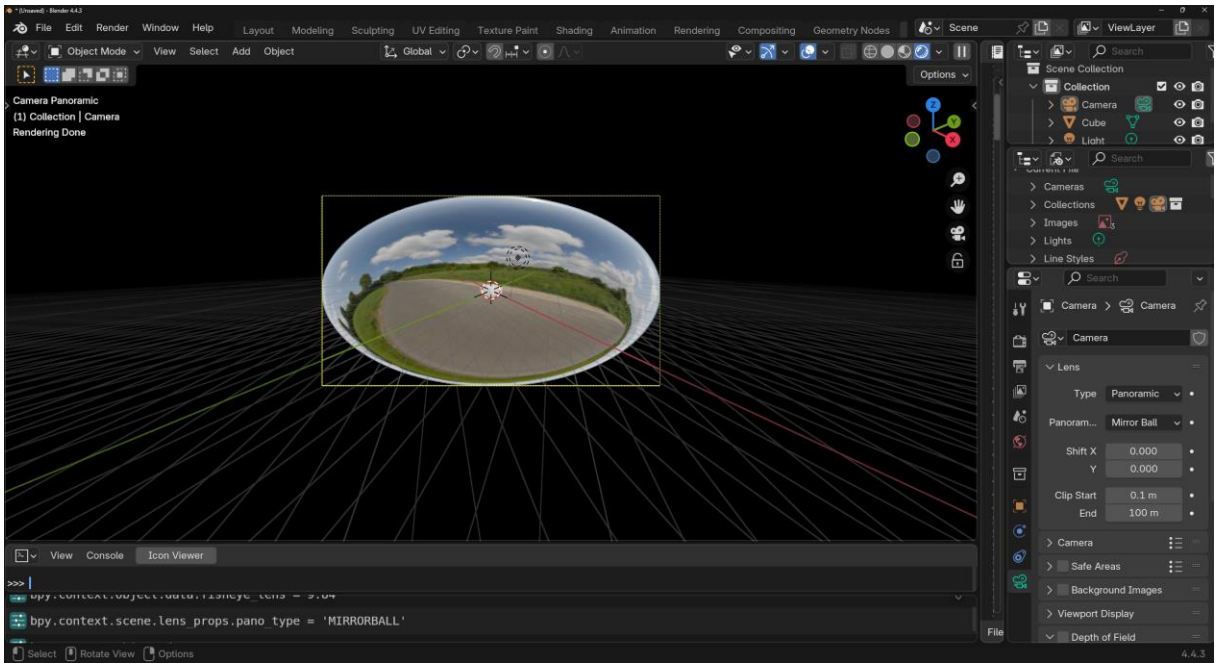


Figure 21: Panorama type set to "Mirror Ball".

The mirror ball setting appears to have a more oval structure and renders the image as if it is taking a photo of a reflective mirror ball.

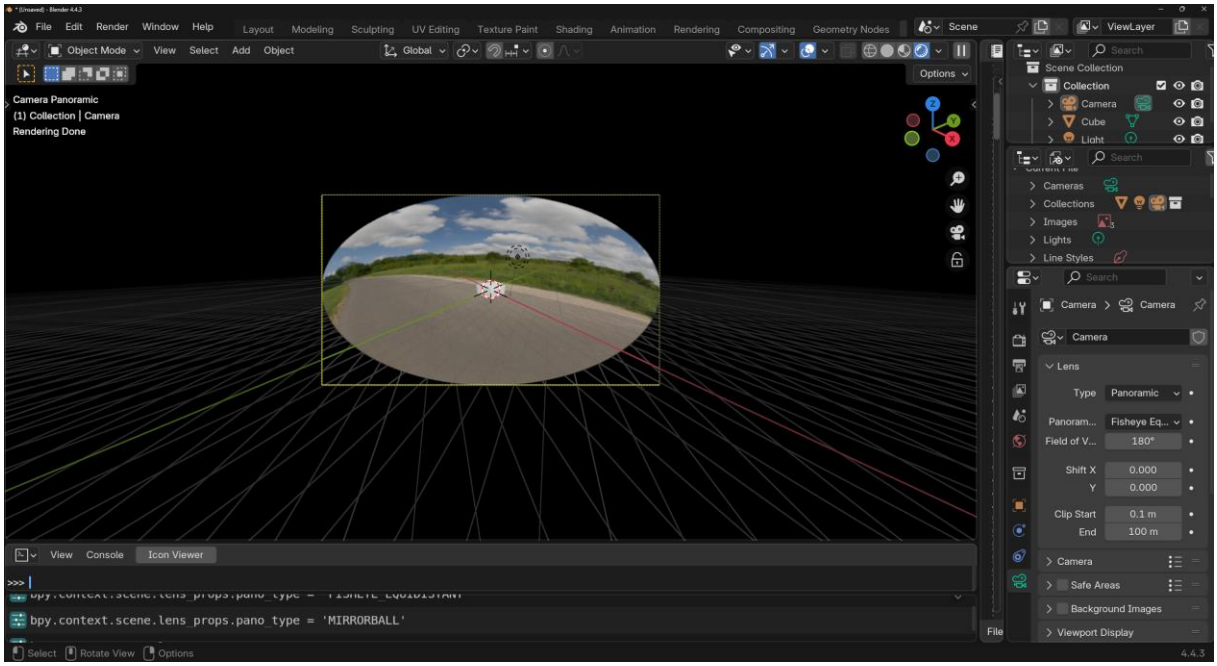


Figure 22: Panorama type set to "Fisheye Equidistant".

This model does not replicate any real-world lens model. It produces a circular fisheye image that does not take sensor dimensions into account. Instead, it utilizes the entire sensor area.

The last camera type is called Central Cylindrical; this type works by projecting the image onto a virtual cylinder from the center. It is analogous to a rotating panoramic camera [2].

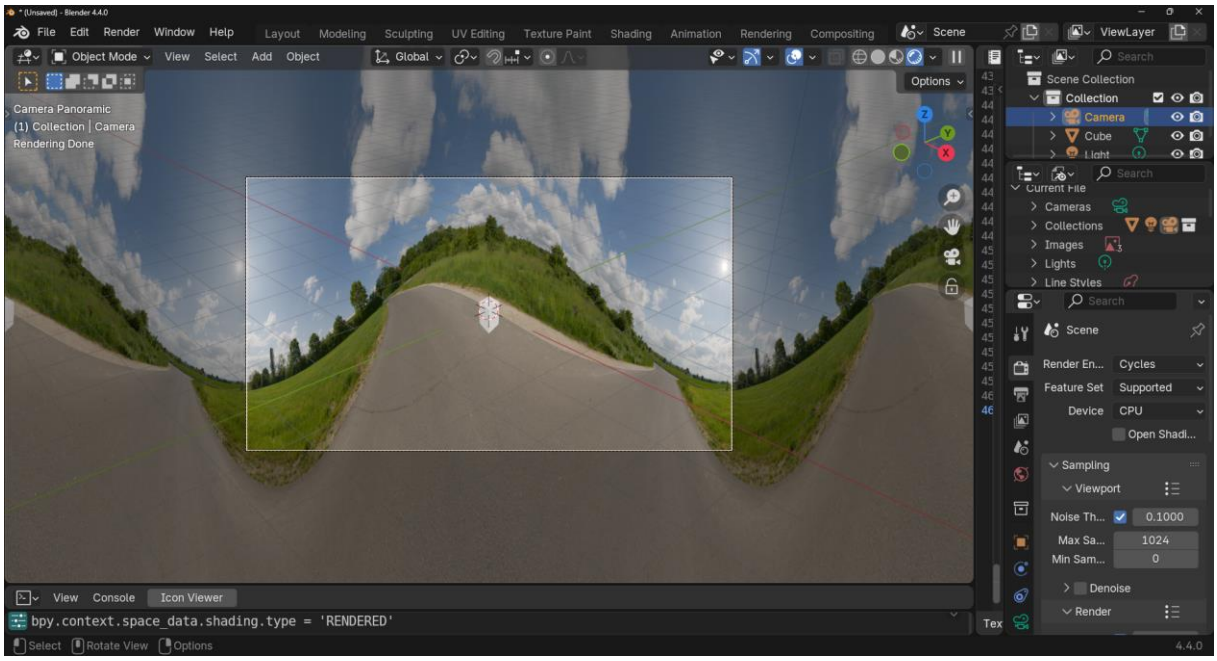


Figure 23: Camera point of view for the "Central Cylindrical" camera type.

2.3.3 The Python API and Blender

Blender offers the ability to create and run scripts using the Python API. Scripting is another way of programming. The one that many are familiar with is called traditional programming, which often consists of complex and large code, like applications or systems using languages such as C/C++, Java or C#. Scripting involves a more flexible and fast approach. It can be used to automate certain tasks, process text files, create reports or be used to link other programs [Python Scripting for computational science, third edition].

Application Programming Interfaces, or API: s, is code that helps applications communicate back and forth. The API acts as a "messenger" or "translator" to allow two different software programs to communicate. Before API: s, software was either closed, meaning that only the original developers could use it, or it was open, meaning that anyone could get access to the code. However, API: s came along and made it so that developers could share just enough of the software, without giving away all the access [5].

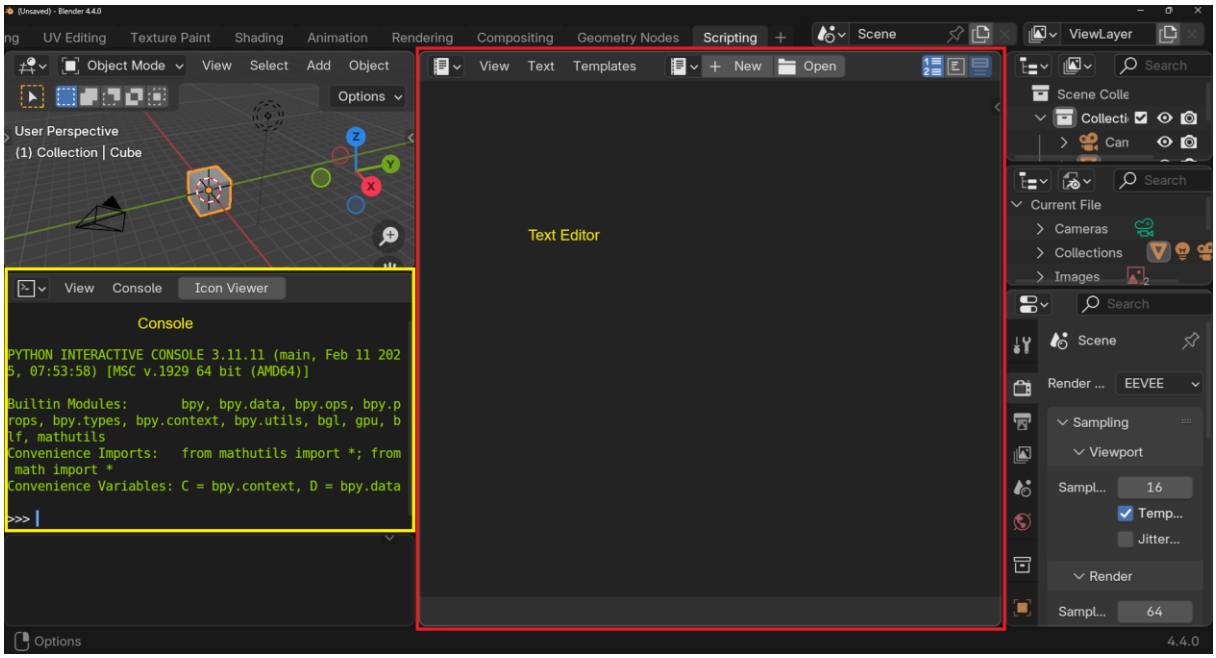


Figure 24: "Scripting" tab in Blender.

Inside Blender, there is a tab known as "scripting". In figure 24, highlighted in red, is the text editor. This is where users write their scripts. Highlighted in yellow, is the console. The console plays a key role in aiding the scripting process. As shown in the figure, the green text gives us information about what modules are built into Blender. A module in Python stores information about definitions and statements. In Blender, the most important module is the bpy module, which stands for Blender Python. Python is an Object-Oriented Programming language. A key concept in OOP is objects. An object is an instance of a class, and classes are templates for creating objects. For example, to build a house, one would need to have a blueprint which shows how to build it. The same applies to classes. In the example, the house created from that blueprint is referred to as an instance.

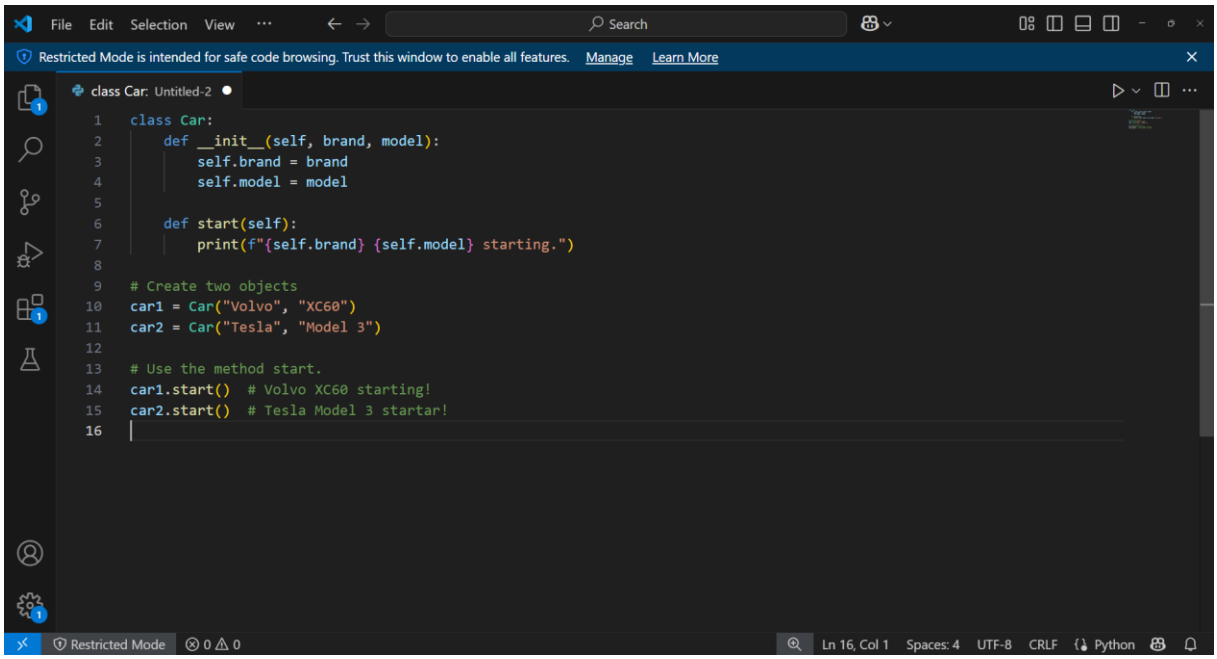


Figure 25: Example of a python "class".

In figure 25, a class named "Car", is created. Inside of this class, on line two, is a special type of method. A method is a function that is associated with a given class. The method takes the parameters self, brand and model. Self is a reference to the object that is being created, and with it, it allows the developer to get access to, in this case, the brand and model of a car. Note, that the method is associated with two underscores, before and after the keyword "init". This is a special type of method called the constructor method [6]. Each time an instance of the class "Car" is created, this code is automatically run by Python. An important note with Python is that it is case and indentation sensitive, meaning that "class" is not the same as "Class", the interpreter will not recognize "Class" as a keyword. Also, code must be indented to let Python know that it belongs to a specific block. In the figure, "def" is a special keyword that defines a function. On line ten and eleven, two instances are created, car1 and car2. Now, car1 is an object with the brand "Volvo" and model "XC90". On line six, a method named "start" is defined, now that car1 and car2 have attributes such as brand and model, it is possible to use them within the method. When car1.start() or car2.start() is called, it prints a message showing the brand and model of the car starting. To call or access attributes like brand and model, a dot is used. For example, car1.brand accesses the brand attribute of car1, which returns "Volvo".
These concepts are heavily used in Blender to create both functionality and UI.

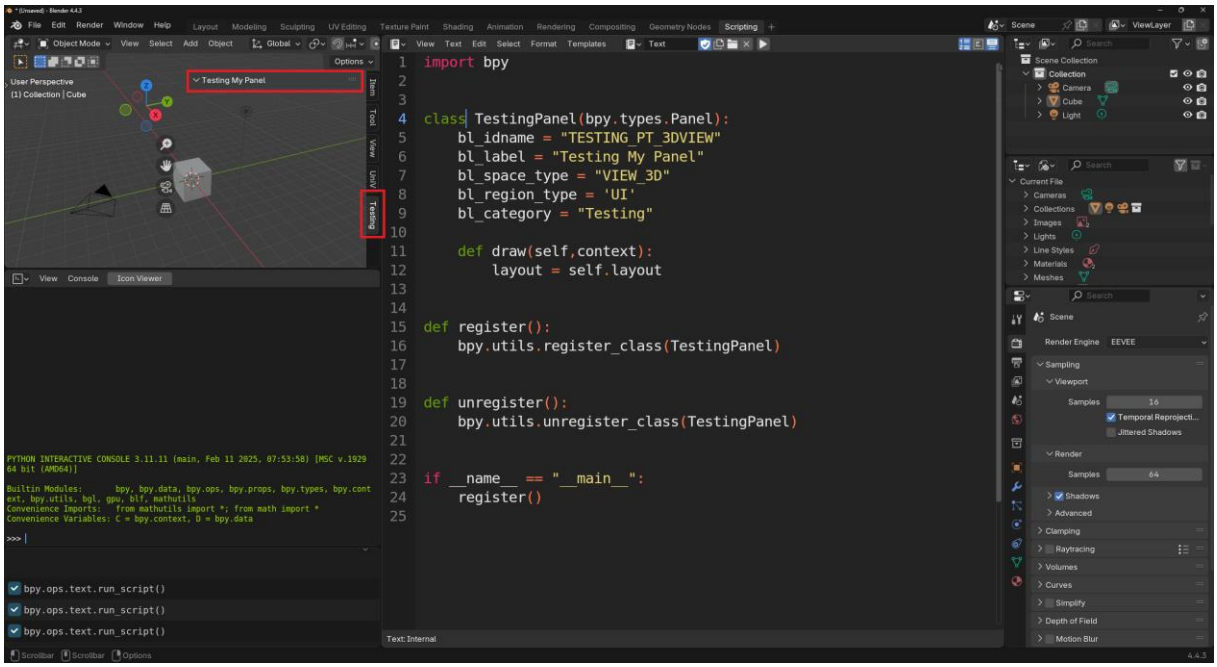


Figure 26: Blender script in the text editor tab.

In figure 26, a practical example is given which shows how classes can be used to create UI. Highlighted in red is the result of the script. When the user presses “N” in the 3D – viewport, a new category appears, named “Testing”. Inside of this category, it is possible to design and implement functionality.

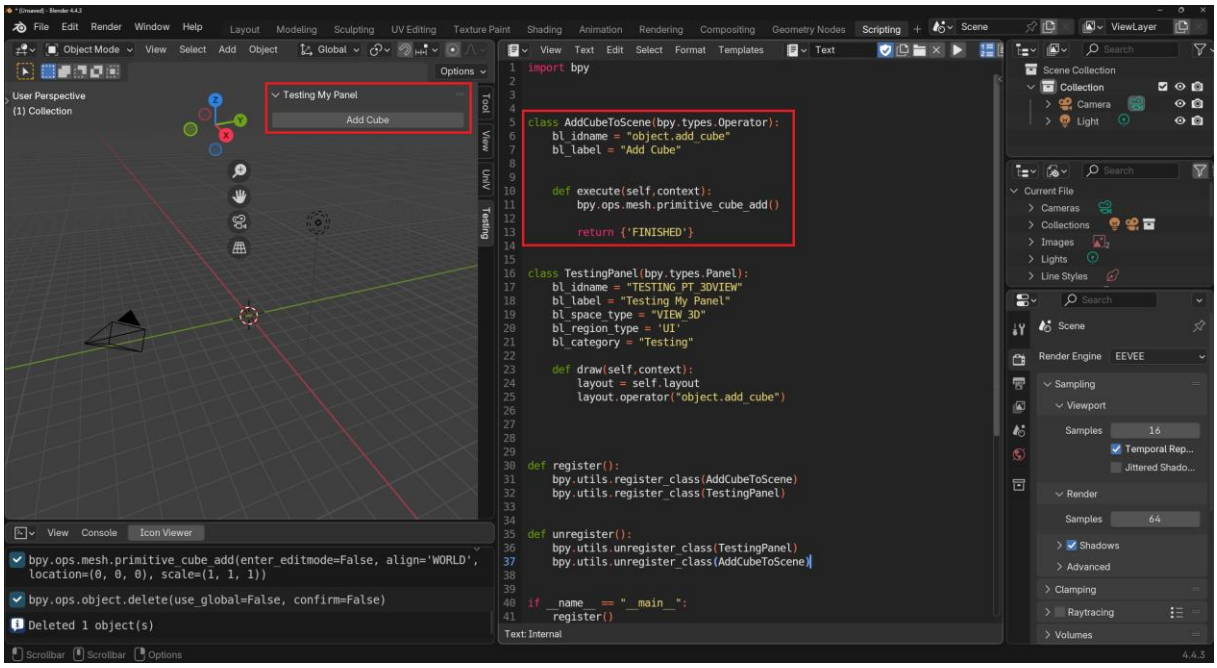


Figure 27: An updated script with added functionality.

Here in figure 27, an example is given with the same script as in figure 25, except that a button is now implemented to add a cube in the 3D – view. In the highlighted box to the left, a button is now present, which allows the user to add a 3D – cube.

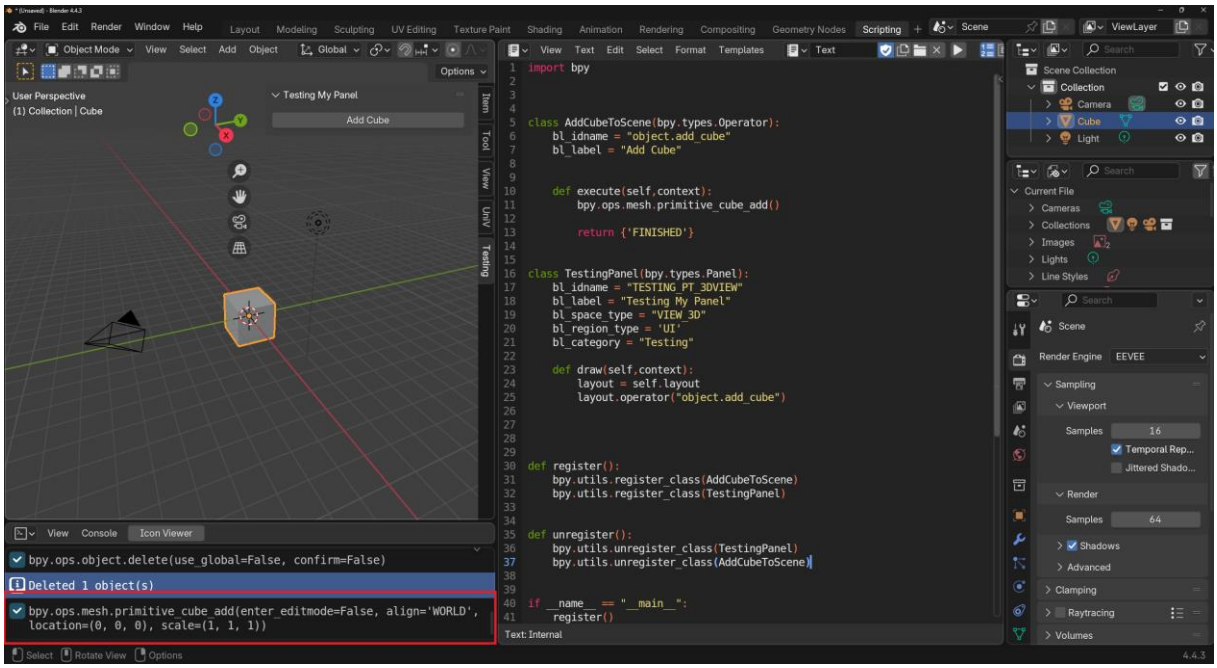


Figure 28: Displays information about the added object in the info panel.

When the user presses the button, the line of code, which added the cube is shown. This is what is written on line 11. In Blender, developers can always get information about the specific script, which performs certain actions using this info panel.

2.3.4 Add-on Development

Scripting in Blender is mainly used to develop add-ons. Add-ons are downloadable packages that extend the already existing functionality that Blender offers. Add-ons require three components to be able to run properly, and to help users navigate. The first component should show the user important metadata regarding the add-on.

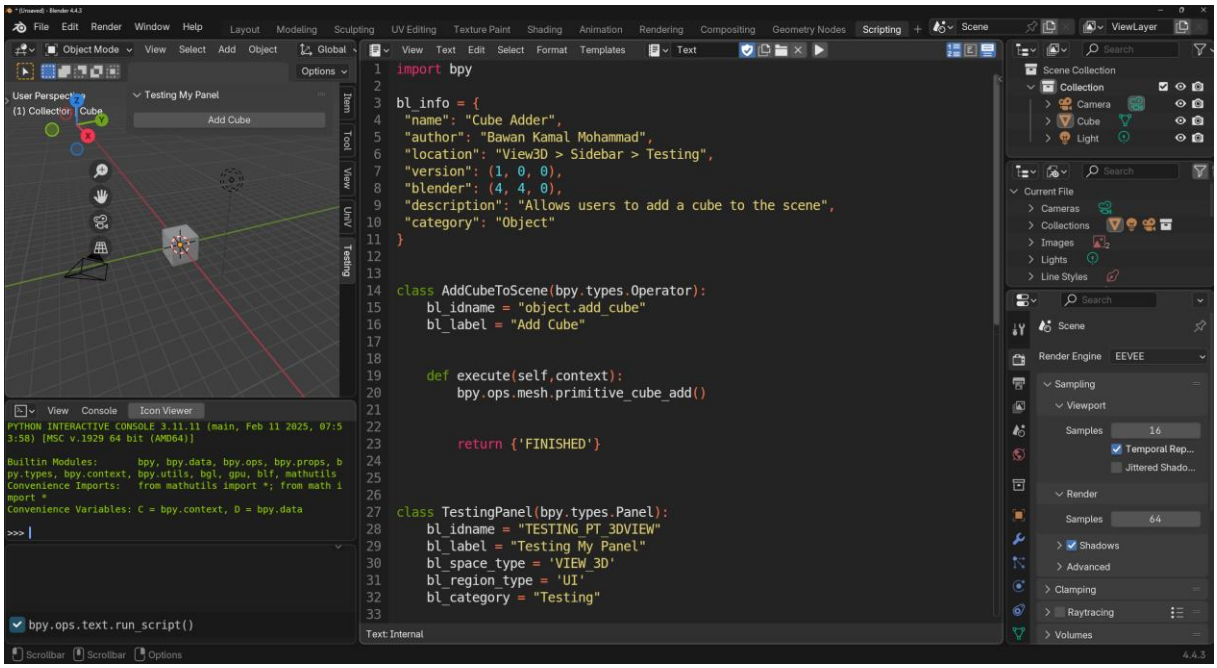


Figure 29: Updated example of a piece of code with a dictionary.

In figure 29, an added code block is shown that is added to the example in figure 27. In line 3 to 10, metadata is added to show information regarding the creator of the add-on, where it is located after downloading, the latest version and more.

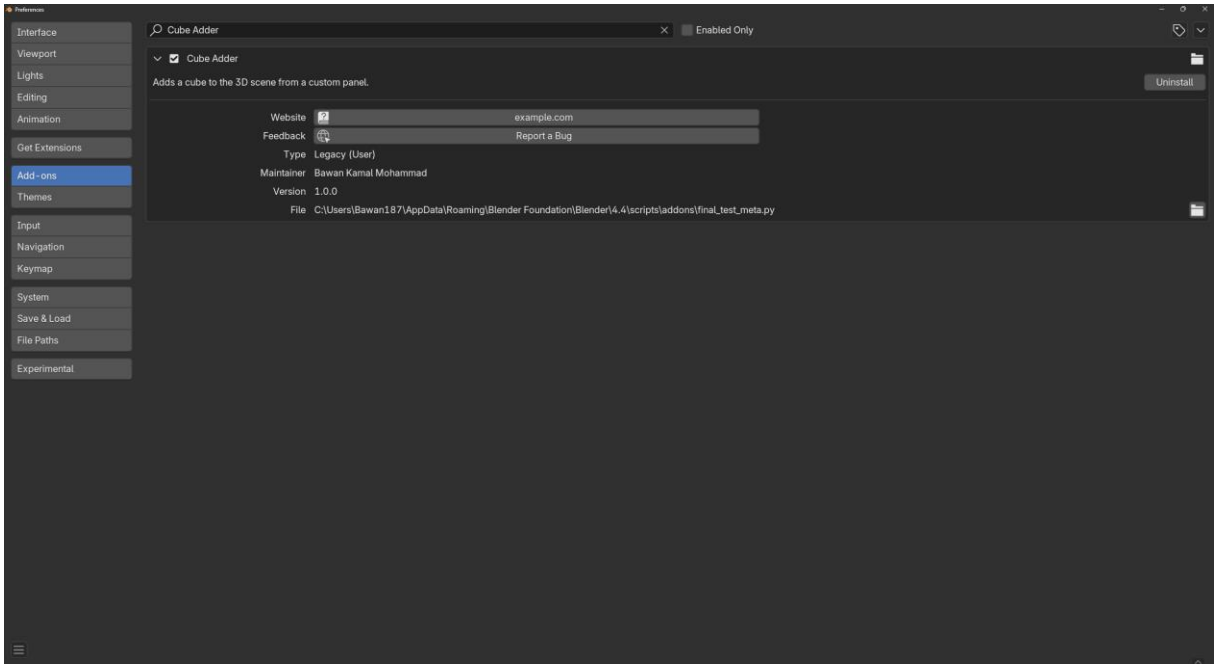


Figure 30: Specific "meta-data" of the add-on tool.

After the user downloads the add-on tool, the meta-data is now visible under the "Add-ons" tab inside Blender.

The next two required components are visible on line 30 and 35 in figure 3.6. These functions are called when the user enables the add-on and tells Blender to register all the classes that are declared within the program. On line 35, the classes are then unregistered, when the user disables the add-on, the panel and the buttons are then removed [7].

2.4 Environment

The environment is the surroundings to the main object and or objects which creates the scene to make the final render product more realistic.

2.4.1 Nodes

Nodes is a very useful tool accessible in Blender which allows the user to customize, for example, the material and or the environment's behavior in relation with light or other objects. It is a way of filtering and processing data to manipulate the results and final output in the layout tab [7].

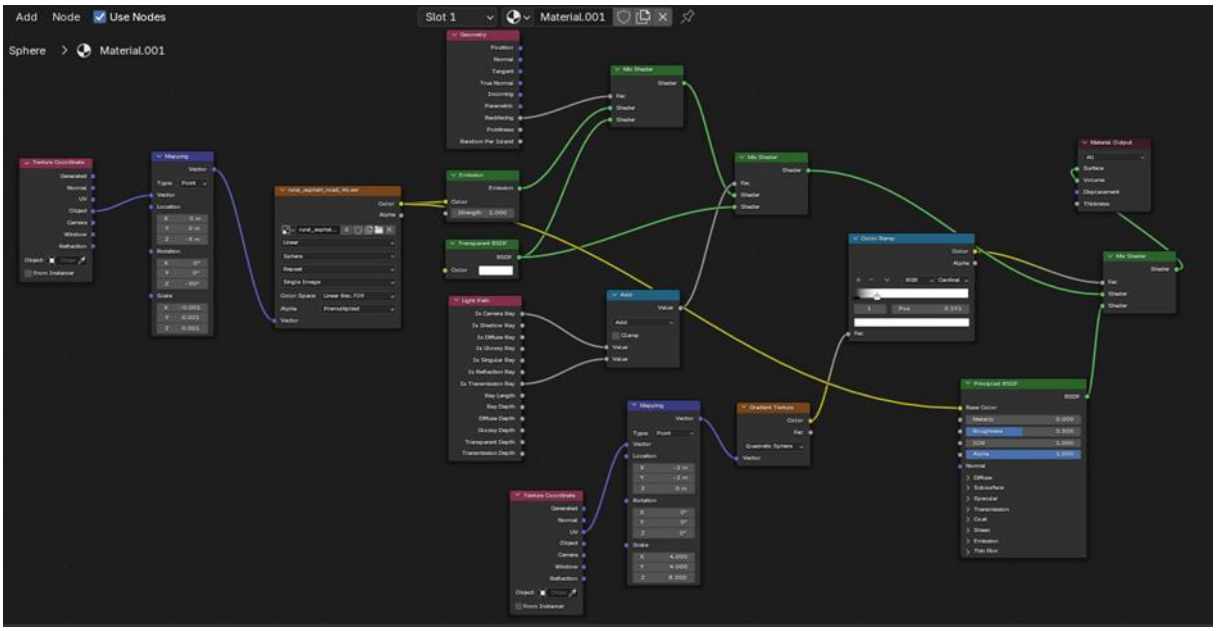


Figure 31: An example image on how it looks when several nodes are connected.

Nodes can be divided into four different categories geometry, shaders, composite and texture nodes.

Geometry nodes are used to change the geometry of an object through node-based operations.

Composite nodes allow for image enhancing, it allows for glueing two images together or enhancing the colors of a single image.

Shader nodes are used for material, lights and backgrounds and the key concept to understanding shader nodes is the output to all surface and volume shaders is a shader, this shader does not describe the color of the surface rather than the light interaction with the surface.

With for example a BSDF shader the reflection or how matt and object surface is can be changed and modified.

Texture nodes allow for the creation of custom textures through the combination of images, patterns and colors [7].

2.4.1 Object Types

Through the add menu additional objects can be added which creates the opportunity to internally add meshes, lights and cameras. The different meshes that can be added includes for instance cones, spheres, cubes and planes. Moreover, the meshes can be adjusted and modify in multiple ways from the size to shape [7].

Light in Blender appears in the 3D view as an empty object that emits light to cast shadows depending on the light type and the light types includes sun, spot, point and area. Sun light emits light with constant intensity from infinitely far way and with single direction. The point light is. Light source that emits light in all directions and it radiates the same amount of light.

2.4.2 CATIA

Catia blends modeling and simulation, commonly used for 3D CAD modeling, assembly design, product lifecycle management, simulation and analysis [8].

3 Method

3.1 CAD- Parts

All the 3D models that were used in this project regarding the vehicle and the cameras were made by Polestar before the start of this project. However, the obstacles that were used to test the positioning of the cameras when placed on the vehicle were made to follow Polestars internal regulations.

3.2 Creating the Environment

In the thesis work description, a specification for the environment was for the final render images to include a rural road to make the gap between simulation and the real world smaller. Therefore, the first step towards the final product was to create a setting for the environment to look realistic and behave like the real world. To create a realistic environment, parameters had to be set to be able to have a tangible end goal.

The environment is built using nodes and the main piece of the node network is the image texture node which uses an image and sets it as background in the layout tab. The nodes around the image texture node are there to adjust the image texture node so that it can be modified in ways such as rotation, light and scale.

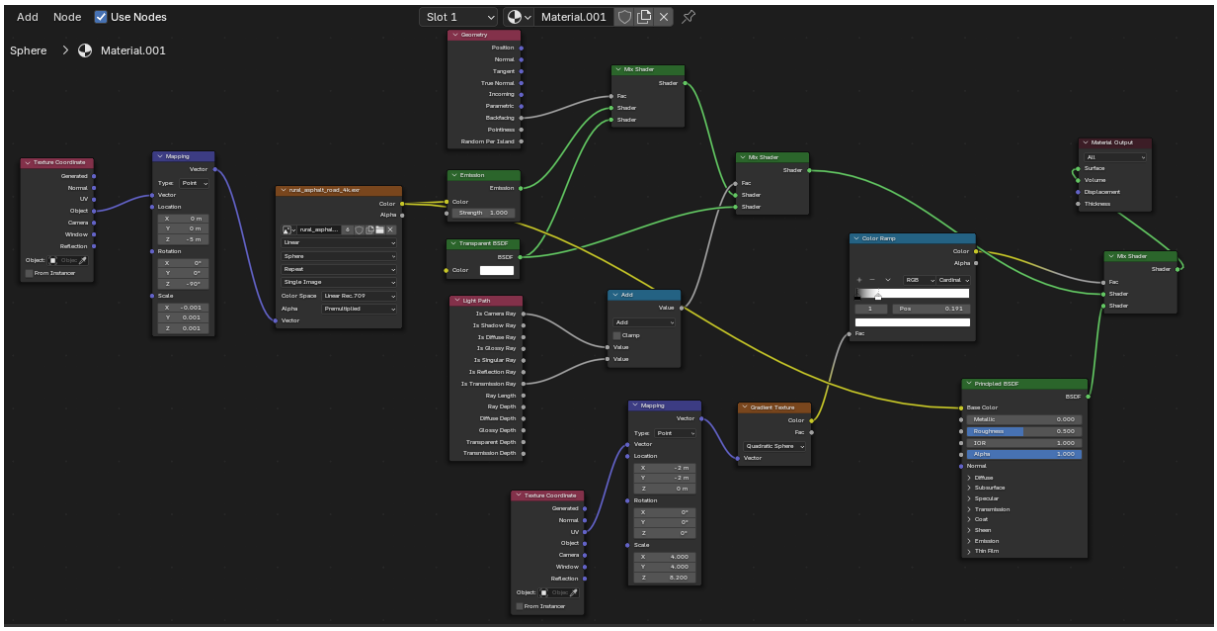


Figure 32: Overview of the node network.

The first parameter was for the object, in this case a vehicle to be located at ground in relation to what the background is. The initial problem was that when using an environment texture node that it was not possible to add a ground level so that the vehicle was for example always and in all angels on the road and with this inconsistency it would become impossible to know if the angels or if the field of view was correct. Furthermore, this led to the solution to create a dome-like structure that has an image texture node that projects the image file onto the dome.

3.3 Learning Blender

To begin developing the tool, Blender needed to be studied. The first step in the process involved gathering user documentation. Fortunately, Blender's official site offered a support page, where it was possible to review the basics of using the software.

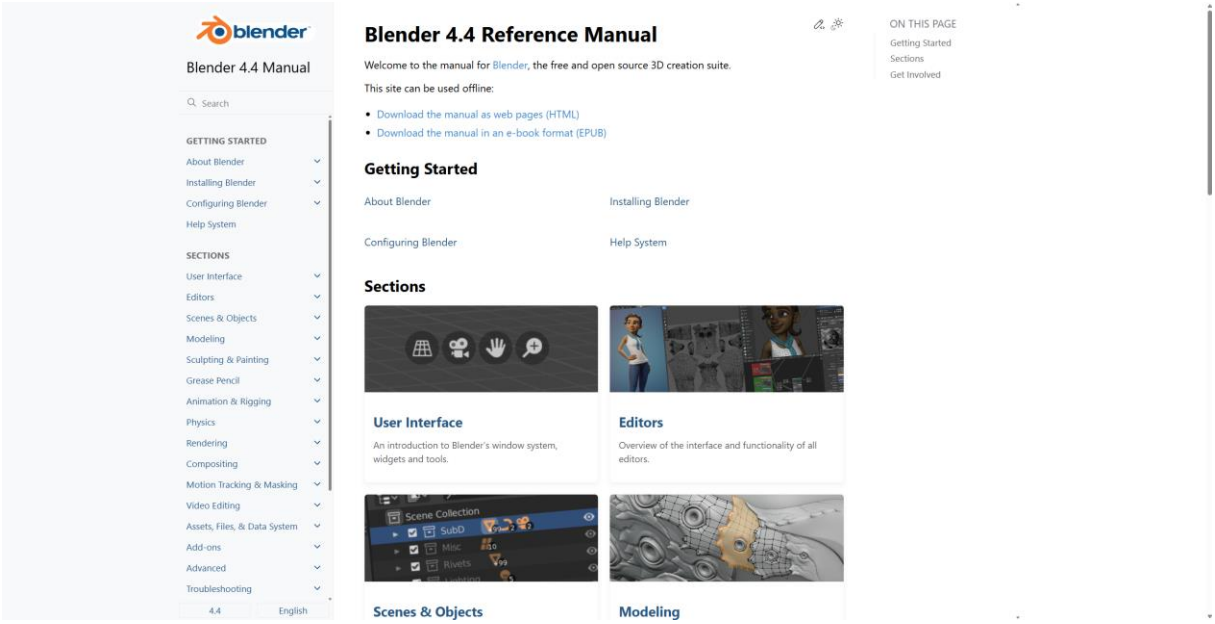


Figure 33: The Blender version 4.4 Reference Manual

The official reference manual included information needed to firstly install and configure the program. After this process had been completed, the group needed to get familiar with navigating the different parts such as the user interface and camera functionality.

3.4 Reviewing Python

After learning the basics of navigating inside Blender, Python needed to be reviewed. The python API is embedded in Blender, and with it, users can create scripts that automate tasks and create custom interfaces. However, before starting the scripting process, the group needed to gather reliable sources on Python. On Blender's official site, an API reference was available.

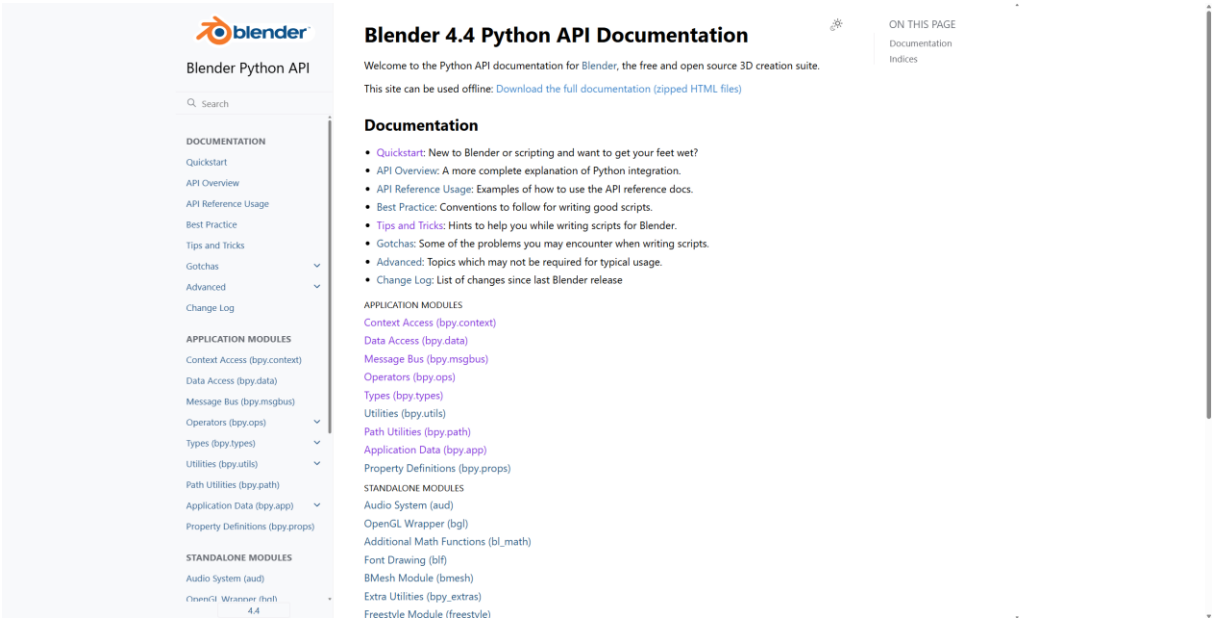


Figure 34: Python API documentation

In figure 34, the documentation contains quick-start guides, tips and tricks for writing scripts and most importantly, gives the user knowledge about the different modules that are available in Blender and how to use them.

3.5 Identification of Basic Functional Requirements

After the necessary knowledge about Blender and Python had been gathered, the next step of the process was to discuss the layout and functionality of the software. Blender offers many types of UI elements; these can be placed inside different areas.
The tool needed to be easily accessible to users, and some test – scripts were created to showcase the different locations that the tool could be put inside of.

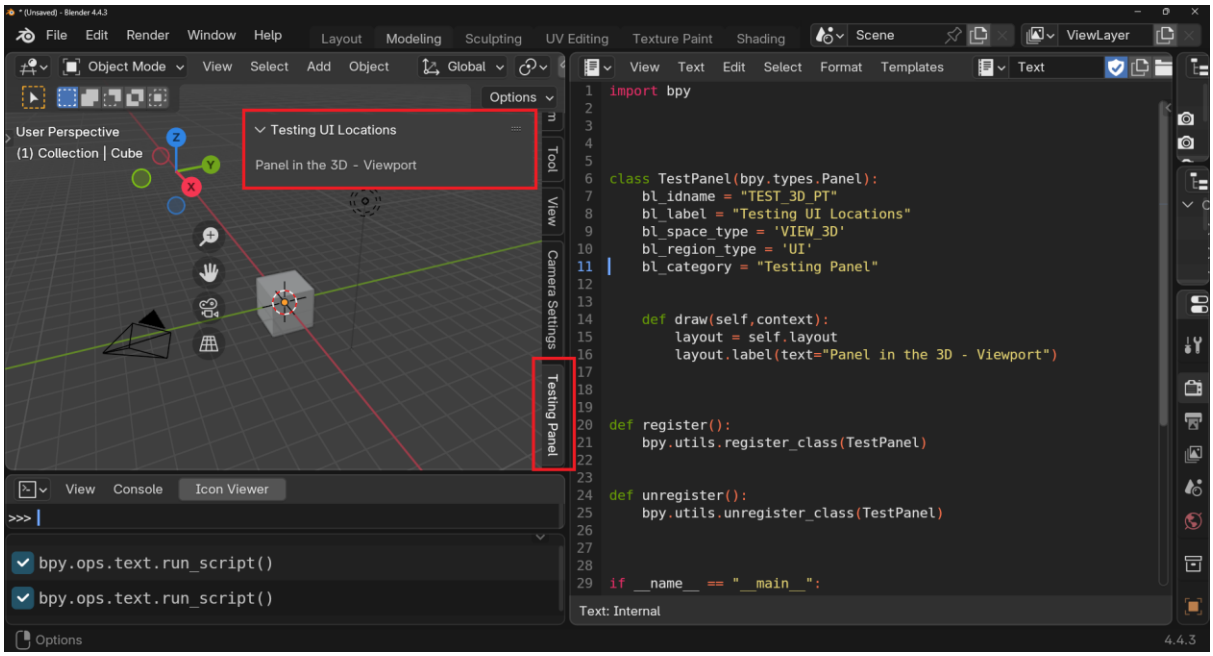


Fig 35: Test-script in Blender, panel placed inside of the "3D – Viewport".

Here, the first location is the 3D – Viewport, and is accessible to the user by simply pressing N on the keyboard. A test – panel is created, and UI such as buttons and menus can be placed inside of it.

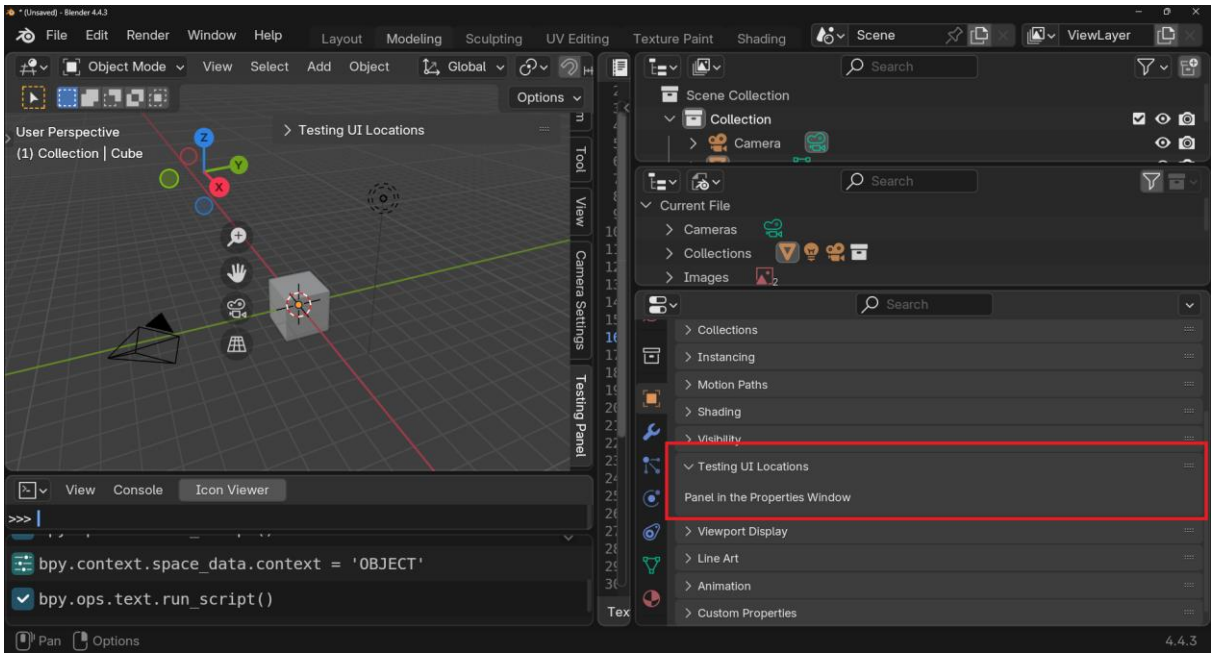


Figure 36: A panel placed in the "Properties Window"

The next location is inside of the "Properties Window", and is the same exact panel as shown in figure 35. This area has many sub-areas and is slightly less visible as the panel can be placed in different categories.

The decision was made to place the functionality inside of the “3D – Viewport”, mainly due to the slightly better accessibility. The user can simply press “N” and view and collapse the panels, comparing this to the “Properties Window”, the panel location appears to be less clear, and can be harder to find. After this, the next step was to start designing the functionality of the first panel, namely the “Camera Selection” panel. Here, the first part was to create a drop-down menu, which allowed the user to select three different types of camera setups, or presets. In the menu, fisheye, fixed-zoom in and wide-angle cameras can be added.

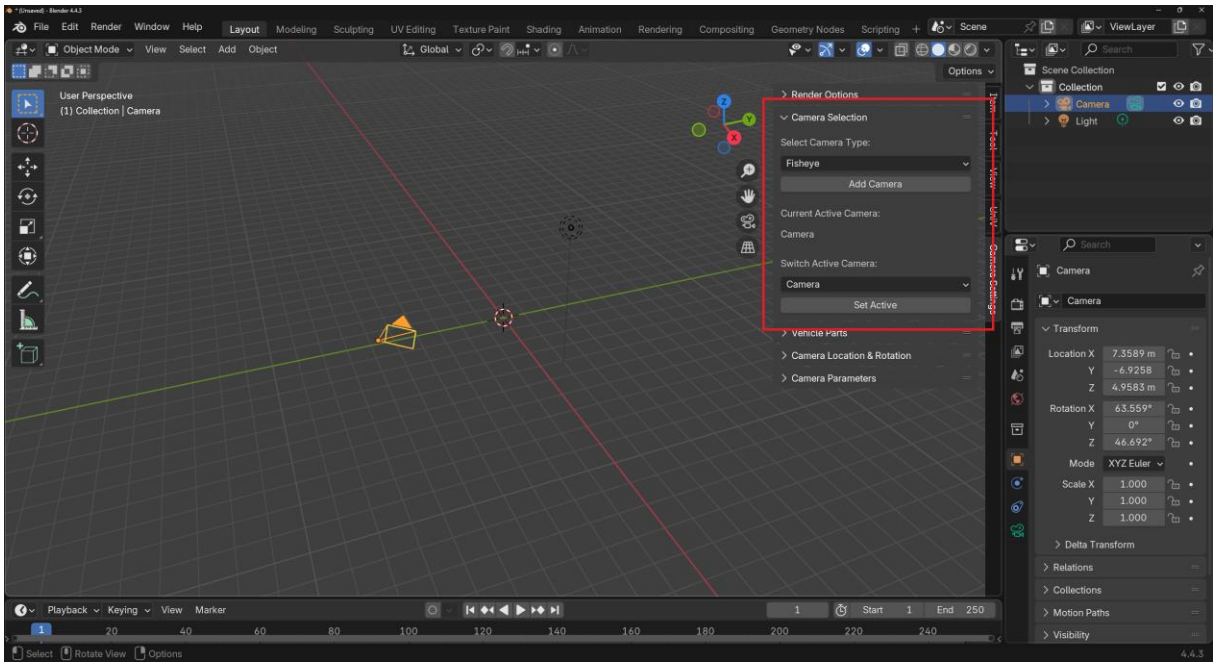


Figure 37: The developed “Camera Selection” panel.

Inside of this menu, a simple text prompts the user to select a camera type using the drop-down menu below. Pressing the menu displays three options.

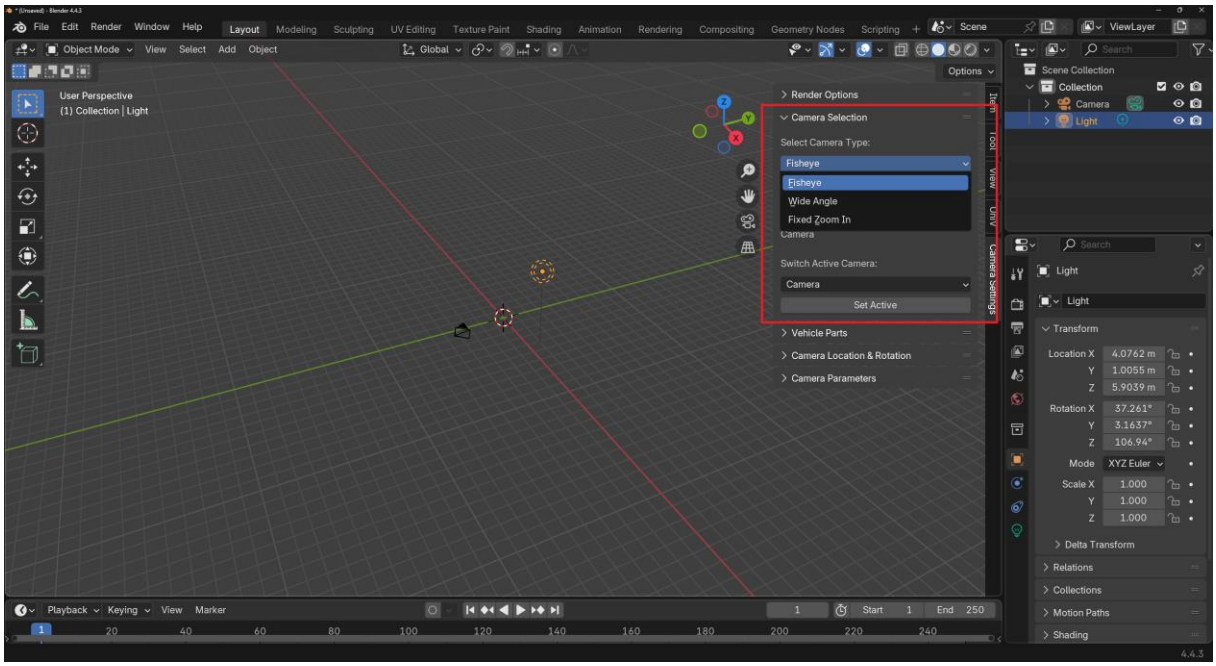


Figure 38: The scripted drop-down menu inside of the "Camera Selection" panel.

Here, the user can select three presets. The presets have different focal lengths by default. Adding the fish-eye camera sets the focal length of the added camera to 10mm, the wide-angle 18mm and lastly, the fixed-zoom has a focal length of 85mm.

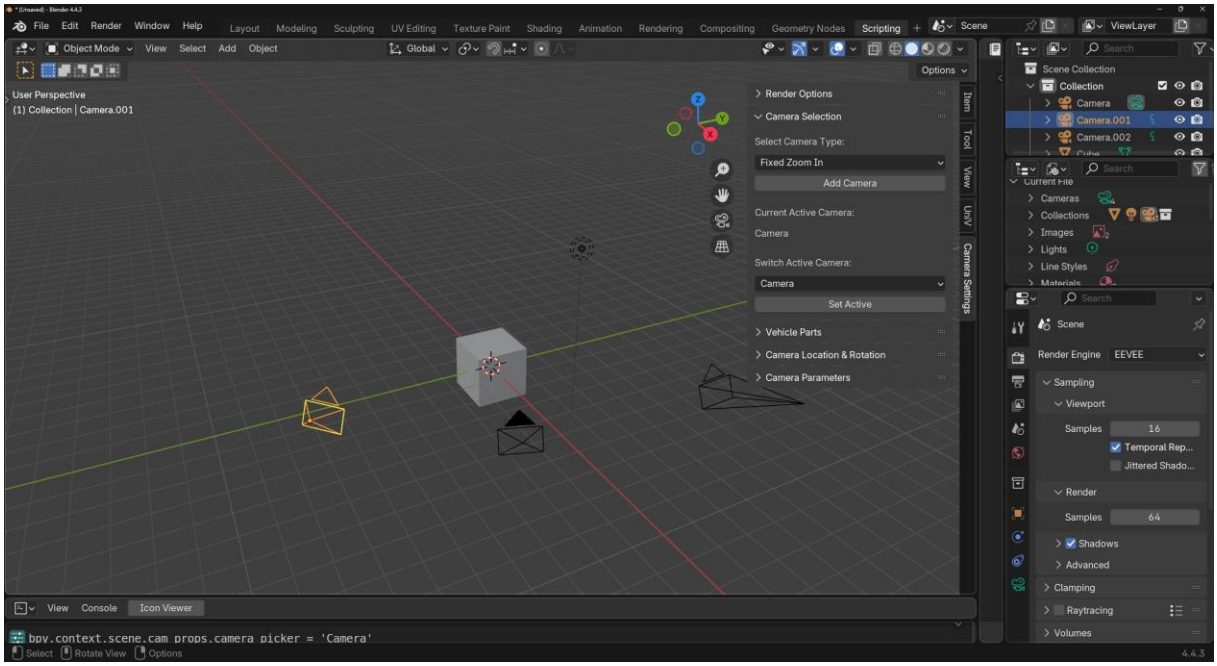


Figure 39: All three cameras added from the drop-down menu.

In figure 39, all three cameras have been added to the scene using the “Add camera” button. After a camera is added, the name of that specific camera is displayed below the text “Current Active Camera”. The active camera can be changed using another drop-down menu.

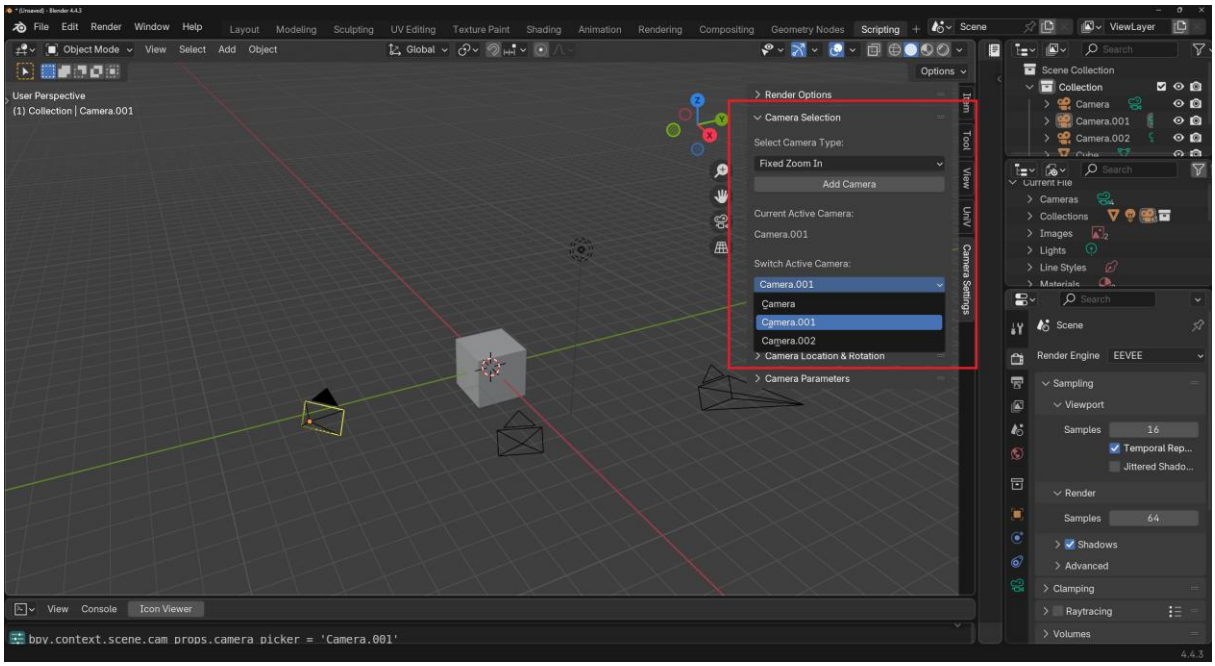


Figure 40: The drop-down menu for changing active camera.

As figure 39 shows, all three cameras are added to the menu and only one can be active at a time. For example, selecting “Camera.002” as the active camera generates a message that is returned to the user. Also, a black triangle appears above the active camera.

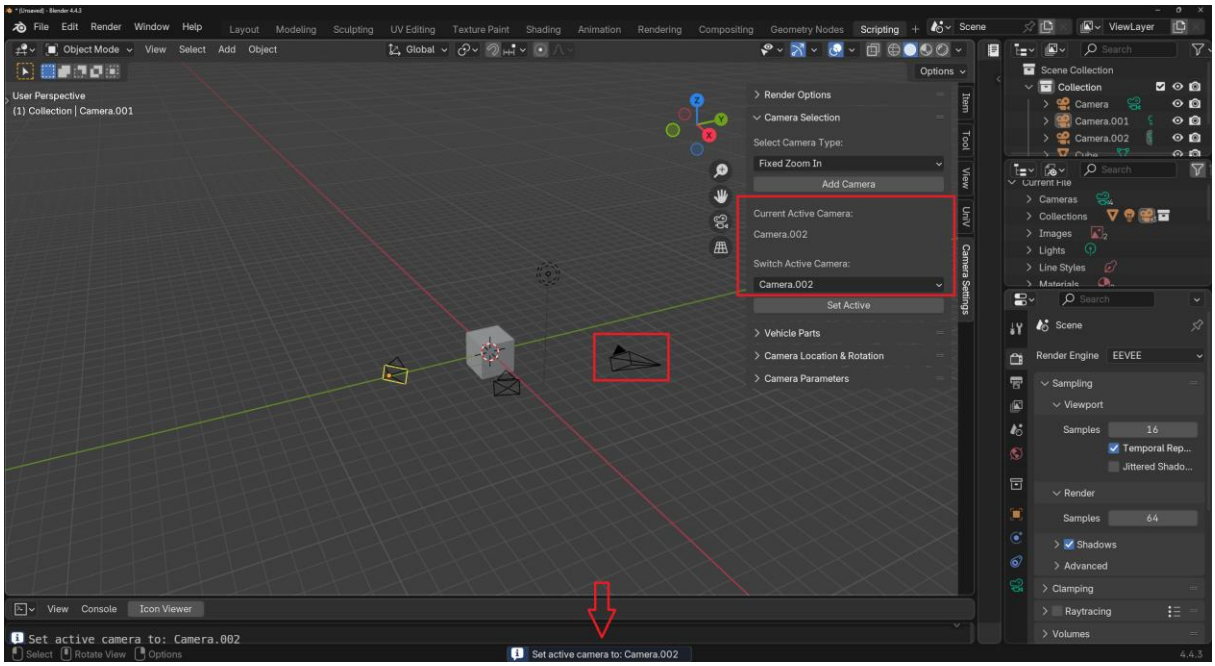


Figure 41: The newly selected active camera.

Now, in figure 41, "Camera.002" is the active camera. The message at the bottom is generated when the button "Set Active" is pressed.

Next, the cameras needed to be able to move, rotate and scale all along the XYZ-axis. This is done through the "Camera Location & Rotation" Panel.

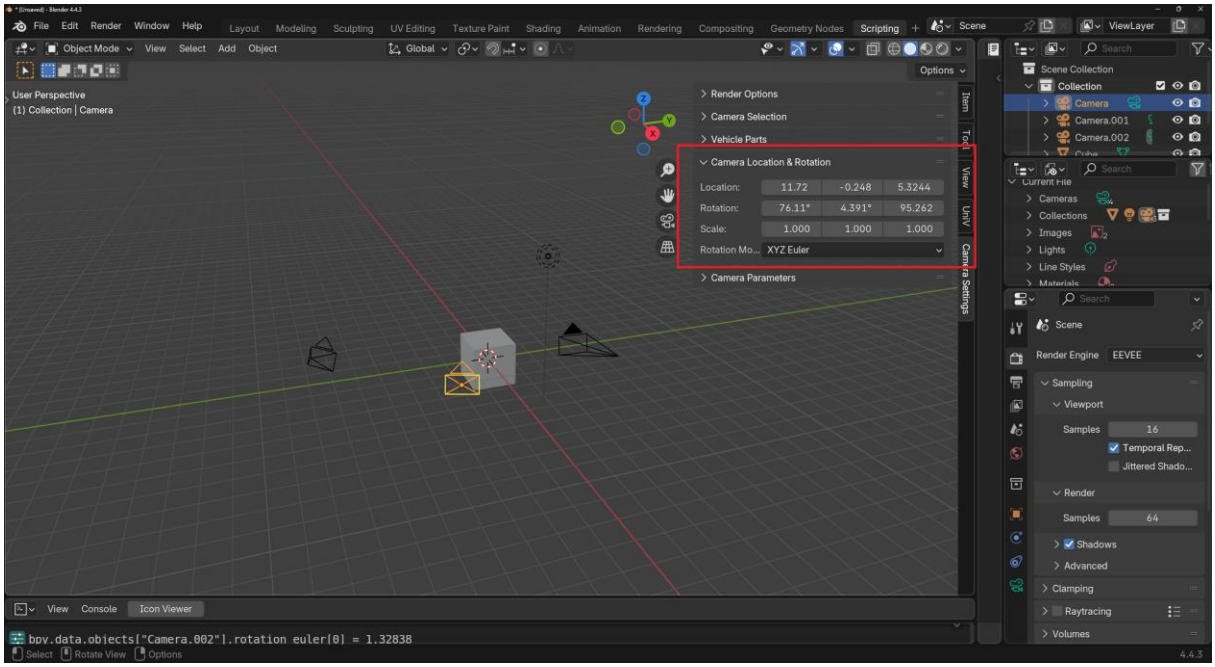


Figure 42: The developed "Camera Location & Rotation panel".

In figure 42, the panel allows the user to adjust a camera's position along the X, Y and Z axes. These values can be modified either by using the sliders associated with each axis or by manually entering the values via the keyboard.

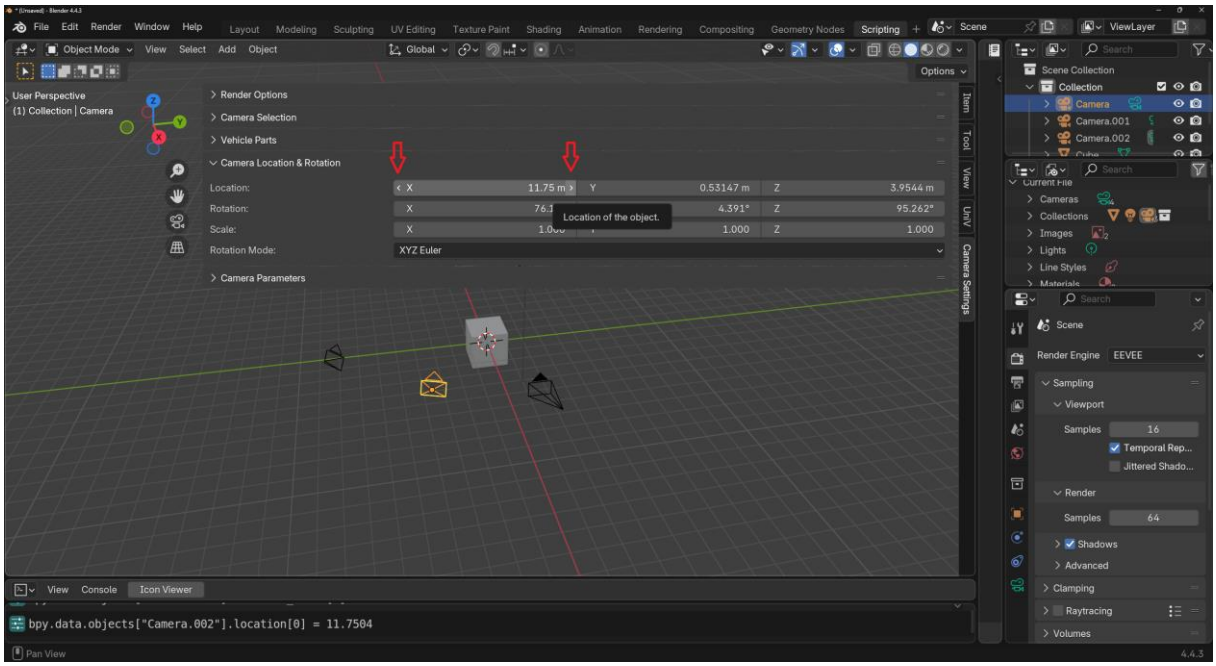


Figure 43: Sliders that control the coordinates, rotation and scale of a selected object.

In figure 43, the arrows point to the buttons that allow the user to change the location of the camera in the X-axis. Next, is the rotation, which is set in degrees and allows for rotation along the X, Y and Z axis. Inside the panel,

4 Results

Shortly after the development phase of the project had been completed, the finished add-on was handed over to the ADAS – team at Polestar for testing and finalization.

A comparison was made between the setup – time for the previous tool with respect to the new add-on. The engineers found that the add-on provided better functionality as more detailed setups and conditions could be tested for. These included more accurate lighting and more flexible camera placement. The final tool did not require any background knowledge regarding programming, however, if the engineers wish to change or add to existing functionality, the script that the tool is based on can be easily extended.

It provides all the necessary camera setups with the option to add three different camera presets directly to the simulation. The location of these cameras can be adjusted, along with the rotation. Also, parameters such as focal length, sensor width and height, resolution are all accessible to the user.

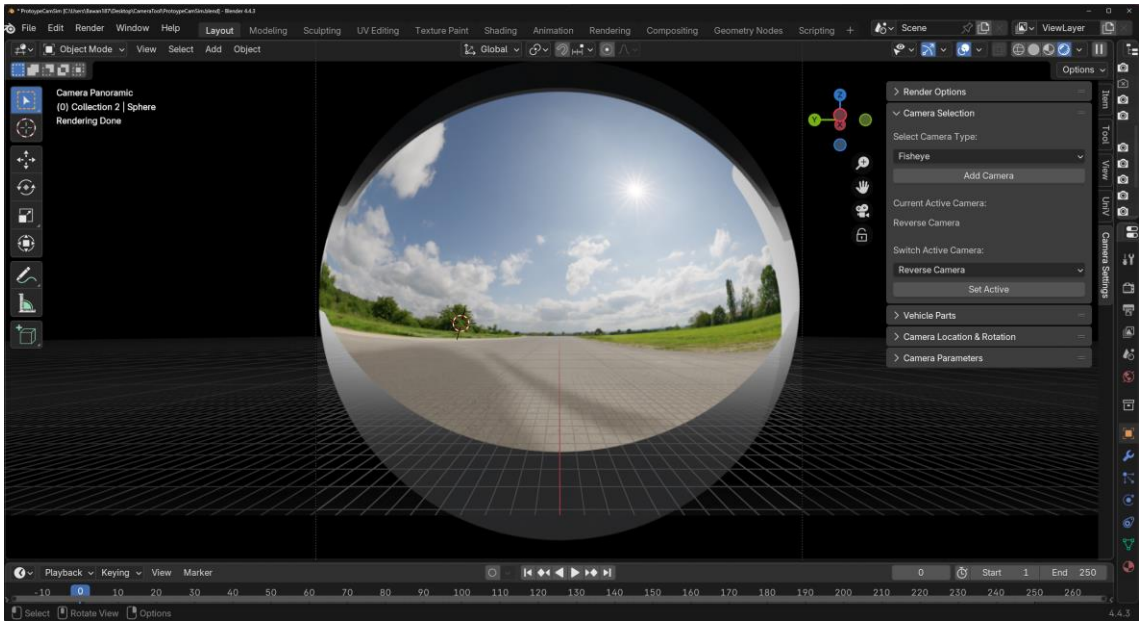


Figure 44: A camera with a fisheye lens mounted on the rear bumper.

Figure 44 shows a camera which has been mounted on the rear bumper of a test vehicle. Note that the vehicle in the figure is not an official Polestar vehicle, as this would violate the code of conduct. The camera provides an accurate image of the surroundings with a very wide field of view with minimal amounts of distortion.

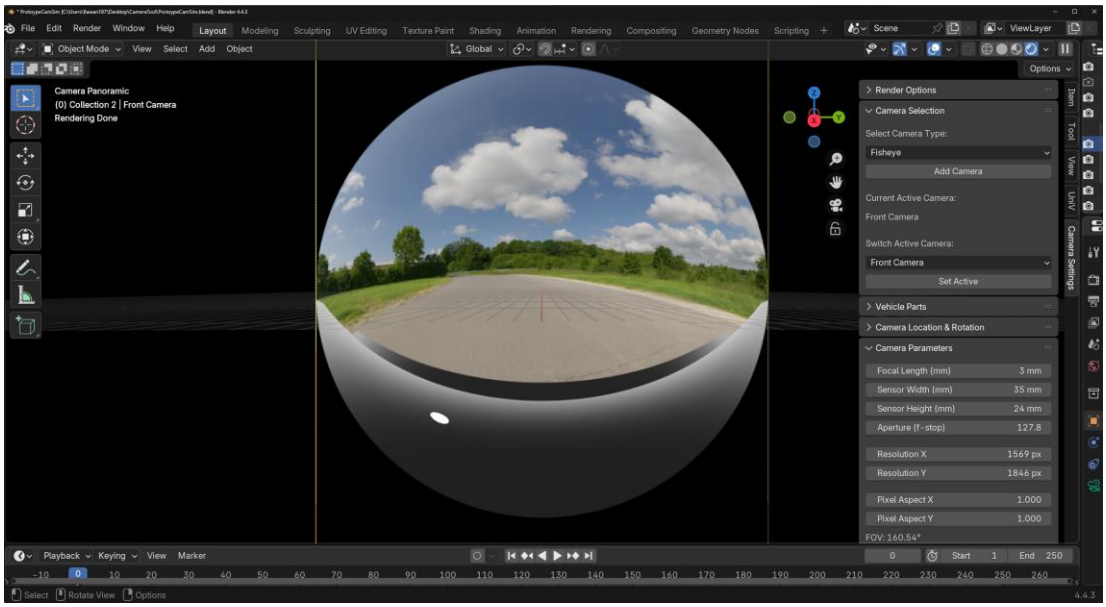


Figure 45: The camera view of a camera placed on the front bumper of the vehicle.

In figure 45, the camera is placed on the front bumper. Also providing a very wide field of view, along with minimal amounts of distortion. The panel in the right then allows the user to change specific parameters such as focal length, sensor width and height, pixel aspect and more.

Figure 46: Represents the post processed image that the driver sees on the infotainment screen.

In figure 46, the camera view is now set to the central cylindrical type. In the middle of the image, a test obstacle has been created to showcase how the camera would view a real object.

5 Conclusion

The aim of this chapter is to highlight some of the problems that were encountered along the way of the project and what could have been done differently.

5.1 Specific Problems

Although the project has been mostly successful, some problems have been encountered along the way. The first problem occurred in the code that controls the add-on tool. When creating the panels, we would have the code side by side next to the UI. Each time the code had been updated, the changes that were made did not appear in the panels.

After troubleshooting, the problem lied in that Blender is very meticulous when it comes to panel registration. The registration order needed to follow the structure of the code, otherwise, it could result in the program returning errors that are nonexistent. Hours that were then spent on troubleshooting could have been avoided if we were aware of this and spent on other parts of the project.

5.1.1 Further Developments

Because the time of the project had been further limited due to administrative issues, the functionality could have been further improved. One of the parts that could have been developed, given enough time, was regarding the inclusion of weather-based simulations. Because camera lenses are sensitive to external conditions such as rain, snow and fog, the possibility of simulating these conditions could have helped the team gather more data surrounding the cameras.

Moreover, some parts of the code behind the tool could have been more efficient, due to Blender having many ways of scripting, this was not obvious to us during the beginning of the project.

5.1.2 Timeline

The project was set to start in January; however, some issues have forced us to delay the project until April. After assessing the situation, we found that it was necessary to compromise on some parts of the tool. Our plan was then to build the foundation of the tool, then assess how much time that is left and try our best to adhere to the requirements.

6 References

[1] L. Flavell, *Beginning Blender : Open Source 3D Modeling, Animation, and Game Design*. Berkeley, Ca: Apress, 2010.
https://www.3ds.com/products/catia

[2] “Cameras — Blender Manual,” *docs.blender.org*. Available: https://docs.blender.org/manual/en/latest/render/cameras.html, [Online].

[3] S. Grotta, “Anatomy of a Digital Camera,” 2001. Available: http://www.blueopal.com/pdf/digital_camera_inside.pdf, [Online]

[4] E. Allen and S. Triantaphillidou, *Manual of Photography and Digital Imaging, 10th Edition*. Focal Press, 2012.

[5] Hans Petter Langtangen, *Python scripting for computational science*. Berlin: Springer, 2009.

[6] M. Lutz, *Learning Python*. Sebastopol, Ca: O’reilly, 2018.

[7] “Blender 2.92.0 Python API Documentation — Blender Python API,” *docs.blender.org*. https://docs.blender.org/api/current/index.html

[8] “CATIA,” *Dassault Systèmes*, Apr. 25, 2023. https://www.3ds.com/products/catia