The British School in the Netherlands - Senior School Voorschoten

PG3D: A 3D GRAPHICS ENGINE

Centre Number: 74373 Daniele Golzio Candidate Number: 8054

Contents

Part I

Analysis

Chapter 1

What is the problem?

Pygame is a popular 2D graphics library for Python, but its limitations in rendering 3D graphics make it unsuitable for developing modern, immersive applications. To overcome this challenge, my project aims to develop a new library that can leverage the capabilities of Pygame while offering full support for 3D graphics rendering. Achieving this goal will require addressing several technical challenges, such as developing advanced matrix functions, implementing a user-friendly interface, and packaging the library into a distributable format. Overall, this project represents an important contribution to the field of computer graphics and has the potential to empower Python developers to create sophisticated 3D applications with ease.

Chapter 2

Current System

Pygame is a robust 2D game engine that facilitates a seamless learning experience for beginners in creating an array of graphical elements, such as shapes, sprites, and their motion. Furthermore, the software provides advanced features that may pose a challenge to the less-experienced, but upon mastering them, they can elevate one's game development capabilities to a professional level. However, it is worth noting that Pygame lacks support for 3D graphics. Nevertheless, this shortcoming can be remedied by writing a bespoke library that employs Pygame as the underlying framework for rendering 2D graphics.

Utilizing the Pygame library, I have written a program that demonstrates the ease of use of the library's functions for rendering shapes. However, I have also noticed that there is a significant amount of setup code that needs to be written in order to get the library to work. This can be quite tedious for those who use the library frequently. Therefore, when designing my own library, I will strive to ensure that minimal to no setup is required.

As you can see in Figure 2.1, the top-left of the window is currently occupied by the Pygame logo with some text saying: pygame window. While this is a nice touch, I believe that the space can be put to better use. I plan to remove the logo and use the freed up space to add something more practical. This could be anything from a FPS counter to an interactive button. Whatever I decide to add, I'm sure it will make the window look even better!

```python
# importing the library
import pygame

# pygame setup
pygame.init()
screen = pygame.display.set_mode((600, 600))
clock = pygame.time.Clock()
running = True

# main game loop
while running:
```

```
    # checks if the user clicks the X to close the window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill("white") # fills the screen with a color to wipe anyhting from last from

    ### GAME LOOP ###

    pygame.draw.circle(screen, "green", (300, 300), 40)
    pygame.draw.rect(screen, "grey", (100, 100, 200, 100))

    #################

    pygame.display.flip() # displays anything

    clock.tick(60) # sets fps to 60

pygame.quit()
```

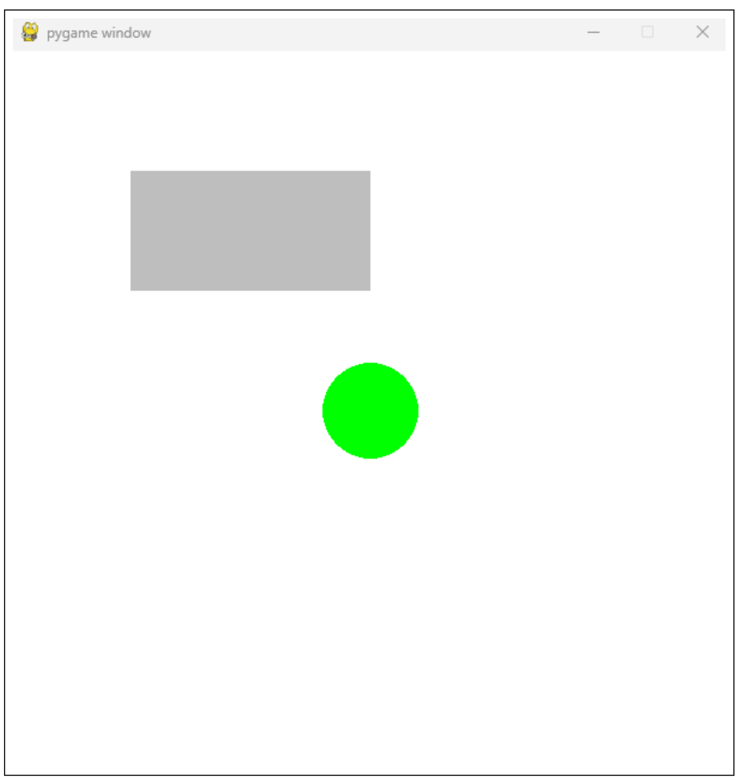


Figure 2.1: Pygame window

Chapter 3

Research

3.1 Perspective Projection Matrix

Rendering a 3D image on a 2D screen is thought to be a simple process. Most think that you can simply take the 3D coordinate and draw the x- and y-coordinate. In theory this works, but you don't get a realistic projection of the point. This is because the z coordinate isn't taken into account. this would mean that if two shapes are drawn but one is closer to the screen, they will both have the same size. In Figure 3.1 you can see that both of the triangles are the same size even though triangle 2 is further away with a z-coordinate of 10.

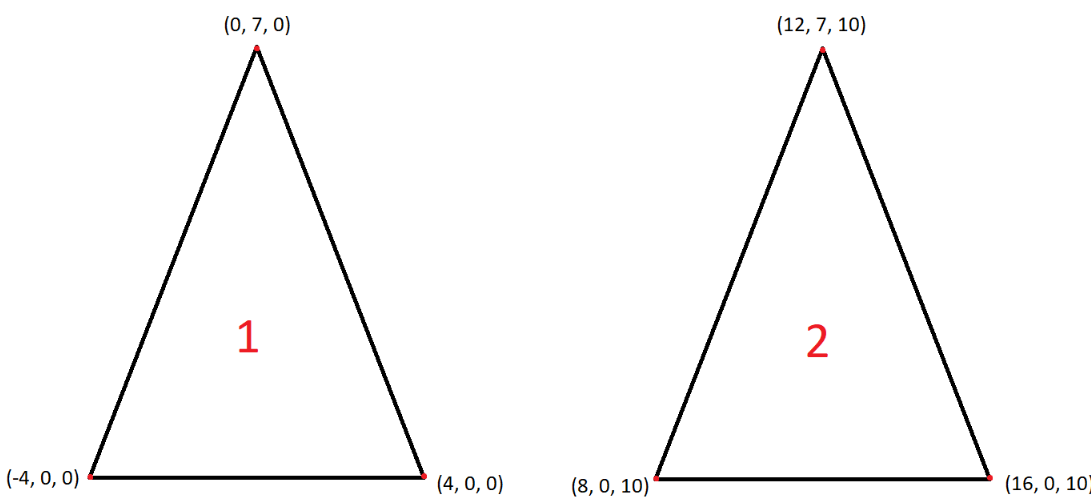


Figure 3.1: Orthographic projection

This type of projection is called **orthographic projection**. This type of projection is most commonly found in CAD software as it makes it easier for people to create technical drawings of 3D objects.

Orthographic projection can be implemented by simply multiplying the coordinate of the point with the following 3x3 matrix:

$$\begin{bmatrix} i' \\ j' \\ k' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ 0 \end{bmatrix}$$

This matrix multiplies the z-coordinate by 0 leaving you with the x- and y-coordinate. Similar matrices are used for rotation, scaling and transformations but, you are still left with an unrealistic projection.

This is where **perspective projection** comes in. As the name implies, this type of projection projects a point with perspective. This means that the further away a shape is from the screen, the smaller it appears. As you can see in Figure 3.2 the second triangle is further away with a z-coordinate of 10, therefore it appears to be smaller.

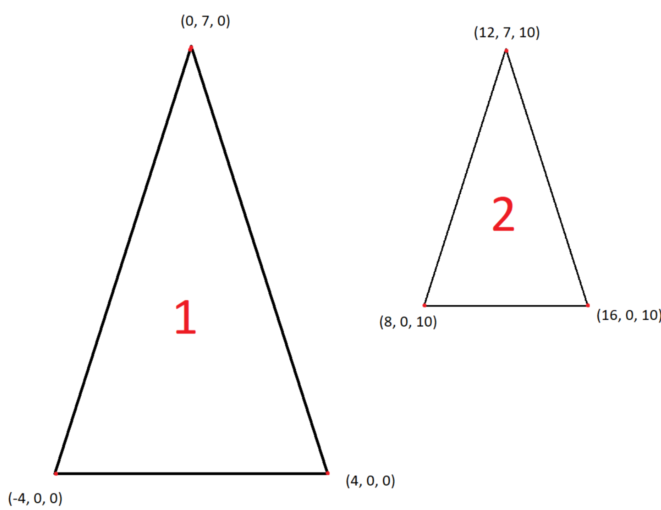


Figure 3.2: Perspective projection

This new projection gives the scene a more natural and realistic feel but this makes it much more complicated to do.

To project a 3D point to 2d space you can create a matrix similar to the one above.

The first step to creating this projection matrix is to normalize the screen space so that its maximum and minimum values lies between -1 and 1 As shown in Figure 3.3. This is done so that the projection will not warp based on the dimensions on the screen.

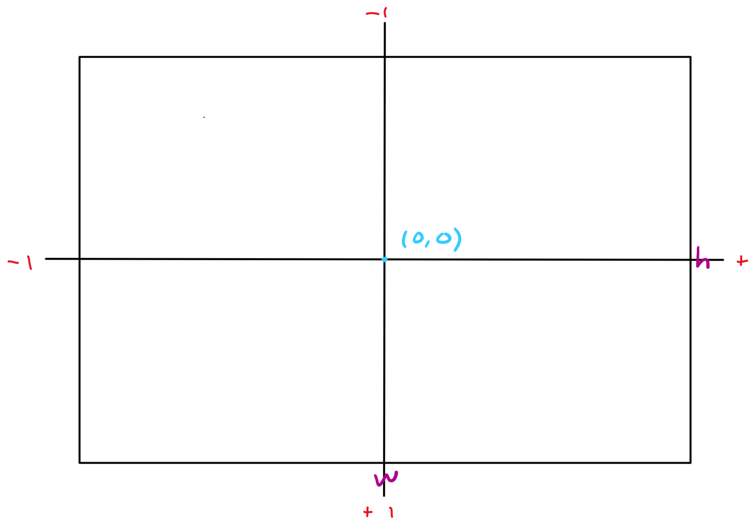


Figure 3.3: Screen space normalization

To normalize the point you have to multiply the x-coordinate by the aspect ratio where: $a = \frac{h}{w}$. When normalizing the screen space we get the following:

$$\begin{bmatrix} x & y & z \end{bmatrix} \longrightarrow \begin{bmatrix} (\frac{h}{w}) \cdot x & y & z \end{bmatrix}$$

Normalizing the screen space has an additional advantage that anything above +1 and below -1 will not be drawn to the screen. As you can see in Figure 3.4 the object on the right is not fully drawn because two of its vertices lie out side of the -1 to +1 range.

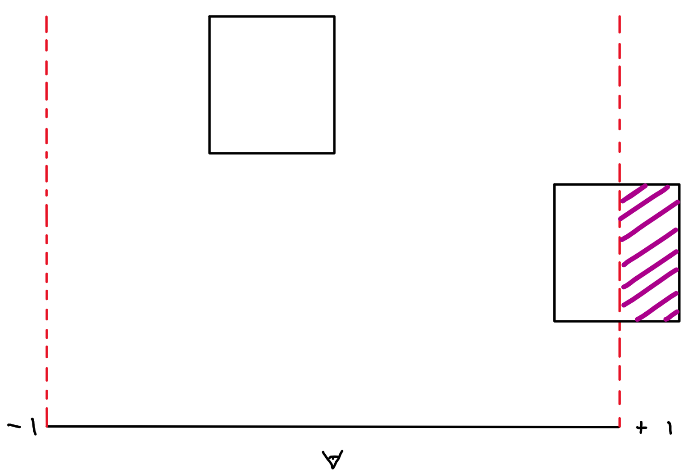


Figure 3.4: Normalization clipping

However, humans do not see screens in this manner. Instead humans see a field of view. As you can see in Figure 3.5, I have drawn two rays separated by an angle θ. To get the new coordinates of the point we will consider $\tan\frac{\theta}{2}$ because we know the length of the opposite and adjacent sides of the right angle triangle from the coordinates of the point. As you increase the field of view, you zoom out of the scene because you would see more of it; and if you decrease the field of view, you zoom into the scene because you see less of it. However, in our case, when θ increases, $\tan\frac{\theta}{2}$ also increases. This is a problem because as we increase the field of view, points will be displaced out of it and if we decrease the field of view, more objects will appear. To solve this problem, we do the inverse of this operation: $\frac{1}{\tan\frac{\theta}{2}}$

We can now add this step to our point by multiplying the x-coordinate by $\frac{1}{\tan\frac{\theta}{2}}$

$$\begin{bmatrix} x & y & z \end{bmatrix} \longrightarrow \begin{bmatrix} (\frac{h}{w}) \cdot (\frac{1}{\tan\frac{\theta}{2}}) \cdot x & (\frac{1}{\tan\frac{\theta}{2}}) \cdot y & z \end{bmatrix}$$

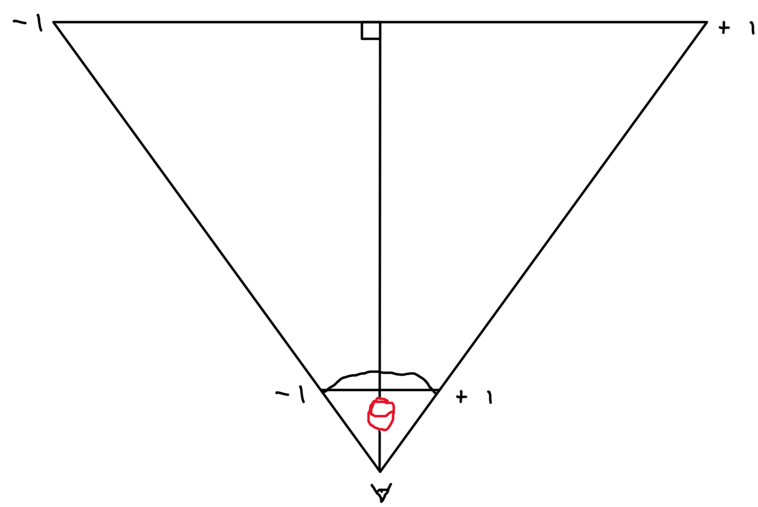


Figure 3.5: Field of view

Now that we have normalized the x- and y- coordinates, we can begin to normalize the z-coordinate. In Figure 3.6 you I have defined the furthest distance that the viewer can see as Z far (zf). Most would then think that the nearest distance that you can see is 0 but this assumes that the viewers eyes are resting on the screen. Since this isn't the case, there is a small distance between the viewers head and the screen called Z near (zn).

To work out where the position of a point in this frustum is we first need to scale it to a normalized system. The following equation accomplishes this $z' = \frac{z}{zf - zn}$. This gives us a point between 0 and 1 therefore we need to scale the point back up to fit the plane. Therefore, $z' = \frac{z \cdot zf}{zf - zn}$. However, this still leaves us with a discrepancy in the gap between the viewers head and the screen. Therefore we need to offset the transformed point by this discrepancy. This leaves us with the final equation to normalize the z-coordinate $z' = \frac{z \cdot zf}{zf - zn} - \frac{zf \cdot zn}{zf - zn}$

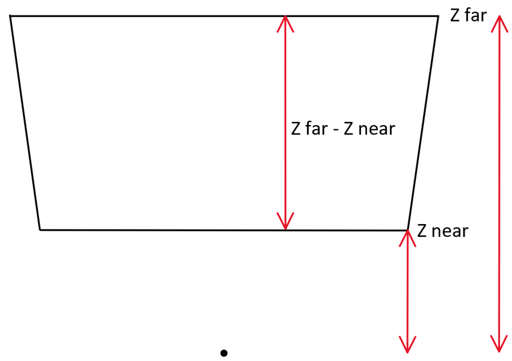


Figure 3.6: Z normalization

Putting this normalization in points coordinates leaves us with the following:

$$\begin{bmatrix} x & y & z \end{bmatrix} \longrightarrow \begin{bmatrix} (\frac{h}{w}) \cdot (\frac{1}{\tan\frac{\theta}{2}}) \cdot x & (\frac{1}{\tan\frac{\theta}{2}}) \cdot y & z \cdot (\frac{zf}{zf-zn}) - (\frac{zf \cdot zn}{zf-zn}) \end{bmatrix}$$

Now there is one last step required to fully transform our point. Intuitively we know that as something moves further away, they appear smaller. Therefore, we can do the following operations:

$$x' = \frac{x}{z}$$
$$y' = \frac{y}{z}$$

Now if we put everything together we get the following transformation:

$$\begin{bmatrix} x & y & z \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{(\frac{h}{w}) \cdot (\frac{1}{\tan\frac{\theta}{2}}) \cdot x}{z} & \frac{(\frac{1}{\tan\frac{\theta}{2}}) \cdot y}{z} & z \cdot (\frac{zf}{zf-zn}) - (\frac{zf \cdot zn}{zf-zn}) \end{bmatrix}$$

Although this transformation works, it looks quite messy therefore I will substitute some values with the following variables:

$$a = \frac{h}{w}$$
$$f = \frac{1}{\tan\frac{\theta}{2}}$$
$$g = \frac{zf}{zf - zn}$$

After substitution you get the following transformation:

$$\begin{bmatrix} x & y & z \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{afx}{z} & \frac{fy}{z} & z \cdot g - zn \cdot g \end{bmatrix}$$

Now that we have derived these three equations to transform the coordinates to 2D space we could directly implement them in the program but, a matrix could be created which would be directly multiplied to the coordinate.

In orthographic projection a simple 3x3 matrix is used to transform the 3D coordinate to 2D space but, for perspective projection a 3x3 matrix isn't large enough to allow for the z-coordinate to be subtracted by $zn \cdot g$. Secondly, if the multiplication were to be done with a 3x3 matrix, the division by z wouldn't be possible. The way to fix this is to use a 4x4 matrix. In this way, the $zn \cdot g$ could be put in the last row, and now we would be able to store the z-coordinate to then use it to divide the x- and y-coordinate in a second operation. Since we are now using a 4x4 matrix we must also use a put a fourth component, w, in our point so that matrix multiplication can be done. The w component has to be 1 or else the z-coordinate won't be copied into it.

The final perspective projection matrix looks like this:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \cdot \begin{bmatrix} (\frac{h}{w}) \cdot (\frac{1}{\tan\frac{\theta}{2}}) & 0 & 0 & 0 \\ 0 & (\frac{1}{\tan\frac{\theta}{2}}) & 0 & 0 \\ 0 & 0 & \frac{zf}{zf-zn} & 1 \\ 0 & 0 & \frac{-zf \cdot zn}{zf-zn} & 0 \end{bmatrix} = \begin{bmatrix} afx & afy & z \cdot g - zn \cdot g & z \end{bmatrix}$$

After performing this matrix multiplication, you need to divide the x-, y- and z-coordinates by the w component, giving us a coordinate in Cartesian space:

$$\begin{bmatrix} \frac{afx}{z} & \frac{fy}{z} & \frac{z \cdot g - zn \cdot g}{z} & z \end{bmatrix}$$

We can now take the x- and y-coordinates and directly plot them onto the screen.

3.2 Cameras

If you implement the perspective projection matrix described above, you will get an accurate representation of a 3D object in 2D space but, it would be much more immersive if you could look around the space in a first person view. This is why we need a virtual camera.

Camera movement can be thought of in two different ways. You either move the camera around the scene or you move the objects around the camera. My approach will use the second method.

Creating a camera is as simple as defining a couple variables such as its position and its orientation with the x, y, and z axis.

The position of the camera will be written using a simple 4D vector:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

The orientation of the camera will be stored in three vectors: forward, up and right. The forward vector is the cameras orientation along the z axis. The up vector is the cameras orientation along the y axis. And finally, the right vector is the cameras orientation along the x axis. These vectors are important for allowing the movement keys to move the camera forwards, upwards and side to side even when the camera has bee rotated.

To look around the scene you simply multiply the forward, up and right vectors by the rotation matrices for x and y axis rotation. To move around the scene, you simply you add the product of the axis and a velocity to the position of the camera.

The following matrix is multiplied to all the objects in the scene to translate them and rotate them to allow for camera control:

r = right, u = up, f = forward and C = camera

$$\begin{pmatrix} rx & ry & rz & 0 \\ ux & uy & uz & 0 \\ fx & fy & fz & 0 \\ Cx & Cy & Cz & 1 \end{pmatrix}^{-1}$$

The inverse is taken because when we move or rotate the camera in one direction or angle, all the objects will do the opposite.

This matrix will be multiplied to all the points before they are projected.

3.3 Matrix Multiplication

Now that we have gained an understanding of the mechanics behind perspective projection, our focus can now shift to the multiplication of matrices. To expand upon the concept of matrix multiplication, the Pearson AS level Further maths pure book [2] will serve as a valuable resource, given its comprehensive coverage of the topic. As illustrated in Figure 3.7, it is imperative that the number of columns in the first matrix match the number of rows in the second matrix. In our current scenario, employing a 4x4 matrix allows us the option to utilize either a 4x1 or 1x4 matrix for our point. While either may be used, it is vital that we remain consistent to avoid the potential for errors. I shall opt for a 4x1 matrix to represent the coordinate, thereby placing it on the left side during multiplication. Furthermore, the utilization of a 4x4 matrix ensures that the output will also be a 4x1 matrix, removing the need for transposition.

6.2 Matrix multiplication

Two matrices can be multiplied together. Unlike the operations we have seen so far, this is completely different from normal arithmetic multiplication.

- **Matrices can be multiplied together if the number of columns in the first matrix is equal to the number of rows in the second matrix.**

Notation If **AB** exists, then matrix **A** is said to be **multiplicatively conformable** with matrix **B**.

The **product matrix** will have the same number of rows as the first matrix, and the same number of columns as the second matrix.

AB = C — If **A** has size $n \times m$ and **B** has size $m \times k$ then the product matrix, **C**, has size $n \times k$.

The **order** in which you multiply matrices is important. This has two consequences:

- In general **AB** ≠ **BA** (even if **A** and **B** are both square matrices).
- If **AB** exists, **BA** does not necessarily exist (for example if **A** is a 3 × 2 matrix and **B** is a 2 × 4 matrix).

- **To find the product of two multiplicatively conformable matrices, you multiply the elements in each row in the left-hand matrix by the corresponding elements in each column in the right-hand matrix, then add the results together.**

$$\begin{pmatrix} 5 & -1 & 2 \\ 8 & 3 & -4 \end{pmatrix} \times \begin{pmatrix} 2 & 2 \\ 9 & -3 \\ 7 & 4 \end{pmatrix} = \begin{pmatrix} 15 & 21 \\ 15 & -9 \end{pmatrix}$$

You are multiplying a 2 × 3 matrix by a 3 × 2 matrix, so the product matrix will have size 2 × 2. To find the bottom left element, work out $8 \times 2 + 3 \times 9 + (-4) \times 7 = 16 + 27 - 28 = 15$

Figure 3.7: Matrix Multiplication

Example 5

Given that $\mathbf{A} = \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix} -3 \\ 2 \end{pmatrix}$

a find **AB** **b** explain why it is not possible to find **BA**.

a First calculate the size of **AB**.

$(2 \times 2) \times (2 \times 1)$ gives 2×1

The number of rows is two from here.

The number of columns is one from here.

$\mathbf{AB} = \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} -3 \\ 2 \end{pmatrix} = \begin{pmatrix} p \\ q \end{pmatrix}$

$p = 1 \times (-3) + (-2) \times 2 = -7$ — The top number is the total of the first row of **A** multiplied by the first column of **B**.

$q = 3 \times (-3) + 4 \times 2 = -1$ — The bottom number is the total of the second row of **A** multiplied by the first column of **B**.

So $\mathbf{AB} = \begin{pmatrix} -7 \\ -1 \end{pmatrix}$

b **BA** cannot be found, since the number of columns in **B** is not the same as the number of rows in **A**.

Watch out Remember that order is important. **B** is not multiplicatively conformable with **A**, but **A** is multiplicatively conformable with **B**.

Figure 3.8: Matrix Multiplication Example

Chapter 4

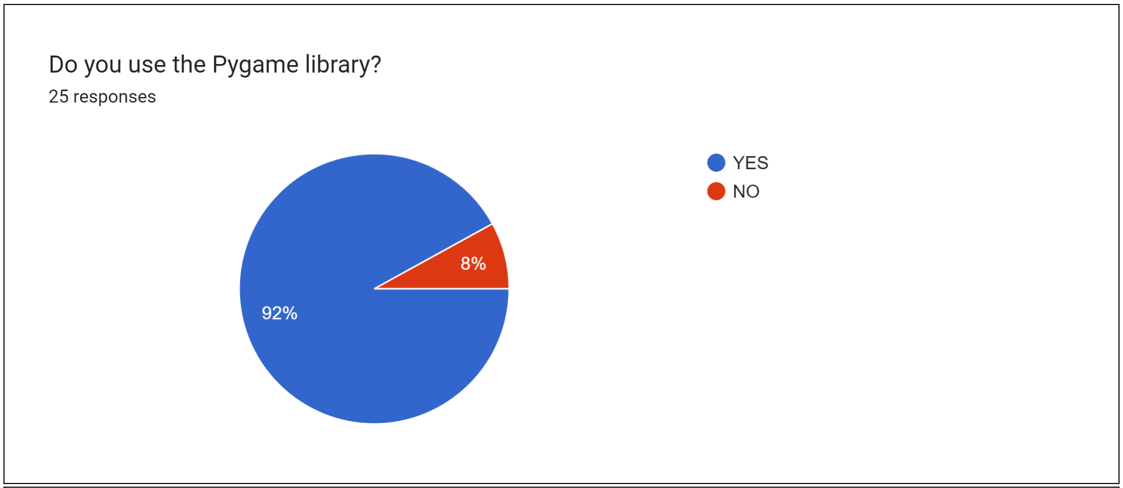
End-Users

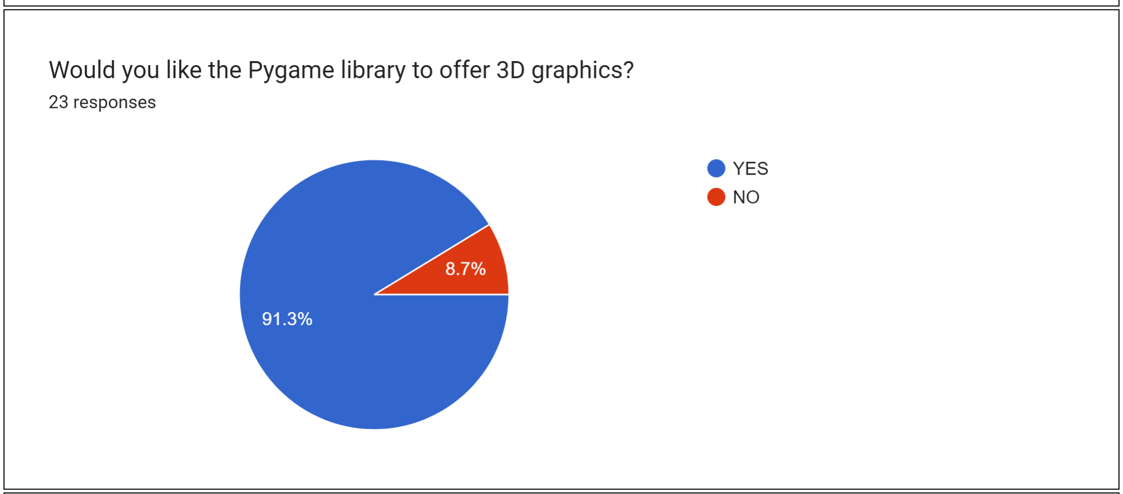
The intended audience for my library comprises individuals utilizing the Pygame library who aspire to augment their programming capabilities through the incorporation of 3D graphics. My library aims to alleviate any concerns regarding the intricacies of 3D graphics, thus enabling users to concentrate on their programming tasks. Users can effortlessly integrate 3D graphics into their programs by leveraging the library's pre-built functions within their code. In order to ensure the library aligns with the needs of the target audience, I have identified an appropriate end-user to evaluate the library and provide constructive feedback on areas for improvement, ultimately enabling me to refine the library to better meet the expectations of the target audience.

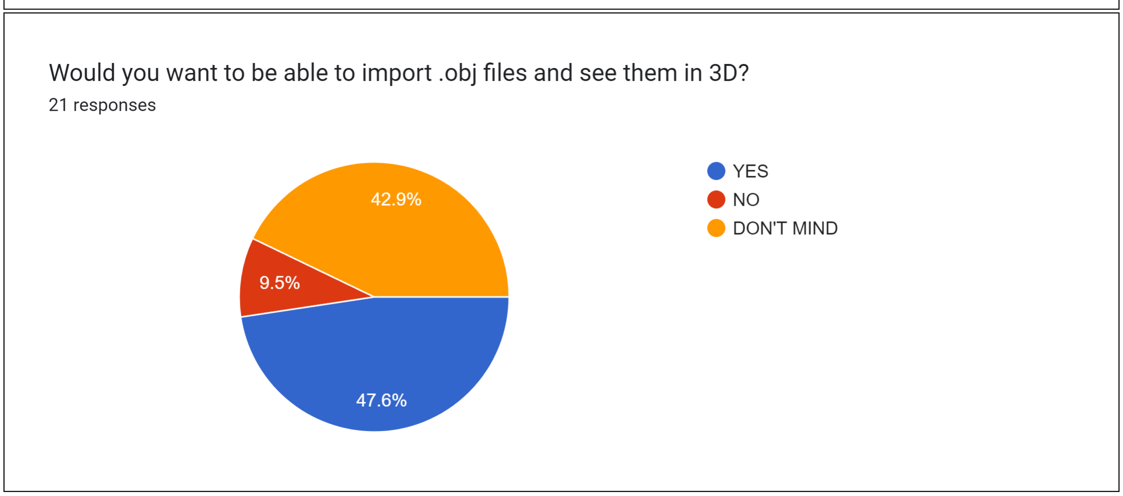
The individual I have selected as the end-user is Mr.S, a high school student who is currently undertaking his A Levels in Further Mathematics, Physics, and Computer Science. As per Mr.S's remarks, he has been "using the Pygame library extensively" [3]. However, he also pointed out that "the Pygame library's functionality is currently limited due to its inability to support 3D graphics" [3]. Furthermore, Mr.S emphasized that functions such as "ready to use 3D shapes and movement controls" [3] are fundamental for any graphics library, as acquiring the knowledge necessary to construct them independently would be excessively time-consuming. Regarding the lack of lighting effects, Mr.S commented that it "may not pose a significant issue since this is merely a basic Pygame extension" [3]. He also supported this statement by mentioning that "since Python is not ideal for quick calculations, implementing lighting effects may result in a significant reduction in the program's performance" [3]. Mr.S emphasized that ".obj files are an essential part of 3D graphics"[3], as they are a "key component in creating realistic visuals"[3]. He explained that this is why libraries must support them, as they are necessary for creating high-quality 3D graphics. Furthermore, he noted that this feature is becoming increasingly important as 3D graphics become more popular.

To further inform my approach to solving the problem, I developed an online questionnaire and distributed it among members of an online programming forum to solicit the perspectives of individuals who might have an interest in the matter at hand.

The questionnaire consisted of the following inquiries and responses:

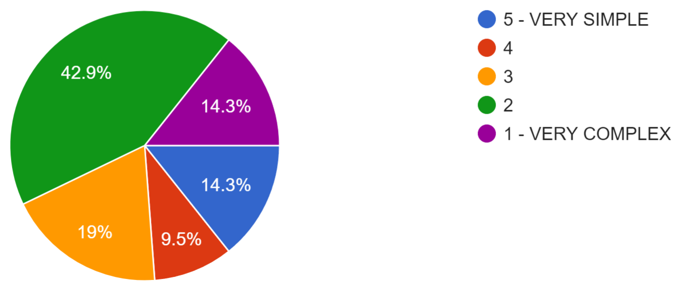






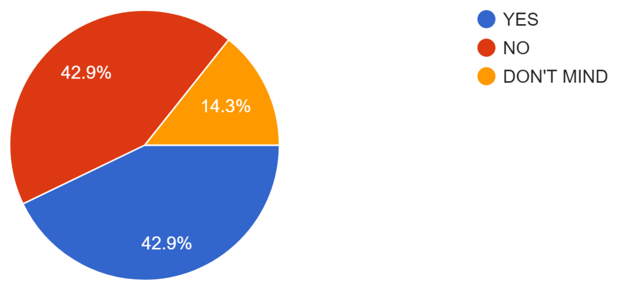
How simple should the library be to use? (5 = very simple, 1 = very complex)

21 responses



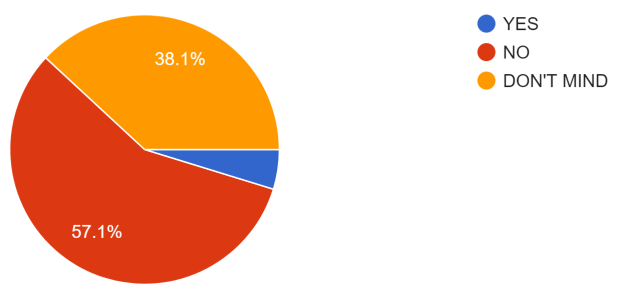
Should the library offer ready to use 3D shapes?

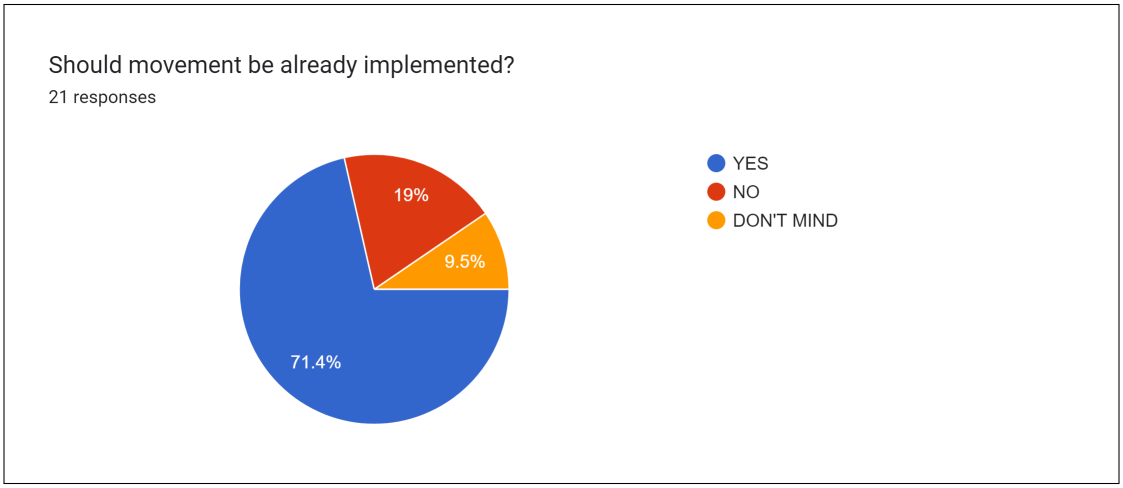
21 responses



Should the library include lighting effects?

21 responses





Based on the results of the questionnaire, it was determined that the target audience would prefer a library that is of moderate difficulty to use since `42.9%` respondents voted for the second highest tier of complexity. This approach would enable users to exercise greater autonomy with the available functions, albeit at the cost of requiring a longer learning curve. There was a split decision in response to the query concerning the library's provision of pre-made shapes, as `42.9%` of the respondents voted in favor of the proposition, and an equal percentage were against it. This result may suggest that certain individuals are reluctant to utilize three-dimensional shapes due to the complexities associated with their full customization, such as size and orientation. Conversely, those who favored this option sought to access these shapes for time-saving purposes. Additionally, a majority of `71.4%` respondents favored a pre-existing movement system. A majority of `57.1%` of the respondents voted against the inclusion of lightning effects in the library. This outcome can be attributed to the fact that Python is not a high-speed language, and the additional calculations required for such effects are likely to significantly impede the performance of the user's program. Moreover, a significant proportion of `47.6%` of the total votes expressed a favorable stance towards the incorporation of the functionality to import .obj files. Such a result is unsurprising considering that this feature is already a prevalent component of several 3D graphics libraries. Additionally, this feature offers users the convenience of avoiding the need to construct their 3D scenes using pre-built shapes provided by the libraries. Ultimately, a significant majority of `71.4%` of respondents expressed a preference for the inclusion of pre-existing movement controls within the library. This result aligns with the prevailing usage of pygame as a game engine, thus rationalizing the expectation for movement controls to be pre-implemented.

After conducting an interview and a questionnaire, I have deducted the following user needs:

- The library must be able to project a 3D point onto a 2D screen.
- The library must incorporate movement controls to enable users to navigate the scene.
- The library must provide pre-built shapes that users can readily include.

- The library should allow users to designate a file path for a .obj format file and visualize its rendering on the screen.
- The library must possess a high degree of complexity to grant users extensive freedom in their programming endeavors.
- The library must come with documentation which explain the objects and methods that are in it.

Chapter 5

Existing Systems

5.1 Ursina Engine

The Ursina engine is a Python-based software development tool that serves as a wrapper around the Panda3D game engine, enabling users to conveniently create applications with 3D graphics. The library contains an extensive collection of models, textures, and shaders, which provide a broad scope for designing customized shapes. Nevertheless, the engine encompasses numerous functions, modules, classes, and parameters that require mastery before effective utilization of the tool. Additionally, the Ursina engine incorporates an inbuilt user interface that offers users the option to navigate to the API reference, reload the model, reload the texture, or reload the code. Furthermore, the engine presents the frames per second information on the top-right corner of the screen.

Ursina engine is a library which can create basic 3D programs with very few lines of code. I wrote the following program which renders a red cuboid on the screen after reading Ursina's documentation [1].

```
# import library
from ursina import *

# create an instance of the engine
app = Ursina()

# create an instance of entity class and specify shape
box = Entity(model='cube', color=color.red)

# function that calls itself every  frame
def update():
    # changes box position based on key input
    box.x -= held_keys['d'] * time.dt
    box.x += held_keys['a'] * time.dt
```

```
    box.z -= held_keys['w'] * time.dt
    box.z += held_keys['s'] * time.dt

    # rotates box every frame
    theta = 100 * time.dt
    box.rotation_z += theta
    box.rotation_x += theta

# calls the ursina run function which runs the program
app.run()
```

As you can see in Figure 5.1, the program successfully renders a spinning red cube. The positives of Ursina engine are that there is minimal library configuration that needs to be done unlike Pygame. Furthermore, creating shapes and moving around the scene is very easy but, there are many parameters that the user can tinker with to make it their own. The one downside to this library is that movement is not built into the engine and a user has to program it themselves.

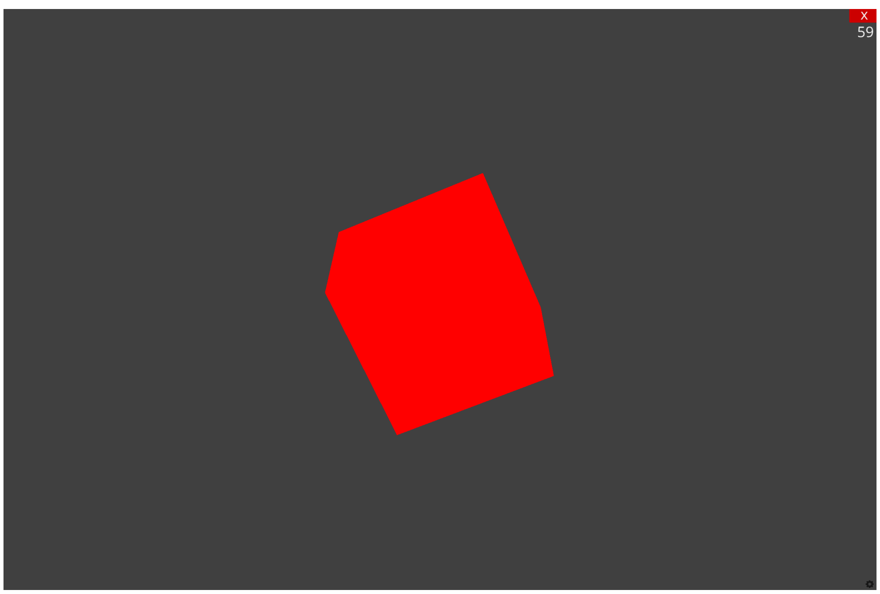


Figure 5.1: Ursina engine window

5.2 Panda3D

positives: quick because its written in c++, commands to load models is easy, a lot of setup, a lot of customisation negatives: user has to make their own class to simply get a window to show, module names aren't obvious and simple from beginners.

Panda3D is a 3D game engine written in C++. Since its written in C++, it is a very quick library meaning that it is extremely well optimized library. The libraries intended language is Python meaning that it has all the benefits that you get from an interpreted language, plus it has the speed of C++.

```
from direct.showbase.ShowBase import ShowBase

class MyApp(ShowBase):
    def __init__(self):
        ShowBase.__init__(self)

        # Load the environment model.
        self.scene = self.loader.loadModel("models/environment")
        # Reparent the model to render.
        self.scene.reparentTo(self.render)
        # Apply scale and position transforms on the model.
        self.scene.setScale(0.25, 0.25, 0.25)
        self.scene.setPos(-8, 42, 0)

app = MyApp()
app.run()
```

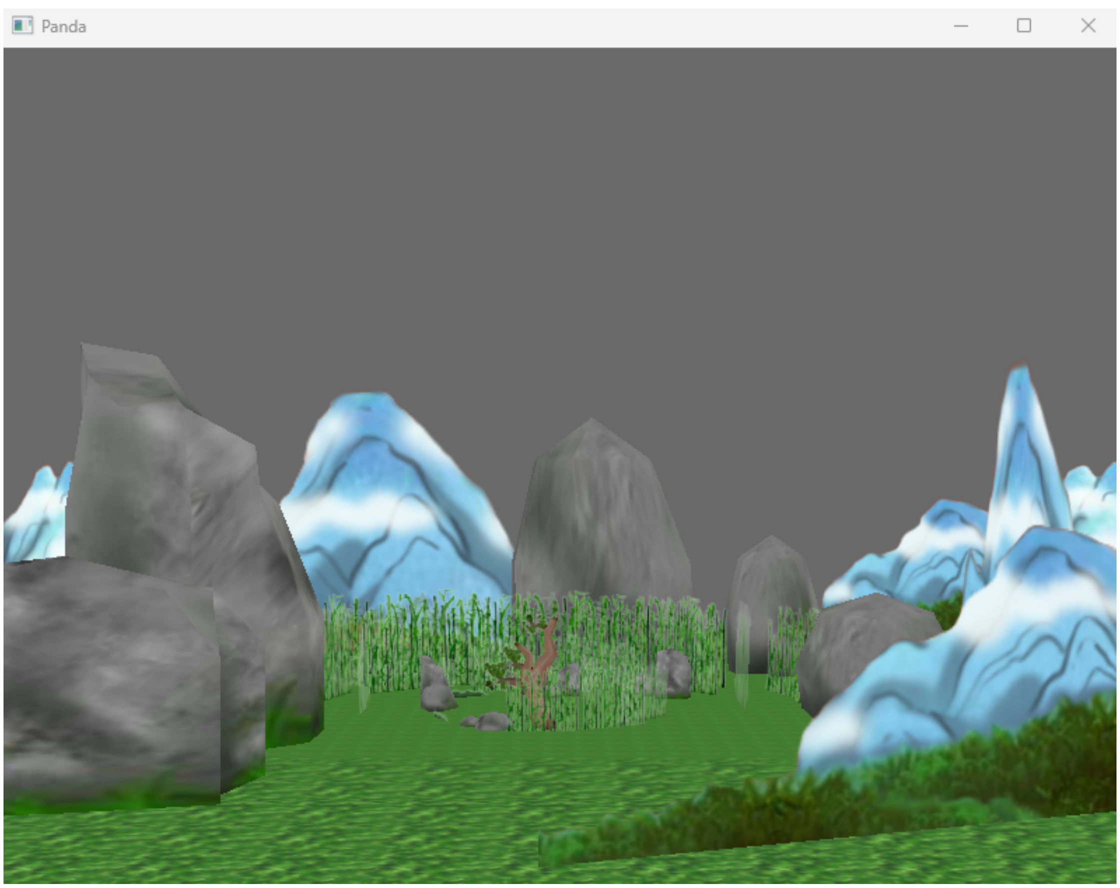


Figure 5.2: Panda3D window

Chapter 6

Project Objectives

1. The program should allow the user to draw a 3D point.
   (a) The user should be able to instantiate a 'Point' class
   (b) The user should be able to specify the coordinates of the point as the parameters of the point object.
2. The user should be able to move around the scene.
   (a) The user should be able to use keyboard input to move around the scene.
   (b) The user should be able to use the W, A, S, D, E, Q keys to move around the scene.
      i. W should move the camera forwards
      ii. A should move the camera to the left
      iii. S should move the camera backwards
      iv. D should move the camera to the right
      v. E should move the camera upwards
      vi. Q should move the camera downwards
   (c) The user should be able to use their mouse to look around the scene.
3. The user should be able to use ready made 3D shapes by using individual classes.
   (a) The library should have a general 'Shape' class that is inherited by all the different shapes.
   (b) The user should be able to specify various parameters such as: size and position of the shape.
   (c) The class should have a cube class which inherits the shape class.
   (d) The class should have a pyramid class which inherits the shape class.
   (e) The class should have a tetrahedron class which inherits the shape class.

4. The user should be able to use the scroll wheel to change FOV.
5. The user should be able to change the window size while the program is running.
6. The user should be able to import a .obj file.
    (a) The user should be able to see the .obj file render in their scene.
    (b) There should be an 'Mesh' class where the user can input the .obj file path as a string in the parameters.
    (c) The class should then read the file and create the necessary triangles to draw on the screen.
7. The library should have a documentation file containing a description of all the classes and their methods.
    (a) This file should describe what all the methods, attributes and parameters of each class do.
8. The top part of the window of the program should display the Frames per Second (FPS).
9. The library should have minimal to no setup to initialize the library.
10. The user should have the option to see certain statistics displayed on the screen to see what is going on in the background.
    (a) The user should be able to turn on this option by pressing the "o" key.
    (b) The user should be able to see the FPS.
    (c) The user should be able to see the dimensions of the screen.
    (d) The user should be able to see the FOV of the camera.
11. The user should be able to press the escape key to close the window.
12. When the user specifies a background color or line color, the one that hasn't been specified should be the opposite color so that the colors don't collide.
13. The color of the text for the optional statistics should be the opposite of the background color so that it is always visible.

Chapter 7

General solution

The library will have a main loop which will project and draw all the points at a fixed number of times per second. The loop is depicted in the following flowchart:

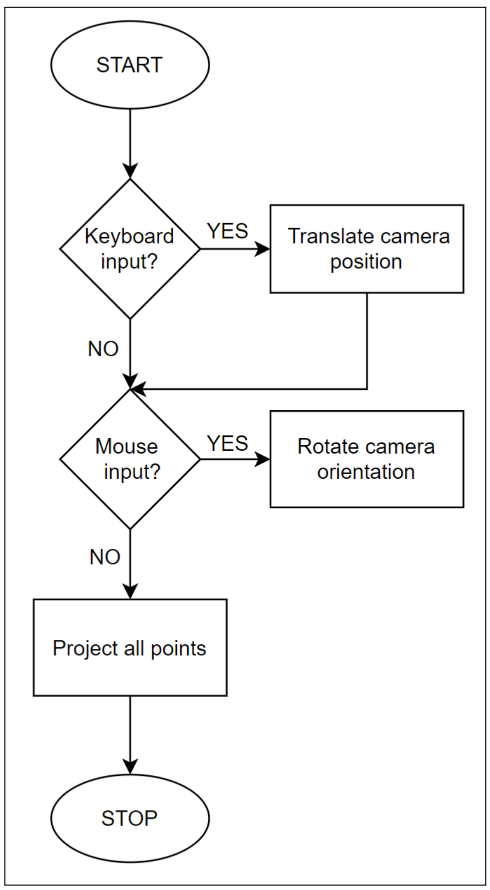


Figure 7.1: Flowchart of main loop

The user will be able to access two main data structures to create their shapes: point and triangle. A point is a 3D coordinate and a triangle is a group of three points. With these two data structures, an shape can be made. The user will also have access to some simple ready made shapes such as cubes, tetrahedrons, pyramids and various prisms. Finally, the user will be able to input a .obj file which is a common way to store 3D models. These objects will look like this:

```
# import the library
from pg3d import *

app = pg3d.App((500, 500))

a = pg3d.Point([4, 5, 2], app)
b = pg3d.Point([1, 3, 8], app)
c = pg3d.Point([7, 2, 3], app)

t = pg3d.Triange(a, b, c, app)

box = pg3d.Shape("Cube", center=[0, 0, 0], size=10, orientation=[0, 0, 0])
pyramid = pg3d.Shape("Cube", center=[8, 0, 12], size=50, orientation=[32, 7, 0])

model = pg3d.Model("models/house.obj", app)

if __name__ == "__main__":
    app.run()
```

As you can see in the code above, I create an instance of the App class called ”app”. This is where the main ”run” method is Figure 7.1 and is also where all the shapes that the user has created are stored. In each shape/model/point/triangle, the last parameter is reserved for the instance of the App class, this is what allows the points to be stored in one location.

Part II

Design

Chapter 8

Development environment

8.1 Python

As the objective of this project is to enhance the Pygame library, it is imperative to utilize Python as the programming language. Python's interpreted nature enables efficient debugging and facilitates making changes to the code in a more expeditious manner since there is no need for repeated compilation during run-time. Additionally, Python offers a wide range of fundamental yet advantageous data structures, which will prove indispensable when constructing the matrix data structure. Additionally, Python has a large and active community of users and developers, which makes it easy to find support and resources for creating a 3D graphics library.

8.2 Visual Studio Code

My primary IDE of choice is Visual Studio Code due to its ability to facilitate the development of programs that require the use of multiple files. This is attributed to its support of simultaneous opening and manipulation of multiple files. Given the extensive number of files that I will be working with in the creation of the library, Visual Studio Code's multi-file handling capability is particularly desirable.

Moreover, the IDE offers a range of built-in tools that streamline code debugging, thereby enhancing the development process. Additionally, Visual Studio Code features numerous community-written extensions that can enhance the coding experience. Specifically, I will use Pylance for Python-specific debugging, isort for import sorting, and Rainbow Brackets for better visualization of bracket ordering.

Lastly, Visual Studio Code offers built-in version control functionality, which enables me to update the code on different computers using Git. This feature provides an added layer of convenience and ensures that the code remains updated across multiple platforms.

8.3 Pygame

The exclusive library to be employed for the project at hand is Pygame, with the objective of imparting 3D capabilities to the software. Pygame offers the benefit of abstracting the generation of 2D graphics, which consequently saves considerable time and facilitates the undivided concentration on the development of the program's 3D element.

Pygame will be used to create a display and draw circles and polygons onto it. The circles will be drawn at the vertices of the points, and the polygons will be used to draw triangles.

The function calls look like this:

```python
import pygame

# Command to create a screen
surface = pygame.display.set_mode(size, flags, depth, display, vsync
    )

# Command to draw a circle
circle = pygame.draw.circle(surface, color, center, radius)

# Command to draw a polygon
polygon = pygame.draw.polygon(surface, color, points, width)
```

Chapter 9

IPSO diagram

When the user wants to input a model and see it rendered, they input the file path as a parameter of the class as a string. The class then calls a function which reads the file and generates all the points and triangles to render the model on the screen.

Inputs	**Processes**
File path	Function that generates points and triangles
Storage	**Outputs**
List	Model drawn on screen

Chapter 10

Data storage

The user will be able to generate as many shapes, points, triangles and models as they wish. Since they will all need to be rotated, translated, and projected it is useful to store them in a single variable so that when it comes to projecting these points, you can simply loop through the list and project them one by one. Therefore, I will store all these data structures in a single list called "mesh". This will be a 1 dimensional list which will only contain triangles and points. The triangles and points will be appended to the list in their "`__init__`" methods. The loop will then check if the current shape is a Point or Triangle object and will perform the necessary operations to project and draw them.

Chapter 11

Class diagrams

The following diagrams show all the attributes and methods which will be needed for each class. These diagrams contain all the data types of the attributes and the data types for the data that the methods take as parameters and the data types for the data that they return.

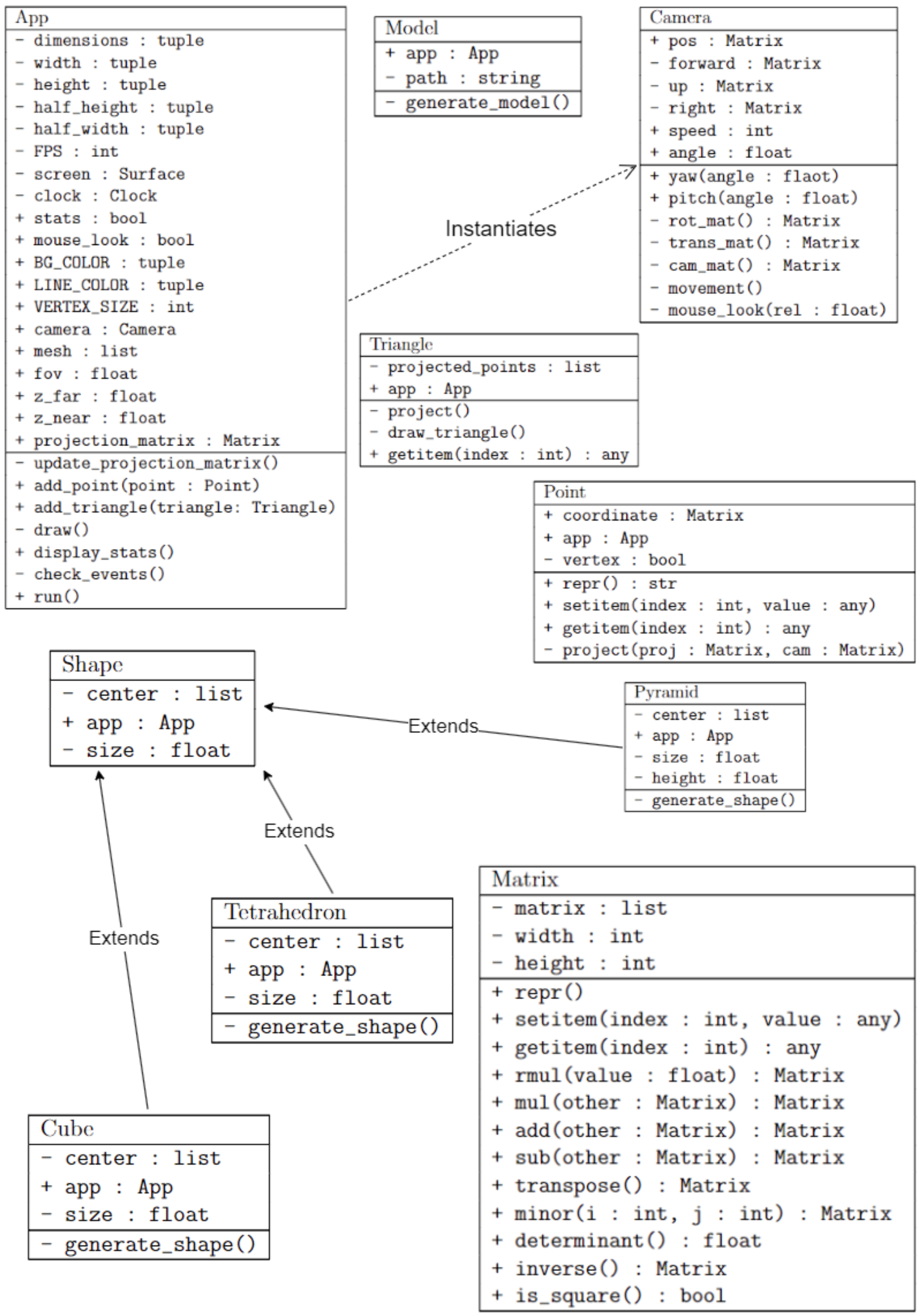


Figure 11.1: Panda3D window

Chapter 12

Class description

Class	Attribute	Access Type	Default Values	Description
App	dimensions	private	(1000, 700)	The dimensions of the screen
	width	private	1000	The width of the screen
	height	private	700	The height of the screen
	`half_width`	private	---	The half width of the screen
	`half_height`	private	---	The half height of the screen
	FPS	private	60	The number of times the program loop runs per second
	screen	private	Surface	The window where the scene will be rendered
	clock	private	Clock	The clock that will tick the number of FPS
	stats	public	False	Flag that allows for stats to be displayed on the screen
	`mouse_look`	public	False	Flag that dictates whether mouse input can be used to look around the scene
	`BG_COLOR`	public	(0, 0, 0)	Screen background color
	`LINE_COLOR`	public	(255, 255, 255)	Color used to drawn vertices and triangles
	`VERTEX_SIZE`	public	2	The size of the drawn points
	camera	public	Camera	Camera which is used to look and move around the scene
	mesh	public	---	List that stores all points and triangles
	fov	public	90	The field of view of the camera
	`z_far`	public	1000	Distance to far plane
	`z_near`	public	0.1	Distance to near plane

	`projection_matrix`	public	Matrix	Matrix used to project points on screen
	`check_stats`	private	0	Variable which is used to determine the previous state of the optional statistics button
Camera	pos forward	private private	--- [0, 0, 1]	Starting position of camera Camera orientation along z-axis
	up	private	[0, 1, 0]	Camera orientation along y-axis
	right	private	[1, 0, 0]	Camera orientation along x-axis
	speed	public	1	Movement speed
	angle	public	1	Rotation speed
Model	app	public	App	App object used to create the triangles and points
	path	private	---	Path of model that will be drawn
Point	coordinate	public	---	The coordinate of the point
	app	public	App	App object used to add the point to mesh
	vertex	private	True	Used to determine whether the point will be drawn
Shape	center	private	[0, 0, 0]	The center coordinate of the shape that is to be drawn
	size	private	1	Size of the shape that is to be drawn
	app	public	App	App object used to create the triangles and points
Cube	center	private	[0, 0, 0]	The center coordinate of the shape that is to be drawn
	size	private	1	Size of the shape that is to be drawn
	app	public	App	App object used to create the triangles and points
Pyramid	center	private	[0, 0, 0]	The center coordinate of the shape that is to be drawn
	size	private	1	Size of the shape that is to be drawn
	app	public	App	App object used to create the triangles and points
Tetrahedron	center	private	[0, 0, 0]	The center coordinate of the shape that is to be drawn

	size	private	1	Size of the shape that is to be drawn
	app	public	App	App object used to create the triangles and points
Triangle	`projected_points`	private	---	Used to store the projected point for clipping
	app	public	App	App object used to add the triangle to mesh
Matrix	matrix	private	---	The actual matrix in list for
	width	private	---	Number of columns
	height	private	---	Number of rows

Class	**Access Type and Method**	**Parameters**	**Return Values**	**Description**
App	- `update_projection_matrix`()	---	---	Re-defines the projection matrix if parameters are changed
	+ `add_point`()	point	---	Adds a point to mesh
	+ `add_triangle`()	triangle	---	Adds a triangle to mesh
	- draw()	---	---	Draws all the points and triangles on the screen
	+ `display_stats`()	---	---	Displays all optional stats on screen
	- `check_events`()	---	---	Checks for mouse and keyboard input
	+ run()	---	---	Runs main program loop
Camera	+ yaw()	angle	---	Rotates camera along y-axis
	+ pitch()	angle	---	Rotates camera along x-axis
	- `rot_mat`()	---	Rotation matrix	Creates camera rotation matrix
	- `trans_mat`()	---	Translation matrix	Creates translation matrix

	- `cam_mat`()	---	Camera matrix	Multiplies translation and rotation matrix
	- movement()	---	---	Checks for movement keys
	- `mouse_look`()	rel	---	Checks for mouse movement
Model	- `generate_model`()	---	---	Generates points and triangles for model
Point	+ repr()	---	coordinate[0]	Defines behaviour of printing a point
	+ setitem()	index, value	---	Defines behaviour of setting indexed point to a value
	+ getitem()	index	coordinate[index]	Defines behaviour of getting item of indexed point
	- project()	proj, cam	Coordinate or None	Projects a 3D point to 2D space
Cube	- `generate_shape`()	---	---	Generates points and triangles for Cube
Pyramid	- `generate_shape`()	---	---	Generates points and triangles for Pyramid
Tetrahedron	- `generate_shape`()	---	---	Generates points and triangles for Tetrahedron
Triangle	- project()	---	---	Projects the vertices of the triangle
	- `draw_triangle`()	---	---	Draws the triangle

	+ getitem()	index	points[index]	Defines behaviour of getting item of indexed triangle
Matrix	+ repr()	---	---	Defines behaviour of printing a matrix
	+ setitem()	index, value	---	Defines behaviour of setting indexed matrix to a value
	+ getitem()	index	matrix[index]	Defines behaviour of getting item of indexed matrix
	+ rmul()	value	Matrix	Defines behaviour of multiplying a matrix with a number
	+ mul()	other	Matrix	Defines behaviour of multiplying a matrix with another matrix
	+ add()	other	Matrix	Defines the behaviour of adding two matrices together
	+ sub()	other	Matrix	Defines the behaviour of subtracting two matrices together
	+ transpose()	---	Matrix	Defines the behaviour of transposing a matrix
	+ minor()	i, j	Matrix	Finds the minor of a matrix

	+ determinant()	`---`	float	Finds the determinant of a matrix
	+ inverse()	`---`	Matrix	Finds the inverse of a matrix
	+ `is_square`()	`---`	bool	Sees if a matrix is square

Chapter 13

Algorithms

13.1 Matrix Multiplication

This algorithm will find the product of two matrices, A and B, where A has the same number of columns as the number of rows of B. The algorithm will consists of a nested loop which loops over the rows of A and the columns of B. Then, there will be a third loop which loops over the columns of A. Lastly, the product of the two numbers of the two matrices is found by using the indexes of the loops.

In pseudo code the algorithm for matrix multiplication looks like this:

```
A is a matrix
B is another matrix
Result is an empty matrix with size A.rows and B.columns

if A.columns = B.rows
    for i from 0 to A.rows
        for j from 0 to B.columns
            sum = 0
            for k from 0 to A.columns
                sum += A[i][k] * B[k][j]
            Result[i][j] = sum
```

13.2 Matrix addition and subtraction

Matrix addition and subtraction are two very similar algorithms which first check that both matrices are the same size, then a nested loop loops through all the positions of the matrices. Finally at the same position in the result matrix, the addition or subtraction of the two matrices is inserted.

In pseudo code the algorithm for matrix addition looks like this:

```
A is a matrix
B is another matrix
Result is an empty matrix with size A.rows and A.columns

if A.columns = B.columns AND A.rows = A.columns
    for i from 0 to A.rows
        for j from 0 to A.columns
            Result[i][j] = A[i][j] + B[i][j]
```

In pseudo code the algorithm for matrix subtraction looks like this:

```
A is a matrix
B is another matrix
Result is an empty matrix with size A.rows and A.columns

if A.columns = B.columns AND A.rows = A.columns
    for i from 0 to A.rows
        for j from 0 to A.columns
            Result[i][j] = A[i][j] - B[i][j]
```

13.3 Minor of a matrix

The minor of a matrix is a smaller matrix obtained by deleting one row and one column from a larger square matrix. The algorithm takes the i th row and j th column to be deleted and then iterates over the rows of the original matrix excluding the i th row using slicing. For each row it also excludes the j th column by concatenating two slices of the row - the slice from 0 to j and the slice from j + 1 to the end of the row.

In pseudo code the algorithm for finding the minor of a matrix looks like this:

```
A is a matrix
i is the row that should be deleted
j is the column that should be deleted
Result is an empty matrix with size A.rows and A.columns

Result = [row[:j] + row[j+1:] for row in (A[:i] + A[i+1:])]
```

13.4 Determinant of a matrix

The determinant of a matrix is a scalar value that is calculated from a square matrix. To find the determinant we begin by checking if the matrix is square. Then you check if the matrix has size 1x1, in this case the determinant is it's only value. If it isn't a 1x1 matrix, we start by iterating over all of the elements in the first row and return the value of the element and the index. Then you find the minor of the matrix where i = 0 and j = i. Then you use the co-factor formula which is defined as $(-1)^{i+j} \times$ determinant of minor. This method gives you O(n!) time complexity as it uses recursion.

In pseudo code the algorithm for finding the determinant of a matrix looks like this:

```
A is a matrix
determinant = 0

if A.rows = 1
    determinant = A[0][0]
else:
    for i, value in enumerate(A[0])
        minor = A.minor(0, i)
        determinant += (-1) ** i * value * minor.determinant()
```

13.5 Inverse of a matrix

To find the inverse of a matrix I will use the Gauss-Jordan elimination method[5]. The process involves performing a sequence of elementary row operations on a matrix until it is reduced to the identity matrix. The same sequence of operations is then applied to an augmented matrix that includes the identity matrix on the right-hand side, resulting in the inverse matrix on the left-hand side. Firstly, we check whether the matrix isn't square and if the determinant is equal to 0, because we can't find the inverse when these conditions are true. The next step is to create a copy of the matrix and create an identity matrix of the same size. Then we iterate through each diagonal element of the matrix and for each element, we divide the entire row by that diagonal element to make it equal to 1. Then we subtract the diagonal element from all the other elements in the same column to make the equal to zero. This process is repeated for all diagonal elements until the entire matrix is transformed into an identity matrix. The resulting transformed identity matrix is the inverse of the original matrix.

In pseudo code the algorithm for finding the inverse of a matrix looks like this:

```
A is a matrix
C is a copy of A
I is an identity with size A.rows

```

```
 if A.determinant != 0 AND A.columns == A.rows
     indices = list(range(A.rows))
     for current_diagonal in range(A.rows)
         cd_ividor = 1 / C[current_diagonal][current_diagonal]
         for i range(A.rows)
             C[current_diagonal][i] * cd_dividor
             I[current_diagonal][i] * cd_dividor
         for j in indices[:current_diagonal] + indices[
current_diagonal+1:]
             current_row = C[j][current_diagonal]

             for k in range(A.row)
                 C[j][k] -= C[cd][k] * current_row
                 I[j][k] -= C[cd][k] * current_row
```

13.6 Projecting a 3D point to 2D space

To project a point to 2D space you first have to make a copy of the point. Then you multiply the copied point by the camera matrix. Next you multiply the copied point by the projection matrix. Then, you divide the x-, y-, and z-coordinates by the w-coordinate if its not equal to 0. Finally, if the x- and y-coordinates lie between 2 and -2 they scaled to screen size and drawn.

```
P is the point
CP is the copied point
C is the camera matrix
Proj is the projection matrix

CP *= C
CP *= Proj

x, y, z, w = CP
if w != 0
    x /= w
    y /= w
    z /= w

    if (x < 2 and x > -2) AND (y < 2 and y > -2)
        x, y = (x + 1) * window_height * 0.5, (y + 1) *
window_height * 0.5
```

13.7 Reading .obj files

A .obj file is comprised of a list of numbers that each start with a letter. If the letter is a v, the numbers refer to the coordinates of the point. If the letter is a f the numbers refer to the positions of the vertices in the file that make up the face. To generate all the vertices and triangles we simply need to read each line, if it starts with a v the point is appended to a points list, and if it starts with an f we append the face to triangles list. Since indexing in .obj files start at 1 we have to subtract 1 from the value. Finally, we create triangle objects by indexing the points list with the values in the triangles list.

```
vertices = []
triangles = []

for line in file
    if line starts with "v"
        vertices.append([float(i) for i in line.split()[1:]]
    elif line starts with "f"
        faces = line.split(1:)
        triangles.append([int(face.split("/")[0]) - 1 for face in faces])

for triangle in triangles:
    create Triangle(vertices[triangle[0]], vertices[triangle[1]], vertices[triangle[2]])
```

Chapter 14

Modular structure of the system

The hierarchy chart in Figure14.1 shows how each module will interact with each other.

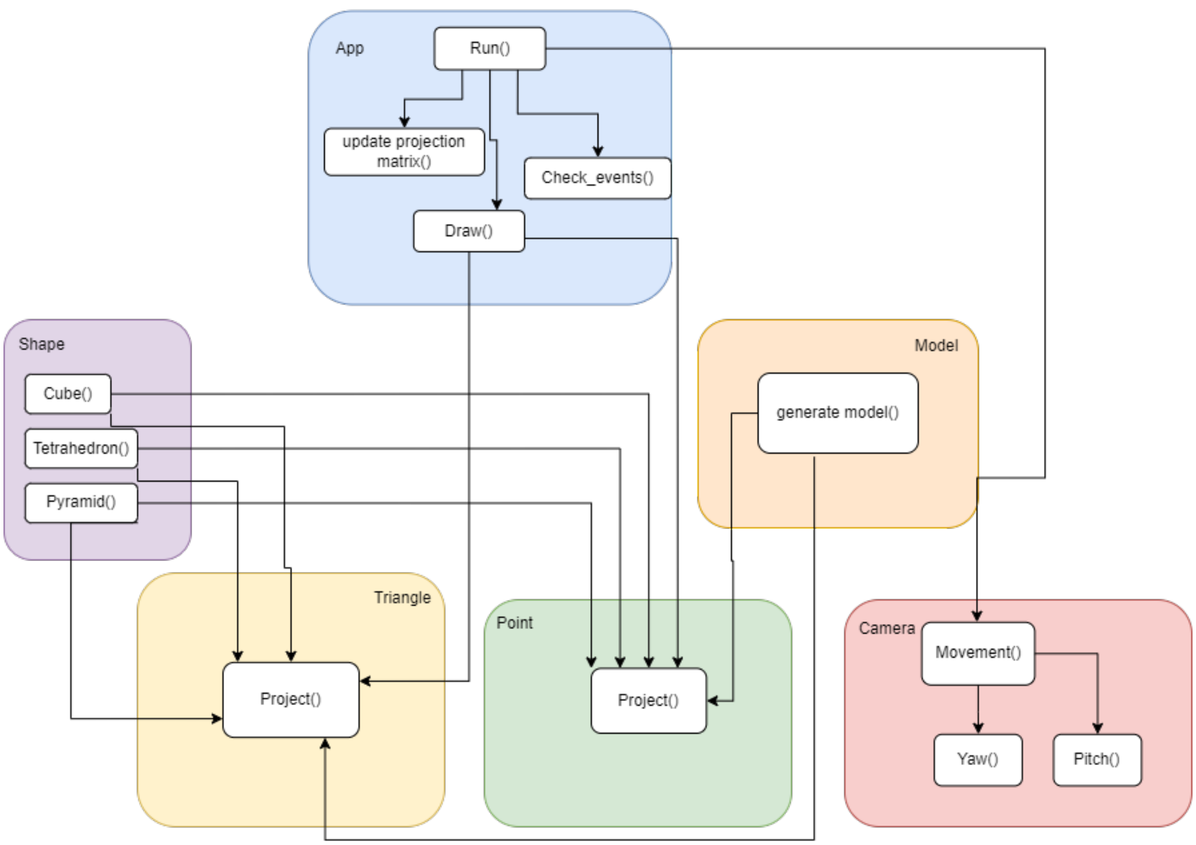


Figure 14.1: Hierarchy Chart

Chapter 15

User Interface

15.1 Visual aspect

The visual aspect of the user interface of the window of the library will be quite simple. The only requirements are that the user must be able to see the FPS on the top bar of the screen, and if the user presses the "o" key they should be able to see FPS, FOV and screen dimensions on the screen itself. The optional stats will be located on the top left of the screen and will be written with a small font so they do not take up the whole screen as you can see in Figure 15.1. Since the extra stats will be drawn on the same window where the scene is drawn, it is imperative that they are visible when any background color is selected. To do this I will simply subtract 255 from the r, b, and g components of the background color to ensure that the text is always the opposite color of the background.

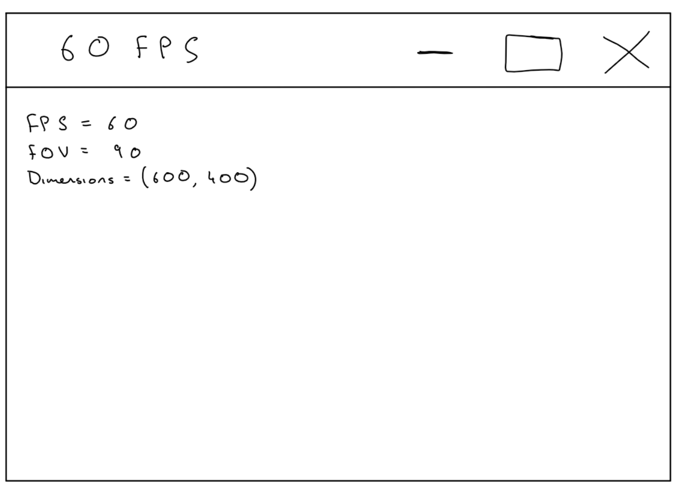


Figure 15.1: Program window

If the user draws a shape and moves and looks around the scene, the shape should distort while keeping its perspective as shown in Figure 15.2.

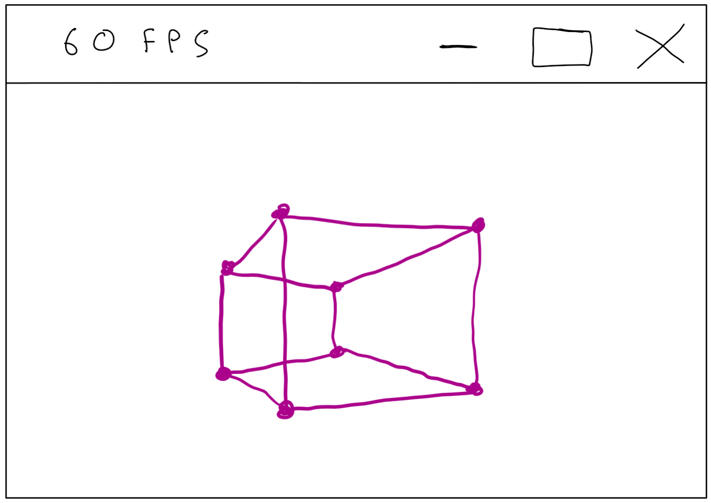


Figure 15.2: Program window

15.2 Movement controls

When the user runs their program, they should be able to use the w, a, s, and d keys to move around the scene and use the arrow keys, press the esc and o key to exit the screen and toggle statistics and use their mouse to look around the scene.

Key input can be implemented by using a Pygame command which will generate a dictionary of the all the keys that have been pressed followed by a Boolean value that tells you if they have been pressed.

The command in question is:

```
pygame.key.get_pressed()
```

After this command if statements can check whether the keys that were pressed are the same ones that the user will press to move and look around the scene. If the Boolean value for the key is True, a translation or rotation will be applied to the camera to make the movement correct.

A similar command is used for mouse motion:

```
pygame.event.get()
```

This command produces a list of all the events that have taken place in the frame. Now we just have to check whether the event is `pygame.MOUSEMOTION`. This event returns a value called `rel` which is a tuple containing the relative motion of the mouse in the x and y directions. These values can be used to rotate the camera.

The only issue with this method is that the position of the mouse is never fixed therefore the camera will continuously rotate. To fix this we can simply fix the position of the mouse at the center of the screen at each frame using the following command:

```
    pygame.mouse.set_pos((self.half_width, self.half_height))
```

15.3 Example of interaction

1. User installs library in command prompt using pip:
    - `pip install pg3d`
2. User imports library into their program
    - `from pg3d import *`
3. User creates an app object with no parameters
    - `app = App()`
4. User creates a shape object with the app, cube and center position as its parameters
    - `app = Shape(app, "cube", center=[0, 0, 2])`
5. User calls run method in app object.
    - `app.run()`
6. Window with a cube in front of the camera is drawn.

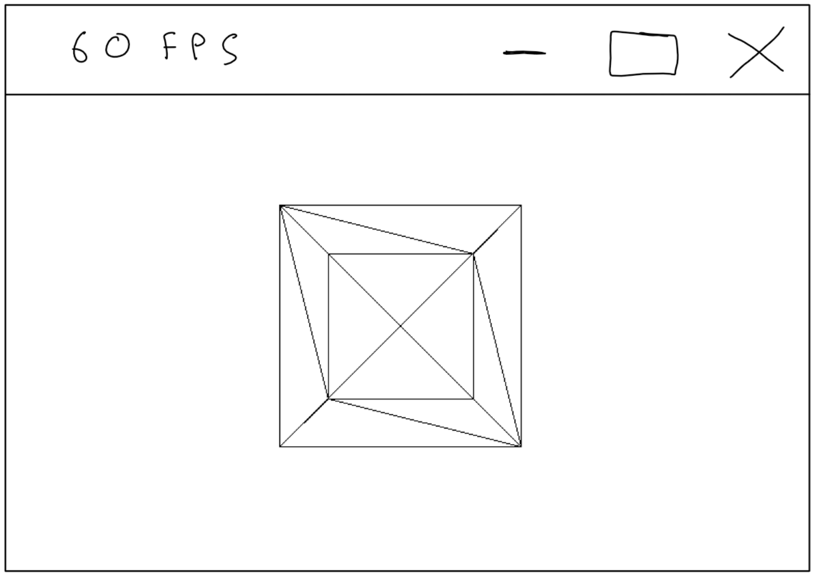


Figure 15.3: Initial window with cube

7. User looks to the left using arrow keys.

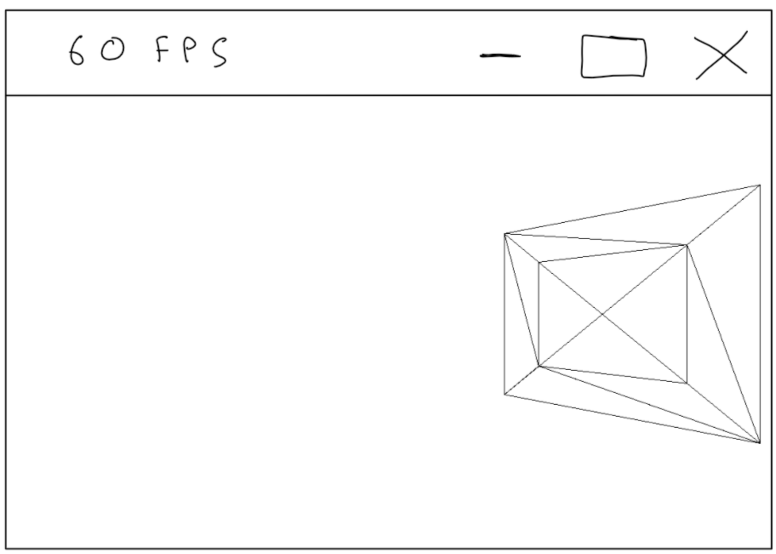


Figure 15.4: Arrow key input

8. Users presses escape key and window closes.

Part III

Implementation

Contents

- **Chapter 28, lines 54 - 76:** - Advanced matrix operations

  These lines of code show the dot() function which is used in the `__mul__()` method to multiply to matrices together. This is a magic method meaning that the * operator can be used to multiply matrices together.

- **Chapter 28, lines 158 - 178:** - Advanced matrix operations

  These lines of code show the `__rmul__()` which is used to find the product of a matrix and a number.

- **Chapter 28, lines 198 - 240:** - Advanced matrix operations

  These lines of code show the methods used for addition and subtraction of matrices: `__add__()` and `__sub__()`. These are magic methods meaning that the * operator can be used to add and subtract matrices together.

- **Chapter 28, lines 253 - 277:** - Advanced matrix operations

  These lines of code show the method to find the minor of a matrix. This method uses list comprehension and slicing.

- **Chapter 28, lines 279 - 303:** - Advanced matrix operations and Recursive algorithm

  These lines of code show the method to find the determinant of a matrix. This method uses recursion and the minor of a matrix to find the determinant.

- **Chapter 28, lines 305 - 349:** - Advanced matrix operations

  These lines of code show the method to find the inverse of a matrix. This method uses the minor, determinant and many for loops to find the inverse. The inverse of a matrix is used in **Chapter 18, line 88** to find the inverse of the camera matrix.

- **Chapter 17, lines 130 - 132:** - Complex mathematical model

  These lines of code show how the projection matrix is defined.

- **Chapter 24, lines 66 - 90:** - Complex mathematical model

  These lines of code take the current point, the projection matrix, and the camera matrix to project the point from 3D space to 2D space.

- **Chapter 23, lines 18 - 43:** - Reading from files

  These following lines of code show the method used to generate all the triangles and points that make up a model in a .obj file.

Chapter 16

.\pg3d__init__.py

```python
from pg3d.point import Point
from pg3d.app import App
from pg3d.triangle import Triangle
from pg3d.cube import Cube
from pg3d.pyramid import Pyramid
from pg3d.tetrahedron import Tetrahedron
from pg3d.model import Model
```

Chapter 17

.\pg3d\app.py

```
import pygame as pg
import math as m
from pygame.colordict import THECOLORS
from typing import Optional, Tuple, Sequence
import pg3d.MatrixMath.matrix as mm
from pg3d.camera import Camera
from pg3d.triangle import Triangle
from pg3d.point import Point


class App:
    def __init__(
        self,
        dimensions=(1000, 700),
        cam_pos=[0, 0, 0],
        BG_COLOR=None,
        LINE_COLOR=None,
        VERTEX_SIZE=2,
        fullscreen=False,
        mouse_look=False,
    ):
        """
        Initialises the library, creates the projection matrix and creates a camera

        Args:
            dimensions ([tuple], optional): [window dimensions]. Defaults to (1000, 700).
            cam_pos ([list], optional): [position of camrea]. Defaults to [0, 0, 0].
            BG_COLOR ([tuple], optional): [background color]. Defaults to (0, 0, 0).
```

```python
            LINE_COLOR ([tuple], optional): [color for drawing lines and points]. Defaults to (255, 255, 255).
            VERTEX_SIZE ([int], optional): [size of points]. Defaults to 2.
            stats ([bool], optional): [shows some stats on screen]. Defaults to False.
            fullscreen ([bool], optional): [makes screen fullscreen]. Defaults to False.
            mouse_look ([bool], optional): [use mouse movement too look with camera]. Defaults to False.
        """
        if not isinstance(dimensions, list) and not isinstance(dimensions, tuple):
            raise Exception("dimensions must be list or tuple")

        if isinstance(dimensions[0], list) or isinstance(dimensions[0], tuple):
            raise Exception("dimensions cannot be larger than 1D")

        if not (len(dimensions) == 2):
            raise Exception("dimensions must contain 2 values")

        if not isinstance(cam_pos, list) and not isinstance(cam_pos, tuple):
            raise Exception("cam_pos must be list or tuple")

        if isinstance(cam_pos[0], list) or isinstance(cam_pos[0], tuple):
            raise Exception("cam_pos cannot be larger than 1D")

        if not (len(cam_pos) == 3):
            raise Exception("cam_pos must contain 3 values")

        if not isinstance(VERTEX_SIZE, int):
            raise Exception("VERTEX_SIZE must be int")

        if (
            not not isinstance(BG_COLOR, bool)
            and not isinstance(BG_COLOR, list)
            and not isinstance(BG_COLOR, tuple)
        ) or (
            not not isinstance(LINE_COLOR, bool)
            and not isinstance(LINE_COLOR, list)
            and not isinstance(LINE_COLOR, tuple)
        ):
            raise Exception("BG_COLOR and LINE_COLOR must be either list or tuple")
```

```

        pg.init()

        if fullscreen:
            self._screen = pg.display.set_mode((0, 0), pg.FULLSCREEN
    , vsync=1)
            self._dimensions = (
                self._width,
                self._height,
            ) = pg.display.get_surface().get_size()
        else:
            self._dimensions = self._width, self._height =
    dimensions
            self._screen = pg.display.set_mode(self._dimensions, pg.
    RESIZABLE, vsync=1)

        self._half_width, self._half_height = self._width / 2, self.
    _height / 2
        self.FPS = 60
        self._screen = pg.display.set_mode(self._dimensions, pg.
    RESIZABLE, vsync=1)
        self._clock = pg.time.Clock()
        self.stats = False
        self.check_stats = 0

        if mouse_look:
            pg.mouse.set_visible(
                0
            )  # sets the mouse to invisible if mouse movement is
    turned on

        self.mouse_look = mouse_look

        self.BG_COLOR = BG_COLOR
        self.LINE_COLOR = LINE_COLOR
        self.VERTEX_SIZE = VERTEX_SIZE

        if (self.BG_COLOR is None) and (
            self.LINE_COLOR is not None
        ):  # user inputs line color but not bg color
            self.BG_COLOR = (
                255 - self.LINE_COLOR[0],
                255 - self.LINE_COLOR[1],
                255 - self.LINE_COLOR[2],
            )
        elif (self.BG_COLOR is not None) and (
            self.LINE_COLOR is None
```

```python
        ):  # user inputs bg color but not line color
            self.LINE_COLOR = (
                255 - self.BG_COLOR[0],
                255 - self.BG_COLOR[1],
                255 - self.BG_COLOR[2],
            )
        else:  # user inputs nothing for bg and line color
            self.BG_COLOR = (0, 0, 0)
            self.LINE_COLOR = (255, 255, 255)

        self.camera = Camera(cam_pos)

        self.mesh = []

        self.fov = 90
        self.z_far = 1000
        self.z_near = 0.1

        m00 = (self._height / self._width) * (1 / m.tan(m.radians(self.fov / 2)))
        m11 = 1 / m.tan(m.radians(self.fov / 2))
        m22 = self.z_far / (self.z_far - self.z_near)
        m32 = -self.z_near * (self.z_far / (self.z_far - self.z_near))

        self.projection_matrix = mm.Matrix(
            [[m00, 0, 0, 0], [0, -m11, 0, 0], [0, 0, m22, 1], [0, 0, m32, 0]]
        )

    def _update_projection_matrix(self):
        """
        Updates the projection matrix when the values of fov and aspect ratio are changed by the user
        """
        m00 = (self._height / self._width) * (1 / m.tan(m.radians(self.fov / 2)))
        m11 = 1 / m.tan(m.radians(self.fov / 2))
        m22 = self.z_far / (self.z_far - self.z_near)
        m32 = -self.z_near * (self.z_far / (self.z_far - self.z_near))

        self.projection_matrix = mm.Matrix(
            [[m00, 0, 0, 0], [0, -m11, 0, 0], [0, 0, m22, 1], [0, 0, m32, 0]]
        )
```

```python
    def add_point(self, point):
        """
        When a user creates a point object this function is called and adds the point to mesh

        Args:
            point ([Point]): [a point object]
        """
        self.mesh.append(point)

    def add_triangle(self, triangle):
        self.mesh.append(triangle)

    def _draw(self):
        self._screen.fill(self.BG_COLOR)

        for shape in self.mesh:
            if isinstance(shape, Triangle):
                shape._project()

            elif isinstance(shape, Point):
                projected = shape._project(
                    self.projection_matrix, self.camera._cam_mat()
                )

                if projected is not None:
                    x, y, z = projected

                    if shape._vertex == True:
                        pg.draw.circle(
                            self._screen, self.LINE_COLOR, (x, y), self.VERTEX_SIZE
                        )

    def display_stats(self):
        """
        If self.stats is true, this method will display stats on screen every frame
        """
        if self.stats == True:
            bg = self.BG_COLOR
            font_color = (255 - bg[0], 255 - bg[1], 255 - bg[2])
            font = pg.font.Font("freesansbold.ttf", 10)
            fov = font.render(f"fov = {self.fov}", True, font_color)
            fps = font.render(f"fps = {round(self._clock.get_fps())}", True, font_color)
            dimensions = font.render(
```

```
                f"dimensions = {self._dimensions}", True, font_color
            )
            self._screen.blit(fov, (5, 5))
            self._screen.blit(fps, (5, 15))
            self._screen.blit(dimensions, (5, 25))

    def _check_events(self):
        for event in pg.event.get():
            if event.type == pg.QUIT:
                exit()

            elif event.type == pg.KEYDOWN and event.key == pg.K_ESCAPE:
                exit()

            elif event.type == pg.KEYDOWN and event.key == pg.K_o:
                if self.check_stats % 2 == 0:
                    self.stats = True

                else:
                    self.stats = False

                self.check_stats += 1

            elif event.type == pg.MOUSEWHEEL:
                if event.y == 1:
                    self.fov -= 1
                else:
                    self.fov += 1
                self._update_projection_matrix()

            elif event.type == pg.VIDEORESIZE:
                self._screen = pg.display.set_mode((event.w, event.h), pg.RESIZABLE)
                self._dimensions = self._width, self._height = (event.w, event.h)
                self._half_width, self._half_height = self._width / 2, self._height / 2
                self._update_projection_matrix()

            elif (self.mouse_look == True) and (event.type == pg.MOUSEMOTION):
                self.camera._mouse_look(event.rel)

    def run(self):
        """
        Main loop of the library which checks for camera control and
```

```
 other events, and draws and projects the points
        """
        while True:
            self._draw()
            self.camera._movement()
            self._check_events()
            self.display_stats()

            if self.mouse_look == True:
                pg.mouse.set_pos((self._half_width, self._half_height))

            pg.display.set_caption(f"{round(self._clock.get_fps())} FPS")
            pg.display.update()
            self._clock.tick(self.FPS)
```

Chapter 18

.\pg3d\camera.py

```python
import pg3d.MatrixMath.matrix as mm
from pg3d.matrices import rotate_x, rotate_y, rotate_z
import pygame as pg
import math as m


class Camera:
    def __init__(self, position, forward=[0, 0, 1], up=[0, 1, 0],
 right=[1, 0, 0]):
        """
        Args:
            position ([list]): [Starting position of camera]
            forward ([list], optional): [Orientation along z-axis].
 Defaults to [0, 0, 1].
            up ([list], optional): [Orientation along y-axis].
 Defaults to [0, 1, 0].
            right ([list], optional): [Orientation along x-axis].
 Defaults to [1, 0, 0].
        """
        self.pos = mm.Matrix([[*position, 1]])
        self._forward = mm.Matrix([[*forward, 1]])
        self._up = mm.Matrix([[*up, 1]])
        self._right = mm.Matrix([[*right, 1]])

        self.speed = 1
        self.angle = m.radians(1)

    def yaw(self, angle):
        """
        Rotates camera directions along y-axis by angle

        Args:
```

```
            angle ([float]): [Angle used to rotate camera orientation]
        """
        self._up *= rotate_y(angle)
        self._forward *= rotate_y(angle)
        self._right *= rotate_y(angle)

    def pitch(self, angle):
        """
        Rotates camera directions along x-axis by angle

        Args:
            angle ([float]): [Angle used to rotate camera orientation]
        """
        self._up *= rotate_x(angle)
        self._forward *= rotate_x(angle)
        self._right *= rotate_x(angle)

    def _rot_mat(self):
        """
        Creates rotation matrix

        Returns:
            [Matrix]: [Rotation matrix used to rotate all point in world space around camera]
        """
        fx, fy, fz, fw = self._forward[0]
        rx, ry, rz, rw = self._right[0]
        ux, uy, uz, uw = self._up[0]

        return mm.Matrix(
            [[rx, ry, rz, 0],
             [ux, uy, fz, 0],
             [fx, fy, fz, 0],
             [0, 0, 0, 1]]
        )

    def _trans_mat(self):
        """
        Creates translate matrix

        Returns:
            [Matrix]: [Translation matrix used to put camera at center of 3D space]
        """
        x, y, z, w = self.pos[0]
```

```python

        return mm.Matrix(
            [[1, 0, 0, 0],
             [0, 1, 0, 0],
             [0, 0, 1, 0],
             [x, y, z, 1]]
        )

    def _cam_mat(self):
        """
        Creates camera matrix

        Returns:
            [Matrix]: [Camera matrix which is multiplied to all points]
        """
        camera_matrix = self._trans_mat() * self._rot_mat()
        return camera_matrix.inverse()

    def _movement(self):
        """
        Checks for user input and then performs the necessary rotations and translations
        """
        key = pg.key.get_pressed()

        if key[pg.K_a]:
            self._right = self.speed * self._right
            self.pos = self.pos - self._right
        if key[pg.K_d]:
            self._right = self.speed * self._right
            self.pos = self.pos + self._right
        if key[pg.K_w]:
            self._forward = self.speed * self._forward
            self.pos = self.pos + self._forward
        if key[pg.K_s]:
            self._forward = self.speed * self._forward
            self.pos = self.pos - self._forward
        if key[pg.K_q]:
            self._up = self.speed * self._up
            self.pos = self.pos + self._up
        if key[pg.K_e]:
            self._up = self.speed * self._up
            self.pos = self.pos - self._up

        if key[pg.K_LEFT]:
            self.yaw(-self.angle)
```

```
        if key[pg.K_RIGHT]:
            self.yaw(self.angle)
        if key[pg.K_UP]:
            self.pitch(self.angle)
        if key[pg.K_DOWN]:
            self.pitch(-self.angle)

    def _mouse_look(self, rel):
        """
        Moves camera when mouse is moved

        Args:
            rel ([float]): [relative position of mouse]
        """
        x, y = rel
        self.yaw(x / 1000)
        self.pitch(-y / 1000)
```

Chapter 19

.\pg3d\Cube.py

```python
from pg3d.shape import Shape
from pg3d.triangle import Triangle


class Cube(Shape):
    def __init__(self, app, size=1, center=[0, 0, 0]):
        super().__init__(app, size, center)

        self._generate_shape()

    def _generate_shape(self):
        """
        Creates points and vertices of cube

        Returns:
            [tuple[list, list]]: [tuple containing list of vertices
    and list of triangles]
        """
        x, y, z = self._center
        half_size = self._size / 2

        vertices = [
            [x - half_size, y + half_size, z + half_size],  # Front-
    bottom-left
            [x + half_size, y + half_size, z + half_size],  # Front-
    bottom-right
            [x + half_size, y - half_size, z + half_size],  # Front-
    top-right
            [x - half_size, y - half_size, z + half_size],  # Front-
    top-left
            [x - half_size, y + half_size, z - half_size],  # Back-
    bottom-left
```

```
            [x + half_size, y + half_size, z - half_size],  # Back-
bottom-right
            [x + half_size, y - half_size, z - half_size],  # Back-
top-right
            [x - half_size, y - half_size, z - half_size],  # Back-
top-left
        ]

        triangles = [
            [0, 1, 2],
            [0, 2, 3],  # Front face
            [1, 5, 6],
            [1, 6, 2],  # Right face
            [3, 2, 6],
            [3, 6, 7],  # Top face
            [4, 5, 1],
            [4, 1, 0],  # Bottom face
            [4, 0, 3],
            [4, 3, 7],  # Left face
            [7, 6, 5],
            [7, 5, 4],  # Back face
        ]

        # creates the triangles
        if (triangles != None) and (vertices != None):
            for triangle in triangles:
                Triangle(
                    self.app,
                    [
                        vertices[triangle[0]],
                        vertices[triangle[1]],
                        vertices[triangle[2]],
                    ],
                )
```

Chapter 20

.\pg3d\Pyramid.py

```
from pg3d.shape import Shape
from pg3d.triangle import Triangle


class Pyramid(Shape):
    def __init__(self, app, size=1, center=[0, 0, 0]):
        super().__init__(app, size, center)

        self._generate_shape()

    def _generate_shape(self):
        """
        Creates points and vertices of pyramid

        Returns:
            [tuple[list, list]]: [tuple containing list of vertices
   and list of triangles]
        """
        x, y, z = self._center
        half_size = self._size / 2

        vertices = [
            [x - half_size, y - half_size, z - half_size],  # Bottom
   -back-left
            [x - half_size, y - half_size, z + half_size],  # Bottom
   -front-left
            [x + half_size, y - half_size, z - half_size],  # Bottom
   -back-right
            [x + half_size, y - half_size, z + half_size],  # Bottom
   -front-right
            [x, y + self._size, z],  # Top
        ]
```

```python

        triangles = [
            [0, 1, 2],
            [0, 2, 3],
            [1, 2, 3],  # Base face
            [0, 4, 3],  # Front-left face
            [1, 4, 0],  # Front-right face
            [2, 4, 1],  # Back-right face
            [3, 4, 2],  # Back-left face
        ]

        if (triangles != None) and (vertices != None):
            for triangle in triangles:
                Triangle(
                    self.app,
                    [
                        vertices[triangle[0]],
                        vertices[triangle[1]],
                        vertices[triangle[2]],
                    ],
                )
```

Chapter 21

.\pg3d\Tetrahedron.py

```
from pg3d.shape import Shape
from pg3d.triangle import Triangle


class Tetrahedron(Shape):
    def __init__(self, app, size=1, center=[0, 0, 0]):
        super().__init__(app, size, center)

        self._generate_shape()

    def _generate_shape(self):
        """
        Creates points and vertices of tetrahedron

        Returns:
            [tuple[list, list]]: [tuple containing list of vertices
 and list of triangles]
        """
        half_size = self._size / 2
        x, y, z = self._center
        height = (
            self._size * 0.86
        )  # Multiply by 0.86 to adjust height to make it
 equilateral

        vertices = [
            [x, y + height / 3, z],  # Top
            [x - half_size, y - height / 3, z - half_size],  #
 Bottom-front-left
            [x + half_size, y - height / 3, z - half_size],  #
 Bottom-front-right
            [x, y - height / 3, z + half_size],  # Bottom-back
```

```python
        ]

        triangles = [
            [0, 1, 2],  # Front face
            [0, 2, 3],  # Right face
            [0, 3, 1],  # Left face
            [1, 3, 2],  # Bottom face
        ]

        if (triangles != None) and (vertices != None):
            for triangle in triangles:
                Triangle(
                    self.app,
                    [
                        vertices[triangle[0]],
                        vertices[triangle[1]],
                        vertices[triangle[2]],
                    ],
                )
```

Chapter 22

.\pg3d\matrices.py

```
import pg3d.MatrixMath.matrix as mm
import math as m


def rotate_x(angle):
    """
    Rotation matrix on x-axis

    Args:
        angle ([float]): [Angle for rotation]

    Returns:
        [Matrix]: [Creates rotation matrix along x-axis]
    """
    return mm.Matrix(
        [
            [1, 0, 0, 0],
            [0, m.cos(angle), m.sin(angle), 0],
            [0, -m.sin(angle), m.cos(angle), 0],
            [0, 0, 0, 1],
        ]
    )


def rotate_y(angle):
    """
    Rotation matrix on y-axis

    Args:
        angle ([float]): [Angle for rotation]

    Returns:
```

```
        [Matrix]: [Creates rotation matrix along y-axis]
    """
    return mm.Matrix(
        [
            [m.cos(angle), 0, -m.sin(angle), 0],
            [0, 1, 0, 0],
            [m.sin(angle), 0, m.cos(angle), 0],
            [0, 0, 0, 1],
        ]
    )


def rotate_z(angle):
    """
    Rotation matrix on z-axis

    Args:
        angle ([float]): [Angle for rotation]

    Returns:
        [Matrix]: [Creates rotation matrix along z-axis]
    """
    return mm.Matrix(
        [
            [m.cos(angle), m.sin(angle), 0, 0],
            [-m.sin(angle), m.cos(angle), 0, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1],
        ]
    )
```

Chapter 23

.\pg3d\model.py

```
from pg3d.triangle import Triangle
from pg3d.point import Point


class Model:
    def __init__(self, app, path):
        """
        Args:
            app ([App]): [instance of App class]
            path ([str]): [path of .obj file]
        """
        if not isinstance(path, str):
            raise Exception("Path must be string")
        self.app = app
        self._path = path
        self._generate_model()

    def _generate_model(self):
        """
        Gets vertex and triangle information from .obj file and creates the necessary triangles and points
        """
        vertices, triangles = [], []
        with open(self._path) as file:
            for line in file:
                if line.startswith("v "):
                    vertices.append([float(i) for i in line.split()[1:]])
                elif line.startswith("f "):
                    faces = line.split()[1:]
                    triangles.append([int(face.split("/")[0]) - 1 for face in faces])
```

```python

        if len(triangles) > 0:
            for triangle in triangles:
                Triangle(
                    self.app,
                    (
                        vertices[triangle[0]],
                        vertices[triangle[1]],
                        vertices[triangle[2]],
                    ),
                )
        else:
            for vertex in vertices:
                Point(self.app, vertex)
```

Chapter 24

.\pg3d\point.py

```
import pygame as pg
import pg3d.MatrixMath.matrix as mm
import math as m
from pygame.colordict import THECOLORS


class Point:
    def __init__(self, app, coordinate, vertex=True):
        """
        Creates the point

        Args:
            app ([App]): [instance of App class]
            coordinate ([list]): [coordinate of point]
            vertex (bool, optional): [flag that says whether point is drawn]. Defaults to True.
        """
        if not isinstance(coordinate, list) and not isinstance(coordinate, tuple):
            raise Exception("Coordinate must be list or tuple")

        if len(coordinate) > 3:
            raise Exception("Coordinate must have 3 numbers")

        if not isinstance(vertex, bool):
            raise Exception("Vertex type must be bool")

        self.coordinate = mm.Matrix([[*coordinate, 1]])
        self.app = app
        self.app.add_point(self)
        self._vertex = vertex

```

```
    def __repr__(self):
        """
        Defines the behaviour of printing Point objects

        Returns:
            [str]: [String representation of point]
        """
        return str(self.coordinate[0])

    def __setitem__(self, index, value):
        """
        Defines the behaviour of setting an indexed Point object to
  a value

        Args:
            index ([int]): [position of point]
            value ([float]): [vew value of coordinate]
        """
        self.coordinate[0][index] = value

    def __getitem__(self, index):
        """
        Defines the behaviour for indexing a Point object

        Args:
            index ([int]): [position of coordinate]

        Returns:
            [float]: [coordinate that was indexed]
        """
        if index == 0 or index == 1 or index == 2 or index == 3:
            return self.coordinate[0][index]

        else:
            return "invalid position"

    def _project(self, proj, cam):
        """
        projects point

        Args:
            proj ([Matrix]): [projection matrix]
            cam ([Matrix]): [camera matrix]

        Returns:
            [tuple]: [Returns projected point]
        """
```

```
        copy = mm.copy_matrix(self.coordinate)
        copy *= cam
        projected = copy * proj
        x, y, z, w = projected[0]

        if w != 0:
            x /= w
            y /= w
            z /= w
            if (x < 2 and x > -2) and (y < 2 and y > -2):
                x, y = (x + 1) * self.app._half_width, (y + 1) *
    self.app._half_height
                return (x, y, z)
            else:
                return None
```

Chapter 25

.\pg3d\shape.py

```
class Shape:
    def __init__(self, app, size=1, center=[0, 0, 0]):
        if not isinstance(center, list) and not isinstance(center,
    tuple):
            raise Exception("center must be list or tuple")

        if not isinstance(size, float) and not isinstance(size, int)
    :
            raise Exception("size must be int or float")

        if len(center) != 3:
            raise Exception("center must have 3 values")

        self.app = app
        self._center = center
        self._size = size
```

Chapter 26

.\pg3d\triangle.py

```
from pg3d.point import Point
import pygame as pg


class Triangle:
    def __init__(self, app, vertices):
        """
        Creates triangle

        Args:
            app ([App]): [Specify App object]
            vertices ([list[list]]): [list with 3 cartesian coordinates]
        """
        if not isinstance(vertices, list) and not isinstance(vertices, tuple):
            raise Exception("vertices must be list or tuple")

        if not isinstance(vertices[0], list) and not isinstance(vertices[0], tuple):
            raise Exception("vertices must be 2D list or tuple")

        if len(vertices) != 3:
            raise Exception("vertices must contain 3 coordinates")

        self.points = [
            Point(app, vertices[0], False),
            Point(app, vertices[1], False),
            Point(app, vertices[2], False),
        ]
        self._projected_points = []
        self.app = app
```

```
        self.app.add_triangle(self)

    def _project(self):
        """
        Projects triangle
        """
        self.projected_points = []
        for point in self.points:
            projected = point._project(
                self.app.projection_matrix, self.app.camera._cam_mat
()
            )
            if projected != None:
                self.projected_points.append(projected)

        self._draw_triangle()

    def _draw_triangle(self):
        """
        Draws triangle
        """
        if len(self.projected_points) == 3:
            a, b, c = self.projected_points
            pg.draw.polygon(
                self.app._screen, self.app.LINE_COLOR, (a[:-1], b
[:-1], c[:-1]), 1
            )

    def __getitem__(self, index):
        """
        Args:
            index ([int]): [index of triangle]

        Returns:
            [Point]: [returns point]
        """
        return self.points[index]
```

Chapter 27

.\MatrixMath__init__.py

In order to import the 'matrix.py' file, which is located in its own folder, an accompanying '__init__.py' file is required. This file acts as a module initializer, allowing the 'matrix.py' file to be accessed and used by other Python scripts. However, as the '.\MatrixMath' folder contains only a single file, no additional code is required in the '__init__.py' file. As such, it has been deliberately left empty, simplifying the module structure and avoiding unnecessary code complexity.

Chapter 28

.\pg3d\MatrixMath\matrix.py

```python
def zeroes(height, width):
    """
    Creates a matrix of size h x w and fills it with zeroes

    Args:
        height ([int]): [rows of matrix]
        width ([int]): [columns of matrix]

    Returns:
        [Matrix]: [Matrix filled with zeroes]
    """
    if type(height) != int or type(height) != int:
        raise TypeError("height and width must be integer")

    return Matrix([[0 for w in range(width)] for h in range(height)
    ])


def identity(n):
    """
    Returns an identity matrix of size n x n

    Args:
        n ([int]): [size of square matrix]
    """
    if type(n) != int:
        raise TypeError("n must be integer")

    matrix = zeroes(n, n)
    for i in range(matrix.height):
        matrix[i][i] = 1

```

```
    return matrix


def copy_matrix(matrix):
    """
    Returns a copy of the inputted matrix

    Args:
        matrix ([Matrix]): [matrix that needs to be copied]

    Returns:
        [Matrix]: [copied matrix]
    """
    if type(matrix) == list:
        return Matrix(
            [[matrix[h][w] for w in range(len(matrix[0]))] for h in range(len(matrix))]
        )
    else:
        m = matrix._matrix
        return Matrix([[m[h][w] for w in range(len(m[0]))] for h in range(len(m))])


def dot(a, b):
    """
    Finds dot product of matrices a and b that are compatible

    Args:
        a ([Matrix]): [Matrix used for dot product]
        b ([Matrix]): [Matrix used for dot product]

    Returns:
        [Matrix]: [result]
    """
    result = zeroes(a.height, b.width)

    for height in range(a.height):
        for width in range(b.width):
            sum = 0

            for b_height in range(a.width):
                sum += a._matrix[height][b_height] * b[b_height][width]

            result[height][width] = sum

```

```python
    return result


class Matrix:
    def __init__(self, matrix):
        """
        initialises the matrix and finds the height and width of the matrix

        Args:
            matrix ([list]): [2D array]
        """
        self._matrix = matrix

        if type(self._matrix) != list:
            raise TypeError("Matrix must be a list")

        self.width = len(self._matrix[0])
        self.height = len(self._matrix)

    def __repr__(self):
        """
        Defines behaviour of printing a matrix object

        Returns:
            [str]: [String representation of matrix obejct]
        """
        print("[", end="")
        # loop that iterates through every item of the matrix
        for height in range(self.height):
            print("[", end="")
            for width in range(self.width):
                if width != self.width - 1:  # if the number is'nt the last in its row
                    print(f"{self._matrix[height][width]}, ", end="")

                else:
                    print(
                        f"{self._matrix[height][width]}", end=""
                    )  # if the number is that last in its row

            if height != self.height - 1:
                print("]")

            else:
                print("]", end="")
```

```python

        print("]", end="")

        return ""

    def __setitem__(self, index, value):
        """
        Defines the behaviour of changing the value of the matrix at a specific index

        Args:
            index ([int]): [position of matrix]
            value ([float]): [new value at position of matrix]
        """
        if type(index) != int:
            raise TypeError("Index must integer")

        self._matrix[index] = value

    def __getitem__(self, index):
        """
        Defines behaviour of using square brackets on matrix objects

        E.g:
        > a = Matrix([1,2,3],[4,5,6])
        > a[0]
          [1,2,3]

        Args:
            index ([int]): [position of matrix]

        Returns:
            [any]: [returns list or float]
        """
        if type(index) != int:
            raise TypeError("Index must integer")

        return self._matrix[index]

    def __rmul__(self, value):
        """
        Defines behaviour of multiplying matrix object with non-matrix object which is to the right of the matrix

        Args:
            value ([float]): [number that is multiplied to matrix]

```

```
        Returns:
            [Matrix]: [Result of multiplication]
        """
        if type(value) == int or type(value) == float:
            result = zeroes(self.height, self.width)

            # iterates through each number and multiplies it with
 the value
            for height in range(self.height):
                for width in range(self.width):
                    result[height][width] = self._matrix[height][
width] * value

            return result
        else:
            raise TypeError("Index must integer or float")

    def __mul__(self, other):
        """
        Defines the behaviour of the * operator for multiplication

        Args:
            other ([Matrix]): [other matrix that is multiplied with]

        Result:
            [Matrix]: [Result of matrix multiplication]
        """
        if type(other) != Matrix:
            raise TypeError("Can only multiply with another Matrix
object")

        if self.width == other.height:
            return dot(self, other)
        else:
            raise Exception("COLUMNS OF MATRIX A MUST EQUAL ROWS OF
MATRIX B")

    def __add__(self, other):
        """
        Defines the behaviour of the + operator for addition

        Args:
            other ([Matrix]): [other matrix that is added to]

        Result:
            [Matrix]: [Result of matrix addition]
        """
```

```
        if type(other) != Matrix:
            raise TypeError("Can only add with another Matrix object")

        if (self.height == other.height) and (self.width == other.width):
            result = zeroes(self.height, self.width)
            for height in range(self.height):
                for width in range(self.width):
                    result[height][width] = self[height][width] + other[height][width]
            return result
        else:
            raise Exception("CANNOT ADD MATRICES WITH DIFFERENT SHAPE")

    def __sub__(self, other):
        """
        Defines the behaviour of the - operator for subtraction

        Args:
            other ([Matrix]): [other matrix that is subtracted to]

        Result:
            [Matrix]: [Result of matrix subtraction]
        """
        if type(other) != Matrix:
            raise TypeError("Can only subtract with another Matrix object")

        if (self.height == other.height) and (self.width == other.width):
            result = zeroes(self.height, self.width)
            for height in range(self.height):
                for width in range(self.width):
                    result[height][width] = self[height][width] - other[height][width]
            return result
        else:
            raise Exception("CANNOT SUBTRACT MATRICES WITH DIFFERENT SHAPE")

    def transpose(self):
        """
        Returns a transposed copy of the matrix

        Returns:
```

```
            [Matrix]: [transposed matrix]
        """
        # 1. uses the zip function to transpose the unpacked matrix
        # 2. uses the map function to turn the sets into lists
        return Matrix(list(map(list, zip(*self._matrix))))

    def minor(self, i, j):
        """
        Returns a copy of the matrix with the row and column, i and j, deleted

        Args:
            i ([int]): [row to be deleted]
            j ([int]): [column to be deleted]

        Returns:
            [Matrix]: [matrix without specified row and column]
        """
        if type(i) != int or type(i) != int:
            raise TypeError("i and j must be integer")

        if self.is_square():
            # removes the i-th row and j-th column using slicing
            return Matrix(
                [
                    row[:j] + row[j + 1 :]
                    for row in (self._matrix[:i] + self._matrix[i + 1 :])
                ]
            )

        else:
            raise Exception("CANNOT FIND MINOR OF NON-SQUARE MATRIX")

    def determinant(self):
        """
        Returns the determinant of a matrix using the method of cofactors

        Returns:
            [float]: [returns determinant of matrix]
        """
        if self.is_square():
            # returns the determinant of a 1x1 matrix
            if self.height == 1:
                return self._matrix[0][0]
```

```python

            determinant = 0

            for i, value in enumerate(
                self._matrix[0]
            ):  # iterate over elements in first row of matrix
                minor = self.minor(0, i)  # calculate minor at position [0, i]
                determinant += (
                    (-1) ** i * value * minor.determinant()
                )  # cofactor formula

            return determinant
        else:
            raise Exception("CANNOT FIND DETERMINANT OF A NON-SQUARE MATRIX")

    def inverse(self):
        """
        Returns the inverse of the matrix using Gauss-Jordan Elimination method

        Returns:
            [Matrix]: [inverse of matrix]
        """
        # check if matrix isnt square
        if not self.is_square():
            raise Exception("CANNOT FIND INVERSE OF NON-SQUARE MATRIX")

        else:
            # check if matrix determinat is equal to 0
            if self.determinant == 0:
                raise Exception("CANNOT FIND INVERSE OF MATRIX WITH DETERMINANT = 0")

            # if both conditions are not met, the inverse will be calculated
            else:
                i = identity(self.height)
                # copies of matrix and identity matrix
                m_copy = copy_matrix(self._matrix)
                i_copy = copy_matrix(i._matrix)

                indices = list(
                    range(self.height)
                )  # list of all the indices in the matrix row
```

```

                for cd in range(self.height):  # cd = current diagonal
                    cd_factor = 1 / m_copy[cd][cd]

                    # divide all the values in the current row by the diagonal item
                    # this is done to make the diagonal item equal to one
                    for i in range(self.height):
                        m_copy[cd][i] *= cd_factor
                        i_copy[cd][i] *= cd_factor

                    for j in indices[:cd] + indices[cd + 1 :]:
                        cr_factor = m_copy[j][cd]  # cr = current row

                        # subtract the current value by the pivot on its row multiplied by the value on the row above
                        for k in range(self.height):
                            m_copy[j][k] -= m_copy[cd][k] * cr_factor
                            i_copy[j][k] -= i_copy[cd][k] * cr_factor

                return i_copy

    def is_square(self):
        """
        Checks whether the matrix is a square matrix

        SQUARE MATRIX:  |   NON-SQUARE MATRIX:
        [[1,2],         |   [[1,2,3],
         [3,4]]         |     [4,5,6]]

        Returns:
            [bool]: [determines whether matrix is square]
        """
        if self.width == self.height:
            return True

        else:
            return False
```

Chapter 29

LICENSE

Including a licence is very important to be uploaded with the package because it tells the users that install the package the terms under which they can use the package.

```
MIT License

Copyright (c) 2023 Daniele Golzio

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Chapter 30

pyproject.toml

pyproject.toml tells frontend build tools what backend tool they should use to create the distributable packages for the library. In my case I will use the setuptools backend tool. "requires is a list of packages that are needed to build your package. You don't need to install them; build frontends like pip will install them automatically in a temporary, isolated virtual environment for use during the build process."[6]. "build-backend is the name of the Python object that frontends will use to perform the build."[6].

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
```

Chapter 31

setup.cfg

This file is used to configure the library metadata. All these options will change how the library will be displayed on the PyPI website. In the metadata you can also specify a file to display the description of the program. In the next chapter Will describe everything that is in the file.

```
[metadata]
name = pg3d
version = 0.0.0
author = Daniele Golzio
author_email = danielegolzio@gmail.com
long_description = file: README.md
long_description_content_type = text/markdown
url = https://github.com/poonchoi/3D-GRAPHICS-ENGINE
classifiers =
    Programming Language :: Python :: 3
    License :: OSI Approved :: MIT License
    Operating System :: OS Independent

[options]
packages = find:
python_requires = >= 3.7
include_package_data = True
```

Chapter 32

README.md

The README.md is displayed on the front page of the libraries page therefore, I have written a guide on how to install the library, basics on how to use it and documentation of the classes that the user will be able to access.

````
# PG3D

PG3D is a simple 3D graphics library written using Pygame.


## Installation
---
1) Install Python 3.7 or newer. https://www.python.org/downloads/
2) Open cmd/terminal and type:
```
pip install pg3d
```

## Dependencies
---
* python 3.7+
* pygame

## Usage
---
1) Import the library
```py
from pg3d import *
```
2) Create an App instance and call the run function
```py
app = App()

````
````

````
#---Code goes here---#

#--------------------#

app.run()
```

## API Reference
---
```py
App(kwargs):


kwargs:
    dimensions=(1000, 700)
    cam_pos=[0, 0, 0]
    BG_COLOR=(0, 0, 0)
    LINE_COLOR=(255, 255, 255)
    VERTEX_SIZE=2
    stats=False                # Show stats on screen
    fullscreen=False
    mouse_look=False           # Use mouse for camera orientation


functions:
    run()  # Draws all vertices and checks for movement
```
---
```py
Model(args)


args:
    app   # Specify the App() object
    path  # Specify path of .obj file
```
---
```py
Shape(args, kwargs)


args:
    app    # Specify the App() object
    shape  # "cube" | "pyramid" | "tetrahedron"


kwargs:
````

```
    size=1
    center=[0, 0, 0]
    width=1
```

```py
Triangle(args)


args:
    app       # Specify the App() object
    vertices  # List or tuple with 3 cartesian coordinates


functions:
    __getitem__(index)  # Returnes indext point of Triangle object
```

```py
Point(args, kwargs)


args:
    app          # Specify the App() object
    coordinate   # A list with a single cartesian coordinate


kwargs:
    vertex=True  # If true the point is drawn and vice versa


functions:
    __repr__()                  # Prints Point object
    __setitem__(index, value)  # Sets indexed coordinate of Point object to value
    __getitem__(index)         # Returns indexed coordinate of Point object
```

```py
Matrix(args)


args:
    matrix  # A 2D array


```

```
functions:
    __repr__()                  # Prints a Matrix object
    __setitem__(index, value)  # Sets indexed value of Matrix object
    to value
    __getitem__(index)          # Returns value of indexed Matrix
   object
    __rmul__(value)             # Returns product of Matrix object
   and number
    __mul__(other)              # Multiplies 2 matrices together
    __add__(other)              # Adds 2 matrices together
    __sub__(other)              # Subtracts 2 matrices together
    transpose()                 # Transposes Matrix object
    determinant()               # Finds determinant of Matrix object
    inverse()                   # Finds inverse of Matrix object
    is_square()                 # Checks whether matrix is square or
   not

```
```

The renderd documentation part of the README file is shown in Figure 32.1.
```

API Reference

```
App(kwargs):


kwargs:
    dimensions=(1000, 700)
    cam_pos=[0, 0, 0]
    BG_COLOR=(0, 0, 0)
    LINE_COLOR=(255, 255, 255)
    VERTEX_SIZE=2
    stats=False              # Show stats on screen
    fullscreen=False
    mouse_look=False         # Use mouse for camera orientation


functions:
    run()  # Draws all vertices and checks for movement
```

```
Model(args)


args:
    app   # Specify the App() object
    path  # Specify path of .obj file
```

```
Shape(args, kwargs)


args:
    app    # Specify the App() object
    shape  # "cube" | "pyramid" | "tetrahedron"


kwargs:
    size=1
    center=[0, 0, 0]
    width=1
```

```
Triangle(args)


args:
    app       # Specify the App() object
    vertices  # List or tuple with 3 cartesian coordinates


functions:
    __getitem__(index)  # Returnes indext point of Triangle object
```

```
Point(args, kwargs)


args:
    app         # Specify the App() object
    coordinate  # A list with a single cartesian coordinate


kwargs:
    vertex=True  # If true the point is drawn and vice versa


functions:
    __repr__()               # Prints Point object
    __setitem__(index, value)  # Sets indexed coordinate of Point object to value
    __getitem__(index)         # Returns indexed coordinate of Point object
```

```
Matrix(args)


args:
    matrix  # A 2D array


functions:
    __repr__()                 # Prints a Matrix object
    __setitem__(index, value)  # Sets indexed value of Matrix object to value
    __getitem__(index)         # Returns value of indexed Matrix object
    __rmul__(value)            # Returns product of Matrix object and number
    __mul__(other)             # Multiplies 2 matrices together
    __add__(other)             # Adds 2 matrices together
    __sub__(other)             # Subtracts 2 matrices together
    transpose()                # Transposes Matrix object
    determinant()              # Finds determinant of Matrix object
    inverse()                  # Finds inverse of Matrix object
    is_square()                # Checks whether matrix is square or not
```

Figure 32.1: README.MD

Chapter 33

Library File Structure

```
.
└── 3D_Graphics_Engine/
    ├── LINCENSE
    ├── pyproject.toml
    ├── setup.cfg
    └── README.MD
        └── dist/
            ├── pg3d-X.Y.Z-py3-none-any.whl
            ├── pg3d-X.Y.Z.tar.gz
        └── pg3d.egg-info/
            ├── dependency_links.txt
            ├── PKG-INFO.md
            ├── SOURCES.txt
            ├── top_level.txt
        └── pg3d/
            ├── __init__.py
            ├── app.py
            ├── camera.py
            ├── matrices.py
            ├── model.py
            ├── point.py
            ├── shape.py
            ├── triangle.py
            └── MatrixMath/
                ├── __init__.py
                └── matrix.py
```

Chapter 34

Packaging the library

To package the library and upload it to the PyPI website, you first have to run the build command in the same directory as `pyproject.toml`:

```
python -m build
```

This command will output two new files in the `dist/` folder. After this you have to make an account on PyPI, create an API token and run this final command:

```
python -m twine upload --repository pypi dist/*
```

After inputting your username as `__token__` and your password as the API key, the files in the `dist/` folder will be uploaded to PyPI.

Now that the library is on PyPI the user can install it by using the following command:

```
pip install pg3d
```

The package front page on PyPI can be found on:

`https://pypi.org/project/pg3d/`

Chapter 35

Prototype 1

This program was the first one that I had made which successfully projected and rotated a 3D point. This program is extremely basic and had a projection matrix which didn't allow for camera movement. In this program I used the Numpy library to be able to use matrices which defeats the purpose of my project. However, this program taught me how to work with classes and the basics of 3D graphics. As you can see in Figure 35.1, the cube is drawn onto the screen but there is no perspective. The way you can tell that there is no perspective is that you don't know which cube face is closer to you.

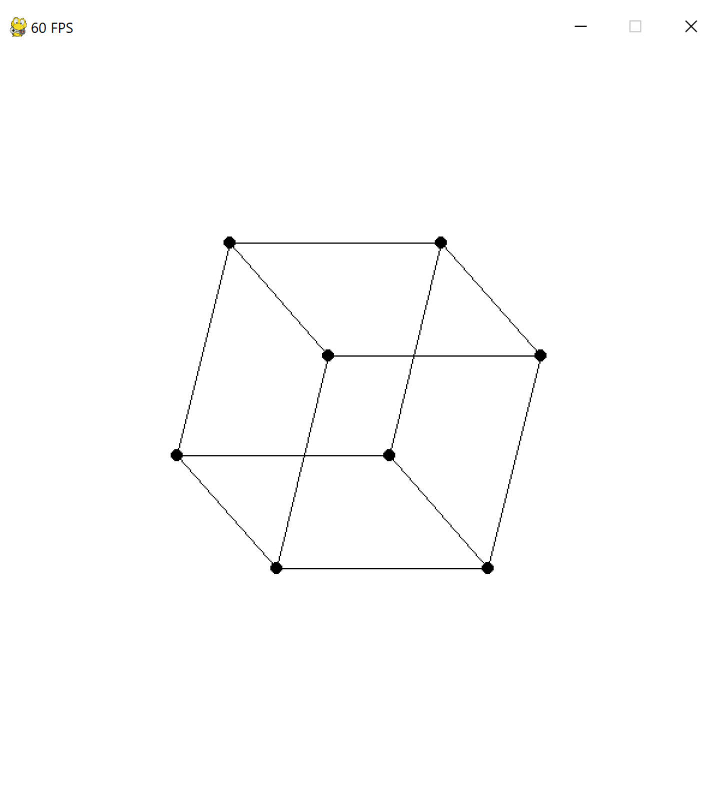


Figure 35.1: Prototype 1

```python
import pygame as pg
from pygame import gfxdraw
import numpy as np
BLACK = (0, 0, 0)
GRAY = (127, 127, 127)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
YELLOW = (255, 255, 0)
CYAN = (0, 255, 255)
MAGENTA = (255, 0, 255)


class App:
    def __init__(self):
        pg.init()
        self.RES = self.WIDTH, self.HEIGHT = 600, 600
        self.H_WIDTH, self.H_HEIGHT = self.WIDTH // 2, self.HEIGHT
    // 2
        self.FPS = 100
        self.screen = pg.display.set_mode(self.RES)
        self.clock = pg.time.Clock()
        self.angle = 0
        self.scale = 100
        self.distance = 9


    def connect_points(self, i, j, points):
        pg.draw.aaline(self.screen, BLACK, (points[i][0], points[i
    ][1]), (points[j][0], points[j][1]))


    def project(self, point):
        z = 1 / (float(self.distance) - point[2])
        p = np.matrix(
            [[z, 0, 0],
            [0, z, 0],
            [0, 0, 0]]
        )
        point = np.array(point)
        projected = np.dot(p, point.reshape(3, 1))
        projected = projected.tolist()
        return [projected[0][0],projected[1][0],projected[2][0]]


    def rotateX(self, point, angle):
```

```
        rotateX = np.matrix(
            [[1, 0, 0],
            [0, np.cos(angle), np.sin(angle)],
            [0, -np.sin(angle), np.cos(angle)]]
        )
        point = np.array(point)
        rotated = np.dot(rotateX, point.reshape(3, 1))
        rotated = rotated.tolist()
        return [rotated[0][0],rotated[1][0],rotated[2][0]]


    def rotateY(self, point, angle):
        rotateY = np.matrix(
            [[np.cos(angle), 0, -np.sin(angle)],
            [0, 1, 0],
            [np.sin(angle), 0, np.cos(angle)]]
        )
        point = np.array(point)
        rotated = np.dot(rotateY, point.reshape(3, 1))
        rotated = rotated.tolist()
        return [rotated[0][0],rotated[1][0],rotated[2][0]]


    def rotateZ(self, point, angle):
        rotateZ = np.matrix(
            [[np.cos(angle), np.sin(angle), 0],
            [-np.sin(angle), np.cos(angle), 0],
            [0, 0, 1]]
        )
        point = np.array(point)
        rotated = np.dot(rotateZ, point.reshape(3, 1))
        rotated = rotated.tolist()
        return [rotated[0][0],rotated[1][0],rotated[2][0]]


    def translate(self, point, vec):
        tx, ty, tz = vec[0], vec[1], vec[2]
        translated = [point[0] + tx, point[1] + ty, point[2] + tz]
        return translated


    def draw(self):
        # self.angle = pg.mouse.get_pos()[0] / 1000
        self.angle += 0.01
        self.screen.fill(WHITE)
        points = [[3, -1, 3], [5, -1, 3], [5, 1, 3], [3, 1, 3], [3,
    -1, -3], [5, -1, -3], [5, 1, -3], [3, 1, -3]]
```

```
        #[0,0,0],[-1,1,1],[1,1,1],[-1,-1,1],[1,-1,1],[-1,1,-1],[1,1,-1],[-1,-1,-1],[1,-1,-1]
        projected_points = [[n, n] for n in range(len(points))]
        i = 0
        for point in points:
            rotated = self.rotateY(point, self.angle)
            #rotated = self.rotateX(rotated, self.angle)
            #rotated = self.translate(rotated, [0,0,-self.angle*5])
            projected = self.project(rotated)
            x = projected[0] * self.scale + self.H_WIDTH
            y = projected[1] * self.scale + self.H_HEIGHT
            projected_points[i] = x, y
            pg.draw.circle(self.screen, BLACK, (x, y), 2)
            i += 1


        for p in range(4):
            self.connect_points(p, (p+1) % 4, projected_points)
            self.connect_points(p+4, ((p+1) % 4) + 4, projected_points)
            self.connect_points(p, (p+4), projected_points)


    def run(self):
        while True:
            self.draw()

            [exit() for i in pg.event.get() if i.type == pg.QUIT]
            pg.display.set_caption(f"{round(self.clock.get_fps())} FPS")
            pg.display.flip()
            self.clock.tick(self.FPS)


if __name__ == '__main__':
    app = App()
    app.run()
```

Chapter 36

Prototype 2

Unlike the first prototype, this program uses my own matrix class to deal with the matrices; furthermore, this program also implements a perspective projection matrix. As you can see in Figure 36.1, the rear face of the cube appears to be smaller. This proves that the perspective is now working.

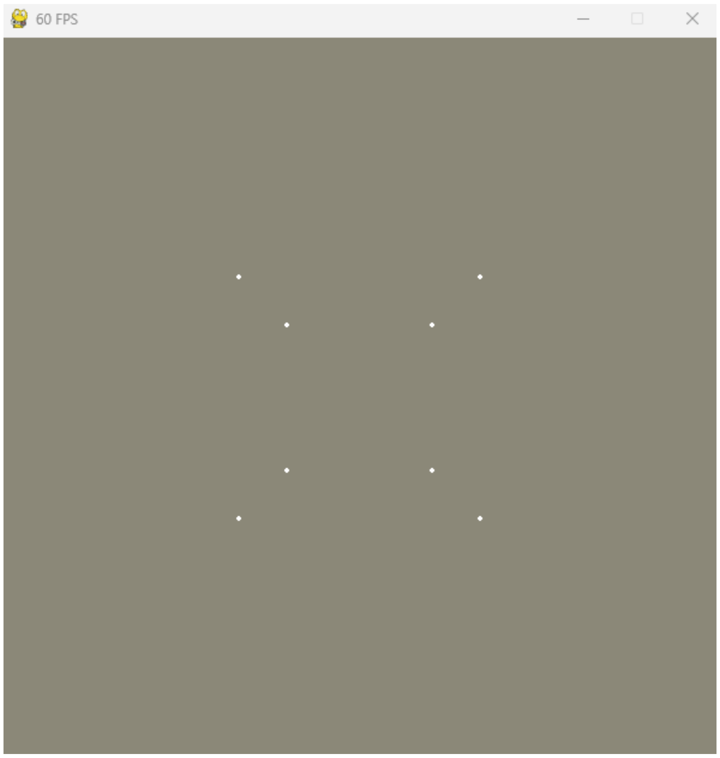


Figure 36.1: Prototype 2

```
import pygame as pg
import math as m
```

```python
from MatrixMath import matrix as mm
from matrices import *
import time as t
from pygame.colordict import THECOLORS


class Camera:
    def __init__(self, app, position):
        self.app = app
        pg.mouse.set_visible(1)

        self.pos = mm.Matrix([[*position, 1]])
        self.forward = mm.Matrix([[0, 0, 1, 1]])
        self.up = mm.Matrix([[0, 1, 0, 1]])
        self.right = mm.Matrix([[1, 0, 0, 1]])

        self.speed = 1
        self.angle = m.radians(1)

    def yaw(self, angle):
        self.up *= rotate_y(angle)
        self.forward *= rotate_y(angle)
        self.right *= rotate_y(angle)

    def pitch(self, angle):
        self.up *= rotate_x(angle)
        self.forward *= rotate_x(angle)
        self.right *= rotate_x(angle)

    def rot_mat(self):
        fx, fy, fz, fw = self.forward[0]
        rx, ry, rz, rw = self.right[0]
        ux, uy, uz, uw = self.up[0]

        return mm.Matrix(
            [
                [rx, ux, fx, 0],
                [ry, uy, fy, 0],
                [rz, uz, fz, 0],
                [0, 0, 0, 1]
            ]
        )

    def trans_mat(self):
        x, y, z, w = self.pos[0]

        return mm.Matrix(
```

```
            [
                [1, 0, 0, 0],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [-x, -y, -z, 1]
            ]
        )

    def cam_mat(self):
        return self.trans_mat() * self.rot_mat()

    def movement(self):
        key = pg.key.get_pressed()

        if key[pg.K_a]:
            self.right = self.speed * self.right
            self.pos = self.pos - self.right
        if key[pg.K_d]:
            self.right = self.speed * self.right
            self.pos = self.pos + self.right
        if key[pg.K_w]:
            self.forward = self.speed * self.forward
            self.pos = self.pos + self.forward
        if key[pg.K_s]:
            self.forward = self.speed * self.forward
            self.pos = self.pos - self.forward

        if key[pg.K_LEFT]:
            self.yaw(-self.angle)
        if key[pg.K_RIGHT]:
            self.yaw(self.angle)
        if key[pg.K_UP]:
            self.pitch(-self.angle)
        if key[pg.K_DOWN]:
            self.pitch(self.angle)


class App:
    def __init__(self):
        pg.init()
        self.res = self.width, self.height = 600, 600
        self.hwidth, self.hheight = self.width / 2, self.height / 2
        self.fps = 60
        self.screen = pg.display.set_mode(self.res)
        self.clock = pg.time.Clock()

        self.cam = Camera(self, [0, 0, 0])
```

```

        # vertices of cube
        self.points = [
            mm.Matrix([[-1, 1, 3, 1]]),
            mm.Matrix([[1, -1, 3, 1]]),
            mm.Matrix([[1, 1, 3, 1]]),
            mm.Matrix([[-1, -1, 3, 1]]),
            mm.Matrix([[-1, 1, 5, 1]]),
            mm.Matrix([[1, -1, 5, 1]]),
            mm.Matrix([[1, 1, 5, 1]]),
            mm.Matrix([[-1, -1, 5, 1]]),
        ]

        self.fov = 90
        self.f = 1 / m.tan(m.radians(self.fov / 2))
        self.zf = 1000
        self.zn = 0.1
        self.g = self.zf / (self.zf - self.zn)
        self.a = self.height / self.width
        self.angle = m.radians(1)

        self.proj = mm.Matrix(
            [
                [self.a * self.f, 0, 0, 0],
                [0, self.f, 0, 0],
                [0, 0, self.g, 1],
                [0, 0, -self.zn * self.g, 0],
            ]
        )

    def rotx(self, point):
        rotx = mm.Matrix(
            [
                [1, 0, 0, 0],
                [0, m.cos(self.angle), -m.sin(self.angle), 0],
                [0, m.sin(self.angle), m.cos(self.angle), 0],
                [0, 0, 0, 1],
            ]
        )
        return point * rotx

    def roty(self, point):
        roty = mm.Matrix(
            [
                [m.cos(self.angle), 0, m.sin(self.angle), 0],
                [0, 1, 0, 0],
                [-m.sin(self.angle), 0, m.cos(self.angle), 0],
```

```
                [0, 0, 0, 1],
            ]
        )
        return point * roty

    def rotz(self, point):
        rotz = mm.Matrix(
            [
                [m.cos(self.angle), -m.sin(self.angle), 0, 0],
                [m.sin(self.angle), m.cos(self.angle), 0, 0],
                [0, 0, 1, 0],
                [0, 0, 0, 1],
            ]
        )
        return point * rotz

    def draw(self):
        self.screen.fill(THECOLORS["cornsilk4"])

        for i in range(len(self.points)):
            copy = mm.copy_matrix(self.points[i])
            copy *= self.cam.cam_mat()
            self.project(copy)

    def project(self, point):
        projected = point * self.proj

        x, y, z, w = projected[0]

        if w != 0:
            x /= w
            y /= w
            z /= w
            # projected[0][3] /= w

            if (x < 2 and x > -2) and (y < 2 and y > -2) and not (w
   < 0):
                x, y = (x + 1) * self.hwidth, (y + 1) * self.hheight
                pg.draw.circle(self.screen, (255, 255, 255), (x, y),
    (2))
                # self.screen.set_at((int(x), int(y)), (0))

    def run(self):
        while True:
            self.draw()
            self.cam.movement()

```

```
            [exit() for i in pg.event.get() if i.type == pg.QUIT]
            pg.display.set_caption(f"{round(self.clock.get_fps())}
   FPS")
            pg.display.flip()
            self.clock.tick(self.fps)


if __name__ == "__main__":
    app = App()
    app.run()
```

Part IV

Testing

Chapter 37

Testing strategy

Since the user will mainly be able to create objects and customize their parameters, the testing that I will do will be on the `__init__()` methods of the classes. In these tests I will make sure that if the user enters erroneous data, an exception will be raised with an error message so that they know instantly when and where their error occurred. The library also gives the user the option to use keyboard and mouse input whilst their program is running therefore I will test each physical input to make sure that it does what its meant to do. The test data that I will use will mainly be the erroneous and normal data because there aren't many places where boundary data can be applied.

Chapter 38

Test video link

Link to video:

`https://www.youtube.com/watch?v=vFAq4WTou7Q`

Chapter 39

Test table

Test No.-objective	Purpose	Description	Test data	Expected result	Actual result	Evidence
1.1-1(a)(b)	To see whether user can create a point object and see it on screen	Class should project a 3D point	Point(app, [0, 0, 1])	Circle drawn in the middle of the screen	Circle drawn in the middle of the screen	Figure 41.1
1.2-1	tests for coordinate parameter	when user inputs a coordinate it should only accept a list type	Point(app, "1, 3, 4")	raise an exception telling the user that the data type is wrong	Exception raised and program turned off	Figure 41.2
1.3-1	tests for coordinate parameter	when user inputs a coordinate it should only accept a list with 3 numbers	Point(app, [1, 3, 4, 2])	raise an exception telling the user that coordinate length should be 3	Exception raised and program turned off	Figure 41.3
1.4-1	tests for vertex parameter	when user inputs vector flag it should only accept a Boolean value	Point(app, [0, 0, 1], True)	Point drawn	Point drawn	Figure 41.4
1.5-1	tests for vertex parameter	when user inputs vector flag it should only accept a Boolean value	Point(app, [0, 0, 1], False)	Point not drawn and no errors	Point drawn and no errors	Figure 41.5
1.6-1	tests for vertex parameter	when user inputs vector flag it should only accept a Boolean value	Point(app, [0, 0, 1], 1)	Exception raised	Exception raised	Figure 41.6
2.1-2(bi)	To see if W key can be used to move to the forwards	W key should translate camera forwards	Press W key	All objects should get bigger as they get closer	All objects get bigger	Time stamp: 2.58-3.23
2.2-2(biii)	To see if S key can be used to move to the backwards	S key should translate camera backwards	Press S key	All objects should get smaller as they get further away	All objects get smaller	Time stamp: 2.58-3.23
2.3-2(bii)	To see if A key can be used to move to the left	A key should translate the camera to the left	Press A key	All objects move to the right	All objects move to the right	Time stamp: 2.58-3.23
2.4-2(biv)	To see if D key can be used to move to the right	D key should translate the camera to the right	Press D key	All objects move to the left	All objects move to the left	Time stamp: 2.58-3.23
2.5-2(bv)	To see if E key can be used to move to upwards	E key should translate the camera to upwards	Press E key	All objects move down	All objects move down	Time stamp: 4.49-5.05
2.6-2(bvi)	To see if Q key can be used to move downwards	Q key should translate the camera down	Press Q key	All objects move up	All objects move up	Time stamp: 4.49-5.05
3-2(c)	To see if mouse input can be used to look around the scene	Pygame detects relative mouse movement and then camera axis are multiplied with x or y rotation matrices	Mouse movement	Moving the mouse will emulate looking around the scene with your eyes	If mouse is moved to the right the object is moved to the left, if mouse is moved to the left the object is moved to the right, if mouse is moved up the object is moved downwards, if mouse is moved down the object is moved upwards	Time stamp: 9.35-10.0
4-3(b)	To see if the parameters are functional	The shape generates the vertices depending on the position and size that the user inputs	Cube(5, [5, 0, 10], Pyramid(1, [-4, 0, 10]), Tetrahedron(3, [0, 0, 10])	3 Shapes all next to each other in decreasing size from right to left should be drawn on the screen	Three shapes of identical size all placed in the center of the screen	Figure 41.7

5-4	To see if the user can use scroll wheel as input	Pygame detects scroll wheel input and fov attribute is incremented by 1 and the projection matrix is updated	Scroll wheel	The object is zoomed in when fov decreases and zooms out when fov increases	The object is zoomed in when fov decreases and zooms out when fov increases	Time stamp: 4.07-4.21
6-5	To see if user can change dimensions of window whilst program is running	Pygame will get the dimensions of the screen every frame and if they change, the projection matrix is updated	User changes window size by dragging sides of window with mouse	When screen is resized, all objects keep the same relative position	When screen is resized, all objects keep the same relative position	Time stamp: 3.24-3.54
7-6	To see if user can input a .obj file path and see it render on the screen	The model class will open the obj file and generates the points and triangles based on the contents of the file	Model(app, "obj/cube.obj")	The screen should display a cube	A cube with all the triangles is drawn	Time stamp: 2.34-2.57
8-8	To see if the user will be able to see the frame rate on the top bar of the window when it's not in full screen	The Pygame `display.setcaption()` function will be used to put a title on the window and the `get_fps()` function will be used to get the current number of frames		Around 60 FPS should be displayed on the top bar of the screen at all times	A number that flickers from 50-60 is displayed at the top of the screen	Time stamp: 1.06-1.10
9-9	The library should have minimal setup	Pygame has a lot of initialising that needs to be done by the user but my library should have much less setup	Figure 41.9	When program is run an empty window should appear	When the program was run an empty window appeared therefore the library has 3 lines of setup and still works as intended	Figure 41.10
10-10(b-d)	To see if user can view stats on screen: FOV, FPS, dimensions	The display stats method should print all stats on the screen	`app.display_stats()`	Stats should be displayed in white on screen when background color is black.	Stats displayed in white on screen	Figure 41.11
10.1-10(a)	To see if by pressing "o" user can see FOV, FPS and dimensions on screen	When the "o" key is pressed, the display stats method should be called every frame	Press "o" twice	Stats are displayed and then go away	Stats are displayed but then stay	After pressing "o" once: Figure 41.12. After pressing "o" a second time: Figure 41.13.
11-11	To see if by pressing escape key, window is closed	Pygame check events detects whether escape key is pressed calls exit() function	Escape key	program should stop running	program stops running	Time stamp: 1.13-1.21
12.1-12	To see if background color is opposite line color when it isn't specified	background color gets set to the opposite of line color so that lines are always visible	`LINE_COLOR` set to (0, 0, 0)	background is white when program is run	background is white	Figure 41.14
12.2-12	To see if line color is opposite background color when it isn't specified	line color gets set to the opposite of background color so that lines are always visible	`BG_COLOR` set to (255, 255, 255)	line color is black when program is run	line color is black	Figure 41.15 and Time stamp: 8.22-8.58
12.3-12	To see if background color and line color still works when neither are specified	background color and line color default to black and white respectively	run library program with a cube on the screen with no parameters for color	Screen should be black and cube should be white	screen is black and cube is white	Figure 41.16
13-13	To see if color of text of stats is opposite of background color	When stats are displayed, font color is made as opposite of background	`BG_COLOR`=(255,255,0)	Text is blue since background is yellow	Text is blue since background is yellow	Figure 41.17 Time stamp: 9.18-9.26
14.2	Test if normal size obj models reduce frame rate of program	The library projects each point individually using a for loop therefore very large models could slow down the real time render	sphere.obj	This is a normal size model with average amount of points so I expect the model to run at 60 frames per second	The program runs at 60 frames per second	Figure 41.18
14.2	Test if very large obj models reduce frame rate of program	The library projects each point individually using a for loop therefore very large models could slow down the real time render	cow.obj	This is a very large model with many points so I expect the model to be very slow	The program runs at zero frames per second	Figure 41.19
15.1	Test dimensions input for App class	dimensions stores height and width of window	(400, 100)	No error message and screen with dimensions (400, 400)	No error message and screen with dimensions (400, 400)	41.20
15.2	Test dimensions input for App class	dimensions stores height and width of window	[400, 400]	No error message and screen with dimensions [400, 400]	No error message and screen with dimensions [400, 400]	41.20
15.3	Test dimensions input for App class	dimensions stores height and width of window	"hello world"	error message displayed	error message displayed	41.21
15.4	Test dimensions input for App class	dimensions stores height and width of window	10	error message displayed	error message displayed	41.21

15.5	Test dimensions input for App class	dimensions stores height and width of window	[100, 100, 100]	error message displayed	error message displayed	41.22
15.6	Test dimensions input for App class	dimensions stores height and width of window	[[100], 100, 100]	error message displayed	error message displayed	41.22
16.1	Test `cam_pos` input for App class	`cam_pos` stores the initial position of the camera	[0, 0, 0]	camera placed at [0, 0, 0]	camera placed at [0, 0, 0]	41.23
16.2	Test `cam_pos` input for App class	`cam_pos` stores the initial position of the camera	(0, 0, 0)	camera placed at (0, 0, 0)	camera placed at (0, 0, 0)	41.23
16.3	Test `cam_pos` input for App class	`cam_pos` stores the initial position of the camera	10	error message displayed	error message displayed	41.24
16.4	Test `cam_pos` input for App class	`cam_pos` stores the initial position of the camera	[[0], 0, 0]	error message displayed	error message displayed	41.25
17.1	Test `VERTEX_SIZE` input for App class	`VERTEX_SIZE` stores size of the points	5	no error message and point drawn	no error message and point drawn	41.26
17.2	Test `VERTEX_SIZE` input for App class	`VERTEX_SIZE` stores size of the points	”5”	error message displayed	error message displayed	41.27

Chapter 40

Test solutions

Test number	Solution	Evidence
4-3(b)	When cube, tetrahedron and pyramid inherited shape, the super function was used to get all attributes but they were all set with a default value therefore no matter what the parameter was, the shapes were always at the center of the screen with size 1. To fix it I removed the default values.	Figure 41.8 SHAPE LINES OF CODE
10.1-10(a)	The `display_stats()` method was called every frame and if the stats flag was True, the stats would be displayed. By default the stats flag is set to False so that nothing is displayed. When the "o" key is pressed, the stats flag was changed to True. This successfully displays the stats but when pressed again nothing happened. To fix this I added an extra variable called `check_stats`. Every time "o" is pressed the variable is incremented by one. Then the modulus division by 2 is done to the variable to check if it's an even number. If its even, stats is set to True, if its odd, stats is set to False. This successfully displays and removes the stats when "o" is pressed multiple times.	PUT VIDEO TIME STAMP HERE

Chapter 41

Test Images

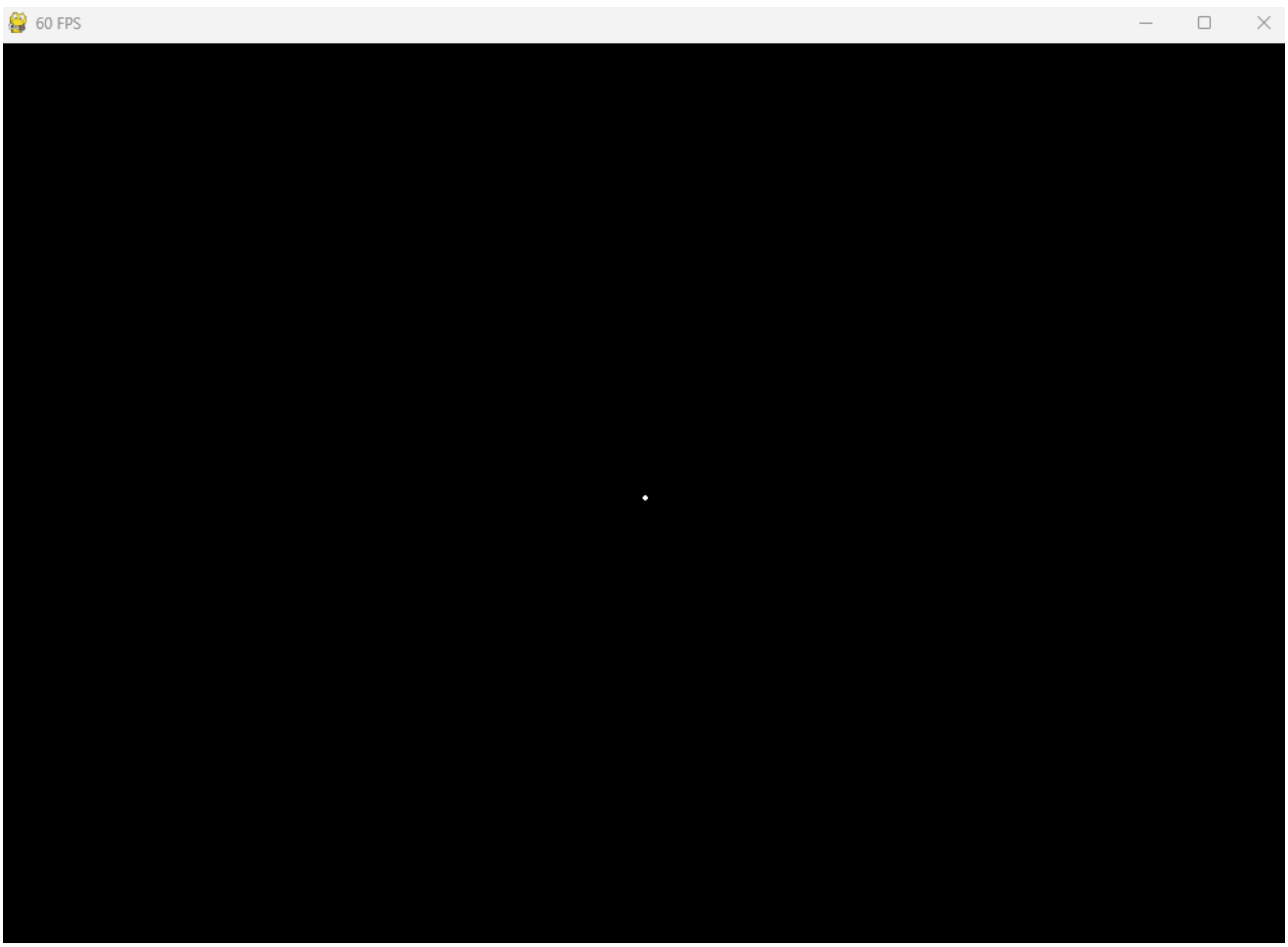


Figure 41.1: Test 1.1 - Point drawn

```
PS C:\git-repos> python .\libtest.py
pygame 2.0.1 (SDL 2.0.14, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\git-repos\libtest.py", line 5, in <module>
    Point(app, '1, 3, 4')
  File "C:\git-repos\Apg3d\point.py", line 18, in __init__
    raise Exception("Coordinate must be list")
Exception: Coordinate must be list
```

Figure 41.2: Test 1.2 - Exception

```
PS C:\git-repos> python .\libtest.py
pygame 2.0.1 (SDL 2.0.14, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\git-repos\libtest.py", line 5, in <module>
    Point(app, [1, 3, 4, 2])
  File "C:\git-repos\Apg3d\point.py", line 20, in __init__
    raise Exception("Coordinate must have 3 items")
Exception: Coordinate must have 3 items
```

Figure 41.3: Test 1.3 - Exception

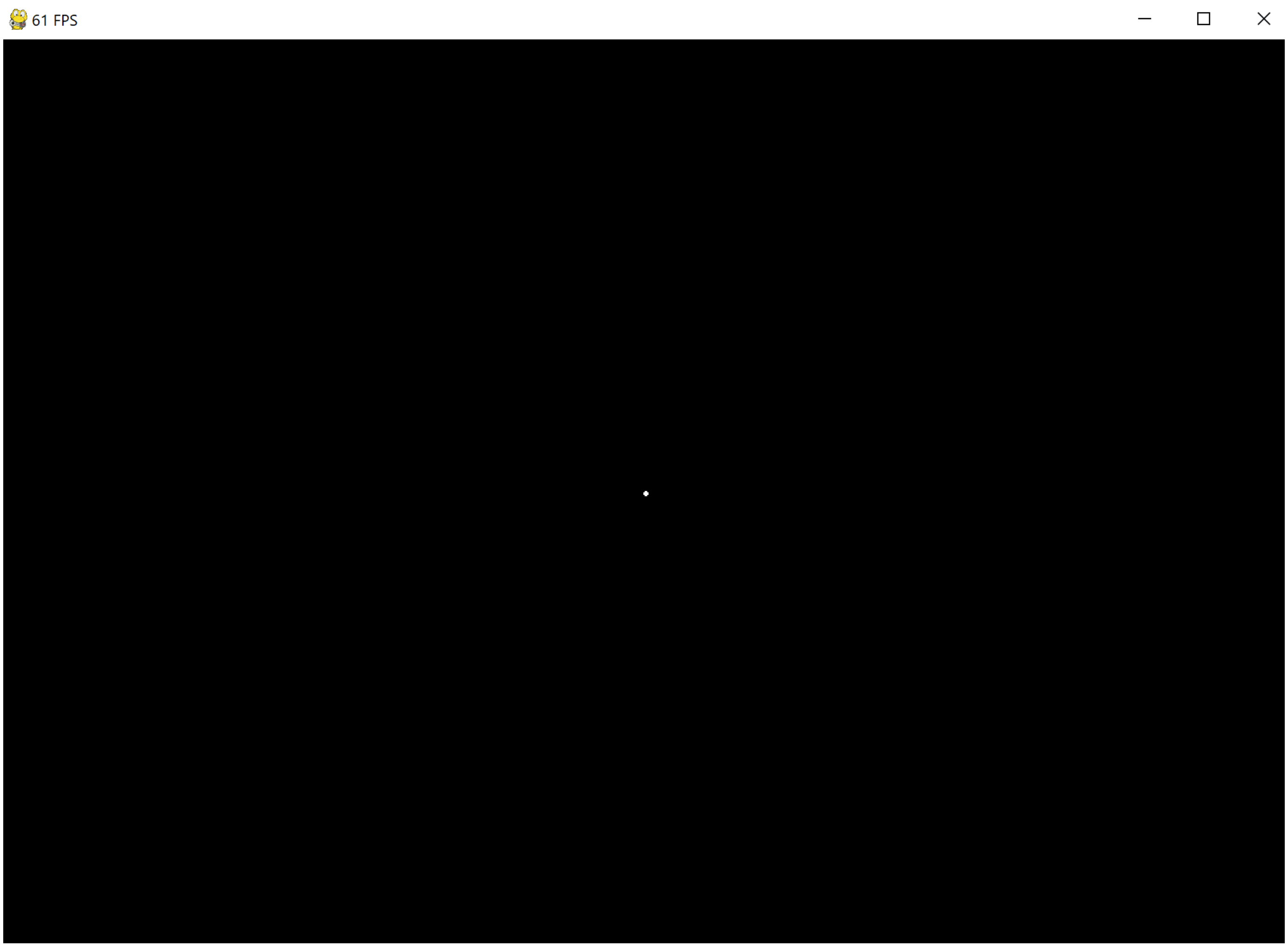


Figure 41.4: Test 1.4 - Point drawn

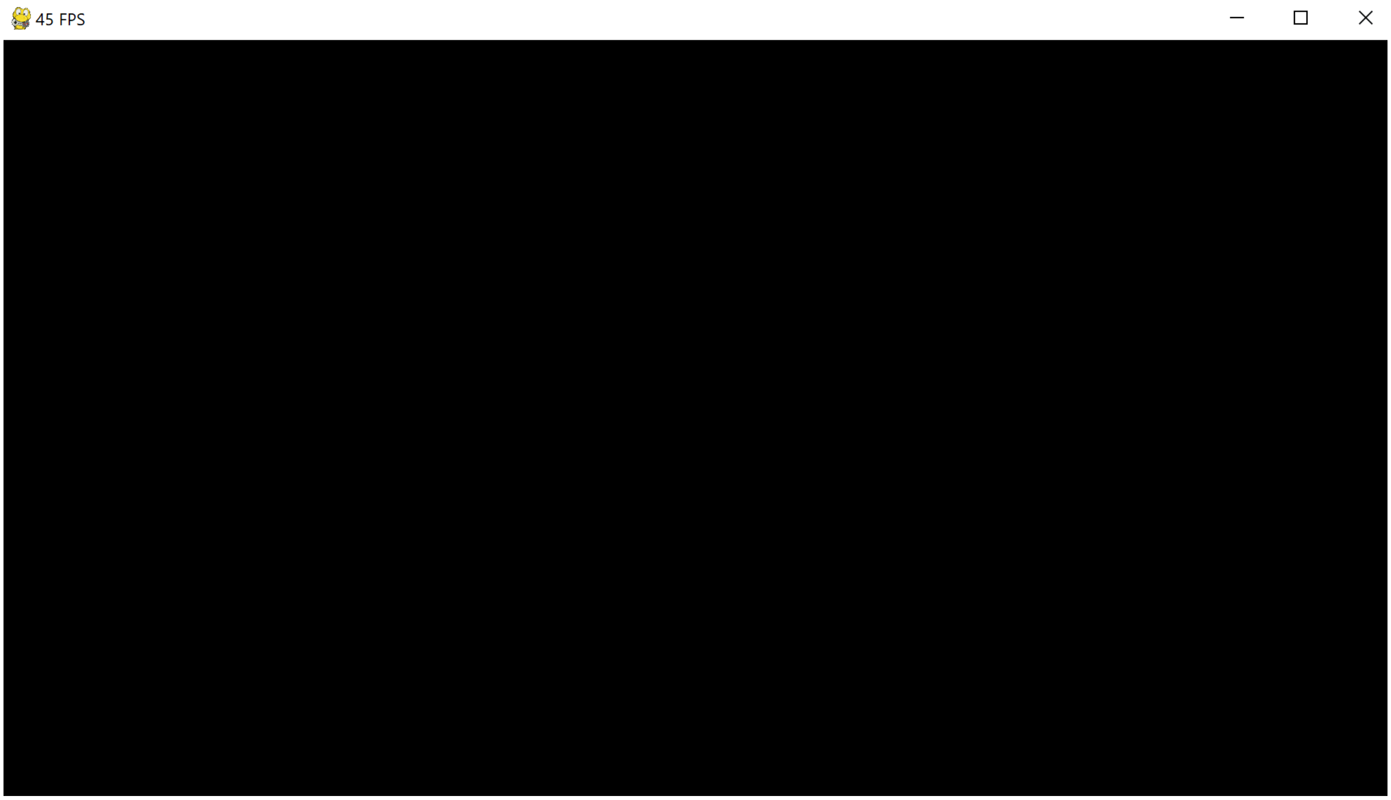


Figure 41.5: Test 1.5 - Point not drawn

```
PS C:\git-repos> python .\libtest.py
pygame 2.0.1 (SDL 2.0.14, Python 3.9.13)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\git-repos\libtest.py", line 5, in <module>
    Point(app, [0, 0, 1], 1)
  File "C:\git-repos\Apg3d\point.py", line 22, in __init__
    raise Exception("Vertex type must be bool")
Exception: Vertex type must be bool
```

Figure 41.6: Test 1.6 - Exception

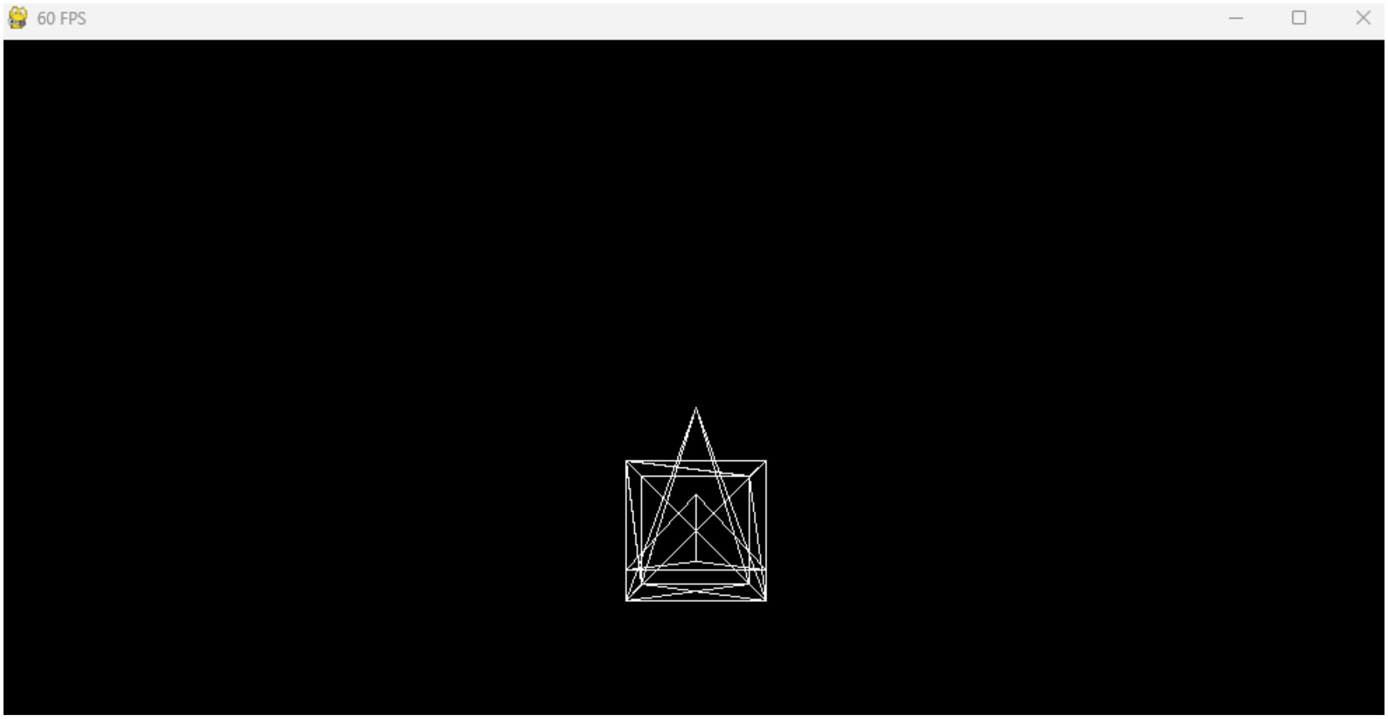


Figure 41.7: Test 4 - Fail

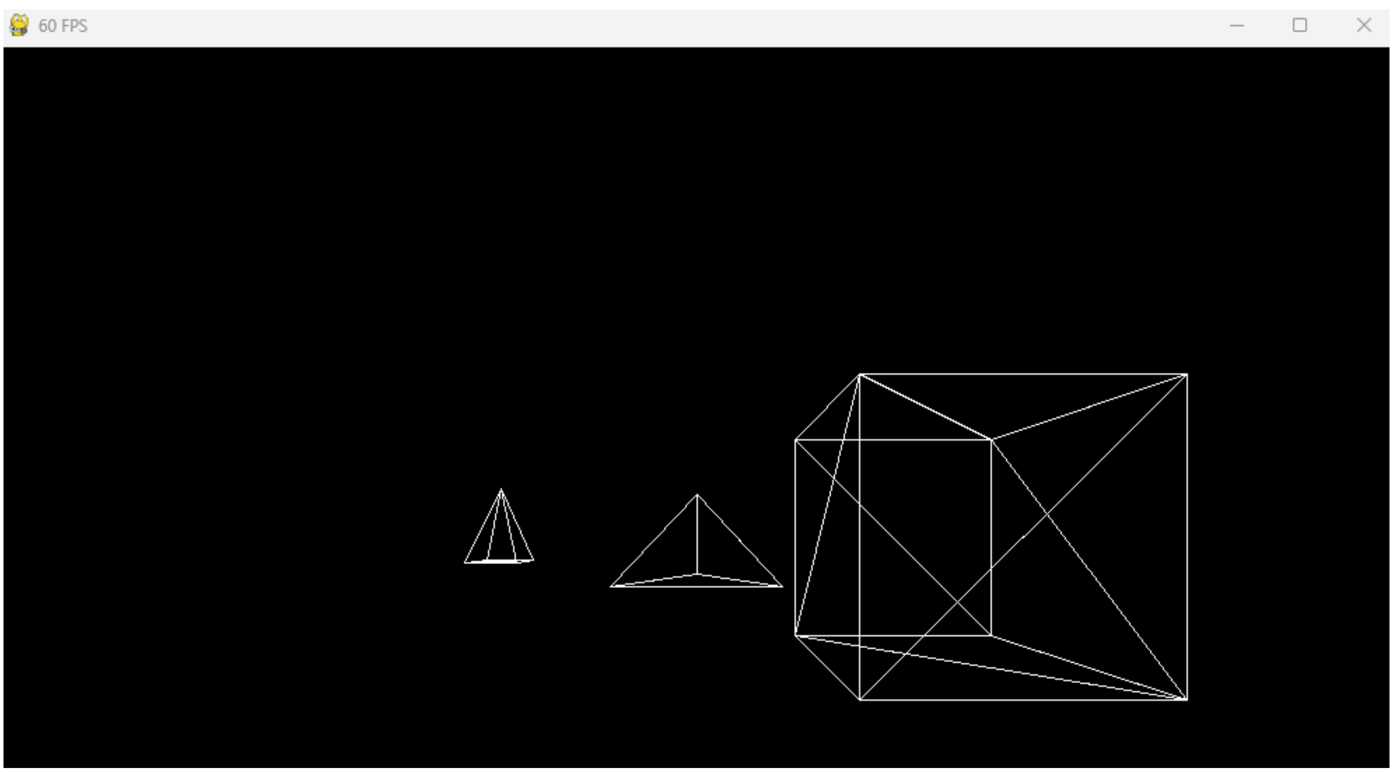


Figure 41.8: Test 4 - Fix

```python
from pg3d import *

app = App()

app.run()
```

Figure 41.9: Test 9 - Test data

Figure 41.10: Test 9 - Evidence

Figure 41.11: Test 10 - Evidence

Figure 41.12: Test 10.1 - o pressed once

Figure 41.13: Test 10.1 - o pressed a second time

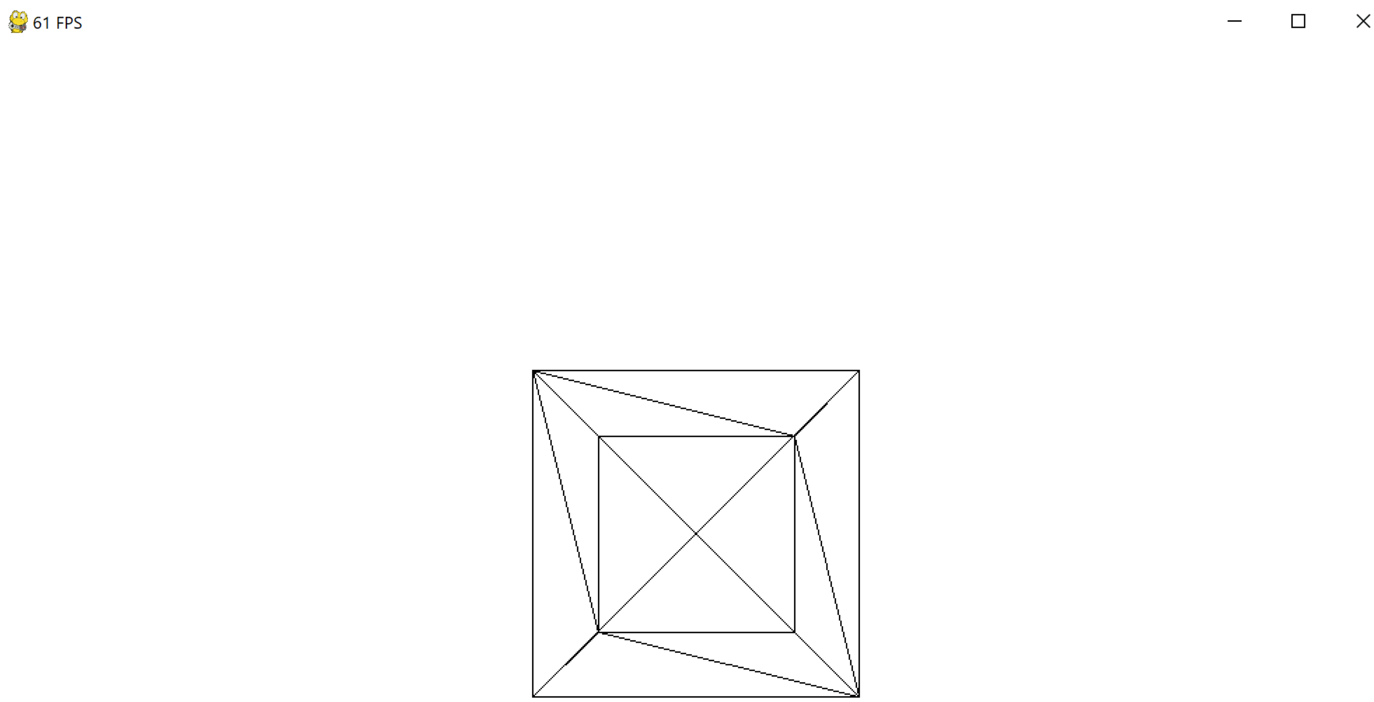


Figure 41.14: Test 12.3 - line color set to black

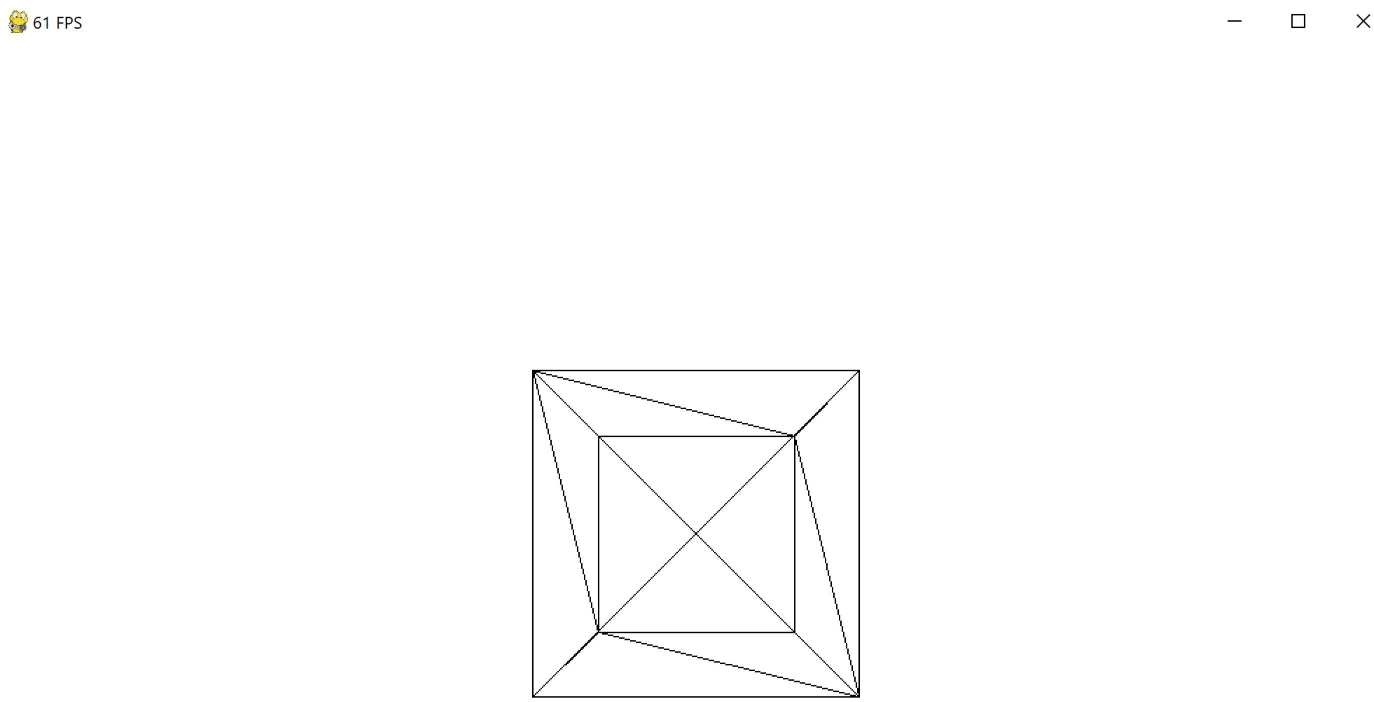


Figure 41.15: Test 12.2 - background color set to white

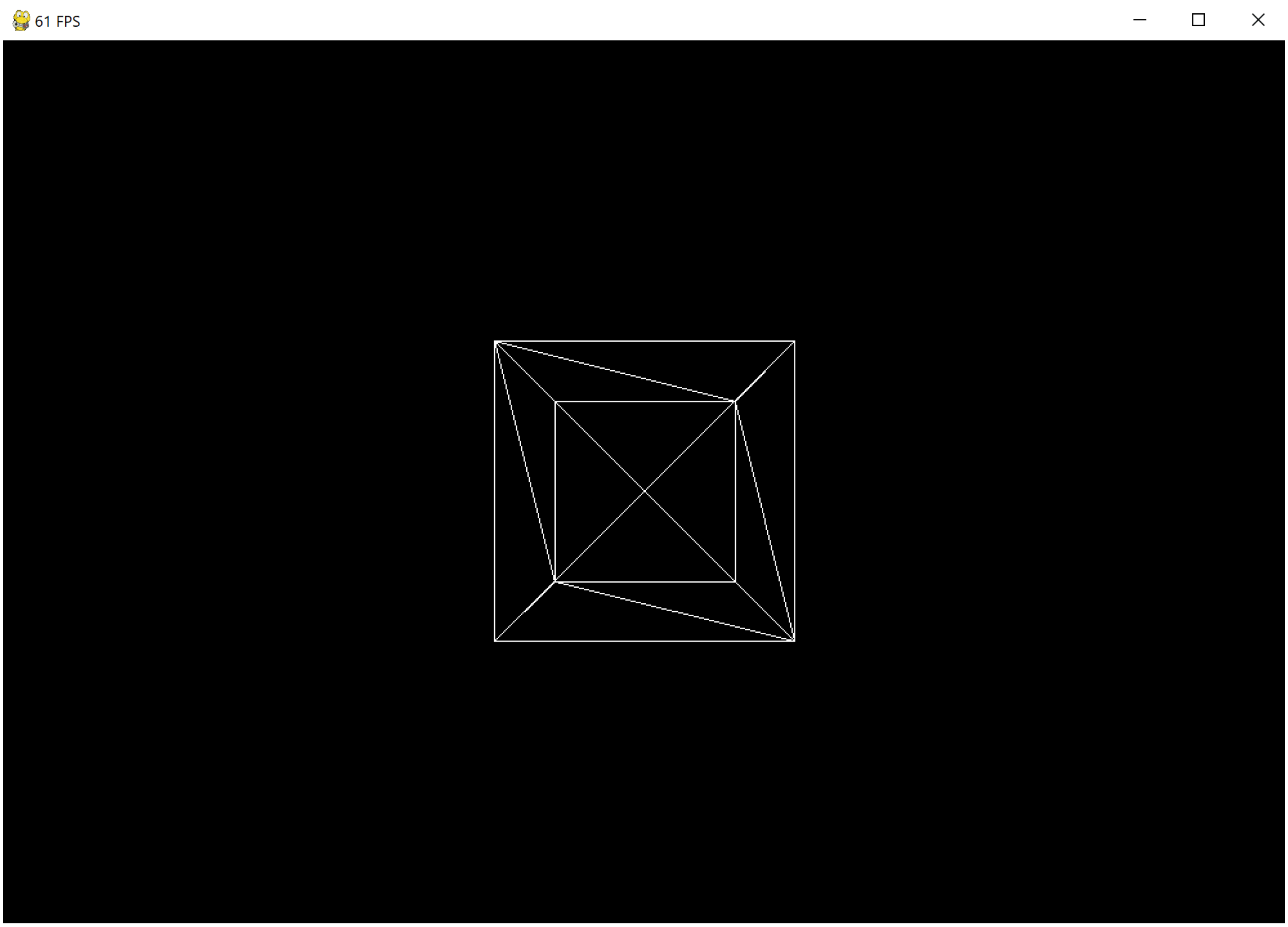


Figure 41.16: Test 12.3 - no color parameters

Figure 41.17: Test 13 - statistics font color

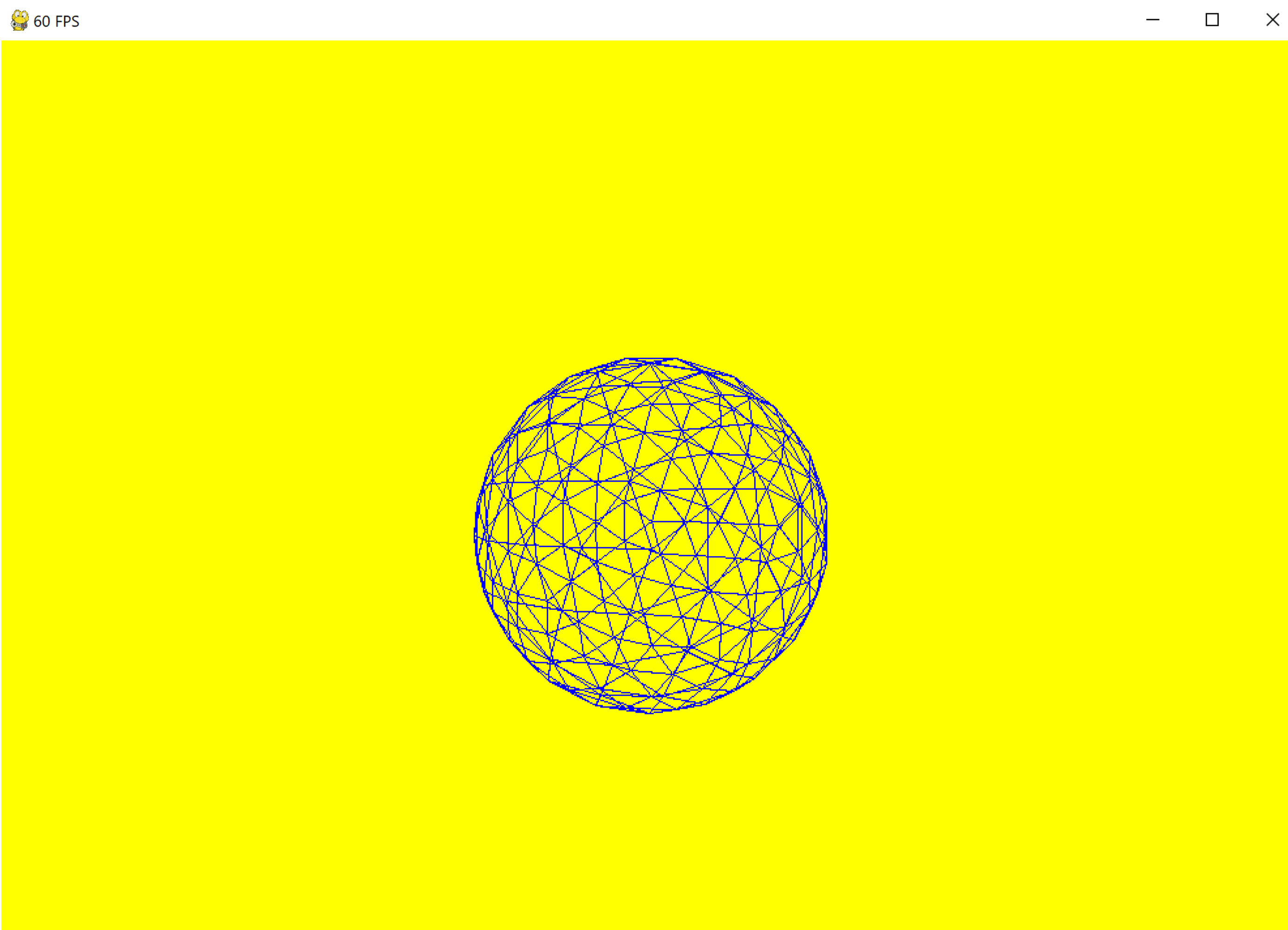


Figure 41.18: Test 14.1 - small models

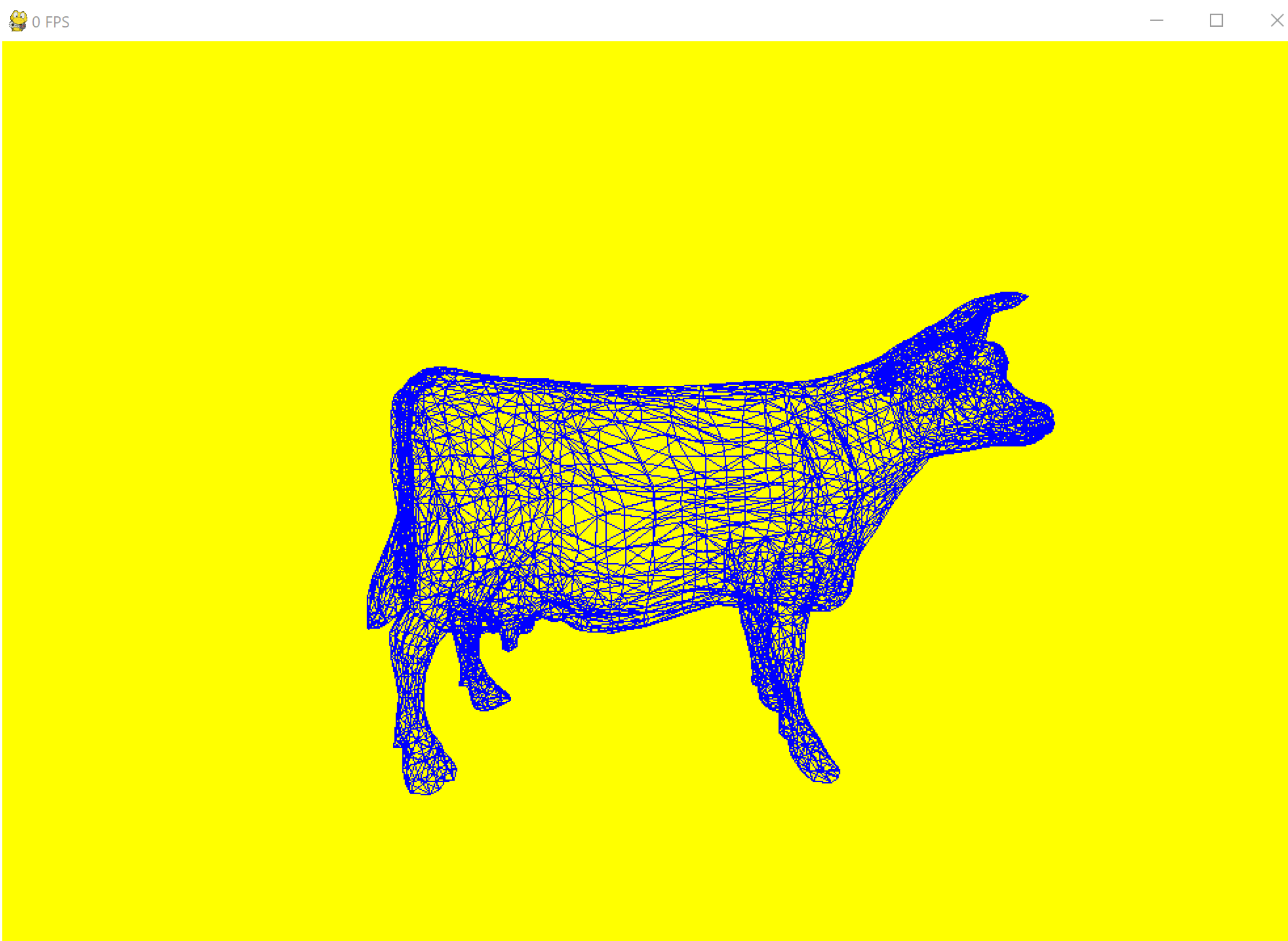


Figure 41.19: Test 14.2 - large models

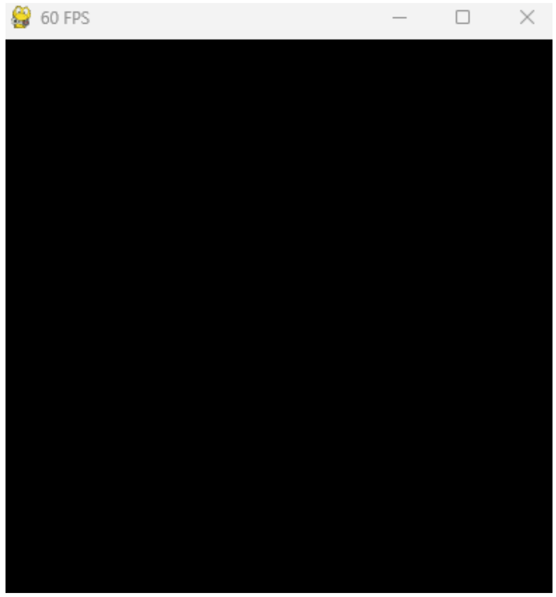
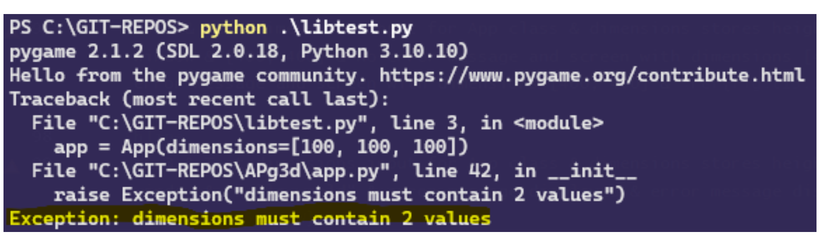


Figure 41.20: Test 15.1 and 15.2 - tuple and list

```
PS C:\GIT-REPOS> python .\libtest.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.10)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\GIT-REPOS\libtest.py", line 3, in <module>
    app = App(dimensions="hello world")
  File "C:\GIT-REPOS\APg3d\app.py", line 36, in __init__
    raise Exception("dimensions must be list or tuple")
Exception: dimensions must be list or tuple
```

Figure 41.21: Test 15.3 and 15.4 - string and number



```
PS C:\GIT-REPOS> python .\libtest.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.10)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\GIT-REPOS\libtest.py", line 3, in <module>
    app = App(dimensions=[100, 100, 100])
  File "C:\GIT-REPOS\APg3d\app.py", line 42, in __init__
    raise Exception("dimensions must contain 2 values")
Exception: dimensions must contain 2 values
```

Figure 41.22: Test 15.5 - boundary data

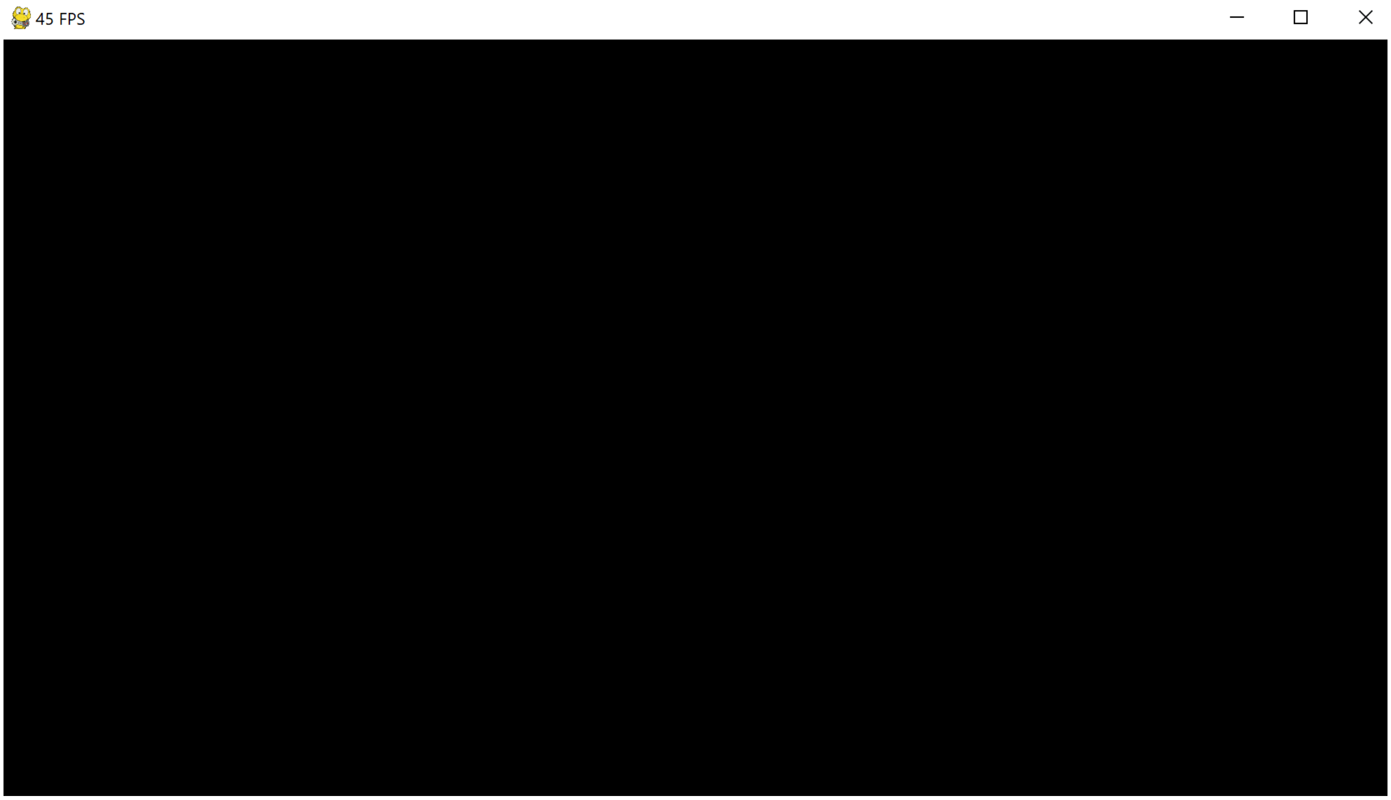
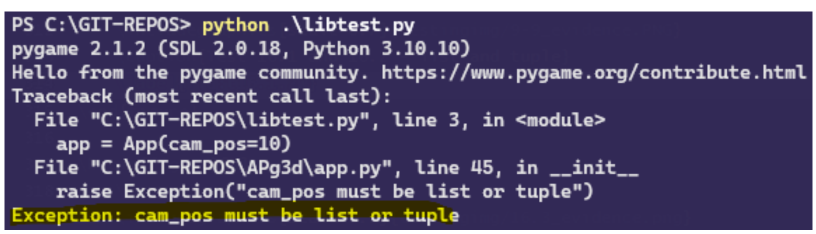
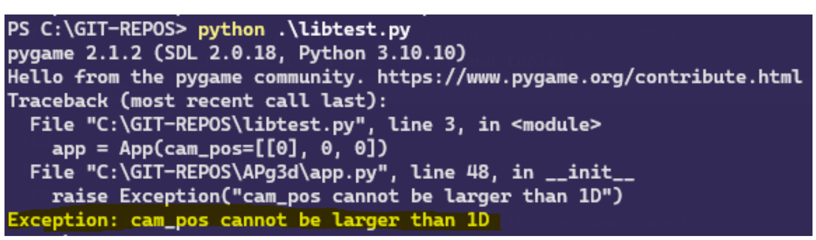


Figure 41.23: Test 16.1 and 16.2 - list and tuple



```
PS C:\GIT-REPOS> python .\libtest.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.10)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\GIT-REPOS\libtest.py", line 3, in <module>
    app = App(cam_pos=10)
  File "C:\GIT-REPOS\APg3d\app.py", line 45, in __init__
    raise Exception("cam_pos must be list or tuple")
Exception: cam_pos must be list or tuple
```

Figure 41.24: Test 16.3 - erroneous data



```
PS C:\GIT-REPOS> python .\libtest.py
pygame 2.1.2 (SDL 2.0.18, Python 3.10.10)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\GIT-REPOS\libtest.py", line 3, in <module>
    app = App(cam_pos=[[0], 0, 0])
  File "C:\GIT-REPOS\APg3d\app.py", line 48, in __init__
    raise Exception("cam_pos cannot be larger than 1D")
Exception: cam_pos cannot be larger than 1D
```

Figure 41.25: Test 16.3 - 2D array

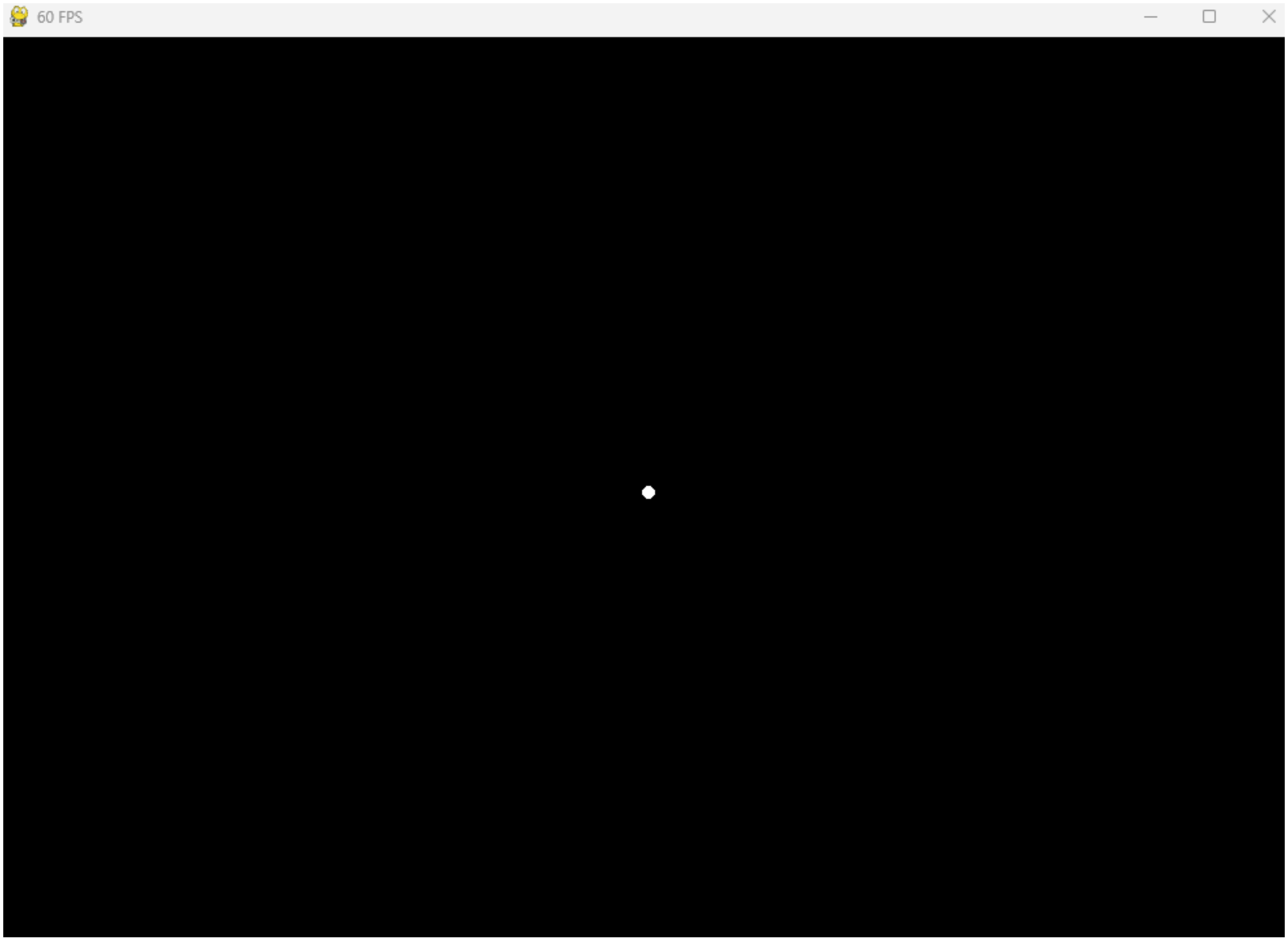


Figure 41.26: Test 17.1 - normal data

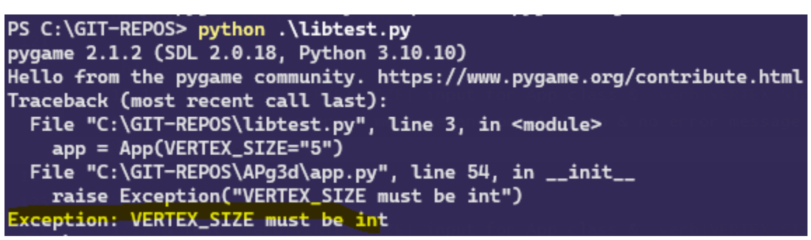


Figure 41.27: Test 17.2 - erroneous data

Chapter 42

Test evaluation

I tested all my objectives, the user input whilst the program was running and the classes that the user will use. I chose this method because the other modules will never be used by the user so human error is impossible to occur. All the objectives passed their tests and proved to work. The only test that failed and couldn't be solved was test number 14.2. This test showed that if a very large obj file was inputted, the frame rate would tank to 0. This is because there was a massive amount of triangles that had to be projected and drawn every frame. Since the draw method uses a for loop to project and draw all the points and triangles, the time complexity of the program is O(n). This is without considering the $O(n^2)$ time complexity for matrix multiplication, addition and subtraction, and the $O(n!)$ time complexity for finding the inverse of a matrix. These operations are done on each triangle and point which significantly slows down the program when there are a lot of them. A solution to this would be to use python just in time compilers such as the one provided in the Numba library. This JIT complier comes in the form of a decorator and it turns Python code into C code. The only issue is that it doesn't work if you use objects that it doesn't recognise; this includes my Point, Triangle and Matrix objects.

Part V

Evaluation

Chapter 43

Feedback

After performing another interview with my end user, Mr.S, I have come back with the following positives and negatives of the library.

The first positive is that importing 3D models is very easy to use and that this feature "makes programs more special"[4]. Furthermore, the ability to change the FOV and window size "has proved to be very useful in programs"[4]. Lastly, the end user liked how simple and complex the library was at the same time.

The main negative of the library is that "it's performance deteriorates when you add too many vertices and triangles"[4]. This was expected due to the time complexity of the algorithms used in the library. Lastly, Mr.S stated that he did not like that "there was no option to remove the see through aspect of the objects"[4]. Mr.S is referring to the wire frame look of the program and how he didn't like how you couldn't turn it off and have faces that are behind another face not be drawn.

Lastly, Mr.S recommended some improvements for the library. The first improvement that was recommended was to "improve the performance of the library"[4]. This could be done by writing the library in C so that it can still be used in python, but it has the speed advantages of the compiled code written in C. Mr.S also added that he would like the ability to "fill in the shapes"[4] to remove the transparency wire frame effects. This could be done by implementing a back-face culling or painters algorithm to not render hidden vertices. This also means that a triangle clipping algorithm would need to be implemented so that triangles don't fully disappear when only a part of them is out of the screen.

Chapter 44

Review of objectives

1. Specific objective: The program should allow the user to draw a 3D point.

   The objective was met, as evidenced by test 1, the class works as intended and also raises exceptions when the user inputs data that is not accepted. This made sure that there was no confusion in the use of the class because the exception would give a clear error message that described what the user did wrong. This objective was solved very well because no drawbacks came with it.

2. Specific objective: The user should be able to move around the scene.

   The objective was met, as evidenced by test 2, the movement input works as intended and has no way that it can break due to the nature of the Pygame event getting function. Furthermore, since my matrix data structure works as planned, the movement is also seamless and feels just like a first person view.

3. The user should be able to use ready made 3D shapes by using individual classes.

   This objective was met, as evidenced by test 4, these shapes render very well and are very easy to use. However next time, I would add an extra parameter to change the orientation of the shape and methods to rotate and translate the shape within its local space.

4. The user should be able to use the scroll wheel to change FOV.

   This objective was met, as evidenced by test 5, this functionality also has no way for human error to break it because it uses the Pygame event getting function to check for scroll wheel input.

5. The user should be able to change the window size while the program is running.

   This objective was met, as evidenced by test 6, again this functionality has no way for human error to break it because it uses the Pygame event getting function to check for window resizing. This is also shown in the test video.

6. The user should be able to import a .obj file.

   This objective was met, as evidenced by 7, the class raises an exception if the path is not a string therefore the user will know how to fix their error if they make one. next time I will add methods to rotate and translate the model in its local space, I will also add parameters to give the model custom color and finally I will edit the method that generates the triangles and points to also generate the normal's to the faces so that lighting and back-face culling algorithms will work.

7. The library should have a documentation file containing a description of all the classes and their methods.

   This objective was met, as evidenced by chapter 32, the README.md file contains all the classes accessible to the user and all of their public methods, attributes and the parameters that they need to input.

8. The top part of the window of the program should display the Frames per Second (FPS).

   This objective was met, as evidenced by test 8, the FPS is displayed on the top bar and it is rounded to the nearest integer. There is nothing that can be done to improve this feature.

9. The library should have minimal to no setup to initialize the library.

   This objective was met as evidenced by the second interview with the end user [4]. Mr.S said that "initialising the library is the easiest part because it can be done in three lines of code"[4].

10. The user should have the option to see certain statistics displayed on the screen to see what is going on in the background.

    This objective was met as evidenced by test 10. The user can press the O key to toggle the statistics which are the opposite color of the background so that they are always visible. The only improvement that I would add is including more statistics such as hardware performance in the form of graphs.

11. The user should be able to press the escape key to close the window.

    This objective was met, as evidenced by test 11. This feature is very useful because when the user has mouse controls turned on, it is impossible to close the window because you would need to click the x at the top right of the window. The esc key allows the user to leave the program anyways. The only thing I would change is for the esc key to have a pause menu pop up instead of fully quitting the program because the esc key can be easily clicked accidentally. This pause menu would make the users mouse re-appear so that they can either resume or quit the program fully and maybe even access some settings.

12. When the user specifies a background color or line color, the one that hasn't been specified should be the opposite color so that the colors don't collide.

This objective was met as evidenced by test 12. This feature is handy because if the user only specifies one of the two colors, the other one is set to the opposite color so that it is visible.

13. The color of the text for the optional statistics should be the opposite of the background color so that it is always visible.

    This objective was met as evidenced by test 13. This feature is extremely useful because it means that the statistics are always readable by the user no matter what color the background is. Next time I would make the font color update every frame so that if the user changes the background color whilst the program is running, the text is still readable.

Chapter 45

Final reflection

Overall, the project was a success. All the objectives were met and the end user was satisfied with the outcome because it met all of his needs [4]. The library is definitely not perfect to substitute the existing systems that I have talked about previously as its performance is quite low when scaled up. But, since the purpose of the library isn't to be a fully fledged 3D graphics engine, it's performance is good enough to help my user add some simple 3D functionality to Pygame.

The main thing that I have learnt about my project management style is that I need to set deadlines with achievable goals to keep myself motivated to finish the project. This helps me a lot because when I don't work on something for a long time, its hard to get back in the flow.

To wrap things up, I have learnt a lot from working this project because its the largest project I have ever worked on. Furthermore, I have never created a library so packaging my code was a very new experience for me. Although the project is not perfect, I am glad to say that I have met all the initial objects.

Part VI

Appendix A

Interview with Mr.S

Q.1: Which library do you use for developing programs with a graphical user interface (GUI)?

A.1: I have been using the Pygame library extensively for my graphics needs, as it has been my go-to library for some time now. It offers a wide range of features and is easy to use, making it an ideal choice for my projects.

Q.2: Do you think the inclusion of 3D graphics in the Pygame library would enhance its utility?

A.2: Certainly, the Pygame library's functionality is currently limited due to its inability to support 3D graphics. Adding 3D graphics support would be a major improvement for the library, allowing developers to create more immersive and visually appealing games. This would open up a whole new range of possibilities for game developers, allowing them to create more complex and engaging experiences for their players.

Q.3: In terms of library usage, do you favor simplicity at the cost of limited flexibility or complexity that affords greater latitude in programming?

A.3: Personally, I would opt for a more intricate library that allows for greater programming flexibility, despite requiring more time to learn. The benefits derived from mastering a complex library make the learning curve worthwhile.

Q.4: In your opinion, should a 3D graphics library provide pre-built 3D shapes and movement controls for ease of use?

A.4: From my perspective, these features are fundamental for any graphics library, as acquiring the knowledge necessary to construct them independently would be excessively time-consuming. Additionally, I believe that users should be able to employ the 3D graphics library without the need to focus on the underlying mechanics of 3D graphics.

Q.5: In your view, is it necessary for a 3D graphics library to comprise lighting effects?

A.5: As previously stated, while certain critical functions ought to be integrated into the library, the lack of lighting effects may not pose a significant issue since this is merely a basic Pygame extension. Additionally, since Python is not ideal for quick calculations, implementing lighting effects may result in a significant reduction in the program's performance.

Q.6: Should the library allow users to upload and render 3D .obj files on the screen? This would provide a more immersive experience for users, allowing them to visualize their 3D models in real-time and interact with them in a more meaningful way.

A.6: I think that .obj files are an essential part of 3D graphics. They allow users to create their own 3D models and import them into their programs, allowing them to see their world perfectly rendered on the screen. This makes .obj files a key component in creating realistic 3D visuals, and is why it is so important for libraries to support them.

Q.7 Would you like the library to allow the user to change the window size as the program is running?

A.7 Yes, this feature would be very useful when the window isn't in full screen mode as it would allow you to change the window size without needing to restart the program.

Part VII

Appendix B

Second interview with Mr.S

Q.1: How easy is the library to use?

A.1: If you read the documentation provided on the libraries PyPI front page, using the library is quite easy. Initialising the library is the easiest part because it can be done in just three lines of code. Creating objects is also very easy, but I like how you can customize them however you want by specifying the parameters. Also I love how movement is already implemented in the library and I can focus fully on my 3D environment.

Q.2: Is there anything that you don't like about the library?

A.2: The main downside to the library is that it's performance deteriorates quite a bit when you add too many vertices and triangles. I tried to input a large object in my program but I got a steady zero frames per second. This wasn't to big of a setback because I aim to use the library for more simple programs that don't require huge objects. Furthermore, although I like the wire frame look that the program gives, I don't like that there is no option to remove the see through aspect of the objects.

Q.3: What did you like the most about the library?

A.3: My favourite feature of the library is the ability to import 3D models. This feature really brings my programs to life and makes them more special. Furthermore, I like how the library has simple commands which can become very complex when you play around with the parameters. Lastly, the ability to change the FOV with scroll wheel and window size have proved to be very useful in my programs.

Q.4: What could be improved?

A.4: Obviously the performance of the library should be improved, but other than this I would like it if the library would offer more pre-made shapes and a way to fill the triangles with color. This color wouldn't need to have shading but being able to fill the shapes and remove the transparency would make for a great improvement.

Q.5: Are you satisfied with the end product?

A.5: Yes, I am very satisfied with the end product because it meets all of my needs. Since I intend to use the library to make simple games and animations, The performance issues that are encountered with many shapes aren't encountered for me. This library is super useful because I can use all my knowledge of the Pygame library and combine it with the 3D graphics.

Bibliography

[1] Petter Amland. Ursina api reference, 2019.

[2] Greg Attwood. *Matrices*, page 94–121. Pearson Education Limited, 2017.

[3] Daniele Golzio. Appendix a, March 2022.

[4] Daniele Golzio. Appendix b, March 2023.

[5] Rod Pierce. Inverse of a matrix using elementary row operations, 2022.

[6] Guido Rossum. Packaging python projects, 2023.