Simplistic Object Rendering in Python

Arin Hartung
Michael Nuttall

The Importance of 3D Rendering

- 3D rendering bridges the gap between imagination and visual reality.
- It enables the creation of detailed images and animations across industries, like design, architecture, and medicine.
- Advancing rendering improves performance, and interactivity.
- Better techniques push the boundaries of visual storytelling, simulation, and user experience.
- 3D rendering drives innovation in both artistic and technical fields.
- Combines mathematical logic with artistic creativity to make a complete software.

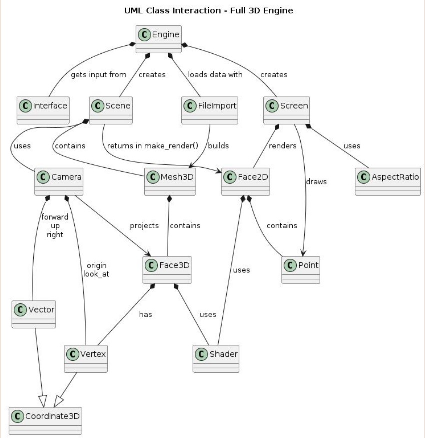
The Project Plan & Requirements

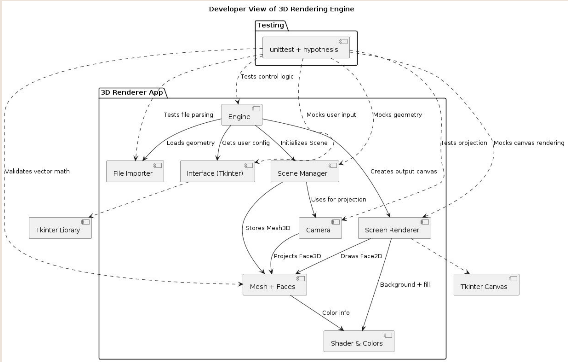
Make a Rendering engine in python that:

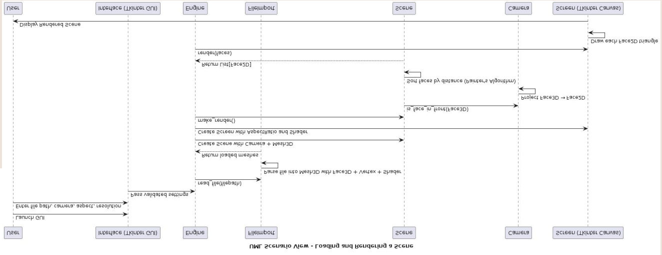
- Defines an object in 3D
- Renders 3D faces in a 2D window
- Uses custom data classes
- Use as few public libraries as possible
- Adheres to Object Oriented Design
- Has been fully tested and is secure

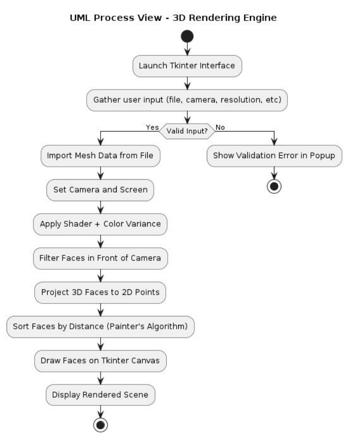
- Represent object to be rendered as a composition of smaller objects.
- Process objects and transform them from 3D to 2D.
- Render objects in order.
- Display objects to screen.

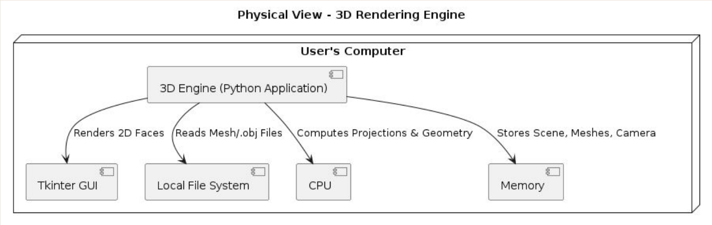
4+1 Views


UML Class Interaction - Full 3D Engine
Engine
gets input from
creates
loads data with
creates
Interface
Scene
FileImport
Screen
uses
contains
returns in make_render()
builds
renders
draws
uses
Camera
Mesh3D
Face2D
AspectRatio
forward
up
right
projects
contains
contains
origin
look_at
Face3D
Point
uses
Vector
has
uses
Vertex
Shader
Coordinate3D

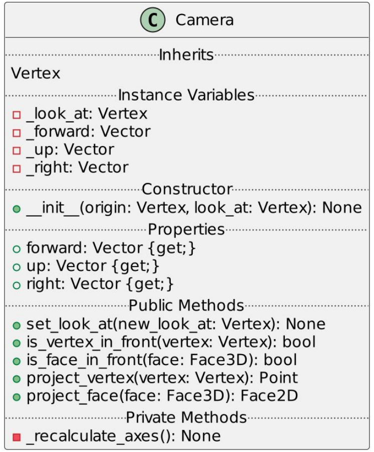

Developer View of 3D Rendering Engine
Testing
unittest + hypothesis
Tests control logic
3D Renderer App
Tests file parsing
Engine
Mocks user input
Mocks geometry
Loads geometry
Gets user config
Initializes Scene
Tests projection
Mocks canvas rendering
Validates vector math
File Importer
Interface (Tkinter)
Scene Manager
Creates output canvas
Uses for projection
Tkinter Library
Stores Mesh3D
Camera
Screen Renderer
Projects Face3D
Draws Face2D
Mesh + Faces
Background + fill
Tkinter Canvas
Color info
Shader & Colors


UML Scenario View - Loading and Rendering a Scene
User
Interface (Tkinter GUI)
Engine
FileImport
Scene
Camera
Screen (Tkinter Canvas)
Launch GUI
Enter file path, camera, aspect, resolution
Pass validated settings
read_file(filepath)
Parse file into Mesh3D with Face3D + Vertex + Shader
Return loaded meshes
Create Scene with Camera + Mesh3D
Create Screen with AspectRatio and Shader
make_render()
is_face_in_front(Face3D)
Project Face3D → Face2D
Sort faces by distance (Painter's Algorithm)
Return List[Face2D]
render(faces)
Draw each Face2D triangle
Display Rendered Scene


UML Process View - 3D Rendering Engine
Launch Tkinter Interface
Gather user input (file, camera, resolution, etc)
Yes
Valid Input?
No
Import Mesh Data from File
Show Validation Error in Popup
Set Camera and Screen
Apply Shader + Color Variance
Filter Faces in Front of Camera
Project 3D Faces to 2D Points
Sort Faces by Distance (Painter's Algorithm)
Draw Faces on Tkinter Canvas
Display Rendered Scene


Physical View - 3D Rendering Engine
User's Computer
3D Engine (Python Application)
Renders 2D Faces
Reads Mesh/.obj Files
Computes Projections & Geometry
Stores Scene, Meshes, Camera
Tkinter GUI
Local File System
CPU
Memory

UML Class Diagram



Camera

Inherits

Vertex

Instance Variables

- □ _look_at: Vertex
- □ _forward: Vector
- □ _up: Vector
- □ _right: Vector

Constructor

- ● __init__(origin: Vertex, look_at: Vertex): None

Properties

- ○ forward: Vector {get;}
- ○ up: Vector {get;}
- ○ right: Vector {get;}

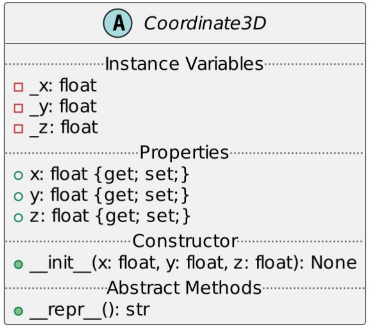
Public Methods

- ● set_look_at(new_look_at: Vertex): None
- ● is_vertex_in_front(vertex: Vertex): bool
- ● is_face_in_front(face: Face3D): bool
- ● project_vertex(vertex: Vertex): Point
- ● project_face(face: Face3D): Face2D

Private Methods

- ■ _recalculate_axes(): None

UML Class Diagram



Coordinate3D

Instance Variables

- □ _x: float
- □ _y: float
- □ _z: float

Properties

- ○ x: float {get; set;}
- ○ y: float {get; set;}
- ○ z: float {get; set;}

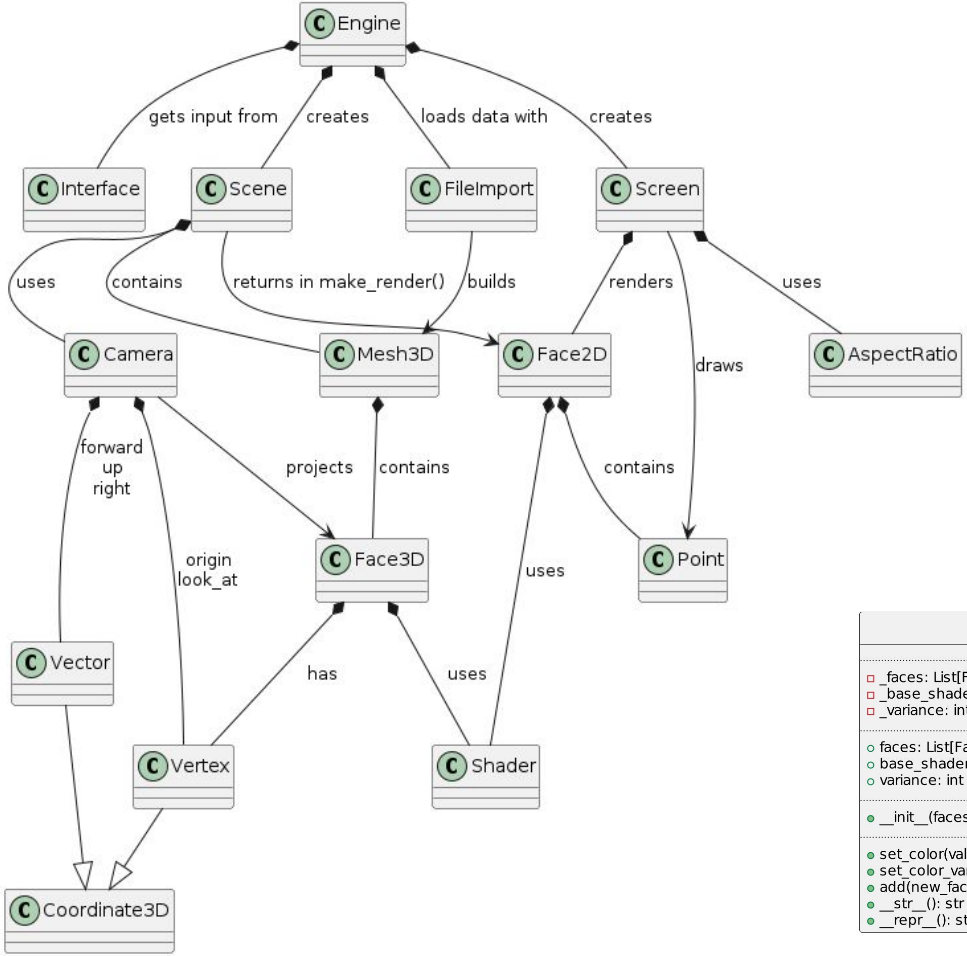
Constructor

- ● __init__(x: float, y: float, z: float): None

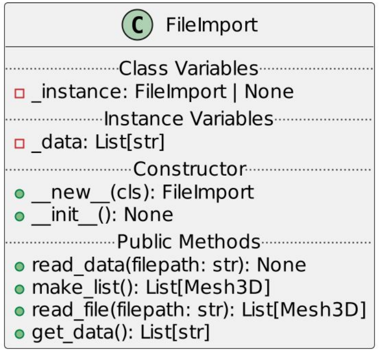
Abstract Methods

- ● __repr__(): str

UML Class Interaction - Full 3D Engine



UML Class Diagram



FileImport

Class Variables

- □ _instance: FileImport | None

Instance Variables

- □ _data: List[str]

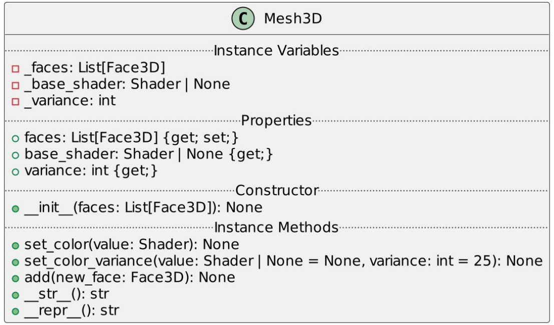
Constructor

- ● __new__(cls): FileImport
- ● __init__(): None

Public Methods

- ● read_data(filepath: str): None
- ● make_list(): List[Mesh3D]
- ● read_file(filepath: str): List[Mesh3D]
- ● get_data(): List[str]

UML Class Diagram



Mesh3D

Instance Variables

- □ _faces: List[Face3D]
- □ _base_shader: Shader | None
- □ _variance: int

Properties

- ○ faces: List[Face3D] {get; set;}
- ○ base_shader: Shader | None {get;}
- ○ variance: int {get;}

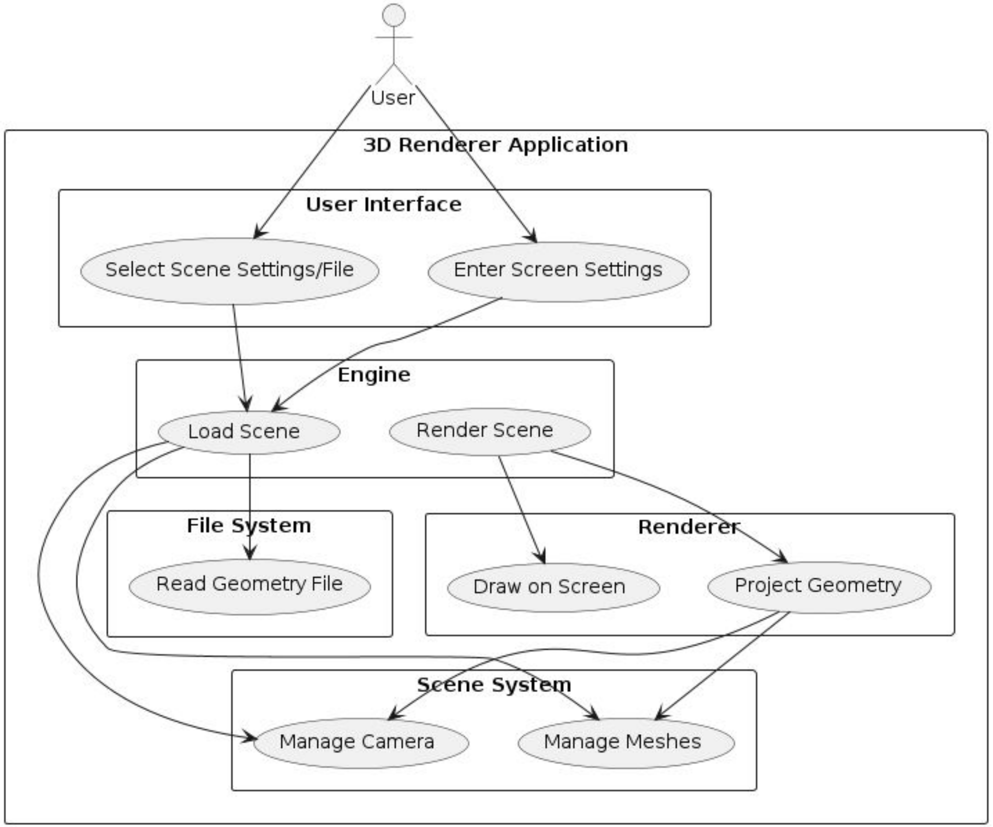
Constructor

- ● __init__(faces: List[Face3D]): None

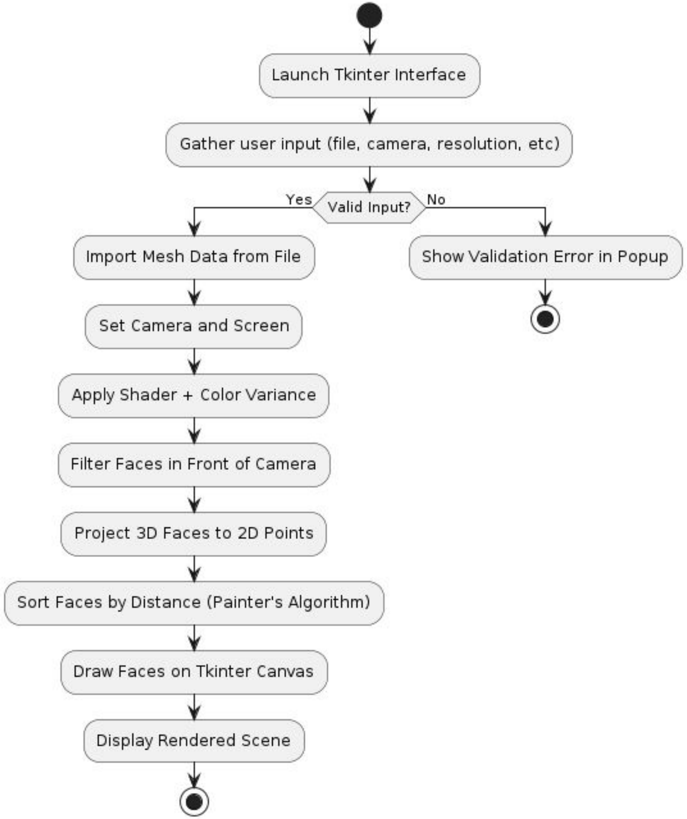
Instance Methods

- ● set_color(value: Shader): None
- ● set_color_variance(value: Shader | None = None, variance: int = 25): None
- ● add(new_face: Face3D): None
- ● __str__(): str
- ● __repr__(): str

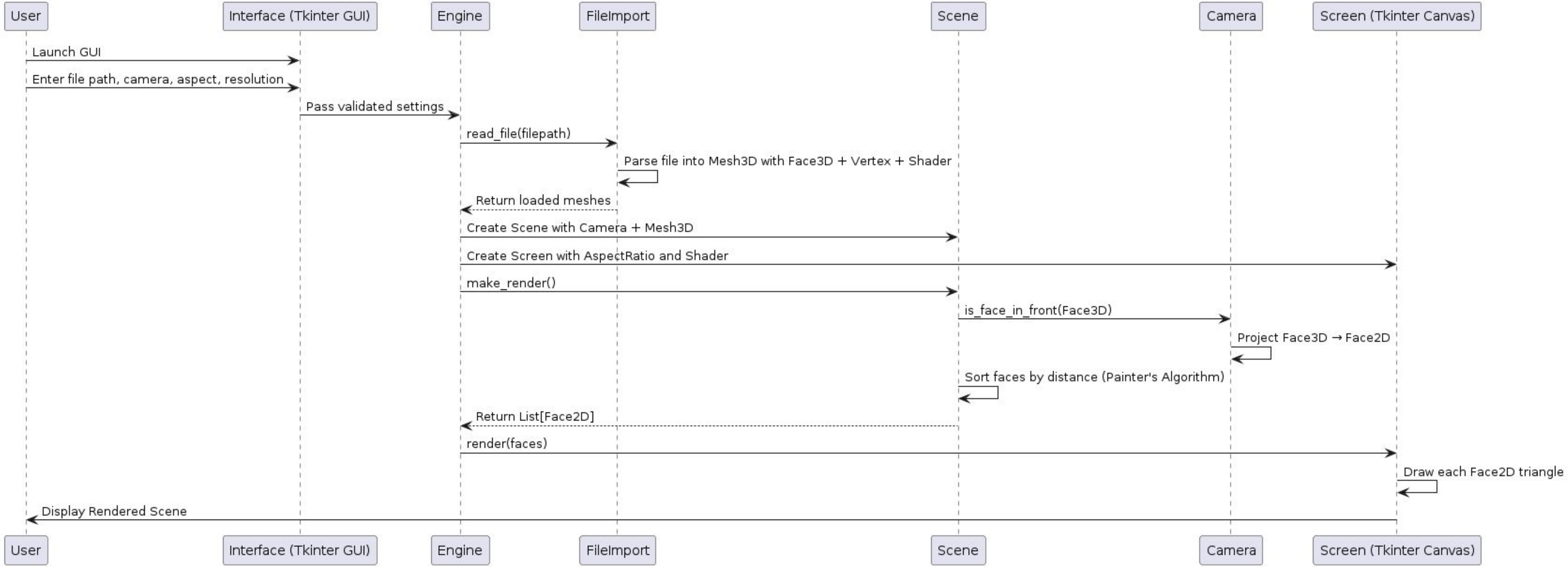
UML Context View - 3D Rendering Engine



UML Process View - 3D Rendering Engine


Launch Tkinter Interface
Gather user input (file, camera, resolution, etc)
Valid Input?
Yes
No
Import Mesh Data from File
Show Validation Error in Popup
Set Camera and Screen
Apply Shader + Color Variance
Filter Faces in Front of Camera
Project 3D Faces to 2D Points
Sort Faces by Distance (Painter's Algorithm)
Draw Faces on Tkinter Canvas
Display Rendered Scene

UML Scenario View - Loading and Rendering a Scene



User | Interface (Tkinter GUI) | Engine | FileImport | Scene | Camera | Screen (Tkinter Canvas)

Launch GUI

Enter file path, camera, aspect, resolution

Pass validated settings

read_file(filepath)

Parse file into Mesh3D with Face3D + Vertex + Shader

Return loaded meshes

Create Scene with Camera + Mesh3D

Create Screen with AspectRatio and Shader

make_render()

is_face_in_front(Face3D)

Project Face3D → Face2D

Sort faces by distance (Painter's Algorithm)

Return List[Face2D]

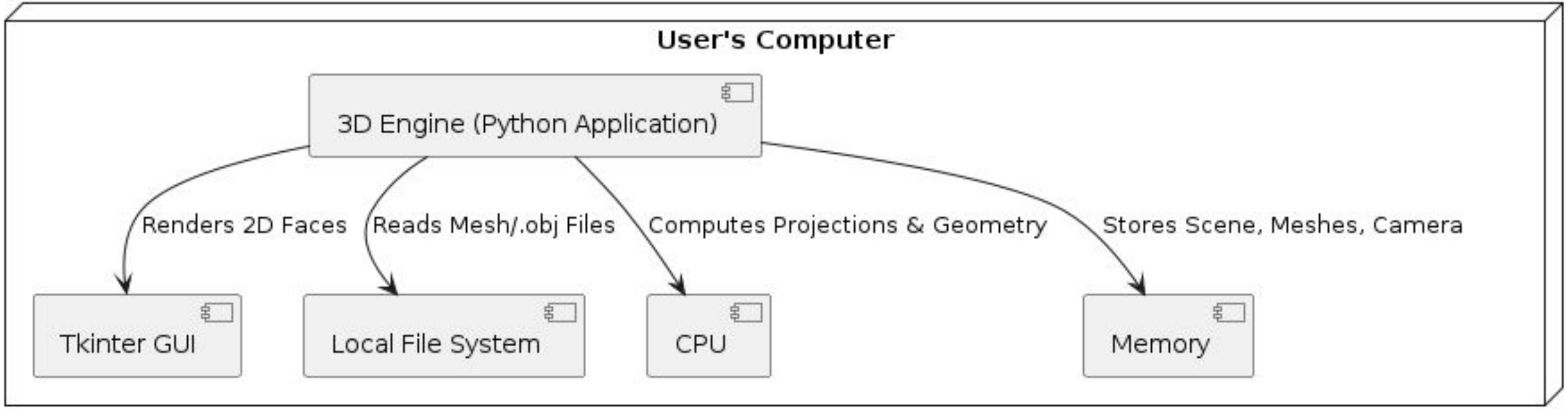
render(faces)

Draw each Face2D triangle

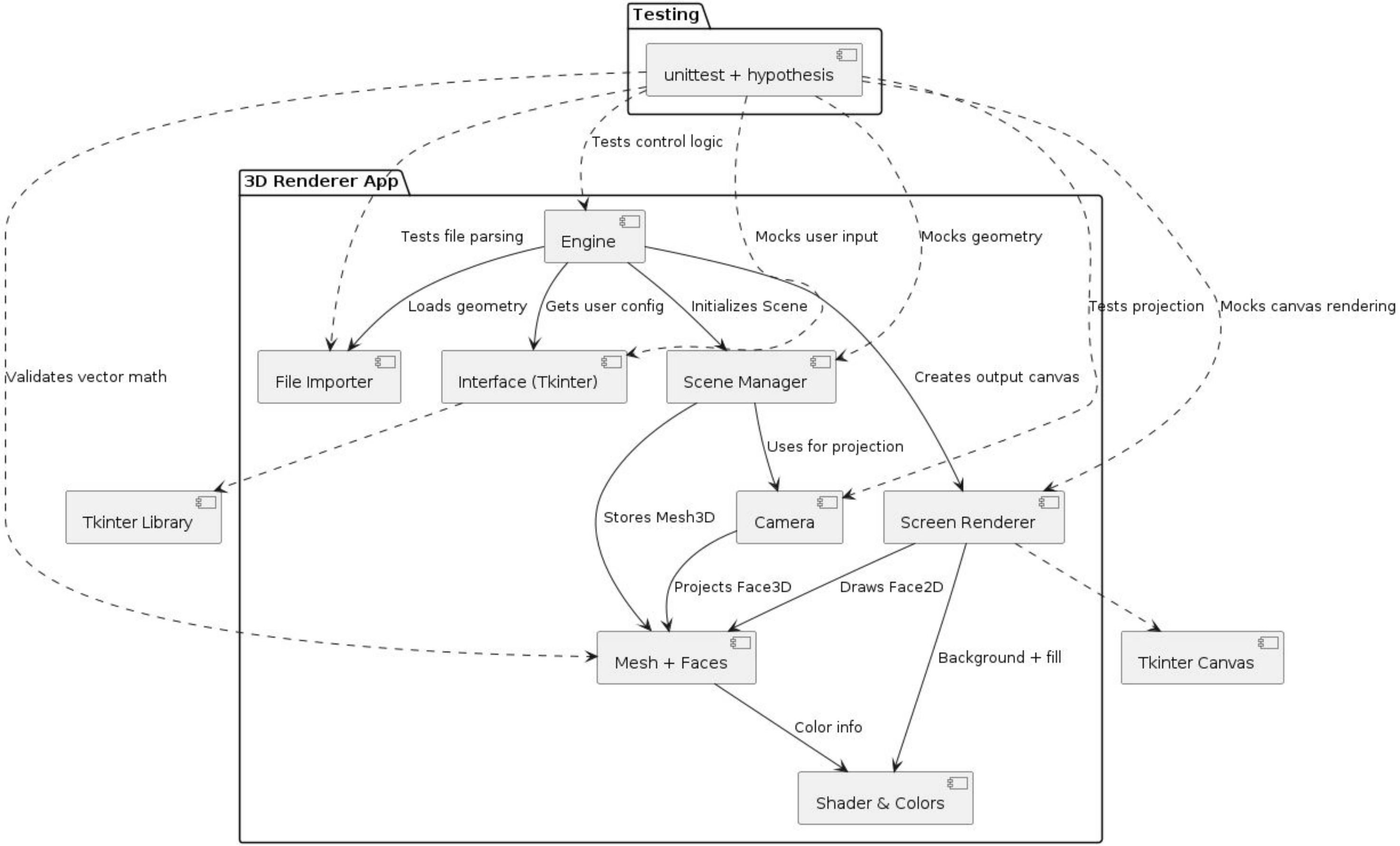
Display Rendered Scene

User | Interface (Tkinter GUI) | Engine | FileImport | Scene | Camera | Screen (Tkinter Canvas)

Physical View - 3D Rendering Engine


User's Computer
3D Engine (Python Application)
Renders 2D Faces
Reads Mesh/.obj Files
Computes Projections & Geometry
Stores Scene, Meshes, Camera
Tkinter GUI
Local File System
CPU
Memory

Developer View of 3D Rendering Engine



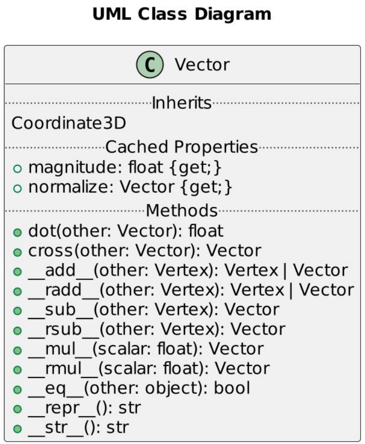
Object Oriented Design & Concepts

Design Patterns

- Singleton
  - Engine
  - File Import
- Builder
  - File Import
- Adapter
  - Camera
- Template
  - Coordinate3D (ABC)
- Composite
  - Nearly all except
    Point, Coord3D, Shader Aspect Ratio, Interface

Fundamental Concepts

- Inheritance
  - Coord3D -> Vertex -> Camera
  - Coord3D -> Vector
- Abstraction
  - Interface hides object processing algorithms from user.
  - Objects do not need to know how processing occurs.
- Properties and Setters
  - Every private instance variable has getter and setter properties, as needed



UML Class Diagram

C Vector

Inherits

Coordinate3D

Cached Properties

○ magnitude: float {get;}
○ normalize: Vector {get;}

Methods

● dot(other: Vector): float
● cross(other: Vector): Vector
● __add__(other: Vertex): Vertex | Vector
● __radd__(other: Vertex): Vertex | Vector
● __sub__(other: Vertex): Vector
● __rsub__(other: Vertex): Vector
● __mul__(scalar: float): Vector
● __rmul__(scalar: float): Vector
● __eq__(other: object): bool
● __repr__(): str
● __str__(): str

Project Management

- Issues created and assigned for each module or feature.
- Kanban Board in Github Projects managed workflow.

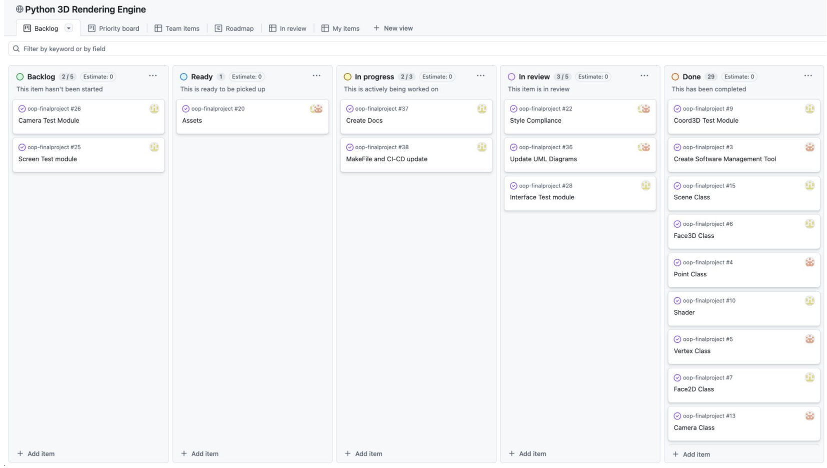
Michael – basic geometric point classes

Arin – geometric data classes

Michael – camera and rendering classes

Arin – file Import class

Michael – interface class



Python 3D Rendering Engine

Backlog 2 / 5 Estimate: 0
This item hasn't been started

- oop-finalproject #26 Camera Test Module
- oop-finalproject #25 Screen Test module

Ready 1 Estimate: 0
This is ready to be picked up

- oop-finalproject #20 Assets

In progress 2 / 3 Estimate: 0
This is actively being worked on

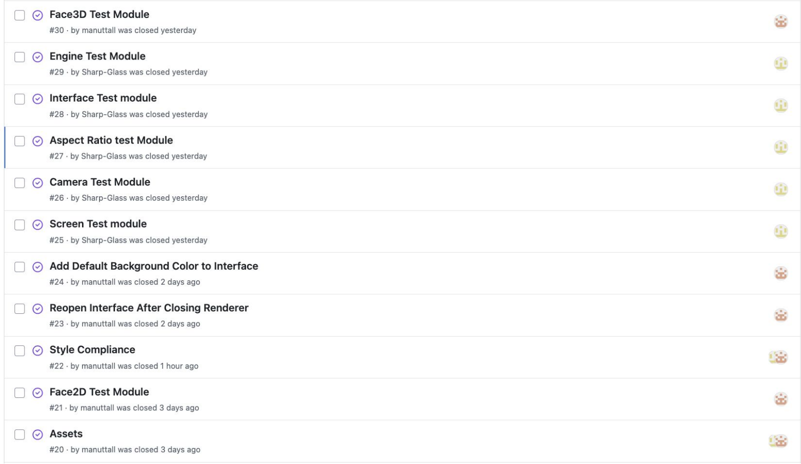
- oop-finalproject #37 Create Docs
- oop-finalproject #38 MakeFile and CI-CD update

In review 3 / 5 Estimate: 0
This item is in review

- oop-finalproject #22 Style Compliance
- oop-finalproject #36 Update UML Diagrams
- oop-finalproject #28 Interface Test module

Done 29 Estimate: 0
This has been completed

- oop-finalproject #9 Coord3D Test Module
- oop-finalproject #3 Create Software Management Tool
- oop-finalproject #15 Scene Class
- oop-finalproject #6 Face3D Class
- oop-finalproject #4 Point Class
- oop-finalproject #10 Shader
- oop-finalproject #5 Vertex Class
- oop-finalproject #7 Face2D Class
- oop-finalproject #13 Camera Class



- □ **Face3D Test Module**
  #30 · by manuttall was closed yesterday
- □ **Engine Test Module**
  #29 · by Sharp-Glass was closed yesterday
- □ **Interface Test module**
  #28 · by Sharp-Glass was closed yesterday
- □ **Aspect Ratio test Module**
  #27 · by Sharp-Glass was closed yesterday
- □ **Camera Test Module**
  #26 · by Sharp-Glass was closed yesterday
- □ **Screen Test module**
  #25 · by Sharp-Glass was closed yesterday
- □ **Add Default Background Color to Interface**
  #24 · by manuttall was closed 2 days ago
- □ **Reopen Interface After Closing Renderer**
  #23 · by manuttall was closed 2 days ago
- □ **Style Compliance**
  #22 · by manuttall was closed 1 hour ago
- □ **Face2D Test Module**
  #21 · by manuttall was closed 3 days ago
- □ **Assets**
  #20 · by manuttall was closed 3 days ago

Modules and Tests

Testing and CI-CD pipeline:

- Each module was run through a style checker called flake8.
- We ran a type checker using Mypy
- We also created a test_.py file and a test class for each module.
- All modules made by one of us were tested by the other.

Clean code and Modularity:
Each Class was sorted into one of four folders all within the larger renderer folder.

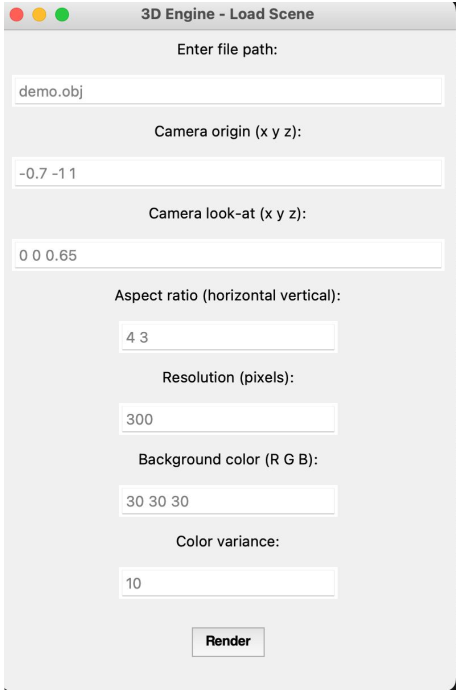
- The renderer parent folder containing the engine. (1)
  - Geometry - A folder made for the geometric data class (8)
  - Scene - A folder made for the objects used to comprise a scene (4)
  - Utility - A folder made for any objects used in the user interface or file management (2)
  - Tests - A folder made for tests (15)

Interactive Documentation

We created an interactive documentation of our code modules using the pdoc system:

After running our tests we also generated an interactive coverage report using pytests:

The Result



3D Engine - Load Scene

Enter file path:

demo.obj

Camera origin (x y z):

-0.7 -1 1

Camera look-at (x y z):

0 0 0.65

Aspect ratio (horizontal vertical):

4 3

Resolution (pixels):

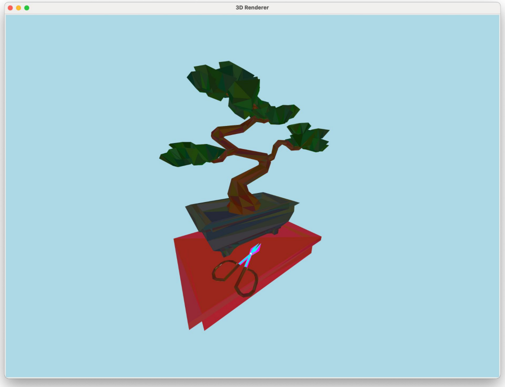
300

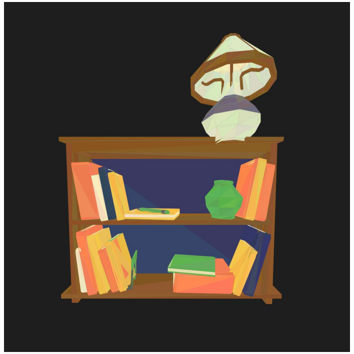
Background color (R G B):

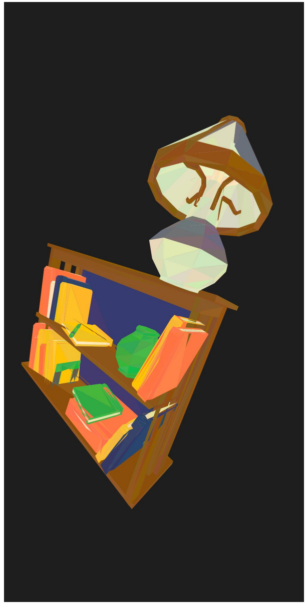
30 30 30

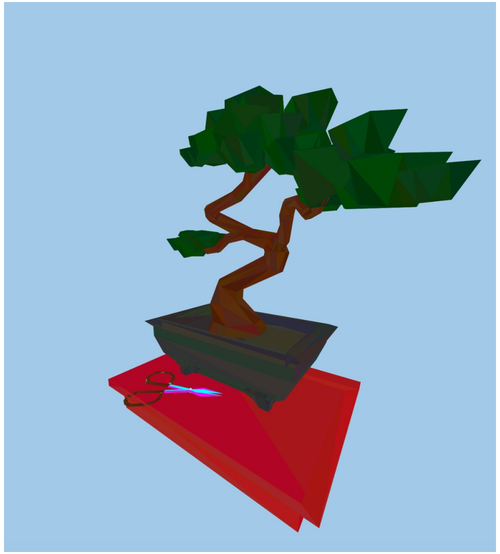
Color variance:

10









Conclusion

Bringing 3D objects to life is a challenging and intricate process that requires careful coordination of many components. Throughout this project, it was essential to track all elements and ensure they worked seamlessly within their respective systems. Object-Oriented Design (OOD) played a crucial role in helping us organize and manage the complexity of the project. By breaking the system into well-defined classes and responsibilities, OOD allowed us to build a flexible, maintainable, and scalable rendering pipeline. This not only improved development efficiency but also enhanced the quality and reliability of the final result.

Questions?