FACULTY OF SCIENCE AND TECHNOLOGY

SV40303 SCIENTIFIC DATA VISUALISATION

SESSION 1-2025/2026

GROUP ASSIGNMENT

ASSIGNMENT 1

MY BLENDER APPLICATION

LECTURER NAME: PROFESSOR DR. ABDULLAH BIN BADE

NO	NAME	MATRICS
1.	EZYAN NURWAHIDAH BINTI AHMAD SUKRI	BS22110082
2.	NUR SAKINAH BINTI MOHAMMAD ALI	BS22110305
3.	SUREKADEVI A/P SHANMUGANATHAN	BS22110175
4.	CHARLENE NG MEI XUAN	BS22110336

TABLE OF CONTENTS

1.0 SYSTEM INTRODUCTION 3

2.0 SYSTEM ARCHITECTURE 5

3.0 SYSTEM FLOWCHART 8

4.0 ALGORITHMS 8

5.0 SIGNIFICANT OUTPUT 12

6.0 GROUP CONTRIBUTION 15

1.0 SYSTEM INTRODUCTION

The VTK My blender System application is an interactive 3D visualization and modeling application system developed using Python, VTK, and PyQt5. The system provides a professional environment for users to create, manipulate, and visualize 3D objects and scenes in real time. It serves as both a learning and experimental platform for computer graphics, geometric modeling, and interactive rendering.

The main purpose of the VTK My Blender is to demonstrate how VTK's rendering pipeline can be integrated with PyQt5's user interface framework to produce an intuitive, real time 3D editing tool. The system enables users to create and visualize primitive 3D objects such as spheres, cubes, and Platonic solids. Users also will be able to import 3D model files in common formats like OBJ, STL, PLY, and VTK. Control lighting, shading and material properties for enhanced realism. The system comes with the feature of interacting with the 3D environment using zoom, rotating, translating, and scale transformations. This application also gives the users the experience of viewing and modifying the camera properties and orientation in a blender style viewport.

This VTK blender system application comes with multiple objectives. Combining VTK's rendering engine with PyQt5's GUI framework to create an interactive 3D workspace. This system also provides realistic rendering using Phong shading, multiple light sources, and grid/axis references. Manipulation works in the term of enabling to transform 3D objects through translation, rotation, and scaling tools. This system also focuses on user interaction where it allows intuitive scene control using an interactive axis widget and transform gizmos. Extensibility also plays a part in designing the system, so it can be expanded for future applications such as CAD tools, scientific visualization, or animation.

The VTK 3D Editor system offers a wide range of key features that make it an efficient and interactive 3D modeling tool. It supports multiple 3D file formats such as STL, OBJ, PLY, and VTK, allowing users to easily import and visualize complex models. The system provides an intuitive interface that includes a scene outliner for managing all objects and lights, as well as real-time rendering powered by VTK’s visualization engine. Users can create various 3D primitives including spheres, cubes, and Platonic solids, and modify their properties through translation, rotation, and scaling tools. Lighting options such as point, directional, and spotlights can be customized to achieve realistic visual effects, while material settings allow users to adjust opacity, shading, and surface reflections using Flat, Gouraud, or Phong shading.

The application also features camera controls for adjusting projection modes and viewpoints, a Blender-style grid and axis system for spatial guidance, and an orientation widget that allows users to quickly change viewing angles. Additionally, the system displays real-time statistics of vertices, edges, faces, and memory usage, ensuring efficient scene management and performance monitoring. Together, these features make the VTK 3D Editor a comprehensive platform for both educational and professional applications in computer graphics and visualization.

The VTK My Blender System stands as a powerful and versatile 3D visualization platform that bridges the gap between professional modeling tools and academic learning environments. By combining the capabilities of VTK's high-performance rendering engine with the flexibility of PyQt5's user interface, the system delivers an intuitive and realistic 3D editing experience. Its wide range of tools for object creation, lighting control, transformation, and camera management allows users to fully explore the principles of computer graphics and interactive design. Overall, this application not only enhances understanding of 3D visualization concepts but also serves as a strong foundation for future development in areas such as computer-aided design, scientific visualization, and digital content creation.

This system was developed not only to explore VTK and PyQt5 but also to create a practical learning platform for understanding rendering pipelines, mesh manipulation, and 3D interaction techniques used in real professional tools like Blender and Maya. The project bridges theory with practical implementation, making it useful both academically and for future software development work.

2.0 SYSTEM ARCHITECTURE

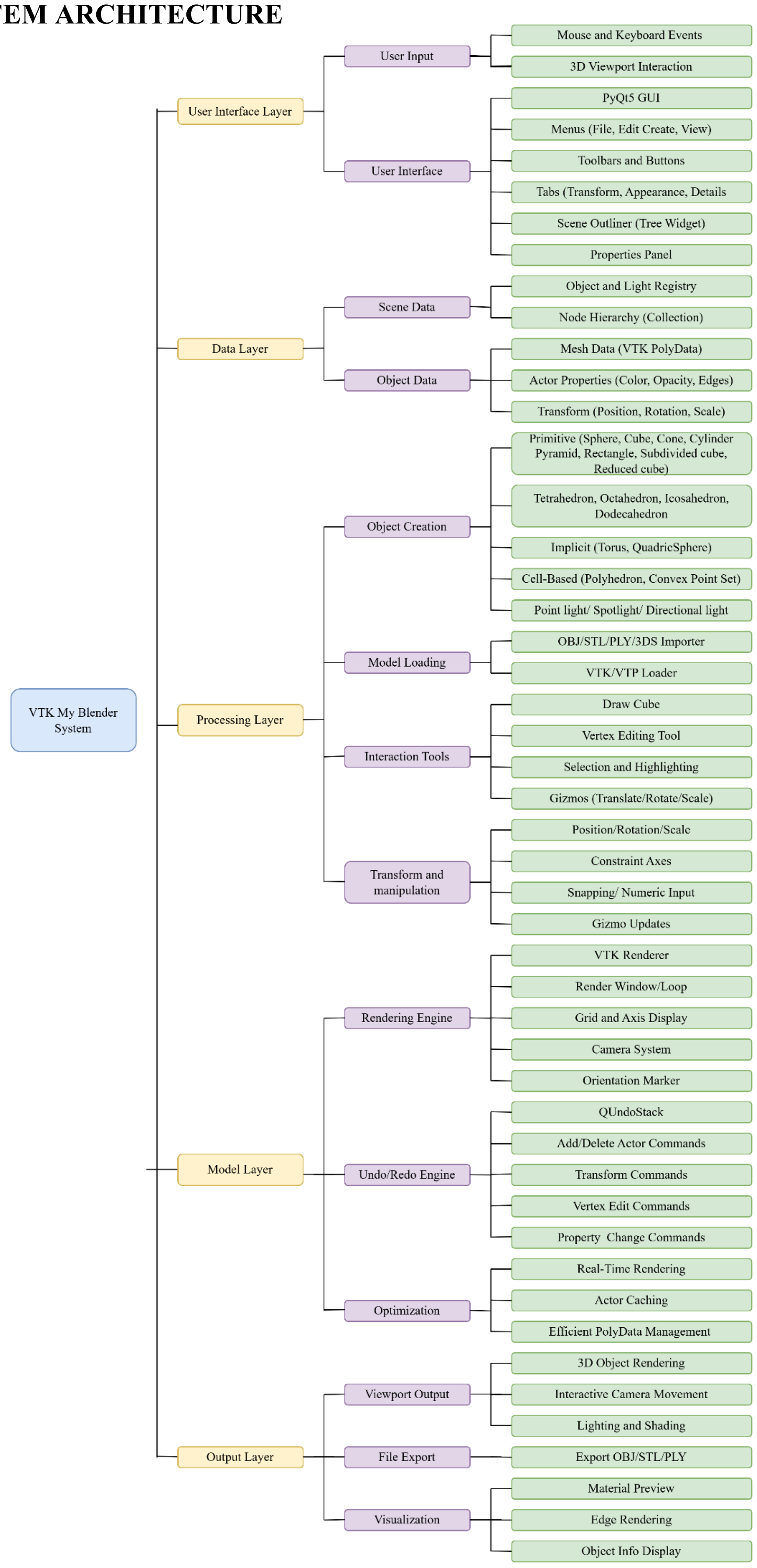


***Figure* 2.1**: System Architecture of VTK My Blender Application

A. User Interface Layer

The User Interface Layer manages all interactions between the user and the system. It provides graphical controls, tools, and panels for creating, editing, and managing 3D objects.

(i) User Input: Handles mouse/keyboard events, 3D viewport interaction, and shortcut keys (Move, Rotate, Scale) for object manipulation and navigation.

(ii) User Interface: Includes PyQt5 GUI components such as menus, toolbars, tabs (Transform, Appearance, Details), the Scene Outliner (QTreeWidget), and properties panels. This layer ensures an intuitive and organized editing environment.

B. Data Layer

The Data Layer stores all structured information that defines the 3D scene.

(i) Scene Data: Maintains the object and light registry, scene hierarchy (collections, parent-child), and scene organization metadata.

(ii) Object Data: Contains mesh data (vtkPolyData), actor properties (color, opacity, edges), and transform data (position, rotation, scale), ensuring consistency between UI, logic, and rendering.

C. Processing Layer

The Processing Layer performs all logical operations needed to generate, modify, and load 3D content.

(i) Object Creation: Generates primitive shapes (cube, sphere, cone, cylinder), implicit shapes (torus, quadric sphere), polyhedrons, and lights (point, spotlight, directional).

(ii) Model Loading: Imports external 3D models using OBJ, STL, PLY, 3DS, VTK, and VTP loaders.

(iii) Interaction Tools: Provides editing tools such as the Add Cube Tool, Vertex Editing Tool, selection/highlighting, and transformation gizmos.

(iv) Transform & Manipulation: Controls object movement, rotation, scaling, axis constraints, snapping, numeric input, gizmo updates, and camera-aligned dragging.

D. Model Layer (Core Engine)

The Model Layer contains the core computational systems that drive rendering, performance, and editing history.

(i) Rendering Engine: Built with VTK, featuring the renderer, render window, camera system, grid/axis display, and orientation marker for real-time 3D visualization.

(ii) Undo/Redo Engine: Manages editing history using QUndoStack with commands for adding/deleting actors, transforming objects, editing vertices, and modifying properties.

(iii) Optimization: Maintains smooth real-time rendering through actor caching, efficient PolyData management, and an optimized rendering pipeline.

E. Output Layer

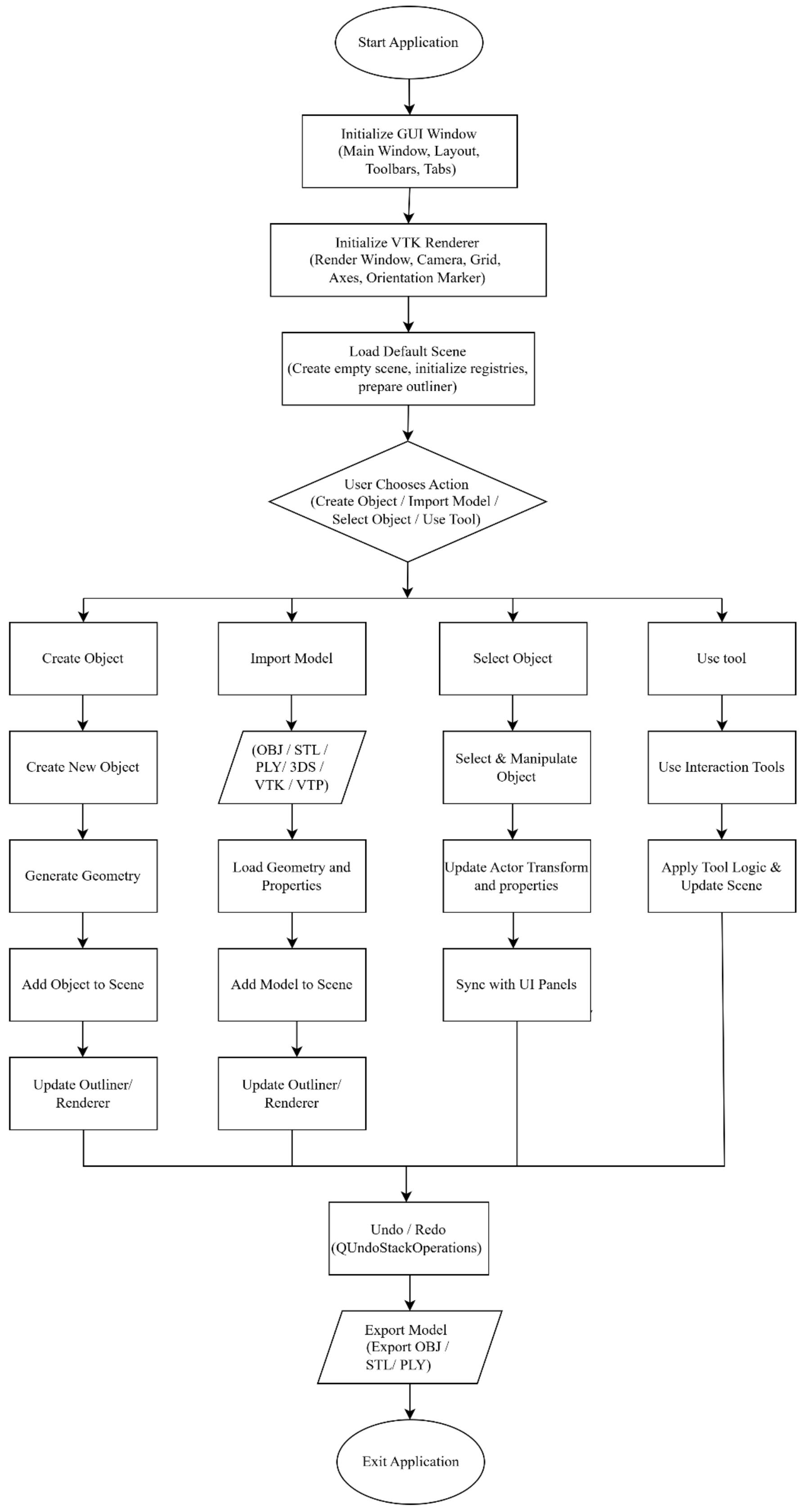
The Output Layer manages visual output and file export.

(i) Viewport Output: Displays real-time rendering, camera movement, lighting, shading, and the orientation marker.

(ii) File Export: Supports exporting 3D models in OBJ, STL, and PLY formats.

(iii) Visualization: Provides material preview, edge rendering, and object information display to enhance scene clarity.

3.0 SYSTEM FLOWCHART



Start Application
Initialize GUI Window (Main Window, Layout, Toolbars, Tabs)
Initialize VTK Renderer (Render Window, Camera, Grid, Axes, Orientation Marker)
Load Default Scene (Create empty scene, initialize registries, prepare outliner)
User Chooses Action (Create Object / Import Model / Select Object / Use Tool)
Create Object
Create New Object
Generate Geometry
Add Object to Scene
Update Outliner/ Renderer
Import Model
(OBJ / STL / PLY/ 3DS / VTK / VTP)
Load Geometry and Properties
Add Model to Scene
Update Outliner/ Renderer
Select Object
Select & Manipulate Object
Update Actor Transform and properties
Sync with UI Panels
Use tool
Use Interaction Tools
Apply Tool Logic & Update Scene
Undo / Redo (QUndoStackOperations)
Export Model (Export OBJ / STL/ PLY)
Exit Application

***Figure* 2.1:** System Architecture of VTK My Blender Application

4.0 ALGORITHMS

Algorithm 1: Algorithm to Create a 3D Primitive Object

```
Result: A new 3D primitive object is created and displayed in the scene
Input: object_type (sphere, cube, cone, cylinder, plane)
Initialization: Create VTK source according to object type
if object_type == "sphere" then
    Create SphereSource with radius and center
else
    if object_type == "cube" then
        Create CubeSource with dimensions
    else
        Create corresponding VTK primitive source
    end
end
Create PolyDataMapper and connect it to source output
Create Actor and assign mapper to actor
Set actor color and material properties
Add actor to renderer
Render the updated scene
```

Algorithm 2: Algorithm to Load an External 3D Model

```
Result: A 3D model file is loaded, converted into an actor, and rendered
Input: file_path
Extract file extension from file_path
switch file extension do
    case .stl do
        Use STLReader
    case .obj do
        Use OBJReader and check for textures
    case .ply do
        Use PLYReader
    case .vtp do
        Use XMLPolyDataReader
    case .3ds do
        Use 3DSImporter
end
Read the model data using the selected reader
Create mapper and actor from the model output
Apply texture if available
Correct orientation if needed
Add actor to renderer
Render the updated scene
```

Algorithm 3: Algorithm to Switch Application Theme

```
Result: Application UI and 3D view switch to the selected theme
Input: theme_type
if theme_type = "Blender" then
    Create dark color palette
    Apply dark palette to application
    Apply dark stylesheet (buttons, panels, text)
    Set renderer background color to dark gray
else
    if theme_type = "Light" then
        Create light color palette
        Apply light palette and stylesheet
        Set renderer background color to light gray or white
    else
        Restore original system palette and stylesheet
        Set renderer background color to default
    end
end
Refresh all visible widgets
Render the scene with the new background color
```

Algorithm 4: Algorithm to Manage Lights in the Scene

```
Result: Lights are added, modified, or removed from the scene
Input: operation_type (add, edit, remove), selected_light, parameters
if operation_type = "add" then
    Create a new light object
    Set light position, color and intensity from parameters
    Add light to renderer and internal light list
else
    if operation_type = "edit" then
        If selected_light is not NULL then
            Update light position, color or intensity using parameters
    else
        if operation_type = "remove" and selected_light is not NULL then
            Remove selected_light from renderer
            Remove selected_light from internal list
        end
    end
end
Render the updated scene
```

Algorithm 5: Algorithm for Scene Management

```
Result: Scene contents are managed (objects added, deleted or cleared)
Input: operation_type (add, remove, clear), actor
if operation_type = "add" then
    Add actor to renderer
    Append actor to internal actor list
else
    if operation_type = "remove" then
        If actor is not NULL then
            Remove actor from renderer
            Remove actor from internal actor list
    else
        if operation_type = "clear" then
            For each actor in internal actor list
                Remove actor from renderer
            Empty internal actor list
            Reset camera to default view
        end
    end
end
Render the updated scene
```

5.0 SIGNIFICANT OUTPUT

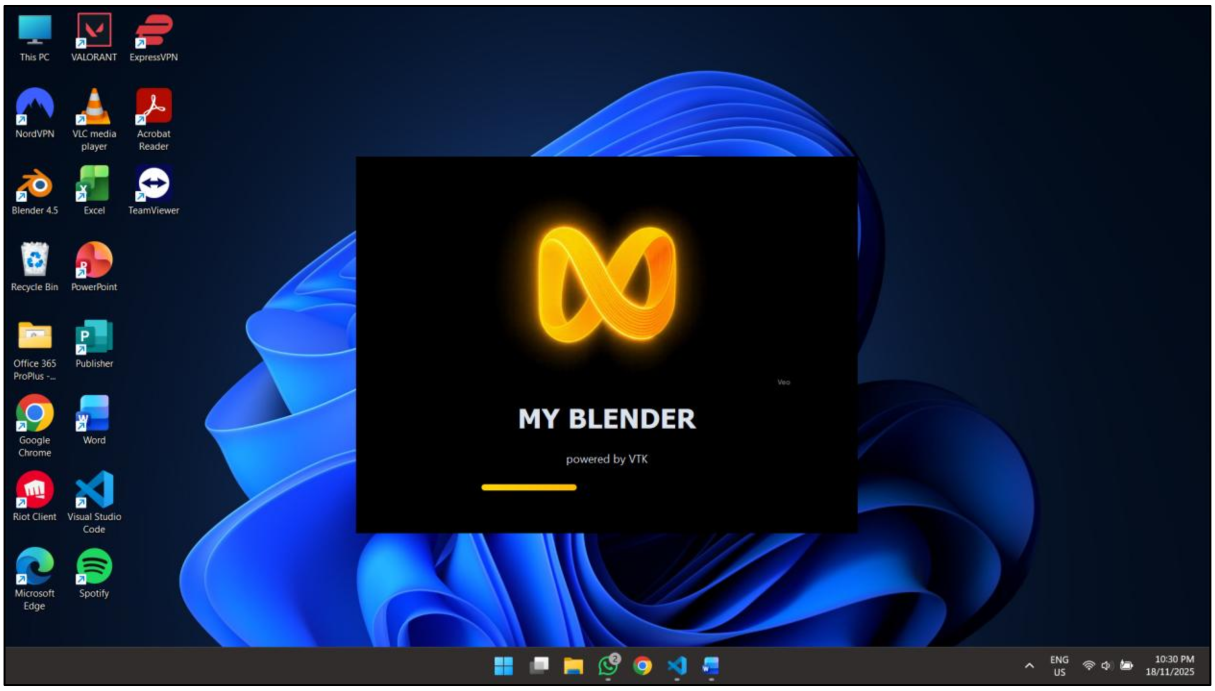
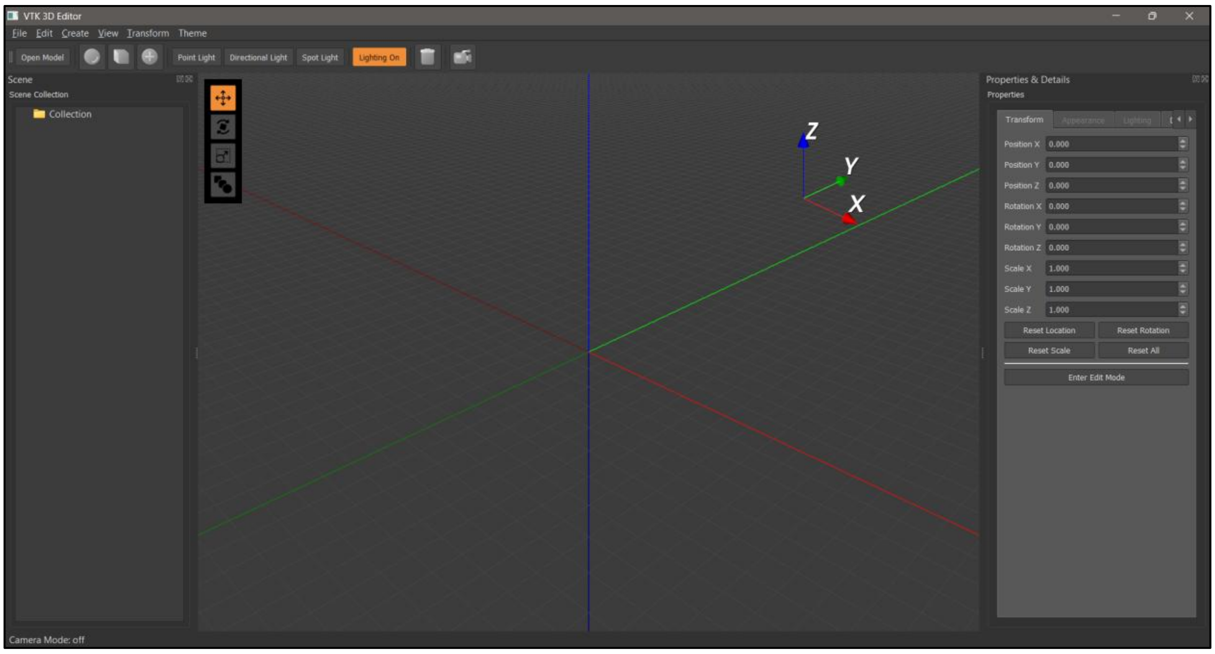
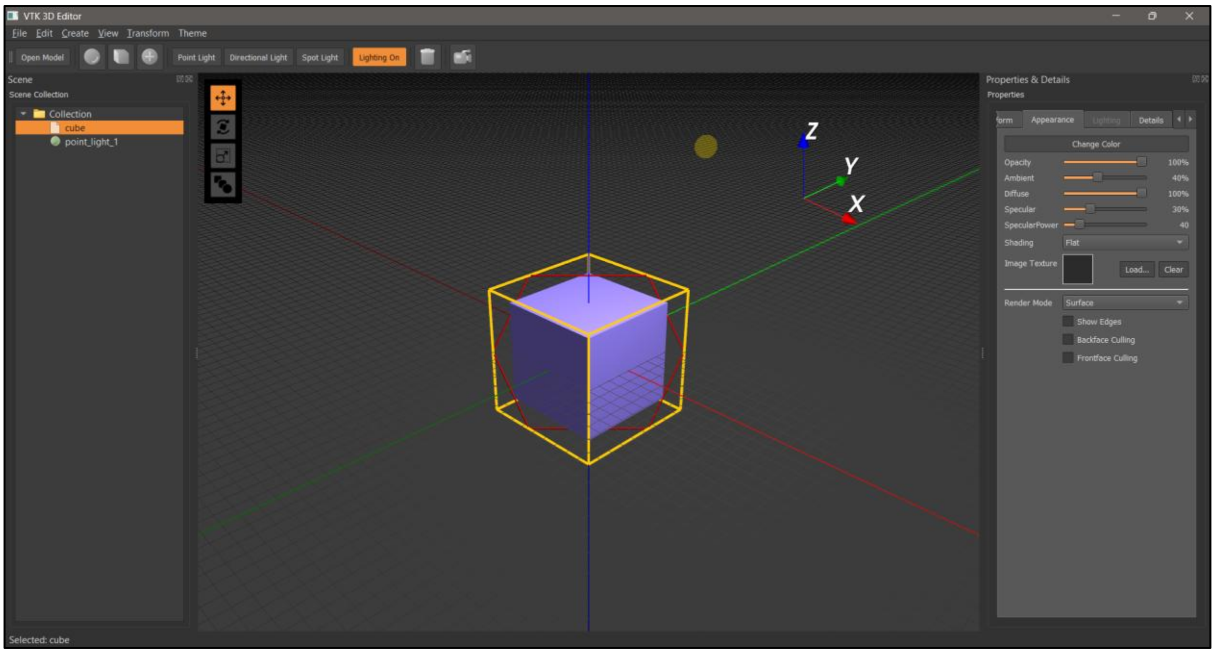


Figure **5.1**: Interactive Splash Screen Startup.



Position X	0.000
Position Y	0.000
Position Z	0.000
Rotation X	0.000
Rotation Y	0.000
Rotation Z	0.000
Scale X	1.000
Scale Y	1.000
Scale Z	1.000

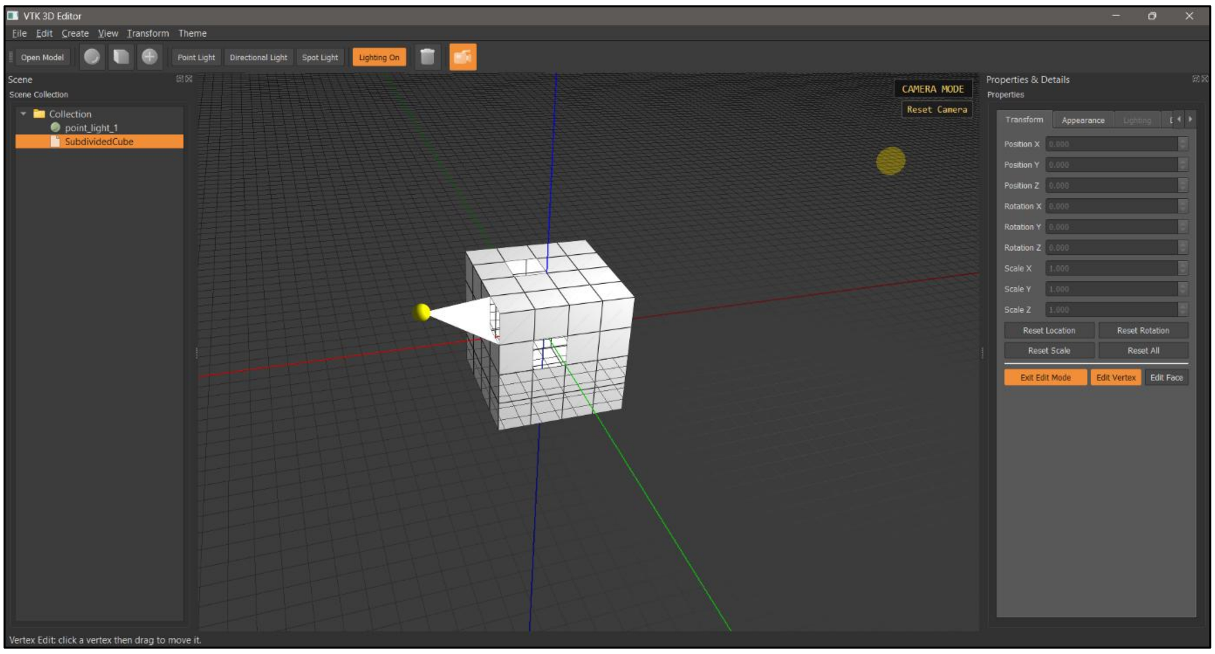
Figure **5.2**: Main graphical user interface of the 3D editor showing a default scene.



Opacity 100%
Ambient 40%
Diffuse 100%
Specular 30%
SpecularPower 40
Shading Flat
Image Texture
Render Mode Surface
Show Edges
Backface Culling
Frontface Culling

Selected: cube

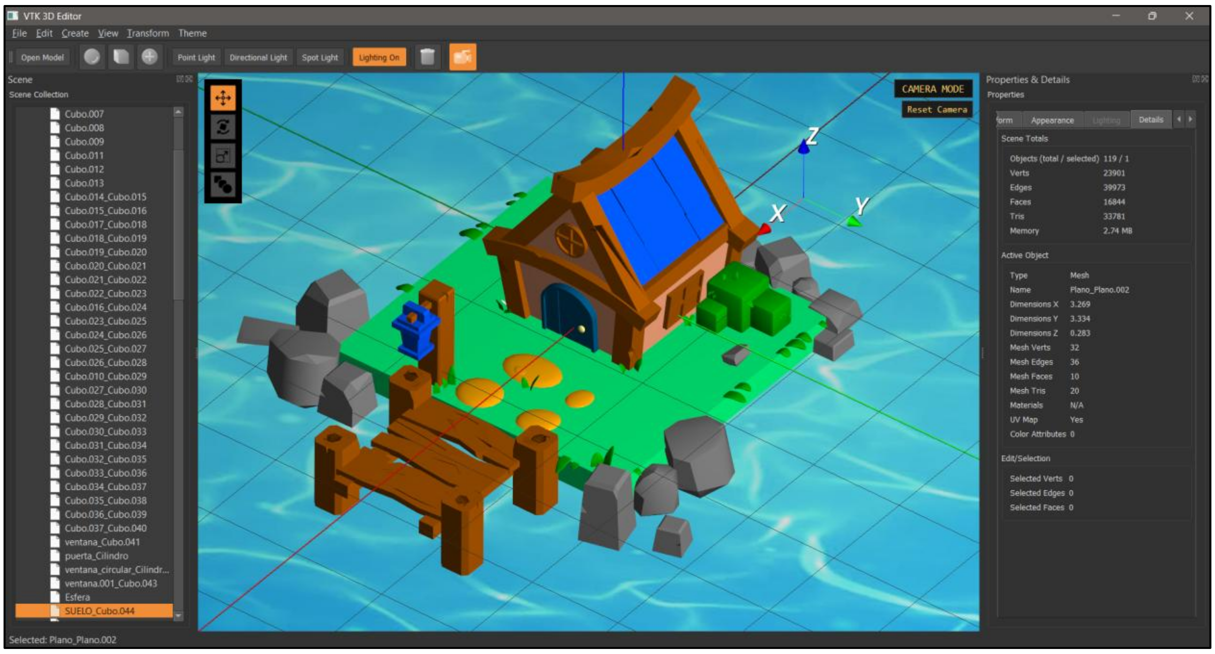
Figure 5.3: Rendered cube primitive through the object menu.



Position X 0.000
Position Y 0.000
Position Z 0.000
Rotation X 0.000
Rotation Y 0.000
Rotation Z 0.000
Scale X 1.000
Scale Y 1.000
Scale Z 1.000

Vertex Edit: click a vertex then drag to move it.

Figure 5.4: Demonstration of vertex editing and face-level manipulation on a subdivided cube.



Cubo.007
Cubo.008
Cubo.009
Cubo.011
Cubo.012
Cubo.013
Cubo.014_Cubo.015
Cubo.015_Cubo.016
Cubo.017_Cubo.018
Cubo.018_Cubo.019
Cubo.019_Cubo.020
Cubo.020_Cubo.021
Cubo.021_Cubo.022
Cubo.022_Cubo.023
Cubo.016_Cubo.024
Cubo.023_Cubo.025
Cubo.024_Cubo.026
Cubo.025_Cubo.027
Cubo.026_Cubo.028
Cubo.010_Cubo.029
Cubo.027_Cubo.030
Cubo.028_Cubo.031
Cubo.029_Cubo.032
Cubo.030_Cubo.033
Cubo.031_Cubo.034
Cubo.032_Cubo.035
Cubo.033_Cubo.036
Cubo.034_Cubo.037
Cubo.035_Cubo.038
Cubo.036_Cubo.039
Cubo.037_Cubo.040
ventana_Cubo.041
puerta_Cilindro
ventana_circular_Cilindr...
ventana.001_Cubo.043
Esfera
SUELO_Cubo.044

Scene Totals

Objects (total / selected)	119 / 1
Verts	23901
Edges	39973
Faces	16844
Tris	33781
Memory	2.74 MB

Active Object

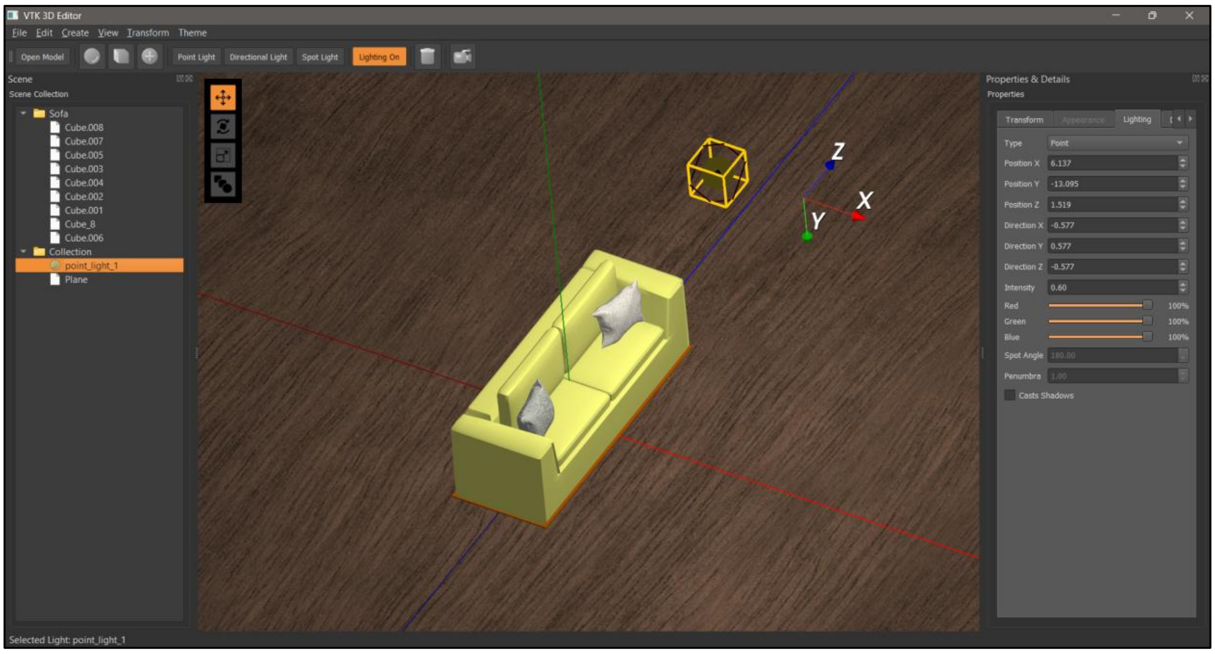
Type	Mesh
Name	Plano_Plano.002
Dimensions X	3.269
Dimensions Y	3.334
Dimensions Z	0.283
Mesh Verts	32
Mesh Edges	36
Mesh Faces	10
Mesh Tris	20
Materials	N/A
UV Map	Yes
Color Attributes	0

Edit/Selection

Selected Verts	0
Selected Edges	0
Selected Faces	0

Selected: Plano_Plano.002

***Figure* 5.5**: Loaded OBJ Scene with Texture and Multi-Object Color Editing.



Sofa
Cube.008
Cube.007
Cube.005
Cube.003
Cube.004
Cube.002
Cube.001
Cube_8
Cube.006
Collection
point_light_1
Plane

Type	Point
Position X	6.137
Position Y	-13.095
Position Z	1.519
Direction X	-0.577
Direction Y	0.577
Direction Z	-0.577
Intensity	0.60
Red	100%
Green	100%
Blue	100%
Spot Angle	180.00
Penumbra	1.00

☐ Casts Shadows

Selected Light: point_light_1

***Figure* 5.6**: Loaded 3DS Model with Color Editing and Lighting Controls.

6.0 GROUP CONTRIBUTION

NAME	CONTRIBUTION
EZYAN	CODE: - Implemented all supported 3D file format loaders including OBJ, STL, PLY, VTK, VTP, and 3DS. - Implemented the full PyQt5 application interface including the main window, menu bar, toolbar, dock widgets and global layout that defines the editor’s entire visual structure. - Implemented the Scene Collection (Outliner) panel including hierarchical collections, object icons, selection syncing with the viewport, context menus (rename, delete, duplicate), and integration with imported/created objects. - Implemented the right-side property panels including the Transform tab, Appearance/Material tab, Lights tab, and Details tab with well-organized labels, sliders, color pickers, and input widgets. - Implemented UI–logic integration for object properties including connecting UI controls to update color, opacity, lighting, transform values, and material properties of the selected object in real time.
	DOCUMENTATION: - Prepared the System Flowchart, detailing each process from model loading to scene rendering. - Wrote the Algorithms section, including object creation, transformation, rendering pipeline, and model loading logic
SAKINAH	CODE: - Implemented the full transform interaction system including translate, rotate, scale, and all-in-one widgets, complete with viewport manipulation, numeric panel sync, pivot handling, and undo/redo integration. - Implemented Edit Mode with vertex and face editing including precise vertex picking, vertex dragging, face

	selection, face highlighting, moving faces along their normals and face deletion. - Implemented advanced viewport picking and interaction logic including actor picking, face ID picking, vertex picking, drag detection and translating 2D mouse movement into accurate 3D operations. - Implemented interactive light manipulation tools including selecting lights in the viewport, visual gizmos for point/directional/spot lights and two-way synchronization between gizmos and the Lights panel.
	DOCUMENTATION: - Provided and organized the UI screenshots used in the Significant Output section, ensuring clarity and readable demonstration of system functionality. - Standardized, organized, and finalized the overall formatting of the documentation to ensure readability and consistency for submission.
SUREKADEVI	CODE: - Implemented multiple geometric and data formats, including cell-based objects, implicit surfaces, and parametric/ functional models, to ensure broad compatibility with VTK's 3D object requirements. - Developed the geometric object creation system, enabling the generation of primitive objects such as cubes, spheres, cones, cylinders, planes, and platonic solids using VTK source classes, all integrated into the "Create" menu for user accessibility. - Implemented cell-format model generation, including ConvexPointSet and Polyhedron Cell structures, fulfilling the assignment's cell-format criteria and incorporating them into the scene as editable polygonal objects. - Contributed to the design and implementation of the light theme, applying consistent styling across the MainWindow,

	DialogBox window, and Side Panel to improve visual clarity and user experience. - Enhanced the overall user interface by assisting with interactive UI elements, improving layout organization, menu usability, and general application responsiveness.
	DOCUMENTATION: - Authored the System Architecture section, including the complete architectural diagram and comprehensive descriptions of all layers which are User Interface Layer, Data Layer, Processing Layer, Model Layer (Core Engine) and Output Layer. - Assisted with proofreading UI-related parts of the documentation to ensure clarity and consistency.
CHARLENE	CODE: - Developed and enhanced the Splash Screen System, creating a custom startup interface that displays loading status, branding elements, and application initialization progress. The splash screen improves user experience by providing a professional and polished entry point to the 3D editor. - Improved the QWidget Tab Interface, enhancing layout organization, tab spacing, icon alignment, and visual clarity across the Transform, Appearance, Lighting, and Details panels. This update resulted in a more intuitive and consistent user interface. - Implemented multiple render modes including Surface, Wireframe, and Points, allowing users to switch between different visualization styles through the Properties Panel. This feature enhances model inspection and debugging. - Built and integrated the Details Tab, displaying technical metadata for the selected object such as the number of vertices, edges, faces, memory usage, and mesh statistics. The tab is fully synchronized with the Scene Collection and viewport selection system.

	DOCUMENTATION: - Wrote the System Introduction, explaining the purpose, scope, functionality, and objective of the VTK-based 3D editor. - Enhanced the formatting and layout of the Significant Output section, ensuring clean alignment, consistent captions, and improved readability of all included screenshots.