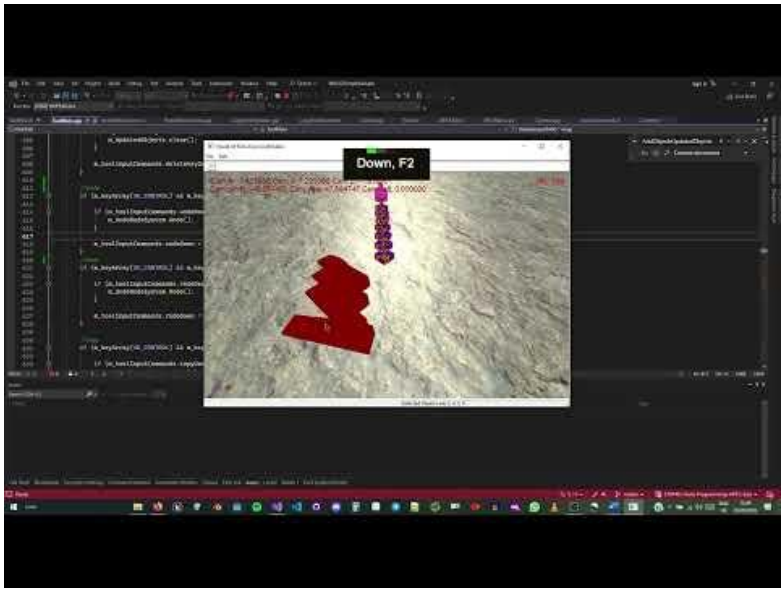
1905406 Utkarsh Yadav

Documentation

CMP 405 – Tools Programming

GitHub Repository Link: https://github.com/utkarshyadav009/CMP405-Tools-Programming-WFFC-Edit

Demonstration Video Link: https://youtu.be/KKyD4yt40S8



1 CONTENTS

2 Introduction 2
3 Controls 3
4 Features 4
4.1 Camera Class 4
4.2 Object Selection 6
4.3 Object Manipulation 6
4.4 Copy/Paste 7
4.5 Undo/Redo 7
5 Conclusion 8
6 References 8

2 INTRODUCTION

The task necessitated employing a specific tool framework and augmenting its functionality and usability. Research was conducted focusing on game development tools, with a special emphasis on the Unreal Engine, due to its status as the preferred choice among top AAA game studios. As of 2023, Unreal Engine, in conjunction with Unity, still leads the market, acclaimed for its advanced features and capacity to render exceptionally realistic visuals (Nuclino, 2023). The project aimed primarily at expanding the camera's functionality and usability in the tool. In addition, utility functions such as copy-paste and undo-redo were integrated into the tool's functionality. These additions were designed to simplify the user experience and further augment the tool's usability. This holistic approach to enhancing the tool served to enrich the overall functionality and usability of the project itself.

Here is a list of all the new functionality added to the framework:

1. New Camera class with improved camera movements and control.
2. Multi-Object selection, allowing for simultaneous interaction with several objects.
3. Object Manipulation, enabling the modification of Position, Rotation, and Scale of objects in the scene.
4. Copy-Pasting functionality, which permits duplication of Multiple or Single Objects in the scene.
5. Undo-Redo, for ease of reverting and reapplying actions, which offers a safety net for experimental changes and aids in error correction.

3 CONTROLS

- W – Move Forwards
- S – Move Backwards
- A – Move Left
- D – Move Right
- Q – Move Up
- E – Move Down
- Up – Rotate Up
- Down – Rotate Down
- Left – Rotate Left
- Right – Rotate Right

- Mouse Right Click – Hides the Cursor and performs Camera Rotation

- Mouse Left Click – Object Selection
    - Ctrl + LMB – Multiple Object Selection

- Object Editing – Select the Object and then Press the Following Keys
    - F1 – Edit the Position.
        - Up – Move Positive X
        - Down – Move Negative X
        - Right – Move Positive Z
        - Left – Move Negative Z
    - F2 – Edit the Rotation
        - Up – Move Positive X
        - Down – Move Negative X
        - Left – Move Positive Z
        - Right – Move Negative Z
    - F3 – Edit the Scale
        - Up – Move Positive X
        - Down – Move Negative X
        - Left – Move Positive Z
        - Right – Move Negative Z

- Copy-Paste – While the object is selected press:
  - Ctrl + C – Copy
  - Ctrl + V – Paste

- Undo-Redo
  - Ctrl + Z – Undo
  - Ctrl + Y – Redo

4 FEATURES

4.1 CAMERA CLASS

The Camera class serves as a representation of a controllable viewpoint in a 3D environment. Here's a detailed breakdown of how it functions:

Initialization

The Camera class can be initialized in two ways. The default constructor (Camera()) sets the movement speed, rotation speed, mouse sensitivity, and the initial position and orientation of the camera to predefined values. All vectors (position, orientation, etc.) are also initialized.

The second constructor allows more customized initialization by passing the desired values of the movement speed, rotation speed, mouse sensitivity, and initial position and orientation.

In both constructors, a call to HandleInput() is made with a null delta time and an empty InputCommands object. This updates the camera look direction, right vector, and look-at point based on the initial orientation and position.

Updating Camera Orientation

Camera orientation is updated in the HandleInput() function, which is meant to be called every frame with the elapsed time since the last frame (delta time) and the current state of inputs (encapsulated in an InputCommands object).

If the mouse is not controlling the camera, the orientation is updated based on arrow key inputs: left and right keys rotate the camera (yaw), and up and down keys change the pitch. The actual rotation is scaled by the rotation speed and delta time to ensure consistent speed regardless of the frame rate.

If the mouse is controlling the camera, the change in mouse position from the center of the viewport is used to update the orientation. The rotation speed, mouse sensitivity, and delta time are all factors in the calculation.

Updating Camera Position

Camera position is also updated in HandleInput(). The camera position can move in six directions: forward, backward, left, right, up, and down. The exact direction is determined based on the current look direction (for forward/backward), the right vector (for left/right), and the y-axis (for up/down). The movement is scaled by the move speed and delta time and halved if the "slow move" input command is active.

Updating Camera Look Direction and Right Vector

After updating the orientation, the look direction is recalculated based on the new orientation. This is done by creating a unit vector from the Euler angles of the orientation. The right vector is then calculated as the cross product of the look direction and the y-axis.

Updating LookAt Point

The look-at point is updated to be a point in space in the direction that the camera is looking, starting from the camera's position. This point is used to construct the look-at matrix.

Getting LookAt Matrix

The GetLookAtMatrix() function constructs a matrix using the current position, look-at point, and up-vector (which is simply the y-axis). This matrix is a key component in 3D graphics rendering as it defines the transformation from world space to view space, effectively controlling what the camera sees of the 3D world.

Setting and Getting Camera Properties

Several inline functions are provided to set and get various camera properties such as move speed, rotation speed, mouse sensitivity, camera position, and camera orientation. These functions allow other parts of the program to interact with and control the camera.

In summary, the Camera class encapsulates all the functionality required to control a viewpoint in a 3D environment, updating its orientation and position based on user input, and providing the necessary transformations for rendering the 3D world from the camera's perspective.

4.2 Object Selection

Variable: std::vector<unsigned int> m_SelectedObjectIDs;

m_SelectedObjectIDs is a vector storing unique identifiers (IDs) for the objects that are currently selected in the scene.

The ObjectSelection() inside the Game class, is responsible for determining which object in the game is currently under the mouse cursor. It does this by raycasting, essentially drawing a line from a start point in the direction of the mouse cursor. The function uses screen dimensions, mouse input, and the world and projection matrices to transform the ray from screen space to world space, which then intersects with the bounding boxes of objects in the scene. The ID of the closest intersected object is stored in the selection variable, and the function returns this ID at the end.

The object's ID is later used for operations such as highlighting the selected object(s), by changing their colour. Multiple objects can be selected by pressing Ctrl + Left Mouse Button (LMB), which adds the new selection to the m_SelectedObjectIDs vector without clearing previous selections.

4.3 Object Manipulation

The EditObjectTransform class is used to manipulate the position, rotation, and scale of one or more objects in a 3D scene.
The object manipulation is performed by three distinct functions

1. **ChangePos(),**
2. **ChangeRot(),**
3. **ChangeScale()**

These three methods handle changes in the position, rotation, and scale of the selected objects in the scene. They take three parameters: deltaTime, actionDirection, and objectID.

- deltaTime is used to ensure that transformations occur at a consistent speed, regardless of frame rate.
- actionDirection indicates the direction of the transformation in 3D space. It can be positive or negative in the X, Y, or Z direction.
- objectID contains the unique IDs of all selected objects.

In each method, the function loops over the provided objectID vector and fetches the corresponding SceneObject instance from ToolMain. It then adjusts the relevant transformation property (position, rotation, or scale) based on actionDirection and deltaTime.
The ChangeScale method also checks if the scale of the object is greater than 0.01 before decreasing it. This prevents the object from disappearing entirely or inverting if the scale drops too low. After all transformations have been applied, ToolMain->RebuildDisplayList() is called to update the display list with the new object transformations.

4.4 COPY/PASTE

The CopyPasteSystem class is responsible for handling the copying and pasting of SceneObject instances.

It performs these actions using these two functions.

1. CopySelectedObjects (CopyPasteSystem::CopySelectedObjects): It takes a vector of object IDs as input, representing the objects to be copied. For each ID, it gets the corresponding SceneObject from the scene graph in ToolMain, then adds a copy of the SceneObject to a local selectedObjects vector. If at least one object has been selected, it clears the m_CopyedObjects vector and copies over the selectedObjects.
2. PasteCopiedObjects (CopyPasteSystem::PasteCopiedObjects): This function is called when the user wants to paste previously copied objects. It goes through each object in m_CopyedObjects and moves their position slightly so that they won't just pile up at their original position when pasted. Then, it passes the m_CopyedObjects vector to the onActionPasteObjects method of ToolMain to paste the objects into the scene.

The key combinations CTRL+C and CTRL+V are used to copy and paste objects, respectively. When CTRL+C is pressed, m_CopyPasteSystem.CopySelectedObjects is called with m_d3dRenderer.m_SelectedObjectIDs as the argument, which are the currently selected objects' IDs. When CTRL+V is pressed, m_CopyPasteSystem.PasteCopiedObjects is called to paste the copied objects into the scene.

4.5 UNDO/REDO

The UndoRedoActions class facilitates the creation and handling of 'undo' and 'redo' actions. It is part of a system which allows for scene objects to be added, modified, and deleted with the option to reverse and reapply those changes.

It achieves this by maintaining a vector of Action structs, each representing a discrete change made by the user in the scene. These actions are of type Addition, Deletion and Default. The Action struct also includes pre-change and post-change snapshots of the scene objects, which helps in facilitating the undo and redo operations.

To manage the above-mentioned actions, several functions are defined in the class, they are as follows:

1. **AddNewAction**: Records an action as it happens, the type of action and the object IDs affected are stored, along with a snapshot of the affected objects in their pre-change state.
2. **AddPostAction**: captures a snapshot of the affected objects in their post-change state. This is used when an action is reversed or reapplied.
3. **Undo and Redo**: Revert or Reapply the most recent action, they alter the scene objects as required and update the action stack position to accurately reflect the current state.
4. **IsObjectFlagedForDeletion**: checks if a given object ID is flagged for deletion.

5. **DeleteAllFlaggedObjects:** It deletes all the objects that are flagged for deletion in the scene graph.

5 CONCLUSION

Through out the development phase of this project, significant insights have been gained in the field of tools programming, its intricacies, and functional mechanics. The project has successfully incorporated essential features, such as a functional camera, multiple object selection, copy and pasting of multiple objects, object manipulation of multiple objects, and the undo and redo utility actions.
The project has successfully laid the groundwork by including the most fundamental tools essential for game development. However, it has not yet incorporated more advanced features such as terrain editing, multiple camera perspectives, or an archball or focus camera on selected objects. An area for future improvement would be the user interface. Currently, most interactions are facilitated through keyboard or mouse controls. Introducing UI elements could enhance the intuitive aspect of the project, similar to the user-friendly interfaces provided by renowned game engines such as Unreal Engine 5 and Gamemaker (GameDesigning, 2023).

In Conclusion, I learned a lot about tools programming by doing this project and the importance of basic functionality tools that streamline game development. Furthermore, the importance of a user interface was also a key takeaway, shedding light on its role in facilitating more intuitive interactions.

6 REFERENCES

GameDesigning, 2023. *How to Choose the Best Video Game Engine.* [Online]
Available at: https://www.gamedesigning.org/career/video-game-engines/
[Accessed 16 May 2023].

Nuclino, 2023. *Best Game Development Software in 2023.* [Online]
Available at: https://www.nuclino.com/solutions/game-development-software
[Accessed 16 May 2023].