BSc (Hons) Computer Game Applications Development

CMP 405: TOOLS PROGRAMMING

Joseph Roper
UNIVERSITY OF ABERTAY

Contents

Links

Git Repo: https://github.com/1901881/CMP405-Tools-Coursework

Demo Vid: https://youtu.be/_YA87t9nIgc

Summary of Features

The WOFFC editor has undergone significant improvements, which include the addition of a camera class to abstract the camera code from the project. This camera class enhances movement and camera control, with the ability to rotate and focus on selected objects. Furthermore, object manipulation features have been added to the editor, providing users with the ability to modify an object's rotation, scale, and position. Users can select and deselect multiple objects and apply these manipulation features, as well as copy and paste them to create new objects. Additionally, the editor allows users to switch the terrain and objects to wireframe mode and save their objects after modifying their properties back into the database, making it easier to load and continue working on saved projects.

Feature Controls

Camera – WASD to move E for up Q for down, Arrow keys to rotate the camera.

Object Movement - MFC button press then IJKL OU, I - Forward, K - Backward, J - Left, L - Right, O - Up, U - Down.

Object Rotation - MFC button press then IJKL.

Object Scaling - MFC button press then mouse scroll wheel up for scale increase and down for decrease. J - X Axis scaling, K - Y Axis scaling, L - Z Axis scaling, Nothing held down - Whole object scaling.

Highlight - Select object with left mouse button click.

Object focus - left mouse button double click on selected object.

Object Wireframe - select objects and click MFC Button

Terrain Wireframe - click MFC button.

Multiselect - Hold Ctrl and left click on objects you want to select.

De-select - Hold Ctrl and Alt and left click on object you want to de-select.

Object copy - Press Ctrl and C to copy selected Objects

Object Paste - Pres Ctrl and V to paste copied objects.

Object saving - click MFC button

Feature Review

Object Camera Focus

Feature Explanation

It was considered vital for the user to be able to traverse the world quickly in the WOFFC editor to allow for faster development. To achieve this, it was decided that a good idea would be to enable the camera to move closer to the selected object and focus on it for viewing and editing when the user double-clicks the object. This feature is very common in game engines due to its usefulness, and was therefore a necessary addition to the editor.

To make the feature more accessible and easier to use and remember, it was made accessible by a double click instead of requiring the user to remember a key or find it on the keyboard. The way the feature is accessed is similar to that in Unity, meaning that it has similar controls, making the workflow smoother as it is already ingrained in the user's mind to focus on an object that way.

Technical Discussion

In the game class mouse picking function, a selection counter was added so that if an object is selected, the counter goes up by one. If the same object is clicked again, the counter goes up to 2. This was achieved by saving the previous selection and checking it against the current selection.

If an object is clicked on a second time, the camera function Object focus is called, and the selected object's position is passed to it. From this, the camera position and look at direction are updated, using the object's position to move closer towards it.

Multiselect

Feature Explanation

Users can select multiple items at a time and perform other object manipulation features such as copy and paste, object highlight, movement, rotation, scale, and wireframe mode. This feature is useful for group object manipulation to save development time. The feature can be activated by holding Ctrl and continuing to select other objects. Users can deselect objects by holding Ctrl and Alt down and clicking the object they no longer want to be a part of the object multi selection. This control scheme is similar to most editors such as Photoshop, Unity, and Unreal Engine, which makes editing less tedious when switching between programs as new controls don't need to be learnt.

Technical Discussion

The multiselect feature was implemented by detecting the Ctrl input while the control key is held down. Within the tool main class, the boolean *multiSelectActive* is set to true when the key is pressed, and set to false when the key is released. In the game class, the *selectedID* was changed from an integer to an integer type vector since there are going to be multiple objects selected.

In the game class's mouse picking function, the object selection proceeds as usual, getting an integer for the selected object ID. However, before selecting an object, the function first checks whether an object is already selected by verifying if the *selectedID* does not equal -1. If the multiselect boolean is active and the multiselect vector is not empty, it checks if the *selectedID* is already within the multiselect vector using the standard C++ library find function (cplusplus, 2000).

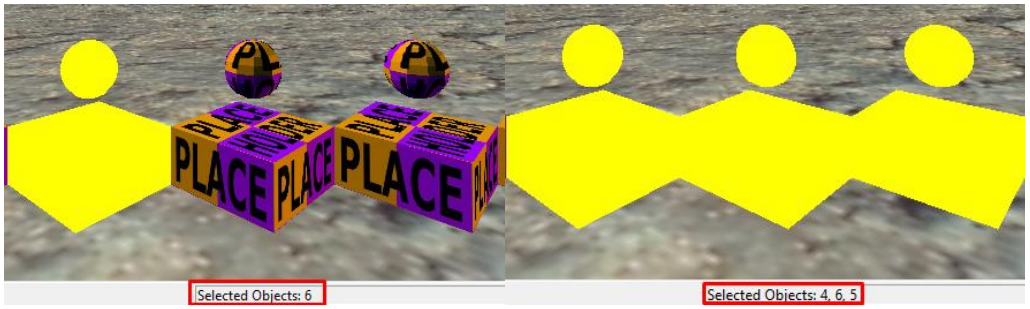
If the selectedID is not in the multiselect vector, the front of the vector is added to the multiselect vector, and the front is then set to the currently selected ID. If the *selectedID* is already in the vector, and the alt key is held down, the multiselect vector is iterated through, adding every element that does not equal the *selectedID* to a temporary vector. After looping through the multiselect vector, the temporary vector is assigned to the multiselect vector, which acts as the deselect feature, removing the *selectedID* from the multiselect vector. In hindsight, a map could have been used to make this process easier and less hardware-intensive, as the program would not have to iterate through the whole vector to remove the selected ID.

When the multiselect boolean is false, the program iterates through the multiselect vector, checking if the newly selected object ID is within the multiselect vector. If it is not, it means a new object has been hit, so the multiselect vector is cleared, and the new object ID is added.

If the multiselect is not active, and the vector is empty, the *selectedID* is added to the vector. If the vector is not empty, the front of the vector is changed to the currently selected ID, allowing the vector to behave like the original mousepicking function when multiselect is not active.

The program's other mechanics, such as object movement and scaling, had to be altered to accommodate multiselect. This was done by passing the multiselect vector instead of the previous integer. The program then iterates through the vector, treating each element as it did previously.

To display the multiselectIDs on the window, MFC was used to update the integer with the vector. The vector was first converted into a stringstream using the c++ standard library copy function, placing a comma after every integer (Dibling et al., 1956). The stringstream was then converted into a wstring (Pavlek, 2020), and the last comma from the sting was erased. Finally, the multiselectIDs were added to the statusString and displayed on the program's window frame.



Img.: Showing object multiselect

Object Copy and Paste

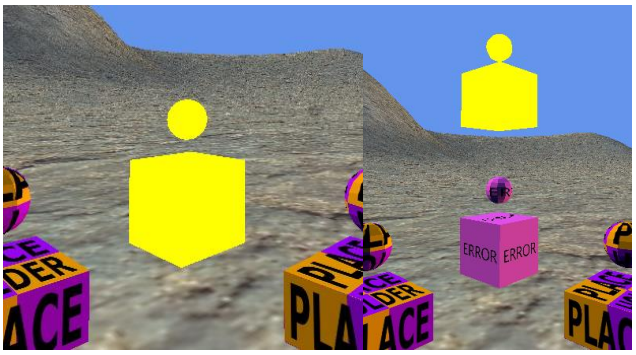
Feature Explanation

When a user selects an object, the selected object can be copied by pressing Ctrl and c. Subsequently, the copied object can be duplicated by using Ctrl v. This feature facilitates level creation by allowing the user to quickly place multiple objects. The controls for this feature are similar to those found in other programs, including Word, Photoshop, Unreal Engine, and Unity. As a result, users are likely to find the UX design to be user-friendly, since they are already familiar with the controls.

Technical Discussion

The input commands class contains inputs for Ctrl, c, and v. Pressing Ctrl and c simultaneously saves the currently selected object ID as an integer called *m_copiedID*. When Ctrl and v are pressed together, the program checks whether the copiedID integer is empty and uses it to add a copy of the object to the object display list. To ensure ease of use for the user, the pasted object is translated above the original.

During testing, it was observed that the program would continue to paste the object multiple times while Ctrl and v were pressed down, as the computer is faster than the user's finger. To prevent this, a boolean named *m_pastePlayedOnce* is created and set to false. After the input check, the program verifies if the boolean is false, then pastes the object and sets the boolean to true. When the v key is lifted and no longer pressed, the boolean is reset to false, allowing for the object to be pasted only once per v press.



Img.: Showing object copy paste

Originally, the paste function would copy the object and push it back into the display list with a different ID. However, this resulted in confusion for the user when the copied object or its parent was selected, as both objects would highlight, but only one would be selected. To address this issue, a new object had to be created from scratch and added to the display list. To achieve this, the object's model and texture path were saved. At the start of the program, when the display list is built, these variables are filled with the scene graph information so that when an object is pasted, it accesses the copied object's model and texture paths to create the object. The new object is created by copying the copied object's position, rotation, and scale, making it an exact replica. Because a new model is created, the highlight now works independently on the parent and copied objects as they have different models.

Object Scaling

Feature Explanation

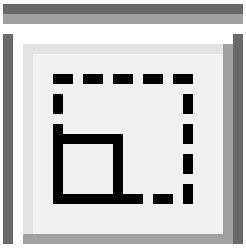
Object scaling is a useful feature in game engines as it provides game developers with more flexibility in level design with the ability to create variations of game assets easily. This feature allows game designers to resize and manipulate game assets to create the desired environment, reduce development time and costs, and enhance the overall gameplay experience for players. That is why it was essential for the WOFFC editor to have this feature implemented. Within the editor, the user can increase or decrease the size of the object, first selecting the object and using the mouse scroll wheel to change its scale.

Technical Discussion

To implement this feature, first the mouse scroll wheel input is read in the tool main by utilizing the delta value of the mouse wheel. A positive value indicates upward movement, while a negative value indicates downward movement. When either of these values is detected, a boolean is set to true or false to indicate whether the mouse wheel is moving up or down. If the boolean for the mouse wheel up is true, the game class's scaleUp and Down function is called with the *scaleUpOrDown* boolean parameter set to true. This function gets the selected object and adds to its scale, scaling it up. Conversely, if the mouse wheel down boolean is true, the selected object's scale is subtracted from the current value.

Initially, the objects would continuously increase or decrease in scale after the first scroll, so a boolean named *m_scrollPlayedOnce* was created to make sure the scale function was only called once per mouse wheel scroll.

To provide more flexibility, object scaling in a single axis was added, with the controls modified to allow scaling in a particular direction when j, k, or l keys were held for x, y, or z axis respectively. Scaling up and down was still executed using the mouse wheel up and down, with scaling only happening if the MFC object scaling button was pressed.



Img.: Object scaling button

An MFC button was created to toggle object scaling mode within the game class, with object movement and rotation mode set to false, to prevent controls overlap. When this boolean was true, the scaling code in the game class ran. The game class's scale up and down function iterated through the selected objects, applied the object's current scale to a temporary vector, and subtracted or added the scale depending on the mouse scroll wheel direction. The input command enumerator for the object scaling was modified to include Scaling Direction, representing the different scaling axes (whole, X, Y, and Z). Depending on

the input command (set when the jkl keys were pressed within the tool main class), either the whole display object scale was set to the temporary vector scale or just the axis pressed.

Img.: Object scaling in each direction, Order: Original size, scale in X, scale in Y, scale in Z, whole.

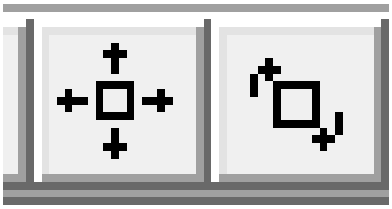
Selected Object Movement

Feature Explanation

In a game engine, the ability to move and rotate objects around is crucial for creating dynamic and interactive game environments. The movement of objects is essential for designing challenging levels and immersive worlds. Additionally, the ability to rotate objects adds an extra level of realism to the game world, allowing developers to simulate the effects of forces such as gravity and friction. Overall, the ability to move and rotate objects is a fundamental aspect of game development, making it imperative that the WOFFC editor has this feature as it enables developers to create immersive and engaging gaming experiences.

Accessing the movement of an object in the game engine is done through an MFC button, and upon pressing it, the selected objects can be moved in different directions based on the camera view using the IJKL keys. Additionally, the object can be translated upwards using the O key and downwards using the U key.

To rotate a selected object, there is another MFC button that, when pressed, enables the user to rotate the object using the IJKL keys. The object movement and rotation keys have been made the same for the user's convenience, eliminating the need to remember multiple key commands for different actions.



Img.: Sowing object manipulation buttons in order of object movement then rotation

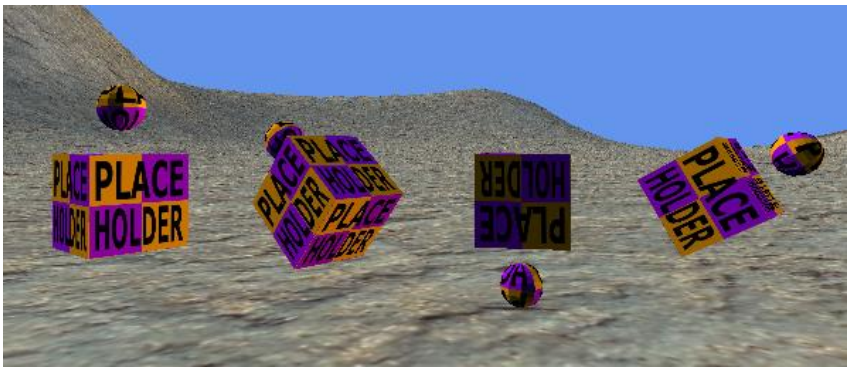
Technical Discussion

The implementation of object movement is similar to that of camera movement. An enum named "move direction" was created within the input commands class to hold different movement directions.

Within the game class, an object move function has been implemented to take the selected object ID and the enum as inputs. These inputs are passed through a switch statement to determine the movement direction.

To make the movement more intuitive for the user, the movement direction is based on the camera's "look at" direction. This makes it feel more like controlling the object in a third-person game, increasing user-friendliness.

Similarly, object rotation is accessed through a function in the game class that takes the same parameters. The movement and rotation of objects were separated into different classes so that they can work in conjunction with the MFC buttons. These buttons disable other object manipulation actions from being performed, as the movement and rotation use the same control scheme. The function for object rotation also uses a switch statement in conjunction with the input command to change the selected object's orientation.



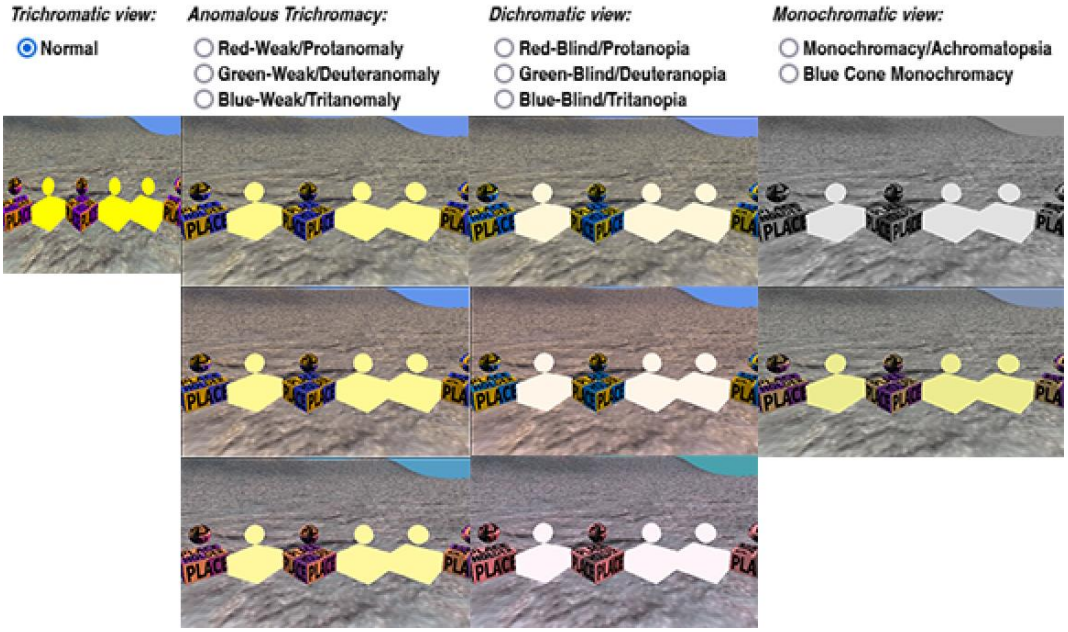
Img.: Showing object rotation

Object Highlight

Feature Explanation

The addition of object highlights to the WOFFC editor was important, as it enhances the gameplay experience by providing visual feedback to the player. Object highlights draw attention to the object the user is currently selecting.

In addition to enhancing gameplay, object highlights can also be used for accessibility purposes. Players with visual impairments may have difficulty distinguishing objects in the game world, and object highlights can make important elements more visible and easier to interact with. To ensure that the highlight was visible to colour-blind individuals, it was tested using a colour-blindness simulator (Wickline, 2006), as shown in the pictures below.



Img.: Showing object highlight colour blind test

Overall, object highlights are a useful tool for game developers to improve the player experience and accessibility of their games. The object highlight feature in this editor is applied automatically when the user selects an object, making it more user-friendly since a separate command is not required to activate it.

Technical Discussion

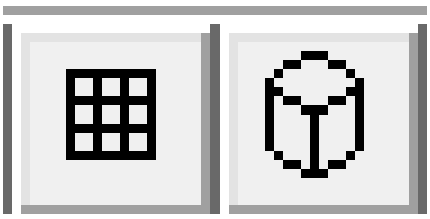
The effect was achieved by updating the display object class with a highlight function that accepts a boolean parameter. A DirectX fog effect is created and instantiated by the function, enabling the fog using the boolean parameter and changing its colour to yellow. In the game class, a new function called ObjectHighlightUpdate was created, which is called within the mouse picking function. When an object is selected, the highlight update function is called. First, the function checks if there are any objects in the display list. If there are, it iterates through the list, de-highlighting all objects. Then, the function iterates through the selectedIDs vector, highlighting them. This approach ensures that only the selected objects are highlighted when the selected object list is changed.

Terrain and Object Wireframe Mode

Feature Explanation

Wireframe mode is a highly beneficial tool for game developers and designers as it allows them to see the underlying structure of objects and terrain in a game engine. This feature can be used for debugging, identifying and fixing mesh issues, optimizing game performance, and creating more complex environments. As a result, wireframe mode is considered to be a necessary addition to the WOFFC editor, as it can significantly improve the quality and performance of games. To access the wireframe mode feature, users can simply click on the two corresponding buttons in the MFC (Microsoft Foundation Class)

interface. One button enables the terrain wireframe mode, while the other button converts selected objects into wireframe mode, allowing developers and designers to easily toggle between wireframe and solid modes as needed.



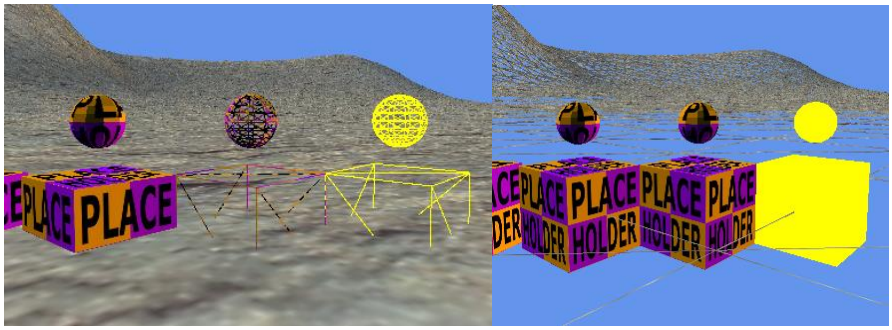
Img.: Showing wireframe mode buttons

Technical Discussion

To create the MFC buttons, the toolbar was opened in the Visual Studio resource view. From there, two new buttons were added: one for switching the wireframe mode of the terrain and the other for the selected objects.

When the terrain wireframe button is clicked, a function in the toolmain class is accessed, which changes a boolean to the opposite of its value. This boolean is a part of the game class and encapsulates a line of code in the render function with an if statement. This line of code sets the terrain to wireframe mode when played.

Similarly, the object wireframe button behaves like the terrain wireframe button, but instead calls a function in the game class when clicked. This function takes in the multi-select IDs as a parameter and loops through the vector, setting each selected ID in the display lists wireframe boolean to the opposite of its value in the same way as the terrain wireframe button.



Img.: Showing wireframe modes

Object Saving

Feature Explanation

Object saving is an essential feature in the world of gaming, and particularly for content creation in WOW (World of Warcraft). Without this functionality, the task of creating content would be virtually impossible. That's why it was crucial for the WOFFC editor to have the ability to save the user's progress. The ability to save objects allows designers to create

complex structures and designs, providing a more seamless and efficient content creation process, which ultimately enhances the overall designing experience. Thus, object saving is a fundamental component that cannot be overlooked, especially in the context of WOW content creation.

Technical Discussion

Object saving is initiated through an MFC button located in the tool bar. Upon clicking this button, the OnActionSave function is executed in the tool main class. The content of the database is first deleted by this function, and then the object list is retrieved from the game class using a getter function. The information about the temporary display list objects is then added to the scene graph by iterating through it. Finally, the scene graph is added back to the database for the next time the object list is built.

Conclusion

The existing coding structure was used to implement the additional features in the editor, ensuring that another worker could easily understand the code. It was realized that to increase performance, maps could have been utilized instead of vectors, which require looping through to access individual elements.

The basic features needed for an editor, such as object manipulation, copy-paste, wireframe, highlight saving, and object focus, were successfully implemented, allowing content creation to be started. For future work, a new object creation feature and an easier way to import new models and textures into the database from the editor will need to be added.

To improve the workflow, it is planned to add a gizmo to the selected object using MFC. This will allow object manipulation features to be performed by clicking and dragging with the mouse, making it faster and more intuitive than having to remember key commands. Additionally, the object manipulation MFC buttons will be made accessible by key presses such as W for object movement, E for rotation, and R for scale, making it easier for users familiar with game engines like Unity to transfer their knowledge to the WOF editor.

Overall, the editor's new features will improve content creation, but there are still some shortcomings in their implementation. These can be addressed through iterative feedback-based programming. This assessment serves as a demonstration of the editor's potential for future feature development.

References

cplusplus (2000) *STD::Find*, *cplusplus.com*. Available at: https://cplusplus.com/reference/algorithm/find/ (Accessed: 13 May 2023).

Dibling, J., Benedict, J. and Pollex, B. (1956) *How to transform a vector into a string?*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/2518979/how-to-transform-a-vectorint-into-a-string (Accessed: 13 May 2023).

Pavlek, J. (2020) *Converting STD::String to std::wstring*, *Hashnode*. Available at: https://hashnode.com/post/converting-stdstring-to-stdwstring-cka1q0zj5020fbls1ml34841w (Accessed: 13 May 2023).

Wickline, M. (2006) *Coblis - color blindness simulator*, *Colblindor*. Available at: https://www.color-blindness.com/coblis-color-blindness-simulator/ (Accessed: 13 May 2023).