T34 vs Tiger Modding Guide

Table of Contents

Introduction

This document serves as a comprehensive guide for modders looking to explore and modify the World War II tank combat game, "T34 vs Tiger." This game, developed by G5 Software and released by Lighthouse Interactive, utilizes a custom-built engine and a script-driven system, which opens up numerous possibilities for customization and enhancement.

However, it is important to note that a significant portion of the game's code is reused from G5's previous game, "Whirlwinds over Vietnam," and that the game was released prematurely due to financial difficulties at Lighthouse Interactive. This explains why some systems may be incomplete, and why some game settings are located in the core game engine, and can not be modified using scripts.

This guide aims to provide a clear understanding of the game's architecture, its key systems, and the techniques and best practices for modding, irrespective of the modder's skill level. It is designed to help you navigate the game's code, understand its limitations, and unlock the potential for new content and experiences. Whether you're a seasoned modder or a beginner, we hope this document will provide you with the knowledge to create awesome mods for T34 vs Tiger.

Game Architecture Overview

"T34 vs Tiger" employs a unique and flexible game architecture centered around a custom-built engine and a script-driven approach. This architecture, while powerful, also presents some limitations.

Custom Engine

The game uses a custom-built engine, rather than relying on a pre-existing engine. This custom engine provides complete control over the game, but it also means that some features and functionalities are not as well documented, and they may not be accessible for modification. It also means that the core functionality and rendering is likely handled by the game engine, and is not visible to the script files.

Script-Driven System

The core of the game's functionality is controlled by scripts. This includes, but it is not limited to:

* AI behaviour.
* User interface (UI) elements.
* Gameplay mechanics and logic.
* Visual elements and rendering.

These script files are plain text files with the `.script` extension. They can be opened and edited with any standard text editor. This script-driven approach offers immense flexibility, allowing modders to change many aspects of the game. However, it also requires a thorough understanding of the codebase and the different interdependencies between the script files.

Object-Oriented Design

The codebase is organized using object-oriented principles with base and derived classes. This system helps to organize the code, and it makes it easier to manage, but it can also be difficult to locate where different functionalities are defined. This means that you may have to examine multiple script files to fully understand a specific system.

Modular Systems

The game systems are broken down into separate script files. This modular approach simplifies organization, maintenance, and reuse of the code. However, it also makes it harder to see how the different components are interacting with each other, and it can make it more difficult to debug or change specific systems.

Legacy Code

A significant portion of the game's code is reused from G5 Software's previous game, "Whirlwinds over Vietnam". This explains some of the quirks and limitations in the code, and it means that some systems may not be fully functional, or that some parts of the code may be obsolete. This legacy code also means that some systems may not be fully implemented, and that the code structure may not be optimal.

File Formats

*   **Script Files:** As mentioned above, script files have the `.script` extension and are plain text files.
*   **Texture Files:** Texture files have the `.tex` extension, and they are actually in the DirectDraw Surface (.dds) file format. You will need a special image editor that supports .dds files to modify them.

Key Game Systems

This section outlines the key game systems in "T34 vs Tiger," based on our investigation of the game's script files.

Terrain System

The terrain system in "T34 vs Tiger" is responsible for creating and rendering the game's landscapes. It uses a heightmap to define the terrain's shape and a layered texture system for surface details. The terrain system also uses a microtexturing system to add fine details at different distances.

Heightmaps

The base terrain is created using heightmaps. These heightmaps are stored as raw data in `.raw` files and they are used to specify the height and shape of the landscape. The different terrain layers are defined using bitmap images, and these layers are used to add additional details to the terrain, including different textures or visual effects.

Microtexturing

The game uses a layered microtexture system to add fine details to the terrain surface. This system uses different texture layers that are faded in and out based on the camera distance. The microtexture system is controlled by the following variables:

*   **`CloseMicroTextureScale`:** This variable defines the scale at which microtextures are applied at close distances. This is the scale of the textures when they are rendered close to the player camera.
*   **`NearMicroTextureScale`:** This variable defines the scale at which microtextures are applied at medium distances. This is the scale of the textures that are rendered at medium distances.
*   **`FarMicroTextureScale`:** This variable defines the scale at which microtextures are applied at far distances. This is the scale of the textures when they are rendered at the far distances.
*   **`MicroTexNearMin`:** This variable defines the distance at which the near microtextures start to fade in. This is the distance at which the near microtextures begin to appear.
*   **`MicroTexNearMax`:** This variable defines the distance at which the near microtextures completely disappear. This is the maximum distance for the near microtextures.
*   **`MicroTexFarMin`:** This variable defines the distance at which the far microtextures start to fade in. This is the distance at which the far microtextures begin to appear.
*   **`MicroTexFarMax`:** This variable defines the distance at which the far microtextures completely disappear. This is the maximum distance for the far microtextures.
*   **`MicroTexDistFactor`:** This is a vector value that seems to be related to the calculation of the fade distances for the micro textures, and to define the overall look of the fade effect.
*   **`MicroTexCloseMax`:** This variable specifies the maximum distance for the close microtexture.
*   **`NoiseTexturePoints`:** This variable defines a quantity of random points that are used for the generation of the texture of noise on the terrain surface.
*   **`NoiseTextureAmplitude`:** This variable controls the maximum amplitude of the noise on the terrain surface.

These settings can be found and modified in the `CBaseTerrain.script` and `CBaseMegaTerrain.script` files.

Terrain System

The game uses the `TerrainDetail` variable from the `GameSettings` to control the detail level of the terrain. However, this variable is mostly controlled by the core game engine, and cannot be changed through the script files. The core engine is responsible for generating the terrain mesh, and for updating the detail level based on this setting.

The following variables are used to control the terrain details:

* **`MinimumResolution`:** This variable defines the minimum resolution of the terrain mesh.
* **`DesiredResolution`:** This variable defines the desired resolution of the terrain mesh.
* **`ShoreLineDetail`:** This variable defines the level of detail of the shoreline.
* **`OnTerrainDetailChanged()`:** This function is called when the game detail settings are changed, and it calculates the new values for the terrain rendering.
* **`UpdateTerrainDetails()`:** This function is used to update the terrain details, but the implementation of this function is located in the core game engine, and it cannot be modified with scripts.

These variables are used by the `OnTerrainDetailChanged()` function to set the rendering details, but the exact implementation is not accessible through the scripting system. The `TerrainDetail` setting can be set by the player in the game's option menu, and it is used to control the different terrain detail settings.

Differences from "Whirlwinds over Vietnam"

It's important to note that the terrain system in "T34 vs Tiger" has been modified compared to its predecessor, "Whirlwinds over Vietnam." Here are some of the key differences that can be found in the `BaseTerrain.script` files:

* **Microtexture Scales:**
    * In "Whirlwinds over Vietnam," the microtexture scales (`CloseMicroTextureScale`, `NearMicroTextureScale`, `FarMicroTextureScale`) are set to much smaller values (e.g., `0.0625`, `1.0`, `16.0`) compared to "T34 vs Tiger" (e.g., `0.5`, `2.0`, `32.0`). This suggests that "Whirlwinds over Vietnam" used higher resolution microtextures that were applied more often on the terrain surface. The much lower scale settings used in "T34 vs Tiger" means that the textures are rendered at a lower resolution, and that they are repeated less often.
* **Microtexture Fade Distances:**
    * The fade distances for microtextures (`MicroTexNearMin`, `MicroTexNearMax`, `MicroTexFarMin`, `MicroTexFarMax`) are significantly larger in "Whirlwinds over Vietnam," which would mean that the transition between the different texture levels would be smoother. The T34vT version uses much smaller fade ranges, which would lead to a more abrupt transition between the different textures.
* **Noise Texture Settings:**
    * The settings for the noise texture (`NoiseTexturePoints` and `NoiseTextureAmplitude`) are slightly different, which means that the noise that is applied to the terrain surface would look slightly different in the two games. The T34vT uses more noise points, and a lower overall amplitude.
* **Water Texture Scale:**
    * The water texture scale is set to a much higher value in T34vT, which suggests that the water textures are rendered at a lower resolution in this game.
* **Water Settings:**

    * Several settings related to the water rendering are different, including the `WaterMicroTexScale`, `WaterPenaTexScale`, `ClipPlaneShift`, and `AnimationSpeed` settings.
* **Material settings:** The `CTerrainMaterial` settings are significantly different, and that the texture names and overall rendering parameters are different between the two games.
*   **Other Settings:**
    *   The T34vT includes other settings, such as `MicroTexCloseMax` and `NormalNoise`, which are missing from the WoV version.

These changes indicate that the terrain rendering system was modified and optimized for "T34 vs Tiger," and that the new settings were optimized for that specific game, and its rendering system. These differences can also provide insight into how these settings work, and what modding possibilities they provide. Understanding these differences is key for modders who are trying to change the way that the terrain is rendered, and how they can create new mods for the game.

View-Specific Rendering Differences

It's important to note that the terrain rendering can behave differently depending on whether the game is viewed in first-person or third-person perspective.

*   While modifications to the microtexture settings have improved the appearance of the terrain in third-person view, they may not fully resolve the issues in first-person view.
*   Specifically, the "white areas" or other rendering artifacts that were previously visible at far distances may still be present in the first-person view, while they are significantly reduced or eliminated in the third-person view.
*   This difference suggests that the core game engine may use different rendering techniques, camera settings, or other optimizations that affect the visual quality of the terrain depending on the view.
*   Therefore, modders should always test their modifications in both first-person and third-person perspectives to ensure that the desired effects are correctly applied to both views, and that no issues are visible in any of the different camera views.

Forest System

The forest system in "T34 vs Tiger" is responsible for creating and rendering the game's forest regions. It uses a layered approach, similar to the microtexture system, to create the illusion of dense forests, and it also uses different types of forest elements that are rendered at different distances.

The forest system is controlled by the following functions:

*   **`RegisterForestRegion()`:** This function is used to register and configure a horizontal forest region. It takes the following parameters:
    *   **`ZoneMask`:** An array of zone masks to which the forest region is applied.
    *   **`Materials`:** A material manager component containing the forest skin.
    *   **`Texture`:** The texture to be used for the forest.
    *   **`Levels`:** An array of level names that define different lighting levels.
    *   **`Densities`:** An array of float values that define the density of each layer.
    * **`StartFadeDistances`:** An array of float values that define the distances at which each level will start to fade in.
    *   **`MaxFadeDistances`:** An array of float values that define the distances at which each level will be completely faded out.

*   **`RegisterVerticalForest()`:** This function is used to register and configure a vertical forest region. It takes the following parameters:
    *   **`ZoneMask`:** An array of zone masks to which the vertical forest region is applied.
    *   **`Texture`:** The texture to be used for the vertical forest.
    *   **`Rings`:** An array of float values that define the number of rings of the vertical forest elements.
    *   **`Height`:** A float value that defines the height of the vertical forest elements.

The forest system also uses a variety of other settings to control the look and behavior of the forests, including:

*   **`VertForestDistMin`:** This variable defines the minimum distance for the vertical forest elements.
*   **`VertForestDistMax`:** This variable defines the maximum distance for the vertical forest elements.
*   **`VertForestAngleMin`:** This variable defines the minimum angle for the vertical forest elements.
*   **`VertForestAngleMax`:** This variable defines the maximum angle for the vertical forest elements.
*   **`VertForestBeginFadeDist`:** An array of float values that define the distances at which the vertical forest elements will start to fade in.

*   **`ForestTextureScale`:** This variable defines the scale of the forest texture.
*   **`ForestBorderScale`:** This variable defines the scale of the border for the horizontal forest.
*   **`ForestShadowBorderScale`:** This variable defines the scale of the shadow border for the horizontal forest.
*   **`ForestAlphaScale`:** This variable defines the alpha scale for the horizontal forest.
*   **`ForestShadowAlphaScale`:** This variable defines the alpha scale for the shadows of the horizontal forest.
*   **`ForestBeginFadeDist`:** An array of float values that define the distances at which the horizontal forest elements will start to fade in.

* **`ForestLightFadeDist`:** An array of float values that define the distances at which the horizontal forest lighting will start to fade in.
* **`FakeForestBeginFadeDist`:** An array of float values that define the distances at which the fake forest elements will start to fade in.
*   **`FakeForestLightFadeDist`:** An array of float values that define the distances at which the fake forest lighting will start to fade in.

These settings can be found and modified in the `CBaseTerrain.script`, `CBaseMegaTerrain.script` and in the mission specific `Terrain.script` files.

The forest system uses a layered approach that is similar to the microtexture system, and that is used to control the density of the forest, and the distance at which different forest elements are rendered. The layered system is also used to create a smooth transition between the different forest elements, and to control the performance of the game when rendering complex forest regions.

Water System

The water system in "T34 vs Tiger" is responsible for rendering the water surfaces and the associated effects, including reflections and wave animations. The water system is complex, and it is controlled by a large number of settings that define the look and behavior of the water.

The water system is controlled by the following variables:

* **`WaterMirrorDetail`:** This variable controls the quality of the water reflections, and how the different reflections are rendered on the water surface.
* **`WaterMirrorTexSize`:** This variable specifies the size of the texture that is used for the water reflections.
* **`WaterMirrorMasks`:** This array defines what objects are rendered in the water mirror. It uses an array of object types that will be visible in the water reflections, and object types that will not be visible in the water reflections.
* **`WaterMirrorLodMasks`:** This array defines what object types are rendered in the water mirror at different levels of detail. It uses an array of object types that will be visible in the water reflections at different detail levels.
* **`WaterMirrorLodDistances`:** This array defines the distances at which the different levels of detail are used for the water mirror. It uses an array of floating values that are used to define the ranges for the water LOD.

* **`LargeWavePeriod`:** This variable defines the period of the large waves on the water surface.
* **`LargeWaveWidth`:** This variable defines the width of the large waves on the water surface.
* **`LargeWaveHeight`:** This variable defines the height of the large waves on the water surface.

* **`WavesTexDimension`:** This variable defines the texture dimension for the waves.
* **`WavesTimeScale`:** This variable controls the animation speed of the water surface waves.
* **`WavesBumpScale`:** This variable controls the bump scale for the waves.
* **`WavesLumScale`:** This variable controls the luminance scale for the waves.
* **`WavesLumOffset`:** This variable controls the luminance offset for the waves.
* **`WaterMirrorFactor`:** This variable controls the factor for the water mirror.

* **`RipplesWavePeriod`:** This variable defines the period of the ripples on the water surface.
* **`RipplesBrightness`:** This variable controls the brightness of the ripples on the water surface.

* **`WaterColor`:** This variable defines the color of the water, and can be used to define the overall look of the water.
* **`WaterBorderColor`:** This variable defines the color of the border of the water surface.
* **`WaterFresnelPower`:** This variable controls the fresnel effect on the water surface.
* **`WaterTexScale`:** This variable defines the scale of the water texture.
* **`WaterMicroTexScale`:** This variable defines the scale of the microtextures for the water.
* **`WaterPenaTexScale`:** This variable defines the scale of the penalty textures for the water.
* **`ReflectDistortion`:** This variable controls the distortion of the water reflections.
* **`BumpSlopeness`:** This variable controls the slopeness of the bump effect on the water surface.
* **`ReflectTextureShift`:** This variable controls the texture shift of the water reflections.
* **`WaterBorderScale`:** This variable defines the scale of the border of the water surface.
* **`WaterPenaBorderScale`:** This variable defines the scale of the penalty border for the water.
* **`BorderOffset`:** This variable defines the offset for the border.

* **`ClipPlaneShift`:** This variable defines the shift for the clip plane.
*   **`AnimationSpeed`:** This variable defines the animation speed for the water effects.
*   **`WaterClearColor`:** This variable defines the clear color for the water.

These settings can be found and modified in the `CBaseTerrain.script` and `CBaseMegaTerrain.script` files.

The water system uses a combination of different settings to create realistic looking water surfaces, and the settings can be used to customize the visual quality, and the performance of the water rendering. The water system also includes a basic Level of Detail system, that is used to render different object types in the water reflections, and to reduce the rendering cost for far away objects.

Game Settings

The `GameSettings` component is responsible for storing various game settings that control different aspects of the game, including graphics, audio, and gameplay. These settings are read at the start of the game and are primarily used to configure the different systems.

It is crucial to understand that almost all settings in the `GameSettings` component are defined as `final static`. This means that they are constant values that are set during compilation and *cannot be directly modified during gameplay* through script files. This is a significant limitation for modding, as most of the settings can not be changed using scripts.

The `GameSettings` component includes the following important settings:

*   **`TerrainDetail`:** This variable controls the level of detail for the terrain mesh. It is used by the `OnTerrainDetailChanged` function to set the different rendering settings. However, the actual implementation is controlled by the core game engine. This setting can be changed by the user in the game's option menu, and is used to control the terrain rendering quality.
*   **`WaterDetail`:** This variable controls the level of detail for the water rendering. Similar to the `TerrainDetail` variable, the actual implementation is controlled by the core game engine, and the variable can not be modified by the scripts. This setting can be changed by the user in the game's option menu, and is used to control the quality of the water rendering.

The `GameSettings` component also includes a large number of other settings that are used to control the behavior of the game, including:

*   **Graphics Settings:** `WindowWidth`, `WindowHeight`, `RefreshRate`, `ColorDepth`, `WindowMode`, `LightingModel`, `RenderDetail`, `ForestDetail`, `GrassDetail`, `GammaCoef`, `MaxLightsQty`, `TextureBestLOD`, `ShadowDetail`, `AntiAliasing`, `UseInstancing`, `IsBloomEnabled`, `InteriorDetail`, and `ObjectsDetail`.
*   **Sound Settings:** `ChannelsQty`, `SoundFrequency`, `SoundBitRate`, `SoundVolume`, and `MusicVolume`.
*   **Gameplay Settings:** `DifficultyLevel`, `TargetingMode`, `Crosshair`, `WingmanMenuStyle`, `CockpitDevicesColor`, `ManualControlMode`, `TargetsOnMap`, `OrientationMode`, `DynamicLift`, `IntelligentCompass`, `WindEffectMode`, `FlaterMode`, `AircushionMode`, `PayloadMode`, `ShockwaveMode`, `VortexRingMode`, `RetreatingBladeStallMode`, `InjuryMode`, `CockpitMode`, `FlaresControlMode`, `InverseY`, `MouseSensitivity`, `IsTorqueControl`, and `NavpointNavigationMode`.
* **Input Settings:** `ForceFeedbackEnabled`, `ForceFeedbackStrength`, `JoystickSensitivity`, `MinJoystickDeadZone`, `MaxJoystickDeadZone`, `JoystickDeadZone`, `ArcadeSliderDeadZone`, `InvertRotateLRAxis`, `InvertMoveFBAxis`, `InvertMoveLRAxis`, `InvertSpeedAxis`, `InvertBladesSpeedAxis`, and `InvertCameraZoom`.

The `GameSettings` component is defined in the `GameSettings.script` file, and it contains all of the static variables that are used to initialize the game.

While we cannot directly modify these settings through scripts, understanding them is essential for modding, as they influence how the different systems behave.

AI System

The Artificial Intelligence (AI) system in "T34 vs Tiger" controls the behavior of the non-player units (primarily enemy tanks). The AI system is primarily controlled through scripts, which makes it a good candidate for modding and customization.

Here are some of the key aspects of the AI system that we have identified:

*   **Script-Driven Behavior:** The AI behaviors are primarily defined through script files. This is a major advantage for modders, as it allows for fine-grained control over how the AI units behave.
*   **Simple Radar:** The AI uses a basic radar system to detect nearby units. This system seems to be limited and it does not take into account the terrain or other objects. The radar system is mostly used to find the player, and for tracking the enemy units.
*   **Basic Targeting:** The AI uses a basic targeting system, and it prioritizes targets using a simple ranking system. This system may not be fully functional, and it may not always choose the best targets.
*   **Aggressive Behavior:** The AI is designed to be aggressive, and it usually tries to move towards the player's unit, and attack it at the closest range. This may not be realistic, or challenging for the players, and the AI may not act according to the different situations.
*   **Limited Awareness:** The AI units do not seem to take into account many outside factors, such as terrain or the line of sight. This limitation may make the AI units predictable, and not very challenging for experienced players.
*   **State-Based Animations:** The AI uses a state-based animation system to control the look of the units and to implement the different behaviours. This system can be complex, but also very flexible.

While the core logic of the AI may be complex, its script-driven nature makes it a great target for modders who want to change the way the enemies behave. It also means that modders can implement new AI behaviors, or add other features to make the game more engaging and challenging.

Camera System

Testing of ZNear and ZFar Settings:

During our investigation, we tested the `ZNear` and `ZFar` settings, and discovered that, while these settings appear to be related to view distance in theory, they *do not* have a significant impact on the draw distance of terrain or other objects in the game world.

* Modifying `ZNear` and `ZFar`, even to values as extreme as those used in the ZeeWolf mod, did not result in any visible changes to how far objects are rendered, and that those settings did not affect the core rendering of the game.
* The `ZNear` and `ZFar` settings also do not have any noticeable impact on the microtextures, or the visual rendering of the terrain, and that they are not related to the issue with the first-person view.
* It is still possible that these settings are used for other aspects of the rendering pipeline or are used by other game systems, but that those uses are not easily visible using the different camera views.

This means that the draw distance and other aspects of the rendering system are likely controlled by other parts of the game engine, that are not easily modifiable using the scripting system. These settings may also be directly controlled by the game engine, and that we can not use the script files to modify them.

Testing of Render Target Masks

We also tested the `SetMask` function, used with the `RenderTargetControl` in `CCockpitGunLayerScreen` and `CCockpitCommanderScreen`, to see if it would allow us to change the rendering of the terrain, or to fix any other visual issues, but it appears that this is not the case.

* Modifying the `SetMask` to include or exclude different classifications, or to use only the terrain classification, or to exclude all of the available settings did not have a visible effect on the way that the terrain is rendered, and that this function is probably not related to the core rendering process, or to how the first-person camera is being rendered.
* It appears that the main purpose of `SetMask` is to control which UI elements or other specific objects are rendered in the different viewports, and not to control how the core game engine handles the rendering of the terrain, or other game objects.

This means that the `SetMask` function, although it controls what is being rendered in the viewports, does not influence the visual quality, or the draw distance of the different objects, and it is probably used only to exclude or include specific objects in the viewport.

Modding Guidelines and Techniques

This section provides guidelines and techniques for modding "T34 vs Tiger," based on our investigation of the game's architecture and systems.

Script Analysis

The most important skill for modding "T34 vs Tiger" is the ability to analyze and understand the game's script files. Here are some helpful tips:

*   **Text Editor:** Use a text editor with syntax highlighting to make the code easier to read and understand. Some text editors will also automatically show you if the script file contains an error.
*   **Code Structure:** Pay attention to the structure of the code, including classes, functions, and variables. Try to understand how the code is organized, and how the different systems are implemented.
* **Comments:** Look for comments in the code. While not always present, comments can provide valuable information about what a specific section of the code is intended to do.
*   **Experimentation:** Don't be afraid to make changes and experiment with the code, but always back up your original script files.
* **Start Small:** Begin with small modifications and then gradually move on to larger and more complex changes. Start with a specific variable and then test the change in the game.
* **File Structure:** Pay attention to the file structure, and try to understand how the different files are organized. Understanding the file structure will help you to navigate the different script files, and to locate the settings that you want to modify.

Keyword Searching

Use keyword searching to find specific code sections or variables. Here are some useful keywords:

*   **\`LOD\`**, **\`detail\`**, **\`distance\`**, **\`heightmap\`**, **\`render\`**, **\`mesh\`**, **\`texture\`**, **\`level\`**: These keywords are useful for finding code that is related to the rendering and detail levels of the different systems.
*   **\`Terrain\`**, **\`Forest\`**, **\`Water\`**, **\`AI\`**, **\`Menu\`**, **\`Settings\`**, and other relevant terms: Use these keywords to find code that is related to a specific system or functionality.
*   **Specific Function Names:** Use specific function names such as \`UpdateTerrainDetails\`, \`OnTerrainDetailChanged\`, \`RegisterForestRegion\`, and others, to locate where those functions are defined or used in the different script files.
* **Variable Names:** Use specific variable names such as \`TerrainDetail\`, \`WaterDetail\`, \`MicroTexNearMax\`, or others to locate where those variables are defined or used.

Modding Limitations

It's also crucial to be aware of the limitations of modding "T34 vs Tiger":

*   **Core Engine Logic:** Much of the core game logic and rendering is implemented within the game engine, and *is not accessible through scripts*. This limits our ability to make fundamental changes to how the game works or renders the different systems.
*   **\`final static\` Settings:** Most of the settings in the \`GameSettings\` component are defined as \`final static\`, and cannot be modified through scripts. This means that many of the game settings, including the graphics and the detail settings, cannot be changed.

*   **Limited Terrain LOD Control:** The terrain LOD is mostly controlled by the core game engine, and it is very limited, and you can only modify the detail settings indirectly through the `TerrainDetail` and `WaterDetail` variables.
* **Limited Texture LOD:** The texture LOD is very limited, and is mostly implemented using the microtexture system. The main textures are usually rendered at the maximum resolution, and do not have a level of detail system.
* **Legacy Code Issues:** Due to the large amount of reused code, and the premature release, the game contains a lot of bugs, and inconsistencies, that may make some modding tasks more difficult.
*   **Reverse Engineering:** If you want to modify the core game engine, you would have to use reverse engineering techniques, which is a much more difficult modding task.

Despite these limitations, there are still many areas where modders can make significant changes and create new experiences for the game.

Camera System

Testing of ZNear and ZFar Settings:

During our investigation, we tested the `ZNear` and `ZFar` settings, and discovered that, while these settings appear to be related to view distance in theory, they *do not* have a significant impact on the draw distance of terrain or other objects in the game world.

*   Modifying `ZNear` and `ZFar`, even to values as extreme as those used in the ZeeWolf mod, did not result in any visible changes to how far objects are rendered, and that those settings did not affect the core rendering of the game.
*   The `ZNear` and `ZFar` settings also do not have any noticeable impact on the microtextures, or the visual rendering of the terrain, and that they are not related to the issue with the first-person view.
*   It is still possible that these settings are used for other aspects of the rendering pipeline or are used by other game systems, but that those uses are not easily visible using the different camera views.

This means that the draw distance and other aspects of the rendering system are likely controlled by other parts of the game engine, that are not easily modifiable using the scripting system. These settings may also be directly controlled by the game engine, and that we can not use the script files to modify them.

Conclusion

"T34 vs Tiger," despite its limitations, offers a great opportunity for modders to explore and customize various aspects of the game. While the core game engine and many settings are not directly accessible through scripts, there are still many areas where modders can make significant changes and create new experiences.

By focusing on the systems that can be controlled through scripts, such as the microtexture system, the water rendering, the AI behaviors, and the various gameplay settings, modders can implement a wide variety of changes. We can also try to fix the many bugs, and inconsistencies that were introduced due to the reuse of old code, and the premature release of the game.

This document provides a solid foundation for understanding the game's architecture and for starting your modding journey. By studying this document, and by analyzing the script files carefully, you will be able to create new and exciting mods for T34 vs Tiger, and to expand the core gameplay with new features and changes.

The key for a successful modding is to:

*   **Understand the systems:** Spend time analyzing the game's systems, and how they interact with each other.
*   **Start Small:** Begin with small changes, and then gradually expand to more complex modifications.
*   **Test Thoroughly:** Test all of your changes carefully, and make sure that they are working as expected.
*   **Be Creative:** Use your imagination, and try to create unique modifications that add a new layer of complexity to the game.

We encourage you to join the modding community, share your discoveries, and contribute to making "T34 vs Tiger" an even better and more engaging game. With the right level of skill, and by having a deep understanding of the code, you can make amazing changes, and add unique features to the game.