Ubiquitous Reality

Members:

Alexander Claridge

Tejaswa Rizyal

Abhigyan Gandhi

Sahil Puri

Michael Asiamah

With thanks to Markus Billeter

Overview

The Ubiquitous Reality game engine (UBQR) brings style and artistic design to the forefront of game development. Inspired by a retro, sci-fi aesthetic, the engine specialises in a car/racing style gameplay. UBQR engine allows rapid development by leveraging a powerful component system and event handling that allows the developer to have complete control of the gameplay.

UBQR provides multiple features for a game developer to use, but the main focal points for the engine are its **renderer**, **architecture systems** and **physics**.

Figure 1. Title card scene of "Axiom Swerve" Rendered in UBQR.

Rendering

Default Features

- Multi Sampling Anti-Aliasing
- PBR BRDF Shading
- Shadow Mapping
- Normal Mapping
- Skyboxes per Scene
- Directional Light per Scene
- Light Attenuation
- Colour Grading

Post Processing Effects

- Multipass Bloom
- Chromatic Aberration
- Toon Shading
- Artistic Motion Blur
- Sharpening Filter
- Energy Ball Rendering
- Fast Approximate Anti-Aliasing
- RGB Channel Filtering

Artistic Stylisation

The out of the box features contain default styles and further utilities to set up the scene of your choice. The post processing effects give the user a detailed and configurable setup that allow them to be used not only for pure aesthetics but also in gameplay. For example, chromatic aberration for health and damage effects, and toon shading for smoke particles. We also provide both sharpening and FXAA filters that especially help with blurry or aliased textures allowing custom control without taking anything away from core game development.

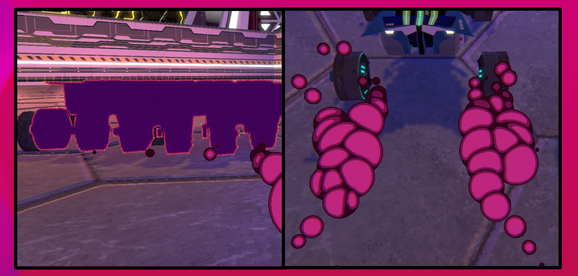

Figure 4. Smoke and motion blur effect used for object specific post processing.

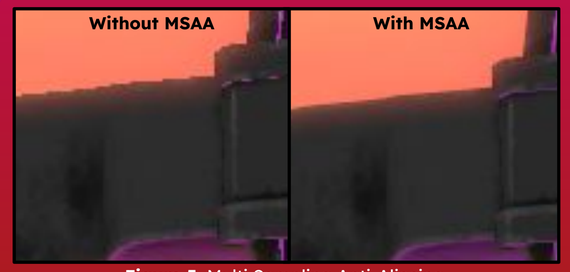

Figure 5. Multi Sampling Anti-Aliasing.

Architecture

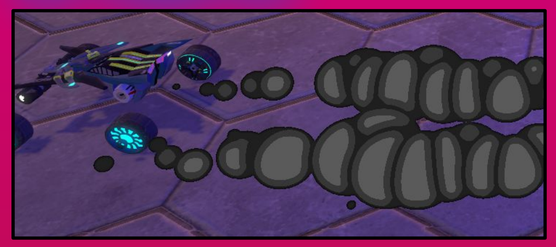

Figure 2. Smoke spheres generated using the engine's custom gameware/pooling system.

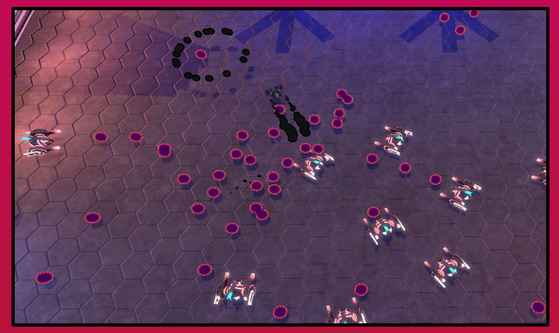

Figure 3. Enemies individually programmed to spawn custom bullets and particles, all reacting to the event handling system.

GameWares - Entity Component System

GameWares are an interface into UBQR's Entity-Component System; they represent an entity which can have any combination of different components. The engine then supports switching between scenes of these entities. Basic GameWares act purely as an interface, leaving no footprint on the engine once initialisation has been finished.

Custom GameWares

The developer can inherit from GameWares to create **custom** programmed GameWares. UBQR is specially designed to handle custom GameWares such that the developer can use any number of them in the engine's pooling system.

Recipe Pooling

This system allows the developer to create a "**Recipe**" of multiple GameWares and spawn any number of these Recipes dynamically. This system is used by the showcase game as seen in **Figure 2** and **Figure 3**.

Event Handling system

The developer can implement their own **events** and **listeners**, or create listeners for engine defined events. Using a queuing system. UBQR supports custom GameWares which allows for easy custom event based programming.

Audio

UBQR integrates the SoLoud audio library to give game developers the ability to play audio on command, allowing for background tracks, as well as **3D audio**. This is directly integrated with the GameWare system.

Input

UBQR provides support for **controller** input and **Rumble Feedback** alongside **keyboard and mouse**. The input system is designed to be dynamic and modular, giving easy options for runtime **remappings**.

User Interfaces

Debugging

UBQR offers support for debugging each GameWare in a scene, displaying debugging information per component of a GameWare.

In-Game UI

UBQR gives access for custom styles and integrated use of **image rendering** with vulkan. So the game developer is able to create their own **custom UI**.

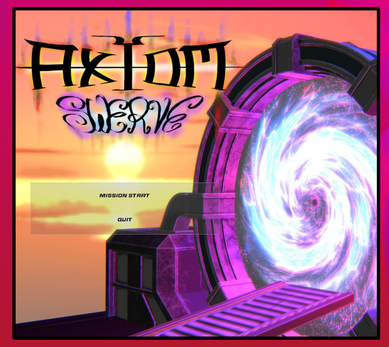

Figure 6. Main menu scene with interactable buttons.

Physics

Jolt Physics Engine

UBQR utilizes the Jolt Physics library for its physics system, utilizing their multithreaded system for collision handling, rigidbody calculations and layering.

Camera

The game developer is offered multiple different camera types. One of these camera types, "follow", uses physics raycasting to determine its correct position.

Vehicle

Finally, the engine provides support for car/vehicle physics within the engine, including tire animations.

Results

UBQR provides a performative engine while implementing unique renderpasses and post processing effects, allowing for heavy stylization. Rendering a minimal scene is as easy as setting up a model and a camera which can be done in 10 lines of code. Using UBQR's default physics, custom GameWares and a car model achieves a simple game loop. Complex game mechanics are easy to implement with Recipes, pooling, and the event system allowing the developer a direct interface into rendering and gameplay. The debugging features allow the user to not only be able to see all of the details of entities in game, but also find the perfect visual effects to create the aesthetics they desire. UBQR provides all of this and more, while being highly performant on both CPU and GPU.

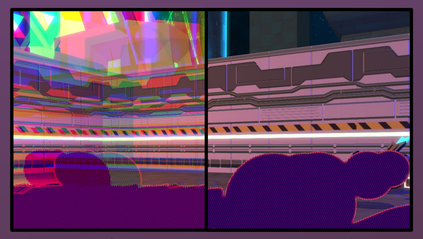

Figure 7. Difference between a scene with and without strong chromatic aberration.

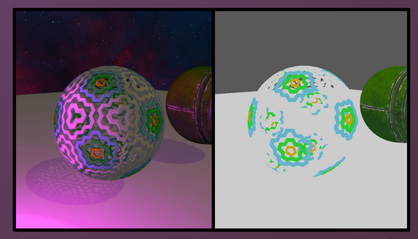

Figure 8. UBQR's lighting and shadow rendering next to just their textures.

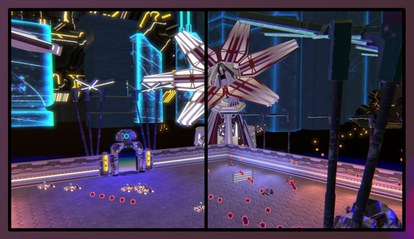

Figure 9. Difference between a scene with and without emissive bloom.

UNIVERSITY OF LEEDS