Vr Flight

From Ultra Real

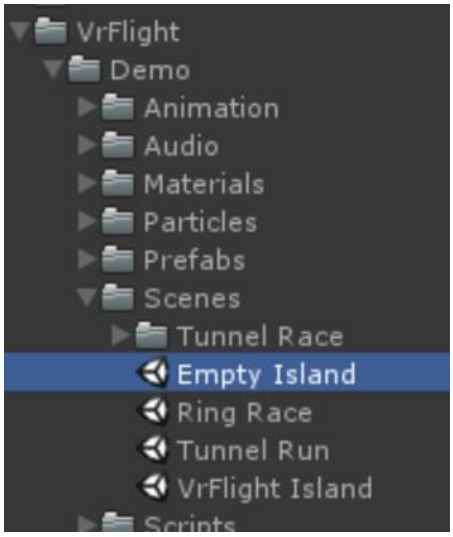
Support

Web Site http://nisbetcraig.wixsite.com/ultrarealassets
E-mail nisbet.craig@yahoo.com

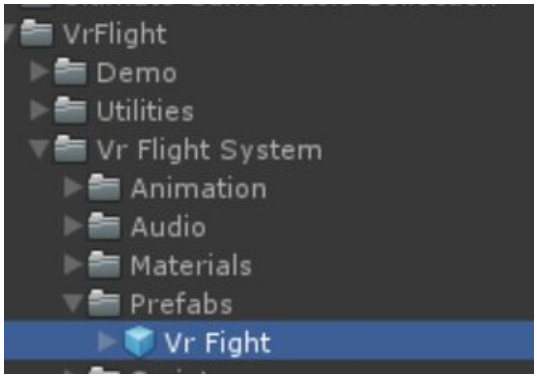
Quick Start Guide

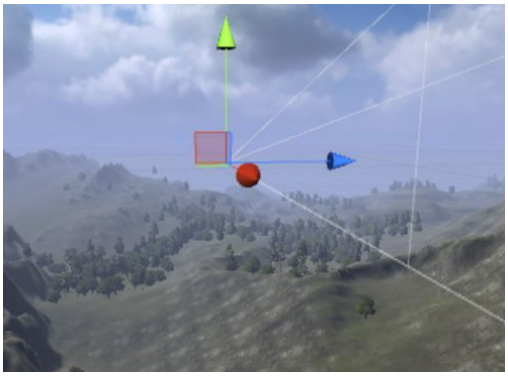
1. Start a new empty project and a new empty scene.
2. Download and import the VrFlight asset package from the Unity Asset Store.
3. Find and load the scene “Empty Island” from the **VrFlight/Demo/Scenes directory**.

   This will give you an empty landscape to experiment with.

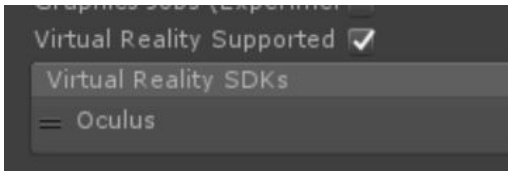
4. 

5. Find the **VrFlight** prefab in the **VrFlight/VrFlight System/Prefabs** directory and move it into the scene. Be sure to move it somewhere up above the ground.





6. Make sure “**Virtual Reality Supported**” is checked in Player Settings under the Other Settings tab.



7. Press **play**...oh yeah and put on your VR headset.

In the future, if you would like to only import the VrFlight prefab and scripts and not all the demo content, simply import only the VrFlight System folder from the asset package.

Flight System Controls

Look Flight: Look in any direction that you would like to fly.

Turning: Tilt your head left and right to turn or perform sharp u-turns.

Controller Input : (Optional) Right trigger will accelerate your flight and the left trigger will decelerate your flight.

VrFlight Prefab Breakdown

Basic Features

1. Natural flight motion that uses head looking and tilting.
2. Input converter scripts that turns your VR Headset into a game controller with Axis inputs.
3. Animated motion sickness blinders that respond to head motion and the proximity of game world surroundings.
4. Animated camera wind effects that react to players actions.
5. Simulated gravity acceleration effects. You'll move faster as you dive to the ground.
6. Simulated death on collision.
7. Can use additional input from a game controller for flight acceleration

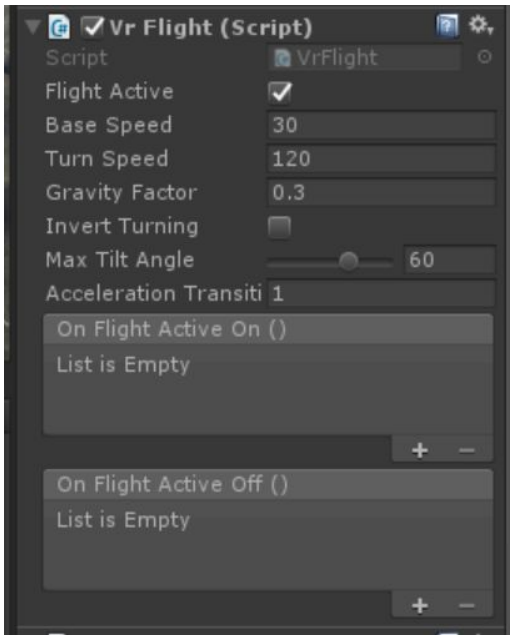
Code Features

1. Scripts contain both Unity Events and C# Events for important flight events.
2. Code architecture is built on the use of C# Interfaces rather than direct script access, allowing for greater customization.
3. All code is commented.

Scripts

There are several scripts on the VrFlight prefab that work together to create the VrFlight experience. Here we will try to give a brief description of each script and their general usage.

VrFlight



The Vr Flight script controls the forward and turning motion during flight. Here you can control many of the flight parameters to your liking.

You'll have access to a speed modifier variable from script that allows you to create acceleration effects like boost power ups.

You can also simulate how the players head pitch controls the speed. For instance, it can slow down if the player is looking up as well as speed up if they are looking down. This can help simulate the effect of racing to the ground with gravity.

Inspector Properties

Flight Active: This enables and disables the flight system. This is extremely useful if you want to use the flight camera as a standard VR camera when you are not in flight mode. There are examples of this in the demos.

Base Speed: This is the base velocity that the flight system moves by default. Think of it like an airplane's forward momentum that is needed to maintain flight.

Turn Speed: Angular speed that the flight system will turn when the player tilts their head.

Gravity Factor: This value affects the speed change as the player pitches their head up and down. By default this is set to a positive value to simulate gravity pulling the player down faster as they look down, but this can just as easily be a negative number to simulate the opposite effect such as a diver swimming and moving faster as they look upwards.

Invert Turning: In testing it was discovered that the Gear Vr banking input was inverted for some reason. This issue may have been corrected already or will soon be corrected. Either way, this toggle provides a way to combat this issue if it arrives.

Max Tilt Angle: This is the maximum angle in degrees that the player can tilt their head in either direction to control the turning effect. 60 degrees felt about right in our testing.

Acceleration Transition Speed: The speed of a player's flight can be modified via script at run-time using the SpeedMod property. Acceleration Transition Speed controls the duration of the transition to the new speed, and helps the flight motion not look jerky.

OnFlightActiveOn and OnFlightActiveOff: Unity Events that fire off when your game activates and deactivates the flight system via the FlightActive Property.

Properties Available From Scripting

FlightActive: This enables and disables the flight system. This is extremely useful if you want to use the flight camera as a standard VR camera when you are not in flight mode. There are examples of this in the demos.

SpeedMod: This is a speed value that is added on top of the base speed. Use this to create acceleration effects like boost pickups. Here's an example of a coroutine that modifies the SpeedMod Property to boost the player for a duration.

```
IEnumerator AddBoost(VrFlight vrFlight, float speed, float duration)
{
    vrFlight.SpeedMod += speed;

    yield return new WaitForSeconds(duration);

    vrFlight.SpeedMod -= speed;
}
```

FlightActiveChanged: C# event that fires off when the FlightActive property has been changed.

Methods Available From Script

Reset(): Resets the flight systems speed values back to where they were set when the script awoke.

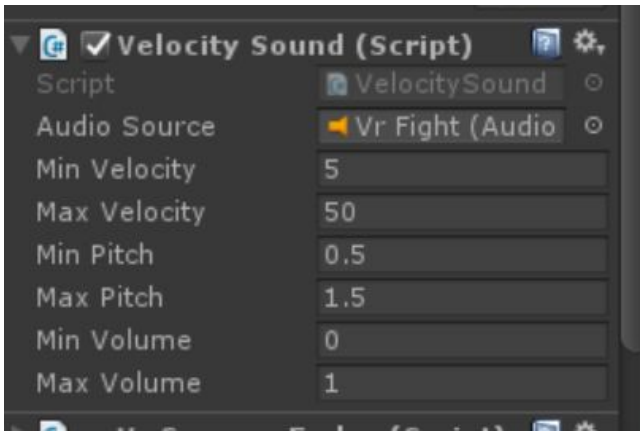
Float **GetVelocity():** Returns the velocity of the flight system.

Camera **GetCamera():** Returns the Camera attached to the flight system.

Vector3 **GetForward():** Returns the world forward vector of the flight system.

Transform **GetTransform():** Returns the cached transform from the flight system. Faster than using “tranform” directly.

ICameraFader **GetCamaraFader():** Returns the CameraFader interface.



Velocity Sound

The Velocity sound script simulates the effect of sound changing pitch and volume as the player moves at different speeds. This script is completely generic and will work on any GameObject that has a Rigid Body on it.

Provide the script with a minimum and maximum velocity speed and the script will scale the pitch and volume accordingly.

Inspector Properties

Audio Source: Reference to an audio source that will have it's pitch controlled.

Min Velocity: Minimum velocity for pitch and volume change.

Max Velocity: Maximum velocity for pitch ann volume change.

Min Pitch: Pitch value when the velocity is at the Min Velocity.

Max Pitch: Pitch value when the velocity is at the Max Velocity.

Min Volume: Volume value when the velocity is at the Min Velocity.

Max Volume: Volume value when the velocity is at the Max Velocity.

Special Note: While in play mode, this script will show the player’s current velocity, and max velocity reached as they play. Use this to help find your setting for Min and Max Velocity.

Vr Camera Fader

The Vr Camera Fader script acts as a translator between the VrCameraFade script that is located deeper in the prefab. The VrCameraFade is from the Unity Vr Samples

Methods available from script

Void **FadeIn():** Fades in from black using the default duration in the VrCameraFade script on the camera.

Void **FadeOut():** Fades out to black using the default duration in the VrCameraFade script on the camera.

Void **FadeIn(float duration):** Fades in from black over the duration entered.

Void **FadeOut(float duration):** Fades out to black over the duration entered.

IEnumerator **BeginFadeIn(float duration):** Fades in from black over the duration entered as a coroutine. Exits coroutine when the fade is complete.

IEnumerator **BeginFadeOut(float duration):** Fades out to black over the duration entered as a coroutine. Exits coroutine when the fade is complete.

Vr Flight Basic Input

The Vr Flight Basic Input is a simple Unity Input script. The VrFlight script uses this for it's player acceleration input. Usage of this script is completely optional.

If you're a coder, you can easily write your own version of this to support other control input systems like Rewired or CInput. Most of the scripts within this system rely on C# interfaces allowing for easy customization without the need to break the original code to accomplish what you want.

Inspector Properties

AccelerateSpeed: Speed that will be added to VrFlight's BaseSpeed value when the Accelerate Input is pressed

DecelerateSpeed: Speed that will be added to VrFlight's BaseSpeed value when the Decelerate Input is pressed. Can be a negative value.

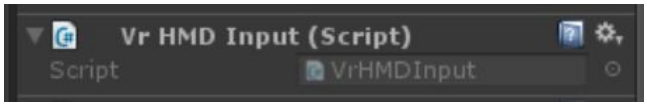
AccelerateInputName: Accelerate Unity input name.

DeceleateInputName: Decelerate Unity input name.

Methods available from script

Float **GetAccelerationSpeed():** Returns the acceleration speed.

Vr HMD Input



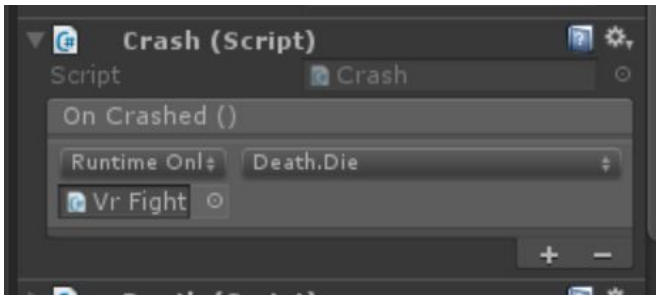
This is purely a technical script. Vr is new and things change A LOT. This is a translator script that gets the Input from the Head Mounted display and converts the inputs into something the flight system can use. It also creates axis inputs that can be used in a very similar fashion to the Unity input system's axis inputs.

This script is currently using the input information gathered from Unity's UnityEngine.VR.InputTracking system, but who knows, you may need to use a Vr Input device this isn't supported by Unity by default. If that is the case, different versions of this script can be written to support your desired input device.

Methods available from script

Vector3 **GetHmdGlobalForward():** Returns the forward direction of the camera relative to the VrFlight rig.
float **GetHmdHeadingAxis():** Returns the Heading in a axis value (-1 to 1).
float **GetHmdPitchAxis():** Returns the Pitch in a axis value (-1 to 1).
float **GetHmdBankAxis():** Returns the Bank in a axis value (-1 to 1).
float **GetHmdHeadingEuler():** Returns the heading angle in degrees from the Head Mounted Display and converts it relative to forward head facing.
float **GetHmdPitchEuler():** Returns the Pitch angle in degrees from the Head Mounted Display and converts it relative to forward head facing.
float **GetHmdBankEuler():** Returns the bank angle in degrees from the Head Mounted Display and converts it relative to forward head facing.
Vector3 **GetHmdLocalPosition():** Pass through for the raw data from the tracking system.
Quaternion **GetHmdLocalRotation():** Pass through for the raw data from the tracking system.

Crash



The Crash script sends an event to a desired target reporting that the player has crashed.

Note: Script contains a C# event as well.

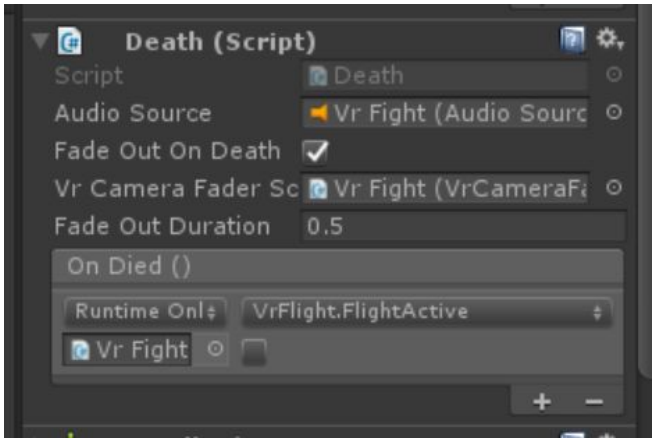
Unity Events

OnCrashed : Fires off this event if the flight system hits another collider

C# Events: (from script)

Crashed : Fires off this event if the flight system hits another collider

Death



This script can play a sound and cause the camera to fade to black when the player dies.

It can also be used to send events to other scripts letting them know that the player died.

Audio Source : Reference to the AudioSource for the death sound. use of this is completely optional.

FadeOutOnDeath : (Toggle)Tells the camera to fade out on death. Requires a ICameraFader reference.

VrCameraFaderScript : Reference to a script that implements the ICameraFader interface. This will most likely be the CamaraFader script. You can write a script that implement ICameraFader if you want to use your own custom camera transition system.

FadeOutDuration: Duration of the Camera fade out.

Unity Events

OnDied : Unity event that is called when the Die method is called.

C# Events

Died(from script) : Event that is called when the Die method is called.

Methods available from Script

Void **Die()** : Activates the death effects in this script and fires off Died events.

Auto Respawn

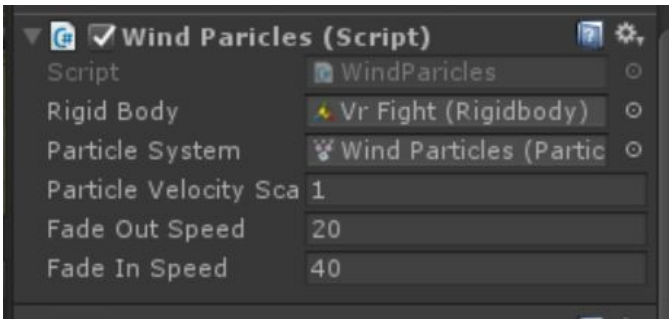
The Auto respawn script is provided as a demo of the Respawn function in the VrFlight script. With it on the VrFlight prefab, it will automatically respawn the player back to it's start position after 2 seconds.

Typically in a game, respawns would not be done in the player Game Object itself, but in a Game Manager somewhere else in the scene. If you would like to try this, remove the Auto Respawn script from the VrFlight prefab in your scene, and Create an empty Game Object at the scene level. Then add the SampleGameManager script to that Game Object. You can modify this script to make your own game manager.

Vr Tracking Negate

Currently there is no way to turn the Unity Vr position tracking off. Vr Flight does not rely on this feature so we need to turn it off. This script employs a transform trick to accomplish this.

Wind Particles



Wind particles animates the appearance of a particle system to simulate wind blowing past the camera.

This script doesn't require the flight system to work. It relies on the velocity from a Rigid body, so it can just as easily be used on any camera that just has a RigidBody on it.

It's important to note that this script does not do all the work of representing the wind effect on it's own. You will still need to make a particle system that is similar to the one on the VrFlight Prefab. It's simply a ring of particles that whiz past the camera. For most purposes we suggest just modifying the one provided in the VrFlight prefab.

Currently this script only works with forward velocity. This may be modified in future versions.

Inspector Properties

Rigid Body: Reference to a RigidBody, it uses this for velocity information.
Particle System: Reference to the particle system this script will affect.

Particle Velocity Scale: Use this to scale the velocity before it gets applied to the particle velocity. It testing, 1 to 1 scale velocity didn't always look right when applied to the particles. A value of 0.5 seemed look good.

Fade Out Speed: Velocity at which the particles would not be visible. This should be a slow speed.

Fade In Speed: This is the velocity when the particles are fully visible. This should be a high speed.