TRINUS VR FOR UNITY

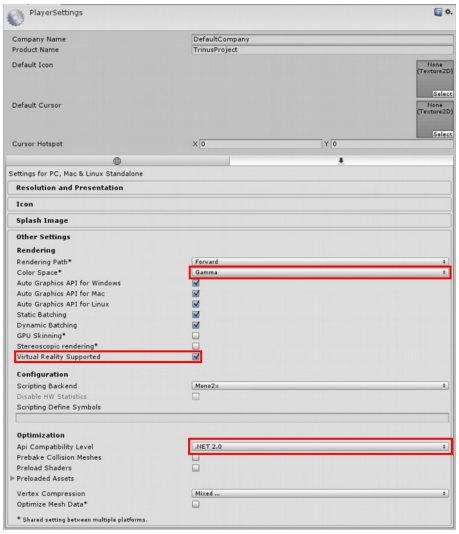
Table of Contents

Quick Guide

Trinus VR is a cost effective way to enjoy high quality VR. Sitting between PC VR and Mobile VR, it uses the PC (or console) to generate high quality visuals, and the smartphone to display the content and provide head tracking.

These are the steps to integrate the Trinus technology into your own projects:

1.- Import the Trinus package (or copy the /Assets/trinusLib folder)



2.- Set the project settings as follows

- Set platform to PC/Mac/Linux standalone (ie. not Web Player or other)
- * Enable VR Supported (Stereoscopic rendering checkbox is optional, adds extra parameters to camera). Color Space needs to be set to Gamma when using VR
- Api Compatibility Level needs to be set to .NET 2.0 in Player Settings (i.e. NOT .NET 2.0 subset)

* This is optional, see below for alternative camera setup (which doesn't have the Color Space restriction).

3.- Add the Trinus prefabs

- TrinusManager
- TrinusCamera (this should be the active main camera)
- TrinusUI
- DefaultEventSystem (only if you don't already use EventSystem)
- SampleTrinusOnlyUIManager (in /trinusLib/Prefabs/sampleUIManagers). This is a simple example to get you started, alternatives and customizations explained below.

4.- Done!

Start the project and follow the instructions to establish the Trinus link.
Note that the image might be blank while running from editor if Game window is set to locked ratio (eg. 16:9). Leave it on Free Aspect. (This doesn't affect the built result)

Found a problem or have a suggestion/request? Shoot an email at support@trinusvr.com

About Trinus VR

Trinus is a software solution that links computer (or console) and phone to allow playing games (or any 3D software) in VR.
With this approach, AAA Games (high quality PC content not available on phones) can be played with an inexpensive VR setup.
On the smartphone end, we have the Trinus VR client app. On the PC side, either the Trinus standalone server or a project that uses this Trinus Unity plugin.

The Trinus system connects the computer and the phone via local network. This is done via Wifi (PC and phone are on the same network, eg. connected to the same router),
or via USB Tethering (phone is plugged to PC and tethering enabled through the phone settings).
To ensure the link can be established the network needs to be fully working (ie. no firewalls, Wifi incompatibilities, correct USB Tethering drivers installed, etc.).

Once both ends are on the same network, starting the server and the client will initiate the handshake and connection process, which will finally result in the PC content being displayed on the phone.

The Trinus server has several options to deliver a high quality option regardless of the game played, or the head mount (Cardboard or any of its variants, like Homido, VR One or GearVR).
To improve the image quality depending on the head mount, Trinus has Lens presets that will format the image shown on the phone (ie. rounded shape with lens and chromatic correction).
The details for these and other options are described at http://Trinus VR.com/help. Further assistance can be found at the Trinus forum (http://oddsheepgames.com/?page_id=208) or contacting support@trinusvr.com

Trinus VR for Unity Plugin

As an alternative to the aforementioned server-client combo, developers can now use this Trinus Unity plugin. It replaces the PC server (no longer needed) and allows your project to connect directly to the smartphone Trinus app.

With this approach, the setup process is streamlined (user doesn't need to use and tweak the separate server application, as most options can be fine tuned during development).
Additionally, the plugin ensures best performance and use of VR options (like accurate full rotation tracking and stereoscopic view).

Trinus ensures better ROI for your projects. The cost of dedicated hardware fragments the audience and limits the market share. With Trinus support, developers can ensure their products can reach a much larger user base due to it's accessible pricing.

The Prefabs and scripts included in the trinusLib folder contain all the necessary elements to establish the connection, stream the video content and adjust the camera view (rotation) to match the user's current view in the virtual space.

Getting started with Trinus VR for Unity

A complete example using the original Survival Shooter by Unity Technologies as template can be downloaded from the Assets Store: https://www.assetstore.unity3d.com/en/#!/content/43781

The plugin has been tested with Unity 5.3.2f1 and Windows 8.1. It works on Mac too (USE_SYS_DRAW define needs to be disabled, otherwise it will crash on Mac unless certain additional libraries are installed).
The plugin will also work with consoles (Playstation/Xbox).
Trinus VR is still a product under development, further improvements will boost performance and add features.

IMPORTANT: Api Compatibility Level needs to be set to .NET 2.0 in Player Settings (i.e. NOT .NET 2.0 subset)
Make sure you are using the latest version of the Trinus client.

Trinus Objects

This section describes the Trinus objects and how to use them, providing info on most relevant parameters and methods.

TrinusManager

TrinusManager contains the TrinusProcessor script, the base script that access Trinus core library.
TrinusManager can also be preloaded on a previous scene, to do the connection before loading the actual level, make sure Preserve Game Object is checked in the TrinusProcessor script
Lens distortion correction may lower the performance (depending on phone capabilities, since that's where it is done). This lens correction setup can be disabled and added on the Unity side.
TrinusProcessor has a USE_SYS_DRAW define which will use System.Drawing for faster performance when enabled, but it may not work on all systems. Windows will be able to use it fine, but for Mac/Linux additional steps are required, see
http://answers.unity3d.com/questions/53170/using-drawing-package-like-systemdrawing.html
Better platform support is in the TODO list.

TrinusProcessor has a few additional parameters that can be set via the editor. For the most part, these won't need to be changed (hover over each parameter for details).

TrinusProcessor starts paused by default. When unpaused, it will start looking for a connection to the Trinus client app. If you are not using the TrinusUI (see below) you will have to start the connection by unpausing the TrinusProcessor script, TrinusProcessor.trinusPause(false).

Main methods

For the most part, all the methods in TrinusProcessor are accessed by the TrinusUI and don't need to be accessed from your project. The common exceptions would be resetView() and switchCamera()

- **trinusPause**: TrinusProcessor is paused by default. By unpausing it will attempt to establish connection with the client.

- **resetView**: Resets the direction of view. An angle parameter can be set to specify what should be the starting angle. This is useful to call after connected, to ensure the view is set to the right

direction. A key can be assigned in TrinusProcessor to call the reset (by default set to R).

- **endStreaming**: Finishes the connection.

- **switchCamera**: Use this method to switch between TrinusCameras (activating head tracking on the new camera). You can use Auto Switch Camera (see parameters below) to automatically switch cameras. To avoid discomfort, transitions should be smooth (eg. fade out/in, single camera moving from point A to B, etc.).

- **disableHeadTracking**: There are occasions where you'll want to disable head movement, for example when focusing on a UI menu or a game feature (zooming in to an incoming enemy, focusing on a game monitor, etc.). This method can be used for that.

- **decideResolution**: Change game resolution. This method only requires specifying the height. Width is calculated automatically depending on stereoscopic or monoscopic view (see TrinusCamera modes for details).

Parameters

- **Preserve Game Object**: Keep the TrinusProcessor (and its connection to Trinus client) alive even when loading a new Unity scene.

- **Allow Fullscreen**: Windowed mode is more flexible on the resolutions allowed (specially important when using monoscopic mode) and might give slightly better performance on some PCs.

- **Create Hotspot**: Hotspot functionality is very limited (Windows only) and generally not recommended at this point. If enabled, Trinus will attempt to create a 'trinus' hotspot for the client to connect to directly to the PC (as opposed to router connection). This is only beneficial when user has a low quality router connection and the Wifi card on the PC can deliver better results.

- **Default Yaw**: Starting angle when the view is reset.

- **Reset View Key**: Allows user to recenter the view. Might be set to none and managed separately by calling resetView()

- **Ignore View Key**: Head tracking will be disabled while this key is pressed. Can be set to none.

- **Trinus Camera**: Default VR camera to use. If not set, TrinusProcessor will look for a TrinusCamera within the scene.

- **Auto Switch Camera**: If set, whenever a TrinusCamera is enabled it will be set as the current TrinusCamera with headtracking (ie. no need to use TrinusProcessor.switchCamera()).

- **Project Id**: Identifier used by Trinus to keep track of the project stats. It is recommended to request an Id (contact info@trinusvr.com) once you start your development with Trinus. While you don't need an Id during development stage, if you publish your project without an Id, functionality may be restricted in the future.

UserSettings

TrinusProcessor.getUserSettings() gives access to the Trinus settings. This includes connection information (eg. IP and ports), lens distortion parameters and a few more options.

There's usually no need to manage these settings, as it is all taken care of by the Trinus UI.

TrinusCamera

The TrinusCamera object manages the view that will be displayed on the phone, including head tracking. It has 3 cameras that are activated depending on the current mode selected.

Optionally, you can also skip adding TrinusCamera objects to your project, and add the TrinusCamera script to your existing cameras. This feature needs improvement.

The script has mode selection parameter, defined as follows:

- **Disabled**: No cameras

- **Single**: a single camera is active, to use in monoscopic mode. This is a 2D view, with same image shown to each eye. While losing 3D depth, this mode can deliver better performance on lower end systems. The TrinusUI has an option for the user to enable/disable this mode.

- **Dual**: a left and right camera setup. This mode requires a bit more work to manage post processing effects. A syncCameras() method is available to ensure effects on one camera are also present in the other (ie. You can add all your effects to the Left Camera and call this method to duplicate them on the Right Camera).

If you use effects that need to dynamically change, you will have to update the corresponding components manually, for both cameras. You can use getMainDualCameras() to retrieve Left and Right Cameras and then access their effects/components to modify values.

- **Unity VR**: this is a single camera (SplitCamera) that enables Unity Stereoscopic mode. There are currently a few limitations (due to Unity implementation). For one, Color Space needs to be set to Gamma (and VR support enabled in Player Settings). Additionally, rolling the camera causes some image distortion.

On the plus side, simplifies complex camera operations and avoids additional management of post processing effects, required in Dual mode.

For most cases, you will want to set the TrinusCamera default mode to either Dual or Unity VR.

You can set the default TrinusCamera in TrinusProcessor. If not set, the script will look for a TrinusCamera object within the project.

You can have several TrinusCamera objects. Whenever you want to switch to a different camera, use the TrinusProcessor.switchCamera() method or use Auto Switch Camera in TrinusProcessor to automatically switch to a camera when it is enabled.

TrinusUI

The Trinus UI is a set of pages to navigate the user through the connection process and the settings (FOV, Lens parameters, etc.).

While optional, using this UI is recommended as it takes care of configuration and connection management. It also explains the user how to download the Trinus client and connect.

Sample UI managers are included in the /trinusLib/Prefabs/sampleUIManagers to show the different ways the UI can be used.

Main methods

There are a few methods that are relevant for integrating the Trinus UI with the rest of the project UI (see also SampleIntegratedUIManager).

- **openXXX**: Methods to open specific UI pages (intro, settings, …).
- **updateTrinusCursor**: Places the VR cursor at the right position, depending on the camera setup.
- **setTrinusCursor**: Activates the VR cursor (it also disables the standard cursor).
- **start/restartConnection**: Initiate the Trinus connection process.

Parameters

- **xxx Pages**: UI pages to show (pre-connection intro, connection, in-game and settings).

- **Trinus Theme**: Theme to change the look of the Trinus UI and make it match your project looks.

- **Camera** (in Canvas Component): Camera used for the UI. This is automatically set (to the TrinusUICamera). In most cases, you'll want your UI elements to use this camera too. You can also override it and choose a different camera.

Events

You only need to set those for higher customization requirements. SampleIntegratedUIManager shows how you can use these events to fully integrate the TrinusUI within your own UI system.

- **ConnectedEvent**: called when the Trinus connection has been stablished. Can be used to initiate gameplay, show your own game UI, etc.

- **DisconnectedEvent**: triggered on Trinus disconnection. Use it to save and quit, go back to main menu, etc. If not set, Trinus will attempt to reconnect again.

- **FinishedSettingsEvent**: when the Trinus settings page is closed, this event is fired. This can be used to go back to game main settings, show game UI to continue playing, etc.

- **ExitEvent**: called from the TrinusUI.quit() method (which is called when clicking exit button in TrinusUI). Use to go back to main menu, actually quit or similar. If not set, Trinus will close the game.

TrinusUICamera

The TrinusUICamera is included in the TrinusUI prefab. It overlays content set in the UI layer (make sure your UI elements are set to that layer if you want them displayed).

Localization

The UI has a localization system where elements with name starting with 'label' have the text replaced with a matching entry in /trinusLib/Resources/text_xx (xx being the language code). You

can optionally use this mechanism for your own texts, by calling Localization.setLocalization() with the parent transform containing the texts to localize, and adding your entries in the text_xx file.

Themes

The Trinus UI look can be modified using the TrinusTheme script. By attaching an instance of TrinusTheme to TrinusUI, the colors, font and sprites can be customized to match your project style.

A couple of sample themes, Dark and Clear are included in the Prefabs folder.

Sample UI Managers

Three prefabs are included in the /trinusLib/Prefabs/sampleUIManagers folder that show how to use the Trinus UI. You may build on top of these or use as reference to build your own UI system.

SampleNoUIManager

This is the most simple manager. It includes the basic parameters to initiate the Trinus connection. No UI, just straight to connect (developing a UI to explain the user how to use Trinus would be highly recommended).

SampleTrinusOnlyUIManager

This manager initiates the TrinusUI. It is a good example on how the UI can be managed. Might be used on projects where you want to start the Trinus connection when you load a game scene (ie. separate from previous game menu scenes), without any additional UI.

SampleIntegratedUIManager

This manager showcases how the TrinusUI can be integrated with your own UI. It includes UI mock ups that redirect the flow in and out of the TrinusUI, using events and calls to the openXXX methods in TrinusUI.

Final Notes

There are a few points to consider when designing a game to support VR. For example, UI/text should be large enough to be read even on lower quality settings, and positioned within safe margins to ensure visibility.
UI design is specially important for VR: should it be minimalistic and external or be part of the game world (e.g. Pip boy/Google Glass)?
Certainly avoid important information being displayed on the edges. It is a good option to add a vignette around the corners and keep it as reference for low visibility items.
Other things to consider are how to deal with external head movements (e.g. earthquakes, G force while driving, etc.). Please refer to online recommendations for VR design.
It is a good idea to have gamepad support, to allow wireless gameplay.

TODOs

Improve performance (current max recommended resolution is 1600x900 for a mid-high end CPU, with future updates to support resolutions above full HD)
Positional tracking
Add more Lens presets
Found a bug or have a suggestion? E-mail the details to support@trinusvr.com

Cool Works

If you create a cool project with Trinus support, it will be featured on Trinus VR.com (if you want).

More Info And Support

Check Trinus VR.com for more information.
You can use the dev section of the forum to start discussions and comment issues:
http://oddsheepgames.com/?forum=developers
For additional support, contact support@trinusvr.com

Enjoy!