Triple Screen Camera

Asset for rendering your game on triple monitor setups

Table of contents

Overview

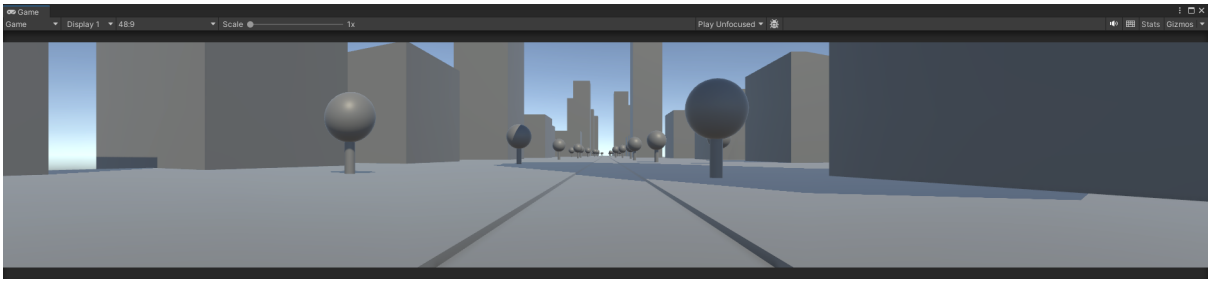
This asset allows the use of three monitors in applications such as driving or flight simulators. It adjusts the cameras and their frustum rendering in such a way as to perfectly represent the real-world display setup. This makes the view in the application more immersive and provides a better representation of speed and distance.
It works by aligning the asymmetrical camera frustum to match the display corners. These corners are calculated with the given parameters.

Setup

Place **TripleScreenCamera** prefab on your scene as a child of your player object or any other camera controller.

Editor

To test camera setup in the editor you can create a new **Game** window with 48:9 display aspect ratio (or other if different aspect ratios are used in setup). It will simulate a triple monitor setup.



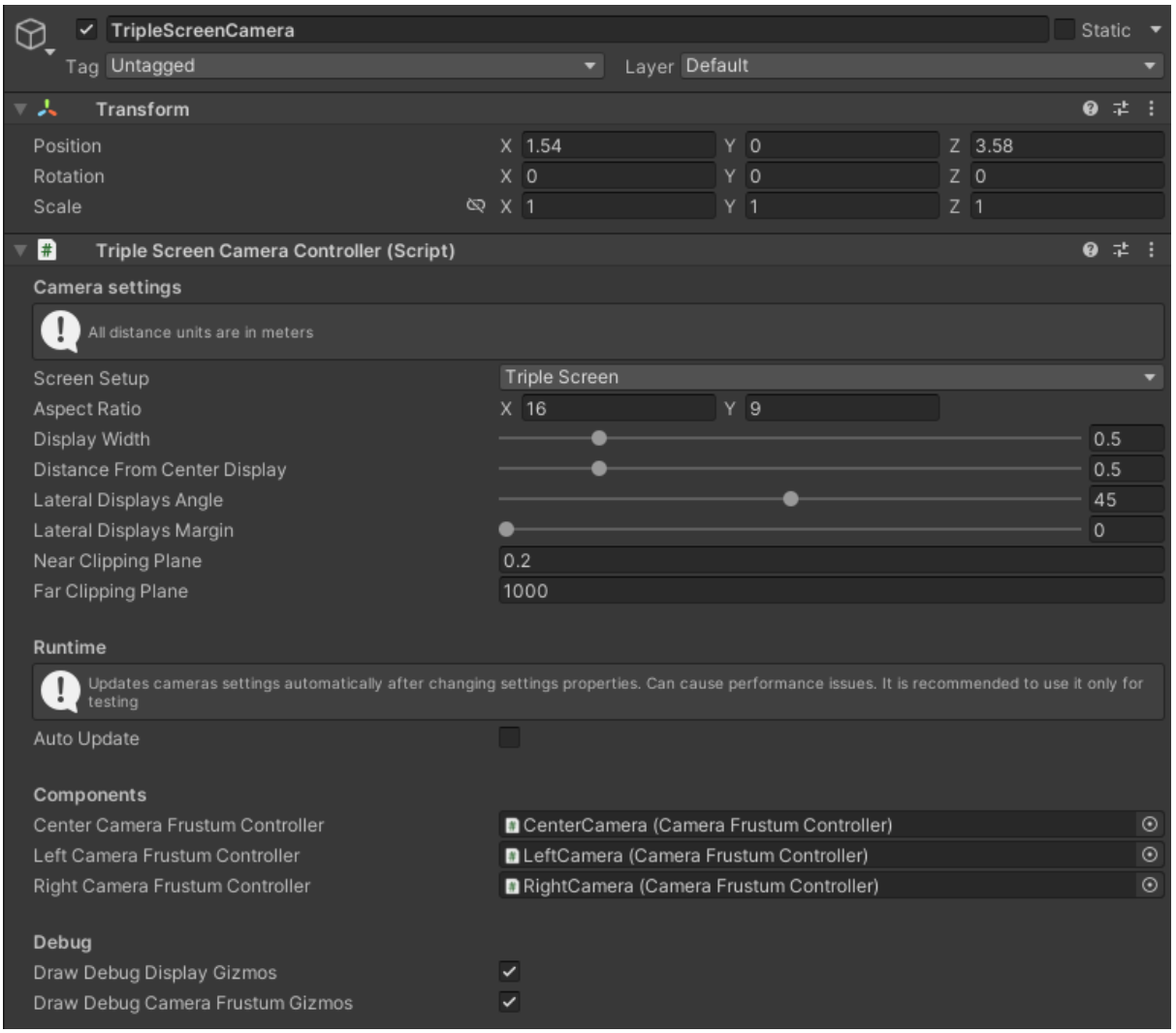
Build

Before launching the application, change the display settings using Nvidia Surround or AMD Eyefinity. You need to combine the three monitors into one display, on which the three cameras will be displayed.

Nvidia Surround
https://nvidia.custhelp.com/app/answers/detail/a_id/5335/~/gettings-started-with-nvidia-surround

AMD Eyefinity
https://www.amd.com/en/support/kb/faq/dh2-014

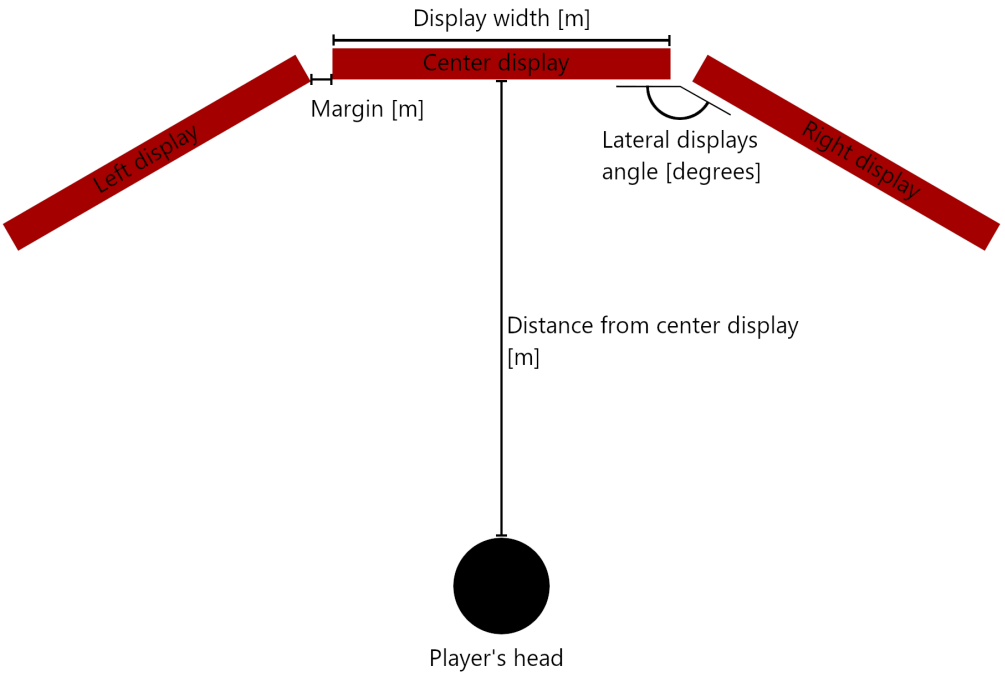
Camera Settings



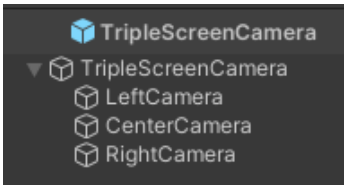
- **Screen Setup** - sets camera settings to match your screen setup. **Single screen** mode will disable lateral cameras and center camera will render the whole screen. With **triple screen** mode all three cameras are enabled and each camera will render a one-third of the screen. This option changes viewport values of the center camera.
- **Aspect Ratio** - aspect ratio of used displays, by default set to 16:9. X is horizontal value, Y is vertical.
- **Display Width** - width of each display measured in meters.
- **Distance From Center Display** - distance from center monitor to player's head measured in meters
- **Lateral Displays Angle** - only visible when using triple screen setup. Angle between center and lateral display
- **Lateral Display Margin** - only visible when using triple screen setup. Distance from center display edge to edge of lateral display measured in meters.

- **Near** and **Far Clipping Plane** - clipping planes values that will be set in cameras. Lateral cameras have clipping planes calculated to match center one - they will be matched with edges of center camera render frustum.
- **Auto Update** - triggers automatic camera settings update after changing any of above values in runtime. Can cause performance issues, it is recommended to use it only for testing
- **Components** section - references for **CameraFrustumController** components of each camera. Each camera has its own controller which independently sets the frustum to match the given display parameters.
- **Draw Debug Display Gizmos** - rectangular red gizmos which reproduce the setup of monitors based on given parameters
- **Draw Debug Camera Frustum Gizmos** - white gizmos showing the rendering frustum of all the cameras without having to select them in the hierarchy

The image below shows the values that should be set:



Other camera options such as culling mask, post processing or target display can be set directly on the Camera components. These are attached as children to the TripleScreenCamera prefab.



Code API

TripleScreenCameraController

The main component which calculates the position of the displays and sets the camera options on the basis of the given parameters.

Properties

Each option described in the **Camera Settings** section of this document is available as a public property, making it possible to change its value at runtime. These include:

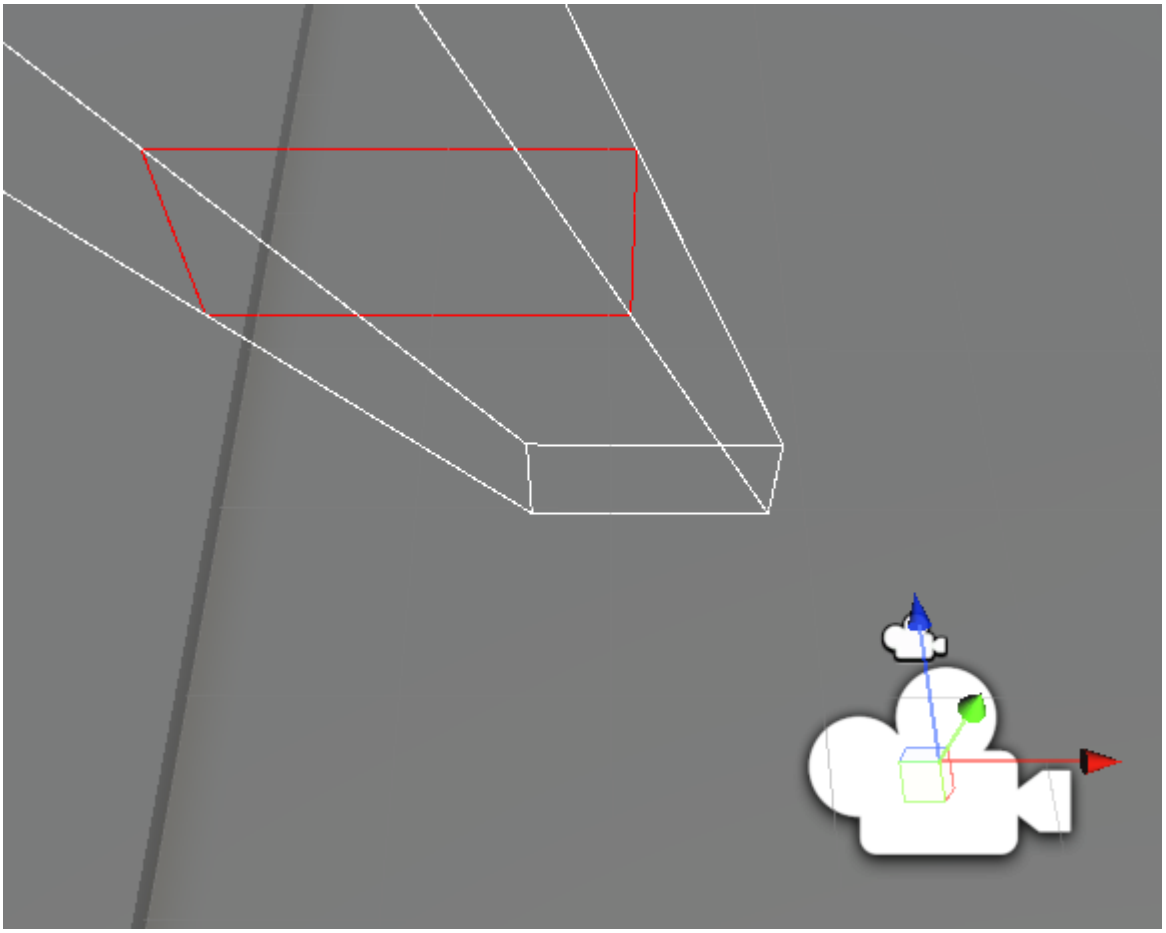
- ScreenSetup
- AspectRatio
- DisplayWidth
- DistanceFromCenterDisplay
- LateralDisplaysAngle
- LateralDisplaysMargin
- NearClippingPlane
- FarClippingPlane
- AutoUpdate

Methods

- **public void UpdateAfterSettingsChange()** - triggers camera settings update if **Auto Update** option is disabled

CameraFrustumController

Can be used independently if you write custom code setting asymmetric camera frustum. Must be added to object where the **Camera** component exists.



Methods

- **public void SetFrustum(Vector3[] corners, float nearClippingPlane, float farClippingPlane)** - sets the frustum of the attached camera to match the given corners of the display. Takes an array parameter with the four positions of the corners in world space. First element is bottom left, last is top left, goes counter clockwise.
- **public void SetFrustum(Vector3 bottomLeftCornerPosition, Vector3 bottomRightCornerPosition, Vector3 topLeftCornerPosition, float nearClippingPlane, float farClippingPlane)** - it works in the same way as the method above except that it accepts parameters in the form of separate values for the position of corner displays.