CyberCANOE Stereo Rendering Library

© August 30, 2025 - Jason Leigh
Laboratory for Advanced Visualization & Applications
University of Hawai'i at Mānoa

This code base provides off-axis stereoscopic 3D rendering for Unity. This is the basis for proper 3D rendering in systems like display walls, CAVEs, CAVE2s, CyberCANOEs, where the stereo image displayed is correctly rendered according to a user's head position and orientation. This also ensures that objects in the virtual world appear to be correctly sized when seen from the real world- in other words, a 1 meter cube looks like it's really 1 meter in size.

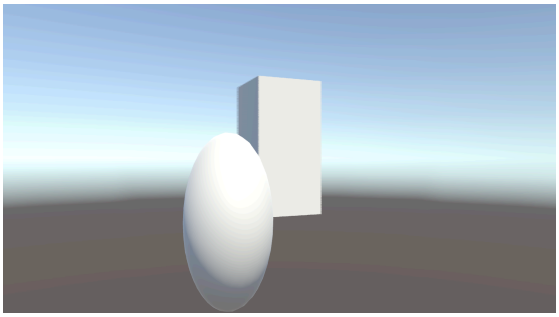
Using the Code

To use the code, start with one of the project templates (preferably using Unity 2022 and up:

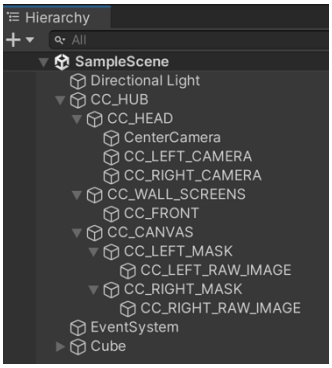
- CC2024-BRP (for Built-in Rendering Pipeline)
- CC2024-URP (for the Universal Rendering Pipeline)
- CC2024-HDRP (for the High Definition Rendering Pipeline)

Unity claims to be gradually deprecating the old built-in render pipeline. So, if you are using / purchasing assets on the Unity asset store you may want to opt for either URP or HDRP, especially those that have shaders.

Running the project in the Unity editor, you should see a simple scene with a cube and a sphere that spins around the cube. The image might appear squashed like below. This is normal as your Unity Scene view is not in the same aspect ratio as the display wall that will eventually portray the 3D image. When you finally build and run your standalone app on the wall, it will fill the full aspect of the wall- provided you have the physical dimensions of the wall correctly entered into the CCUnityConfig.xml file, (more on this later).



The part of the scene hierarchy that is most relevant is the one under CC_HUB as shown here →



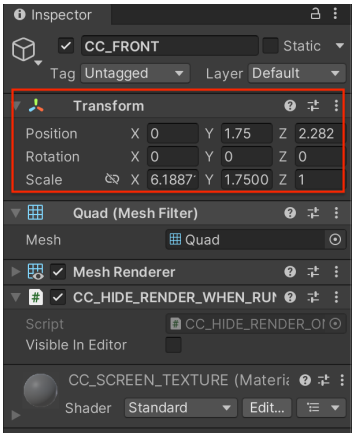
Think of CC_HUB as the spaceship that moves through the virtual space and you are standing inside that spaceship looking out the window in front of you. To move through space you simply adjust the position and orientation of the CC_HUB gameobject.

CC_HEAD is the gameobject representing your head. Your head resides within the CC_HUB. If you are using some kind of 6 Degree of Freedom tracker (like Vive or Kinect), you can feed the tracker's head position and orientation to CC_HEAD. If you don't intend to do any kind of tracking in general, just leave CC_HEAD alone. It is currently set to place your head at the center of the CC_HUB looking along the positive Z axis. It is also defaulted to a Y value (height) of 1.75m to emulate the height of an average 5'10" male.

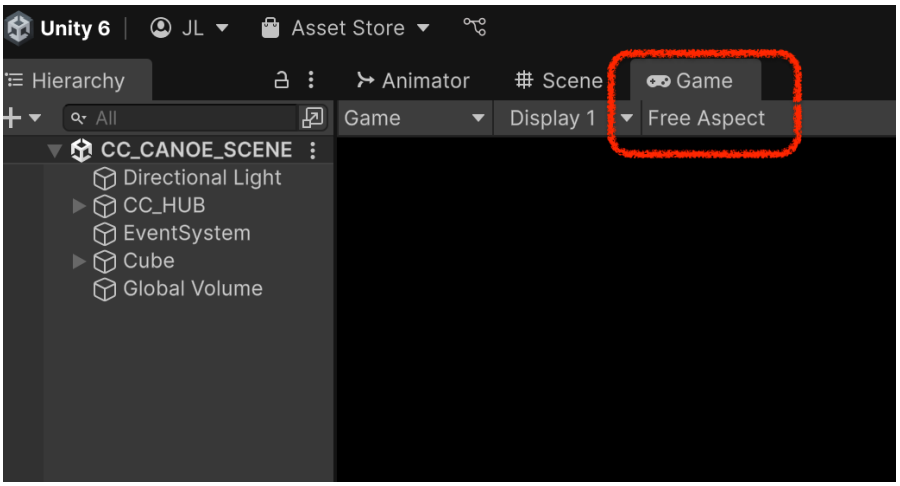
If you wish to change your scene's background or near and far clipping planes, do it to all the cameras in CC_HEAD (i.e. CenterCamera, CC_LEFT_CAMERA and CC_RIGHT_CAMERA).

CC_WALL_SCREENS - contains a CC_FRONT. CC_WALL_SCREENS essentially holds the window(s) on your spaceship. CC_FRONT is the front window. In technical terms, these determine the position of the 3D projection screen and therefore it should be set to the same position as the *physical* display wall. By default it is set to 6.18m x 1.75m to emulate LAVA's 3D display wall. Also by default this display is placed at position (0,1.75,2.282). Only 1 screen is available in the implementation of this library (CC_FRONT). I may extend this in the future to support multiple screens in the future (e.g. for a CAVE.

When using the Unity *editor*, you can adjust the size and position of CC_FRONT to match your physical projection screen by adjusting the position, rotation and scale values. As mentioned earlier, it is set to LAVA's 3D display wall dimensions by default. Position should be set wherever the center of your wall is relative to what you declare as the origin in your physical space. Units are in meters. Scale refers to the width and height of your wall in meters.

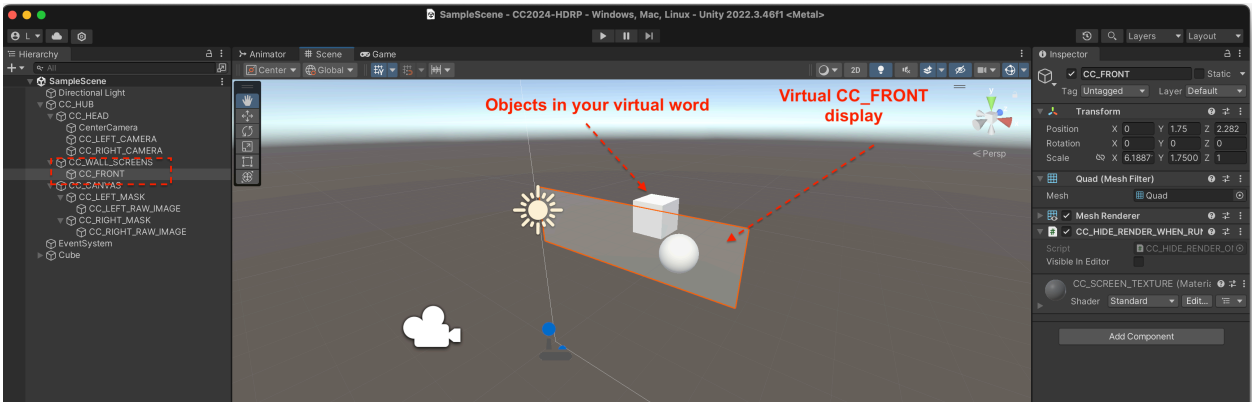


Also in Unity's Game window, make sure to also set it to Free Aspect.

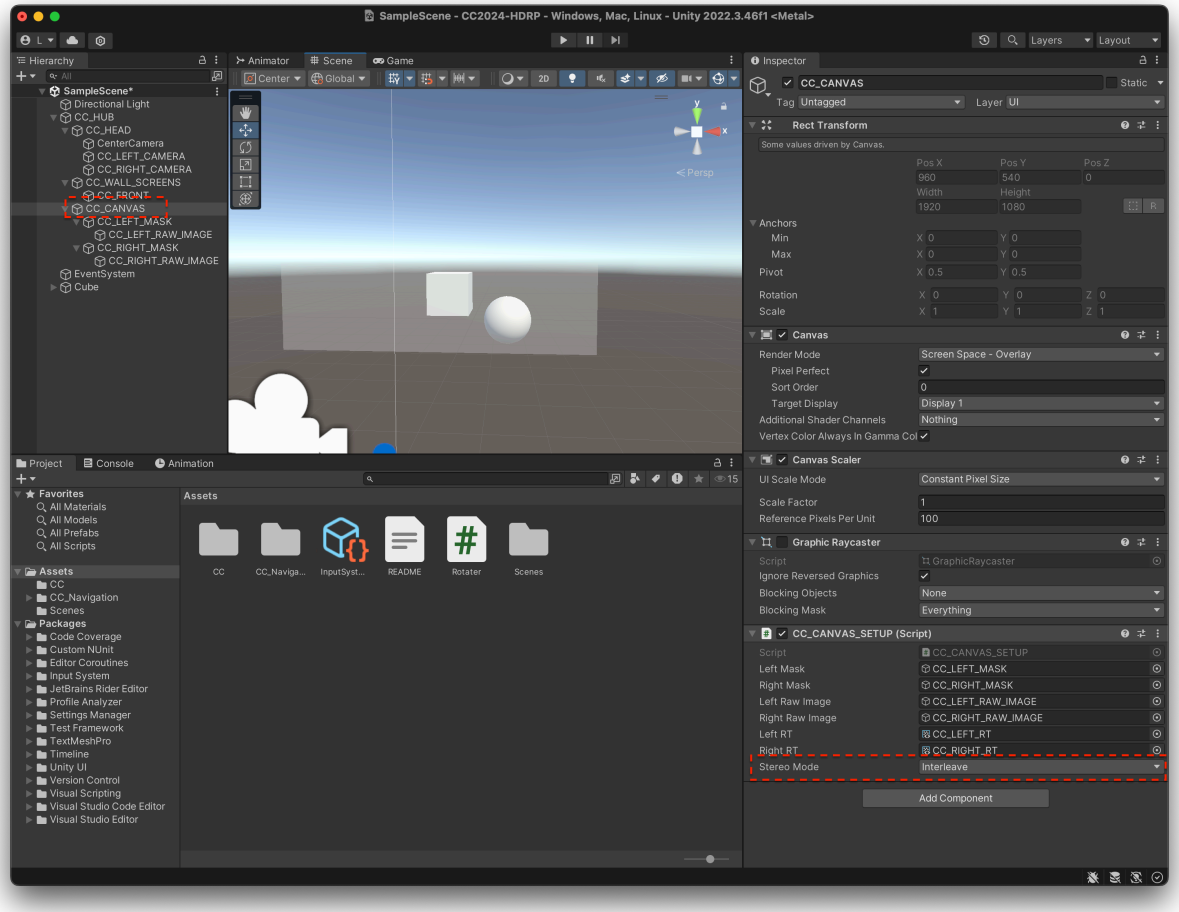


Set this information also in the XML file (take a look at CCUnityConfig.xml in CCUnityConfig folder, for an example) so that the proper stereo configuration is loaded during runtime. This includes entering the resolution of your display wall. **The CCUnityConfig folder should be placed at the same directory level as your application's .exe file so it has access to it.**

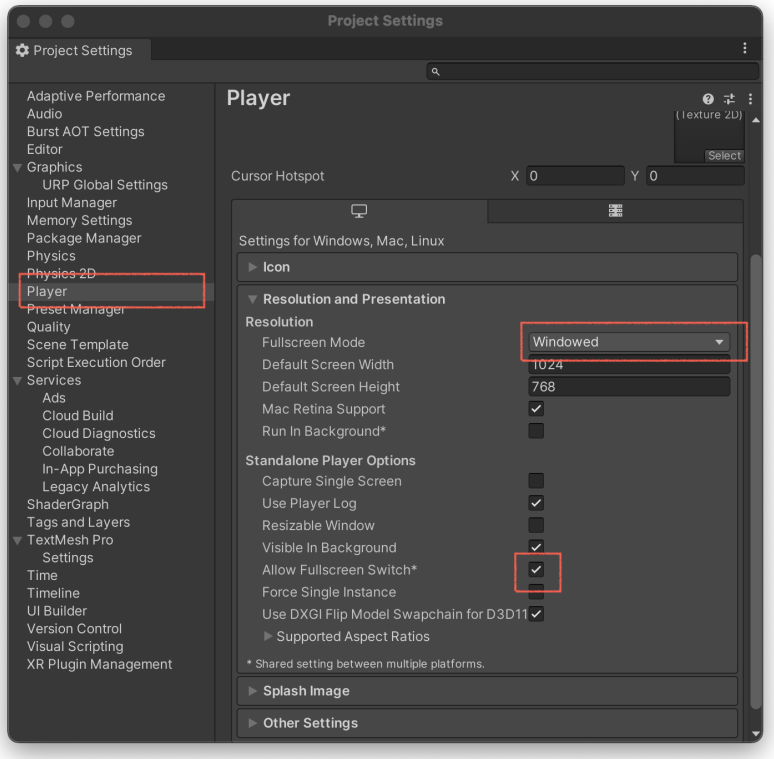
Also notice, the CC_FRONT screen appears ghosted. This is just to help you see where your projection screen is relative to the objects in your virtual world. When you actually run your built app, that ghosted window is not shown.



Lastly, to set the rendering to a different stereo mode (Interleaved, Side-by-Side, Top-Bottom) go to the CC_CANVAS object and change it in CC_CANVAS_SETUP's Stereo Mode drop down menu.



NOTE:
When building your Unity application for LAVA's 3D wall (Pele), make sure the following are set in Edit→Project Settings.



Also in CCUnityConfig.xml, make sure the following is set: <fullscreen>0</fullscreen>

Enhancements

If you look at the CC_HUB game object there are also two attached scripts (DPadFlyer and CCaux_Waypointer, SpaceMouseFlyAround).

The DPadFlyer lets you use a PC Xbox-like Bluetooth game controller to move around the scene. This controller also works for Macs.
The SpaceMouseFlyAround lets you use a Space Mouse to move around the scene (https://3dconnexion.com/us/product/spacemouse-pro-wireless).
The CCaux_Waypointer lets you set waypoints in the scene and either move back and forth quickly between way points or travel through all the waypoints like a pre-scripted movie.

DPad Control

The game controller mappings for DPadFlyer are:

- **Left joystick** – Move forwards/backwards and strafe.
- **Right joystick** – Pitch and Yaw.
- **Left/Right triggers** – Roll.
- **Left/Right shoulder buttons** – Move Up and Down.
- **A button on game controller** – hold this down to go back to the origin.

Space Mouse Control

Warning: if you intend to use the Space Mouse, develop your CC_CANOE app only in Windows. If you try to do it on a Mac and then try to run the app on the display wall in Windows, it will not work. I don't know how to fix this yet. If you don't care about the Space Mouse then by all means use a Mac to develop your app.

Space Mouse is supported with the Unity SpaceNavigator Asset – which is already included in the CyberCANOE library. But in case you'd like to look at the Space Navigator separately, it can be found here: https://assetstore.unity.com/packages/tools/spacenavigator-driver-9774.

Pictured here is the Space Mouse Pro Wireless.

In the CyberCANOE library, the center nob on the Space Mouse is used to control the CC_HUB's position and orientation in full 6 degrees of freedom.

When using the Space Mouse with the Unity editor, I recommend first going to the Window menu and selecting SpaceNavigator, then when the control panel appears, *check* the Horizon box, and *uncheck* the Runtime Editor Navigation box. Checking the Horizon box will prevent your Space Mouse from causing your Unity Scene view's camera to roll (very annoying). Unchecking Runtime Editor Navigation ensures the Space Mouse does not continue to change the Scene window while you are running your app in the Game window.

Also, I recommend the following button bindings for your Space Mouse:

- Space Mouse's 1 button to the Spacebar – to Play/Stop waypoint animation
- Space Mouse's 2 button to the P key – to go to Previous waypoint
- Space Mouse's 3 button to the N key – to go to Next waypoint
- Space Mouse's 4 button to the A key – to return to starting position and orientation

Lastly, I recommend that when you build your application, you name it Unity so that the above bindings will work for all your CyberCANOE apps without having to create new Space Mouse keybindings for every Unity app. To name your app "Unity", go to Edit→Project Settings, click on Player and then look for Product Name. Enter "Unity" as the product name. Now when you attempt to build your app, it will create an executable file called Unity.exe.

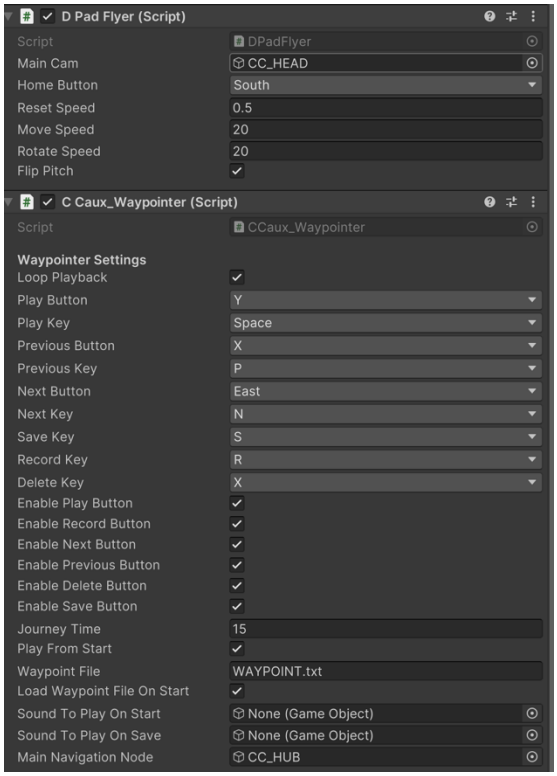
CCaux_Waypointer

The user-interface controls for the CCaux_Waypointer are:

- **Y button on game controller** (or SPACEBAR) – start/stop waypoint movie.
- **X and B buttons** on game controller – go to previous and next waypoint.

- **R key on keyboard** – record current waypoint
- **X key** – delete current waypoint
- **S key** – save all the waypoints into a file.

- When your program loads, it will automatically load this same file.
- Waypoint file name is set in the CCaux_Waypointer control panel (pictured above).



The waypoint script essentially records the position and orientation of whatever you designate as the main navigation node (in this case the CC_HUB is the navigation gameobject). When the waypoint script flies from waypoint to waypoint it will modify the position and orientation of the CC_HUB gameobject. **The recorded wavepoints can be stored to a file like WAVEPOINT.txt. To use the file, you need to place it at the same directory level as your standalone app.**

If you wish to have the waypoint playback run in conjunction with a musical soundtrack, add an audio sample gameobject to your scene and drag it into the "Sound to Play On Start" property in CCaux_Waypointer. Then set the Journey Time to the length of audio sample in seconds. The waypoint script will ensure all the waypoints are traversed within that duration.

Note: the waypoint script will also loop the waypoint traversal if desired. In that case, the music will also be looped. Anytime during playback you can stop the movie and use the game controller to fly around the scene.

How the Stereo Rendering Works

The previous generation of the library used Shaders to merge together the left and right stereo images into a single interleaved, top-bottom, side-by-side image. This worked fine until Unity required the use of Render Handlers in 2022- which essentially killed the performance of anything trying to work with multiple cameras and render textures. After two years of waiting for a fix, I finally gave up and came up with this alternative technique.

In a nutshell, the new technique renders the left and right camera images on 2 separate render textures, then it displays the render textures as Raw Image objects on a Canvas, allowing them to be placed either side-by-side, or top-bottom, or overlapping each other (for interleaved 3D). To achieve interleaved 3D, masks are created with interleaved transparent and opaque lines that filter out odd and even image lines from the raw image objects.

One of the disadvantages of this technique is that it could be slightly slower in performance. So far on testing it wasn't slow enough to make the technique infeasible.

The advantage of this technique, however, is that it is easy to port between all 3 of Unity's render pipelines (BRP, URP and HDRP). Another advantage is that it will likely be easy to support multiple display walls.

Detailed Notes

Explanation of Scene Hierarchy:

CC_HUB - this is the root node for the system. It has settings for you to adjust navigation through your virtual scene using a DPad controller. The CC_Navigation library is what does the actual work. Look in the CC_Navigation folder for details.

CC_HEAD - represents the position of your "head" in the scene. It is the point where the viewer centered perspective is rendered. So you need to input the Position X, Y, Z of where you expect a viewer to be normally standing. By default it is set to (0, 1.76, -2) which assumes an average male height (1.76m - 5ft 9in) standing 2 meters (~6ft) back from the origin. If you have some kind of tracker you can pass the position and orientation info of the tracker into the position and orientation of the CC_HEAD game object to perform correct viewer centered perspective rendering while the person is moving. Also this is where you can set your stereo eye separation.

The CC_HEAD also contains a center (non-stereo) camera, a left eye camera (CC_LEFT_CAMERA), and a right eye camera (CC_RIGHT_CAMERA). The left and right eye cameras use a script CC_CAMERAOFFESET that does the transformations to create an asymetric viewing frustum that is necessary to create propery off-axis projection. The center camera is primarily used during software development but is automatically disabled when your application is run as a standalone app.

CC_WALL_SCREENS - hold the display wall(s). In this implementation there is only 1 wall, the CC_FRONT. Some day I may extend this to multiple walls (e.g. for a CAVE). Note, the CC_FRONT screen has values inputted already for position and scale. This assumes the wall is about 2.282 meters from the origin and 1.75m high (to the center of the wall).
The screen itself is set to 6.18m (~20ft) by 1.75m (~5ft 9in) to match the size of LAVA's display wall. You must change these to match the physical dimensions of your display.

CC_CANVAS - In the Unity UI Canvas we have 2 raw image objects positioned to show the left and right stereo images. In side-by-side stereo, the two raw image objects are essentially side by side. For interlaved stereo they are placed on top of each other and a mask is used to filter out successive lines of the images to get an interleaving of left and right eye scan lines. For top-bottom, the left image is placed on the top half of the screen, and the right image is placed on the bottom half of the screen. Whichever mode you choose is dependent on the modes supported by your 3D display.

Each of the cameras (CC_LEFT_CAMERA and CC_RIGHT_CAMERA) render their image to render textures (look in the CC folder for CC_LEFT_RT and CC_RIGHT_RT. The CC_SETUP_RENDER_TEXTURES script in the CC_HUB game object is what assigns the render textures to the respective left and right raw image on the canvas. The CC_CANVAS_SETUP script in the CC_CANVAS is what deals with configuring the masks and positioning/sizing left/right raw images for the 3 stereo modes.

Quick Reference to the Scripts: [contained in CC folder]

CC_CAMERA_SETUP: [found in CC_HEAD] - sets up the eye separation for the cameras as well as lets you invert stereo if necessary.

CC_CAMERAOFFSET: [found in CC_LEFT_CAMERA and CC_RIGHT_CAMERA] - creates asymetric view frustum for off-axis viewer centered
perspective rendering.

CC_CANVAS_SETUP: [found in CC_CANVAS] - used to set up the arrangements of left and right images on the canvas for display.

CC_INTERLEAVE_MASK: [found in CC_LEFT_MASK and CC_RIGHT_MASK] - creates the interleave masks for interleaved stereo rendering mode.

CC_CONFIG: [found in the Project window and not attached to any object] - This reads the CCUnityConfig XML file.
It is used by other scripts: CC_CONFIG_WALLS, CC_CANVAS_SETUP, CC_CAMERA_SETUP.

CC_CONFIG_WALLS: [found in CC_WALL_SCREENS] - This loads wall configuration from the CCUnityConfig XML file during runtime.

CC_SETUP_RENDER_TEXTURES: [found in CC_HUB] - Assigns the render textures to the raw image objects in the canvas.

CC_HIDE_CAMERA_ON_APP_RUN: [found in Center Camera] - Hides the CenterCamera when an app is run as a built app. This is not something terribly important to worry about.

CC_HIDE_RENDER_ON_APP_RUN: [found in CC_FRONT] - The wall has a soft haze ordinarily to make it easier for the developer to see the bounds of the wall. When the app is actually run it is hidden by default. Should you wish to keep it visible when the app is running in the editor, you can override it by setting Visible In Editor to true.

CC_Navigation folder: contains scripts used to let you move around your scene with a DPAD controller or a Space Mouse. It also lets you store waypoints and play them back in a loop-useful for making flythrough demos. Lastly it will let you play an audio file at the start of the waypoint animation. This is useful for adding a voice narration to your waypoint tour.
The length of the animation should be set to the same length as your audio file.
You can also specify an audio sample to be played at the end of the waypoint animation.

SpaceMouseFlyAround script: [found in CC_NAVIGATION] - It contains the code for using the Space Mouse to fly around the scene. This script should be added to CC_HUB.