4-2017

Capturing Omni-Stereo Panoramas Within Unreal Engine 4

Evan Preslar

Capturing Omni-Stereo Panoramas Within Unreal Engine 4

by

Evan Preslar

Undergraduate honors thesis under the direction of

Dr. Robert Kooima

Department of Computer Science

Submitted to the LSU Honors College in partial fulfillment of the Upper Division Honors Program

Louisiana State University
Agricultural and Mechanical College
Baton Rouge, Louisiana

April 2017

Abstract

Virtual reality systems have previously been used in combination with stereoscopic high resolution photography in order to create photo-realistic immersive environments[1]. Although the mechanism used to capture these images, the CAVEcam system, is a physical implementation, the principles of its design can be adapted to function within a suitable virtual environment.

By emulating the behavior of the CAVEcam system[1] within Unreal Engine 4, it becomes possible to render a set of high-dynamic range images from any point within a virtual environment, producing a full stereo surround vision view around the center of the apparatus. This implementation retains many of the features which make the CAVEcam system highly adaptable and desirable for ongoing SENSEI (Sensor Environment Imaging) Instrument research[2], while improving on some of its functionality as a result of being a virtual construct.

The quality, number and arrangement of the images produced by this apparatus are fully configurable, allowing for samples to be captured with arbitrarily high detail. Additionally, the entire apparatus can be automated using Unreal Engine 4 to move within the scene, making it possible to produce a complete set of images at regular intervals along its path. By adjusting the rate at which these images are captured, omni-directional stereoscopic video can effectively be captured at any frame rate.

1. Introduction

In order to capture an image within a virtual setting, game engines and other applications make use of scene capture, a virtual analog to photography. In most cases, it is used at low resolutions to efficiently render reflections and projections, although it is also possible to use at high resolutions in place of a scene's primary viewport. The act of capturing a scene necessitates that a scene be rendered from the perspective of the scene capture object. In most situations, this involves the full range of calculations associated with a primary viewport, including shadows, lighting, and other effects in addition to the scene geometry. Unreal Engine 4 natively includes two scene capture objects which accomplish these tasks, allowing for scenes to be rendered either from a planar (SceneCapture2D) or a spherical (SceneCaptureCube) projection[4].

Scene Capture in Unreal Engine 4 involves three main parts: an actor, a capture component, and a render target. Actors are the highest-level objects that are positioned within the scene. Scene capture actors typically do little more than delegate tasks to the component while serving as an anchor in world space. Components are attached to actors and perform the core operations related to rendering the scene. The rendered scene data is directed from these components to a render target object which stores it. Render targets can be used to produce HDR-radiance image files natively in Unreal Engine 4. For the included scene capture implementations, each scene capture component renders its perspective to exactly one render target, meaning that if more than a single image is to be captured per frame, a second capture component must be used or lower level manipulation must take place.

The two provided scene capture actors each possess their own associated component and render target types which cannot be interchanged in the editor. Despite this, both types ultimately make use of two-dimensional scene capture, with SceneCaptureCube employing six distinct objects to

render each side of a cube. However, rather than have each component render to its own render target, they instead individually modify a portion of the parent SceneCaptureCube's render target, producing a single equirectangular projection rather than six distinct perspectives[4], preserving the convention that component types should only render to a single render target. This attribute makes the SceneCaptureCube unsuitable for more advanced scene capture which demands many different perspectives per frame, even though CAVEcam footage is often stitched into the same equirectangular form.

2. System Design

In order to faithfully replicate the behavior of the CAVEcam within the Unreal Engine, a third distinct actor type must be created which does not inherit from either SceneCapture2D or SceneCaptureCube. This type, SceneCaptureCAVE, was created to resemble the CAVEcam in both structure and function, while drawing on influences from the other Unreal Engine 4 scene capture types. All source files related to the current implementation of SceneCaptureCAVE can be found on https://github.com/evanpreslar/UnrealSceneCaptureExportPlugin, where they are publicly available for modification and improvement.

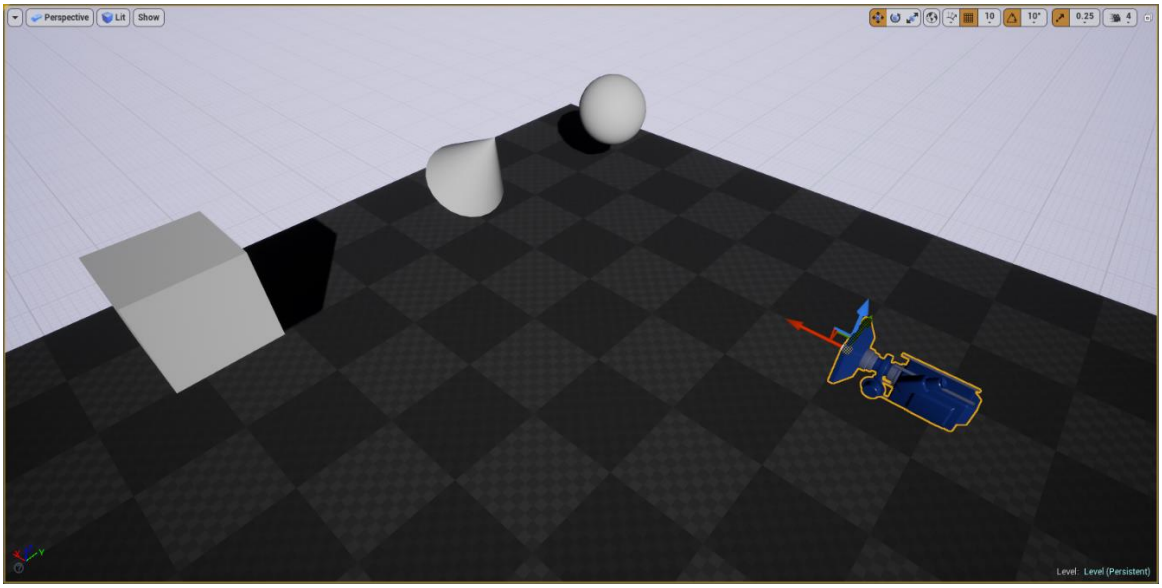

Figure 1. SceneCaptureCAVE (highlighted, right) viewed from within the Unreal Engine 4 editor. The use of a camera mesh as visualization is equivalent to that of other scene capture actors.

2.1. Structure

The SceneCaptureCAVE object is structured similarly to the existing scene capture implementations, with a single parent actor containing a single SceneCapture2D component. It does not, however, require a render target reference, as it produces more than one image per frame. Although it is possible to predict the number of images that will be produced by the SceneCaptureCAVE each frame at compile time, creating the required number of render targets beforehand suffers several issues.

Firstly, the contents of a render target are not guaranteed to update by the end of a frame, as scene capture objects are intended to be used in real time within Unreal Engine 4. This uncertainty can result in skipped or duplicated images, due to the render target's contents being overwritten before or after it is ordered to export them to disk. In order to assure that every image is reliably exported for each frame, a new render target is created for each image that is to be captured. Disposal of these images is left to the engine's garbage collector, which will remove any render targets that have successfully exported their contents (and thus have no active references).

Secondly, by not holding onto a reference to every render target, resources may actually be conserved in cases where image size and count are very large. In the above case, if a render target exports its contents, it might be forced to wait for new data. During this period, any resources it holds are no longer needed, but cannot be freed they are still referenced. By severing all connections to expended render targets, resources are freed as they complete the export process, making requirements much less severe at extreme parameter values.

Interestingly, all of the rendering done by SceneCaptureCAVE is accomplished by a single CaptureComponent2D. Unlike the CAVEcam, which requires two precisely aligned cameras, SceneCaptureCAVE is able to reposition a single capture component between these two locations every iteration. This does not interfere with rendering as all relevant data, including the render target that is to be written to, is conveyed by calling the `CaptureScene()` method, which makes it unnecessary to create new capture components along with each new render target.

2.2. Behavior

The actor component of SceneCaptureCAVE possesses several adjustable parameters (Figure 2) which modify the quality and number of images it will create. Each frame, SceneCaptureCAVE makes one call to the `CaptureAndExport2DCaptureSet()` function (Figure 3), to which these parameters are passed. Many of these fields directly map to the functionality of the CAVEcam, such as NumLongitudinalSections and NumLatitudinalSections, which together determine the number of images that will be captured.

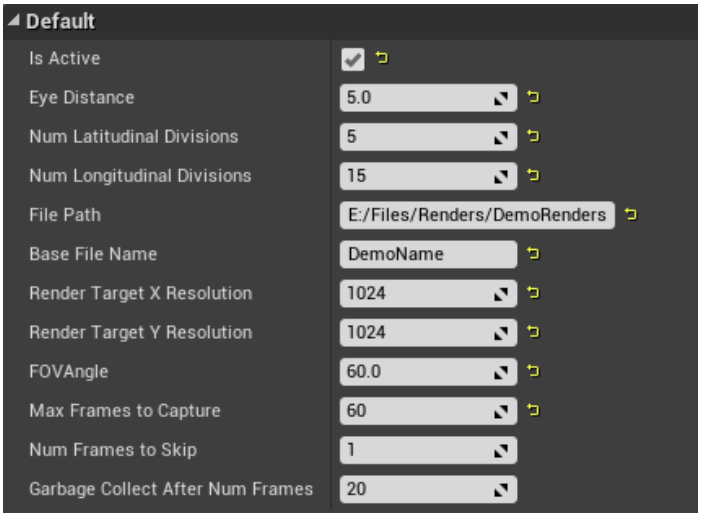

Figure 2. All SceneCaptureCAVE parameters as viewed in the Unreal Engine 4 editor

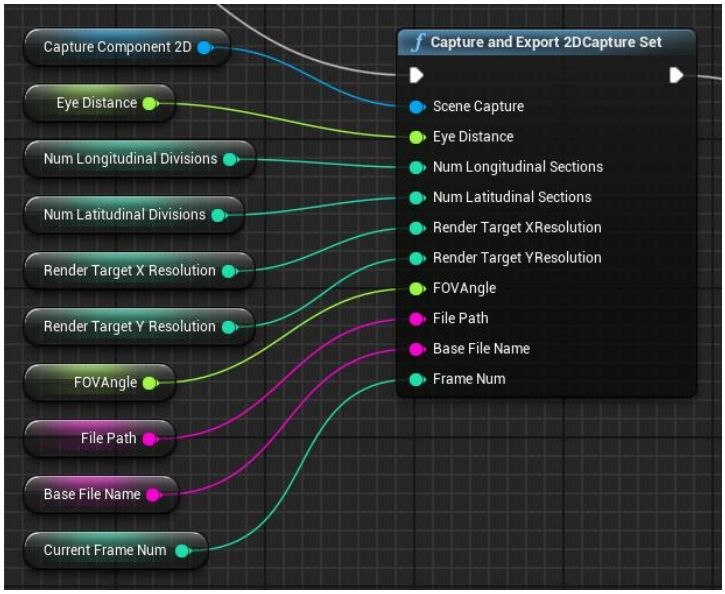

Figure 3. Parameter passing from the SceneCaptureCAVE actor Blueprint to the plugin function `CaptureAndExport2DCaptureSet()`

Using these values, the function calculates the angles that will separate each image horizontally and vertically, respectively. It then iterates twice across these values, rotating the capture component first left to right 360°, then bottom to top 180° for each iteration of the first, simulating the process of rotating the CAVEcam by hand. For example, if 4 horizontal and 3 vertical segments are requested, 12 images will be produced, four sets of three, with one call made to `CaptureScene()` for each. Each set will contain an image oriented directly down, forward and up. The total number may be doubled if the EyeDistance parameter, which defines the interocular distance, is greater than 0.0. This distance, measured in centimeters, determines the offset between the two captured images and represents the distance between the viewer's eyes. If there is no offset, then each eye effectively has the same perspective and only the right eye will be rendered.

SceneCaptureCAVE is also unlike other scene capture objects in that it automatically exports all captured images to disk. Due to the massive number of render targets which are created during execution, it is not feasible to retain them for use in the editor. Instead, they exist only as long as it takes to export their data to an image file. The destination and prefix of these exported images is defined by the FileName and FilePath variables. Files are named according to the following convention:

`[FileName]_[FrameNumber]_X[LongitudeIndex]_Y[LatitudeIndex]_[Eye].hdr`

Where the FrameNumber, LongitudeIndex, LatitudeIndex, and Eye fields are determined iteratively during execution following the procedure outlined above.

Figure 4. These 75 images represent a single frame of capture data from the right eye channel.

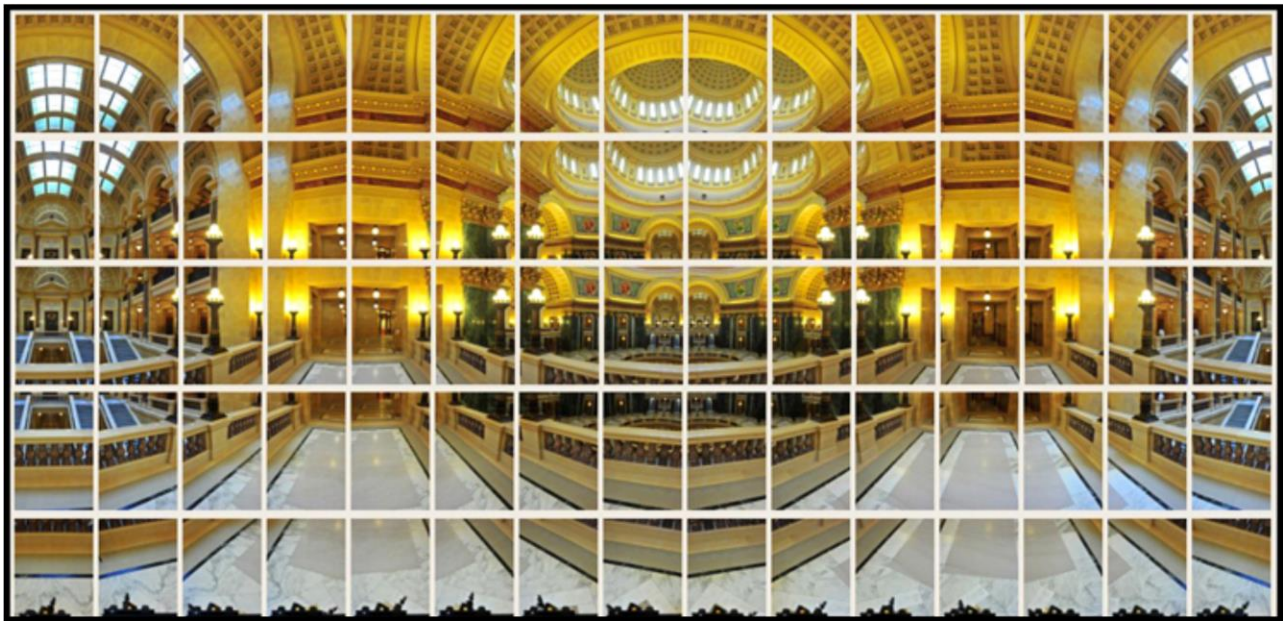

Figure 5. Sample output from the CAVEcam: 75 images taken using one of its two cameras[1]

The FOVAngle parameter is analogous to a camera's focal length, and is interpreted by the engine in horizontal degrees. Unreal Engine 4 uses this value in combination with the aspect ratio of the render target's resolution to determine the precise area that will be rendered during scene capture. As the number of images to be captured increases, it is desirable to lower the FOVAngle in order to reduce overlap between segments. Figure 4 shows a 15x5 array of images captured using the SceneCaptureCAVE with FOVAngle set to 60 and a resolution of 500x1000 per image. An equivalent output set produced by the CAVEcam is displayed in Figure 5 for comparison.

Any seams and inconsistencies in color that may be present in footage from CAVEcam or SceneCaptureCAVE is solved during stitching, which blends and the individual segments together, producing a single equirectangular projection, as seen in Figure 6.

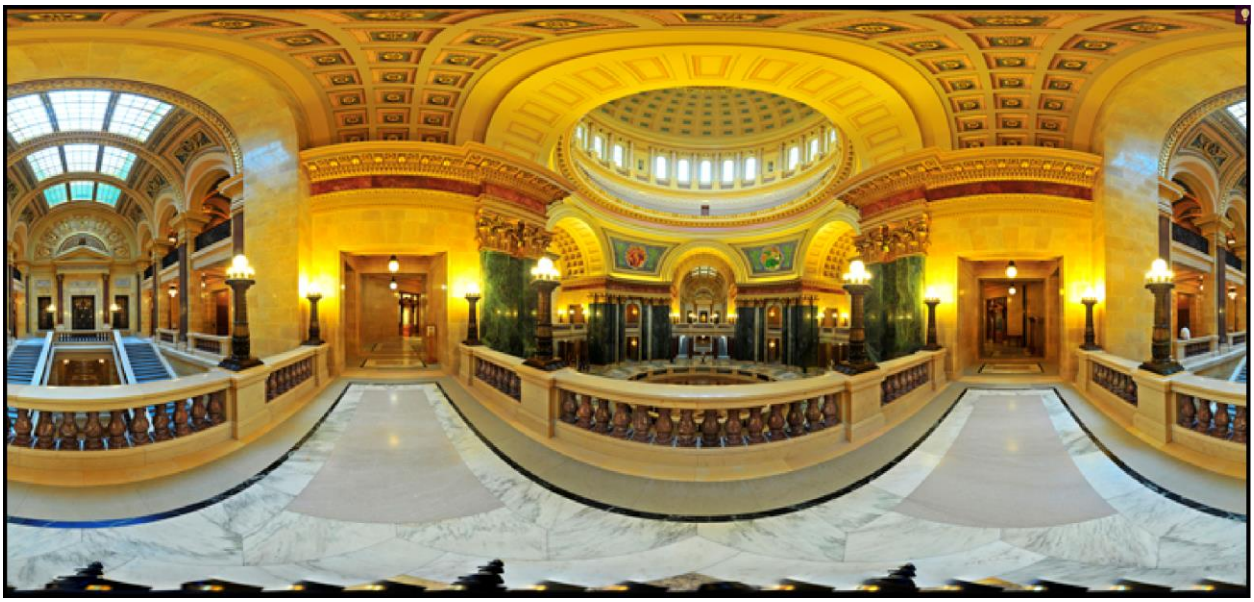

Figure 6. Resultant equirectangular projection obtained after processing Figure 5 with stitching software[1]

As with every actor in Unreal Engine 4, it is possible to automate many parameters of SceneCaptureCAVE using Level Sequences. Aside from parameter values themselves, the world position of the SceneCaptureCAVE actor can be changed over time in order to simulate a moving camera. Unreal Engine 4 by default handles all physics and automation independently of framerate, which can cause problems for scene capture as it may take a second or more to render a single high resolution image. If a scene involves any physics calculations or automation (affecting SceneCaptureCAVE or otherwise), it is a requirement that its framerate be locked if scene capture (not limited to SceneCaptureCAVE) is expected to reliably capture every frame of the simulation.

If automation and simulation are integral aspects to a scene, it may also be desirable to disable scene capture until a specific moment in a simulation, as it is often impossible to skip to a particular point in such a scene. Rather than simply wait for the signal to begin capture, SceneCaptureCAVE includes parameters allowing for precise control over when scene capture begins and terminates.

These values count the number of frames, avoiding unnecessary rendering and ensuring that SceneCaptureCAVE is only active for a specific amount of time.

2.4. Engine Integration

Unreal Engine 4 offers three methods of adding custom behavior to a project. The first is to add files directly to a project's source folder, which makes it accessible only to that project. This method is the most simplistic, but the least adaptable, as functionality becomes dependent on that specific project. The second method involves directly modifying the engine's source code, which affects every project using that engine version. This, however, forces the manual compilation of the modified engine code, making it an even less useful method for smaller projects.

The third and final option is to build an engine plugin which is compiled separately from any project and engine source files, making it the most portable option by far. Unreal Engine 4 plugins can be located in one of two locations which alter the scope of the plugin. If placed in a project's plugins folder, it effectively becomes a part of that project's source code; however, if added to the engine's plugins directory, the plugin behaves as a part of the engine code and becomes available to all projects which utilize that engine build. Regardless of its scope, plugins must be manually enabled in projects, after which all of their functions become available.

The core functionality of SceneCaptureCAVE was developed as a plugin to Unreal Engine 4 as it is intended to provide supplementary functionality to existing projects. Plugins promote this use case, and otherwise present no obstacles to implementation. This plugin, however, only makes up the portion of SceneCaptureCAVE which manipulates the capture component. The remainder of the implementation, namely the actor component, is included as a Blueprint which makes use of the plugin function `CaptureAndExport2DCaptureSet()`. Unreal Engine 4 Blueprints can contain logic as well as structural elements and often bridge the gap between code and the engine editor. Because the intended usage of SceneCaptureCAVE may change depending on the context in which it is being applied, it is beneficial to keep its broader functionality available for modification as a Blueprint. This blueprint must reside in the project's content folder, separate from the plugin, where it may then be added as an actor to any scene.

3. Extensibility

Although SceneCaptureCAVE meets the goal of emulating the CAVEcam within Unreal Engine 4, there are remain some improvements which would allow it to be better integrated with the engine. This includes the process by which it is added into projects, as well as optimization and usability.

3.1. Plugin-Defined Actor Type

Presently, SceneCaptureCAVE is a hybrid implementation consisting of both plugin and Blueprint modules. Although this solution provides a simple way to modify the SceneCaptureCAVE actor's logic, it inhibits the streamlined addition of the plugin to a project. Ideally, this method of using

SceneCaptureCAVE would be one available option alongside an all-in-one solution much like the included SceneCapture2D and SceneCaptureCube. This object would be represented by an entirely new type which inherits from ASceneCapture (the base scene capture object) and encapsulates all the functionality from the blueprint module. This object would accept no modification aside from the already established parameters, but in exchange would be far easier to add to a project, requiring only an installation of the plugin to become available in the scene editor.

Additionally, if a new SceneCaptureComponent type were defined to supplement the above actor type, it would become possible to define a custom rendering process for this component, opening up new opportunities for optimization in a similar manner to the way SceneCaptureCube operates. Using this approach, it might even be possible to bypass the creation of render targets altogether and directly write captured scenes to disk. This would involve significant additions to existing rendering code, some of which may not be possible within a plugin, instead requiring direct modification to engine code.

3.2. File Output Format

As of engine version 4.15.0, Unreal Engine 4 contains a complete pipeline which allows for render targets to be exported to a single format, the HDR-radiance image file, which requires an unconventional image viewer to inspect. Some support exists within the engine source for exporting to .bmp, .png, and other file types, but it is currently incomplete. In this absence, several third-party plugins, some even bundled with the engine, have defined their own libraries for exporting to these file types. These plugins are typically much larger in scope than this one, and usually depend on the ability to perform such an action. If it proves useful in the future for SceneCaptureCAVE to output alternative file types, it would be possible to adapt or adopt one of these implementations if native support has not been completed.

4. Conclusions

Prior to the development of SceneCaptureCAVE, there were successful substitutions of Unreal Engine 4 SceneCaptureCube images in place of CAVEcam output, albeit with modifications made to produce the piecewise output that the CAVEcam natively produces[2]. SceneCaptureCAVE completely removes the need to used SceneCaptureCube, by instead providing a dedicated method of achieving the same goal.

SceneCaptureCAVE significantly improves upon the usability of SceneCaptureCube by offering a more robust system for capturing a full 360x180° view of a scene. Because SceneCaptureCAVE was not designed to minimize resource use in favor of real-time performance, many concessions made by SceneCaptureCube to optimize performance can be ignored to increase render quality. By increasing the number of individual samples beyond the six used by the SceneCaptureCube, it reduces the distortion caused when blending the images together into an equirectangular projection. SceneCaptureCAVE also outputs its constituent images directly rather than blending them internally, eliminating a dependency on the Unreal Engine 4 blending algorithm, and removing the

requirement that the output of SceneCaptureCube be reverse-engineered to fit the input standards of the necessary software utilities.

By emulating the behavior of the CAVEcam in a virtual environment, SceneCaptureCAVE effectively allows for the acquisition of stereo panoramas in environments which could not be photographed conventionally. This opens up applications in fields such as architecture, where rendering an intermediate stage in an immersive view could be beneficial to the design process, or even in the rendering fictional locations such as those found in video games or animated films. Additionally, There also exist numerous examples of virtual reality installations utilizing CAVEcam footage successfully, specifically in sites of historical or cultural importance[3]. Such installations could easily accommodate images captured by SceneCaptureCAVE, allowing participants to experience otherwise inaccessible locales.

SceneCaptureCAVE was intentionally developed to mimic the CAVEcam in function and output in order to function as a valid proxy in cases where a CAVEcam system is unavailable. This makes SceneCaptureCAVE an effective substitute for applications which have the ability to display CAVEcam footage, which includes most common virtual reality headsets and 3D televisions, as well as more complex systems such as starCAVE and other CAVE-like environments[2].

In a larger context, SceneCaptureCAVE will satisfy a necessary requirement in ongoing SENSEI research, which specifies the development of a comprehensive pipeline for manipulating omni-stereoscopic panoramic images and video. The success of this project depends on the development of two different systems: a hardware component that is able to capture high-resolution omni-stereoscopic panoramic video and still images, and a software component which is capable of processing, displaying, and analyzing such footage. Research into each of these components is currently being pursued simultaneously, which presents challenges for validating functionality of both elements.

The CAVEcam has been selected as one of the potential SENSEI hardware solutions, and is thus a valid interface against which the associated software can be developed. In the development of this software which accepts CAVEcam imagery, it is not always possible to directly gather data using the CAVEcam due to time or scheduling constraints. In these situations, SceneCaptureCAVE can function as an alternative source, providing data that can be substituted in place of that from an authentic CAVEcam. A full set of suitable images can also be produced by SceneCaptureCAVE in a matter of minutes, a fraction of the time the CAVEcam would require to produce a similar collection. This increases the availability of testable data and allows research into a SENSEI software solution to continue independently of a concrete hardware prototype, thus allowing researchers without access to a CAVEcam an opportunity to continue concurrent development.

By making progress towards a particular software solution, the potential of its respective hardware will be improved, meaning that any advances in one department will have tangible effects on the other, and thus a significant effect on the entire research initiative.

As valid input to SENSEI software, SceneCaptureCAVE images will be immediately receptive to the complex analytical processes it already provides. These primarily include point cloud projection and depth map stitching, which allow for scene geometry and other data to be extrapolated from panoramic images.[2] In some cases, the accuracy of these techniques may even be improved though the integration of data that can be extracted from Unreal Engine 4 such as mesh geometry, movement vectors, and high-dynamic range exposure, all of which directly pertain to particular problems that are currently being investigated under the SENSEI initiative[2].

It is anticipated that the development of SceneCaptureCAVE will have immediate applications in many of the aforementioned areas of research, particularly in the ongoing development of a SENSEI software component. If SceneCaptureCAVE proves to be a useful element in these applications, it may also generate interest in exploring the potential of omni-stereo panoramic scene capture as not only a supplement to conventional photography but a valuable technique in its own right.

References

[1] Ainsworth, Richard A., Daniel J. Sandin, Jurgen P., Schulze, Andrew Prudhomme, Thomas A. DeFanti, and Madhusudhanan Srinivasan. (2011). Acquisition of stereo panoramas for display in VR environments. In *IS&T/SPIE Electronic Imaging*, pp. 786416-786416. International Society for Optics and Photonics.

[2] Brown, Maxine, Mahmoud Manzoul, Robert Kooima, Jules Jaffe, Jason Leigh, and Truong Nguyen. (2016). SENSEI Annual Report, October 1, 2015 – September 30, 2016

[3] Smith, Neil G., Richard A. Ainsworth, Jurgen Schulze, Robert Kooima, Daniel J. Sandin, and Thomas A. DeFanti. (2013). Cultural Heritage Omni-Stereo Panoramas for Immersive Cultural Analytics – From the Nile to the Hijaz. In *Image and Signal Processing and Analysis, 2013 8th International Symposium*.

[4] Unreal Engine 4. https://github.com/EpicGames/UnrealEngine/.