TELEMETRY RECONSTRUCTION AND ANALYSIS USING VIRTUAL REALITY

Katherine Verges, Richard Graham
NSWC Corona Division
Norco, CA 92860
katherine.verges@navy.mil, richard.a.graham@navy.mil

ABSTRACT

Currently, telemetry analysis is visually disconnected from the system being analyzed; analysts look at a series of two dimensional functions plotted over time that represent everything that happens. As the digital age continues to evolve and grow, a new technology is emerging in the world of entertainment: Virtual Reality (VR). VR describes a system that uses a headset to create a completely manufactured environment for the user to utilize and explore. This technology can be harnessed in order to translate raw telemetry data into an all-inclusive image of a system being analyzed in a 3-dimensional (3D) format. It would allow an analyst to fully visualize results and better understand what is occurring and has the potential to remove some of the subjectivity that comes with analyzing functions in order to help scientists and engineers to more efficiently improve their products. VR technology could be applied in a variety of fields--defense, medicine, biology, and many more—and could help pave the way to technical advancements for a better world.

INTRODUCTION

In today's world, telemetry is utilized across a myriad of fields including flight testing (aircraft, missiles, spacecraft, etc.), animal tracking, and medicine. Currently, telemetry data is displayed as line functions plotted against time. It can take anywhere from months to years for a new analyst to become competent at analyzing data. The connections between line graphs and the physical system do not easily print a clear picture to the inexperienced and can present opportunities for inaccurate analysis. Problems within the system can be easily overlooked.

Telemetry data functions are also difficult to communicate to those outside of the analysis process such as managers, sales teams, or customers. The implementation of VR would allow the collected data to be displayed as a visual representation of the system. Instead of looking at hundreds of disjointed functions, VR would provide an all-inclusive visualization of the system as well as the relationships between and within all recorded subsystems. This method would aid analysts in finding and identifying problems within the system.

The Test and Evaluation process could be streamlined, data could be analyzed more accurately, and results could be more effectively communicated. A generic flight payload was used to

demonstrate this method of telemetry display and analysis. This method allows for multiple layers of visualization in order to demonstrate the ability to analyze the system as a whole as well as the subsystems and individual components.

This report outlines the implementation of telemetry data into *Unity3D*™, a game design engine, in conjunction with an HTC Vive, a VR headset. A generic data stream was created with several variables that would generally be used for an aircraft flight test. This report explains the process for applying this data to a 3D model and visualizing the recorded data. It explains methods for applying VR technology to the 3D display. Finally, this report provides an explanation of how the user would interact with the telemetry reconstruction in order to thoroughly analyze the data.

DATA PROCESSING

For the purpose of this project, a CSV telemetry data stream was created with several variables that would generally be used for an aircraft flight test. A small sample of this array can be seen in Table 1 below.

In order to display the data as a 3D model, the data was imported into *Unity3D*™. The variables in the array were then linked to components of a 3D model that the user can interact with in order to visualize the events that transpired through the data stream in the system.

```
public List<double[]> data;
string[] column_descriptons;
public void Read_CSV(string filename_in);
{
    StreamReader reader = new StreamReader(filename_in);
    string[] lines = reader.ReadtoEnd().Split('\n');
    column_descriptions = lines[0].Split('.');
    for (int line_index = 1; line_index < lines.Length; line_index++)
    {
        string[] cols = lines[line_index].Split(',');
        if (cols.Length >1)
        {
            double[] values = new double[cols.Length];
            for (int col_index = 0; col_index < cols.Length; col_index++)
            {
                cols[col_index] = cols[col_index].Replace("\r\, "");
                Double.TryParse(cols[col_index], out values[col_index]);
            }
            data.Add(value);
        }
    }
}
```

Figure 1: CSV Implementation Code

Table 1: Sample of CSV Data

Time	Time Relative to Launch	Velocity	Altitude	Right Engine	Left Engine	Navigational System
sec	sec	ft/s	ft	1=on	1=on	1=on
0	-2.5	0	0	0	0	0
0.1	-2.4	0	0	0	0	0
0.2	-2.3	1	0	0	0	0
0.3	-2.2	2	0	0	0	0
0.4	-2.1	3	0	0	0	0
0.5	-2	4	0	0	0	0
0.6	-1.9	5	0	0	1	1
0.7	-1.8	6	0	0	1	1
0.8	-1.7	10	0	0	1	1
0.9	-1.6	15	0	0	1	1
1	-1.5	20	0	0	1	1
1.1	-1.4	25	0	0	1	1
1.2	-1.3	30	0	0	1	1

Each variable in the above array applies to the status of a physical component. Each subsystem within the main system--Right Engine, Left Engine, and Navigational System--would either be controlled by extending this array or by attaching another array to the subsystem. This project used a monolithic array for all variables. However, for larger data series the arrays could be separated by subsystem to prevent all of the data being processed for each subsystem. The full CSV stream/file for this project was imported into *Unity3D*^{TM} using the code displayed in Figure 1.

This code calls the CSV into the scene and initializes it as an object array. This allows the variables within the array to be applied to individual GameObjects within the scene. These GameObjects can now be modified according to the values of the array. The specifics of these modifications are discussed in the next section.

SYSTEM VISUALIZATION

Once the telemetry data was imported to *Unity3D*^{TM}, it was applied to a 3D model for visualization in 3D space. In order to achieve this visualization, a model of the system was required and the components of the system were modified to reflect the data.

A) Model Formation and Variable Definition

Figure 2 defines the breakdown of the systems and subsystems for this project. Each system and subsystem was displayed as separate scenes with different variables within each scene.

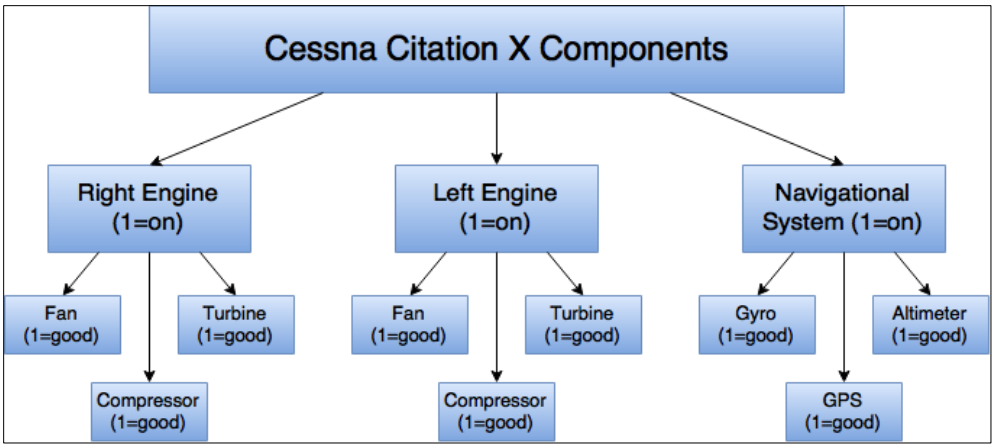

Figure 2: System and Subsystem Breakdown

The main scene displays the entire system being tested, in this case, a Cessna Citation X model was used. The main components used for the test were the Right Engine, Left Engine, and Navigational System. This scene is displayed in Figure 3.

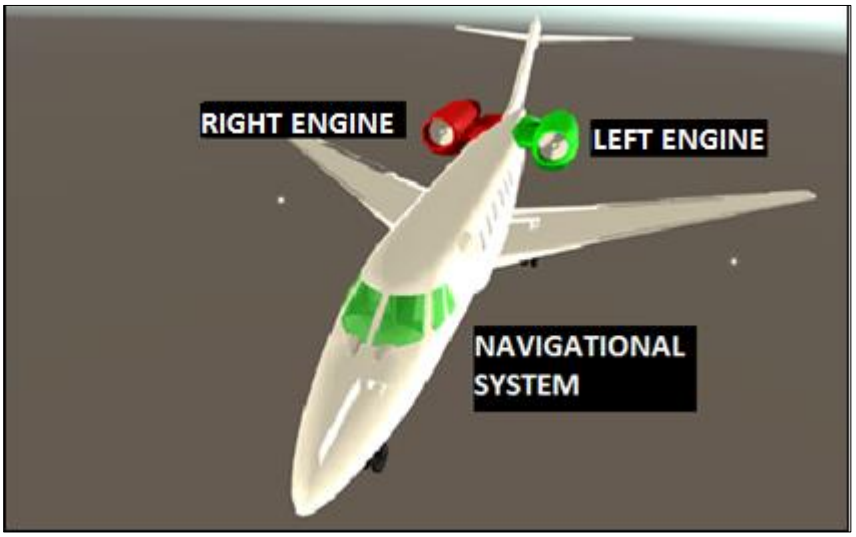

Figure 3: Main Scene

In order to assign variables from the imported array to components in the scene, the CSV was tied to a GameObject array. The code in Figure 4 shows how the GameObject array is instantiated, as obj_array. In *Unity3D*^{TM} the specific game objects were designated for each element in the array by clicking and dragging each game object into the appropriate slots.

```
public GameObject[] obj_array;

//Use this for initialization
void start()
{
    print("Setting Data");
    data = new List<double[]>();
    Read_CSV();
    slider.maxvalue = data.Count - 1;
    print("Max: " + slider.maxValue.ToString());
    set_Data_Raw(0);

    set_obj_color(obj_array[0], (int)data[0][3];
    set_obj_color(obj_array[1], (int)data[0][4];
    set_obj_color(obj_array[2], (int)data[0][5];
    set_obj_color(obj_array[3], (int)data[0][6];
    set_obj_color(obj_array[4], (int)data[0][7];
}
```

Figure 4: Variable Assignment Code

Each subsystem of the plane was developed using separate scenes. Each of these scenes was written with a separate script, however, each scene sourced from the same CSV file. An example of one of the sub-scenes is shown in Figure 5

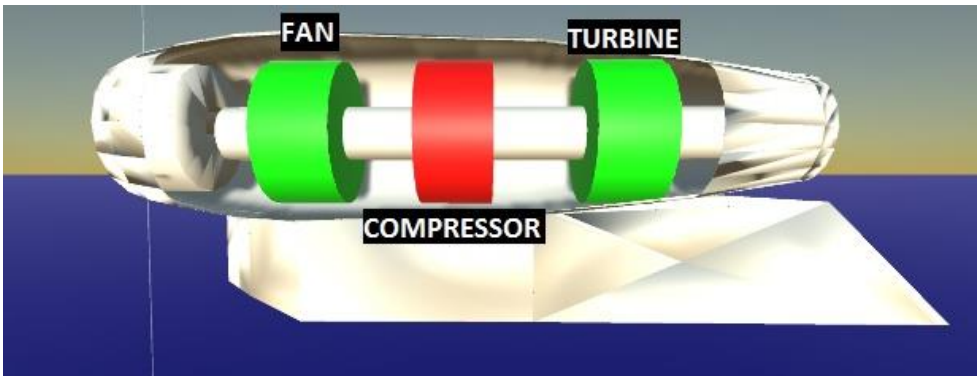

Figure 5: Right Engine Scene

As shown in Figure 2, each sub-scene has its own set of components. These components are GameObjects in the scene and are tied to variables in the CSV.

Figure 5 displays the three components of the Right Engine: Turbine, Compressor, and Fan. These are the main components that make up the turbofan engine of a Cessna Citation X. Each of these components was tied to a variable in the CSV that was imported just like the components in the main scene.

The telemetry data indicate the status of the systems and subsystem as either ON (value =1) or OFF (value =0). These systems, and their corresponding GameObjects, can be color coded to visually inform the user of the status. The script for applying these colors to the components is shown in Figure 6

For example, at 0.6 sec, the main scene in Figure 3 of the full Cessna system shows the Left Engine and the Navigational System in green while the Right Engine is red. This indicated that something within the Right engine is off. Table 1 is a limited list, but within the CSV it was

indicated that the Compressor within the Right Engine was malfunctioning. This was observed in Figure 5.

```
public void Back()
{
    SceneManager.LoadScene("main");
}

private void set_obj_color(GameObject obj, int set)
{
    if (set>=1)
    {
        obj.GetComponent<Renderer>().material.color = Color.green;
    }
    else
    {
        obj.GetComponent<Renderer>().material.color = Color.red;
    }
}
```

Figure 6: Criteria Definition Code

B) Application of VR

Once the data is imported and the GameObjects are designated to their respective telemetry variables, , the conversion to VR is relatively simple. Applying the VR component is wholly dependent on which device is being used. *Unity3D*™ uses a check box in Player Settings that allows for a smooth conversion to VR when Oculus Rift drivers are installed. The in-scene camera will automatically change to VR view.

This project utilized the use of an HTC Vive which has a little more of an in-depth process. Separate assets had to be downloaded from the Unity Asset Store which included a specialty camera rig to allow for VR view. From there, the camera rig just had to be tied to the desired object to be used by simply dragging and dropping the rig onto the GameObject

C) User Interaction

In order for this system to be useful, the user must be able to navigate the scene while wearing the VR goggles. They must be able to select and isolate components for thorough scrutiny. They must also be able to manipulate time to be able to see cause and effect or slow down an event. A User Interface (UI) as well as input controls had to be put into place.

In order for the user to navigate the scene, a First Person Controller (FPC) was added to the scene. The FPC is a standard software asset within *Unity3D*™. In the scene, the FPC is nothing more than a cube GameObject with an active camera on it. Once VR was applied and the VR camera rig was tied to the FPC, the user could control the camera by simply moving around the scene while seeing through the camera. Head rotation was able to be detected by applying the code shown in Figure 7 below.

```
//Store the Euler Rotation of the FPC Game OBject
var eulerRotation = trasnform.rotation.eulerAngles

//set the rotation to the same as the user's
eulerRotation.x = 0;
eulerRotation.y = 0;
eulerRotation.y = InputTracking.GetLocalRotation(VRNode.Head);
```

Figure 7: VR Head Tracking Code

The user must also be able to select components in order to isolate different subsystems. *VREyeRayCaster* is a script placed on the main camera in the scene, in this case, the VR camera rig on the FPC. Every update casts a ray forward using *Physics.RayCast* to see if it hits any colliders. If a collider is detected, it tries to find a *VRInteractiveItem.* These items were the system components.

A *VRInteractiveItem* can be modified to react upon VR interaction. In this case, the component would brighten in color when its collider was detected. The next step is to select it. Code was implemented to allow the scene to switch to the associated sub-scene when the component was clicked on. This "click" is something different depending on the hardware used. In this case, the "click" was mapped to a button push on an HTC Vive controller. Alternatively, it could also be a mouse click or a button on a standard video game controller. The code to allow the user to gaze at a component, select it, and change scenes is shown in Figure 8 below.

```
//Update is called once per frame
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;
    if (Physics.Raycast(ray, out hit))
    {
        if (hit.trasnform.name == "Cube (1)")
        {
            SceneManager.LoadScene("Scene0");
            }
            else if (hit.trasnform.name == "Cube (1)")
            {
                SceneManager.LoadScene("Scene1");
            }
        }
    }
}
```

Figure 8: User Gaze and Select Code

Although a great deal of telemetry data for most systems can be displayed as a part of a 3D model, some of it cannot. For example, flight systems, such as the one demonstrated, have functions of velocity and body motion. The visualization of these variables is relative to the user's view. In order to display these things, as well as view the time at which events occur, User Interface (UI) components were put into place. These elements were displayed on a canvas that is parallel to the camera view at all times. They auto propagate so they are always in the user's field of view. Other UI elements were input in order to allow the user to hide these displays.

The outputs displayed in the UI for this project are Time (sec), Time Relative to Launch (sec), Velocity (ft/s), Altitude (ft), and Events. All of these outputs were variables in the imported CSV file and were displayed in all scenes. Time represented the time from when the data began the simulation. It was displayed in 0.1 sec intervals. Time Relative to Launch displayed the time where launch (or take-off) occurred. Anything before that point was a negative value. A UI Slider was implemented in order to allow the user to view the data at certain points in time. The user could use the slider to move forward or backward through the data or they could use the arrows to skip forward by 0.1 s with each button push. Velocity displayed the total system velocity during flight. Altitude displayed the height of the system relative to the sea level. The Events Display showed a log of Events that occurred such as Take-Off, Left Engine Fail, and the time at which the Event occurred relative to launch. The events and times propagated a list for user reference. For variable functions, boundaries and criteria can be set so that if the value of the function varies outside of the criteria, the subsequent Event will display. Figure 9 shows the final scene with the numerical outputs activated.

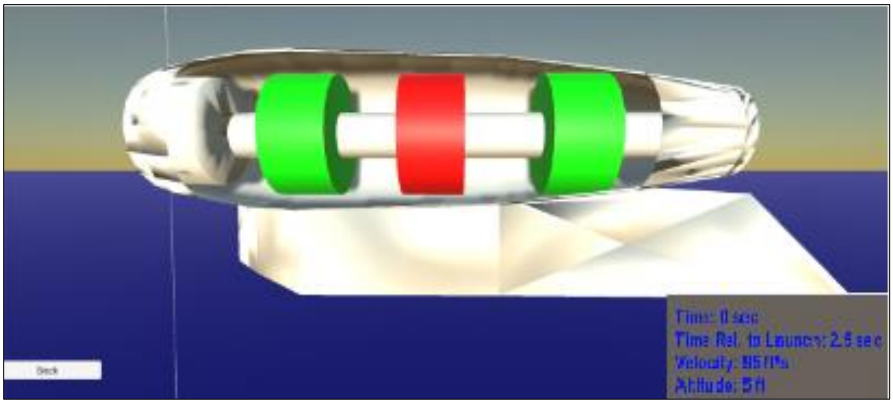

Figure 9: Scene with Floating UI

CONCLUSION

At this juncture, telemetry data is displayed as a series of functions for analysis. Analysts need a lot of time to learn the system and leaves room for errors to be made. Some relatively simple 3D reconstructions exist in order to display data or to untangle more complex events. However, these are generalized and do not display the connections between all the components of a system. The application of VR would bridge the gap between the data and the system. It would allow for connections to be made more easily and could increase the amount of certainty analysts have in what is occurring. In addition to VR, there is also the potential that Augmented Reality (AR), where the simulated model is layered on the real world, could be applied.

REFERENCES

[1]A. Moran, V. Gadepally, M. Hubbell and J. Kepner, "Improving Big Data Visual Analytics with Interactive Virtual Reality", Cambridge, MA, 2015.
[2]M. Geig, *Unity Game Development in 24 Hours, Sams Teach Yourself*. Upper Saddle River: Pearson Education, 2013.

[3]"Template Cessna Citation X free 3D Model .max .obj .3ds .fbx .stl .dae", *CGTrader*, 2015. [Online]. Available: https://www.cgtrader.com/free-3d-models/aircraft/sport-private/template-cessna-citation-x. [Accessed: 25- May- 2016].
[4]"Unity - Manual: How to do Stereoscopic Rendering", *Docs.unity3d.com*, 2016. [Online]. Available: http://docs.unity3d.com/Manual/StereoscopicRendering.html. [Accessed: 02- Jun- 2016].
[5]"Unity - Interaction in VR", *Unity3d.com*, 2016. [Online]. Available: https://unity3d.com/learn/tutorials/topics/virtual-reality/interaction-vr. [Accessed: 01- Jun- 2016].
[6]"Unity - Manual: Scripting", *Docs.unity3d.com*, 2016. [Online]. Available: http://docs.unity3d.com/Manual/ScriptingSection.html. [Accessed: 16- Mar- 2016].