Immersive Visualisation of Big Data for River Disaster Management

Matthew Ready* Monash University
Tim Dwyer† Monash University
Jason H. Haga‡ National Institute of Advanced Industrial Science and Technology

Abstract

We present a virtual reality visualisation of pre-recorded data from approximately 18,000 weather sensors placed across Japan. Utilising the HTC Vive and the Unity engine, we developed a novel visualisation that allows users to explore data from these sensors in both a global and local context. We also investigated a variety of methods for interaction using the Vive's motion controllers. The user is able to fly around the map, to open an interactive window for any sensor, and to seek their position in time throughout the month of recorded data.

Index Terms: K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; K.7.m [The Computing Profession]: Miscellaneous—Ethics

1 Introduction

The Japanese government makes a large amount of data from a wide variety of weather sensors available to the public via a simple web interface (`http://www.river.go.jp/`) provided by the Ministry of Land, Infrastructure, Transport and Tourism (MLIT) [?]. Browsing the data with this interface is performed through a list of small, pre-defined maps on which links to the sensors are overlayed. Using this system alone, it is impossible to get any "big picture" idea of how the sensors appear on a state or even country level. Yet, the website is intended to be used to assist in organizing a response to potentially large scale natural disasters (storms and related flooding, etc.). To investigate the utility of a broader perspective for disaster management, we scraped data from the website over a period of a month and brought the millions of collected sample points into a virtual reality visualisation. Keeping the shortcomings of the website in mind, we created an interface that allows both a "big picture", country-wide view while also allowing a the user to query each individual sensor. Such a design should allow for great flexibility in data analysis and interpretation. A recent publication highlighted the use of local rainfall data from X band multi-parameter radar provided by MLIT on the DIAS [?] system for disaster management, which is a platform for global environmental big data [?]; however this work used traditional webpage visualisations for the rainfall data. We were interested in creating a novel application that presented data in different ways to faciliate understanding and engagement of novice users. Inspired by recent work in Immersive Analytics [?], we also opted to explore the possibilities offered by new immersive display and interaction technologies.

2 Technical obstacles

Several technical obstacles had to be overcome in order to display all the sensors on screen at once, whilst also keeping above the strict 90 FPS requirement to avoid VR induced motion sickness [?].

*e-mail: matt-r@hotmail.com
†e-mail: Tim.Dwyer@monash.edu
‡e-mail: jh.haga@aist.go.jp

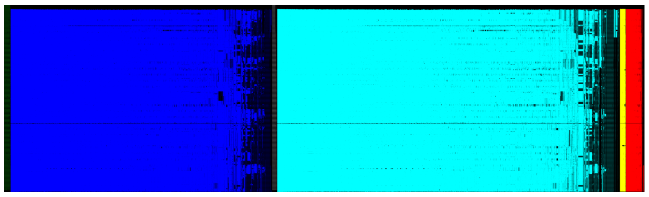

Figure 1: A visualisation of recorded data points. Time runs from top to bottom, 10 minutes per row. Each column represents a single sensor, and the colour represents the type of sensor (green = water quality, blue = river height, white = snow, light blue = rainfall, yellow = shoreline, red = dam). A pixel is coloured when data was recorded, and black when offline.

2.1 Custom database

The visualisation required some method of seeking to any time in the recorded data and viewing the state of all sensors at that time. Initial testing revealed that running such a query took approximately 10 seconds on the 6.9GB SQLite database that was scraped from the website. Card *et al.* [?] claim that "perceptual processing" requires a response time of under 10ms, so a faster data store was required. While SQLite is excellent at answering many different kinds of queries very quickly, we care only about one specific kind of query. It was decided to create a simple custom binary database to store the downloaded data.

In order to design a database format, understanding the data to be stored is key. To this end, a simple visualisation of the sensor data was created (Fig. 1). After careful inspection of this visualisation, it was immediately seen that we had a very dense matrix of data. Some sensors were offline or otherwise reported no data, but this was for a very small amount of time. Thus, complex data structures to compress empty space were overkill, and a very simple format could be constructed.

Each sensor on the website consists of several 'channels'. For example, a rain sensor has a channel for both instantaneous and cumulative rainfall. Each channel consists of a series of numbers mixed with occasional null values which represent when the sensor was offline. Values for each channel are updated on the website every 10 minutes. For each channel, an appropriate storage class is selected: 1 byte, 2 bytes or 4 bytes depending on the maximum number of decimal places and the range of possible values. Then, for each ten minute period in the recording, the current value of each channel is encoded into a stream of bytes (a 'row'). The database is simply a concatenation of these rows. Since the length of a row is a constant, running a query involves seeking to the correct location in the file and reading exactly one row. Essentially, the custom database becomes a greyscale image where bit depth varies in each column.

Initial testing revealed the same queries running against this custom database take ∼7ms: 1500 times faster. Additionally, the custom database format is also around 50 times smaller, at about 132MB. Responsive, interactive seeking was now easily possible.

2.2 Bar visualisation

In order to display the sensors on the 3D map of Japan, it was decided to use vertical bars that change in height to represent the current value of their sensor. However, given the sheer number of bars, there was not enough bandwidth between the GPU and CPU to update the entire mesh of bars every frame that the heights change as the user seeks through the data in time. Moreover, every bar has to be constantly oriented to face the camera, otherwise the flat nature of the bar is obvious (this technique is known as "billboarding"). Recent GPUs have a feature known as 'geometry shaders'. Such a feature allows one to generate the mesh directly on the GPU without any data needing to be transferred between the CPU and GPU, and was a good choice here[1]. With this approach, only a small 18KB data texture that contained the heights of all bars needed to be transferred between the CPU and GPU when the heights changed.

2.3 Terrain

Due to development time limitations, the map of Japan is currently rendered using a single instance of Unity's built in terrain engine. As a result, the terrain's heightmap is only 2049x2049 pixels, and the terrain's texture is only 8192x8192 pixels. This makes the terrain look quite blurry when viewed close up. Because GPU support for tiled resources is limited to Direct X 11.2 and above, we decided to avoid them. As a result, only software emulation (swapping in and out varying levels of the map as the camera moves around) remained a viable option. Implementing this technology would require additional time and is an important aspect of future work for this application.

3 User Interaction

3.1 Movement

We implemented two different methods of controlling user movement in the visualisation.

- View orientation based movement: In this method, we used the orientation of the user's head to direct the movement. By pushing up and down on the controller, the user could move forwards and backwards with respect to where they were looking.

- Controller orientation based movement: In this method, we used the orientation of the controller to direct the movement. By pushing up and down on the controller, the user would be pushed in whatever direction the controller was facing.

After testing both of these methods, we found that the controller based movement felt more natural. Thus, in the virtual environment, where you look has no effect on where you move, and it was much easier to control where you wanted to go by simply pointing with your hand.

3.2 Selection

A key feature of the visualisation was the ability to point at a sensor with the motion controller to select it. Implementing this required both an octree implementation (for locating the nearest sensor when the user is not pointing directly at one), and leveraging Unity's internal physics engine for raycasting onto objects in the scene.

Objects in Unity's physics engine are each represented by 'colliders'. There are various types of collider, however the one selected for the bar visualisations was the capsule-shaped collider. Testing revealed that it was impossible to update the heights of all 18k colliders when the user seeked through the data in time without bringing the frame-rate to a crawl. Unity's physics engine does not seem to be optimised for such frequent updates. To address this, a scheme was devised to allow fast raycasting using Unity's physics engine, but without having to resize all the colliders (Fig. 2).

[1]While a simple vertex shader could suffice, Unity vertex count limits would require splitting the mesh into several pieces adding unnecessary complexity.

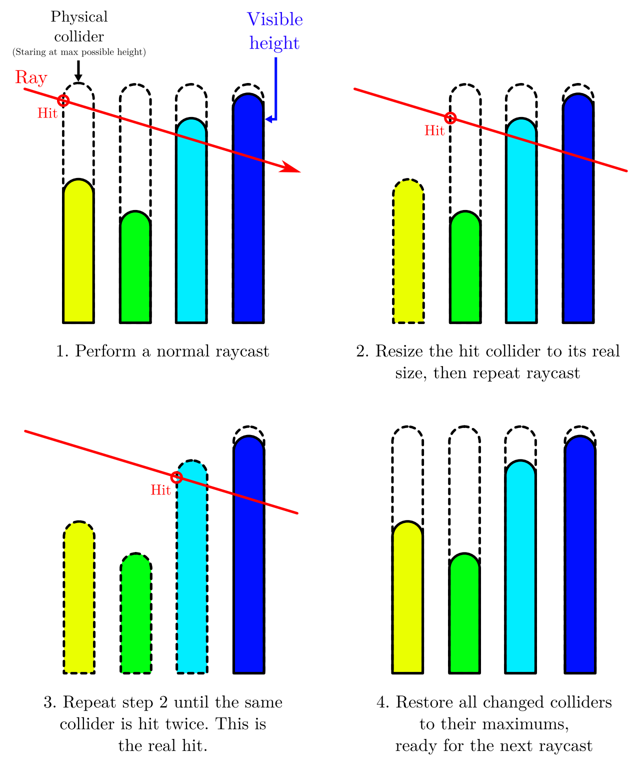
Physical collider
(Staring at max possible height)
Visible height
Ray
Hit
1. Perform a normal raycast
Hit
2. Resize the hit collider to its real size, then repeat raycast
Hit
3. Repeat step 2 until the same collider is hit twice. This is the real hit.
4. Restore all changed colliders to their maximums, ready for the next raycast

Figure 2: The simple raycasting procedure used to deal with Unity's inability to deal with frequent updates

In this scheme, we do not resize the visualised bar's collider to the bar's actual height, but instead set all colliders to the maximum height the visualised bar can ever reach. In this case, a raycast will usually hit the empty space above the visible portion of a bar, when instead it should pass through it. Clearly this is not the correct behaviour, thus when a collider is hit with a raycast, we resize the collider to the correct size (matching the bar visuals) and repeat the raycast. The second raycast will pass straight through the empty space, and hit the next collider behind it. We then repeat this process at the next collider. However, if the raycast were to instead hit the first collider again after it has been resized, then we know that the raycast does indeed terminate at this bar visualisation. In other words, this approach provides an admissible heuristic for the true raycast.

3.3 Physical Windows

Another major feature of the visualisation was the ability to see individual charts of a sensor's data over time. This involved a variety of difficulties:

- Unity has no way to draw a set of lines in it's UI system. A custom UI component had to be created that could directly draw lines to vertex/index buffer objects.

- It was decided that if a sensor is offline for a period of less than one hour, the data at each ten minute period during this timespan should be an interpolation of the two surrounding data points. In this case, such interpolated points are shown in grey to differentiate them from real points. There were a number of edge cases involved that made this necessary in the chart visualisation.

- The chart had to be able to be zoomed in and out.

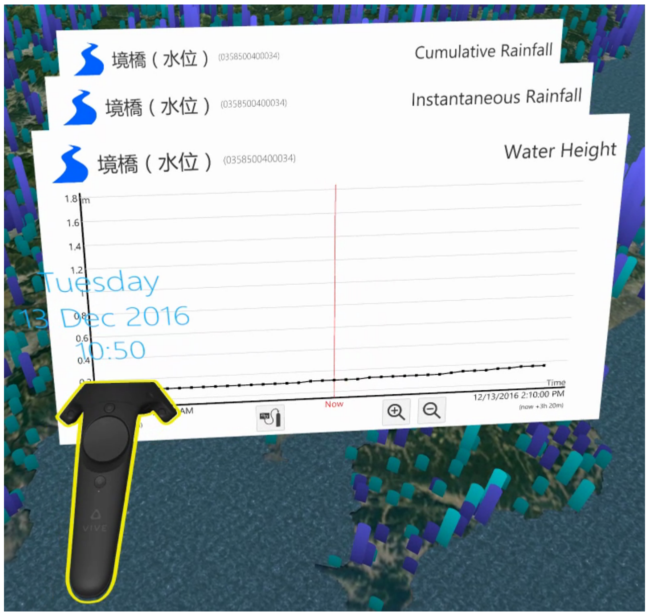

Figure 3: Several charts available to view. Note that the yellow outline of the controller is a visual cue indicating that the user can grab the selected chart and pull it out of the stack.

- The range of the chart and where to draw the axis ticks (the numbers on an axis) had to be decided.
- The charts needed the flexibility of physical manipulation in the virtual space (i.e. they could be grabbed and moved around), but they also needed to stay in a fixed position when placed in the virtual space.
- The buttons in Unity's UI system needed to be clickable by pointing and grabbing with the motion controller.

With this design, users are able to point at any of the sensors and press the controller's trigger to open a list of charts for that sensor (Fig. 3). The other controller is then used to grab and pull out charts that the user is interested in, which can then be placed around the user in virtual space. When the trigger is released, any charts that have not been pulled out will then disappear. The charts placed in virtual space float in mid-air wherever the user placed them, and remain there relative to the user's position (including when the user moves, Fig. 4). To remove a chart, the user simply grabs it, throws it away, and it explodes as a firework to increase the level of engagement. This would allow disaster management professionals to maintain a global and localized view at the same time when viewing the data and making decisions.

3.3.1 Elastic ropes

Once a chart had been created, it was difficult to remember which geolocation point it came from. It was decided that the user should be able to toggle the visibility of a line from the chart to the sensor. Rather than drawing a straight line, a more animated, physical approach was taken to increase user engagement.

First, a cubic Bèzier is drawn to connect the chart to its sensor. This curve is approximated with line segments, and then perturbed (Fig. 5). During each frame, 6 springs connected to either side of each vertex to 6 neighbouring vertices are simulated using Hooke's law with an appropriate target length. Note that a single spring is not sufficient: the band will often become extremely pointed and will zig-zag when vertices become too close (Fig. 6). The effect

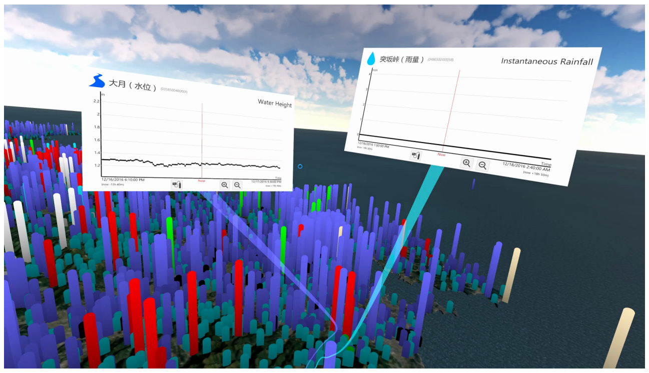

Figure 4: A view of the virtual space with two charts a user has placed in space. The geolocation lines have been toggled on to show which sensors are being referred to. The main visualisation of sensors in Japan appears in the background.

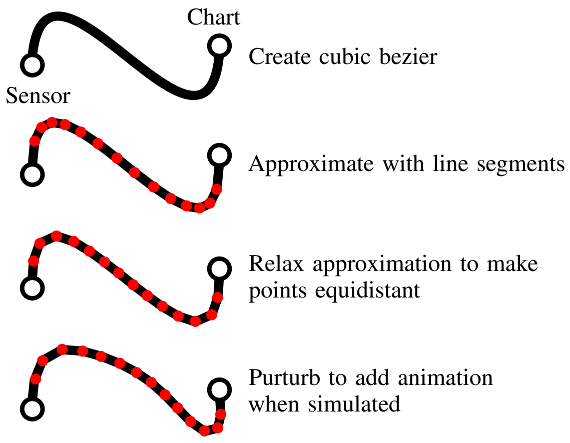

Chart
Sensor
Create cubic bezier
Approximate with line segments
Relax approximation to make points equidistant
Purturb to add animation when simulated

Figure 5: Construction of the line's initial state

produces a springy elastic band type connection between the chart and sensor that users enjoy playing with (Fig. 7).

4 Results

To investigate the visualisation's effectiveness, we asked people who had not seen the application during development to attempt to use the it and we offer here an anecdotal report of their responses and our observations of their behaviour while using the system. This very informal user testing demonstrated that users struggled with moving around in the visualisation. In particular, users will keep their body in a fixed orientation, only looking up and down. As a result, using the controllers to move around the visualisation was difficult. However, once it was demonstrated that you are free to walk and look around in all directions, this problem appeared to quickly subside. Some users also appeared to have difficultly with the two controllers i.e. one controller for movement, the other controller for data seeking. We believe that the reason for this stems from, in part, the lack of a simple tutorial for the application, guiding the users through the controls. Nevertheless, it was encouraging to see even users with no prior experience with VR were able to grasp the concepts and manipulate the visualisation with only a small amount of direction. Users did appear to enjoy the ability to view the sensors on a country-wide and local level simultaneously.

In the future, it would be of great help to receive the insight and advice of domain experts. Only after speaking with them will we be better able to discern exactly what kinds of tasks they need to

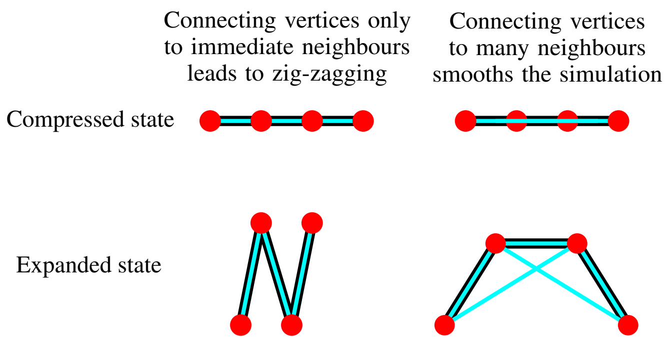
Connecting vertices only to immediate neighbours leads to zig-zagging
Connecting vertices to many neighbours smooths the simulation
Compressed state
Expanded state

Figure 6: Additional springs help to smooth the simulation

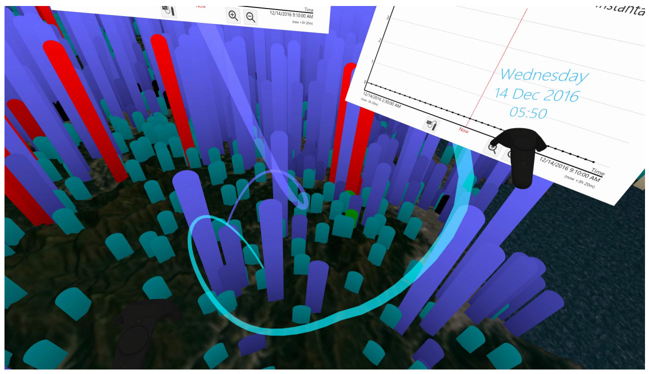

Figure 7: The elastic ropes in the visualisation, joining the sensor chart to the sensor itself.

complete, and how we might address their concerns using VR.

5 Conclusion

Creating a visualisation that is both useful and usable in VR was quite difficult. Maintaining a 90FPS to avoid VR motion sickness while also trying to place as much data on screen as possible introduced several new challenges. Even with only about a month of simple time-series data, several gigabytes had to be processed. It was only through investing additional effort in a custom database solution that the vision of the project could be achieved. The juxtapostion of 2D (i.e chart) and 3D (country-wide) versions of the sensor data appears to be quite successful with intial users. In addressing many of the challenges, we believe this work provides some important insights into working with big data in virtual reality environments. However, while the visualisation may seem a success from our point of view, only through testing with real domain experts will we be able to identify areas that need improvement and extension. It is hoped that the project in its current state can be used as a starting point with which to engage officials from the relevant areas in the Japanese government. They will likely be able to provide us with the necessary input to establish more novel and domain specific visualisation techniques in the project.

Acknowledgments

This work was performed as part of the Monash Undergraduate Research Projects Abroad (MURPA) and the AIST Student Internship Program. It was supported in part by the AIST ICT International Team.

References

[1] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko. vispubdata.org: A Metadata Collection about IEEE Visualization (VIS) Publications. *IEEE Transactions on Visualization and Computer Graphics*, 23, 2017. To appear. doi: 10.1109/TVCG.2016.2615308

[2] G. Kindlmann. Semi-automatic generation of transfer functions for direct volume rendering. Master's thesis, Cornell University, USA, 1999.

[3] Kitware, Inc. *The Visualization Toolkit User's Guide*, January 2003.

[4] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Computer Graphics*, 21(4):163–169, Aug. 1987. doi: 10.1145/37402.37422

[5] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995. doi: 10.1109/2945.468400

[6] G. M. Nielson and B. Hamann. The asymptotic decider: Removing the ambiguity in marching cubes. In *Proc. Visualization*, pp. 83–91. IEEE Computer Society, Los Alamitos, 1991. doi: 10.1109/VISUAL.1991.175782

[7] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for *soft* objects. *The Visual Computer*, 2(4):227–234, Aug. 1986. doi: 10.1007/BF01900346