AR Sandbox Built Within Unreal Engine 5

Gareth McIntosh

***Abstract*—Augmented Reality (AR) is increasingly recognised as a transformative interface for combining the physical and virtual worlds, thereby elevating user engagement. AR Sandbox leverages this capacity through a projection-based AR setup that incorporates a Kinect V2 sensor, a projector, and a box of sand to create an interactive Sandbox environment. In this setup, real-time manipulations inside the physical Sandbox have a direct impact on a digitally rendered visualisation that is then projected onto the sand bed. The defining accomplishment of this project is the porting of Sandbox software that ran on an outdated OpenFrameworks v0.9.8 released in 2018, to the state-of-the-art Unreal Engine 5 platform.**

This technological leap accomplishes two objectives: it updates the application with modern visual capabilities and enhances its scalability and overall performance. The transition grants access to a range of advanced graphical features, debugging tools, and an extensive developer community supportive of Unreal Engine 5. Consequently, this means the project has been freed from the maintenance hurdles that were associated with previous attempts to update the OpenFrameworks system to newer versions. This change opens the door for virtually endless possibilities of further features and improvements. With the switch to Unreal Engine 5, the project now has huge potential for innovative changes that could reshape how we interact with the augmented reality Sandbox.

***Index Terms*—Augmented reality, heightmap, projection, depth detection.**

I. Introduction

AUGMENTED Reality (AR) technology enhances the user's perception of their physical environment by overlaying digital information in real time [1]. This project specifically utilises AR technology to project images onto a bed of sand, achieved through a depth-sensing camera that reads the sand's heightmap and projection software that displays images onto it. This allows users to interact with the sand and change the projections in real time. The objective of this project was to implement newer technologies in order to make long-lasting improvements and reduce the need for frequent updates to outdated frameworks. Specifically, this project ported a working prototype of the code built on OpenFrameworks to Unreal Engine 5.

This project was supervised by Supervisor Dr. Simon McCallum

A. Motivation

The AR Sandbox project offers a captivating and interactive experience that has proven to be highly engaging for audiences, particularly among younger individuals. When showcased at public events like open days and student outreach events, the Sandbox's ability to provide interactivity and immediate feedback has generated significant interest. This concept opens up a wide range of educational possibilities across various fields. For instance, in the medical field, the visualisation of a CT brain scan offers a unique learning tool. The visualisation of Europe with height acting as a scale of time creates exciting opportunities for historical exploration. In addition, another implementation of the Sandbox software utilised by the School of Geography, Environment, and Earth Sciences adds a visualisation to simulate water flow through land formations which provides a hands-on understanding of hydrology. With its versatility and potential for endless educational applications, AR Sandbox great promise. The only limit with regard to societal benefits is one's imagination.

B. The Problem

The main problem this project aimed to solve was the difficulty of updating to newer versions of OpenFrameworks. The existing software was based on Magic Sand [2] which was built on OpenFrameworks v0.9.3 and was released in 2015 [3]. This was a problem as much of the code used in this software had been deprecated in newer versions of OpenFrameworks. Three past students spent significant amounts of time investigating what it would take to update to newer versions of OpenFrameworks. Each student documented their process and while

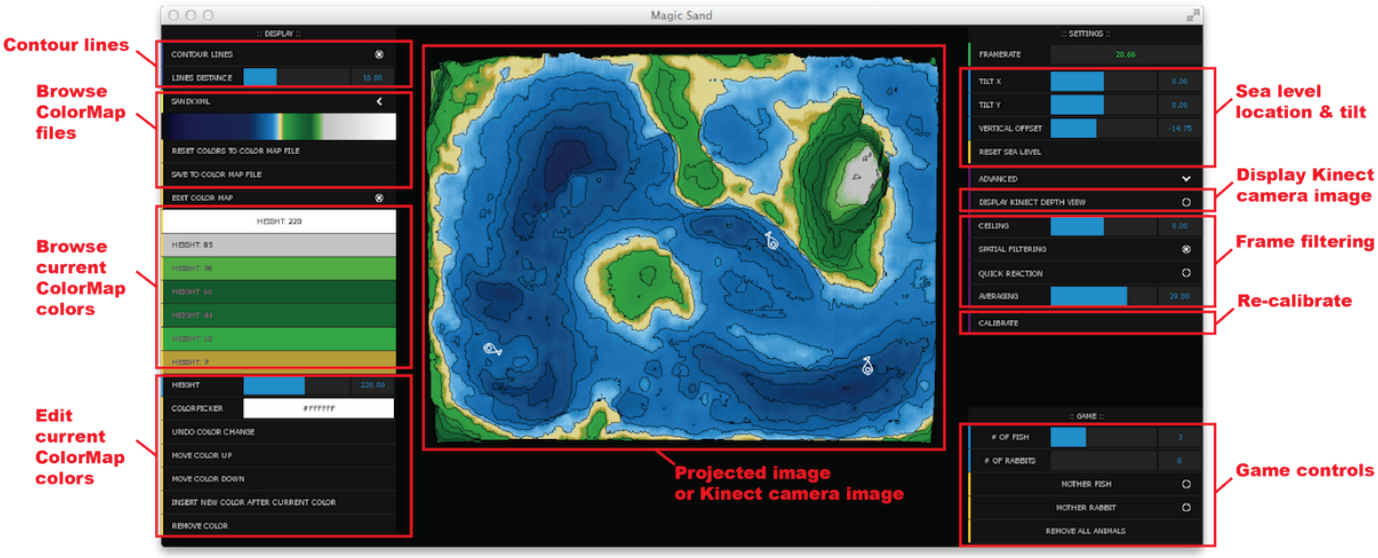

Fig. 1. A screenshot of the Magic-Sand software taken from the project's Github Repository. [2]

some found varying forms of success including updating to OpenFrameworks v0.9.8 [4], the ultimate goal of updating to v0.11.2 was unable to be achieved. The latest student to work on this project discussed the fact that it may be potentially impossible to update to v0.11.2 of OpenFrameworks due to the nature of the segmentation faults and threading issues involved and further mentioned that creating AR Sandbox software from scratch may be the best solution moving forward.

C. Solution

To overcome this problem and ensure future-proofing of the AR Sandbox, the developed solution involved porting the existing codebase from OpenFrameworks to Unreal Engine 5 [5]. This transition to Unreal Engine 5 alleviates the difficulties associated with the outdated frameworks, providing a long-lasting improvement and reducing the need for frequent updates. By implementing newer technologies, this project aims to streamline the update process, saving time and resources while allowing for easier extension and future development of the project. This also has the potential to help the wider community that works on the Magic Sand code on GitHub [2].

The produced artefact is a working prototype that closely resembles the capabilities and functionality of the previous AR Sandbox software while being built within Unreal Engine 5.

II. Related Work

Victoria University's AR Sandbox has been under development since 2018, during which four years involved the contributions of other students' honours projects, as well as input from a Ph.D. student. Understanding the progress made thus far was crucial for this year's project as the goal was to port the code that has been previously developed into Unreal Engine 5 [5].

The initial implementation of the Magic-Sand software involved the integration of an automated calibration system powered by Artificial Intelligence (AI) algorithms. Thomas Gilooly developed this system as part of his Ph.D. research. Thomas successfully developed a fully functional automated calibration system that leverages matrix transformations to ensure swift and precise calibration of the Sandbox. To achieve this, the calibration system utilises two distinct libraries: Levmar [6] and dlib [7]. The levmar library employs iterative techniques for locating the local minimum of a function, working in conjunction with the dlib library, which encompasses various machine learning algorithms. These algorithms allow for accurate calibration of the sandbed.

The next project was undertaken by Nicholas Snellgrove and he achieved the construction of a portable physical Sandbox. As well as this he was the first student to attempt to update the code to the latest version of OpenFrameworks, v0.10.1 [3]. During the update process Nicholas ran into a locking problem which he suspected was caused by changes to the way C++ handled concurrency. In the end, the decision was made that this update was going to be too difficult and therefore take too much time so he settled for updating to OpenFrameworks v0.9.8 instead [3].

Next, David Taing took a unique spin on the project by integrating Virtual Reality capabilities [8]. Similar to Nicholas, David initially attempted to upgrade Magic-Sand to function with the latest version of OpenFrameworks, at this time, v0.11.2, but encountered multiple errors. Consequently, he shifted his focus toward VR implementation. David implemented object detection that would allow users to place objects onto the sand in order to appear in the VR environment. He utilised the Unity Game Engine [9] to develop this as he was familiar with the platform.

Lastly, the previous work done on the AR Sandbox was undertaken by Matthew Jay [10]. The goal of this iteration was to again attempt an update to the latest version of OpenFrameworks, as well as implement a custom visualisation import feature. Matthew was successful in implementing this custom visualisation feature but encountered several difficulties in the attempted update to Open-Frameworks v0.11.2. These problems he encountered were segmentation faults related to memory and threading issues. He concluded his project by stating that future work should not attempt this update again but should instead investigate the possibility of creating the Sandbox software from scratch in Unreal Engine [5].

III. Design

Within this section, an account of the design decisions that were made throughout the project's lifecycle is provided.

A. Depth Sensing Camera

A fundamental element of the AR Sandbox the depth-sensing camera, essential for measuring the sand's height within the Sandbox and facilitating its processing by the Sandbox software.

Fig. 2. A diagram detailing the architecture of a Microsoft Kinect for Xbox 360 [11]

1) Kinect for Xbox 360: The existing AR Sandbox system utilised a Kinect for Xbox 360 due to its availability and compatibility with the existing Sandbox software.

This version of the Kinect utilises a Structured Light process for determining the depth of objects in an image. This process involves projecting a sequence of known patterns onto an object, then observing the object from a camera in a different direction to extract the depth information [12].

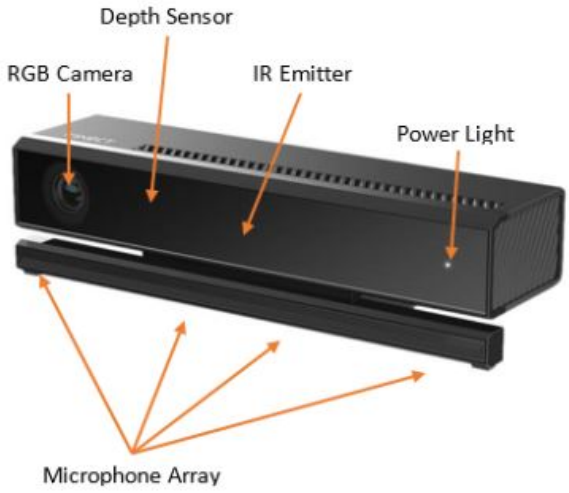

Fig. 3. A diagram detailing the architecture of a Microsoft Kinect for Xbox One [13]

2) Kinect for Xbox One: As this project's goal was to re-build the Sandbox software from scratch, there was a unique opportunity for the system to be designed to work with the newer Kinect for Xbox One.

This newer version Kinect utilises the Time-of-Flight principle to detect the depth. This technology measures the time that light emitted by an illumination unit requires to travel to an object and back to the sensor [12]. This technology would provide more accurate depth detection for the project.

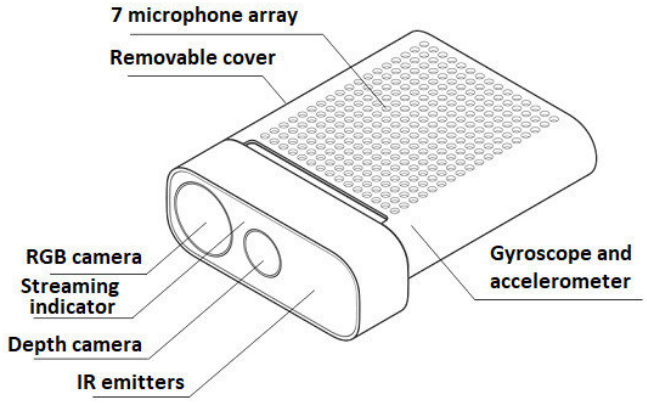

Fig. 4. A diagram detailing the architecture of a Microsoft Azure Kinect [14]

3) Azure Kinect: Another option for a newer depth sensing camera that we had the opportunity to build the system around was the Azure Kinect.

The Azure Kinect, developed by Microsoft, is a depth-sensing camera that was released in 2019.

This version of the Kinect utilises a variation of ToF known as Amplitude Modulated Continuous Wave (AMCW) ToF [15]. In AMCW ToF, a modulated light signal is emitted, and the depth information is determined by analysing both the phase and amplitude of the reflected signal. This approach can offer some advantages, such as improved accuracy and the ability to handle challenging lighting conditions.

4) Advantages and Disadvantages: The advantage of keeping with the Kinect for Xbox 360 is that it is already utilised by the Magic Sand software that is publically used and therefore this project can provide benefits to a wider audience. Utilising the Kinect for Xbox One or Azure Kinect, however, may present a barrier of entry to people who want to use the developed Unreal Engine solution in their existing Sandbox setup.

The advantages of utilising either or these newer iterations of the Kinect is that the depth sensing will be more accurate leading to a more pleasant user experience.

5) Unreal Engine compatibility: The Neo Kinect Plugin is the only plugin available on the marketplace for Unreal Engine 5 that can integrate input from a Kinect into the engine [16]. This plugin supports the Kinect for Xbox One but not the Kinect for Xbox 360 or Azure Kinect [17]. There is however, a plugin on Github that claims to integrate the Azure Kinect into the engine [18].

Considering all these facts and also since we had one in the HCI lab already, we decided to utilise the Kinect for Xbox One for this project.

B. Software Frameworks

Unreal Engine 5 supports development in both C++ and Blueprint, each coming with its own set of advantages and disadvantages.

C++ offers greater performance and lower-level control over game engine internals, attributes that are critical for real-time applications like the AR Sandbox. This low-level control would especially be beneficial in implementing pixel-manipulation algorithms which the Sandbox relies upon.

On the other hand, C++ has a steeper learning curve and since I was not as experienced in programming in C++ compared to Blueprint this would likely lead to more time for development and debugging.

Blueprint, being native to Unreal Engine 5, provides a more intuitive and integrated development environment, particularly advantageous when constructing user interfaces (UI). Its visual scripting approach allows for rapid prototyping and iteration, simplifying the creation of UI elements and interactions. Additionally, Blueprint's real-time visual feedback allows for instant adjustments to the UI, leading to a seamless design workflow and ensuring a user-friendly interface that aligns with the project's objectives.

Another benefit of utilising Blueprint is its alignment with the project's primary objective: porting the original Sandbox to Unreal Engine. Blueprint efficiently harnesses the engine's capabilities for graphical rendering, performance, and scalability. It provides an intuitive and integrated development environment, simplifying the process and ensuring the project can make the most of the engine's features.

Due to these reasons, the decision was made to attempt to achieve as much as possible through Blueprint. If it was found that Blueprint was incapable of a particular feature only then did we resort to coding in C++.

C. AR Sandbox Architecture Design

The AR Sandbox System design can be expained by the following architecture diagram.

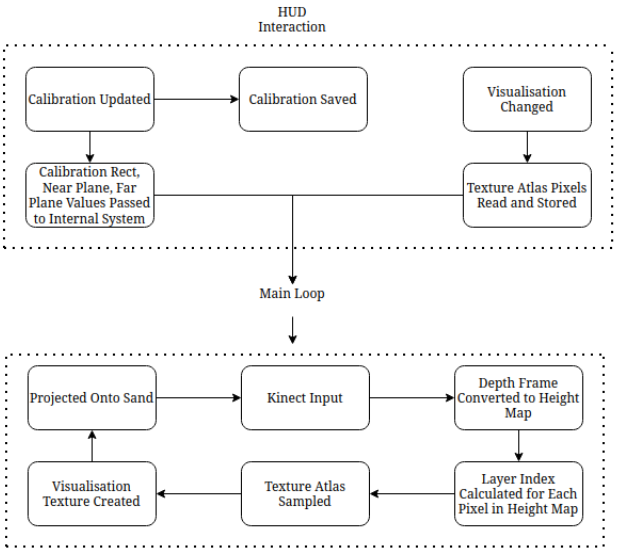

Fig. 5. Architecture of the Sandbox system.

1) HUD Interaction: The HUD design for the system is not complex. The **calibration** is able to be updated manually and **saved**. If the calibration is updated these values are **passed to the internal system** to be utilised in the depth calculations. There is also the option to **change the visualisation** that is being projected. When this is changed, the **texture atlas** for that visualisation will be read and stored.

2) Main Loop: The main loop of the system begins with the **Kinect input**. This depth frame taken from the Kinect then gets converted to a **height map**. The **layer index** is calculated for each pixel in the height map and this is used to sample the **texture atlas**. These sampled pixels create the **visualisation texture** which is then **projected onto the sand**. This loop is occuring on every event tick which is approximately 60 times per second. This means that when the sand is moved the kinect will detect this and the visualisation will be updated instantly.

D. Visualisation layers

Each visualisation is represented in layers where each layer is a seperate .tif file.

E. Height Map

The height map is passed as input by the Kinect, each pixel has a depth value that can range from a distance of 0.5 metres to 4.5 metres [19].

The depth information for each pixel from the height map is used to find the pixel's associated layer index as determined by the following formula:

$$layerIndex = (\text{normalisedDepth} \times (\text{NumLayers} - 1))$$

F. Visualisation texture

The visualisation texture which is projected onto the sand surface is generated by fetching, for each pixel, the RGB colour from the appropriate layer as determined by the calculated layer index.

G. Volume Texture Approach

Unreal Engine 5 includes support for a 3D texture known as a volume texture. By vertically combining each layer image for a visualisation into one extended image file, these volume textures can be created. The idea for this approach was to create these textures for each visualisation then grab the height map depth information from the Kinect and essentially carve these into the volume textures leaving behind, from a top down view, the resulting visualisation that would be projected onto the sand surface.

H. Calibration

Calibration is an important piece of the Sandbox system. The first reason for this is that the Kinect captures an area that is inherently larger than the Sandbox that it is positioned above. Any depth information that lies outside of the bed of sand will provide no useful information to the internal calculations and should therefore not be considered at all.

The second reason for calibration is the boundries of the near and far planes of the sand heights. These planes representing the lowest possible point of the sand i.e. the floor of the physical box the sand sits within, and the highest possible point that the sand realistically will be stacked up to. Any depth information that is outside of these ranges should not be considered in the depth information, this allows for the depth information to be effectively spread out to represent the full range of layers within the visualisation.

The previous Sandbox system utilised an automatic calibration to determine the values for these near and far planes, but it was clear that implementation of this into my Sandbox prototype was beyond the scope of the project. Instead, designing a simple but effective manual calibration system was going to be the best solution.

In order to represent the area of the depth frame that is capturing only the sand area, The decision was made to construct a rectangle variable that can be moved in a particular direction by a small number of pixels at a time. Once this has been calibrated it will be able to saved as theoretically this area should not change unless the Kinect camera is physically moved.

The easiest solution for finding the optimal near and far plane values was just going to be adding a range slider that can be changed in the HUD manually until the visualisation appears to be visually correct.

IV. Implementation

A. Creating Height Maps During Runtime

The first stage of implementation was converting the 2D Render Target depth frames into 2D textures that would act as height maps.

An issue was encountered when implementing the reading of the depth data from the 2D Render Target. I managed to get this feature working with the pre-normalised depth data from the Kinect. However, ideally this would also be working for the raw depth data so there was no risk of the loss of any depth information due to the pre-normalisation.

As it turns out, the 'PF_G16' pixel format used by the raw depth Render Target is not supported by the functions used by 'ReadSurfaceData' [20]. This meant whenever pixels were read from this frame, Unreal Engine would crash.

Ultimately, the choice was made to continue development with the pre-normalised depth input. Only if it became apparent that this depth data lacked the required accuracy would I revisit the issue. In the end, as the entire system became functional, it became clear that the normalised depth was adequately accuracte for the project's purpose. Consequently, the concern was resolved, and the solution was generating height maps from Kinect data for each runtime tick.

B. Height Map Carving Into Volume Textures

I utilised ImageMagick [21] to combine each layer of the visualisations into a single image file. This allowed me to create volume textures for each visualisation.

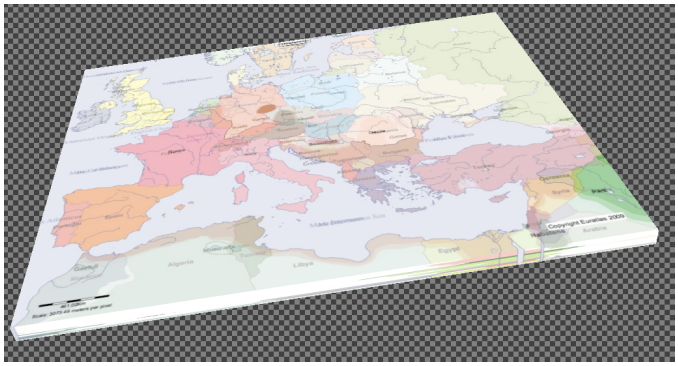

Fig. 6. The Volume Texture of the historical Europe Visualisation created from the combined image file.

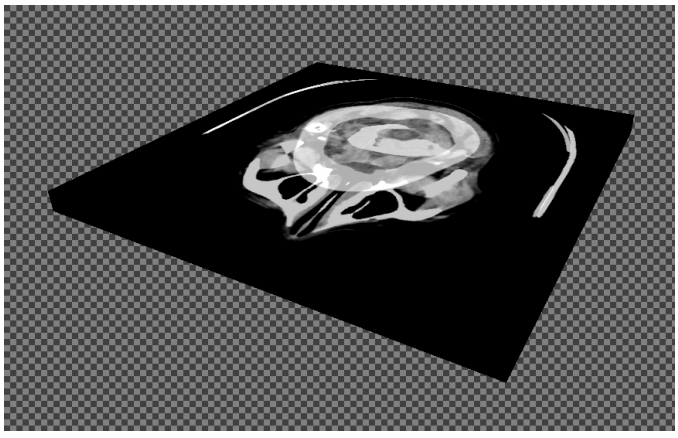

Fig. 7. The Volume Texture of the brain CT scan visualisation created from the combined image file.

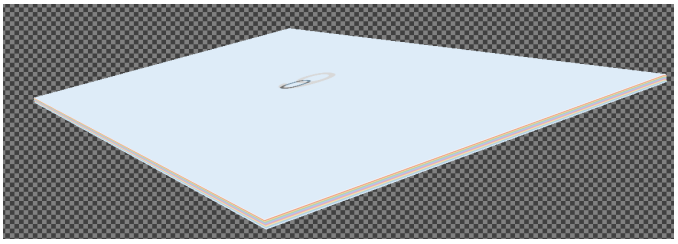

Fig. 8. The Volume Texture of the colour visualisation created from the combined image file.

Next I needed to figure out how to carve the height map depth data into these textures. I attempted this through experimenting with material assets. Inside my material asset I passed the height map as a texture parameter then applied mathematical logic to each pixel. First, I calculated the layer index then found the V offset by using the following equation.

$$V_{\text{offset}} = \text{LayerIndex} \times \left(\frac{\text{LayerHeight}}{\text{AtlasHeight}} \right)$$

Next, I passed in the texture that was used to create my volume textures as a texture parameter in order to act as a texture atlas. I then applied UV sampling to the texture atlas with the calculated V offset in an attempt to sample the rbg colour from the correct layer. Unfortunately, this texture atlas sampling was not working and the result material being created was a duplicate of the texture atlas.

C. Texture Atlas Sampling Approach

It seemed the project had reached a point where it was going to make everything easier to just implement the majority of the texture processing and visualisation logic in C++. This meant moving away from the volume texture approach entirely.

I decided use the texture atlas for this approach and thus I read the pixels inside the texture and stored them inside an array, then once I had calculated the layer index for each pixel I could apply them to the stored texture atlas' by multiplying the layer index by the height and width of the calibration rectangle. This would give me the correct rgb information for the layer and I could use this to construct a new 2D texture to output as the visualisation texture.

After implementation of these calculations I was able to utilise the functions within my blueprint code on every event tick. This created a loop that would pass the height map in and output a visualisation texture which would then update the viewport.

D. Second Viewport

In order for the solution to work seamlessly with the physical Sandbox I had to create a second viewport that would open on the connected second display, in this case a projector. I utilised Slate, Unreal Engine's GUI framework [22] to create the second viewport and update it with my visualisation textures. However, I ran into some issues making this viewport automatically open inside the connected display instead of the main display. I was trying to make this work using Unreal Engine's built in methods in order to not code with any OS specific functions but this approach was resulting in a lot of unspecific error messages.

As a result I decided to utilise the Windows API [23] as it was the only method that was working. Unfortunately this means that this solution will only work on Windows machines.

Implementing this feature resulted in the prototype finally working to a point where I could tape the new Kinect to the Sandbox and begin proper testing with the sand and the projector.

E. Noise

Once I had integrated the new Kinect onto the Sandbox and had gotten this working with my Unreal Engine solution, I discovered that the noise in the visualisation images was a big issue. Notice the difference in the two frames shown in figure 9 and figure 10 where no sand has been moved. When the system is running this was quite jarring considering the amount of textures generated and projected every second and it was taking away from the immersion of the augmented reality aspect. In order to reduce this noise I experimented with a number of different techniques.

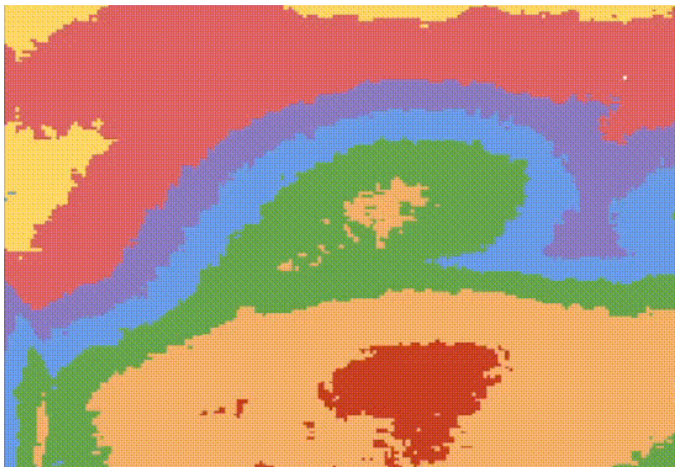

Fig. 9. A section of a visualisation frame before noise reduction techniques were applied

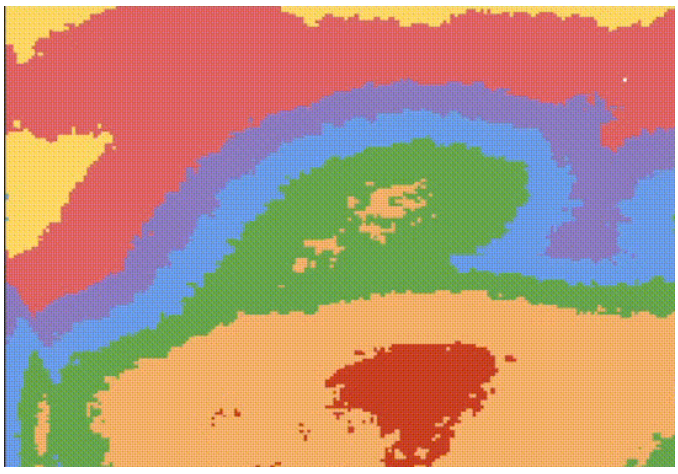

Fig. 10. A section of a different visualisation frame before noise reduction techniques were applied where the sand has not been moved

1) Average Depth Between Two Frames: The first technique I decided to explore was taking the average depth between two frames. In order to actually have two frames to work with I had to slightly adjust the design of the system. I introduced a delay that meant the texture being displayed on the sand was actually calculated from the previous depth frame. This meant I had access to the previous depth frame and the current depth frame of every event tick. After implementing a function that calculated the average depth between the two frames for every pixel and used this information to generate the visualisation texture, I found that the noise had barely been reduced at all.

2) Average Depth Between Five Frames: After the subpar results from the previous technique I decided to extend the delay and capture five frames on every event tick. This level of delay was great enough that there was a visual delay in the movement of sand and the updating of the projected visualisation but it was was still at a palatable level. Also, the previous Sandbox technology also had a slight delay due to similar noise reduction techniques so this was not an issue to worry about. The resulting noise reduction was definitely noticeable but not to a degree that would stop me from experimenting with other techniques.

3) Median Depth Between Five Frames: Sticking with the current five frame delay I decided to replace the averaging algorithm with one that instead takes the median of the depth for each pixel between the 5 frames. This resulted in a much more noticeable noise reduction.

4) Median Depth Between Ten Frames: I also briefly experimented with extending the delay to 10 frames. The level of this delay was definitely bordering on the edge of being too much but I wanted to see if the noise reduction was worth the trade off. Unfortunately, the difference between the 5 frame and 10 frame noise reduction was not noticeable for the median algorithm.

5) Average Depth Between Ten Frames: I noticed similar results with the ten frame averaging to that of the ten frame median.

6) Median Depth Between Five Frames + Gaussian Blur: As the five frame median algorithm produced the best levels of noise reduction I decided to combine this with the Gaussian blur technique. The Gaussian blur noise reduction technique involves applying a mathematical filter that smoothes and blurs an image, reducing noise and enhancing the overall image quality by averaging pixel values within a specified radius using a Gaussian function. This combination resulted in the most noticeable noise reduction amongst all the techniques experimented with thus far.

Fig. 11. A section of a visualisation frame after 5 frame median & Gaussian blur noise reduction techniques have been applied

Fig. 12. A section of a different visualisation frame after the 5 frame median & Gaussian blur noise reduction techniques have been applied where the sand has not been moved

7) Depth Spike Detection: The last technique I implemented was a basic algorithm that detected whether there was any abnormal spikes in the depth between frames. If such a spike was detected this would be replaced with an average value of the five depth frames. This technique did not result in any noticeable noise reduction and the resulting visualisation textures appeared similar to those in which no noise reduction was applied.

8) Depth Reduction Choice: As a result of this experimentation I decided to stick with the five frame median combined with Gaussian blur as it resulted in the most noticeable noise reduction.

F. Manual Calibration

As discussed in the design section, I decided to implement the first part of manual calibration through a moveable rectangle. Using Unreal Engine's widget HUD designer I added a button that, when clicked, toggles the view of this calibration. Once toggled, instead of the visualisation image being shown in the viewport, the depth map within the rectangle area is shown. There are four buttons located at the top, bottom, left,

and right side of the image. Once pressed the rectangle will be moved in that direction by 10 pixels. This allows for fine adjustment to capture only the sand area of the depth frame. There is also a save calibration button that creates a save game state which Unreal Engine has native functionality for. This means once calibrated and saved, the rectangle will remain unchanged.

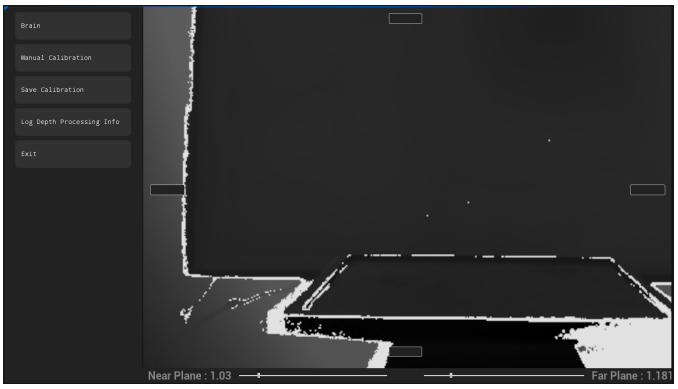

Fig. 13. A screenshot of the toggled manual calibration without moving the frame.

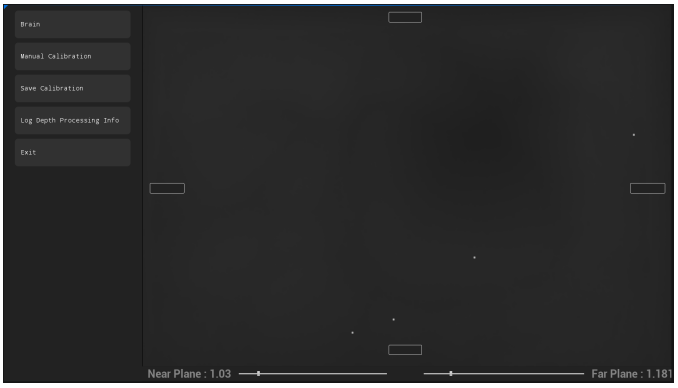

Fig. 14. A screenshot of the toggled manual calibration after the frame has been adjusted to fit within the rectangle

Similarly, to implement the second part of my manual calibration, I found Unreal Engine had widgets available for use that were ideal for my use case. I utilised two range sliders, one for each plane. These can be fine tuned to adjust the near and far plane values that are passed to my underlying depth calculations meaning that when adjusted, the visualisation texture gets updated in real time. In figure 15 these are located on the bottom of the HUD.

V. Evaluation

The output of this project is the successful migration of the existing Sandbox technology into Unreal Engine 5. The achieved solution can be described as a functional prototype of the original software, marking a significant milestone in the development process.

While the project represents a considerable success, it's important to acknowledge that there were certain features and enhancements that couldn't be fully integrated within the given timeframe.

Therefore, this categorisation as a "working prototype" is not a limitation but a proof to the potential for further improvements and refinements in the future. Despite some unimplemented features, the project demonstrates the feasibility and adaptability of Unreal Engine 5 for this purpose, and serves as a solid foundation upon which additional enhancements and feature expansions can be built.

The working prototype consists of three visualisations ported from the existing Sandbox:

Colour Visualisation

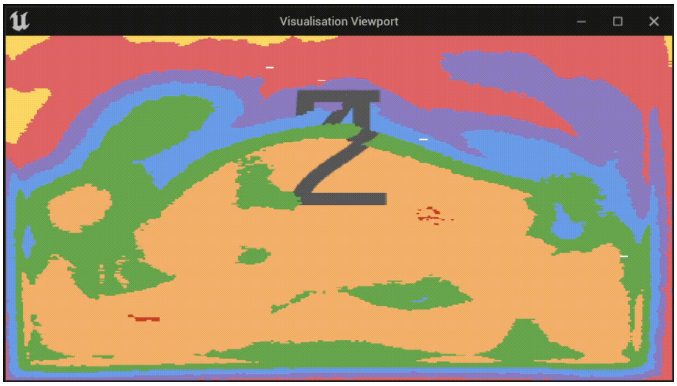

Fig. 16. A screen shot of the viewport containing a texture for the Colour Visualisation

Brain Visualisation

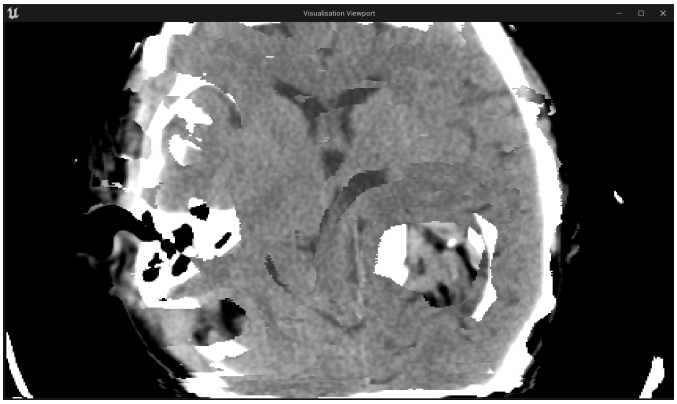

Fig. 17. A screen shot of the viewport containing a texture for the Brain CT Scan Visualisation

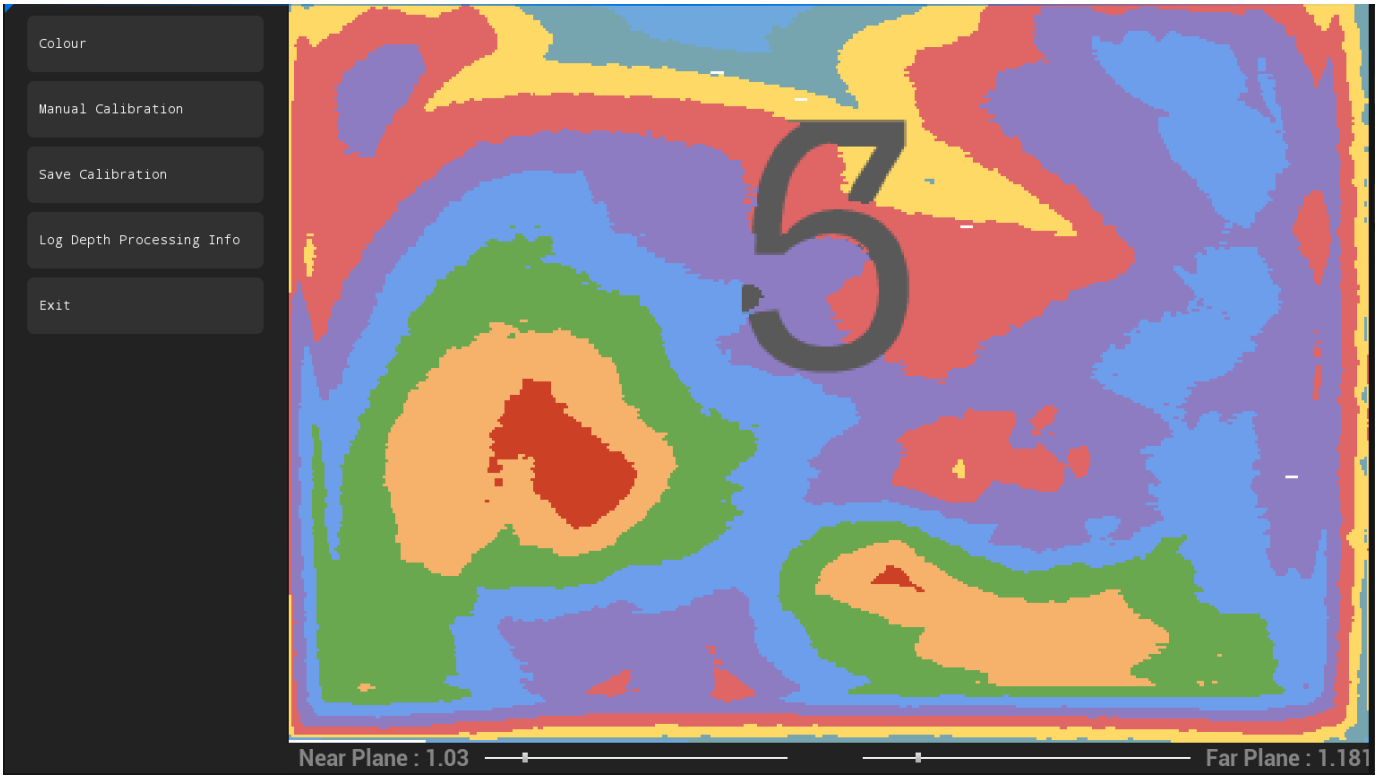

Fig. 15. A screenshot of the HUD built into the solution.

Europe Visualisation

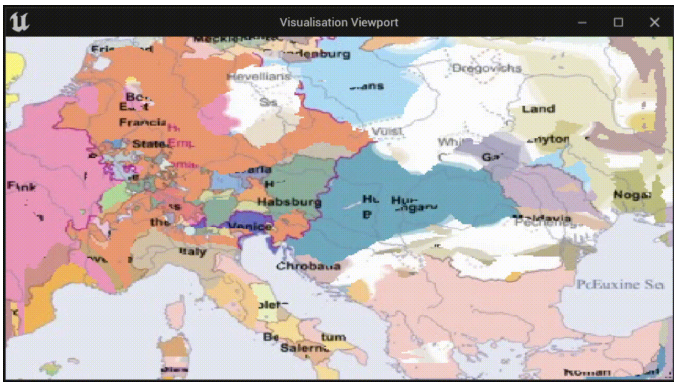

Fig. 18. A screen shot of the viewport containing a texture for the Historical Europe Visualisation

VI. Conclusions and Future Work

The future work for this project has a wide range of possibilities as the Sandbox is now working within Unreal Engine 5. There are also a number of features that I did not manage to port over from the old Sandbox software that the new solution would benefit from.

A. Old Sandbox Features

1) Automatic Calibration: Incorporating the automatic calibration system that is used in the previous Sandbox software, would not only elevate this solution from a functional prototype to a fully operational piece of software but also significantly enhance its usability and precision. This feature would streamline the setup process and make it much more user-friendly and accessible.

2) Custom Visualisations: The previous Sandbox software had a feature where you could upload a powerpoint presentation and each slide would be converted into a seperate layer. This was a simple approach that allowed for anyone to create their own visuaisations. Currently, my software does have an option in the dropdown selection for a custom visualisation it just was not implemented. This feature would be simple to add and would extend the softwares capabilities greatly.

3) Linux Adaptability: As the current solution utilises the Windows API to manipulate the second viewport, in order for the project to work on linux machines some work would need to be done. Unreal Engine 5 does support developing for Linux so this should not be too complex of a task and would align the project with the intended audience that the Magic Sand software caters to.

B. New Features

1) Gameified Visualisations: Visualisations could be created that make use of the Unreal Engine UI to combine into a game of sorts. For example, a treasure hunting game where the users dig through the sand to find a hidden treasure and the UI could help guide them and provide information about what they are digging up. This format could also be adapted to provide historical information for example if the user is digging for fossils then in-depth information about the fossil could be provided as they are digging. Gameifying the

visualisations in this way would greatly increase the appeal and engagement of the software.

2) Educational Modules: Educational Modules could be created within Unreal Engine that are specifically tailored to utilise the Sandbox to educate an audience about a certain topic. These modules could even be combined with AI interation in the form of a virtual tour guide that instructs the users to move sand into a particular area and then informs on what changes were made e.t.c.

3) Enhanced Physics Simulation: As Unreal Engine is renowned for its advanced capabilities in the realm of physics simulations, this could be utilised to create visualisations that make use of high-fidelity physics simulations to model the behavior of objects and materials in a realistic manner. The only thing to note with this is that the more advanced the simulations, the more likely the computer running the Sandbox would need better specifications which could reduce the usefulness of the software in future if there is widespread public use.

ACKNOWLEDGMENTS

I would first like to express my gratitude to Dr. Simon McCallum for his supervision and invaluable guidance throughout the year. Additionally, I extend my thanks to Dr. Craig Anslow, Benjamin Powley, Ben Sanson, and the fellow members of the Human-Computer Interaction (HCI) Research Group for the support and assistance they have generously offered.

REFERENCES

[1] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE Transactions on Information and Systems*, vol. E77-D, no. 12, pp. 1321–1329, 1994.

[2] T. Wolf, "Magic-sand," 2017, gitHub repository. [Online]. Available: https://github.com/thomwolf/Magic-Sand

[3] "openframeworks," 2023, website. [Online]. Available: https://openframeworks.cc/

[4] N. Snellgrove, "Augmented reality sandbox final report," Victoria University of Wellington, Tech. Rep., 2020. [Online]. Available: https://gitlab.ecs.vuw.ac.nz/hci/xr/ar-sandbox/-/blob/master/Reports/Final%20Report/NicholasSnellgrove_FinalReport.pdf

[5] "Unreal engine," 2023, website. [Online]. Available: https://www.unrealengine.com/

[6] M. Lourakis, "Levmar: Levenberg-marquardt nonlinear least squares algorithms in c/c++," Accessed on October 9, 2023, 2009, website. [Online]. Available: http://users.ics.forth.gr/~lourakis/levmar/

[7] "Dlib c++ library," 2023, website. [Online]. Available: http://dlib.net/

[8] D. Taing, "Ar sandbox," Victoria University of Wellington, Tech. Rep., 2021. [Online]. Available: https://gitlab.ecs.vuw.ac.nz/hci/xr/ar-sandbox/-/blob/master/Reports/Final%20Report/Final_Report.pdf

[9] "Unity," 2023, website. [Online]. Available: https://unity.com/

[10] M. Jay, "Engr489 final report," Victoria University of Wellington, Tech. Rep., 2022. [Online]. Available: https://gitlab.ecs.vuw.ac.nz/hci/health/engr489-arsandbox/-/blob/main/Documentation/FinalReport/FinalReportLatex/ENGR489-FinalReport_jaymatt.pdf

[11] Connect microsoft kinect to ubuntu. Accessed on October 10, 2023 - Used graphic from this source in the report. [Online]. Available: https://sharpeespace.blogspot.com/2011/03/connect-microsoft-kinect-to-ubuntu.html

[12] H. Sarbolandi, D. Lefloch, and A. Kolb, "Kinect range sensing: Structured-light versus time-of-flight kinect," *arXiv preprint*, 2015.

[13] Workplace posture assessment and biofeedback with kinect project. Accessed on October 10, 2023 - Used graphic from this source in the report. [Online]. Available: https://projectabstracts.com/21631/workplace-posture-assessment-and-biofeedback-with-kinect-project.html

[14] M. Tölgyessy, "Schematic of the azure kinect," https://www.researchgate.net/figure/Schematic-of-the-Azure-Kinect_fig2_348384835, Accessed: October 19, 2023.

[15] Microsoft. (Accessed: October 19, 2023) Microsoft azure kinect developer kit (dk) documentation. Microsoft Corporation. [Online]. Available: https://learn.microsoft.com/en-us/azure/kinect-dk/

[16] R. Villani, "Neo kinect plugin for unreal engine 5," Accessed on October 9, 2023, 2018, website. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/neo-kinect

[17] ——, "Neo kinect plugin for unreal engine 5 - questions," Accessed on October 9, 2023, 2022, website. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/neo-kinect/questions

[18] C. Madden, "Kinect body tracking unreal engine plugin," https://github.com/cormacmadden/Kinect-Body-Tracking-Unreal-Engine-Plugin, Accessed: October 19, 2023.

[19] G. Kurillo, E. Hemingway, M. L. Cheng, and L. Cheng, "Evaluating the accuracy of the azure kinect and kinect v2," *Sensors (Basel)*, vol. 22, p. 2469, 2022.

[20] R. Villani, "Neo kinect plugin for unreal engine 5 - questions," https://www.unrealengine.com/marketplace/en-US/product/neo-kinect/questions, 2023, accessed on October 9, 2023.

[21] ImageMagick. (2023) Imagemagick. Accessed on October 10, 2023. [Online]. Available: https://imagemagick.org/

[22] Slate - unreal engine ui framework. Epic Games, Inc. Accessed on October 12, 2023. [Online]. Available: https://docs.unrealengine.com/5.1/en-US/slate-user-interface-programming-framework-for-unreal-engine/

[23] *Windows API for Opening Second Viewport in Projector*, https://docs.microsoft.com/en-us/windows/win32/api/index, Microsoft Corporation, accessed on October 12, 2023.