Jurnal
Teknologi

Full Paper

REFLECTIVE PRISM DISPLAY USING PEPPER'S GHOST TECHNIQUE SOFTWARE TOOLKIT PLUGIN FOR UNITY 3D

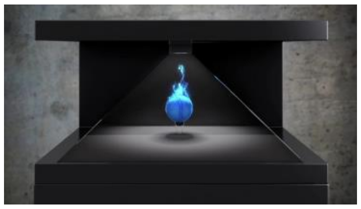
Abu Bakar Abdullah Muhammad[a], Nor Anita Fairos Ismail[a], Mohd Shahrizal Sunar[b]*

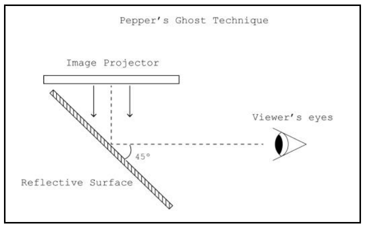
[a]Department of Software Engineering, Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia
[b]MaGICX (Media and Games Innovation Centre of Excellence), UTM-IRDA Digital Media Centre Universiti Teknologi Malaysia, Malaysia

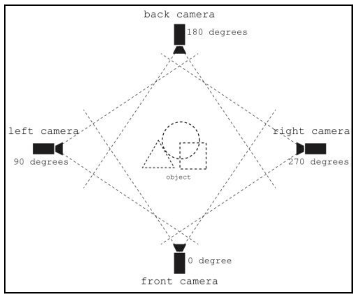
Article history
Received
26 November 2015
Received in revised form
14 January 2016
Accepted
10 October 2016

*Corresponding author
shahrizal@utm.my

Graphical abstract







Abstract

Reflective prism display is a display technology that has potentials in displaying images with fascinating effects. However, the process of creating the display is quite challenging considering the lack of specialized software and bulky hardware setup. In this project, we propose a software toolkit plugin for Unity 3D, called Prismatic, to simplify the process as an alternative over the conventional method of creating a reflective prism display. Adopting the idea from Pepper's ghost technique, a combination of four cameras facing an object were setup within Unity to produce four viewport renderings of the object, easily projected from a device as small as a smartphone to the size of widescreen TVs. This software toolkit combined with Unity offer simple and centralized control over camera, facets, and object. Prismatic has the potential in assisting apps developer in creating the display such as in showcasing models for education and business purposes.

Keywords: Pepper's ghost, Prismatic, software toolkit, viewport, rendering

1.0 INTRODUCTION

1.1 Reflective Prism Display and Pepper's Ghost Technique

Reflective prism display is one of many categories of display technology. The display, as in Figure 1, allow viewers to view images of an object from different perspectives on the prism's multiple sides.

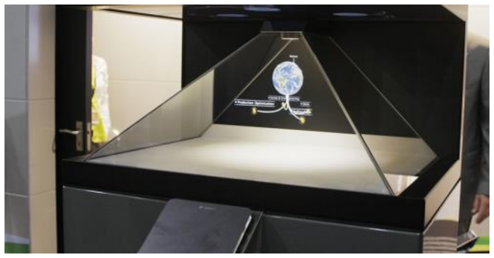


Figure 1 A reflective prism display [3]. A flat screen is placed on top of the prism as an image projector

The display is created by applying Pepper's ghost technique using a combination of two main hardwares, a prism with three or four reflective sides and an image projector [1]. The former serve as the display while the latter as the image source which projects different views of an object onto each surface of the prism (called as facet) – through different viewports (see Figure 2). The technique applied is simple as it is about reflecting an image on the reflective surface at an angle of 45 degree [1], [2] with common method done by placing the image projector on the prism top of bottom [7], [8], [11].

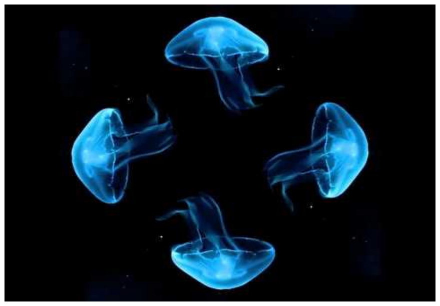


Figure 2 The four viewports with images of a jellyfish as the viewport content. This final image is ready to be projected by an image projector onto the front, back, left, and right side of the prism [16],

Pepper's ghost technique is a renowned illusion technique preceding to the year 1600's. It was described first by Giambattista della Porta [4] and was later developed by Henry Dircks [5] but did not fit with the stage setup at that time [6]. Professor John Henry Pepper then simplified the design and built the first practical version [5][7]. Figure 3 shows the use of this technique for image display.

Despite the fact that the thechnique is fairly simple, the resulting image created is undeniably interesting. They appear to be more or less transparent, but surely does not appear completely solid [8]. Images created on each facet of this prism are flat two-dimensional (2D) but seem to have an illusion as if a three-dimensional (3D) object is floating midair [9][10]. Images produced from this display are also enhanced with a false perception of depth [11].

1.2 Application and Problems

This display could provide ways of presenting physical inventions and applied in various fields including education and entertainment as seen in some showcases and performances [12], [13], [14]. Together with the display setup are a number of methods of image projection such as placing objects hidden from view and using images or videos [15].

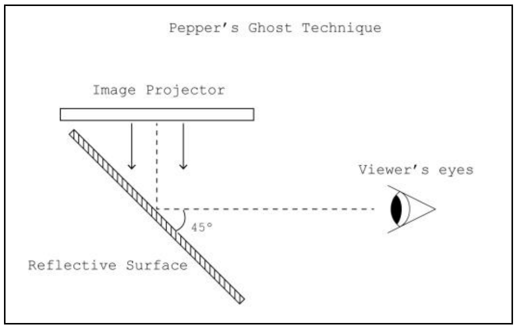


Figure 3 Pepper's ghost technique. Image is projected from the projector onto a surface tilted at 45 degree angle. Viewers are able to see the reflected image on the screen

However, creating a reflective prism display is not very easy considering the bulky hardware requirements. Setting up the hardware necessitate for quite an amount of time, not to mention the time required in addition for the software and content preparation. Moreover, content preparation (creating the four viewports) are rigid with limited control over content manipulation. Imagine recording or rendering each side of an object and pasting it onto individual viewports for the image projector which would be troublesome given that they are plain images or videos [9].

To pin down the mentioned problems, a software toolkit plugin for Unity was designed and developed. Titled as Prismatic, the goal of this work is to simplify the software and content preparation step by providing a set of tools that helps on preparing four viewports of an object to be projected onto each side of a prism - front, back, left, and right. The back side can be deactivated to suit a three-sided prism and the viewports must properly project each viewport and avoiding errors. As for the content, it could be prepared by using 3D models or 2D sprites without setting aside the use of images and videos.

With Unity, user have the freedom to make full use of its built-in functionalities such as adding animation to the object, using shader and material, manipulating by script, and providing interactivity. The toolkit in the other hand provide simple control over the viewport setup. Given these possibilities, the user may easily create their own working reflective prism display to present models and effects with minimal effort.

The image effect produced are somewhat similar to the hologram technology depicted in several sci-fi movies. Hence the terminology 'hologram' are often associated with this display. However it is worth understanding that it is not a hologram as there exist no such properties in the produced image [17].

2.0 METHODOLOGY

This section describes the methods and approaches used to tackle identified problems in the creation steps of a reflective prism display. As mentioned earlier, this work focuses on solving or providing an alternative in relation to the problems regarding software and content preparation (viewport creation) without touching on the hardware difficulties. Included in this section are the initial plannings to obtain the idea of how the toolkit works, softwares that were used, asset and hardware preparations, and finally the development steps.

2.1 Project Initialization

2.1.1 Initial Framework

An initial theoretical framework as in Figure 4 was planned ahead to obtain the overall reflective prism display creation process. Based on this framework, when a user wish to create a reflective prism display, they only need to import their own 3D object or use premade objects within Unity and integrate it with this software toolkit. This toolkit prepares a ready-to-use and easy-to-control viewport setup for the object.

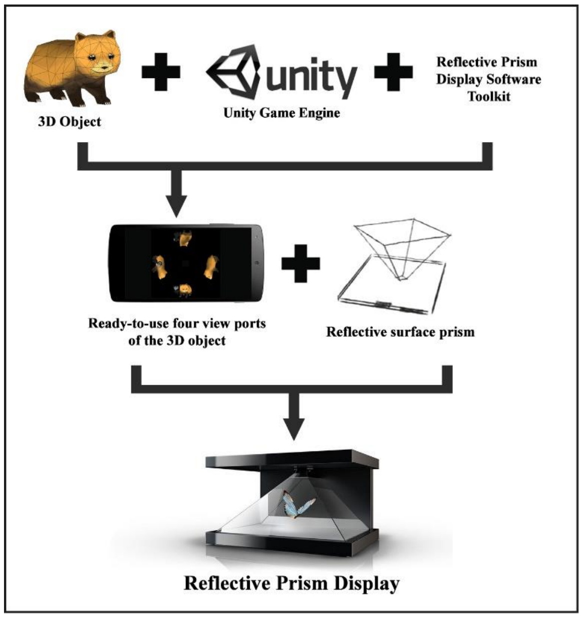

3D Object
Unity Game Engine
Reflective Prism Display Software Toolkit
Ready-to-use four view ports of the 3D object
Reflective surface prism
Reflective Prism Display

Figure 4 The initial theoretical framework

2.1.2 Achieving Four Viewport Renderings

Table 1 shows the procedure of achieving the output of four viewport renderings using this software toolkit. This include the input and processes taken in between before obtaining the output.

Table 1 The input, process, and output of a reflective prism display creation process using the software toolkit

Input	Process	Output
User import assets: • 3D model • 2D sprite	• User -_Manipulate cameras, facets, and object. • Software Toolkit - Render all four cameras onto the render texture facets and render all four viewports.	Four viewport renderings

The four viewport renderings are achievable by the user importing their own assets such as 3D models and 2D sprites into the scene and then manually setting up the objects as needed. Using Prismatic, the user can then easily manipulate the cameras and facets to fit their requirements. Alongside, Prismatic will automatically produce the viewport renderings.

2.2 Softwares Used

The softwares used during development are listed as follows:

- Windows 10 64-bit Operation System (OS)
- Unity 3D v5.0.1f
- Monodevelop
- Blender v2.7

Unity is a game engine that provides control over content creation providing a huge advantage compared to the simplistic use of images/videos for the viewport content. With Unity's built in functionalities, 3D models and 2D sprites could be used to create images and videos without limiting the fact that premade images and videos could also be imported and used within. This software toolkit plugin roots on the game engine and serve as an extension within. Monodevelop is an open source integrated development environment (IDE) shipped alongside Unity. It is used concurrently with Unity during the software toolkit development. On the other hand, Blender is an open source 3D modeling software that is free and lightweight. It is used to create models for use in this project. 3D models are exported into Unity from Blender using the .fbx extension format.

2.3 Asset Preparation

When developing the project, sample objects are needed for testing purpose. The sample objects will be used to check for correct camera setting as obtaining the correct renderings for the viewports are very important. If one or more of the viewport isn't correctly constructed, the resulting image will get twisted out of the project's needs.

It is important to use objects which provide a good way of correction-checking while avoiding objects with similar or hardly-distinguishable sides. A 3D object in the shape of a bear and a bird for example has distinguishable body parts (head, limbs, tail, etc.) and both help distinguish all four sides. These are considered good objects. Meanwhile, without a proper texture, a simple 3D object in the shape of a sphere and a cube doesn't.

Take for example, the front camera which is set to be rendered in the bottom viewport and is positioned facing the front of the 3D bear should render the front side which is the head. And the back camera should do the opposite, rendering the bear's rear. If this is not the case, then we consider that the setting of the camera or viewport is incorrect.

However, it is hard to check for mistakes if we replace this 3D object with a simple 3D cube which has the exact similar sides as distinguishing between the front, the sides and the back is hard. Hence, a good object used for testing is unavoidable.

2.4 Software Toolkit Development

2.4.1 Camera Setup

Creating a four-sided view of an object means using four cameras to render each of the object's side view and projecting it onto a display. Use of the default camera provided in the Unity Editor is sufficient for this purpose without the need of additional scripting to modify the camera rendering behavior.

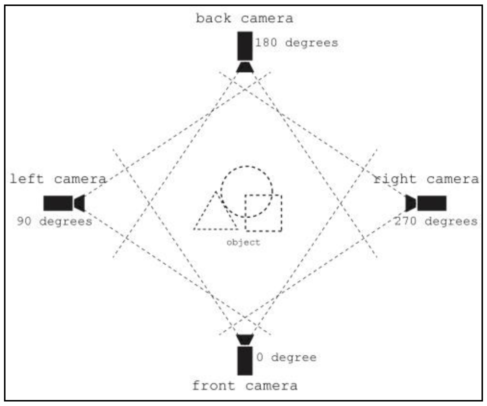


Figure 5 The four-sided camera setup. A camera is placed on each side of the object in the center and rotated accordingly to face the center

All four cameras were placed away from each other with rotation by 90 degrees on the left camera, 180 on the back, and 270 on the right (see Figure 5) on the y-axis. As for the front camera, the rotation was left at 0 degree.

However, problem arise when simply positioning the four cameras and using them without adjusting the Transformation and Viewport Rect variable in the camera's Inspector tab.

In Figure 6, a box-shaped viewport of each camera can be seen layered on top of each other. This is problematic when the object is moved as it may partially cover the object, resulting to an ugly truncated object rendering. For example, in the left viewport, the 3D bear's top part of the body may be hidden behind the bottom viewport when the object is moved away from its rendering camera. The solution to this is rather simple, that is by using a custom facet model in the shape of a prism's surface described in section 2.4.2. Another concern related to the setup is that more cameras actively used for rendering means extra computational power is required. One solution is to use simple shader materials for the objects in the center such as toon shading [18].

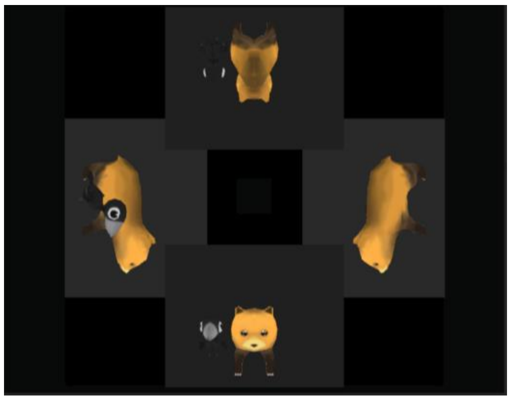


Figure 6 Simply using four cameras without adjusting camera variables create an obstructive view on the viewports in the Game window

2.4.2 Render texture facet

Practically, render texture is similar to the regular texture where it is applied to a material used to wrap around object(s). But unlike the regular texture, render textures have no image or video source assigned to it but rather, images from an active camera do. Render textures are actively updated in runtime which means any update feed from the camera will be rendered onto the render texture and are regularly updated [19]. A 3D model in the shape of a prism's facet was modeled and arranged as four flat surfaces lying on the plane y=0 to make the viewports. The render textures were then applied to the facets respective to their corresponding sides. A camera was placed facing the facets to render the final view in Unity's Game window.

2.4.3 Setup Overview

Figure 7 shows the camera setup arranged in Unity. A camera for each of the side – front, back, left, and right – were setup facing each other's center where the objects are placed. Each camera was assigned with a render texture and each of the object's side will then be rendered by the cameras onto their respective render textures, and successively, to all four facets respective to their correct perspective and position. The blue line for instance shows the front camera view rendering the front part of the object onto the bottom render texture.

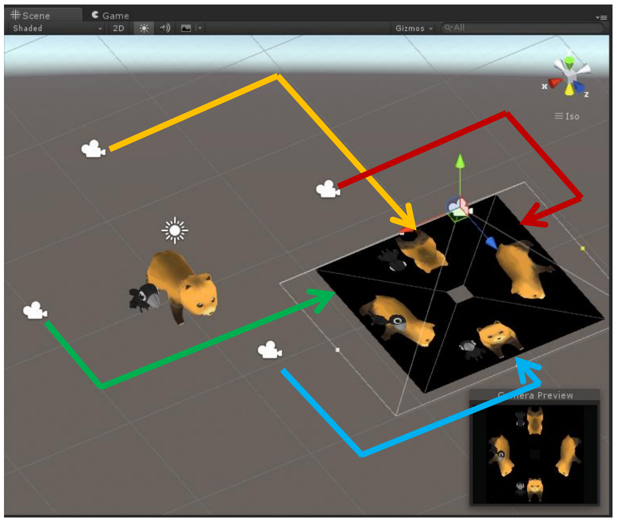


Figure 7 Overview of the setup. Four cameras were placed facing the objects in the center with each camera's rendering projected onto the associated render texture facet respective to each colored lines

2.4.4 Software Toolkit Component Programming

The software toolkit was programmed using Unity-supported language C#. This software toolkit allow a centralized and simple control over the object, cameras and facets. Centralizing the controls were achieved by creating a custom editor using Unity scripting API: `CustomEditor` which derives from Unity's `Editor` base class.

There are three main function components developed within the toolkit, each with specific functionalities:

- Focus Object – handles the functions associated with the object in the center. Assigning a focus object for the toolkit is optional.
- Camera Properties – handles the functions for the camera attributes with a unified control.
- Camera Transform – handles the position, distance, and rotation of all four cameras with a unified control.

Table 2 shows the list of functionalities within the software toolkit based on the three function components. Handling the components are `PrismCameraEditor`, `PrismCameraController`, and `PrismFacetController` classes.

The `PrismCameraController (PCC)` and class takes care of all the software toolkit's functionalities. Functions such as follow focus object, change prism mode, and enable/disable back facet are implemented within. `PrismCameraEditor (PCE)` class creates the interface between the software toolkit and user. This script overrides the inspector for `PCC` and constructs a customized setting for manipulating the cameras. While the `PrismFacetController (PFC)` is the class which hold control over the four render texture facets.

One key functionality in the toolkit is the Prism Mode control which allow users to switch between upright or upside-down prism. Upright prism is used when the prism is placed upright (with the sharp tip on top as seen with pyramids) below a bottom-facing screen projector. Upside-down on the other hand is placed upside-down below the same screen.

Note that however when the screen projector is facing upward, upside-down and upright are in the opposite order. Another thing to note is that both Prism Modes are mirrored to each other when rendered on the render texture facets.

The function is implemented by rotating the four-sided cameras by 180 degrees on each of the camera's local z-axis:

```
cam1_transform.Rotate(Vector3.forward
* angle);
```

where `cam1_transform` is the first camera game object's `Transform` property repeated through cameras 1 to 4, `Rotate()` is a function from Unity's `Transform` class, and `angle` is the angle of rotation which is 180 degrees.

The render texture facets are also flipped on the local x-axis by x=-1 to mirror the renderings:

```
facet_Front.localScale = new
Vector3(facet_Front.localScale.x * -1,
facet_Front.localScale.y, 1);
```

where `facet_Front` is the front facet game object repeated for back, right, and left, localScale is the game object's scale from its `Transform` attribute.

In some occasion of applications or requirements, the number of facets used for projection may be reduced to 3 as the back facet is not used and covered, leaving only the front and two sides. This is to reduce environmental light as too much light in the background could interfere with the image visibility.

Removing the back facet also means less camera and consequently, one less camera view to render. This could save processing power especially when a detailed and high polygon count object are used in the scene. Accordingly, a function to enable or disable the back facet were implemented. Enable and disable could be simply get and set by `GameObject` class' read-only property `activeInHierarchy` and its function `SetActive()`:

```
  cam2_camera.gameObject.SetActive(bool);
```

where `cam2_camera` is the back camera game object's Camera property.

Table 2 Software toolkit functionalities

Action	Focus Object	Assign a focus object for the Prism Camera to follow
Set	Focus Object	Assign a focus object for the Prism Camera to follow
Check/ Uncheck	Follow Object	Check to follow focus object. Accessible only when focus object is assigned
Check/ Uncheck	Smooth Follow	Check to enable smooth follow. Accessible only when follow object is checked
Click	Auto Focus	Move the Prism Camera to the position of focus object
Set	Clear Color	Set background clear color
Click	Switch Projection	Change camera projection
Drag/Set	Size/Field of View	Set orthographic size or field of view
Set	Near and Far Clipping	Set near and far clipping planes
Click	Prism Mode	Change prism mode
Check/ Uncheck	Enable Back Facet	Enable/disable back facet
Drag/Set	Camera Rotation	Set camera rotation
Set	Camera Offset	Set camera offset
Drag/Set	Camera Distance	Set camera distance

2.5 Hardware Preparation

To test the final viewport rendering, a hardware setup is as important as developing the software toolkit. There were two prism sizes used for testing. One is a small setup fitting most 4.5 to 5.5 inch smartphone screens. Another is a huge one covering a much bigger screen setup. For the bigger counterpart, a prism was carefully designed together with some sumptuous measurements to fit a widescreen, 40-inch LCD screen. Acrylic sheet with a thickness of 5mm was used as the prism material. Covering each side of the sheet is a thin layer of tint to allow a vibrant and a clear image reflection.

2.6 Software Toolkit Testing

During and after software toolkit development, some testing are required to ensure the completion of the project objective – simplifying the process of creating the viewport renderings for the display.

2.6.1 Functionality Testing

All of the constructed functionalities require testing to ensure that each one is working in perfect order. For example, functions applied to focus object shall correctly manipulate the object while functions made for the cameras should cooperatively manage every single camera. The same goes for facet-related functions.

Function testing were made in a straightforward fashion with any change of value on the parent Prism Camera object should result instantly on its child objects - the four cameras and render texture facets - in the Scene View. Checking for the correct outcome could also be done by checking individual focus object, cameras and render texture facets for any changes in the Inspector window. The changes done to the cameras and facets should correspond relative to the changes from the Prism Camera Inspector.

Table 3 Functionality testing with given actions and results

Action	Attribute	Description	Result
Set	Focus Object	Assign a focus object for the Prism Camera to follow	Focus Object set correctly
Check/ Uncheck	Follow Object	Check to follow focus object. Accessible only when focus object is assigned	Prism Camera follow Focus Object
Click	Auto Focus	Move the Prism Camera to the position of focus object	Prism Camera move to Focus Object
Set	Clear Color	Set background clear color	Set correctly
Click	Switch Projection	Change camera projection	Set correctly
Drag/Set	Size/Field of View	Set orthographic size or field of view	Set correctly
Set	Near and Far Clipping	Set near and far clipping planes	Set correctly
Click	Prism Mode	Change prism mode	Prism mode set correctly
Check/ Uncheck	Enable Back Facet	Enable/disable back facet	Enable/disabl e correctly
Set	Clear Color	Set background clear color	Set correctly
Click	Switch Projection	Change camera projection	Set correctly

2.6.2 Viewport Testing

Correct viewport rendering is the number one factor considered in this software toolkit. Without it, the final export will produce errors in the image outcome such as mirrored image, image obstruction due to window violation, and other unwanted outcomes.

Performing viewport testing require a scene export beforehand. Using Unity's built in exporter, an Android application containing the sample object (a bear and bird 3D model) was built and later exported into a device running on Android OS. A prism sized to fit smartphone devices was then used together with the exported Android application. A desktop PC build was also built to test on a larger size display.

Figure 8 shows the viewport testing performed on a mobile display. From the viewport testing, the work has been confirmed to produce correct viewport renderings without any errors in the image outcome. Image on the front side of the prism has correctly depict the object's front, so does the sides and rear.

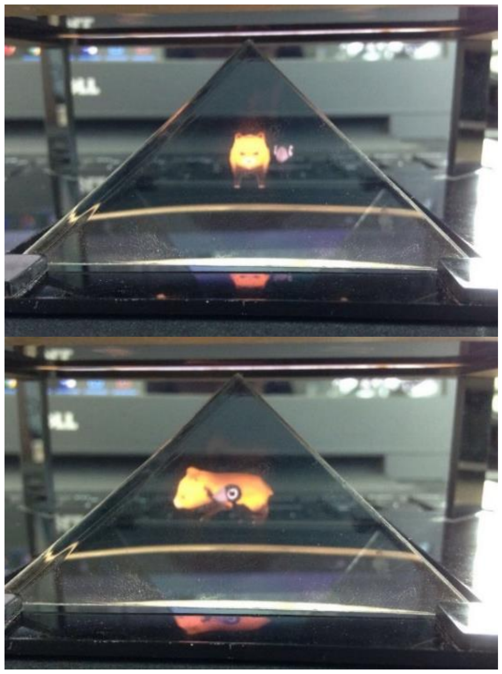


Figure 8 Viewport testing performed on a small size reflective prism display. Top: Front view of the display. Bottom: Side view of the display

3.0 RESULTS AND DISCUSSION

The final outcome of this project is the four veiwport renderings (see Figure 9). There are two deliverables within the project which will then result with this project's final outcome. One is the four-sided cameras setup, named Prism Camera, which were positioned and pointed towards their center point. Another is the four render texture facets along with its camera named as Prism View Renderer. Both prefabs carry a total of 5 cameras and a set of four facets. Both deliverables were prepared in the form of prefabs. Prefabs are a type of asset that allow storage of game object(s) complete with its component and properties. Unlike normal assets, prefabs are considered asset templates with editable components. Every single component configuration attached to an object will remain when the object is converted explicitly to a prefab [20].

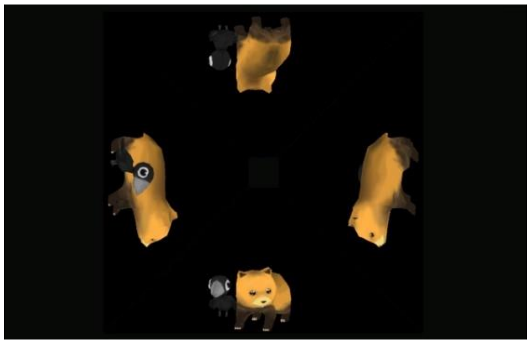


Figure 9 Final outcome, four viewport renderings ready to be used with a reflective prism to create the display.

When an instance of the prefab Prism Camera is created, it will have the `PCC` class script component pre-attached to it. User could edit the settings for the focus object and four-sided cameras in the Prism Camera Inspector window or just leave everything with the default setting.

4.0 CONCLUSION

Development of this project was made to expose one of the way of achieving an interesting image effect using a reflective prism display by the application of Pepper's ghost technique. The theory behind the technique was fairly easy to understand but yet, achieving one such display was not quite. Hence, the project was seen as a bridge that shorten the process of creating a reflective prism display. In the end, this project shall help future developers and end-users to create their own display wonder, with one less problem to consider.

Acknowledgement

Huge appreciation goes to the initial inventors, Professor John Henry Pepper and English engineer Henry Dircks for their idea. A heartfelt gratitude to those who have helped directly or indirectly to this project. Mohd Kufaisal Sidik helped build the hardware setup. Aqly Kamarudin, Ahmad Haryth Hilmy, and Muhd Faris Nazri helped assembling and disassembling the hefty setup. Without their ideas and support, this project wouldn't have been successful.
This project was funded by Media and Game Innovation Centre of Excellence (MaGIC-X), Universiti Teknologi Malaysia (UTM).

References

[1] P. A. Simonsen. 2008. Display Device For Producing Quasi-Three-Dimensional Images. Patent US 20080144175.
[2] J. Nickell. 2005. Gorilla Girl. *Secrets of the Sideshows.* University Press of Kentucky. 288.
[3] 3D Holographic Pyramid [Online]. Available: http://youlalight.com/3dpyramid.
[4] Bradbury and Evans. 1863. How we may see in a Chamber things that are not. Once A Week. 9, London. 362.
[5] J. Brooker. 2007. The Polytechnic Ghost: Pepper's Ghost, Metempsychosis And The Magic Lantern At The Royal Polytechnic Institution. 189-206.
[6] J. H. Pepper. 1890. *True History of the Ghost: And All about Metempsychosis*. Cambridge: Cambridge University Press,
[7] T. F. LaDuke and J. A. 2012. Gutierrez. Apparatus and Method for an Anamorphic Pepper's Ghost Illusion. Patent US 8262226 A.
[8] R. T. Beaver. 1997. Apparatus And Method For Creating Optical Illusion Effects. Patent US 5685625 A.
[9] de Wit, Thomas W., Gill, Mark, Freemon, Scott, & Garland, Preston. 2013. 3Design - Holographic Telecollaboration Interface.
[10] I. S. Rivera. 2012. Sefe Visor. Patent US 20120308743 A1.
[11] I. O'Connell and J. Rock. 2011. Projection Apparatus And Method For Pepper's Ghost Illusion. Patent US 7883212 B2.
[12] The Shirley Spectra. n.d. [Online]. Available: http://www.shirleyspectra.com.au/CowraN.Html.
[13] Tupac 'hologram'. 2012. [Online]. Available: http://arstechnica.com/science/2012/04/tupac-hologram-merely-pretty-cool-optical-illusion/.
[14] Fondali Specchio Olografico. 2016. [Online]. Available: http://www.peroni.com/scheda.php?id=55790.
[15] James, Ryan. 2009. 3D Holographic Projection: The Future of Advertising? [Online]. Available: http://www.activ8-3d.co.uk/aboutus/articles/3D-holographic-projection-technology.php.
[16] Youtube – Hologram Technology. 2016. [Online]. Available: https://www.youtube.com/watch?v=WFvlnYxEJGU.
[17] S. Torok and P. N. Holper. 2016. *Ghostly Images. Imagining the Future: Invisibility, Immortality and 40 Other Incredible Ideas.* Csiro Publishing. 20.
[18] Al-Rousan, R., Sunar, M. S., Kolivand, H., & Alhajhamad, H. 2015. Interactive Non-Photorealistic Rendering. *Jurnal Teknologi.* 75(4): 57-64.
[19] Unity - Manual: Render Texture. 2016. [Online]. Available: https://docs.unity3d.com/Manual/class-RenderTexture.html.
[20] Unity - Manual: Prefabs. 2016. [Online]. Available: http://docs.unity3d.com/Manual/Prefabs.html.