Unity Cluster Package – Example Project Documentation

Prof. Mário Popolin Neto, MSc.
Federal Institute of São Paulo – IFSP
Registro, São Paulo, Brazil
Email: mariopopolin@ifsp.edu.br

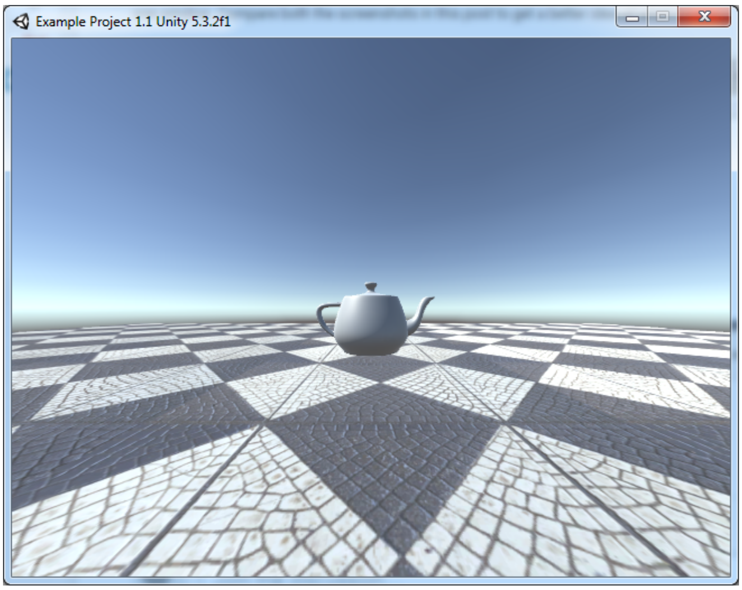
April 2016, v-1.2

1 Introduction

This document is a brief explanation of the Example Project using the Unity Cluster Package. For more specific and scientific information please check the package main paper (1), the survey used as base (2) and/or some multiprojection applications built with this custom Unity package (3).

The Example Project consists of the classic teapot model in constant rotation over a checkered textured terrain, as shown in Figure 1. The user can navigate in the virtual environment by means of a flying camera controlled through the arrow keys on the keyboard.

Figure 1 – Example project.



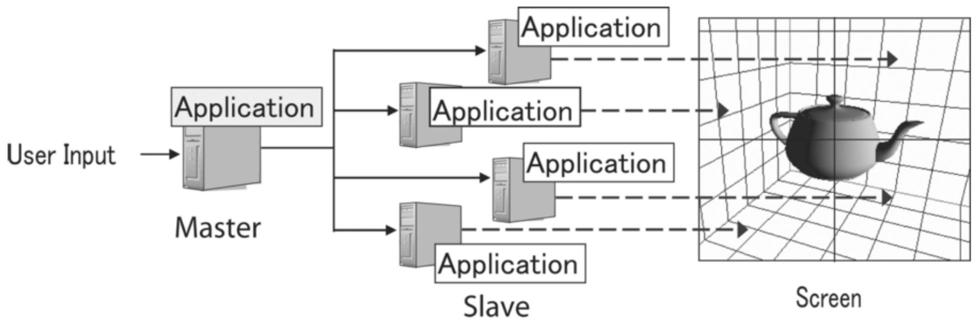
The following sections present the Example Project principal points in order to help developers create multri-projection applications using the Unity Cluster Package.

2 Contextualizing

In the Virtual Reality field, multi-projection applications are virtual environments displayed on multiple screens providing a coherent, seamless and contiguous view.

PC cluster based on rendering models is the most frequently used hardware approach regarding multi-projection of virtual environments created by game engines (2). In the Master-Slave rendering model, all nodes (master and slaves) perform the same application with different virtual camera configurations. The master node application synchronizes the slave node applications and treats the user input (4). Figure 2 shows the Master-Slave rendering model.

Figure 2 – The Master-Slave rendering model (5).



3 State Changes in Objects

Since Unity Cluster Package implements the Master-Slave rendering model, only the master node application performs state changes in objects. These objects must use the NetworkView component from the Unity network module for state synchronization among the slave node applications. As an example, Figure 3 shows the rotation script (TeapotRotate.cs) attached to the teapot model.

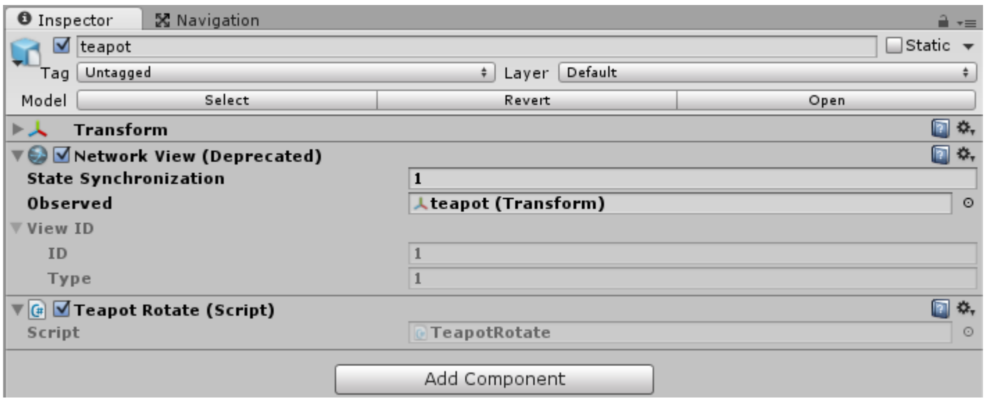
Figure 3 – TeapotRotate.cs - rotation script.

```
1  using UnityEngine;
2  using UnityClusterPackage;
3
4  public class TeapotRotate : MonoBehaviour {
5
6      // Use this for initialization
7      void Start() {
8          if ( NodeInformation.type.Equals("slave") ) {
9              enabled = false;
10         }
11     }
12
13     // Update is called once per frame
14     void Update () {
15         transform.Rotate(Vector3.up * Time.deltaTime * 30.0f);
16     }
17 }
```

Note that, if node type is "slave" the script will be set enabled false (lines 8 and 9), not performing rotations. However, when node type is "master" the script will apply the rotations (line 15). In order to synchronize the rotations performed by the master node application, the teapot model uses the NetworkView component, as shown in Figure 4.

In the same way, the flying camera script (FlyCamera.cs) is only performed by the master node application, meanwhile the *Multi Projection Camera* from the Unity Cluster Package uses the NetworkView component for synchronization purposes.

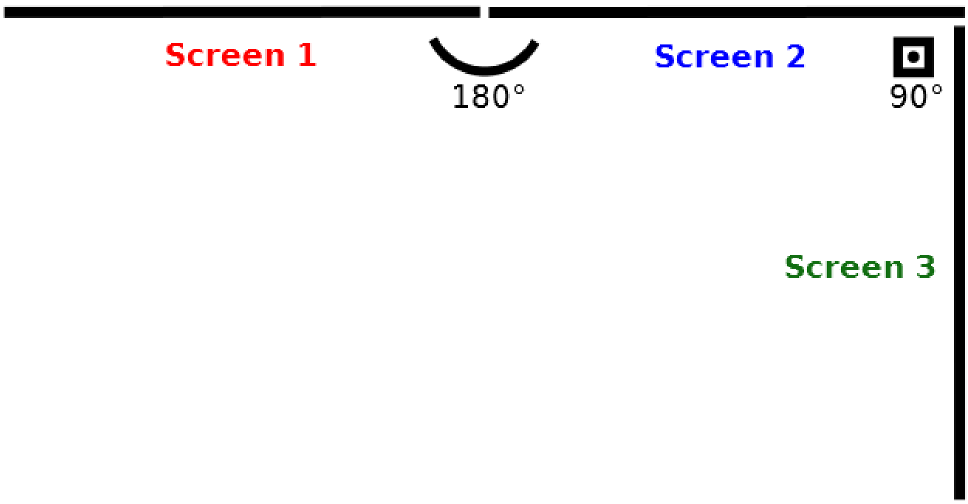
Figure 4 – Teapot NetworkView.



4 Master and Slave Applications Configuration

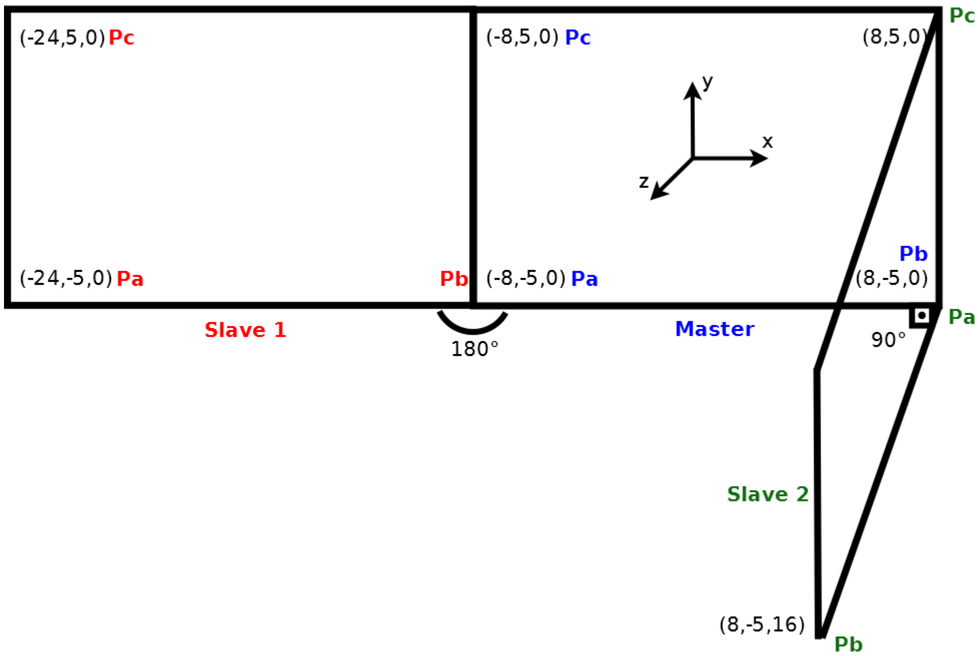
The Unity Cluster Package supports different multi-projection systems regarding the size, aspect ratio, position and orientation of the screens. Figure 5 shows the 3-screens system used as example.

Figure 5 – 3-Screens system - top view.



Applications are assigned to the screen according to the three points (Pa, Pb, and Pc) from the screen corners (Pa at the lower left, Pb at the lower right, and Pc at the upper left) in a coordinate system with the origin in the center of the front screen. Figure 6 shows the Pa, Pb and Pc points for each screen.

Figure 6 – Pa, Pb and Pc points.



The Unity Cluster Package is based on the Unity network support for the Client-Server communication model. The master node application is an Unity server. The slave node applications are Unity clients connected to the Unity server in the master node application.

Applications obtain the cluster node specification through an XML configuration file (node-config.xml) in the StreamingAssets folder. The master node application is assigned to one of the three screens as shown in Figure 6, which is not mandatory. Figures 7, 8 and 9 show the Master, Slave 1 and Slave 2 configuration files. Figure 10 shows the Master, Slave 1 and Slave 2 applications performed side by side on the same computer.

Figure 7 – Master configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<node type="master" name="Master Node" id="0" nodes="3">
    <server ip="192.168.1.5" port="25000"/>
    <screen stereo="false" eye="">
        <pa x="-8" y="-5" z="0" />
        <pb x="8"  y="-5" z="0" />
        <pc x="-8" y="5"  z="0" />
        <pe x="0"  y="0"  z="5" />
    </screen>
</node>
```

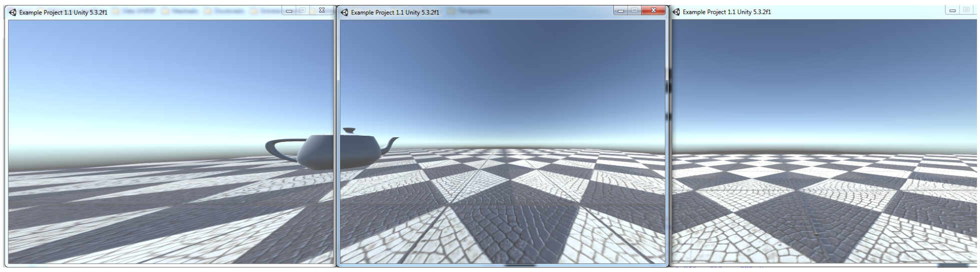
Figure 8 – Slave 1 configuration file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<node type="slave" name="Slave Node 1" id="1" nodes="3">
    <server ip="192.168.1.5" port="25000"/>
    <screen stereo="false" eye="">
        <pa x="-24" y="-5" z="0" />
        <pb x="-8"  y="-5" z="0" />
        <pc x="-24" y="5"  z="0" />
        <pe x="0"   y="0"  z="5" />
    </screen>
</node>
```

Figure 9 – Slave 2 configuration file.

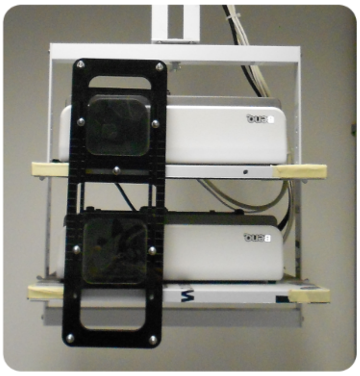
```xml
<?xml version="1.0" encoding="UTF-8"?>
<node type="slave" name="Slave Node 2" id="2" nodes="3">
    <server ip="192.168.1.5" port="25000"/>
    <screen stereo="false" eye="">
        <pa x="8" y="-5" z="0" />
        <pb x="8" y="-5" z="16" />
        <pc x="8" y="5"  z="0" />
        <pe x="0" y="0"  z="5" />
    </screen>
</node>
```

Figure 10 – The Master, Slave 1 and Slave 2 applications performed side by side.



In case of passive stereoscopic, there might be two applications that address the projection for the same screen but for different eyes using polarized systems, such as the polarized projectors in Figure 11. It is possible to set up the configuration file for passive stereoscopic projection, where the stereo attribute must receive “true” and the eye attribute “right” or “left”.

Figure 11 – Polarized projectors.



5 Mottion Paralax using the Mouse Device

The *Multi Projection Camera* from the Unity Cluster Package arranges the scene object *UserHead*. By moving this object based on a head-tracker device the motion parallax is achieve, providing a better perception in the virtual environment. The script ParallaxMouse.cs (Figure 12) moves the object *UserHead* according to the mouse device once the line 21 is not commented.

Since the *UserHead* x axis is rotated, the mouse y axis is the *UserHead* z axis. Note that, this script is only performed by the master node application and the object *UserHead* uses the NetworkView component for synchronization purposes.

Figure 12 – ParallaxMouse.cs - Mottion Paralax x Mouse Device.

```
1  using UnityEngine;
2  using UnityClusterPackage;
3
4  public class ParallaxMouse : MonoBehaviour {
5
6      private float x, y;
7
8      // Use this for initialization
9      void Start() {
10         if ( NodeInformation.type.Equals("slave") )
11         {
12             enabled = false;
13         }
14     }
15
16     // Update is called once per frame
17     void Update () {
18         x = ( Input.mousePosition.x - Screen.width/2 ) * 16/Screen.width;
19         y = ( Input.mousePosition.y - Screen.height/2 ) * 10/Screen.height;
20
21         //transform.localPosition = new Vector3(x, 5, y);
22     }
23 }
```

If you need any references or have any questions, do not hesitate to contact me by email: mariopopolin@ifsp.edu.br

References

1 POPOLIN NETO, M. et al. Computational science and its applications – iccsa 2015: 15th international conference, banff, ab, canada, june 22-25, 2015, proceedings, part v. In: ______. Cham: Springer International Publishing, 2015. cap. Unity Cluster Package – Dragging and Dropping Components for Multi-projection Virtual Reality Applications Based on PC Clusters, p. 261–272. ISBN 978-3-319-21413-9. Disponível em: <http://dx.doi.org/10.1007/978-3-319-21413-9_19>.

2 POPOLIN NETO, M.; BREGA, J. R. F. A survey of solutions for game engines in the development of immersive applications for multi-projection systems as base for a generic solution design. In: *Virtual and Augmented Reality (SVR), 2015 XVII Symposium on*. [S.l.: s.n.], 2015. p. 61–70.

3 POPOLIN NETO, M. et al. An immersive and interactive visualization system by integrating distinct platforms. In: *Information Visualisation (iV), 2015 19th International Conference on*. [S.l.: s.n.], 2015. p. 403–410. ISSN 1550-6037.

4 STAADT, O. G. et al. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In: *Proceedings of the Workshop on Virtual Environments 2003*. New York, NY, USA: ACM, 2003. (EGVE '03), p. 261–270. ISBN 1-58113-686-2. Disponível em: <http://doi.acm.org/10.1145/769953.769984>.

5 HASHIMOTO, N.; ISHIDA, Y.; SATO, M. A self-distributing software environment for immersive multiprojector displays. *Systems and Computers in Japan*, v. 38, n. 2, p. 1–9, 2007.