# OpenSimulator and Unity as a Shared Development Environment

Fumikazu Iseki
Tokyo Univ. of Info. Sci., Dept. of Informatics, Japan

Austin Tate
AIAI, School of Informatics, University of Edinburgh, United Kingdom

Daichi Mizumaki
Tokyo Univ. of Info. Sci., Dept. of Informatics, Japan

Kohei Suzuki
The Daiichi Information Systems Co., Ltd., Japan

## Abstract

We developed a conversion system to load OpenSimulator Archive (.oar) files into Unity. This system allows the user to load world data jointly created in OpenSimulator into Unity on a region-by-region basis, and to ultimately output the data from Unity in a variety of formats. These features allow us to treat the pairing of OpenSimulator and Unity as a shared development environment, which simplifies the low-cost creation of 3D spatial visualization data.

## 1. Introduction.

Several approaches exist to construct three-dimensional (3D) worlds in a computer system at low cost. Representative methods include using the systems Unity [1] and Unreal [2], which are so-called game engines.

Alternatively, one can use so-called metaverse systems. At present, the widest used metaverse-type examples of 3D virtual space (virtual world) platforms are Second Life™ [3], offered by Linden Lab, and OpenSimulator (also known as OpenSim) [4], which is an open-source platform largely compatible with Second Life.

These game engines and metaverse platforms each have advantages and limitations. For example, Second Life and OpenSimulator allow users to construct 3D spaces through collaborative efforts over a network using in-world real time content editing tools, while Unity and Unreal allow the conversion of created 3D spatial data for use in webpages and applications so that general users can easily reference it.

Adopting the game engine Unity and the metaverse OpenSimulator, the present study proposes a construction environment for 3D spaces that takes advantage of the two systems' respective properties.

## 2. Unity and OpenSimulator

### 2.1 Unity

Offered by Unity Technologies, Unity is an integrated game development environment, which is a kind of game engine. Two editions exist as of the time of writing. These are the Professional Edition and the Personal Edition, the latter of which can be used for free despite offering almost all of the major functions of the former.

Several characteristics of this software are notable, but we focus our attention on two functions: direct loading of 3D objects using Collaborative Design Activity (Collada) data (.dae files; a 3D-data-interchange file format) [5], and the output of complete 3D creations as a webpage or application. Unity does operates a shared data space for development, but it is difficult to use due to the program's implementation of this feature.

### 2.2 OpenSimulator

OpenSimulator is an open-source virtual world or metaverse system developed by an open-source community coordinated through the non-profit Overte Foundation; it is the most widespread open-source metaverse system. Originally, OpenSimulator was based on Second Life's protocols though extensions have been made to reflect its use in educational and other environments. However, importantly, OpenSimulator can use variants of the Second Life Viewer (display client) program, which itself is released under a flexible open-source license. There is also an optional OpenSimulator facility called the "Hypergrid" which allows the connection of separately provided or managed OpenSimulator grids. According to the metaverse-related news site Hypergrid Business, over 70,000 standard Second-Life sized region equivalents (256m X 256m) exist worldwide across the various separate OpenSimulator "grids" as of January 2016 [6].

Considering OpenSimulator as a system to construct 3D spaces, its most characteristic feature is that users can jointly perform development in real time while viewing sharing their creations over a network. Essentially, multiple users can proceed with development in a simultaneous and progressive manner while confirming the properties and positions of objects and landscapes in advance.

Its weaknesses, on the other hand, include the construction of the system itself and complicated access methods for normal (non-developer) users. General users must take complex actions in order to see 3D spaces constructed in OpenSimulator (account and avatar creation through a web interface (F.Iseki and M.W.Kim, 2011 [7]), viewer installation and configuration, etc.), and so beginners often become discouraged midway. Many ask why they cannot simply enter and view the 3D worlds and interact with others in them via a web browser, as this is a common access method nowadays.

Regarding the construction of 3D worlds on the OpenSimulator platform system itself, options are available to add 3D space in the form of one or more regions onto grids run by others, such as using the open grid OSGrid [8]. This allows users to connect regions for free, as well as allowing avatars to be created and used for free. Such grids usually run on community donations. This ability to add regions onto grids managed by others is useful and continues to increase for users who cannot construct or manage everything by themselves. Often basic building resources and 3D models are provided in asset stores on such grids, and many provide these assets freely to other builders and users.

### 2.3 Shared Development System

Unity and OpenSimulator each have advantages and disadvantages as systems for 3D spatial construction. If one could load data jointly created in OpenSimulator into Unity and then output it as a webpage or application, it would be possible to construct a development environment that leverages the strengths of both systems.

To achieve this, we have developed an open-source OpenSimulator-to-Unity data conversion system. The developed system allows us to treat OpenSimulator and Unity as a single shared-development, visualization environment.

## 3. Preceding Developments

### 3.1 Case by Tipodean Technologies

In 2011, a virtual world converter was created by Tipodean [9] to take saved OpenSimulator regions as OpenSimulator Archive (OAR) files and convert the content to Collada mesh for import to Unity. It was used to create a number of demonstrations using academic virtual world regions from Rutgers University, the US National Oceanic and Atmospheric Administration (NOAA) and the Virtual University of Edinburgh's Open Virtual Collaboration Environment (OpenVCE) open source available region (see Figure 1). The Tipodean converter did not attempt to convert all content but was able to convert the main primitives and mesh used in OpenSimulator builds, and was able to bring across the core visual appearance of a region, so that further development could be undertaken in the Unity development

environment and editor.

Note that the Tipodean conversion service became unavailable in 2015 but examples are available at http://converter.tipodean.com and http://converter.tipodean.com/unity3d/

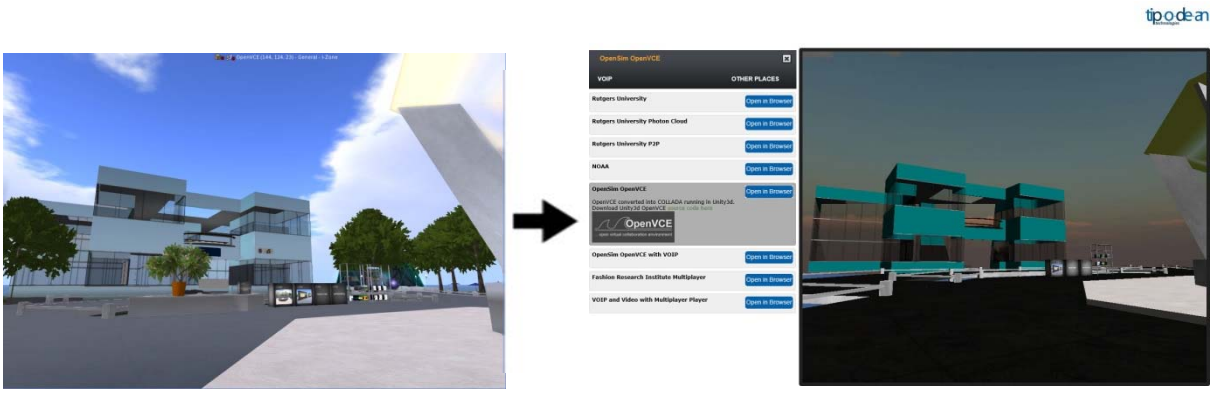


Figure 1: OpenVCE Region in OpenSimulator and via Tipodean Converter to Unity

### 3.2 Export of 3D Mesh via a Compatible Viewer

Singularity Viewer [10] is an open source virtual world viewer (client software) that can be used with OpenSimulator and Second Life. The Singularity Viewer supports the export of Collada format and other 3D formats on an object-by-object basis, but this does not mean it supports the output of all parameters.

The unit of output is the object, and the processing does not take into account system-side considerations of importing the output data (Collada compatibility) to other platforms.

## 4. Conversion System

### 4.1 Outline of Conversion System

Figure 2 shows an outline of our developed conversion system. Region data, the management unit in OpenSimulator, can be output as an .oar (OpenSimulator Archive) file [11]. OAR Converter, a tool we previously developed [12] [13], outputs this OAR file on an object-by-object basis as Collada data (.dae). Conversion also proceeds such that every texture file used for the object can be used in Unity.

Converted .dae file(s) and texture file(s) are then loaded into Unity via "SelectOARShader", a Unity editor extension created for use with OAR Converter. "SelectOARShader" is used to make decisions about which shader to use and about object attributes in Unity on the basis of the filenames (and pathnames) of data.

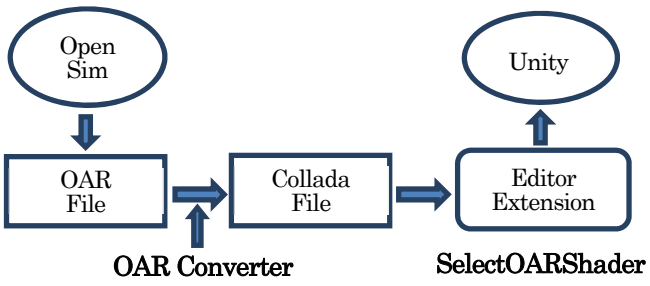


Figure 2: Conversion System Overview

### 4.2 Details of Object Conversion

All region information from OpenSimulator is included in the .oar file. OAR Converter retrieves vertex data separately for each Prim [15], Sculpted Prim [16], and Linden Mesh [17] object. After searching for duplicate vertices in the resulting octrees, the program then generates triangular polygons and converts them to Collada data. The step of checking for duplicate vertices is not essential, but is included because data could otherwise grow very large in size, depending on the objects. For the terrain or ground, we generate triangular polygons from elevation data and convert to Collada mesh data.

When these data are imported separately into Unity using "SelectOARShader" normal solid objects are loaded with collider attributes. For "phantom" objects, on the other hand, collision detection is not performed, and so the conversion process is similar to that of normal objects, except that the results are output to a dedicated ("Phantoms") folder. The processing of this data without collider attributes is performed separately when the data in this folder are loaded into Unity.

The details of the tree generation algorithm in OpenSimulator are unclear for Linden Labs style trees and plants. They are handled specially in Second Life compatible viewers and are not normal objects. Thus, Linden trees and plants are represented in combination with a special Primitive representing the type of tree of plant, without using the conversion algorithm. Regarding avatars and Non-Player Characters (NPCs), drawing avatars is a function of the viewer, which OAR Converter does not yet support (although integration with the viewer is under investigation).

Table 1 shows a list of conversion functions supported by OAR Converter. OpenSimulator repeatedly processes small-size image data dynamically to display ground textures, but loading ground textures into Unity requires there to be only one image. Attempting to maintain high resolution causes data to grow extremely large in size, and so we opted to reduce resolution in consideration of OAR Converter and Unity performance.

In addition, OAR Converter ignores flexibility-related parameters from Flexible Prim objects; they are converted as normal Prim objects.

The implementation and improvement of conversion functions not supported or restricted above constitute topics for future research.

Table 1: OAR Converter Conversion Functions

| Target | Conversion | Notes |
|---|---|---|
| Prim | Supported | |
| Sculpted Prim | Supported | |
| Mesh | Supported | |
| Tree/Grass | Limited Support | Linden Tree generation algorithm is not used |
| Flexible Prim | Limited Support | Flexible parameters are not supported |
| Terrain | Limited Support | Low-resolution textures |
| Avatar/NPC | Not Supported | |
| Water | Not Supported | |
| Script | Not Supported | |

### 4.3 Details of Texture Conversion

OpenSimulator saves texture data internally in JPEG 2000 (.jp2, .jpx) format. But Unity cannot handle JPEG 2000 format. Thus, OAR

Converter performs the task of converting texture files in JPEG 2000 format to common image formats at the same time as it converts object data.

In consideration of the properties of Sculpted Prim objects (i.e., their use of pixel data from images as 3D coordinate data), OAR Converter converts them via lossless compression to .tga files, which have a simple format structure.

We use OpenJPEG for JPEG 2000 conversion processing [17]. OpenJPEG has two versions, Version 1 and Version 2, which both have compatibility issues for a proportion of processing tasks. In other words, some images can be processed by Version 1 but not by Version 2, and vice versa. In its default state, therefore, OAR Converter is designed to perform processing internally using Version 1, and if this results in un-processable images it uses Version 2 (if installed) via external commands.

### 4.4 Output of .dae Files

Collada is a general-purpose 3D data conversion format, and so theoretically OAR Converter can be run without any special consideration regarding Unity as a target. However, Unity does not support all Collada functions (e.g. it cannot import all parameters). Furthermore, depending on Unity specifications, there are conditions under which, for example, material color information fails to import correctly into Unity unless the Collada protocol is partially violated.

Specifically, according to the Collada digital asset schema [18], multiple <diffuse> elements cannot exist within a <phong> element. Therefore, if Collada data is created according to the digital asset schema, a <diffuse> element's child <texture> and <color> elements must be described within the same <diffuse> parent element. However, Unity ignores <color> element parameters if they are described in this way. To make Unity correctly load <color> element parameters, separate <diffuse> elements corresponding respectively to <texture> and <color> elements must be independently prepared and described. That is, the system needs to be able to use multiple <diffuse> elements inside <phong> elements, but this violates the Collada protocol.

Considering these conditions, the default behavior of OAR Converter is to output a .dae file (in which the Collada protocol is partially violated) for use in Unity. In addition, filename extensions are added to texture filenames in anticipation of their loading into Unity (see below).

### 4.5 Import to Unity with Material Parameters

When importing external data into Unity, texture filenames become material filenames by default. Materials contain all kinds of information, including not only the texture itself, but also its geometric parameters, color, transparency, and reflectance. However, Unity prioritizes only the texture parameters last loaded when texture files have the same filename, and materials that were originally different get processed as a single material category.

In addition, Unity cannot read all parameters from .dae files. For example, it ignores information such as reflectance, glow, and brightness contained in .dae files.

Further, OpenSimulator has only one kind of "shader" that it can apply to objects (and materials). In Unity, on the other hand, the user can select various shaders for each material. Thus, objects

created in OpenSimulator cannot be accurately represented in Unity unless an appropriate shader is selected. The user could manually configure the shader for each object, but this would constitute a fatal flaw in the conversion system when one considers the time and effort involved.

Summarizing the above points yields the following two issues.

1. If material parameters differ, even for the same texture file (name), the user must force Unity to recognize them as different materials.
2. The program must be able to, as well as possible, automatically select an appropriate shader based on the material parameters.

To resolve the first problem, major material parameters are converted to a text string and appended to the texture filename when texture files are converted by OAR Converter for later use when they are loaded into Unity. Stringified parameters in OpenSimulator are as below.

1. Color (Red), 1 Byte
2. Color (Green), 1 Byte
3. Color (Blue), 1 Byte
4. Transparent, 1 Byte
5. Alpha Cutoff, 1 Byte
6. Shininess (Reflectance), 1 Byte
7. Glow, 1 Byte
8. Bright, 1 Byte
9. Light (Luminescence), 1Byte
10. Texture Shift in U Direction, 2 Bytes
11. Texture Shift in V Direction, 2 Bytes
12. Texture Scale in U Direction, 2 Bytes
13. Texture Scale in V Direction, 2 Bytes
14. Texture Rotation, 2 Bytes
15. Object Classification, 1 Byte

Object Classification indicates the distinction between normal objects (O), trees (T) and grass (G), and ground (E). Four spare bytes (for future extensions) are added to these 20-byte parameters, and all 24 bytes are converted to a Base64 string (yielding 32 bytes after conversion). The character '/' is typically allowed to appear in Base64 strings, but it cannot be used in filenames; thus, '/' is replaced with '$' when it appears. In addition, '_' is used as a separator character when appending the parameter string, in order to distinguish the new filename from the original filename. Figure 3 shows an example filename (a universally unique identifier ) with an appended parameter string.

cc98df7f-d21c-45b3-aca0-4ac36be289c5_AAAAGQAAAAAAAAAAAAAAACjAQEAAABP.tga

Figure 3: Example Filename with Parameters Appended

Accordingly, materials with different parameters will be converted to files with different names, even if their texture data are the same. Hence, materials will be recognized as different materials (material names) when loaded into Unity.

To resolve the second issue, we utilize an editor extension in Unity. When loading external data

into Unity, Unity can load the data once it has been processed by the editor extension. The editor extension lets users see shader designations and material names (i.e., filenames of texture files). Therefore, the original parameters can be reproduced by removing the Base64 string from the material name, and an appropriate shader can be selected according to the values of the parameters.

We created the Unity editor extension "SelectOARShader" in order to realize these functions and to add collider attributes to objects. This will obviously not always result in the best shader being chosen, but a better shader will be chosen at least, the result of which is expected to greatly reduce efforts spent on later manual configuration.

## 5. Experience with Sample Conversions

### 5.1 Example of Conversion – Tokyo University of Information Sciences

Figure 4 shows the original OpenSimulator snapshot from a model of Tokyo University of Information Sciences and the resulting snapshot after conversion and loading into Unity.

The number of solid and phantom objects loaded into Unity (i.e., .dae file count) was 4,741, and the number of texture data (material data) files was 4,867. The conversion time required was about 9 minutes (PC; CPU: AMD Phenom IIX4 965 @ 3.40 GHz; Memory: 16 GB). Loading into Unity required an additional 45 minutes.

In this test case, only the shader for two materials (data for a fraction of tree leaves) was modified manually after data was loaded into Unity. In addition, problematic or erroneous conversion of object shapes did not occur as far as can be determined from visual inspection.

Figure 4: TUIS region in OpenSimulator and as Converted via OAR Converter to Unity

### 5.2 Example of Conversion – Virtual University of Edinburgh

A further test case was employed during the development of the OAR Converter to provide feedback during its development and refinement. The Virtual University of Edinburgh (Vue) OpenSim grid's Open Virtual Collaboration Environment (OpenVCE) region, which was created by the OpenVCE community and Clever Zebra and is available open source (Tate et al., 2014)[19][20]. The OpenVCE region OAR was 40.8MB with 4,104 objects on the original OpenSimulator region. The OAR file was converted via the OAR Converter (see Figure 5).

The number of solid and phantom objects loaded into Unity (i.e. .dae file count) was 404 compound objects (made up of the original 4,104 sub-elements) and the number of texture data (material data) files was 896. The conversion time required was about 2 minutes (PC; CPU: Intel i7-3770 @ 3.40GHz; Memory: 16MB). Loading into Unity required an additional 6 minutes.

During development of the OAR Converter we were able to identify a number of issues through the trial conversion of the OpenVCE region as described at [13]. All these were addressed in the version subsequently released.

The OAR Conversion process does not bring across any active scripting, particle systems (for smoke, fire, etc.) or special light sources. Where important for the converted scene, these need to be added back into the resulting Unity project.


Figure 5: OpenVCE region in OpenSimulator and as Converted via OAR Converter to Unity

## 6. Conclusion

In this report, we introduced a conversion system developed for importing OpenSimulator content into Unity. We showed that using this system allows users to easily convert data created in OpenSimulator for use in Unity.

This setup lets the two platforms complement each other's limitations; users can interactively create and edit 3D data jointly in OpenSimulator and convert it to a format easily displayable in Unity, allowing developers to treat OpenSimulator and Unity as a single shared development environment.

The OAR Converter is published open source at [12].

## Reference

[1] http://unity3d.com/
[2] https://www.unrealengine.com/blog/
[3] http://secondlife.com/
[4] http://opensimulator.org/
[5] https://www.khronos.org/collada/
[6] http://www.hypergridbusiness.com/2016/01/grid-traffic-slows-over-holidays/
[7] F.Iseki and M.W.Kim, Development of Web Interface for OpenSim, Journal of Tokyo University of Information Sciences (Tokyo University of Information Sciences), Vol.14, No.2, pp.36-43

(March 2011). In Japanese.
[8] http://www.osgrid.org/
[9] http://www.tipodean.com/
[10] http://www.singularityviewer.org/
[11] http://opensimulator.org/wiki/OAR_Format_0.8
[12] http://www.nsl.tuis.ac.jp/xoops/modules/xpwiki/?OAR%20Converter
[13] http://blog.inf.ed.ac.uk/atate/2015/08/30/opensim-oar-convert-to-unity-scene/
[14] http://wiki.secondlife.com/wiki/Prim
[15] http://wiki.secondlife.com/wiki/Sculpted_Prims:_Technical_Explanation
[16] http://wiki.secondlife.com/wiki/Mesh/Mesh_Asset_Format
[17] http://www.openjpeg.org/
[18] https://www.khronos.org/files/collada_spec_1_5.pdf
[19] Tate, A., Hansberger, J.T., Potter, S. and Wickler, G. (2014) Virtual Collaboration Spaces: Bringing Presence to Distributed Collaboration, Journal of Virtual Worlds Research, Assembled Issue 2014, Volume 7, Number 2, May 2014.
[20] http://openvce.net/downloads

All of the above URLs have been confirmed as active as at 16 April 2016.