Mathematical Problems of Computer Science 58, 84–90, 2022.
doi: 10.51408/1963-0095

UDC 004.62:004.946

Data Processing and Persistence in Virtual Reality Systems

Arman A. Hovhannisyan

National Polytechnic University of Armenia
e-mail: aahovhannisyan1@gmail.com

Abstract

Data processing and persistence are key aspects of developing a Virtual Reality system. In this paper, an improvement is offered to the distance calculation algorithm of the Unity Engine. Additionally, data persistence mechanisms provided by the Unity Engine are reviewed, and File System is selected as an appropriate option. Storage of object coordinates to the File System is implemented. The results provide a baseline for developing a system for creating virtual stands for professional research.

Keywords: Virtual reality, Data management, File System, Serialization.
Article info: Received 29 March 2022; received in revised form 9 October 2022; accepted 17 November 2022.

1. Introduction

In virtual reality, data is represented both in primitive types (int, float, string) and complex types provided by the engine. Object positions in space are determined in the Cartesian coordinate system [1] (see Fig. 1).

A common task is to compare the distance of 2 points from a given point $A(x_1, y_1, z_1)$. The Unity Engine Scripting API [2] provides a complex data type called Vector3 to store object coordinates, along with its Vector3.Distance() method to calculate distance between 2 points. Given the points (x_2, y_2, z_2), $C(x_3, y_3, z_3)$, this method may be used to accomplish the task, comparing the following values: Vector3.Distance(B, A), Vector3.Distance(C, A).
A more efficient solution may be applied using the formula of the distance between 2 points [1]:

$$d_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}, \quad (1)$$

$$d_{AC} = \sqrt{(x_3 - x_{1)}{}^2 + (y_3 - y_{1)}{}^2 + (z_3 - z_{1)}{}^2} \quad (2)$$

Instead of comparing values for d_{AB} and d_{AC}, the radicands may be compared, saving CPU time on unnecessary calculations.

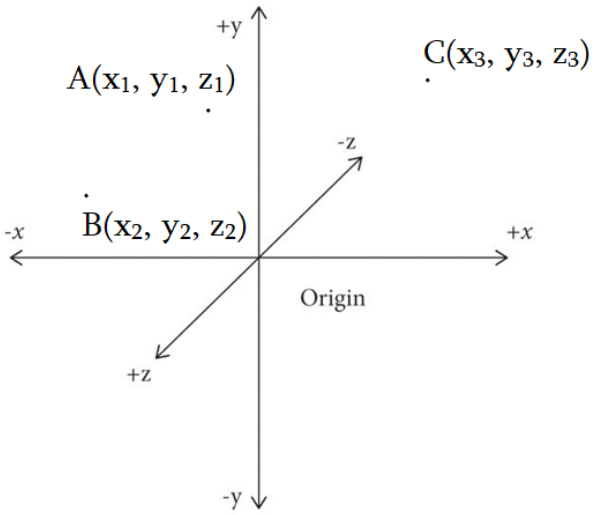

Fig. 1. Object positions in space.

To have persistent data between sessions, the user progress has to be stored on the disk. There are several methods of managing data storage, including SQL database, PlayerPrefs and OS File System. On specific events during the runtime, which are to be defined, data containing all the current values have to be stored. These events may include user interaction, object state mutation, or events may be set to trigger on specific timestamps, e.g., every 10 seconds. Then, on the next program run, these stored values have to be fetched and transmitted to the engine to render the objects in the same state and position, as they were when the last event was triggered.

2. Persistent Data

To have persistent data between sessions, the user progress has to be stored on the disk. Below are listed several methods of managing data storage.

1. SQL Database

SQL is useful when there is relational data. It supports queries to fetch related data sets. In our case, we have just objects that need to be memorized and then retrieved on the next run. Such simple operations are easier to implement and faster in work on File System. SQL is a dedicated software and isn't an integrated part of Operating Systems, as File System is. Also, a connection to SQL service should be kept active during the runtime.

2. PlayerPrefs

PlayerPrefs is a class provided by Unity Engine that stores Player preferences between game sessions. It stores values in the OS registry. Though it is possible to store data using this

method, it is not recommended to do so. This method should be used for data, that can be afforded to lose, such as user settings and preferences. Sensitive and relatively big data should not be kept in registries.

3. File System

To store data in files, it needs to be formatted in some way. It may be serialized [3] to binary format and written to a file. That data will then be successfully deserialized and used in the application. But since binary is not human-readable, it makes this format insufficient. Moreover, it is not possible to edit the saved data manually. Using JSON data type allows bypassing these problems.

Taking into account the points mentioned above, it was decided to handle data storage using File system and Serialization, so every time data needs to be stored, it is serialized to JSON format and written to a file (see Fig. 2). Then, to restore the state in the application, the file is read and data is deserialized to object (see Fig. 3).

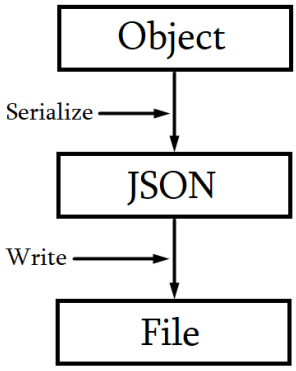

Fig. 2. Storing Data.

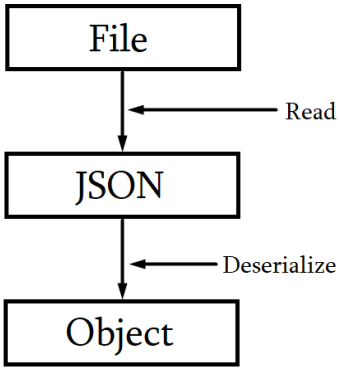

Fig. 3. Using Stored Data.

3. Saving Object Position

A specific and common example is persisting the object position. In this example, we have a cube placed on a table (see Fig. 4). The Origin (0, 0, 0) can be located on the ground.

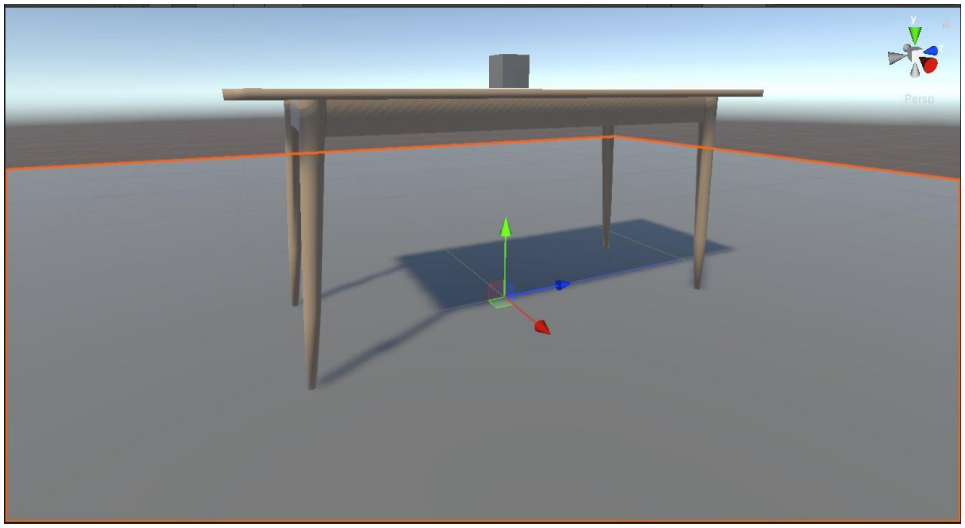

Fig. 4. Cube.

To save the cube position after it is replaced, a class called SaveManager is created, which contains 2 methods: save and load. These methods use a file called "position.dat" to write/read data.

SaveManager.cs

```
public class SaveManager
{
    public static void save(Vector3 pos)
    {
        string path = Path.Combine(Application.persistentDataPath, "position.dat");
        File.WriteAllText(path, JsonUtility.ToJson(pos));
    }

    public static Vector3 load()
    {
        string path = Path.Combine(Application.persistentDataPath, "position.dat");
        string result = File.ReadAllText(path);
        Vector3 pos = JsonUtility.FromJson<Vector3>(result);
        return pos;
    }
}
```

The save and load methods would then be invoked from a script, which is bound to the object. The save method would be bound to the XR Grab Interactable component [4] “Select Exited” event to save data every time the object is released. The load method would be invoked from the Start method to set object positions from the saved data on a fresh program run.

CubeScript.cs

```
public class CubeScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Vector3 position = SaveManager.load();
        transform.position = position;
    }

    public void Save()
    {
        SaveManager.save(transform.position);
    }
}
```

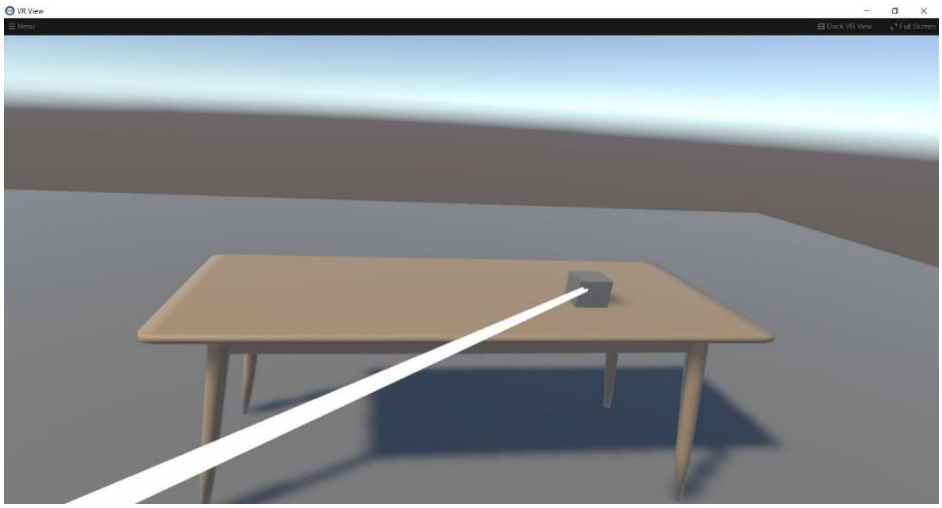

Fig. 5. Cube position changed via left controller.

The saved file position.dat:
{"x":-0.0030583017505705358,"y":0.838373601436615,"z":-2.0934112071990969}
After restarting the program, we still have the cube in its new place (see Fig. 5).
Now we can modify this file content, and set the coordinates to (0, 0, 0).
{"x":0,"y":0,"z":0}
After modifying and saving the file, and running the program again, we can see that the cube appears on the Origin as expected (see Fig. 6).

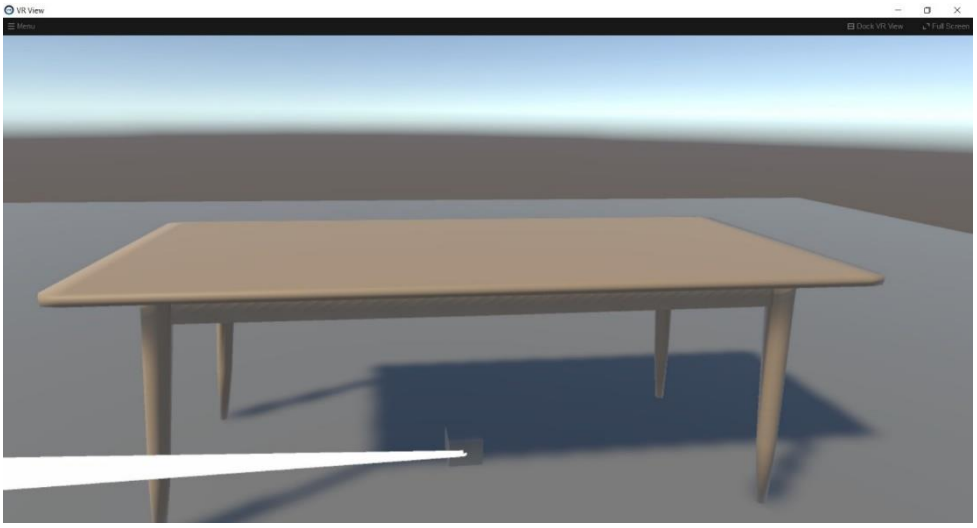

Fig. 6. Cube position manually set to Origin.

4. Conclusion

In this article, an improvement to the Unity Engine distance calculation algorithm was suggested. Additionally, data types provided by the Unity Engine were reviewed. Data storage options were compared and decided to use the OS File System and data serialization. As an example, a cube position storage and loading were implemented. This method will be used also for custom complex data types to store, marking the class representing the data type as Serializable.

References

[1] Wikipedia, (2012) Cartesian Coordinate System. [Online]. Available: https://en.wikipedia.org/wiki/Cartesian_coordinate_system

[2] Unity Engine Scripting API Reference. [Online]. Available: https://docs.unity3d.com/ScriptReference/

[3] Microsoft Docs, (2021) Serialization (C#). [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/

[4] The Khronos Group Inc., “The OpenXR Specification”.

[5] Unity Learning, (2020) Create with VR. [Online]. Available: https://learn.unity.com/course/create-with-vr?uv=2020.3

Տվյալների մշակումը և պահպանումը վիրտուալ իրականության համակարգերում

Արման Ա. Հովհաննիսյան
Հայաստանի ազգային պոլիտեխնիկական համալսարան
e-mail: aahovhannisyan1@gmail.com

Ամփոփում

Տվյալների մշակումը և պահպանումը վիրտուալ իրականության համակարգի զարգացման հիմնական ասպեկտներ են։ Այս հոդվածում առաջարկվում է Unity շարժիչի հեռավորության հաշվարկման ալգորիթմի լավարկում։ Բացի այդ, դիտարկվում են Unity շարժիչի կողմից տրամադրվող տվյալների պահպանման մեխանիզմները, և որպես նպատակահարմար տարբերակ, ընտրվում է ֆայլային համակարգը։ Իրականացվում է օբյեկտի կոորդինատների պահպանումը ֆայլային համակարգում։ Ստացված արդյունքները հիմք են հանդիսանում մասնագիտական հետազոտությունների վիրտուալ ստենդների ստեղծման համակարգի մշակման համար։

Բանալի բառեր՝ Վիրտուալ իրականություն, տվյալների կառավարում, ֆայլային համակարգ, սերիալիզացիա

Обработка и сохранение данных в системах виртуальной реальности

Арман А. Оганесян

Национальный политехнический университет Армении
e-mail: aahovhannisyan1@gmail.com

Аннотация

Обработка и сохранение данных являются ключевыми аспектами разработки системы виртуальной реальности. В данной статье предлагается улучшение алгоритма расчета расстояний Unity Engine. Кроме того, рассматриваются механизмы сохранения данных, предоставляемые Unity Engine, и в качестве подходящего варианта выбирается файловая система. Реализуется хранение координат объекта в файловой системе. Результаты обеспечивают основу для разработки системы создания виртуальных стендов для профессиональных исследований.

Ключевые слова: Виртуальная реальность, управление данными, файловая система, сериализация