Assignment 2
Virtual Assembly

Ozan Tasli
CEM242 Autumn 2016
Advanced Visualization and Interactive Technologies
cr899898@reading.ac.uk

ABSTRACT

The purpose of this project is to show how Virtual Environments (VEs) and virtual prototypes can be used in the construction industry to perform Virtual Assembly (VA) of a construction using Unity 3D software.

Keywords

Virtual Assembly (VA), Unity 3D, Virtual Environment (VE)

INTRODUCTION

Recent developments in the Computer Aided Design (CAD) and visualization technologies have enabled the integration of two concepts, Building Information Models (BIMs) and Virtual Reality (VR). These innovations have enabled construction project teams to work on virtual prototypes in a Virtual Environments (VEs) to make design decisions, clash detections, or project reviews. Visualization tools such as helmet mounted displays, cyber gloves, position trackers etc., one or more users can be immersed in a VE and interact with the artificial reality.

These developments have made Virtual Assembly (VA) an effective tool which facilitates assembly related processes in the Architecture/Engineering/Construction (AEC) industry such as planning assembly processes, evaluating constructability, or worker training, without requiring the creation of a physical prototype. These technologies, which have been used largely in industries like aerospace or automation, have gained importance in the AEC industry as there is increasing tendency to the Design for Manufacture and Assembly (DfMA) approach.

The main aim of this study was to create a VE by utilizing Unity 3D software, and to simulate VA of a construction. This project was developed based on a steel-framework of a construction that was built with DfMA approach. The project showed an example of how VEs and VAs can be used within the AEC industry. Furthermore, this study demonstrated several methods and techniques that can be used to create VEs by using Unity 3D software.

VIRTUAL PROTOTYPING

The first step of the project was to prepare a virtual prototype before creating the VE. To make the project more realistic, virtual prototype was developed by adapting an existing building which was constructed with the DfMA approach. Two levels of steel-framework of the Leadenhall Building, which is a multi-story building located in London, was modelled by using AutoCad 3D and Google Sketchup software (Fig.1).

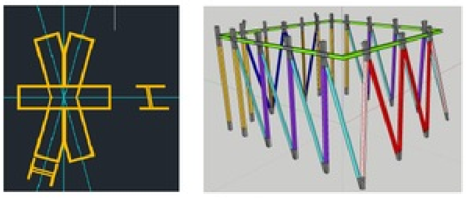


Figure 1

Once the prototype was finalized, the model was exported as an FBX file and then imported to Unity 3D. At this stage, several problems occurred. Firstly, the textures of some joint elements were not visible. Secondly, the pivot points of each component were shifted. To solve this problem, the model was exported to Autodesk 3DS Max. The pivot point of each component and textures were corrected. However, as the Y and Z axes are different in Unity 3D and 3DS Max, the model was rotated in the opposite direction when exported to Unity (Fig. 2).

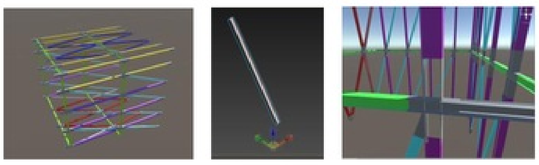


Figure 2

To fix this problem, the model was rotated into the Y axes of 3D Max. After fixing the rotation, textures were attached again and the pivot point of each component was centralized. The model was successfully exported to Unity 3D (Fig. 3). Based on this experience it can be concluded that Unity 3D is more compatible with 3DS Max compared to other software.

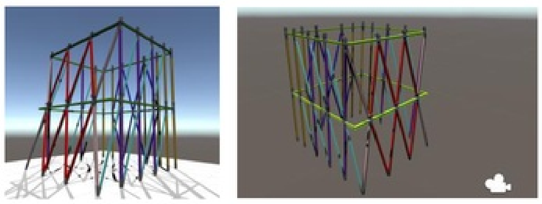


Figure 3

The idea was to immerse the user into a VE in which the ground floor joint elements were activated. Therefore, all the components except the ground floor joints were deactivated from the hierarchy tab in Unity 3D (Fig. 4).

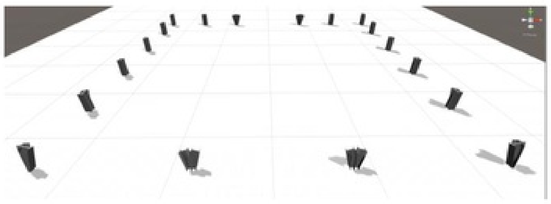


Figure 4

The aim was to activate disabled components when the user physically moved an object and intersected it with another object. In order to realize the physical intersection, two different categories of objects were created. The first category, which included eleven vertical components, was created and named 'Interactables' (Fig. 5).

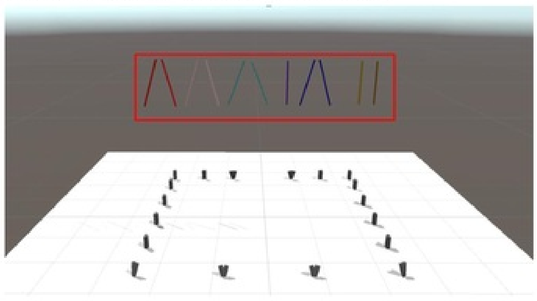


Figure 5

The 'Interactable' objects were placed within the range of the main camera. The second category, which includes a number of cubes, was created in Unity 3D (Fig. 6). The cubes were placed at the edges of each joint element. The cubes were color coded to facilitate the user`s perception of the model.

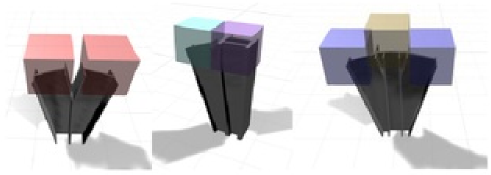


Figure 6

The purpose was to enable the user/s to choose one of the objects from the 'Interactable' category, and then drag it into the cubes to activate a related component. To control this action, the 'OnTriggerEnter' method was used in a script, and several steps were followed. Firstly, the 'Rigidbody' component was added to all the cubes. The "Gravity" checkbox was disabled to prevent the cubes from acting against gravity. The 'Is trigger' and 'Box collider' functions were activated (Fig. 7).

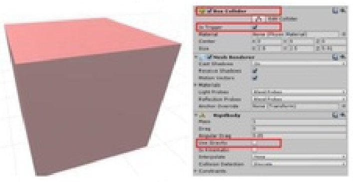


Figure 7

These adjustments enabled use of the 'OnTriggerEnter' method to control the action by the script when an Interactable object entered or exited the trigger volume of the cubes. The 'Trigger Control' script was created and linked with all the cubes. Similarly, another script was named 'Interactable' and attached to the Interactable objects. The second step was to connect the Interactable objects to the relevant cubes. Therefore, eleven different types of color codes were identified in the other script. The 'ItemType' script was applied to both 'Interactable' and 'Trigger Control' scripts (Fig. 8).

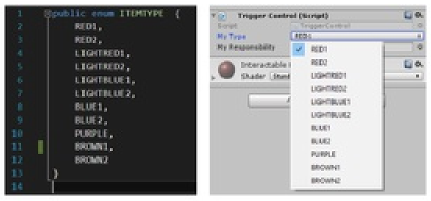

```
public enum ITEMTYPE  {
        RED1,
        RED2,
        LIGHTRED1,
        LIGHTRED2,
        LIGHTBLUE1,
        LIGHTBLUE2,
        BLUE1,
        BLUE2,
        PURPLE,
        BROWN1,
        BROWN2
    }
```
Trigger Control (Script)
Script TriggerControl
My Type RED1
My Responsibility

Figure 8

This move enabled the assignment of an item type for both the Interactable and cube objects. Then, all the components were given a type based on color and direction. The purpose of this was to compare types of both objects when the 'OnTriggerEnter' command was activated. For example, if the user chose the type RED 1 Interactable object, it could interact only with the cubes that had the type RED 1 (Fig. 9).

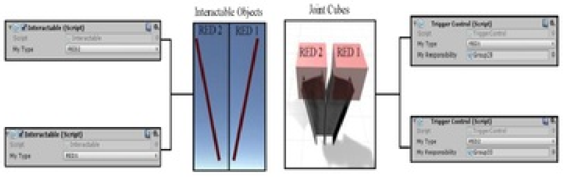


Figure 9

Another method, referred to as 'My Responsibility,' was applied to the 'Trigger Control' script that was attached to the cubes. This function enabled identification of which object to activate when the types of both the Interactable and the cube objects were matched correctly. Therefore, in the 'My Responsibility' variable within the Trigger Control script (Fig. 10) each of the cubes were attached with its deactivated object. This script enabled the use of the 'SetActive' method, which basically activated the deactivated objects, once the OnTriggerEnter function was recognized.

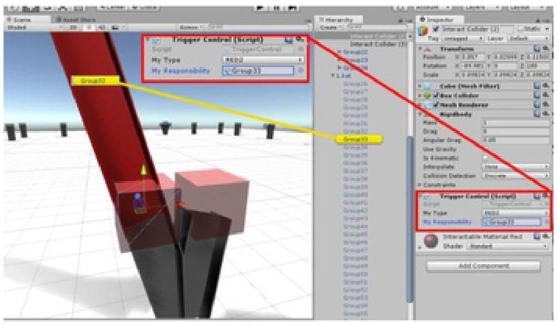
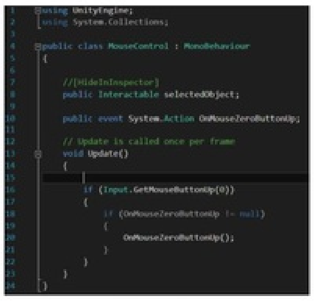


Figure 10

USER CONTROLLERS

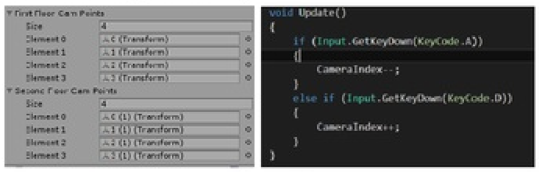
The initial aim of the project was to enable user interaction with the HTC-Vive toolkit. However, due to the limited time and the complexity of the three-dimensional controllers, the PC keyboard and mouse were used instead. In the planned scenario, the user was to use the mouse to choose the Interactable objects by simply clicking and dragging them into the cubes. To control movement of the Interactable objects in a 3D environment, the Vector3, transform position, and transform scale variables were applied within an 'Interactable' script. These variables were controlled by a 'Mouse Control' script, which enabled obtaining mouse position input once per frame as the user dragged the Interactable objects (Fig. 11). In brief, the scripts enabled matching position values of the objects and the mouse controller.



```
using UnityEngine;
using System.Collections;

public class MouseControl : MonoBehaviour
{

    //[HideInInspector]
    public Interactable selectedObject;

    public event System.Action OnMouseZeroButtonUp;

    // Update is called once per frame
    void Update()
    {

        if (Input.GetMouseButtonUp(0))
        {
            if (OnMouseZeroButtonUp != null)
            {
                OnMouseZeroButtonUp();
            }
        }
    }
}
```

Figure 11

Due to the difficulty of controlling the mouse in the Z axes on a 2D screen, four different points of view were identified in a 'Camera Control' script (Fig. 12), which enabled the user to shift among points of view by pressing the A and D keys of the keyboard.

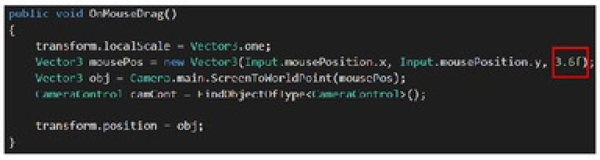


```
▼ First Floor Cam Points
    Size        4
    Element 0   0 (Transform)
    Element 1   1 (Transform)
    Element 2   2 (Transform)
    Element 3   3 (Transform)
▼ Second Floor Cam Points
    Size        4
    Element 0   0 (1) (Transform)
    Element 1   1 (1) (Transform)
    Element 2   2 (1) (Transform)
    Element 3   3 (1) (Transform)
```

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.A))
    {
        CameraIndex--;
    }
    else if (Input.GetKeyDown(KeyCode.D))
    {
        CameraIndex++;
    }
}
```

Figure 12

This approach enabled actualization of the interaction of the Interactables and the cubes by changing the X and Y position values only.



```
public void OnMouseDrag()
{
    transform.localScale = Vector3.one;
    Vector3 mousePos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 3.6f);
    Vector3 obj = Camera.main.ScreenToWorldPoint(mousePos);
    CameraControl camCont = FindObjectOfType<CameraControl>();

    transform.position = obj;
}
```

Figure 13

After several tries, a numeric value, which enabled stabilizing the Z axes of the Interactables, was found. (Fig. 13). This fixed value enabled overlaying the Z axes of the Interactables with cubes as they moved in the X and Y directions. (Fig. 14).

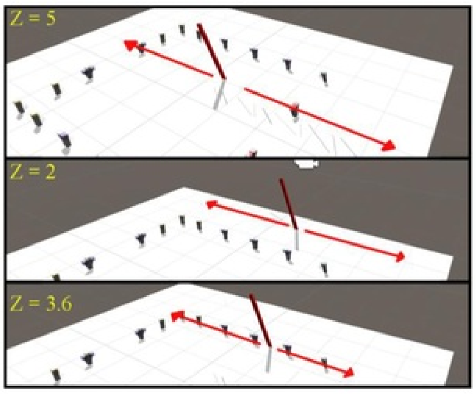
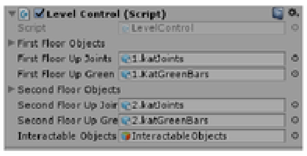


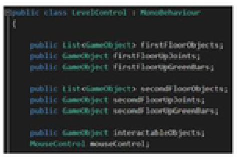
Figure 14

The distance between point of views and the model pivot center were kept the same. Thus, the fixed value worked smoothly for all point of views.

Final part was to activate the horizontal components of the structure once the first level is completed. These components, which had been disactivated in the hierarch tab, were applied to a 'Level Control' script (Fig. 15). The script enabled to check whether the first level objects are activated or not. Once all objects within the first level are activated, 'Set Active' function was enabled to activate horizontal elements. Similar methods were used to deactivate 'Interactable' objects once the whole structure is completed.



```
Level Control (Script)
Script                    LevelControl
First Floor Objects
First Floor Up Joints     1.katJoints
First Floor Up Green      1.KatGreenBars
Second Floor Objects
Second Floor Up Joir      2.katJoints
Second Floor Up Gre       2.katGreenBars
Interactable Objects      InteractableObjects
```



```
public class LevelControl : MonoBehaviour
{

    public List<GameObject> firstFloorObjects;
    public GameObject firstFloorUpJoints;
    public GameObject firstFloorUpGreenBars;

    public List<GameObject> secondFloorObjects;
    public GameObject secondFloorUpJoints;
    public GameObject secondFloorUpGreenBars;

    public GameObject interactableObjects;
    MouseControl mouseControl;
```

Figure 15

The methods that were mentioned earlier were applied for the second level to enable the user finalize the whole structure.

CONCLUSIONS

Overall, the project showed that VEs and VA can be very useful for the AEC industry. Through the Unity 3D software, construction processes of a project can be simulated. Immersive computer-generated environments can be used for several purposes such as training construction workers, sequencing construction phases, performing VA etc., which significantly increases developments of the industry.

This study showed an example for VA within the VE. Even though the level of user interactivity was limited by the computer controllers, this project showed that more immersive technologies such as helmet mounted displays, cyber gloves can be adopted for higher levels of user immersion.

FUTURE DEVELOPMENTS

The future studies can be developing a VE in which user can interact with a virtual world by 3D controllers such as HTC-Vive. 'Raycast' method can be useful for adopting higher levels of interaction. More realistic physics engines would significantly increase the level of immersion.

APPENDICIES

APPENDIX A

Item Type Script

```
C:\Users\Ozan\Desktop\Unity\Assets\Scripts\ITEMTYPE.cs                          1
public enum ITEMTYPE
{
    RED1,
    RED2,
    LIGHTRED1,
    LIGHTRED2,
    LIGHTBLUE1,
    LIGHTBLUE2,
    BLUE1,
    BLUE2,
    PURPLE,
    BROWN1,
    BROWN2
}

```

APPENDIX B

Mouse Control Script

C:\Users\Ozan\Desktop\Unity\Assets\Scripts\MouseControl.cs 1

```
using UnityEngine;
using System.Collections;

public class MouseControl : MonoBehaviour
{

    [HideInInspector]
    public Interactable selectedObject;

    public event System.Action OnMouseZeroButtonUp;

    // Update is called once per frame
    void Update()
    {

        if (Input.GetMouseButtonUp(0))
        {
            if (OnMouseZeroButtonUp != null)
            {
                OnMouseZeroButtonUp();
            }
        }
    }
}

```

APPENDIX C

Camera Control Script

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;

public class CameraControl : MonoBehaviour
{
    public List<Transform> firstFloorCamPoints;
    public List<Transform> secondFloorCamPoints;
    private List<Transform> cameraPoints;
    private int cameraIndex;
        public int CameraIndex
    {
        get { return cameraIndex; }
        set
        {

            cameraIndex = value;

            if (cameraIndex<0)
            {

                cameraIndex = cameraPoints.Count-1;

            }
            else if (cameraIndex>cameraPoints.Count-1)
            {
                cameraIndex = 0;
            }
            else
            {
            }

            Camera.main.transform.position = cameraPoints
              [cameraIndex].position;
            Camera.main.transform.rotation= cameraPoints[cameraIndex].rotation;
        }
    }

    void Start()
    {
        cameraPoints = firstFloorCamPoints;
        CameraIndex = 1;
    }

    internal void FloorUp()
    {
        cameraPoints = secondFloorCamPoints;
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.A))
        {

            CameraIndex--;
        }
        else if (Input.GetKeyDown(KeyCode.D))
        {
            CameraIndex++;
        }
    }
}

```

APPENDIX D

Level Control Script

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class LevelControl : MonoBehaviour
{

    public List<GameObject> firstFloorObjects;
    public GameObject firstFloorUpJoints;
    public GameObject firstFloorUpGreenBars;
    public List<GameObject> secondFloorObjects;
    public GameObject secondFloorUpJoints;
    public GameObject secondFloorUpGreenBars;
    public GameObject interactableObjects;
    MouseControl mouseControl;
    void Start()
    {

    }
            void Update()
    {
        if (Input.GetKey(KeyCode.Escape))
        {
            Application.Quit();
        }
    }

    public void CheckFirstFloorComplete()
    {
        foreach (GameObject item in firstFloorObjects)
        {
            if (!item.activeInHierarchy)
            {
                return;
            }
        }
        firstFloorUpJoints.SetActive(true);
        StartCoroutine(ActivateGreenBars1());
        FindObjectOfType<CameraControl>().FloorUp();
    }

    public void CheckSecondFloorComplete()
    {
        foreach (GameObject item in secondFloorObjects)
        {
            if (!item.activeInHierarchy)
            {
                return;
            }
        }
        secondFloorUpJoints.SetActive(true);
        StartCoroutine(ActivateGreenBars2());
    }

    IEnumerator ActivateGreenBars1()
    {
        yield return new WaitForSeconds(1f);
        firstFloorUpGreenBars.SetActive(true);
    }

    IEnumerator ActivateGreenBars2()
    {
        yield return new WaitForSeconds(1f);
        secondFloorUpGreenBars.SetActive(true);
        interactableObjects.SetActive(false);
    }
}

```

APPENDIX E

Trigger Control Script

C:\Users\Ozan\Desktop\Unity\Assets\Scripts\TriggerControl.cs

```
using UnityEngine;
using System.Collections;

public class TriggerControl : MonoBehaviour
{

    public ITEMTYPE myType;
    public GameObject myResponsibility;
    Interactable inter;

    public void OnTriggerEnter(Collider other)
    {
        if (other.GetComponent<Interactable>() != null)
        {
            inter = other.GetComponent<Interactable>();
            if (inter.myType == myType)
            {
                Debug.Log("Match");
                FindObjectOfType<MouseControl>().OnMouseZeroButtonUp += ActivateMyObject;
            }
        }
    }

    public void OnTriggerExit(Collider other)
    {
        if (other.GetComponent<Interactable>() != null)
        {
            inter = other.GetComponent<Interactable>();
            if (inter.myType == myType)
            {
                Debug.Log("Mismatch ");
                FindObjectOfType<MouseControl>().OnMouseZeroButtonUp -= ActivateMyObject;

            }
        }
    }

        public void ActivateMyObject()
    {
        myResponsibility.SetActive(true);
        FindObjectOfType<MouseControl>().OnMouseZeroButtonUp -= ActivateMyObject;
        FindObjectOfType<LevelControl>().CheckFirstFloorComplete();
        FindObjectOfType<LevelControl>().CheckSecondFloorComplete();
        gameObject.SetActive(false);

    }


}

```

APPENDIX F

Interactable Script

C:\Users\Ozan\Desktop\Unity\Assets\Scripts\Interactable.cs

```csharp
using UnityEngine;
using System.Collections;

public class Interactable : MonoBehaviour
{

    public ITEMTYPE myType;

    Vector3 initialPosition;
    Vector3 initialScale;


    public void Start()
    {
        initialPosition = transform.localPosition;
        initialScale = transform.localScale;

        FindObjectOfType<MouseControl>().OnMouseZeroButtonUp += Interactable_OnMouseZeroButtonUp;
    }


    private void Interactable_OnMouseZeroButtonUp()
    {
        transform.localPosition = initialPosition;
        transform.localScale = initialScale;
    }


    public void OnMouseDrag()
    {
        transform.localScale = Vector3.one;
        Vector3 mousePos = new Vector3(Input.mousePosition.x, Input.mousePosition.y, 3.6f);
        Vector3 obj = Camera.main.ScreenToWorldPoint(mousePos);
        CameraControl camCont = FindObjectOfType<CameraControl>();

        transform.position = obj;
    }
}

```