Spring 2018 Midterm Progress Report

Group 61: iCreate - Generative Design in Virtual Reality

CS 463 | Spring 2018 Term

Hannah Solorzano - Nabeel Shariff - Rhea Mae Edwards

Abstract

In regards to the CS 463 senior capstone project, group 61, Generative Design in Virtual Reality, the purpose of this document is to briefly recap the project's purpose and goals, describe where the group is at currently on the project, describe what the group has left to do, and describe any problems that have impeded the group's progress with any solution that have had or currently still have yet to implement, along with the addition of any particularly interesting pieces of code, a description of results from user studies done in regards to the interfaces design, and any images of the project, including screen shots and any additional files further describing the project itself.

PARTICIPANTS

The ICreate - Generative Design in Virtual Reality senior software engineering project consists of the following team members:

Raffaele de Amicis, *Mentor*

Rhea Mae Edwards | Nabeel Shariff | Hannah Solorzano

The following persons are associated with Intel Corporation in regards to Hardware, Software and Development Environments in relations to the project:

Mike Premi, *Co-Mentor*

The following persons are guidance mentors as a part of the senior software engineering project course for the students:

Kevin McGrath, *Instructor* | **Behnam Saeedi,** *Teaching Assistant* | **Kirsten Winters,** *Instructor*

TABLE OF CONTENTS

1 Introduction **4**
- 1.1 Purpose 4
- 1.2 Goals 4

2 Project's Progress **4**
- 2.1 Current State 4
  - 2.1.1 External Interface Requirements 5
  - 2.1.2 Functional Requirements 5
  - 2.1.3 Performance Requirements 5
- 2.2 Left to Do 5
- 2.3 Impeded Problems and Solutions 6
  - 2.3.1 Previous Impeded Problems and Solutions 6

3 Interesting Pieces of Code **6**
- 3.1 Saving GameObject Characteristics 6
- 3.2 Splits a String into Usable Descriptive Components 7
- 3.3 Creating and Loading GameObjects 7
- 3.4 Algorithms 9
- 3.5 Mesh Drawing 9
- 3.6 VR Environment 12
- 3.7 Curves 12
- 3.8 Transformation and Translation 13
- 3.9 Save and Load 13

4 Expo Poster **14**

1 INTRODUCTION

1.1 Purpose

The purpose of this project is to develop a VR application that allows users to create complex 3D objects by seamlessly creating simpler objects and combining them to form intricate structures.

1.2 Goals

The virtual reality (VR) application will utilize a virtual reality headset with input from the user via a controller recognition software. The VR headset will be used to look around in virtual space while the controllers' recognition software will be used by the user to draw curves. The application will also need to utilize the GPU in a computer to both run the VR application and render 3D objects in the virtual space. Additionally, the 3D modeling will be based on generative design techniques, and the assembly of the complex 3D designs will utilize mathematical equations and algorithms to derive the appropriate structure of the design.

The outcome of this Oregon State University computer science senior capstone project is a virtual reality program that allows the user to utilize generative design to develop complex architectural structure.

2 PROJECT'S PROGRESS

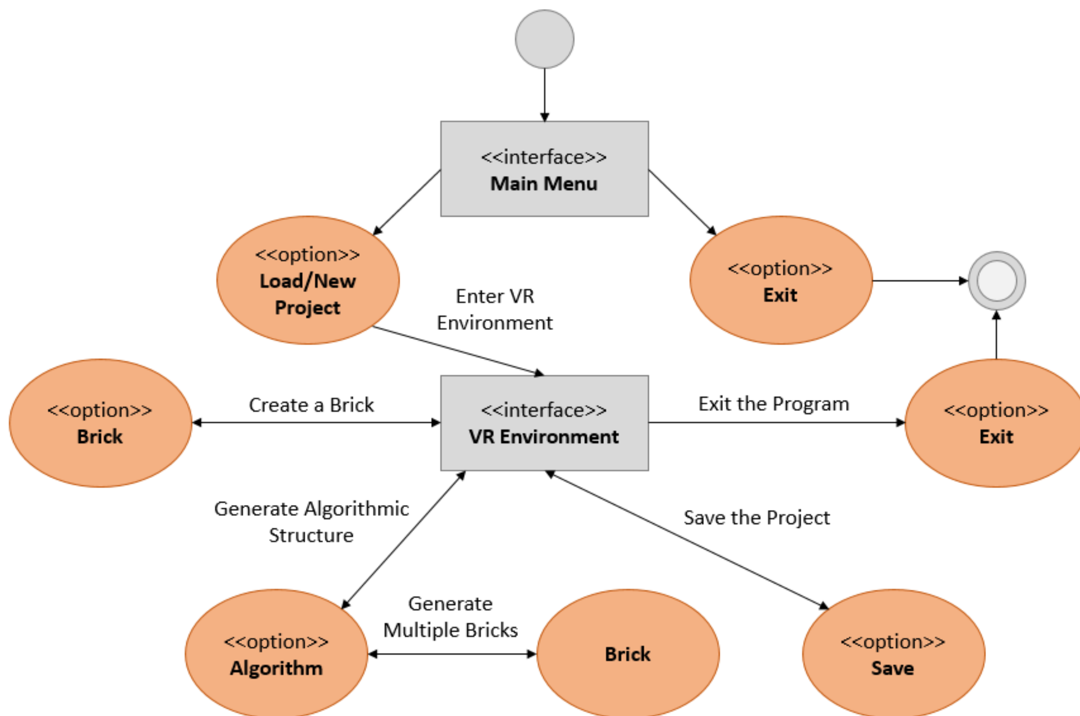

Figure 1. State diagram that discloses the various states and choices the user is presented with when using the iCreate program.

2.1 Current State

The group's project current state satisfies all of the following project requirements, which have also been stated in the team's original software requirements 1.1 document. Further video demonstrations of this progress report's presentation explain these project requirements in further detail and also visual shows the functionality of each of the project's pieces.

2.1.1 External Interface Requirements

- User Interfaces:
  The interface will functionally allow the user to spawn a 3D object, then modify it via a controller input. Additionally, the user will be able to save to a library for later use or load their creations to create complex structures. Finally, the application will display a way to transform the 3D objects by allowing the user to scale or resize the object.

- Hardware Interfaces:
  The VR application will be able to run on a HTC Vive VR headset and a computer that can run VR software. The minimum CPU and GPU requirements to successfully run the VR application are:
  - Minimum CPU requirements: Intel Core i5-4590 or AMD FX 8350 (equivalent or better)
  - Minimum GPU requirements: NVIDIA GTX 970 or AMD Radeon R9 290 (equivalent or better)

- Software Interfaces:
  The VR application will be usable on a Windows operating system capable of running the VR headset's respective proprietary software.

2.1.2 Functional Requirements

- VR Environment:
  The iCreate software will allow the user to move around in VR, instantiate a simple 3d object, and generate multiple instances of the 3d object.

- Object Library:
  The iCreate software will allow the user to obtain 3D objects from the application's library.

- Curves:
  The iCreate software will allow the user to draw a 3D curve, draw a trajectory in the form of a curve, create a B-Spline curve, a Bezier curve, a ellipse curve, a circle curve, a hyperbola curve, and a parabola curve, and use the curve to indicate how the user would like to displace multiple instances of the initial object.

- Transformation and Translation:
  The iCreate software will allow the user to, extrude the object, resize the object, and indicate how they would like to rotate, translate, and scale objects across a curve. The 3D objects that make up a complex structure must be connected with each other.

- Save and Load:
  The iCreate software will allow the user to import geometry from a .fbx extension file format and a 3D object and save a 3D object, structures, and the entirety of the project as a whole.

2.1.3 Performance Requirements

The iCreate program should be able to run on a computer with at least an Intel Core i5-4590 or AMD FX 8350 (equivalent or better), a NVIDIA GTX 970 or AMD Radeon R9 290 (equivalent or better) graphics card, at least 4GB of RAM, at at least 30 frames per second. The requirements to run iCreate will also depend on the user and the scale or intricacy of the architectural structure that is designed as a larger, more complex structure will require more processing power and a stronger graphics card.

2.2 Left to Do

In order to completely finish the group's project implementation is the combining of all of the project's implemented pieces into one single program, all in one Unity project in a sense.

2.3 Impeded Problems and Solutions

Since the last winter 2018 progress report the group has submitted for the course, no additional impeded problems along with solution have not became a concern for group or for the group's overall project implementation.

2.3.1 Previous Impeded Problems and Solutions

- Impeded Problem: Personal Individual Group Members' Situations
  Solution:
  Mainly in order to overcome the situation, we simply just have to keep moving forwards and look on the brighter side of things. Remaining the stay motivate and do what we had to do. In addition, we had to seek out extra help and assistance in moving forward through the project. Just have to make it through and hopefully successfully, or at least just enough.

- Impeded Problem: Client Confusion and Indirectness
  Solution:
  Patience. Also seeking our help and assistance in what each of us has to do and how, in addition to clarifying issues with our client, where at times we had to take it to the TA and instructors level of assistance. In general, the team works the best when we are told what are suppose to do the first time around, and then we work hard on that, but being easily confused week by week, is not helpful. We also completed and went through detailed game plans in order to know how to tackle every challenge we faced and are still facing. As a group, we did gain a lot of highly appreciated assistance from our TA, Behnam Saeedi. As a group, we can say we have not been more thankful.

3 Interesting Pieces of Code

3.1 Saving GameObject Characteristics

```
public GameObject blockList;
string s = "";
string p1, p2, p3;
string sc1, sc2, sc3;
string r1, r2, r3;
string type;

        // Loops through all of the GameObjects in the list of blocks
        // Finds and Stores each GameObject's position, scaling, rotation, and type
        foreach (Transform myChild in blockList.GetComponentsInChildren<Transform>()) {
            s = string.Concat(s, "(");
            p1 = (myChild.position.x).ToString();
            p2 = (myChild.position.y).ToString();
            p3 = (myChild.position.z).ToString();
            s = string.Concat(s, p1, ",", p2, ",", p3, ",");
            sc1 = (myChild.localScale.x).ToString();
            sc2 = (myChild.localScale.y).ToString();
            sc3 = (myChild.localScale.z).ToString();
            s = string.Concat(s, sc1, ",", sc2, ",", sc3, ",");
            r1 = (myChild.eulerAngles.x).ToString();
            r2 = (myChild.eulerAngles.y).ToString();
            r3 = (myChild.eulerAngles.z).ToString();
            s = string.Concat(s, r1, ",", r2, ",", r3, ",");
            type = "T";
            s = string.Concat(s, type, ")\n");
        }
```

Setting aside to research on how to match C# language syntax to design ideas that I planned to implement, this case displays the execution method I used to store float and string values into an ongoing string later to be stored in a file of

game objects being saved within game scene. For now, the string value in to indicate the type of game object being saved is valued at T for all game objects with this current save implementation, which will be fixed to be more appropriate for use in later development. The idea of doing so was planned out with help from our team's TA, Behnam Saeedi.

3.2 Splits a String into Usable Descriptive Components

```
string textIn = "";
string[] object_list;
ArrayList character_list = new ArrayList();

        // Tokenizes GameObjects described in the string from the file by newline (\n)
        // Adds objects in a string array, object_list
        object_list = textIn.Split('\n');

        foreach (string gameobj in object_list) {
            string obj;

            // Ignoring the first and last character on each line
               (the opening and closing parantheses (( and )))
            // Puts a GameObject in a variable called obj for useage
            obj = gameobj.Replace("(", "");
            obj = obj.Replace(")", "");

            // Tokenize each component in each of the GameObjects by comma (,)
            // Takes a GameObject's components and stores them in a string array
               called item_list
            string[] item_list;
            item_list = obj.Split(',');

            // Inputs each component in an ongoing Arraylist called character_list
            foreach (string item in item_list) {
                character_list.Add(item);
            }
        }
```

Above, this piece of code represents the implementation of splitting up the saved string back into small usage pieces to use to load game objects back into a game scene. The extra characters that were used in order to save the string consisting of information of saved game objects in an orderly fashion, is stripped off and removed in order to extract values to be proper used, being stored in an arraylist within the process. Such an idea was also help thought out with the project's TA, Behnam Saeedi, along with additional research of how to implement an arraylist and why such a choice was the proper use to implement and simpler to understand overall, even though such a decision may not be clear at first glance.

3.3 Creating and Loading GameObjects

```
public List<GameObject> spawn_list;
ArrayList character_list = new ArrayList();
int count;
int spawn_index = 0;
GameObject tempObject;
spawn_list = new List<GameObject>();
float po1, po2, po3;
float sca1, sca2, sca3;
float ro1, ro2, ro3;

        // Create/Spawn a GameObject(s) based of the components in the
```

```
   Arraylist (character_list)

// (character_list[0], character_list[1], character_list[2], character_list[3],
    character_list[4], character_list[5], character_list[6],
    character_list[7], character_list[8], character_list[9])
    and so on/repeat in the same list

// (float, float, float, float, float, float, float, float, float, string)
// (p1, p2, p3, sc1, sc2, sc3, r1, r2, r3, type)
// (object, object, object, object, object, object, object, object, object, object)

// For Loop Algorithm:
// For every set of 10 elements in character_list AKA every one object
// - Check which type the object is, every 10 steps,
     first occurrence: character_list[9] and so on..
// - Covert strings/objects back into floats, except for the type,
     as the following occur:
//    - Get the x, y, z position of object
//    - Get the x, y, z scale of object
//    - Get the x, y, z rotation of object
//    - Spawn the object into the game scene

count = character_list.Count;

for (int i = 0; i < count - 1; i++) {
    // i += 9; at the end every time an object is spawned

    string tempType = Convert.ToString(character_list[i + 9]);

    if (tempType == "T") {
        po1 = Convert.ToSingle(character_list[i]);
        po2 = Convert.ToSingle(character_list[i + 1]);
        po3 = Convert.ToSingle(character_list[i + 2]);
        sca1 = Convert.ToSingle(character_list[i + 3]);
        sca2 = Convert.ToSingle(character_list[i + 4]);
        sca3 = Convert.ToSingle(character_list[i + 5]);
        ro1 = Convert.ToSingle(character_list[i + 6]);
        ro2 = Convert.ToSingle(character_list[i + 7]);
        ro3 = Convert.ToSingle(character_list[i + 8]);

        tempObject = GameObject.CreatePrimitive(PrimitiveType.Cube);

        tempObject.transform.position = new Vector3(po1, po2, po3);
        tempObject.transform.localScale = new Vector3(sca1, sca2, sca3);
        tempObject.transform.eulerAngles = new Vector3(ro1, ro2, ro3);

        spawn_list.Insert(spawn_index, tempObject);
        spawn_index += 1;

        i += 9;
    }
    else {
        Debug.Log("Unable to create a GameObject.");
        i += 9;
    }
}
```

The code above illustrates the creation and spawning of game objects back into a game scene. The for loop implemented

for this extraction and spawning functionality, iterates through the specified arraylist. Depending on the type of game object being read, will depend which game object is loaded back into the game scene. Note that with my current implementation, cubes are only being generated which the same string value T is inputted for all of the previous game objects saved. Then based off where certain float values are previous systematically placed within the arraylist, will then determine which float values are which to later be used in creating game objects being loaded back into a game scene.

If a game object for some reason cannot be properly loaded back into a game scene, the program will currently only output an error message to the debug console log, and then move on until the loop has finished iterating through the arraylist.

The spawning of game objects back into the game scene was help created by the project's TA, Behnam Saeedi.

3.4 Algorithms

While there are several algorthims implemented for the creation of the curves, the one I found to be most interesting was the circle curve. The required calculation for this trajectory begins with getting the specified number of segments that make up the circle.

```
segments = (int)(360 / (Mathf.Asin(objectWidth / (xradius)) * 180.0 / 3.1415));
```

It then uses Unity's LineRenderer component to place the lines in the segment's position. After the lines are in place, the function CreatePoints() is called which is what adds the blocks to the rendered circle trajectory. Using the mathematical equations, this function transforms multiple cubes, end to end, to complete an uninterrupted circle.

```
for (int i = 0; i < (segments + 1); i++)
        {

            x = Mathf.Sin(Mathf.Deg2Rad * angle) * xradius + offset;
            y = Mathf.Cos(Mathf.Deg2Rad * angle) * yradius + offset;

            line.SetPosition(i, new Vector3(x, y, z));

            tempCube = GameObject.CreatePrimitive(PrimitiveType.Cube);

            tempCube.transform.position = new Vector3(line.GetPosition(i).x,
                  line.GetPosition(i).y, line.GetPosition(i).z);
            float xAngle = Mathf.Acos((line.GetPosition(i).x - xoffset) / xradius) *
                 (float)(180.0 / 3.1415) * (line.GetPosition(i).y - xoffset)/
                 Mathf.Abs((line.GetPosition(i).y - xoffset));

            tempCube.transform.localEulerAngles = new Vector3(0,0,xAngle);

            cube.Insert(i, tempCube);
            angle += (360f / segments);

        }
```

3.5 Mesh Drawing

In order to be able to complete iCraete's original task of giving the usuer the power to create complex 3d objects seamlessly, we will be utilizing meshes so that we can later extrude them or spawn objects.

The AddPoint() and AddLine() functions create quads and trinagles, both front facing and back facing, to create a mesh line. This will help us move the project further so that we can create mesh objects and extrude them later, adding the main functionality of the project.

```
public void AddPoint(Vector3 point)
    {
        if (s != Vector3.zero)
        //if (firstQuad)
        {
            AddLine(ml, MakeQuad(s, point, lineSize, firstQuad));
            firstQuad = false;
        }

        s = point;
    }

    Vector3[] MakeQuad(Vector3 s, Vector3 e, float w, bool all)
    {
        w = w / 2;

        Vector3[] q;
        if (all)
        {
            q = new Vector3[4];
        }
        else
        {
            q = new Vector3[2];
        }

        Vector3 n = Vector3.Cross(s, e);
        Vector3 l = Vector3.Cross(n, e - s);
        l.Normalize();

        if (all)
        {
            q[0] = transform.InverseTransformPoint(s + l * w);
            q[1] = transform.InverseTransformPoint(s + l * -w);
            q[2] = transform.InverseTransformPoint(e + l * w);
            q[3] = transform.InverseTransformPoint(e + l * -w);
        }
        else
        {
            q[0] = transform.InverseTransformPoint(s + l * w);
            q[1] = transform.InverseTransformPoint(s + l * -w);
        }
        return q;
    }

    void AddLine(Mesh m, Vector3[] quad)
    {
        int vl = m.vertices.Length;

        Vector3[] vs = m.vertices;
        vs = resizeVertices(vs, 2 * quad.Length);

        for (int i = 0; i < 2 * quad.Length; i += 2)
        {
            vs[vl + i] = quad[i / 2];
            vs[vl + i + 1] = quad[i / 2];
        }

        Vector2[] uvs = m.uv;
```

```
uvs = resizeUVs(uvs, 2 * quad.Length);

if (quad.Length == 4)
{
    uvs[vl] = Vector2.zero;
    uvs[vl + 1] = Vector2.zero;
    uvs[vl + 2] = Vector2.right;
    uvs[vl + 3] = Vector2.right;
    uvs[vl + 4] = Vector2.up;
    uvs[vl + 5] = Vector2.up;
    uvs[vl + 6] = Vector2.one;
    uvs[vl + 7] = Vector2.one;
}
else
{
    if (vl % 8 == 0)
    {
        uvs[vl] = Vector2.zero;
        uvs[vl + 1] = Vector2.zero;
        uvs[vl + 2] = Vector2.right;
        uvs[vl + 3] = Vector2.right;

    }
    else
    {
        uvs[vl] = Vector2.up;
        uvs[vl + 1] = Vector2.up;
        uvs[vl + 2] = Vector2.one;
        uvs[vl + 3] = Vector2.one;
    }
}

int tl = m.triangles.Length;

int[] ts = m.triangles;
ts = resizeTriangles(ts, 12);

if (quad.Length == 2)
{
    vl -= 4;
}

// front-facing quad
ts[tl] = vl;
ts[tl + 1] = vl + 2;
ts[tl + 2] = vl + 4;

ts[tl + 3] = vl + 2;
ts[tl + 4] = vl + 6;
ts[tl + 5] = vl + 4;

// back-facing quad
ts[tl + 6] = vl + 5;
ts[tl + 7] = vl + 3;
ts[tl + 8] = vl + 1;

ts[tl + 9] = vl + 5;
ts[tl + 10] = vl + 7;
ts[tl + 11] = vl + 3;
```

```
        m.vertices = vs;
        m.uv = uvs;
        m.triangles = ts;
        m.RecalculateBounds();
        m.RecalculateNormals();
    }

    Vector3[] resizeVertices(Vector3[] ovs, int ns)
    {
        Vector3[] nvs = new Vector3[ovs.Length + ns];
        for (int i = 0; i < ovs.Length; i++)
        {
            nvs[i] = ovs[i];
        }

        return nvs;
    }

    Vector2[] resizeUVs(Vector2[] uvs, int ns)
    {
        Vector2[] nvs = new Vector2[uvs.Length + ns];
        for (int i = 0; i < uvs.Length; i++)
        {
            nvs[i] = uvs[i];
        }

        return nvs;
    }

    int[] resizeTriangles(int[] ovs, int ns)
    {
        int[] nvs = new int[ovs.Length + ns];
        for (int i = 0; i < ovs.Length; i++)
        {
            nvs[i] = ovs[i];
        }

        return nvs;
    }
```

3.6 VR Environment

Process:
In order to test the VR Environment, we entered each scene and insured that the user was able to move around the environment using the controllers, as well as spawn individual blocks at the press of the trigger.

Result:
These two features of movement and block spawning was working correctly and as desired.

3.7 Curves

Process:
To test this feature without using the VR equipment, we assigned the scripts to a game object with specific size parameters provided, and then ran the scene. The parameters were taken in by the script and created the desired algorithmic curve.

To test this feature with using VR equipment, we assigned each curve a button on the menu that is attached to the users' controller, allowing them to select which curve they want. When they press the button that is associated with a curve, that curve is then generated at the current location.

Result:
This feature works correctly, with both forms of testing ensuring that the scripts are working correctly.

3.8 Transformation and Translation

Process:
We tested the transformation and transalation of the objects by using the sliders in one of the diegetic menus to resize, reposition, and extrude the object.

Result:
The feature works well, the user is able to propperly resize and exrude objects, and objects can be grabbed and moved around.

3.9 Save and Load

Process:
In order to test the project's save and load functionality initially, we place 3D objects in a game scene and ran the save and load scripts in regards to these 3D objects.

Result:
The scripts acted properly in the ways the save and load scripts were suppose to run. Corresponding files were created and read from correctly, reacting to object in a game scene, whether that be by developers directly placing 3D object within the game environment or by users generating 3D objects and structures with the team's program.

4 Expo Poster

The following is the final version of the group's 2018 Oregon State University Engineering Expo poster, that provides a brief summary, explanation, and description of the 3D Generative Design in Virtual Reality group 61 CS 463 senior capstone project.

COLLEGE OF ENGINEERING | Electrical Engineering and Computer Science | CS61

STORY BEHIND THE PROJECT

Today's computer systems utilize many forms of user interfaces that allow users to seamlessly interact with their electronic devices.

Alternative methods of user input and interfaces are becoming more popular, creating a basis for a new generation of user interfaces for architectural and industrial designing.

OUR VISION

Goal: To improve the efficiency of the interaction between the user and the program via multiple innovative modalities.

The team has used the Unity game engine and Steam VR's virtual reality plugin to develop the program that puts the power of creation in your hands. Literally.

Simply pick up the HTC Vive controllers and headset to bring your imagination to life!

Oregon State University

DESIGNING IN VIRTUAL REALITY

Generative 3D Design in Architecture

1 HTC Vive Headset and Controllers

2 Generate 3D Object

3 Resize 3D Object

4 Draw Trajectory

5 Watch Structure Become [Virtual] Reality!

DESIGNING EXPLANATION

Power of Unity + Experience of VR + C# = An intuitive tool to create complex structures out of simple gestures and ideas

1) 3D objects can be spawned by selecting from the in-game menu via an HTC Vive controller.

2) Objects can be resized, combined, and altered according to the user's whim.

3) Curves can also be drawn mathematically, and then changed into various 3D structures.

4) Save and load objects and environments to come back to or continually cherish creations.

WHAT HAPPENED IN THE END?

Usable HTC Vive Compatible VR Program!

Successes:

Program meets basic requirements of generating various 3D objects and creation of trajectories with mathematically curves, along with save and load functionalities.

Limitations:

User is unable to free-draw a curve and delete objects within a scene (program restart is needed), but these additions are in the works for the future.

(Pictured from Left to Right)

Nabeel Shariff, Hannah Solorzano, Rhea Mae Edwards

Raffaele de Amicis

Associate Professor at Oregon State University, School of Electrical Engineering and Computer Science, focus in research in Computer Graphics and Visualization
raffaele.deamicis@oregonstate.edu

Nabeel Shariff

Computer Science Student focus in Business Entrepreneurship
shariffn@oregonstate.edu

Hannah Solorzano

Computer Science Student focus in Computer Graphics and Game Simulation
solorzah@oregonstate.edu

Rhea Mae Edwards

Computer Science Student focus in Computer Systems
edwardrh@oregonstate.edu

Figure 2. CS 463, Group 61, 3D Generative Design in Virtual Reality 2018 Oregon State University Engineering Expo Poster