Memoria de prácticas

por Adrián Ferrando Mazarro

1. Introducción

En el presente texto se procederá a exponer el desarrollo de la estancia en prácticas en la empresa Virtual Trips. Se tratará de enfocar no solo el trabajo realizado a lo largo de la duración de la misma, sino cómo me ha afectado a nivel formativo en el desarrollo de mi experiencia laboral.

2. Descripción de la empresa

Virtual Trips es una empresa localizada en la ciudad de Valencia, en pleno barrio de Ruzafa (**FIG 1**).

FIG 1. El local se sitúa en una calle del centro del barrio de Ruzafa

Virtual Trips es una subempresa de Mind Trips, la cual es una empresa de *Escape Room*. Estas empresas de ocio encierran al usuario en una sala *virtualmente* cerrada (*virtualmente* ya que el usuario puede salir en cuanto lo decida). El objetivo es salir de la sala resolviendo una serie de acertijos o enigmas mediante la habilidad y/o el ingenio del usuario. Normalmente estas salas presentan una diégesis y un estilo concretos, tratando de proporcionar a su vez una experiencia narrativa.

A diferencia de las salas físicas, Virtual Trips intenta ofrecer en este caso salas o experiencias de carácter virtual empleando tecnologías como la realidad virtual, como bien indica su nombre. Estas experiencias permiten un mayor abanico de posibilidades que las salas físicas y a un muy menor coste de desarrollo, ya que no se elaboran medios físicos mediante los cuales interactuar con el usuario más allá de los *inputs* clásicos de las experiencias digitales como son los *joysticks*.

La empresa es un pequeño negocio que cuenta con cuatro trabajadores, uno de los cuales fue con quien trabajamos en el desarrollo y antiguo alumno del grado que curso.

3. Descripción del plan de trabajo inicial

FIG 2. El imponente castillo de Mora de Rubielos, de origen musulmán.

El planteamiento del trabajo era sencillo: se desarrollaría una experiencia virtual de realidad aumentada de un museo virtual. Este museo, una vez desarrollado el prototipo, se propondría al ayuntamiento de Mora de Rubielos, el cual ya había mostrado interés por el proyecto. Se planteó realizar este tipo de proyecto ya que el pueblo posee un imponente castillo (**FIG 2**) el cual, sin embargo, carece de elementos interesantes en su interior debido a diferentes causas históricas. Por ello, la experiencia de realidad aumentada permitiría el desarrollo de un interior con elementos históricos ya desaparecidos como armaduras o espadas de la época sin tener que emplear grandes sumas económicas en fabricarlos físicamente (**FIG 3**).

FIG 3. Target tamaño persona del que sale el caballero, contando la historia del castillo.

El plan de trabajo se planteó de la siguiente manera: se realizaría un prototipo para presentarlo el día 20 de mayo. Esta fecha fue escogida ya que era el lunes anterior a las elecciones municipales del 26 de mayo, lo que interesaba al alcalde de cara a la campaña electoral. Las jornadas de trabajo serían de 9:00 a 15:00 horas con un pequeño descanso para almorzar. Esto fue así inicialmente, aunque cuando se fue acercando la fecha de entrega se tuvo que realizar durante una semana jornadas de 9:00 a 17:00 con tal de acabarlo a tiempo. Esto fue consensuado entre la empresa y los estudiantes en prácticas, ya que a los dos nos interesaba acabar lo antes posible.

Una vez acabado el proyecto, se comenzó a realizar otro: un *Escape Hall*, que no es más que una *Escape Room* que por su construcción (normalmente en forma de una caja no muy grande) es fácilmente transportable a ferias o actos lúdicos fuera de donde se sitúa la empresa. Este *Escape Hall* emplearía nuevamente la tecnología de realidad aumentada y trabajaríamos bajo la misma jornada que en el proyecto anterior hasta que completásemos las horas que teníamos que emplear en el desarrollo de la asignatura.

Sin embargo, una vez llegando al final de nuestra estancia en prácticas, solicité a la empresa la posibilidad de aplazar las horas que me quedaban hasta después de realizar el Trabajo de Final de Grado, ya que sin el tiempo extra que me proporcionaría el detener temporalmente las prácticas no hubiese podido finalizarlo. Desde la empresa no me pusieron problemas y, por cuestiones laborales y de disponibilidades, acabé realizando mis últimas horas en septiembre realizando una herramienta de dibujo de cables.

4. Trabajo realizado

Entré en la empresa de prácticas en calidad de programador. Sin embargo, por las características del proyecto, acabé ejerciendo los papeles tanto de modelador adjunto como de texturizador. Mi trabajo, en general, se centró en la realización de efectos, lógica de la aplicación y la creación de texturas realistas (**FIG 4**).

FIG 4. Render del caballero finalizado hecho desde el programa de texturizado Substance Painter.

En lo que respecta al modelado, mi trabajo principal fue asesorar al modelador principal en el funcionamiento de los sistemas de físicas que nos permitieron realizar las ondulaciones de la ropa de manera realista, además de realizar el modelado de elementos menores del proyecto como el cinturón y demás detalles. Para realizar este modelado, empleamos de manera íntegra 3DS Max, ya que es el programa en el que más cómodos nos encontrábamos por haber trabajado con él durante toda la carrera.

Las texturas fueron realizadas en Substance Painter, que si bien no es estudiado en la carrera, lo aprendí a utilizar de manera autodidacta ya que es el estándar más usado tanto en la industria del videojuego como en la del cine, y permite crear texturas extremadamente realistas sin el esfuerzo extra que implicaría utilizar programas de edición de imágenes clásicos como Photoshop.

En la cuestión de la programación, me encargué de profundizar en la API de Vuforia, que permite usar de manera sencilla técnicas de realidad aumentada en el motor de juego de Unity. Usando una imagen, se puede visualizar cualquier objeto a voluntad del programador una vez la cámara del móvil capta el *target* (**FIG 5**).

FIG 5. Imagen del caballero en una versión temprana de la aplicación

Las imágenes que se usan para hacer de *target* tienen que cumplir ciertas propiedades para optimizar la velocidad y calidad del reconocimiento por parte del programa, que se basa al completo en la información captada por la cámara para funcionar. Algunas de las más importantes son un alto contraste de color, ángulos rectos en preferencia a contornos curvos y falta total de simetría. Con esto en mente, diseñamos unos targets propios que pudiésemos usar en el interior de un castillo con una iluminación que podía no ser la más adecuada.

Como se puede observar en las (**FIG 6**), el interior de la imagen está destinado a ser identificado por el usuario, de manera que entienda qué es el cartel y cómo se usa dentro de la aplicación, entre otras cuestiones. Son los bordes los que verdaderamente actúan como *target*. Pese a no usar completamente la imagen, no hubo diferencias sustanciales en la efectividad del reconocimiento por cámara con respecto a la imagen completa.

Esto permite *camuflar* las marcas del texto que presenta el usar una imagen puramente identificativa (como un código QR) en pos de una experiencia más inmersiva y enriquecedora tanto para un público ya relacionado con esta tecnología como para aquellos que tienen más problemas en su uso, como los usuarios de tercera edad.

FIG 6. Dos de los targets diseñados para el castillo. El borde recuerda las cristaleras decorativas de los edificios en la Edad Media.

En cuanto a la programación de efectos, hice un shader para representar hologramas. Se puede ver un ejemplo del mismo shader en la (**FIG 7**). Utiliza coordenadas globales para dibujar las líneas, que van bajando con el tiempo en una animación cuya velocidad es controlable mediante una propiedad. También usa una textura de ruido simple en 3D para darle el efecto granulado variante, que también está animado con el tiempo. Se puede regular tanto la transparencia del objeto (que se consigue haciendo uso de técnicas de *Z-Buffer*) como los componentes *metálico* y *suavidad* del mismo, lo que es básico en cualquier modelo de representación 3D en tiempo real mínimamente realista. Por último, se ha añadido un efecto de fresnel, que dibuja una línea de luz de grosor y color variables en los bordes del objeto en función de la dirección del vector frontal de la cámara. Esto permite darle un aspecto mucho más lumínico, propio de un holograma.

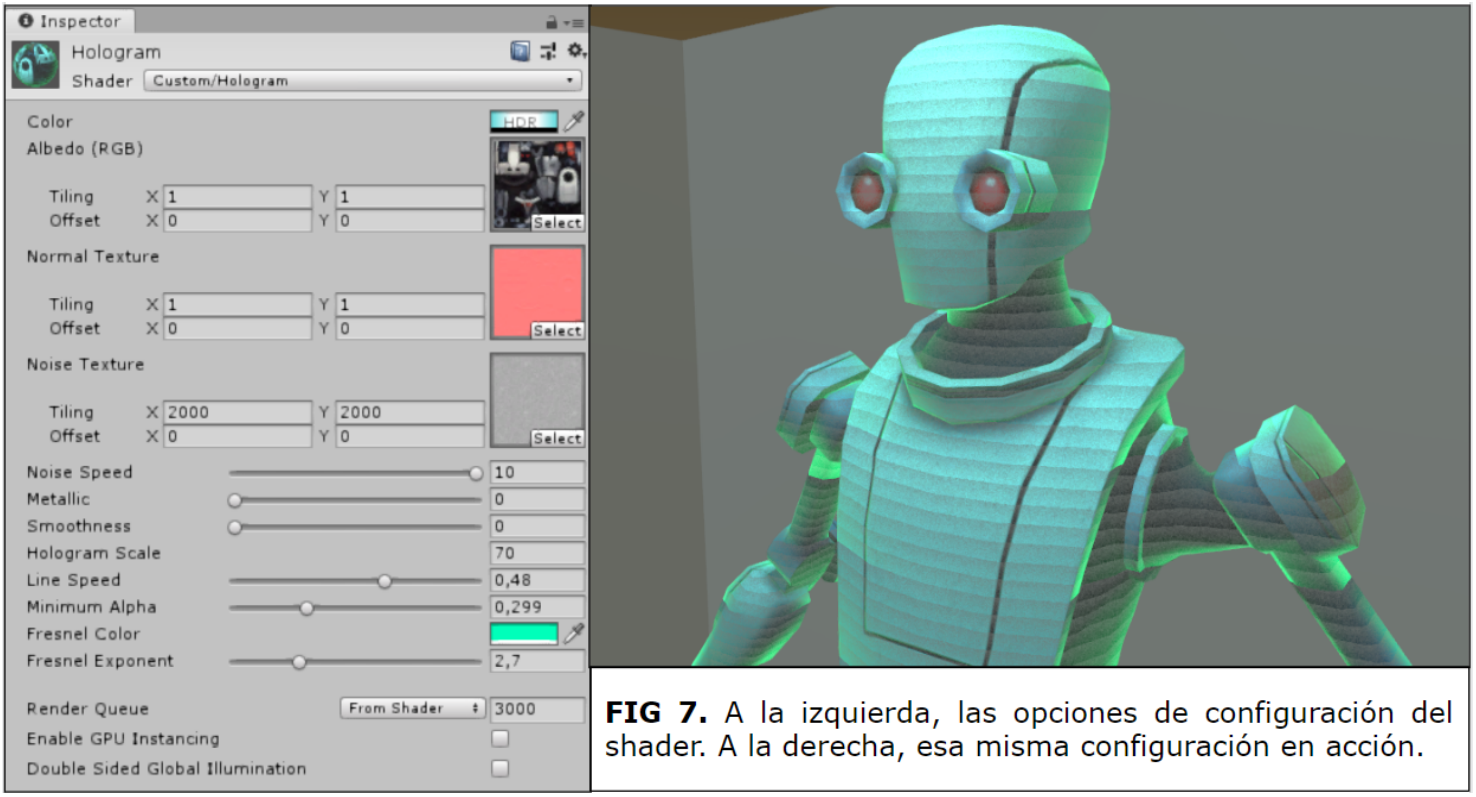


FIG 7. A la izquierda, las opciones de configuración del shader. A la derecha, esa misma configuración en acción.

Finalizándose el proyecto del museo virtual, se puso en marcha el proyecto del *Escape Hall*. En él, el puzzle principal se basaba en una serie de conexiones de cables. Con tal de realizar esto opté por emplear *geometry shaders*, unos pequeños programas que permiten dibujar mallas en la GPU en tiempo real de una manera increíblemente rápida y sólida. Dado un segmento definido por dos vértices, dibuja un *quad* (un cuadrángulo formado por dos triángulos) que vaya del primero al segundo con un determinado grosor definido por el usuario (**FIG 8**). Como se realizó parcialmente a distancia del centro de trabajo, escribí una documentación en inglés que resumía su funcionamiento y que puede encontrarse en el **Apéndice**.

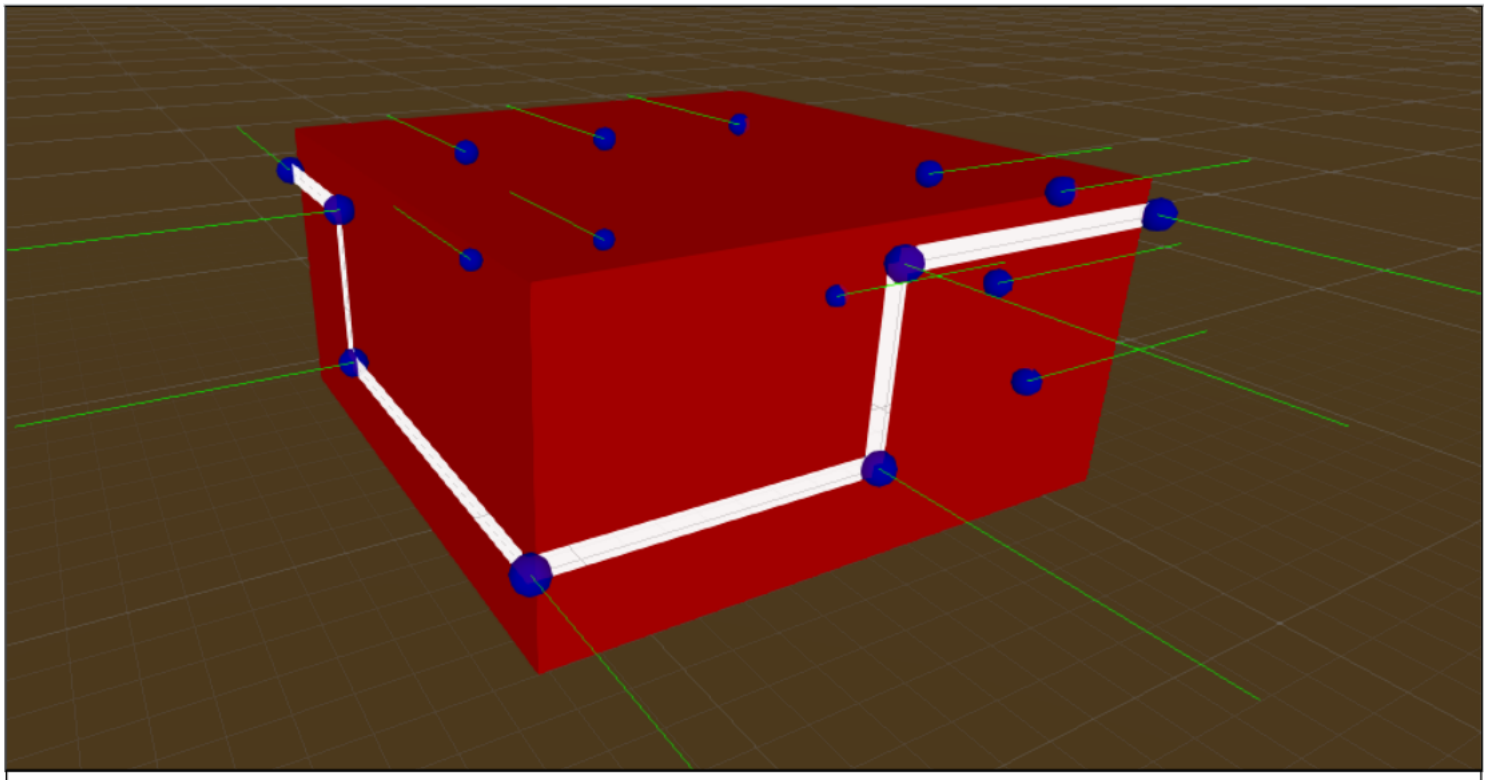


FIG 8. Aspecto de la herramienta. La caja y las esferas están dibujadas usando los *Gizmos* de Unity. Los cables blancos son el resultado del shader de geometría.

5. Conclusiones

Tras pasar mi estancia en prácticas, el balance que hago, si bien no es negativo del todo, no es nada positivo. He aprendido cosas, eso desde luego. He aprendido a coordinarme en un equipo de trabajo con plazos de entrega ajustados. Pero, siendo francos, apenas puede diferenciarse de un trabajo de asignatura con otros compañeros de clase o una *Game JAM* más espaciada en el tiempo. Todo lo que he aprendido a nivel técnico podría haberlo aprendido en trabajos académicos o en mi casa de manera autodidacta. Al final, las prácticas parecen solo una forma más que tienen las empresas de conseguir mano de obra barata. Porque la teoría es muy bonita: todos aprendemos a trabajar en un centro de trabajo. La realidad, sin embargo, es harto diferente. Produces un beneficio a la empresa del cual, pese a haber participado de manera clara, no percibes ni una ínfima parte. Y eso, por algún motivo, se percibe como lo normal.

Considero que esta asignatura como algo que puede llegar a ser extremadamente interesante para finalizar la formación académica del alumno, ya que le confronta con la realidad y forja la teoría aprendida con el fuego de la práctica viva. Sin embargo, su planteamiento hace que muchas empresas intenten aprovecharse para conseguir mano de obra barata (gratis, de hecho), lo que genera dos problemas: estas empresas difícilmente acabarán contratando a alguien, ya que pueden ir supliendo estos puestos con más alumnos en prácticas (sean o no universitarias); estas empresas apenas enseñan nada a los alumnos, ya que estos son vistos más como trabajadores que como estudiantes. Se da una dualidad que beneficia a la empresa que recurre a esto. Es decir, se nos ve como trabajadores de cara a nuestra labor dentro de la empresa, pero como estudiantes en prácticas de cara a nuestro derecho remunerativo.

Y esto ha hecho que, al menos para mí, esta asignatura haya pasado sin pena ni gloria por mi vida, sin ayudarme a mejorar como estudiante y convirtiendo una experiencia que tendría que ser rica en conocimiento e interesante en una experiencia a la que acudía con apatía, con un tesón más propio de quien quiere que algo pase rápido que del que busca aumentar y asentar su conocimiento.

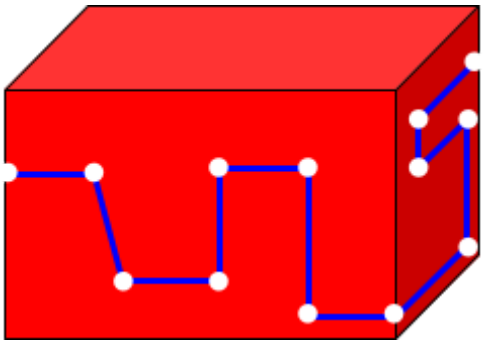
6.Apéndice

Tabla de tareas

Tarea	Descripción	Duración
Creación de un sistema-interfaz para la aplicación	Crear un sistema que permita acceder a cada elemento de la armadura del caballero tocando la figura completa, pudiendo poner y quitar partes	30h
Texturizado del modelado	Crear las texturas que permiten visualizar correctamente el caballero de manera que sea visualmente realista de acuerdo a la evidencia histórica (materiales, símbolos, detalles, etc.)	80h
Estudiar el sistema de *targets* de Vuforia	Estudiar cómo debe ser la imagen usada de *target* en el motor de Vuforia para mejorar al máximo la velocidad y calidad de su detección.	6h
Estudiar tecnologías alternativas para hacer animaciones	Investigar nuevas formas de hacer animaciones de manera que se simplifique el proceso sin necesidad de caros trajes de movimiento.	7h
Modelado de objetos	Modelar detalles del caballero: el cinturón, las armas, el escudo, generar simulaciones de *clothing*.	40h
Programación de efectos	Creación de efectos de partículas y shaders que permitan una mejor visualización de la experiencia: efecto de	50h

	entrada del caballero en la escena, antorchas, efectos de postprocesado, etc.	
Sustitución del sistema-interfaz por una interfaz diegética	Cambiar la vieja interfaz por una interfaz integrada en la escena 3D de la aplicación, de manera que quede integrado con el sistema y facilite su uso por parte de personas de toda índole.	15h
Creación de una herramienta que permita crear puzzles	Implementar una herramienta para Unity que permita crear en este los puzzles de cables de manera sencilla.	50h

WireShader Documentation



1.What it does?

The shader is a vertex-geometry-fragment shader which draws a wire between all the vertices we send to it around a box defined by the user. It can do only lineal wire -each vertex has only two wires connected to it- and it can be oriented to any direction on the plane the vertex lies. It must have one vertex in the corner of the box in order to change the direction of the wire to the perpendicular plane -the next or the previous plane-.

2. How it does?

The SetVertices() function takes a Vector3 array and converts it into a line topology mesh and sends it to the shader manager -the GeometryShader script-. The function also calculates the *normal* of each vertex, which is the vector pointing outside the box, perpendicular to the plane and intersecting the vertex. This will serve to calculate the orientation of the wire quad that will be draw by the shader.

```
public void SetVertices(Vector3[] vertices)
{
    Mesh m = new Mesh();
    List<Vector3> newVertices = new List<Vector3>();
    List<int> indices = new List<int>();
    List<Vector3> normals = new List<Vector3>();
    //int[] indices = new int[vertices.Length];
    for (int i = 0; i < vertices.Length; i++)
    {
        if (i > 1) indices.Add(i - 1);
        Vector3 normal = CheckNormalSide(vertices[i]);
        if(Mathf.Abs(normal.x) == 1)
        {
            normals.Add(new Vector3(normal.x, 0, 0));
            Debug.DrawLine(vertices[i], vertices[i] + normals[normals.Count - 1], Color.green);
            newVertices.Add(transform.InverseTransformPoint(vertices[i]));
            indices.Add(newVertices.Count-1);
        }
        else if (Mathf.Abs(normal.z) == 1)
        {
            normals.Add(new Vector3(0, 0, normal.z));
            Debug.DrawLine(vertices[i], vertices[i] + normals[normals.Count - 1], Color.green);
            newVertices.Add(transform.InverseTransformPoint(vertices[i]));
            indices.Add(newVertices.Count-1);
        }
    }
    m.vertices = newVertices.ToArray();
    m.SetNormals(normals);
    m.SetIndices(indices.ToArray(), MeshTopology.Lines, 0);
    if (this.GetComponent<GeometryShader>() == null) this.gameObject.AddComponent<GeometryShader>().SetMesh(m, _Width);
    else GetComponent<GeometryShader>().SetMesh(m, _Width);
}
```

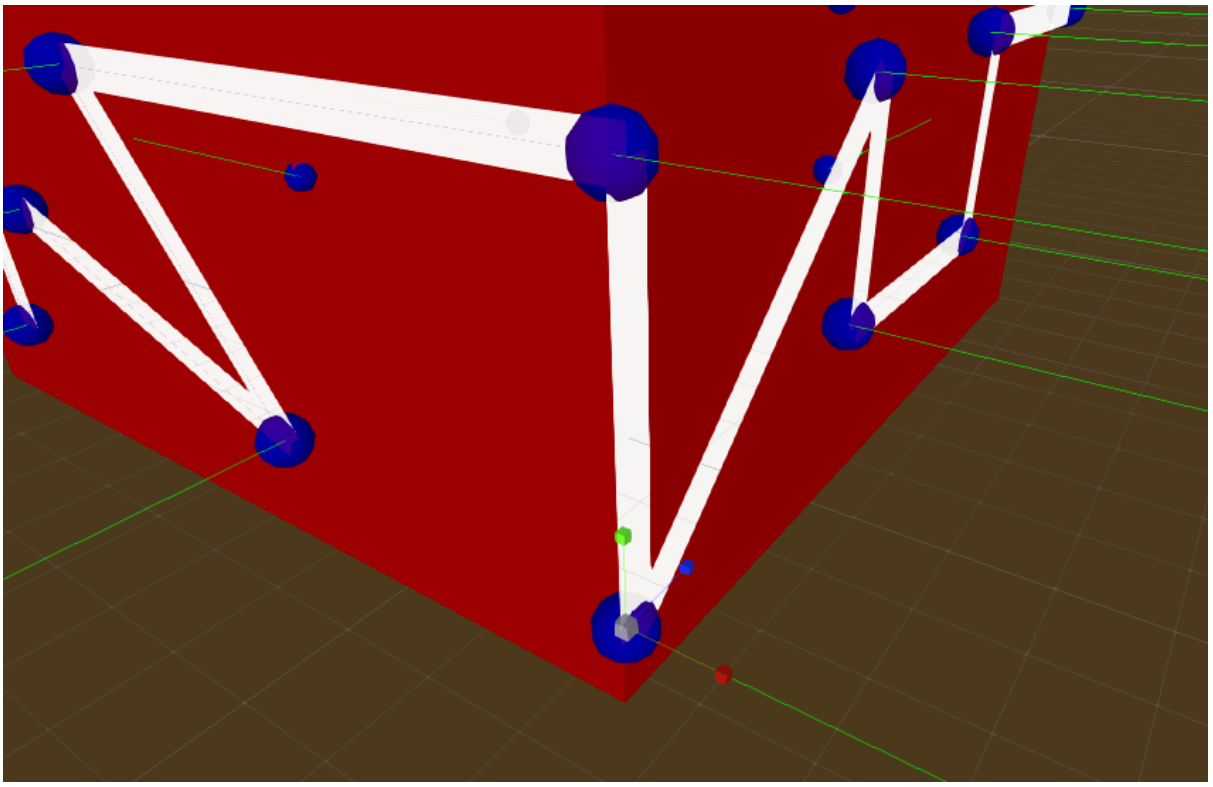
The GeometryShader script takes the info of the wire (width, mesh, etc.) and sends it to the GPU. It also checks some configuration of the GPU Emulation and sets it to the right setting.

```
public class GeometryShader : MonoBehaviour
{
    float width;
    2 referencias
    public void SetMesh(Mesh mesh, float widthHeight) //width = square size : height = wire width
    {
        GetComponent<MeshFilter>().sharedMesh = mesh;
        Shader shader = Shader.Find("Unlit/WireShader");
        Material mat = new Material(shader);
        GetComponent<MeshRenderer>().sharedMaterial = mat;

        if (!shader.isSupported)
        {
            Debug.LogError("LEE ESTO ES IMPORTANTE: si no está funcionando el shader, pon Edit>GraphicsEmulation>None");
            EditorApplication.ExecuteMenuItem("Edit/Graphics Emulation/No Emulation");
        }
        if (mat.shader != shader)
        {
            mat = new Material(shader);
        }
        mat.SetFloat("_Width", width);
        this.width = widthHeight;
    }
}
```

Then the shader takes place. We will skip the vertex and the fragment shaders because they are trivial, so we are seeing the geometry shader. Here we calculate the face direction of the wire quad and we generate it using the _Width property for each line of the mesh. The crossProduct is calculated using different techniques depending on the orientation of the line we are working with -vertical or non-vertical-.

```
//_Distance/_WidthHeight.r --> number of quads in one side | 6 --> vertex per quad | 2 --> both sides
[maxvertexcount(12)]
void geom(line v2g input[2], inout TriangleStream<g2f> triStream)
{
    g2f o;
    v2g first = input[0];
    v2g last = input[1];
    float3 firstToLast = last.vertex-first.vertex;
    float3 normal = last.normal;
    float3 crossProduct;
    if(firstToLast.x == 0){

    }
    if(firstToLast.x==0 && firstToLast.z == 0){
        crossProduct = normalize(cross(normal, firstToLast)) * _Width/2;
    }else{
        crossProduct = cross(cross(float3(0,1,0), firstToLast), firstToLast);
        crossProduct=normalize(crossProduct)*_Width/2;
    }

    //quad 1
    o.vertex = UnityObjectToClipPos(first.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(first.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(last.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);

    triStream.RestartStrip();

    o.vertex = UnityObjectToClipPos(last.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(last.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(first.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);

    triStream.RestartStrip();
    //quad 2
    o.vertex = UnityObjectToClipPos(first.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(last.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(first.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);

    triStream.RestartStrip();

    o.vertex = UnityObjectToClipPos(last.vertex - crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(first.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);
    o.vertex = UnityObjectToClipPos(last.vertex + crossProduct + float3(_Width/4,0,0));
    triStream.Append(o);

    triStream.RestartStrip();
}
```

3. Some problems



If the line is on the intersection of two planes of the box, the quad generated will always be parallel to the plane whose normal is in the X axis. This can be troubleshooted by not setting the vertex exactly on the corner, but a bit after or before that corner.