Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación CAD para la configuración modular de muebles.

Proyecto Final de Carrera

Ingeniería Informática

Autor: Borja Herráez Concejo

Director: Eduardo Vendrell Vidal

Codirector: Carlos Sánchez Belenguer

Tabla de contenidos

1. INTRODUCCIÓN

El objetivo de este proyecto final de carrera se centra en la realización de un configurador modular de muebles, enfocado al diseño de los mismos a través de la aplicación desarrollada.

Más detalladamente los objetivos principales son por un lado la correcta realización de la aplicación que permita de manera sencilla el diseño de muebles de distintas categorías (sillas, mesas, armarios, etc.), empezando por una pieza base predefinida y añadiendo piezas de forma incremental, de manera que el configurador pueda ser usado tanto por gente acostumbrada al uso de aplicaciones y ordenadores como por gente que no está acostumbrada al manejo de los mismos y así facilite el diseño de muebles a un amplio rango de personas, para conseguir esto se pretende realizar una interfaz de usuario amigable que guíe al usuario ayudándose de imágenes y que hagan un entorno de desarrollo sencillo para conseguir resultados en poco tiempo.

Por otro lado diseñar una pequeña base de datos que sea capaz de gestionar la información de la aplicación y aprender a implementar esta en el servidor, aprendiendo también el uso de herramientas básicas para ello que permitan entre otras cosas consultar el estado de las tablas o realizar consultas SQL.

En el ámbito de los configuradores existen ya diferentes tipos de aplicaciones utilizadas por compañías como por ejemplo el “IKEA Home Planner” la aplicación de la compañía “IKEA”, la aplicación “CGBOX”, en este caso es un “configurador de habitaciones” que permite salvar las composiciones personalizadas de cada usuario o la aplicación “Sweet Home 3D” bajo la licencia GNU General Public License. De estas y otras aplicaciones se hablará con más detalle en puntos posteriores.

En comparación con los configuradores mencionados en este caso no se pretende coger un mueble y situarlo en un espacio para diseñar una habitación, sino que se trata de bajar un nivel y diseñar muebles a nivel de pieza de forma que sean totalmente

personalizables según las necesidades del usuario, con lo que en realidad se podría crear cualquier tipo de objeto con conexiones simples.

En cuanto al desarrollo del caso, la principal problemática radica en la definición de un modelo de conectividad que permita la correcta colocación de las diferentes piezas o módulos del mueble, de manera que se tengan en cuenta tanto la posición como el tipo de pieza a colocar, no siendo posible por ejemplo, la colocación de una pata de una silla en el lugar donde debería ir el respaldo con lo que habrá que establecer una serie de restricciones que hagan posible esto.

Una vez definido este modelo de conexión (que se explicará en apartados siguientes de este documento) se cuenta con 2 herramientas.

Para la definición de las posiciones se tiene una herramienta secundaria que permitirá visualizar las piezas y a través de una interfaz gráfica definir la colocación de los puntos de conexión, más tarde la aplicación principal se encargará de comprobar los requisitos de conexión y ofrecerá la colocación sólo de las piezas que los cumplan.

La aplicación principal permitirá seleccionar el tipo de mueble a diseñar, ofrecerá una pieza base inicial con la que empezar a desarrollar el mueble, y a partir de ahí se irán añadiendo piezas y modificándolas hasta obtener el modelo final deseado.

Tanto los modelos como las imágenes utilizadas se almacenarán en un servidor externo en el que también se encontrará la base de datos diseñada para la aplicación. Las consultas a la base de datos se harán a través de ficheros “PHP” que devolverán ficheros “XML” de los que sacaremos la información deseada según el caso. Así pues se necesitará de una conexión a internet para utilizar la aplicación, con lo que esta se podrá distribuir en la mayoría de dispositivos actuales como Smartphones, Tablets y ordenadores personales.

Los objetos 3D de las piezas se han modelado con la herramienta “3DstudioMAX” usando funcionalidades básicas, no

obstante mayor complejidad de los modelos solo afectaría en los tiempos de descarga y latencia según los recursos disponibles de la máquina utilizada, pero no en la funcionalidad de la aplicación, que seguirá siendo la misma.

2. MODELADORES GEOMÉTRICOS

2.1.¿QUE SÓN?

El principal objetivo de un “modelo” es obtener información sobre el objeto representado a partir de ese modelo. Gracias a los modelos se es capaz de comprender el comportamiento y estructura de los objetos a modelar, aparte también se pueden realizar experimentos y con estos predecir y visualizar los efectos de modificaciones sin necesidad de hacerlo con el objeto real, cosa que supondría más coste monetario y en ocasiones resulta imposible físicamente. En el caso de la informática gráfica, disponer de modelos permite sintetizar imágenes para visualizar escenas.

No obstante la principal problemática de los “modelos” es que representar la realidad resulta demasiado complejo en la mayor parte de los casos, con lo se recurre a simplificaciones que alejan del comportamiento real del objeto, cuanto más se asemeje el modelo a la realidad, mejores serán los resultados que se obtendrán.

Un modelador geométrico es una facilidad software que permite la creación del modelo geométrico de una entidad, esto es su representación digitalizada cuyas características básicas pueden describirse a través de su geometría. De esta manera se hace que las propiedades del objeto sean más fáciles de observar que en el objeto real.

2.2. CLASIFICACIÓN MODELADORES GEOMÉTRICOS

2.2.1.-MODELADORES ALÁMBRICOS

La información que hay del objeto es un conjunto de líneas que representan las aristas (delimitadas por vértices) que forman el objeto, lo que permite construir el "esqueleto" del objeto. Esta información se almacena en dos tablas, la tabla de vértices y la tabla de aristas. En la tabla de vértices se indica para cada vértice sus coordenadas (x, y, z) en el espacio tridimensional. En la tabla de aristas se relaciona cada arista con su vértice inicial y su vértice final, utilizando la misma notación para cada vértice que la usada en la tabla anterior.

La principal ventaja de los modelos de alambre es que su sencillez permite generar imágenes muy rápidamente y el bajo coste computacional que supone. Las entidades alámbricas son las entidades básicas de cualquier sistema y existen múltiples formas de definirlas y crearlas: Coordenadas cartesianas, cilíndricas, coordenadas absolutas o incrementales, etc.

Por otro lado entre sus desventajas están la representación ambigua y sin coherencia visual, se pierden las líneas de silueta, pueden representar modelos imposibles y sin sentido. Todos estos problemas vienen derivados de la representación de un modelo tridimensional en un espacio bidimensional, en este caso la pantalla del ordenador.

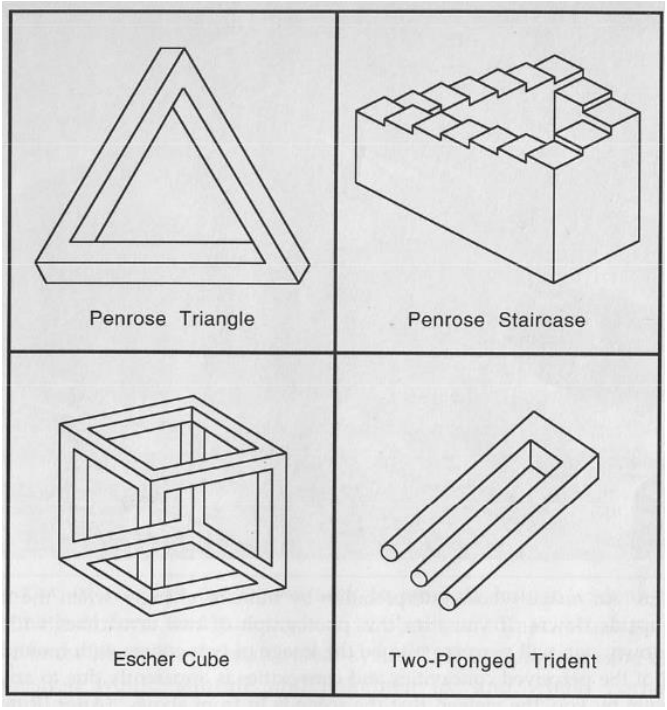

Figura 1. Modelados imposibles.

Fuente: División de Arquitectura de Computadores de la Universidad Rey Juan Carlos.

2.2.2.-MODELADO DE SUPERFICIES

Se basan en la estimación de los valores de una superficie en cualquier punto de esta, a través de un grupo de datos de muestreo (x, y, z) denominados puntos de control. Tiene aplicaciones en numerosos campos como la geología, la meteorología o la medicina entre otros.

Existen dos maneras de representar la superficie externa de los objetos:

En la primera un objeto se representa con una lista de “caras” o facetas, detalladas por los lados y las aristas que las delimitan. La lista de caras puede contener informaciones geométricas propias de cada faceta (tamaño, posición respecto al origen, etc.) o pueden estar estructuradas de manera más compleja, por ejemplo en forma de gráficas o nodos que representan la relación entre caras, aristas y lados en forma de árbol.

Otra manera de representar las superficies externas del objeto es empleando superficies de “forma libre”, que el usuario manipula a través de los “puntos de control”. Se utilizan superficies representadas mediante ecuaciones paramétricas que realizan la aproximación de la envoltura exterior del objeto. Estas ecuaciones dan como resultado una malla de elementos finitos de forma específica (cuadrados o triángulos normalmente) y utilizan puntos característicos para cambiar la forma final de la superficie. El modelo algebraico describe un sólido a partir de su frontera.

Para que el sólido sea representable su frontera debe ser representable por un polinomio de grado infinito. Este modelo permite utilizar representaciones de los sólidos basadas en el almacenamiento de la frontera, que es una entidad bidimensional.

El modelado con superficies puede obtener una mayor precisión geométrica a la hora de obtener modelos complejos.

2.2.3.-MODELADO SOLIDO

Es un tipo de modelado geométrico que hace más hincapié en la aplicabilidad de los modelos, busca crear representaciones “completas” de objetos físicos sólidos del mundo, entendiendo por “completas” que permiten distinguir entre interior, exterior y superficie de un objeto así como incluir la información geométrica externa y la estructura interna del objeto, lo que permite simular sobre ellos procesos físicos y calcular propiedades físicas como por ejemplo el peso del objeto, su volumen, masa, etc.

Los modelos de representación de solidos deben cumplir en la medida de lo posible una serie de requisitos que se enumeran a continuación:

- Precisión: Cuanto más preciso es el modelo, más cercano a la realidad. Un modelo preciso permite representar un objeto como es en el mundo real sin necesidad de aproximaciones.
- Dominio: Hace referencia al conjunto total de objetos que pueden representarse con el modelo.

- No ambigüedad: Una representación debe corresponder a un sólido u sólo a uno.
- Unicidad: Un sólido solo se codifica de una única forma.
- Validez: Debe asegurar que no se creen representaciones no validas de sólidos.
- Cierre: Operaciones sobre modelos válidos dan como resultado nuevos modelos válidos.
- Compacta: Reducir el espacio de almacenamiento, mejorando así el rendimiento del sistema.
- Eficiencia: Algoritmos eficientes en el cálculo de las propiedades físicas de los sólidos, así como su representación en pantalla.

El diseño de una representación que cumpla todas estas características es muy complicado y en la mayoría de casos hay que llegar a un compromiso de equilibrio entre estas propiedades.

2.3.METODOLOGÍAS MODELADO SÓLIDO

2.3.1.-MODELADO SÓLIDO MEDIANTE BARRIDOS.

Al desplazar una primitiva bidimensional a lo largo de una trayectoria tridimensional por el espacio se “barre” una región que define un objeto nuevo. Este proceso se conoce con el nombre de desplazamiento traslacional o extrusión.

Otro tipo de modelado de barrido son los desplazamientos rotacionales o de revolución, en este caso se define el objeto mediante la rotación de un área sobre un eje.

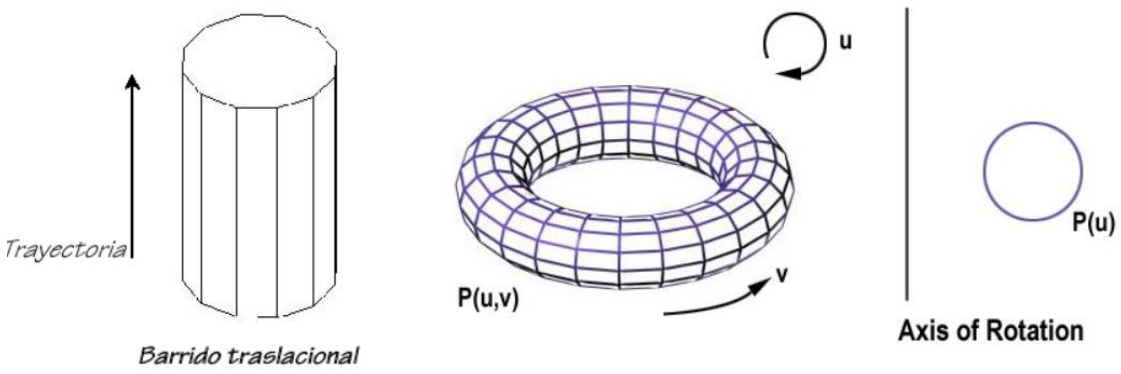

Figura 2. Modelados por barrido.

Fuente: Introducción a la Computación Gráfica. Facultad de Ingeniería de la Universidad de la República – Uruguay.

Es una representación muy útil y versátil para la construcción de objetos que tienen simetrías rotacionales, traslacionales o de otros tipos.

Los desplazamientos donde el área o el volumen que generan cambian de tamaño, forma u orientación y que siguen una trayectoria curva arbitraria se denominan barridos generales. Son muy difíciles de generar de forma eficiente y no siempre generan sólidos, tienen un dominio limitado y muchos de los objetos que generan no tienen utilidad.

2.3.2.-MODELADOS SOLIDO MEDIANTE INSTANCIACIÓN Y PARAMETRIZACIÓN

Estos métodos definen las primitivas mediantes semiespacios, entidades sin límites geométricos que dividen el espacio en 2 partes infinitas, dentro y fuera (in, out). Los elementos se definen mediante ecuaciones implícitas lo que hace fácil cosas como el cálculo de intersecciones pero dificulta el entendimiento de los sólidos. Es la representación de más bajo nivel, es posible que genere solidos abiertos.

2.3.3.-MODELADOS DE DESCOMPOSICIÓN ESPACIAL

Representan objetos sólidos mediante la unión de un conjunto de celdas disjuntas sin agujeros, las celdas pueden tener caras, aristas y/o vértices comunes. A mayor diversidad de células, crece el dominio de objetos. La representación del sólido es la lista de celdas que ocupa, no es más que estudiar si las celdas están ocupadas (total o parcialmente) o vacías. Los métodos de descomposición pueden estructurar las celdas de manera jerárquica, lo que ayuda a localizar información y a compactar la representación del modelo (octrees o arboles de partición binaria).

El primer método que podemos encontrar es la **Enumeración de ocupación espacial**, en él el espacio donde reside el sólido se divide en cubos de igual tamaño denominados "voxel".Por cada voxel se almacena la información de ocupación del objeto, es decir, si el voxel pertenece o no al sólido. La estructura de almacenamiento es simplemente un array 3D de booleanos. No existe el concepto de ocupación parcial con lo que los objetos con superficies curvas sólo pueden aproximarse (falta de precisión).

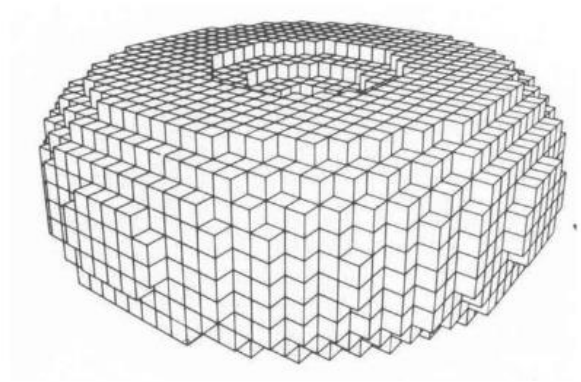

Figura 3. Ejemplo malla ocupación espacial.

Fuente: Sandra Baldassarri(2013-2014). Departamento de Informática e Ingeniería de Sistemas (DIIS). Universidad de Zaragoza.

Las celdas pueden ser tan pequeñas, como se desee, ahora bien si aumenta el número de celdas que componen la malla, también aumenta el espacio de almacenamiento.

Por otro lado se tienen los **arboles de octantes (octree)** que es una variante jerárquica de la enumeración de ocupación espacial en la que se pretende reducir el coste espacial de esta. Se forma una estructura de datos de tipo “árbol” en la que cada nodo tiene exactamente 8 “hijos”, cada nodo es un cubo que podrá estar ocupado por el sólido, vacío o parcialmente ocupado. Para los octantes parcialmente ocupados se hace de manera recursiva la misma división en octantes y así de forma repetida hasta alcanzar el nivel de precisión deseado. De esta manera se puede hacer una gestión más eficiente del tamaño de las celdas con lo que se reduce el coste espacial usando más eficazmente la memoria.

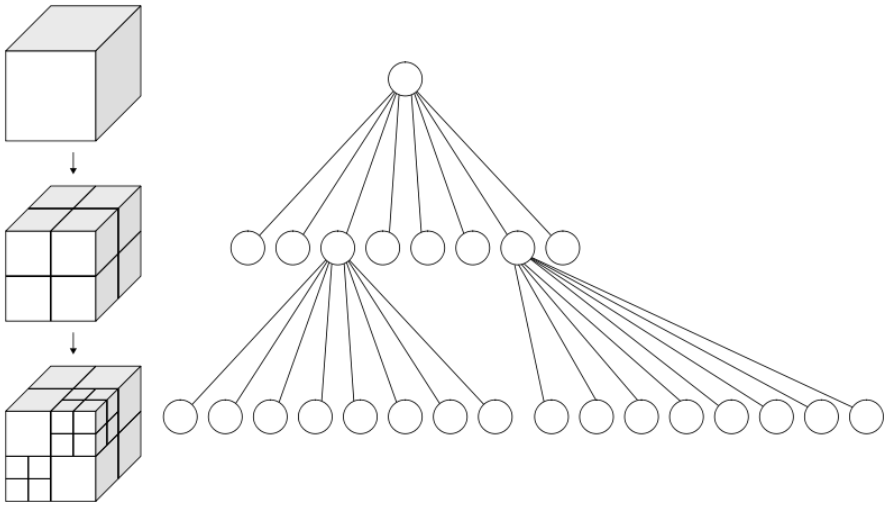

Figura 4. Estructura de un octree.

Fuente: Wikipedia.org.

2.3.4.-MODELADOS CONSTRUCTIVOS

2.3.4.1.-GEOMETRÍA SOLIDO CONSTRUCTIVA “CSG”

Es un modelo de representación en el que existe un conjunto inicial de primitivas que se combinan a través de operadores booleanos de conjunto (unión, intersección, diferencia) o transformaciones (traslación, rotación, escalado). En algunas

implementaciones, las primitivas son sólidos simples (cubos, esferas, cilindros, etc.) para asegurar que las combinaciones den como resultado sólidos válidos también. En CSG los objetos se almacenan como un árbol binario con operadores en los nodos intermedios y primitivas simples en las hojas, otra posibilidad son las llamadas “expresiones CSG” en el que se instancian primitivas básicas a partir de sus parámetros aplicando operaciones y acabamos teniendo un expresión lógica que define el objeto final.

CSG tiene las ventajas de que permite calcular con sencillez las propiedades físicas de los objetos modelados, resuelve con facilidad las interacciones entre objetos, maneja de igual manera superficies curvas y poliédricas y es una forma muy intuitiva de diseñar un gran número de objetos. Sin embargo hay que tener en cuenta que la aplicación de operaciones booleanas sobre dos objetos no da necesariamente como resultado otro objeto y además cada vez que se genera un objeto hay que redibujar el árbol, resolviendo las operaciones booleanas con lo que la generación de escenas complejas con muchos objetos puede ser lenta.

En algunos casos aplicar operaciones lógicas a dos sólidos da como resultado caras no deseadas ya que no envuelven ningún volumen. Para corregir esto se necesita aplicar lo que se denomina “Operaciones Lógicas Regularizadas” y la manera de aplicarlas se define en tres pasos:

El primer paso sería aplicar de manera normal la teoría de conjuntos con lo que se obtiene el objeto con las mencionadas caras sin volumen. A continuación se considera sólo el interior del resultado y por último se forma la clausura de ese interior, añadiendo las caras que envuelven ese interior para obtener el objeto deseado.

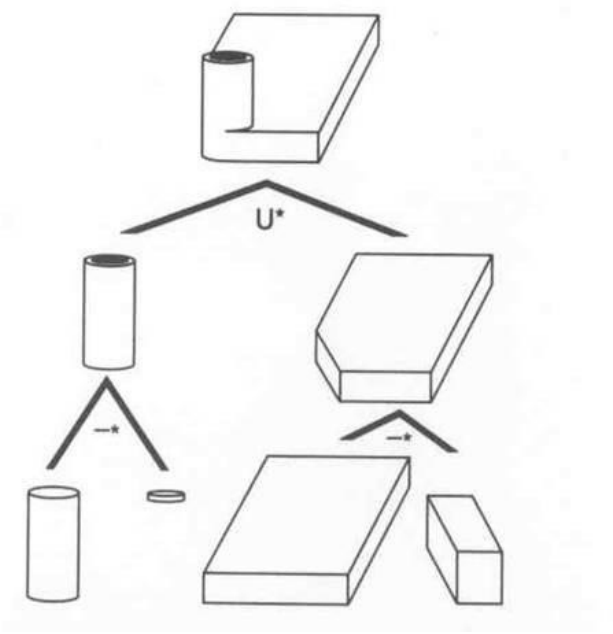

Figura 5. Ejemplo de árbol CSG.

Fuente: Introducción a la Computación Gráfica. Facultad de Ingeniería de la Universidad de la República – Uruguay.

2.3.5.-MODELOS DE REPRESENTACIÓN DE FRONTERAS B-REP (Boundary Representation)

Este método define los objetos en base a las superficies que lo definen, es decir, en base a sus fronteras, siendo una frontera una cara, superficie cerrada y orientable delimitada por sus respectivas aristas y vértices, por lo que se aproximan los objetos como poliedros. La representación B-Rep mantiene las propiedades topológicas del objeto que se modela.

La representación B-rep utiliza por un lado el objeto B-rep en el que se tienen en cuenta todos los elementos básicos del objeto (vértices, aristas y caras) y se les da un nombre o identificador. Por otro lado se tiene el árbol B-Rep en el que el nodo raíz es el objeto, en el siguiente nivel del árbol estan las caras y a continuación los bucles de vértices que definen esa cara, una cara puede estar definida por varios bucles en el caso de que tengan agujeros, pasantes o no, un bucle secundario representaría este agujero.

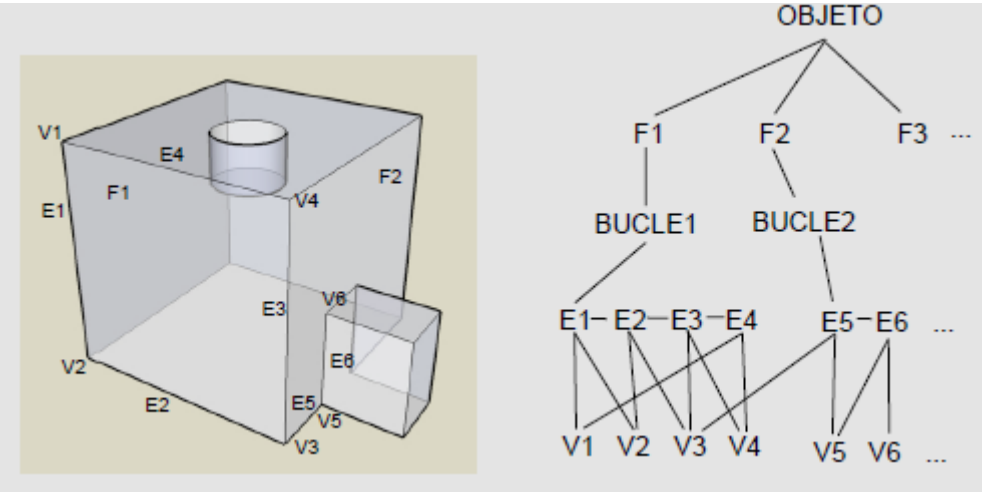

Figura 6. Notación B-Rep.

Una forma de organizar esto, sería la **Representación simple** en la que en una tabla, se listan todas las caras del objeto y en la siguiente columna, se hace lo mismo con los vértices que forman esta cara. No obstante esta representación tiene problemas de redundancia de vértices y problemas de cálculo costoso a la hora de comprobar propiedades.

La representación **arista alada (winged edge)** es la más conocida a la hora de definir un objeto B-Rep. Incluye información topológica del objeto (facilita el cálculo de operaciones en menor tiempo) y el elemento central de representación es la arista.

Se utilizan 3 tablas para la representación, las tablas de vértices, aristas y caras. La tabla de vértices contiene la lista de vértices del poliedro y por cada vértice una arista incidente en él. La tabla de aristas es la tabla más importante, cada arista se define a través de los vértices incidentes, las caras adyacentes a izquierda y derecha, y las aristas precedentes y sucesoras dentro de la definición de las caras adyacentes. Por último se tiene la tabla de caras en la que cada una de las caras se entiende como un bucle orientado (sentido horario u antihorario) de aristas que definen esa cara y se guarda una única arista incidente en esa cara.

arista	vértices		caras		aristas cara izda		aristas cara dcha	
	desde	hacia	izda	dcha	prec	suc	prec	suc
a	2	1	A	B	b	d	e	c
...	...	...	...	...	...	...	...	...

vértice	arista
1	b
2	d
...	...

cara	arista
A	b
B	c
...	...

Figura 7. Tabla de datos B-Rep.

Para poder rellenar estas tablas y así establecer la representación B-Rep, hay que tener en cuenta una serie de consideraciones. La primera y principal es que el poliedro debe considerarse como visto desde fuera del propio objeto, otra consideración sería establecer de antemano el sentido de los bucles que definen las caras, las aristas deben ser orientadas hacia un vértice, desde vértice “x” al vértice “y”.

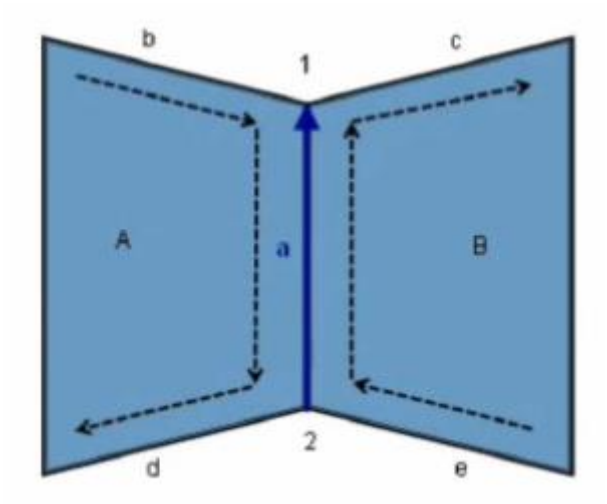

Figura 8. Representación arista alada (winged edge).

La representación arista alada no se puede aplicar a objetos no múltiples, los objetos no múltiples son aquellos en los que una arista comparte más de dos caras. Una posible solución a esta problemática es encontrar una topología similar que no presente el problema y representarla con B-Rep.

Otra contrariedad de la representación B-Rep son las caras con más de un bucle. Este caso se da cuando hay algún agujero en las caras. Una solución posible es considerar el bucle de aristas que define la cara principal en sentido horario y el bucle de la cara secundaria en sentido antihorario. La otra posibilidad consiste en añadir aristas auxiliares con las que la cara pasa a definirse con un solo bucle.

Los sólidos que quieran ser representados mediante B-Rep tienen que cumplir las siguientes características:

- Cada arista tiene que estar delimitada por dos vértices.
- Cada arista separa 2 caras (objetos múltiples).
- Las caras que coinciden en una arista tienen orientación conforme.
- Al menos tres aristas deben unirse en cada vértice.
- Las caras solo se intersectan en los vértices y aristas.

La consistencia topológica de los objetos se comprueba mediante la Fórmula de Euler, la fórmula para sólidos simples (sin agujeros ni huecos) es la siguiente:

$$V-E+F-2=0$$

Fórmula 1. Fórmula de Euler.

Donde “V” es el número de vértices, “E” es el número de aristas y “F” es el número de caras.

La fórmula extendida para objetos con agujeros y huecos es la siguiente:

$$V-E+F-(L-F)-2(S-G)=0$$

Fórmula 2. Fórmula de Euler Extendida.

Los factores que hace falta conocer respecto a la primera son "G" de género es decir el número de agujeros, "S" número de conchas que son superficies múltiples cerradas y por último "L" número de bucles.

Las operaciones para modelos de frontera que cumplen estas fórmulas son los operadores de Euler. Los operadores de Euler trabajan añadiendo y eliminando vértices, aristas y caras a un poliedro de partida, además evitan la construcción de poliedros inválidos (operación cerrada).

Las estructuras de datos para el manejo de representaciones de frontera están basadas en grafos que describen las relaciones topológicas del poliedro.

Las principales ventajas de la representación B-Rep son primero que es capaz de construir sólidos difíciles de modelar con primitivas básicas y segundo que es fácil convertir de B-Rep a alámbrico.

Por otro lado las desventajas son que requiere mucho espacio de almacenamiento y es lento trabajo con los operadores de Euler, por este motivo suelen hacerse combinaciones entre CSG y B-rep.

2.4. REFERENTES MODELADORES CONOCIDOS

2.4.1.-Autodesk 3ds MAX

Es un programa de creación y edición de gráficos y animación 3D, tiene una arquitectura basada en plugins, es uno de los programas más utilizado para la creación de videojuegos y arquitectura en películas. La forma de modelar las superficies es más paramétrica y utiliza distintos modificadores para darle forma a los objetos. Una gran cantidad de superficie de trabajo de modelado se realiza mediante la edición de sub-objetos de la superficie del objeto. Contiene herramientas de texturizado, iluminación, animación y renderizado.

2.4.2.-Autodesk MAYA

Al igual que 3ds Max, MAYA es un programa dedicado al desarrollo de gráficos 3D por ordenador, efectos especiales, animación, etc. MAYA se caracteriza por la posibilidad de personalización de su interfaz y el lenguaje MEL (Maya Embedded Language), con este se pueden crear scripts y personalizar funciones de MAYA.

Maya trabaja con NURBS, polígonos y subdivisión de superficies, e incluye la posibilidad de convertir entre todos los tipos de geometría. Los NURBS son figuras creadas a base de curvas y superficies. Polígonos (vértices, aristas y caras) son los más fáciles de modelar. La subdivisión de superficies es una técnica híbrida de las dos anteriores, aunque solo se puede modelar usando una a la vez pero con esta se puede obtener mayor subdivisión geométrica y mayor detalle de modelado.

2.4.3.-BLENDER

Es un modelador multiplataforma, libre y gratuito y comparado con el resto de modeladores mencionados su tamaño es muy pequeño. Igualmente tiene herramientas para manipular curvas, mallas poligonales, NURBS, etc. También incluye instrumentos de animación, edición de audio y renderizado e incluye un motor de juegos integrado.

2.5.EJEMPLOS CONFIGURADORES SIMILARES.

Referente a configuradores geométricos existen algunos como el “IKEA Home Planner” la aplicación de la compañía “IKEA” con la que los usuarios pueden configurar un entorno tridimensional para más tarde ir añadiendo diferentes muebles y objetos 3D.

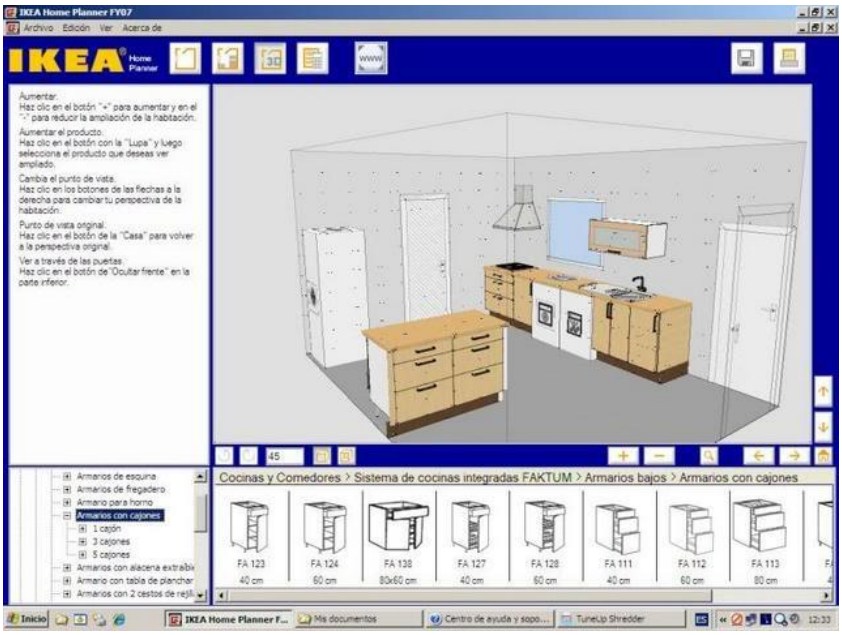

Figura 9. Muestra configurador IKEA.

Fuente: Imagen propiedad de IKEA Ibérica, S.A.

Otro ejemplo se tiene en la aplicación “CGBOX”, en este caso es un sencillo “configurador de habitaciones” en la que el usuario va colocando diferentes modelos 3D dentro de un espacio virtual personalizable y que permite salvar las composiciones personalizadas de cada usuario para más tarde ofrecer una factura detallada del coste de la habitación/composición realizada. También existe la aplicación “Sweet Home 3D” bajo la licencia GNU General Public License, con la que a partir de un plano usado como referencia se puede ir modelando la representación 3D del plano para después ir situando muebles y otros objetos de construcción como ventanas o puertas.

3. BASES DE DATOS CAD

3.1. Definición de “CAD”.

Viene del inglés “**C**omputer **A**ided **D**esign”, o lo que es lo mismo, Diseño Asistido por Computador. Hace referencia cualquier sistema informático que ayude a un diseñador en una tarea específica. Aunque se centre en el diseño, un sistema CAD puede realizar actividades complementarias relacionadas y cada sistema CAD puede funcionar de formas muy diferentes.

Permite trabajar con la información relativa de un objeto material. En el caso de la informática se suelen usar para modelado y estudio de las propiedades de los objetos. Dentro del espacio virtual que define la herramienta CAD se pueden caracterizar los objetos en función de sus propiedades intrínsecas (forma, tamaño, material, etc.) y también permite ver diferentes vistas o cortes del objeto ayudando al modelado. Los cambios producidos sobre el objeto se reflejan de forma instantánea sobre el modelo representado por lo que ayuda al diseñador a la verificación constante de sus decisiones.

Algunas herramientas CAD permiten crear imágenes realistas de los modelos e incluso realizar animaciones con ellos.

3.2. Bases de datos orientadas a objetos.

Los usos de las bases de datos en áreas como el CAD, la ingeniería del software y el procesamiento de documentos no se ajustan al conjunto de suposiciones que se hacen para aplicaciones de procesamiento de datos. El modelo de datos orientado a objetos pretende tratar estos nuevos tipos de aplicaciones que necesitan trabajar con datos de forma diferente a lo que se conoce porque necesitan una serie de cosas como por ejemplo estructuras más complejas para los objetos, transacciones de mayor duración, nuevos tipos de datos para almacenar imágenes o grandes bloques de texto,

necesitan definir operaciones específicas para cada aplicación y por último necesitan sistemas de control de versiones y configuraciones.

El modelo de datos orientado a objetos se basa en el concepto de encapsulamiento de datos y código que opera sobre estos en un objeto. Los objetos estructurados se agrupan en clases, existiendo un conjunto de clases estructurado en sub y superclases. Dado que el valor de un dato en un objeto también es un objeto, es posible representar el contenido del objeto dando como resultado un objeto compuesto.

El objetivo de los sistemas de bases de datos es la gestión de grandes cantidades de información. Las primeras bases de datos surgieron del desarrollo de los sistemas de gestión de archivos, más tarde estos sistemas se convirtieron en bases de datos de red o en bases de datos jerárquicas y después en bases de datos relacionales.

En una base de datos orientada a objetos, la información se representa a través de objetos como los presentes en la programación orientada a objetos como Java, C#, .NET. Un objeto en una Base de Datos Orientada a Objetos (BDOO) es una entidad identificable de forma unívoca que detalla tanto el estado como el comportamiento de una entidad del 'mundo real'. Estos identificadores de objetos se denominan OID Object Identifier únicos para cada objeto y en general no modificables por el usuario. Si dos objetos tienen el mismo estado pero diferentes OID, son equivalentes pero tienen identidades diferentes.

El estado de los objetos se describe mediante atributos y por otro lado su comportamiento es definido mediante métodos. Cada objeto contiene y define estos métodos y la interfaz para acceder a sí mismo y otros objetos pueden manipularlo.

Cuando se fusionan las propiedades de una base de datos con las de un lenguaje orientado a objetos, el resultado es un "SGBDOO" sistema gestor de base de datos orientada a objetos (en inglés ODBMS, object database management system).Los SGBDOO permiten al usuario especificar qué atributos y métodos son visibles en la interfaz del objeto y pueden invocarse desde afuera.

Un SGBDOO extiende los lenguajes con datos persistentes de forma transparente, control de concurrencia, recuperación de datos, consultas asociativas y otras capacidades. Son una buena elección para aquellos sistemas que requieran un rendimiento alto en la manipulación de tipos de datos complejos. Los SGBDOO almacenan objetos en disco y tienen una integración transparente con el lenguaje orientado a objetos, lo que reduce los costes de desarrollo y mantenimiento.

Las ventajas de un SGBDOO son:

Mayor capacidad de modelado. El modelado de datos orientado a objetos permite obtener una representación mucho más fiel del "mundo real" debido a:

- Un objeto permite encapsular tanto un estado como un comportamiento.
- Un objeto puede almacenar todas las relaciones que tenga con otros objetos.
- Los objetos pueden agruparse para formar objetos complejos (herencia).

Ampliabilidad:

- Se pueden generar nuevos tipos de datos partiendo de los ya existentes.
- Permite agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
- Reusabilidad de clases, lo que hace que sea más fácil el mantenimiento y haya un menor tiempo de desarrollo.

Lenguaje de consulta más expresivo. El acceso navegacional de un objeto al siguiente es la forma más común de acceso a datos en un SGBDOO. Mientras que el SQL utiliza el acceso asociativo. El acceso navegacional es más adecuado para operaciones como los despieces, consultas recursivas, etc.

Adecuación a las aplicaciones avanzadas de base de datos. Hay muchas áreas en las que los SGBD tradicionales no han tenido éxito como el CAD, CASE, OIS, sistemas multimedia, etc. en los que las

capacidades de modelado de los SGBDOO han hecho que esos sistemas si resulten efectivos para este tipo de aplicaciones.

Mayores prestaciones. Proporcionan mejoras de rendimiento con respecto a los SGBD relacionales.

Las desventajas de un SGBDOO son:

Carencia de un modelo de datos universal. No existe modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen de una base teórica.

Carencia de experiencia. Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas tradicionales.

Carencia de estándares. Existe una carencia de estándares general para los SGBDOO.

Competencia con respecto a los Sistemas de Gestión de Bases de Datos Relacionales (SGBDR) y lo Sistemas de Gestión de Bases de Datos Objeto-Relacionales. Estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y el Open DataBase Connectivity (ODBC) es un estándar de facto. Además el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.

La optimización de consultas compromete la encapsulación. Esta requiere una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente, esto compromete el concepto de encapsulación.

El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

3.3. ODMG: El estándar para modelos de objetos.

ODMG es un grupo de representantes de la industria de bases de datos el cual fue concebido en 1991 con el objetivo de definir estándares para los SGBDOO. El estándar ODMG que lleva el mismo nombre que el grupo es un modelo que define la semántica de los objetos de una base de datos. El modelo de objetos ODMG es un superconjunto que permite portar tanto los diseños como las implementaciones entre diversos sistemas compatibles.

La última versión del estándar propone los siguientes componentes principales de la arquitectura ODMG para un SGBDOO:

- Modelos de objetos.
- Lenguaje de definición de objetos (ODL, Object Definition Language).
- Lenguaje de consulta de objetos (OQL, Object Query Language).
- Conexión con al menos los lenguajes C++, Smalltalk y Java.

ODL es un lenguaje que permite definir las especificaciones de los tipos de objetos para sistemas compatibles con ODMG. Es el equivalente de DDL (Data Definition Languaje) de los SGBD tradicionales. Define los atributos y las relaciones entre tipos y especifica la signatura de las operaciones. Su principal meta es la de facilitar las portabilidad de los esquemas entre sistemas compatibles al mismo, interpolaridad entre diferentes SGBD.

OQL es un lenguaje declarativo del tipo de SQL que da la posibilidad de realizar consultas sobre las BDOO, incluyendo primitivas de alto nivel para conjuntos de objetos y estructuras. OQL no posee primitivas para modificar el estado de los objetos, ya que éstas se deben realizar a través de los métodos que dichos objetos poseen. La sintaxis básica de OQL es una estructura SELECT…FROM…WHERE…, como en SQL.

Existen otras maneras de realizar consultas a las BDOO que pueden resultar más fáciles que por intermedio del lenguaje OQL:

- Basadas en patrones: Se le da un objeto “patrón” a la consulta, al cual se le dan las misas características de los objetos a buscar. La consulta de vuelve el conjunto de objetos con esas propiedades.
- Basadas en API: A través de métodos de clases especiales, se programan las condiciones que deben cumplir los datos a consultar.
- Consultas nativas: Se crean nuevas clases en las que se programan los condicionales a usar en la consulta.

3.4. COMPARACION SGBD-SGBDOO

SGBD-Relacionales:

- Los datos están en la base de datos y los procesos en las aplicaciones desarrolladas mediante el lenguaje de datos asociado al SGBD (SQL) inmerso en un lenguaje de programación.
- Modelo conceptual de datos-modelo lógico.
- Eficientes para las aplicaciones tradicionales de negocios.

SGBD-Orientadas a Objetos:

- Objetos tienen encapsulados los datos y las operaciones que actúan sobre ellos.
- Desarrollo bajo SGBDOO: un único modelo subyacente, implementado en el SGBDOO, al que pueden acceder directamente las aplicaciones.
- Intentan satisfacer necesidades de aplicaciones más complejas.
- Dan poder al diseñador para especificar la estructura de los objetos complejos como las operaciones que se pueden aplicar a estos objetos.

3.5. Bases de Datos Objeto-Relacionales

Son bases de datos que han evolucionado del modelo relacional a un modelo más amplio que incorpora conceptos del paradigma orientado a objetos para adaptarse a las nuevas aplicaciones. Utilizan el estándar SQL: 2003, que incluye nuevos tipos y procedimientos y soporta objetos de gran tamaño pero siempre basándose en el modelo relacional.

La idea que originó las Bases de Datos Objeto-Relacionales fue fusionar las bases de datos relacionales y las orientadas a objetos intentando juntar sus ventajas y reducir sus deficiencias, conservando las mismas tablas relacionales básicas y el mismo lenguaje de consulta aunque incorporando el concepto del tipo objeto, dando como resultado de los Sistemas Gestores de Bases de Datos Objeto-Relacionales (SGBDOR). Se puede resaltar que se puede acceder a la base de datos usando lenguajes de más alto nivel orientado a objetos, pero donde toda la información es relacional, y las tuplas y las relaciones mantienen el mismo significado que en las bases de datos relacionales.

Entre las ventajas de las Bases de Datos Objeto-Relacionales están:

- Reutilización y compartición, ya que surge la capacidad de ampliar los servicios del Sistema Gestor de Bases de Datos para implementar funcionalidad estándar de manera central, en lugar de codificar dicha funcionalidad en cada aplicación.
- Posibilidad de adaptación de las aplicaciones relacionales a las objeto-relacionales, ya que los SGBDOR pueden introducirse de forma gradual. SQL: 2003 es compatible con SQL: 92(usado por las bases de datos relacionales), por lo que es posible realizar la transformación de un sistema existente.
- Los SGBDOR tienen la capacidad de almacenar instancias de clases y atributos clásicos relacionales, por lo que se puede escoger cuál es la implementación más correcta según la aplicación.

Por el contrario entre sus desventajas estarían:

- Aumenta la complejidad y el incremento de costo del rendimiento, ya que se pierde a simplicidad del modelo relacional.
- Los SGBDOR no añaden todas las características del modelo orientado a objetos, simplemente añaden complejidad al modelo relacional.
- Al existir datos relacionales y objetos se pierde la homogeneidad de acceso a la información. El tratamiento de la información depende de su naturaleza, por lo que se complica más el desarrollo en los SGBDOR.

Por último se muestra una tabla comparativa de los tipos de bases de datos explicados.

	BDR	**BDOO**	**BDOR**
Estándares	SQL2	ODMG-2.0	SQL:2003
Facilidad de uso	Fácil	Fácil para los programadores, no tanto para el usuario final.	Similar a las Bases de Datos Relacionales.
Facilidad de desarrollo	Independencia entre los datos y aplicación.	Los objetos son una forma natural de modelar extensible y adaptable.	Independencia entre los datos y aplicación.
Uso de SQL	Completo	Extensión de SQL con la creación de OQL adaptado a la orientación de objetos.	Definición de SQL3 basado en SQL pero con orientación a objetos.
Datos complejos y relaciones	Difícil de modelar	Facilita la definición del tipo complejos y relaciones.	Facilita la definición de tipo complejos y relaciones.
Soporte a la Orientación	Pobre, es difícil	Directa y Extensible.	Limitado sobre todo

a Objetos	almacenar los objetos en la BD.		para los nuevos tipos de datos.
Distribución	Muy buena	Depende del Sistema Gestor de Bases de Datos.	Muy buena.

Tabla 1. Comparativa tipos bases de datos.

Fuente: Yerandi Marcheco Díaz. I Conferencia Científica Internacional UCIENCIA 2014

Universidad de las Ciencias Informáticas. La Habana, Cuba.

4. UNITY

4.1. INTRODUCCIÓN

Unity es un motor gráfico enfocado principalmente al diseño de videojuegos, pero que actualmente se utiliza para crear todo tipo de aplicaciones, una de sus características más destacadas es que es multiplataforma, es decir, cuenta con posibilidades de crear proyectos tanto para ordenadores de diferentes sistemas operativos como para consolas de sobremesa o portátiles y dispositivos móviles (Android, IOS, Windows Phone) entre otros. Unity cuenta con una gran cantidad de usuarios en todo el mundo, lo que ha generado una cuantía enorme de documentación útil para el aprendizaje del mismo o la resolución de problemas. También cuenta con una versión gratuita totalmente funcional lo que lo hace accesible a todo el mundo y ha facilitado de gran manera su evolución gracias al apoyo de la comunidad de usuarios. Actualmente es uno de los motores de desarrollo más populares del mundo.

Una de las principales características de Unity es su tienda de Activos ("Assets" en inglés) que permite, comprar digitalmente y en ocasiones descargar gratuitamente estos Activos. Los Activos son básicamente cualquier cosa que se utilice para el desarrollo del juego: modelos 3D, ambientes, animaciones, audio, extensiones de editor, scripts, shaders, texturas, materiales, etc.

Unity es por un lado eficiente, a la hora de desarrollar es capaz de conseguir resultados y contenido interactivo en muy poco tiempo. También permite cierta flexibilidad a la hora de programar, ya que el entorno de trabajo es totalmente personalizable, admite la introducción y reubicado de ventanas propias y la adición de funcionalidades particulares. Asimismo permite mucha rapidez en el desarrollo ya que es posible los cambios realizados en el código de forma casi inmediata lo que facilita también la depuración de programas y localización de errores.

4.2.INTERFAZ DE UNITY

El editor de Unity se divide en una serie de “vistas”, cada vista tiene una funcionalidad determinada y tanto su colocación en la pantalla como su tamaño son editables. Aquí tenemos una imagen con las vistas de Unity:

Figura 10. Interfaz de Usuario Unity.

Fuente: Imagen propiedad de Unity Technologies.

4.2.1 PROJECT BROWSER.

En el “Project Browser” o navegador de proyecto (“Project” en la figura 11) es donde se gestionan todos los activos del proyecto. En el panel izquierdo se muestran de manera jerárquica todos los ficheros y carpetas del proyecto. Cuando se hace clic sobre una de estas carpetas, en el panel derecho aparecerá una lista de iconos que son los activos que contiene esta carpeta seleccionada. Hay diferentes tipos de iconos para distinguir si estos son sub-carpetas, materiales, scripts, etc. Los iconos son redimensionables con un

"slider" (control basado es un barra con un botón que puede desplazarse de izquierda a derecha).

Sobre la lista de "assets" se tiene una lista de "favoritos" que permite tener los elementos usados más frecuentemente de manera directa para un acceso más fácil. Para añadir elementos a la lista de favoritos solo hay que arrastrarlos hasta ella.

Encima de este panel hay una "ruta de navegación" del elemento seleccionado. Se puede clicar sobre la ruta para navegar más fácilmente sobre la jerarquía de archivos

En la parte superior de esta vista se tiene una barra de navegación, en ella en la parte izquierda existe un botón para crear activos en la localización actual en la que se está y a la derecha, una barra de búsqueda nos permite localizar estos assets en el proyecto, dispone de diferentes tipos de filtros para hacer la búsqueda más eficiente.

4.2.2 HIERARCHY

Esta parte contiene una lista de todos los "GameObject" de la actual escena. Un "GameObject" es cualquier objeto usado para la escena (modelo 3D, luz, planos, etc.). A medida que se vayan modificando o borrando objetos de la escena también se borrarán o modificarán de este panel. Se pueden arrastrar "GameObjects" dentro de otros para hacer uso del "Parenting" (crianza en inglés).

El "Parenting" es un concepto de Unity que permite hacer un objeto hijo de otro, arrastrando el hijo al padre en la vista de hierarchy. Un "objeto hijo" heredará el movimiento y rotación del padre.

4.2.3 TOOLBAR

Esta barra de herramientas contiene una serie de utilidades relacionadas con varias partes del editor, todas ellas se detallarán en sus correspondientes secciones.

4.2.4 SCENE VIEW

Es la vista donde se posicionan todos los objetos, entornos, objetos, cámara y básicamente todos los "GameObjects". Manipular los objetos en la escena es una tarea muy importante y para eso Unity provee una serie de combinaciones de botones que realizan los trabajos más comunes, como mover la cámara por la escena, centrar la cámara en un objeto, hacer zoom, etc. También hay un Gizmo en la parte superior derecha en el que cada uno de sus brazos representa uno de los ejes geométricos. Con el Gizmo podemos orbitar y situar la cámara en uno de los ejes. El Gizmo solo esta visible si se está realizando una escena tridimensional.

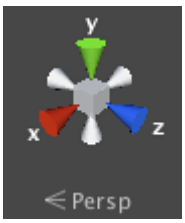

Figura 11. "Gizmo" de Unity.

Fuente: Imagen propiedad de Unity Technologies.

Como se ha dicho anteriormente en la barra de herramientas de Unity existen una serie de funcionalidades específicas, aquí se muestran las referentes al movimiento, rotación y escalado de los objetos, cada una de estas funciones tiene su correspondiente gizmo que aparece al apretar el botón correspondiente.

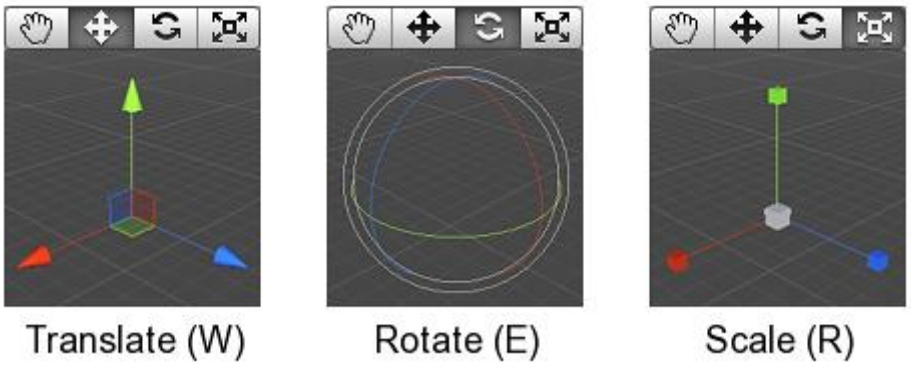

Figura 12. Gizmos, traslación, rotación, escalado.

Fuente: Imagen propiedad de Unity Technologies.

Por último se tiene la barra de control de la escena que permite hacer cosas como apagar/encender el audio, la iluminación, cambiar entre modo 2D/3D y aplicar algunos efectos de imagen.

4.2.5 GAME VIEW (Vista de Juego)

La vista de juego se renderiza de las cámaras que se han colocado en la escena. Esta vista es una muestra de cómo quedará el juego final. Normalmente se requieren varias cámaras para controlar lo que ve el jugador o usuario de la aplicación.

Se puede hacer uso de los siguientes botones de la barra de herramientas para activar o pausar el modo de juego.

Figura 13. Botones “Play Mode”.

Fuente: Imagen propiedad de Unity Technologies.

Una característica importante a tener en cuenta es que mientras estamos en “modo juego” cualquier cambio que se haga en la escena o en el código es temporal y se **reseteará** cuando salgamos del modo de juego.

La vista de juego dispone de una “barra de control” en la parte superior. El primer botón desplegable que se tiene nos permite modificar el “aspect ratio” de la ventana del juego. Esto se usa para testear como se vería el juego en monitores con diferentes “aspect ratio”.

Más a la derecha de la barra de herramientas está el botón de “maximize on Play” es cual tiene dos estados, apagado y encendido, este permite que el juego se maximice al 100% en nuestra pantalla para tener una mejor vista cuando se active el modo de juego.

Continuando por la derecha se sitúa el botón de estadísticas que muestra la ventana estadísticas de renderizado, muy útil para monitorizar el rendimiento de gráficos de un juego.

El último botón es el “Gizmos toogle”. Si está activado, todos los gizmos que aparecen en la escena también aparecerán en la vista de juego. Es botón también dispone de un “pop-up” que muestra una lista de todos los gizmos de la escena y permite personalizar sus iconos y otras propiedades.

Figura 14. Game View Control Bar.

Fuente: Imagen propiedad de Unity Technologies.

4.2.6 INSPECTOR

Los juegos de Unity están compuestos por múltiples “GameObjects” y cada uno de ellos puede contener, scripts, sonidos, luces, mallas 3d, etc. El inspector muestra la información detallada del “GameObject” actual junto con todos sus componentes y propiedades.

Cualquier propiedad mostrada en el inspector puede ser modificada. Incluso variables en los scripts pueden modificarse sin necesidad de modificar el propio script. Estas modificaciones se pueden hacer en el modo de juego para ver cómo quedaría el cambio final (recordar que se resetearán los cambios al salir del modo juego). Si se definen variables públicas de tipo “GameObject” se

pueden arrastrar objetos al inspector para hacer directamente la asignación.

Por otro lado el inspector también permite asignar iconos personalizados para cada componente y otorgar a cada componente una serie de etiquetas para que sean más fáciles de encontrar con las herramientas de búsqueda.

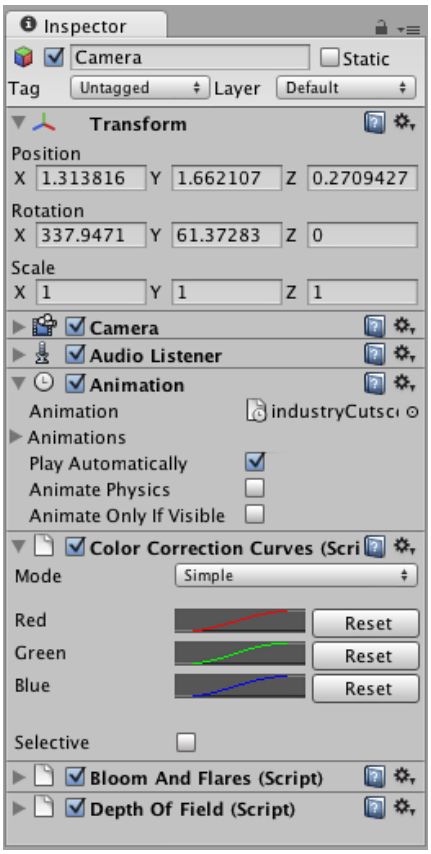

Figura 15. Vista Inspector.

Fuente: Imagen propiedad de Unity Technologies.

4.2.7 OTRAS VISTAS

Las anteriores vistas explicadas son las principales de Unity, no obstante existen otro tipo de vistas como son:

- La vista de consola que enseña mensajes de depuración, advertencias y errores.
- Vista de animación, sirve para animar objetos en la escena.

- El “profiler” que se usa para encontrar los cuellos de botella de rendimiento del juego.
- La “Asset Sever View” (Vista del servidor de activos), utilizada para controlar la versión del proyecto usándolo el servidor de activos de Unity.
- “Lightmapping View”, (vista del mapeo de luces), gestiona los “lightmaps” de juego.
- Por último la “Occlusion Culling View”, maneja la técnica de “Occlusion Culling” para mejorar el rendimiento de esta, esta técnica se basa en no renderizar los objetos que están fuera de la vista de la cámara, es decir objetos que el usuario no puede ver.

4.3. CREAR ESCENAS CON UNITY

Las escenas contienen los objetos del juego o aplicación. Pueden utilizarse para crear menús de inicio, niveles individuales, etc. En cada escena se colocaran escenarios, decoraciones, luces para diseñar y crear el juego o programa en piezas.

Una cosa fundamental para crear escenas es el uso de “prefabs”, un “prefab” es un conjunto de uno o más “GameObjects” con componentes asignados y una serie de propiedades. Los “prefabs” son un tipo de activos que pueden ser usados para crear instancias de objetos personalizados, cada instancia es una copia del “prefab” original. Por ejemplo podemos crear un “prefab” para crear un objeto árbol y luego usar instancias de ese “prefab” para crear un bosque. Por defecto cualquier cambio hecho sobre un objeto de este tipo se aplicará a todas sus instancias, sin embargo también se puede romper este “enlace” entre el “prefab” y su instancia para crear variaciones del objeto original. Una vez creado el “prefab” crear instancias del mismo consiste en arrastrar el “prefab” de la “vista de proyecto” a la vista hierarchy o directamente a la escena. Así tenemos una única instancia del “prefab” para posicionar donde se quiera.

Cuando tenemos un “prefab” o un “GameObject” seleccionado, se les puede agregar funcionalidad usando componentes. Para añadir un componente se selecciona el objeto y después el siguiente que se quiera del menú de componentes. Este aparecerá en el inspector del “GameObject”. Si al añadir un componente se rompe la conexión de un objeto a su “prefab”, siempre se podrá seleccionar ”GameObject->Apply Changes to Prefab” para reestablecer la conexión.

Una se tenga el “GameObject” en la escena, se puedes usar la herramienta de transformación para posicionarlo donde se quiera, también podemos utilizar el inspector para modificar los valores numéricos de traslación, rotación y escalado.

Otro de los aspectos importantes a la hora de crear escenas es el uso de cámaras. Las cámaras representan los ojos de la aplicación. Todo lo que el usuario ve es a través de una o varias cámaras. Una cámara no es más que un “GameObject” con un componente “cámara” asignado y por tanto puede hacer cualquier cosa que haga un “GameObject”. Unity ya contiene una serie de scripts propios con los que controlar la cámara, a pesar de esto siempre se puede crear scripts propios y asignárselos al objeto cámara.

Por otro lado un elemento que, salvo raras excepciones, siempre se necesitará añadir a las escenas son la luces. Estas añaden diferente atmosfera y ambiente a la aplicación y por ello es muy importante usarlas de manera eficiente.

Existen cuatro tipos de luces en Unity y son los siguientes:

- **POINT LIGHT:** Este tipo de luces brilla desde un punto en todas direcciones (como si fuera una bombilla), es el tipo de luz más comúnmente usadas y tienen un coste medio para el computador de gráficos.

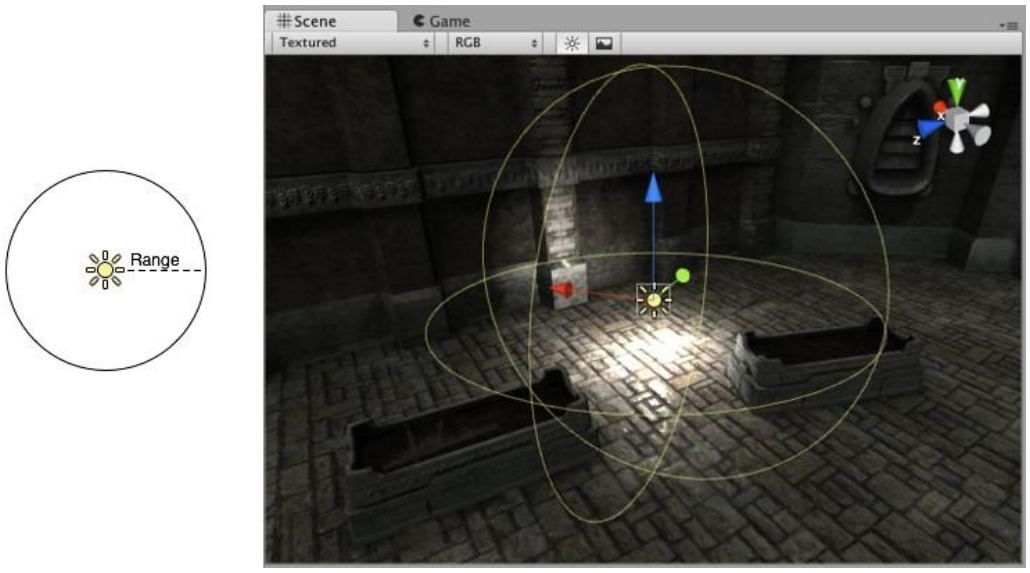

Figura 16. Rango y muestra Point Light.

Fuente: Imagen propiedad de Unity Technologies.

- **SPOT LIGHT: Brillan** en una sola dirección en forma de cono, solo los objetos dentro de este cono se ven afectados por la luz. Se usan para representar linternas, faros de los vehículos o postes de luz. Son las más caras en cuanto a coste computacional.

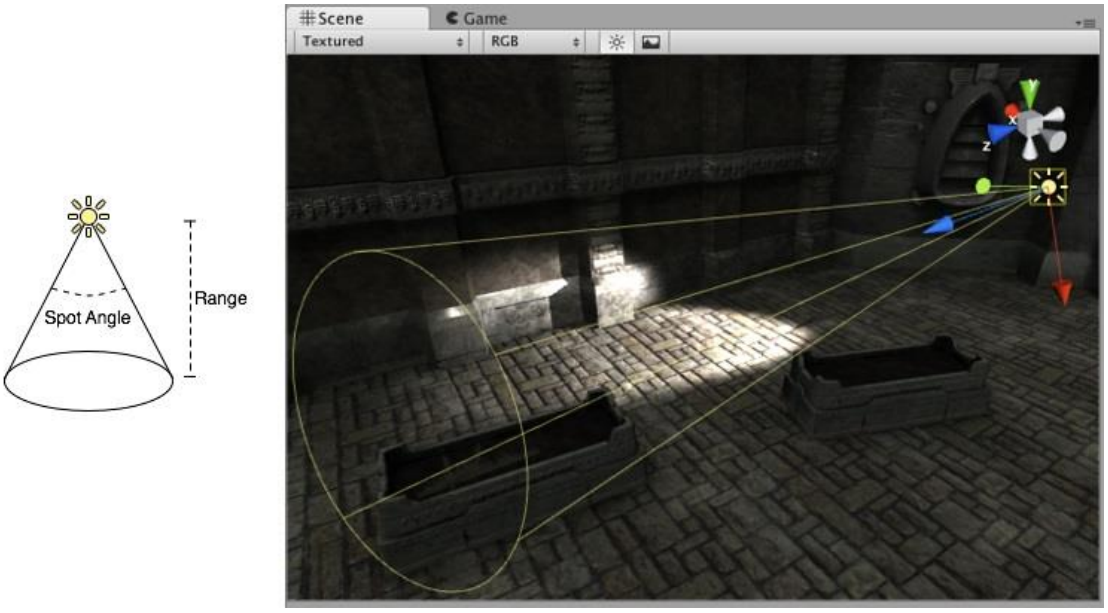

Figura 17. Rango y muestra Spot Light.

Fuente: Imagen propiedad de Unity Technologies.

- **DIRECTIONAL LIGHTS:** Son usadas principalmente para simular la luz del sol o de la luna. Afectan a todos los objetos y superficies de la escena. Son las más baratas en cuanto a procesamiento de gráficos se refiere.

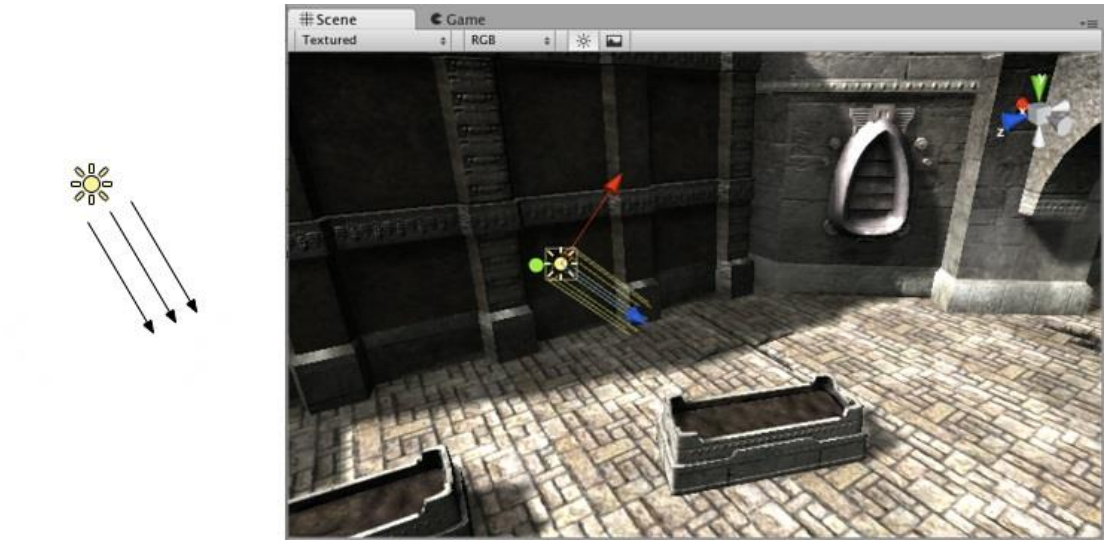

Figura 18. Rango y muestra Directional Light.

Fuente: Imagen propiedad de Unity Technologies.

- **Area Lights:** Proyectan la luz desde un lado de un área rectangular de un plano. La luz afecta a todos los objetos dentro del rectángulo definido por su altura, anchura y la normal al plano, es decir el lado por donde se proyecta la luz. La luz es emitida en toda la superficie del rectángulo, así que las sombras de los objetos afectados suelen ser más suaves que las resultantes con “point lights” o “directional lights”. Estas luces son muy difíciles de calcular y no están disponibles en tiempo de ejecución por lo que hay que utilizar lightmaps con ellas para poder ver su resultado.

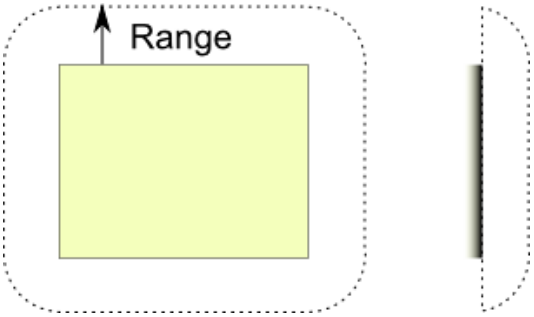

Figura 19. Rango Area Light.

Fuente: Imagen propiedad de Unity Technologies.

En Unity las luces pueden ser renderizadas de una de las dos maneras siguientes: “vertex lightning” (iluminación por vértices) o “pixel lightning” (iluminación por pixel). La técnica del “vertex lightning” solo calcula la luz en los vértices de los modelos 3d e interpola la luz sobre las superficies de los modelos. “Pixel lightning” calcula la luz en cada pixel de la pantalla y por tanto resulta mucho más caro, de hecho las antiguas tarjetas gráficas solo soportaban “vertex lightning”. No obstante el método “pixel lightning” permite gráficos avanzados como por ejemplo sombras en tiempo real.

Las luces tienen un gran impacto en la velocidad de renderizado por lo que se tiene que llegar a un equilibrio entre la calidad de la luz y la velocidad del juego. Como la iluminación por pixel es mucho más cara que la iluminación por vértice, Unity solo renderiza las luces más brillantes con la calidad que ofrece iluminación por pixel. El número de píxeles que utilizan esta técnica se puede configurar en los ajustes de calidad de Unity. También se permite configurar el modo de renderizado para cada luz particular, normalmente Unity las clasifica de modo automático según cuanto del objeto es iluminado por la luz.

4.4. CREAR SCRIPTS CON UNITY

Un script es un archivo de texto plano que contiene las órdenes y lenguaje propios de un lenguaje de programación. En este caso con

Unity el generar scripts es una parte fundamental. Ya que la más simple de las aplicaciones necesitará de scripts para capturar eventos del teclado, crear efectos gráficos, controlar el comportamiento de los "GameObjects" etc. En esta sección se pretende explicar las bases del "Scripting".

4.4.1 USO Y CREACIÓN DE SCRIPTS

El comportamiento de los "GameObjects" en Unity viene definido por los componentes vinculados a este. Los componentes básicos que incorpora Unity pronto se quedan cortos cuando se pretende realizar una aplicación de mayor nivel de desarrollo, por lo que se hace necesaria la creación de componentes o comportamientos personalizados a través de los scripts.

Unity soporta tres lenguajes de programación:

- **C#** un lenguaje similar a JAVA y C++.
- **UnityScript** un lenguaje específico para Unity basado en JavaScript.
- **Boo**, un lenguaje de .NET con sintaxis similar a Python.

Los scripts se pueden crear directamente con Unity desde el menú "Crear" del panel de proyecto o seleccionando Assets→Create→Script del algunos de los lenguajes de programación mencionados anteriormente. El script se creará en el directorio seleccionado y se instará a darle un nuevo nombre.

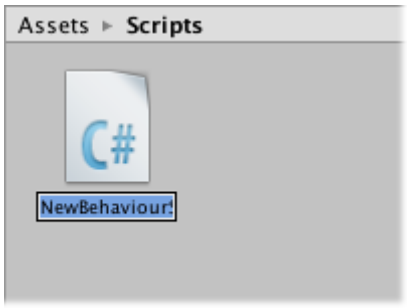

Figura 20. Nuevo Script.

Fuente: Imagen propiedad de Unity Technologies.

Al hacer doble click sobre el script se abrirá el entorno de desarrollo, por defecto Unity usa "MonoDevelop" pero se puede usar cualquier otro seleccionándolo en el menú de preferencias de Unity.

Un script hace su conexión con las funciones integradas de Unity implementando una clase que deriva de la clase incorporada "MonoBehaviour". Cada vez que se une un script a un "GameObject", se crea una instancia de la clase definida en dicho script. El nombre de la clase se coge del nombre que se le ha asignado al script, ambos nombres deben ser iguales.

Dentro de la estructura del script hay dos funciones principales. La función **Update** manejará la actualización del "GameObject" cada frame, esto pude ser el movimiento del objeto, desencadenar acciones, responder al teclado y en definitiva cualquier cosa que deba ser controlada a través del tiempo mientras se usa el programa en cuestión. Por otro lado tenemos la función **Start** que se llama antes de que la aplicación empiece y es aquí donde se debe hacer cualquier tipo de inicialización.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }
}
```

Figura 21. Estructura básica de un script en Unity.

Fuente: Imagen propiedad de Unity Technologies.

Para vincular un script a un “GameObject” solo se tiene que arrastrar al objeto deseado en el panel “hierarchy” o al inspector del objeto actualmente seleccionado.

Conforme se desarrolla un script en Unity se podrá apreciar en el inspector como las propiedades del archivo crecen. Esto es porque Unity permite cambiar desde el inspector el valor de las variables actuales y cada vez que se define una variable nueva se agrega su propiedad al inspector. Esto es útil porque se permite modificar el valor de las variables sin necesidad de modificar el script en si mismo.

4.4.2 CONTROLAR OBJETOS USANDO SUS COMPONENTES

Aunque el inspector permite cambiar las propiedades de los objetos un script puede cambiarlas de forma gradual a través del tiempo o en respuesta a una entrada del usuario. Cambiando,

creando y destruyendo objetos en el momento justo, cualquier flujo de aplicación puede crearse.

El caso más común es que se necesite acceder a alguno de los componentes de un “GameObject”. Como ya se ha dicho un componente no es más que la instancia de una clase por lo que se necesita una referencia a esta instancia. Esto se consigue con la función “GetComponent”, normalmente se almacena esta referencia en una variable y así más tarde es posible acceder a las propiedades de esta clase como lo se haría en el inspector. Esta función puede utilizarse desde cualquier script para acceder por ejemplo a otro script que tenga el mismo objeto. Esta función devolvería “null” si se intenta acceder a un componente que todavía no está vinculado al objeto.

Unity dispone de algunas variables base incorporadas a las cuales no hace falta acceder a través de la función “GameObject”, estas variables son las más usadas comúnmente, como por ejemplo la variable “transform” que contiene los parámetros de posición, rotación y escalado del “GameObject” al que este unido el script.

```
void Start () {
    Rigidbody rb = GetComponent<Rigidbody>();

    // Change the mass of the object's Rigidbody.
    rb.mass = 10f;
}
```

Figura 22. Función “GetComponent”.

Fuente: Imagen propiedad de Unity Technologies.

Sin embargo hay veces en las que no se necesita acceder a un componente de un “GameObject” sino a un “GameObject” completo. Para esto existen diferentes metodologías.

La primera y más directa es añadir una variable pública de tipo “GameObject” al script. Esta variable aparecerá en el inspector y se podrá asignarle un objeto arrastrándolo hasta aquí. Este método es útil cuando se utilizan objetos individuales que tienen conexiones

permanentes. La desventaja es que se tiene que utilizar el editor para hacer las conexiones y por tanto estas no se ejecutan en tiempo de ejecución y se hace tediosa la tarea de actualizar o borrar objetos que utilicen estas variables. Para buscar objetos en tiempo de ejecución Unity dispone de dos métodos.

Una posibilidad es tratar a un montón de objetos idénticos como si fueran hijos de un objeto padre que los controla. Los objetos hijo pueden ser accedidos a través del componente “transform” del padre ya que todos los “GameObjects” tienen implícitamente un componente “transform”. Esto puede ser útil cuando un objeto tiene un objeto hijo que puede ser añadido o borrado durante la ejecución del programa.

La otra posibilidad es utilizar un identificador (el nombre del objeto por ejemplo) y acceder por medio de la función “GameObject.Find(“Nombre”)”, también se pueden acceder a grupos de objetos con las funciones similares:

- GameObject.FindWith Tag(“Nombre”).
- GameObject.FindGameObjectsWithTag(“Nombre”).

4.4.3 EVENTOS

Unity pasa el control a los scripts de manera intermitente llamando a ciertas funciones declaradas en él. Una vez las funciones han terminado de ejecutarse el control es devuelto a Unity. Estas funciones son llamadas “Eventos” ya que son activadas por Unity en respuesta a un evento que ocurre durante la ejecución. Se usa un sistema de nombres para saber que función llamar para un evento en particular. A continuación se hablará de algunos de los eventos más importantes.

Ya se ha visto por ejemplo la función “Update”, en este caso el evento es el de actualizar la posición, estado, etc de los objetos justo antes de que cada frame sea renderizado. Una función de evento separada es “FixedUpdate” que se llama justo antes de la actualización de las físicas de la aplicación o juego. La frecuencia de

llamada de estas dos funciones es diferente, generalmente se obtienen mejores resultados situando código referente a las físicas en la función “FixedUpdate”.

Por otro lado se tienen las funciones “Start” y “Awake”, de la primera ya se ha hablado y de la segunda se llama para cada objeto de la escena justo cuando la escena se ha cargado.

Una de las funciones más importantes es la función “OnGUI”, es el sistema que utiliza Unity para renderizar controles GUI (Graphic User Interface), es decir controles de la interfaz gráfica tales como botones, campos de texto, etc.

También existen una serie de funciones que controlan los eventos del ratón, por ejemplo controlan que botón del ratón se ha pulsado o si el puntero del ratón ha pasado por encima de algún control.

4.4.4 CREAR Y DESTRUIR OBJETOS

En cualquier aplicación o juego es muy común que objetos, personajes o partes del escenario sean creados o borrados durante la ejecución. En Unity se puede crear una copia de un objeto llamando a la función “Instantiate”. El objeto del que se hace la copia no es necesario que este durante la escena. Es más común usar un “prefab” y arrastrarlo a la escena. Además instanciar un “GameObject” copiara el estado y todos los componentes del original.

Luego también tenemos la función “Destroy” (destruir en castellano) que borrará el objeto al finalizar la actualización del frame u opcionalmente lo hará tras un pequeño retraso (“delay”) de tiempo. Esta función puede borrar componentes individuales sin borrar el “GameObject” al que están vinculados.

4.4.5 COROUTINES (Corutinas)

Cuando se hace una llamada a una función esta se ejecuta hasta terminar su ejecución y después el programa principal continua desde donde se dejó antes de la llamada a función. Sin embargo hay veces en las que no se quiere que el programa se bloquee hasta terminar la función, por ejemplo a la hora de descargar objetos de un servidor no se persigue que el programa se quede bloqueado mientras se descarga el objeto. Otra situación es a la hora de hacer una tarea que sea progresiva en el tiempo, si se llama a una función se ejecuta esta en un solo frame, no obstante hay veces que se quiere tareas progresivas que se ejecuten cada frame para conseguir efectos gráficos de difuminado o desaparición progresiva por ejemplo. Para estos casos es mejor el empleo de "Corutinas".

Una corutina es como una función que tiene la habilidad de ejecutarse en un hilo secundario concurrentemente mientras Unity continua con el programa principal, dicho de otra manera la corutina empieza y en algún momento se pausa y devuelve el control a Unity pero sigue ejecutándose en el siguiente frame.

Las corutinas se declaran como cualquier otra función solo que devuelven el tipo "IEnumerator" y deben tener la sentencia "yield return" en alguna parte de su código, esta sentencia es el punto donde la ejecución se pausa y continua en el siguiente frame. Para hacer que una empiece, en lugar de hacer una llamada a función convencional hay que usar la función "StartCoroutine" (excepto en UnityScript) y pasarle como parámetro una cadena de texto con el nombre de la subrutina.

Por defecto una corutina empieza de nuevo al hacer "yield", pero también es posible introducir un tiempo de retardo usando "WaitForSecond" de manera que la sentencia quedaría como sigue:

yield return new WaitForseconds(1.0f).

Esto puede usarse como optimización para tareas periódicas, el modo más obvio de hacer una tarea periódica es en la función "Update" pero de esta manera la función se llamaría muchas veces

por segundo. Si no se quiere que la tarea se repita tantas veces se puede usar las corutinas para hacer una actualización regular pero no tan frecuente.

```
IEnumerator Fade() {
    for (float f = 1f; f >= 0; f -= 0.1f) {
        Color c = renderer.material.color;
        c.a = f;
        renderer.material.color = c;
        yield return null;
    }
}


void Update() {
    if (Input.GetKeyDown("f")) {
        StartCoroutine("Fade");
    }
}
```

Figura 23. Ejemplo corutina y su llamada.

Fuente: Imagen propiedad de Unity Technologies.

4.4.6 La clase WWW

Por último se hablará de la clase WWW que ha sido ampliamente utilizada en este proyecto.

Esta clase proporciona un acceso simple a páginas web, permite recoger el contenido de las URLs (direcciones web). Se empieza una descarga en segundo plano creando una nueva instancia de la clase que devuelve un objeto WWW.

Se puede o bien inspeccionar la propiedad “isDone” para ver si la descarga a finalizado o bien usar una corutina para esperar a la descarga del objeto sin bloquear el resto de la aplicación.

Normalmente se usa para sacar información de un servidor web, por ejemplo archivos con registro de puntuaciones, descarga de imágenes o también se puede hacer llamadas a ficheros php que devolverán una respuesta en un texto.

La clase WWW también puede utilizarse para subir datos a servidores web haciendo uso de las llamadas GET y POST, por defecto se usa GET y POST se deja para subir parámetros de tipo "postData", la clase "WWWForm" sirve como ayuda para crear estos parámetros.

En esta sección se han explicado conceptos base del motor de desarrollo Unity tales como su interfaz gráfica y algunas de sus características principales, Unity contiene muchas más características, gráficos, físicas, audio, animación, etc. Las cuales no se desarrollarán en esta memoria. El resto de aplicaciones más específicas que se hayan utilizado serán explicadas en siguientes secciones.

5. APLICACIÓN DESARROLLADA

A continuación se explicará el desarrollo de la aplicación realizada, la solución propuesta, como se ha llegado a ella, los programas utilizados, la realización de la base de datos utilizada y también se hablará de la interfaz de usuario y de cómo montar un objeto.

5.1. CONEXIÓN ENTRE PARTES

La aplicación propuesta pretende poder "construir" muebles tridimensionales de forma incremental, añadiendo piezas una a una. En este contexto la principal problemática reside en la conexión de las piezas ¿Cómo se sabe dónde se coloca cada pieza?, ¿Cómo sabemos qué tipo de pieza va en este o aquel sitio?

La herramienta principal usada es el programa Unity explicado anteriormente. Unity trabaja con un espacio tridimensional por lo que los objetos3D se sitúan dentro de este espacio usando las coordenadas de los tres ejes X, Y, Z. Estas coordenadas es el principal concepto con el que se trabaja ya que son las coordenadas las que van a determinar la posición de las piezas y por tanto el resultado final de los muebles. Hay que tener en cuenta que Unity sitúa un objeto en unas coordenadas determinadas y que estas hacen referencia al centro del objeto, es decir el punto central del volumen que envuelve el objeto 3D, por lo que se ha tenido que tener en cuenta esto para la colocación de los mismos.

El siguiente punto consiste en definir un modelo de conectividad para las diferentes piezas. El modelo planteado define una conexión sobre cualquiera de las piezas principalmente a través de tres parámetros. El primero la coordenada de la conexión sobre la pieza, en segundo lugar un número y el tercer parámetro un "signo", positivo o negativo, representando una conexión macho, hembra. De

esta manera se define que una conexión solo puede conectarse a otra que tenga el mismo número pero signo contrario. Por ejemplo una pieza con una conexión “1+” solo podría conectarse con otra pieza con conexión “1-”. Esto suponiendo que se está en el caso simple de que una pieza solo tiene un punto de unión. Con estos 2 parámetros se es capaz de discernir la mayoría de casos especiales a la hora de colocar las piezas.

El caso más complicado es cuando una pieza tiene varios puntos de unión y todos estos tienen que encajar con una o más piezas. Para programar este caso se definió un pequeño algoritmo que se pondrá en pseudocódigo a continuación:

```
Para cada ancla1 posible ∈ pieza a insertar {
    Para cada ancla2 ∈ piezas en la escena {
        Si ancla2 está libre y es posible {
            Distancia=coordenadas_ ancla1 - coordenadas_ ancla2;
            Si Distancia==0{
                Conectar ( ancla1 , ancla2);
                ancla1 ocupada;
                ancla2 ocupada;
            }
        }
    }
}
```

La idea básica del algoritmo es ir comprobando todas las conexiones posibles de la pieza a insertar con las piezas que ya hay en escena, cuando se encuentra una posibilidad que ha utilizado todas las conexiones de la pieza a insertar se da como correcta y se ofrece al usuario.

Según todo esto definido anteriormente se puede decir que conectar dos uniones consistirá en colocarlas en el mismo punto geométrico y mover con ellas los objetos o piezas a los que pertenecen.

5.2.¿COMO ES LA BASE DE DATOS?

Lógicamente se necesita algún medio para mantener actualizada la información de la aplicación y este medio es una base de datos. Aquí se explicará la base de datos utilizada y la relación entre sus tablas.

A continuación se puede ver la base de datos realizada:

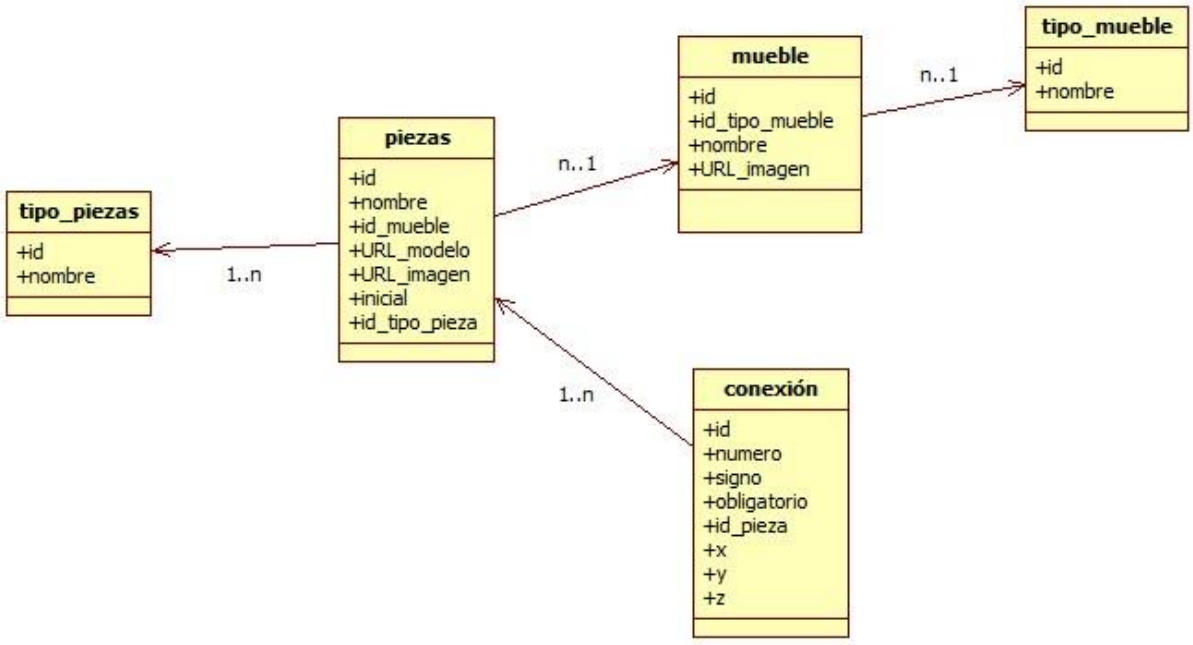

Figura 24. Esquema Base de Datos Utilizada.

Las tablas de las que se dispone son las tablas “conexion”, “mueble”, “piezas”, “tipo_mueble” y “tipo_piezas”.

La tabla “tipo_mueble” almacena las diferentes clases de muebles que hay en el sistema, como por ejemplo, sillas, mesas,

armarios, etc. Solo contiene dos campos, el identificador de tipo y el nombre de este tipo de mueble.

En la tabla “mueble” como su propio nombre indica se almacenan los muebles creados para un tipo en concreto, por ejemplo se pueden tener “sillas cuadradas”, “sillas redondas”, “mesa modelo marca genérica”, etc. En este caso se tienen los siguientes campos:

- Id: El identificador de mueble.
- Nombre: Denota el objeto concreto que se está creando
- Id_tipo_mueble: Este campo lo relaciona como clave ajena con la tabla anterior (“tipo_mueble”).
- url_imagen: Indica la ruta para coger la imagen de este mueble del servidor.

La siguiente tabla es la tabla “piezas”, es la tabla más importante del sistema, aquí se guarda la información correspondiente para las piezas que en definitiva son las que conforman los objetos 3D completos. Sus campos son los siguientes:

- Id: El identificador de pieza.
- Nombre: Denomina la pieza.
- Id_mueble: Hace referencia al mueble al que está asociada la pieza.
- url_modelo: Contiene la dirección web donde está guardado el modelo 3D de dicha pieza.
- url_imagen: De nuevo esto es una url que dirá a la aplicación en qué dirección del servidor web debe coger la imagen de la pieza.
- Inicial: Un booleano que nos indica si la pieza es inicial o no.
- Id_tipo_pieza: Este campo enlaza con la tabla siguiente.

Se continua con la tabla ”tipo_piezas”, tabla sencilla con los tipos de piezas posibles, solo contiene un identificador y el nombre del tipo de la pieza.

Por último se tiene la tabla “conexión” que relaciona las conexiones y las piezas a las que están asociadas, esta tabla guarda la información sobre las uniones explicadas anteriormente,

principalmente el número y el signo. Las columnas de esta tabla guardan la información siguiente:

- Id: Identificador de la conexión.
- Número: Dígito correspondiente a este enlace.
- Signo: Positivo o negativo de la unión.
- Obligatorio: Un booleano que indica si es obligatorio o no que la unión esté conectada a otra para poder completar el mueble.
- Id_pieza: El identificador de la pieza que tiene la conexión.
- X, Y, Z: Los valores de las coordenadas de la conexión.

Con todo esto se tiene información suficiente para gestionar la aplicación realizada, se pueden elegir diferentes tipos de muebles para crear, usar diferentes tipos de piezas y manejar las conexiones entre estas.

5.3.¿COMO SE MONTA UN OBJETO FINAL?

El proyecto realizado consta de dos herramientas, la aplicación de diseño de muebles, y la aplicación de inserción de conexiones. Para montar un objeto se hace uso de la aplicación de diseño y a continuación se explican los pasos a seguir para hacer tal cosa.

Al inicio de la aplicación se tiene un único botón “Crear Mueble”, al pulsarlo se hace una llamada al servidor web que devuelve la información de los tipos de mueble que hay introducidos en el sistema (sillas, mesas,...) y estos se visualizan en forma de nuevos botones. Al elegir uno se muestran los muebles de ese tipo y por último se da la opción de escoger la pieza inicial de entre las posibles con las que se trabajará.

Una vez realizados estos pasos iniciales de elegir lo que se quiere diseñar se puede ver en pantalla el modelo 3D de la pieza inicial. Se puede mover el ángulo la cámara haciendo clic izquierdo y manteniendo el botón en el ratón, desplazarla manteniendo el botón derecho y hacer zoom clicando y manteniendo el botón central de la cámara. Al clicar sobre la pieza inicial se despliega un menú con las

diferentes opciones. Estas opciones son “Añadir”, “Modificar”, “Borrar”.

La función “Añadir” permite, siempre que sea posible, agregar piezas a la pieza base que se tiene o a la pieza seleccionada en ese momento. Al elegir esta opción se abre una nueva ventana en la que se ha hecho una consulta al servidor y se muestran las opciones posibles de piezas a añadir. Al elegir una de ellas, se mostrarán en color amarillo semitransparente las posibilidades de colocación de dichas piezas, al hacer clic sobre estos objetos se confirmará la colocación de esa pieza, se puede elegir de entre todas las posibilidades cuales queremos confirmar clicando sobre ellas y luego usar el botón de parar la introducción. De esta manera, a base de una serie de pulsaciones y agregaciones de piezas es posible tener rápidamente un mueble diseñado.

La siguiente función, la de “Modificar” como su propio nombre indica, da la opción de cambiar la pieza seleccionada por otra del mismo tipo. Se hace una llamada al servidor que da las diferentes opciones posibles y al clicar sobre la deseada se modifica la pieza.

Por último la opción de “Borrar” simplemente permite eliminar la pieza escogida exceptuando que esta sea la pieza inicial en cuyo caso se nos avisa de que esta opción no es posible.

En cuanto a la programación de esta parte cabe destacar la metodología empleada. Una de las características de Unity es que un control gráfico, como por ejemplo un botón, no puede dar directamente como resultado de su pulsación otro elemento gráfico, por lo que se tiene que adoptar algún tipo de solución alternativa para conseguir que al pulsar un botón se generen otros elementos de la interfaz gráfica como listas de botones, ventanas, etc.

La función “OnGUI()” de Unity es la que se encarga de renderizar y manejar los eventos de la interfaz gráfica. Esta función se llama una vez por cada evento gráfico por lo que se llama varias veces por frame. Dentro de esta función es donde se debe colocar el código que resuelva el problema.

La solución al problema planteado es algo similar a una máquina de estados, en la que cada estado es una de las pantallas a

dibujar, con sus botones o controles específicos para ese momento o "estado" de la aplicación. La entradas y salidas de la máquina serían las acciones que desencadenan que se vaya a un estado u otro, en este caso será casi siempre la pulsación de un botón.

Así pues lo primero que se hace es definir un tipo enumerado en el que estarán todos los estados de la aplicación.

```
//Enum estados del programa
public enum tEstado {Idle, Inicial, Tipo_Mueble, Selec_Mueble, Selec_PiezaInicial, Pieza_Inicial, Elegir_Accion,
    Anadir, Modificar, Borrar, DescargandoPiezasPosibles, Aviso_Pieza_Inicial, DescargandoModificaciones,
    PosicionesPosibles, DescargandoAnclas};
tEstado estado = tEstado.Inicial;
```

Figura 25. Tipo enumerado con los estados del programa.

Entonces ahora cada estado del programa se cubren dentro de una sentencia condicional, "Si estado==tEstado.actual" entonces se dibujan las cosas correspondientes a ese estado. Al pulsar un botón en el estado actual se desencadena un cambio de estado, esto se hace simplemente asignado a la variable de estado del programa el estado correspondiente del tipo enumerado.

Para realizar todo esto de manera más organizada se dibujó un diagrama de estados con las diferentes visualizaciones del programa y como unión entre ellas se usaban flechas dirigidas cuyos nombres eran a acción a realizar para pasar al estado siguiente.

5.4.¿COMO SE INSERTAR LAS CONEXIONES?.

En este caso se usa otra herramienta, programada también con el motor de desarrollo Unity.

El inicio del procedimiento es muy similar al anterior, primero se elige el tipo de pieza en el que se clasifica la pieza a la que se quiere añadir conexiones, después se elige de entre ese grupo la pieza específica que se quiere. A continuación se carga el modelo

tridimensional de la parte y además se representaran los puntos de unión existentes a través de esferas.

Lo siguiente que se hace es clicar sobre la pieza en el punto en el que se quiera que exista una unión. Una nueva esfera aparecerá en el punto clicado y se abrirá una un cuadro de dialogo con los datos a rellenar, estos datos son los parámetros necesarios para definir una conexión que ya se ha explicado anteriormente, el número, el signo, el campo que define si es obligatorio u opcional rellenar esa conexión y las coordenadas de la misma. Aquí se puede cambiar de forma más precisa estos valores y ver en la esfera la situación de donde quedaría exactamente la conexión, hay que tener en cuenta que cuando se tiene una pieza que conecta con otras dos los puntos de unión tienen que estar en la misma posición para que esa unión sea posible, por lo que es recomendable el calcular mediante planos o esquemas donde caerán estos puntos de unión y luego introducirlos de forma precisa en la aplicación.

Al hacer clic sobre aceptar el punto guardado queda registrado para esa pieza y de esta manera la aplicación de diseño ya podrá hacer sus cálculos para unir las piezas y ofrecerlas para insertarlas cuando sea necesario.

En siguientes puntos de la memoria se explicará un caso de uso concreto con capturas de pantalla que ayudará a reforzar la explicación del funcionamiento de las aplicaciones realizada en los dos apartados anteriores.

5.5. OTRAS HERRAMIENTAS UTILIZADAS Y LA CONEXIÓN ENTRE ELLAS.

Ya se ha hablado en puntos anteriores de esta memoria de la principal herramienta utilizada para el desarrollo de esta aplicación, el motor gráfico Unity, no obstante en este apartado se detallaran otras herramientas utilizadas así como los distintos lenguajes utilizados para la programación.

5.5.1.- WAMP

WAMP es un acrónimo de **W**indows, **A**pache, **M**ySQL, **P**HP, es un sistema de infraestrucutra de internet que usa estas herramientas, Windows es el sistema operativo sobre el que trabaja, Apache es el servidor web que utiliza, MySQL es un gestor de bases de datos y PHP es un lenguaje de programación que explicaremos más adelante. WAMP permite servir paginas HTML internet, gestionar sus datos y también proporciona lenguajes de programación para desarrollar aplicaciones web. También existen sistemas análogos para otros sistemas operativos como LAMP (**L**inux AMP) o MAMP (**M**acintosh AMP).

En este caso se utilizó WAMP para crear un servidor virtual sobre el que realizar las pruebas de la aplicación sin necesidad de contratar un servidor web con el coste monetario que eso supondría.

5.5.2.-PHP

Es un lenguaje de programación de uso general de código del lado del servidor diseñado originalmente para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado de servidor en incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de sistemas operativos sin ningún costo.

Este lenguaje fue originalmente creado por Rasmus Lerdorf en 1995. En la actualidad PHP sigue siendo desarrollado con nuevas funciones por el grupo PHP. Este lenguaje forma parte del software libre publicado bajo a licencia PHP, que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término PHP.

PHP es un acrónimo recursivo que significa **P**HP **H**ypertext **P**re-processor (inicialmente PHP Tools, o, Personal Home Page Tools). Aunque como se ha dicho fue creado originalmente por Rasmus Lerdorf, la implementación principal de PHP es producida ahora por "The PHP Group" y sirve como el estándar da facto para PHP al no haber una especificación formal. Publicado bajo la PHP License, la Free Software Foundation considera esta licencia como software libre.

PHP puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas de manera gratuita. El lenguaje PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores. El enorme número de sitios en PHP ha visto reducida si cantidad a favor de otros nuevos lenguajes no tan potentes desde agosto de 2005. El sitio web de Wikipedia está desarrollado en PHP. Es también el módulo de Apache más popular entre las computadoras que utilizan apache como servidor web.

La gran similitud que tiene PHP con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones.

Aunque todo su diseño está orientado a facilitar la creación de sitios webs, es posible crear aplicaciones con una interfaz gráfica para el usuario, utilizando alguna extensión como por ejemplo PHP-Qt, PHP-GTK o WxPHP. También puede ser usado desde la línea de órdenes, de la misma manera como Perl o Python pueden hacerlo, a esta versión de PHP se la llama PHP-CLI (**C**ommand **L**ine **I**nterface).

Cuando el cliente hace una petición al servidor para que le envíe una página web, el servidor ejecuta el intérprete de PHP. Éste procesa el script solicitado que generará el contenido de manera dinámica (por ejemplo obteniendo información de una base de datos). El resultado es enviado por el intérprete al servidor, quien a su vez se lo envía al cliente.

Mediante extensiones es también posible la generación de archivos PDF, Flash, así como imágenes en diferentes formatos.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, PostgreSQL, Oracle, ODBC, DB2, Microsoft SQL Server, Forebird, SQLite.

PHP también tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos, tales como Unix y Microsoft Windows y puede interactuar con los servidores web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

5.5.3.- phpMyAdmin

Es una herramienta escrita en PHP con la intención de manejar la administración de MySQL a través de páginas web, utilizando internet. Actualmente puede crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 62 idiomas. Se encuentra disponible bajo la licencia GPL (General Public License).

Este proyecto se encuentra en vigor desde 1988, siendo el mejor evaluado en la comunidad de descargas de SourceForge.net como la descarga del mes de diciembre del 2002. Como esta herramienta corre en máquinas con Servidores Webs y Soporte de PHP y MySQL, la tecnología utilizada ha ido variando durante su desarrollo.

Tobias Rastschiller, por entonces consultor de IT y después fundador de Maguma, una compañía de software, comenzó a trabajar en la elaboración de una red administrativa basada en PHP cliente-servidor en MySQL en 1988 en inspirado por Peter Kuppelwieser y su MySQL-Webadmin. Cuando Rastschiller dejó el proyecto por falta de tiempo, así como el phpAdsNew del cual es también su inventor, el phpMyAdmin se había convertido en una de las aplicaciones PHP más populares, y las herramientas de

administración MySQL constituían una gran comunidad de usuarios y administradores. Es de anotar que esto incluía una buena contribución por parte de distribuidores Linux.

Para coordinar el creciente número de parches, tres desarrolladores de software, Olivier Müller, Marc Delisle y Loïc Chapeaux, registraron el proyecto phpMyAdmin en SourceForge.net y continuó su crecimiento en 2001.

5.5.4.-SQL

Viene de las siglas inglesas “Structured Query Language” o en castellano “Lenguaje de consulta estructurado”, es un lenguaje declarativo para acceder a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre ellas. Una de sus características es el manejo del algebra y el cálculo relacional que permite efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como realizar cambios sobre las mismas.

Los orígenes del SQL están ligados a los de las bases de datos relacionales. En 1970 Edgar Frank Codd propone el modelo relacional y asociado a este un sublenguaje de acceso a datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definieron el lenguaje SEQUEL (**S**tructured **E**nglish **QUE**ry **L**anguage) que más tarde fue ampliamente implementado por el sistema de gestión de bases de dato experimental “System R”, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por vez primera en 1979 en un producto comercial.

El SEQUEL fue el predecesor del SQL, que es una versión evolucionada del primero. El SQL pasó a ser el lenguaje principal de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el “SQL-86” o “SQL1”. Al año siguiente este estándar es también adoptado por la ISO.

No obstante, esta primera versión de SQL no abarcaba todas las necesidades de los desarrolladores e incluía funcionalidades de almacenamiento que se consideró suprimirlas. Así pues en 1992, se lanzó un nuevo estándar ampliado y revisado del SQL llamado “SQL-92” o “SQL2”.

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es en general muy amplio.

Más tarde vino en 1999 el “SQL: 1999” en el que se agregaron expresiones regulares, disparadores y algunas características orientadas a objetos. En el 2003 el SQL introduce algunas características de XML, la estandarización del objeto “sequence” y de las columnas autonuméricas. Dos años después el “SQL-2005” define las maneras en las cuales SQL puede usarse conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma de XML. También proporcionó facilidades que permitían a aplicaciones integrar dentro de su código el uso de “XQuery”, un lenguaje de consulta XML publicado por el World Wide Web Consortium para acceso concurrente a datos ordinarios SQL y documentos XML. En el 2008 se incluía la cláusula ORDER BY, se incluyeron los disparadores tipo INSTEADOF y se añadió la sentencia TRUNCATE.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.

Es un lenguaje declarativo de alto nivel o “de no procedimiento” que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y orientación a objetos. De esta forma una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros.

El lenguaje de definición de datos (LLD) de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación. También incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos, comandos para definir vistas y comandos para definir autorización de acceso a vistas y relaciones.

5.5.5.-C#

Lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma ".NET", que después fue aprobado como un estándar por la ECMA e ISO. C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma ".NET", similar al de JAVA, aunque incluye mejoras que provienen de otros lenguajes.

El nombre C-Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés Sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.

Aunque C# forma parte de la plataforma ".NET", ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya hay un compilador implementado que provee el marco Mono-DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Durante el desarrollo de la plataforma ".NET", las bibliotecas de clases fueron escritas originalmente usando un sistema de código gestionado llamado "Simple Managed C (SMC)". En enero de 1999, Anders Hejlsberg formó un equipo con la misión de desarrollar un nuevo lenguaje de programación llamado "COOL (Lenguaje C

orientado a objetos)". Este nombre tuvo que ser cambiado debido a problemas de marca, pasando a llamarse C#. La biblioteca de clases de la plataforma ".NET" fue migrada entonces al nuevo lenguaje.

5.5.6.-XML

Siglas en inglés de e**X**tensible **M**arkup **L**anguage ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos de forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones se deben comunicar entre sí o integrar información.

XML no ha nacido solo para su aplicación para internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XML proviene de un lenguaje inventado por IBM en los años setenta, llamado "GML (Generalized Markup Language)", que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO, por lo que en 1986 trabajaron para normalizarlo, creando SGML (Standard Generalized Markup Language), capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros sistemas para almacenar información.

En el año 1989 Tim Berners creó la web, y junto con ella el lenguaje HTML. Este lenguaje definió el marco de SGML y fue de

lejos la aplicación más conocida de este estándar. Los navegadores web sin embargo siempre han tenido pocas exigencias con respecto al código HTML que interpretan y así las páginas web son caóticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación de HTML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD. No se pueden combinar elementos de diferentes vocabularios. De la misma manera es imposible para un intérprete (por ejemplo un navegador) analizar el documento sin tener conocimiento de su gramática (del DTD). Por ejemplo, el navegador sabe que antes de una etiqueta debe haberse cerrado cualquier <p> previamente abierto. Los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico. Ambas opciones, de todos modos, son muy complejas para los navegadores.

Se buscó entonces definir un subconjunto del SGML que permitiera por un lado mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles. Por otro lado que permitiera la creación de analizadores simples, sin ninguna lógica especial para cada lenguaje. Y por último empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis

Para hacer todo esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos. XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que estas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. Ejemplos son un tema musical, que se compone de

compases, que están formados a su vez por notas. Estas partes se llaman elementos, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro definido. Las etiquetas tienen la forma <ejemplo>, donde "ejemplo" es el nombre del elemento que se está señalando. Aquí mostramos un ejemplo de la estructura de un documento XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
     <Mensaje>
          <Remitente>
               <Nombre>Nombre del remitente</Nombre>
               <Mail> Correo del remitente </Mail>
          </Remitente>
          <Destinatario>
               <Nombre>Nombre del destinatario</Nombre>
               <Mail>Correo del destinatario</Mail>
          </Destinatario>
          <Texto>
               <Asunto>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Asunto>
               <Parrafo>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Parrafo>
          </Texto>
     </Mensaje>
</Edit_Mensaje>
```

Figura 26. Esquema documento XML.

Fuente: Wikipedia.org.

5.5.7.- 3DMAX y creación de AssetBundles.

Para la creación de los modelos 3D básicos se ha utilizado el programa 3DStudioMAX los modelos de prueba usados en la aplicación son modelos básicos como cubos y esferas. Estos modelos tienen que ser exportados al tipo ".FBX" o a algún otro tipo que sea compatible con Unity y mantenga su geometría por lo que cualquier otro programa de modelado 3d que pudiera crear este tipo de archivos también sería útil.

Al importar estos modelos en Unity hay que convertirlos en Assetbundles. Los “AssetBundles” (la traducción al castellano sería “Conjunto de Assets”) son archivos que se pueden exportar desde Unity para contener activos de elección propia. Estos archivos hacen uso de un formato comprimido patentado y pueden descargarse bajo demanda por la aplicación. Los “AssetBundles” han sido diseñados para simplificar el proceso de descarga de contenidos a las aplicaciones. Se requiere de una versión pro de Unity para utilizar este tipo de objetos.

El flujo de trabajo con “AssetBundles” es el siguiente:

1. Se crean los “AssetBundles”. Para esto se puede hacer acopio de un script que proporciona la página web de Unity. Al colocar este script en una carpeta llamada “Editor” en el panel de proyecto, se añade una función al editor de Unity para crear este tipo de archivos.
2. Subir el “AssetBundles” al almacenamiento web. Unity no contiene funcionalidades para hacer esto. Se deberían subir los archivos al servidor, normalmente usando un cliente FTP como por ejemplo “Filezilla”.
3. Descargar “AssetBundles” en tiempo de ejecución. Esto se hace utilizando scripts, concretamente usando la clase “WWW” que ya se ha explicado en puntos anteriores de esta memoria. Accedemos a la URL del “AssetBundle” y se crea una instancia de la clase del mismo nombre.
4. Una vez descargado podemos acceder a activos individuales dentro del conjunto. Para ello se utilizan las funciones “AssetBundle.Load”, “AssetBundle.LoadAsync”, “AssetBundle.LoadAll”.

De esta manera ya se puede cargar en las escenas modelos3D que almacenen en un servidor web. Estos archivos tendrán la extensión “.unity3d”.

5.5.8.-CONEXIÓN ENTRE LAS HERRAMIENTAS

Al tener los archivos con extensión ".unity3d" ya se pueden colocar estos en una carpeta en el servidor web para que la aplicación tenga acceso a ellos de manera remota.

Como ya se ha dicho utilizamos el lenguaje C# para trabajar con Unity y programar sus scripts. En concreto cada vez que se quiera acceder al servidor para recuperar o insertar información se utilizará la clase "WWW" y una llamada a corutina para pedir un servicio al servidor. Este servicio se programará con el lenguaje "PHP", si se quiere insertar, borrar o editar información cuando llamamos al fichero "PHP" directamente se le pasan los parámetros necesarios en la URL. En cambio para la consulta de datos, el fichero "PHP" devuelve la respuesta en forma de fichero "XML", así, con unas funciones específicas de Unity para el tratamiento de ficheros XML se puede acceder de manera más sencilla y eficiente a la información de la base de datos. Los ficheros "PHP" también permanecen almacenados en el servidor por lo que el acceso a ellos se hace a través de una URL que indica donde se encuentran alojados. Estos lo que hacen principalmente es lanzar una consulta SQL al servidor cual devuelve una respuesta y el fichero PHP la organiza de manera que imprime la respuesta en formato XML.

La herramienta "phpmyAdmin" permite visualizar las tablas de nuestra base de datos e introducir y modificar tablas. Se ha usado esta herramienta para introducir de manera manual los datos de las piezas y los muebles ya que es de esta base de datos donde la aplicación saca la información que necesita, en el caso de las conexiones se introducen de manera automática a través de la aplicación de administración hecha para ello. El resto de su uso en este caso ha sido principalmente para tareas de depuración, ya que permite entre otras cosas hacer consultas SQL sobre la base de datos.

6. EJEMPLO DE USO.

En esta sección se ofrecerá un ejemplo de uso de las herramientas explicadas, acompañando con capturas de pantalla que ayudarán al mejor entendimiento de la aplicación en general y a tener una visión más clara de su interfaz gráfica de usuario.

6.1. HERRAMIENTA DE DISEÑO

Nada más iniciar la aplicación solo disponemos del botón de "Crear Mueble" así que se pulsa y a continuación se nos ofrecen los tipos de muebles a crear disponibles en el sistema.

Figura 27. Aplicación al iniciar.

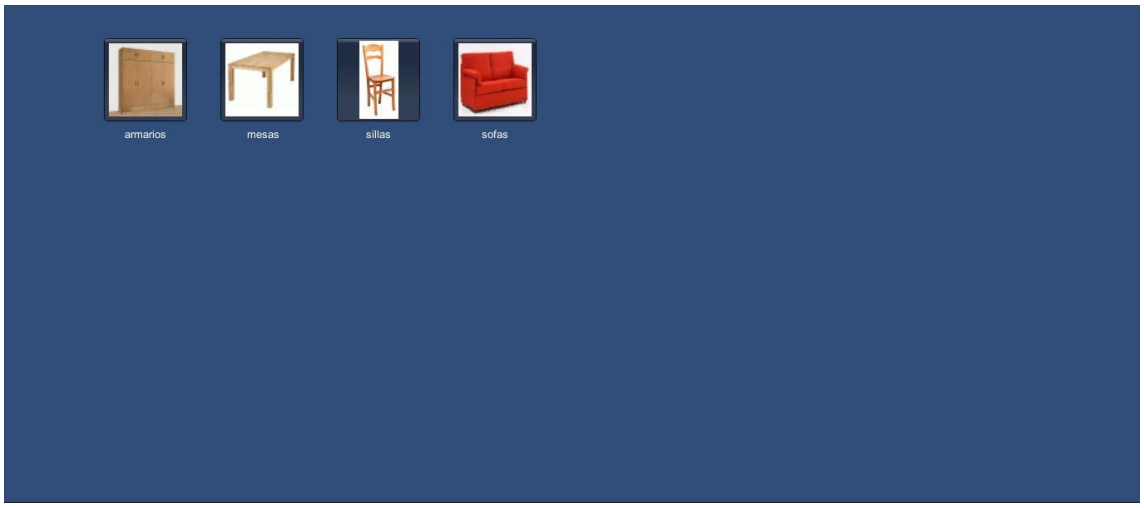

Figura 28. Tipos de mueble ofrecidos por el sistema.

Luego al elegir uno de estos tipos la aplicación ofrecerá los muebles de ese tipo, en este ejemplo se va a realizar la configuración de una “silla cuadrada” refiriéndose por cuadrada a la forma del asiento de la silla.

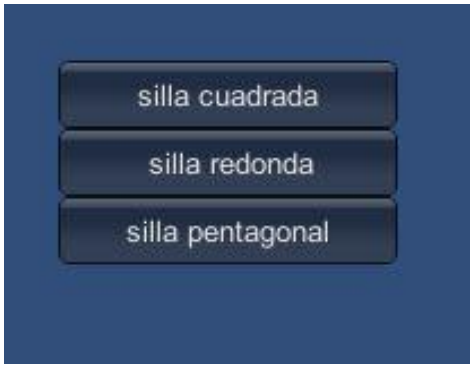

Figura 29. Muebles concretos de un tipo.

Ahora al elegir uno de estos tipos se nos dará la opción de elegir entre las diferentes piezas iniciales, por ejemplo para la “silla cuadrada” la pieza inicial es el asiento de la silla, luego el sistema nos dejará elegir entre los diferentes asientos cuadrados que tenga en la base de datos.

Figura 30. Elección de pieza inicial.

Al seleccionar la pieza inicial esta se cargará en el visor, en este momento ya se tiene libertad de movimiento con la cámara, con el botón izquierdo del ratón se mueve el ángulo de la cámara, con el botón derecho se mueve la posición de la cámara y con el botón central del ratón se hace zoom. Dicho esto al clicar sobre la pieza inicial, esta cambiará de color (indicando que esta seleccionada) y se ofrecerán 3 opciones, añadir, modificar y borrar. En este ejemplo se explicará la opción de añadir ya que es la más completa y con ella se realizará una silla entera.

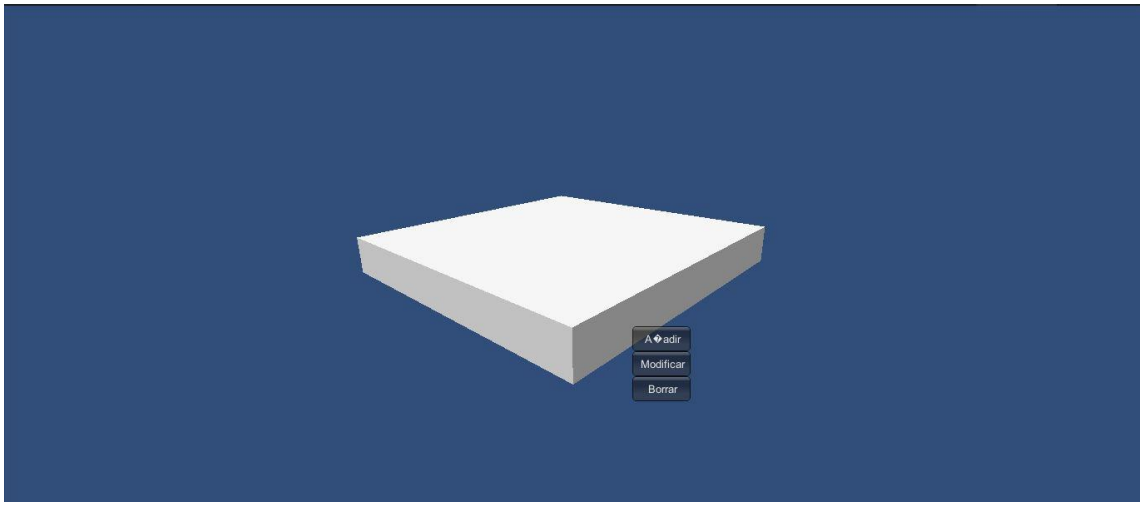

Figura 31. Acciones posibles al seleccionar una pieza.

Cuando elegimos la opción de “Añadir” se nos abre la siguiente ventana:

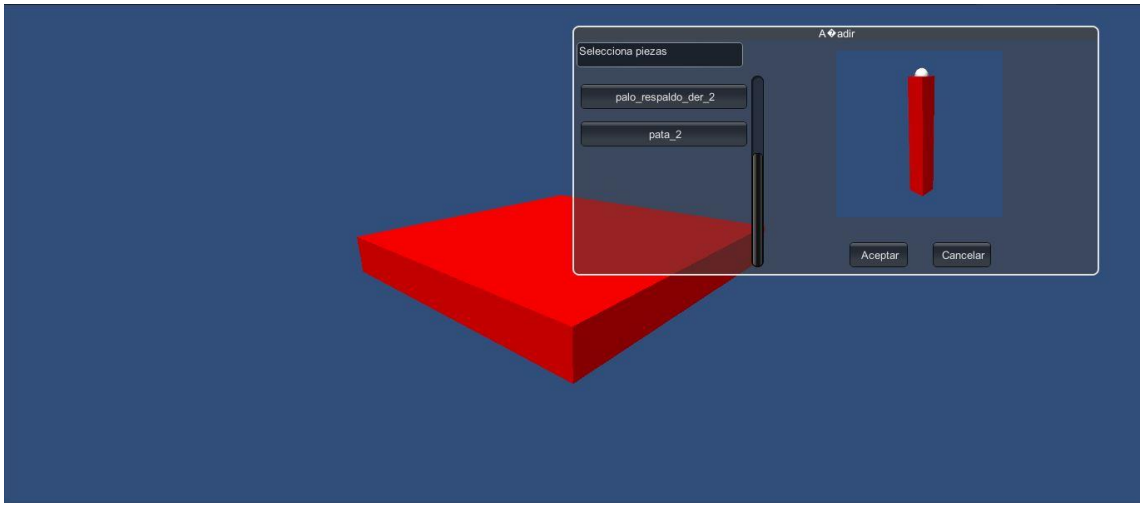

Figura 32. Ventana de agregar pieza.

Como se puede ver se ofrece por un lado el nombre de las piezas que es posible introducir y por otro una vista previa de esta pieza. Al elegir la pieza deseada y darle a “Aceptar” se ofrecerán las localizaciones posibles para esa pieza. En este caso se ha optado por colocar primero las patas de la silla.

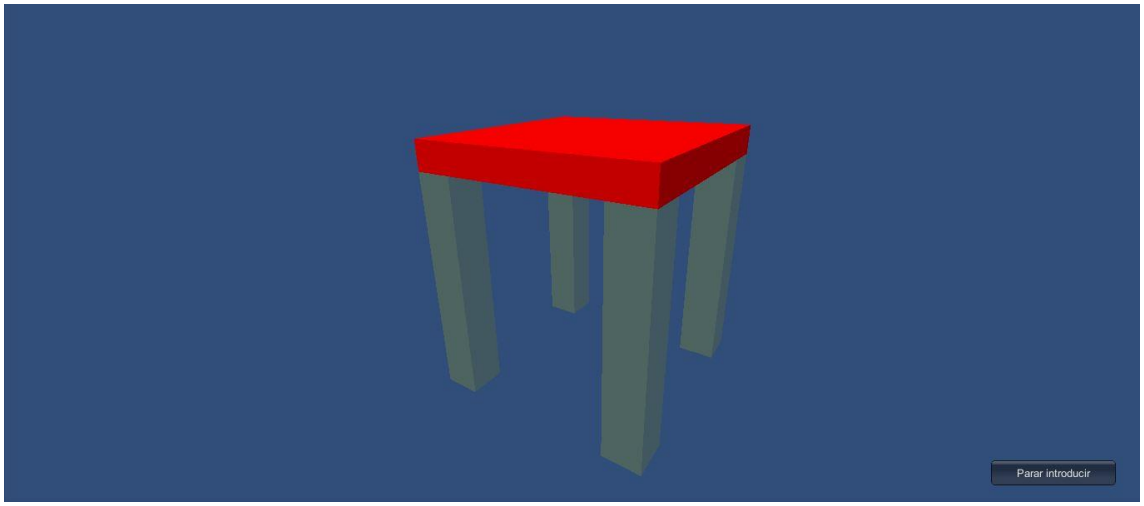

Figura 33. Colocación posible de la pieza seleccionada.

Se hace clic sobre estos objetos semitransparentes para confirmar que se quiere agregar dicho objeto en esa posición y luego

se hace clic sobre el botón “Parar introducir” para confirmar los cambios.

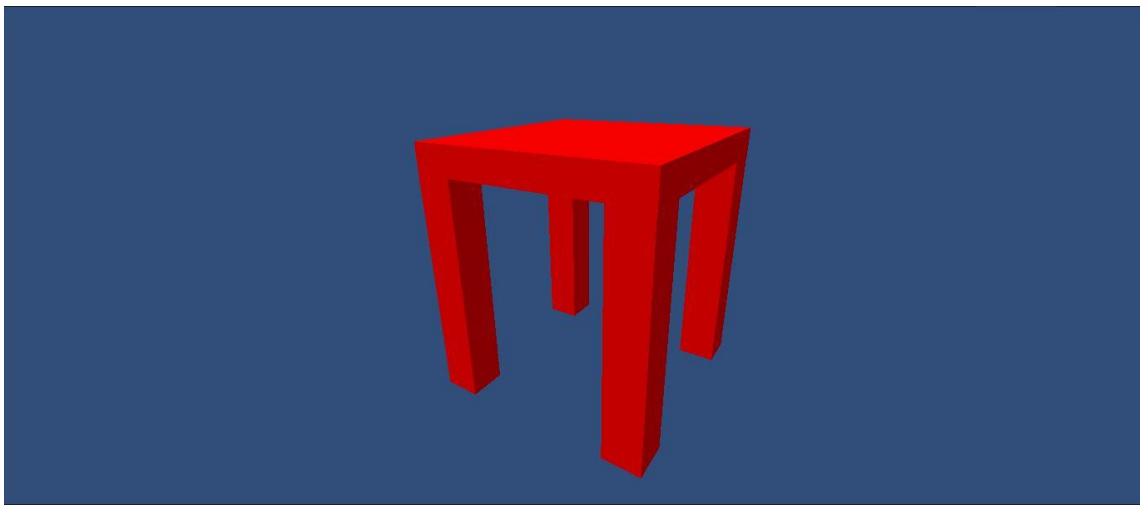

Figura 34. Resultado final de la selección.

Ahora simplemente se repite este mismo proceso hasta que se tiene la silla completa. El resultado final dependediendo de las piezas elegidas será algo similar a la siguiente figura.

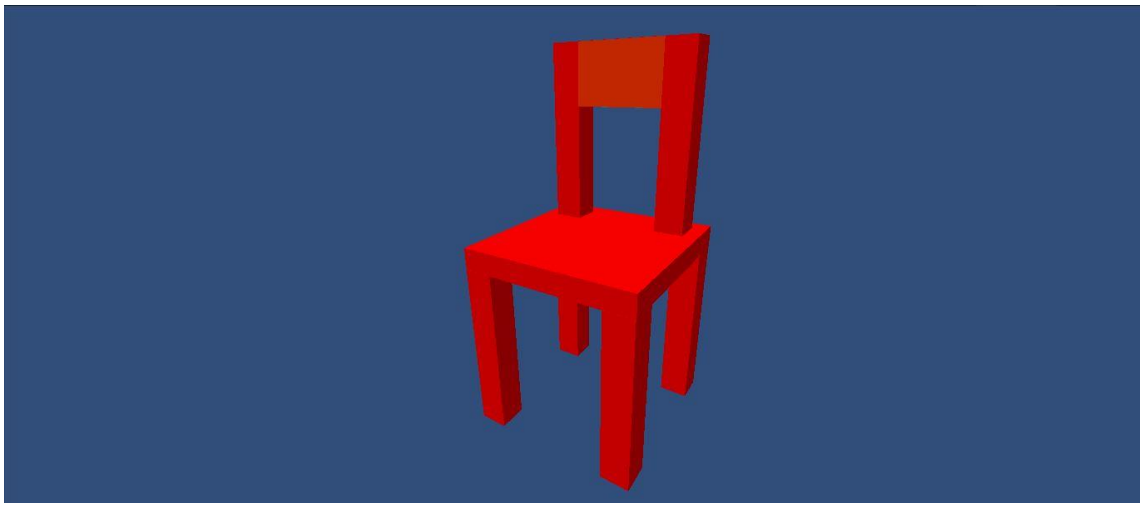

Figura 35. Resultado final del diseño.

6.2. HERRAMIENTA DE INTRODUCCIÓN DE CONEXIONES.

La primera parte de esta herramienta es muy similar a la anterior, esta vez el boton inicial es el boton de “Definir Conexiones” y a continuación se elege primero entre los tipos de piezas posibles (patas, asientos, respaldos, etc) y segundo se selecciona la pieza en concreto que se quiere.

Figura 36. Estado inicial herramienta de conexiones.

Hecho esto se tendrá una vista del objeto como la que se muestra a continuacion, las esferas blancas reflejan la posicion donde estan ya definidos para esa pieza los puntos de anclaje.

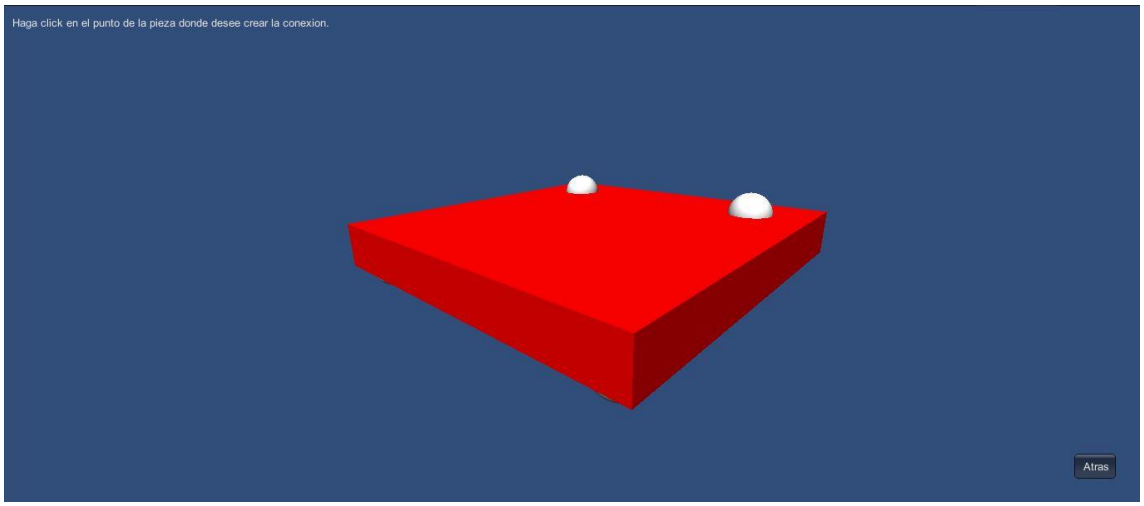

Figura 37. Pieza para insertar conexiones.

Al clicar sobre un punto cualquiera del objeto se definirá una nueva esfera y aparecerá el cuadro que se muestra a continuación(figura 38). Como se puede ver contiene campos de texto y controles que permiten definir los parametros necesarios para crear una conexión. Notar también que gracias a los campos “X,Y,Z” se puede ver en tiempo real el movimiento de la esfera de conexión, de esta manera es posible situar la conexión con mucha más precisión que con el simple clic del raton.

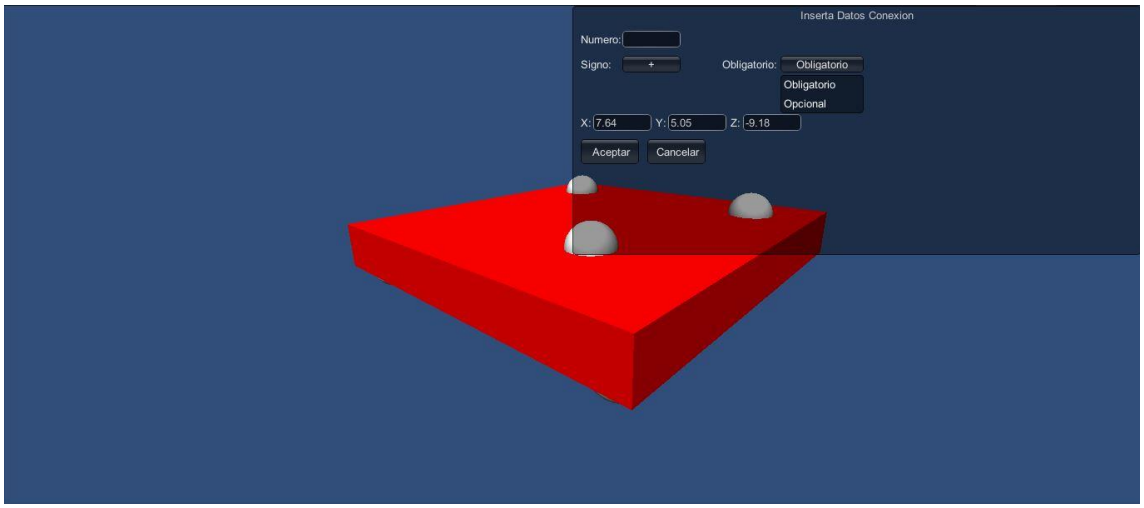

Figura 38. Ventana de introducción de conexión nueva.

Aquí concluye la metodología para insertar nuevas conexiones, si se quisiera insertar más conexiones habría que repetir el proceso hasta obtener las conexiones deseadas.

7. CONCLUSIONES

Se ha conseguido resolver el problema inicial planteado, se ha realizado una aplicación que permite el diseño de muebles, a través de un proceso de selección inicial, se llega a la pieza base del objeto a construir y una vez aquí se van agregando piezas una a una, con posibilidades de modificación hasta que tenemos el resultado final deseado de manera muy sencilla. Todo este proceso se acompaña de una serie de imágenes que hace que la aplicación sea mucho más intuitiva y rápida de usar. Además, gracias a la polivalencia de Unity se pueden añadir objetos 3D simples o complejos al visor y el resultado seguiría siendo adecuado.

Por otro lado con la herramienta de inserción se puede en pocos pasos añadir conexiones simples y más aún se puede hacer con una gran precisión gracias los campos que permiten cambiar las coordenadas de cada punto de anclaje.

En el futuro podrían introducirse una serie de mejoras a la aplicación como por ejemplo las siguientes:

- Herramienta de simetría para la herramienta de inserción: Actualmente los puntos de unión entre piezas se introducen con un clic y luego de manera más precisa podemos moverlo en cualquiera de los 3 ejes de un espacio tridimensional. Una de las características que contienen los muebles es que muchos de ellos son simétricos o al menos contienen muchas partes simétricas, por lo que es lógico pensar que los puntos de unión entre de sus partes también son simétricos. En este contexto sería interesante introducir una funcionalidad a la herramienta que permitiese establecer ejes de simetría o que insertara puntos de unión simétricos de manera automática. Esto podría hacerse por ejemplo haciendo el cálculo de la distancia de un punto de inserción hasta un eje perpendicular situado empezando en el punto medio del objeto en ese eje de coordenadas. También podría hacerse la creación de ejes de simetría y ver en tiempo real donde “caería” el punto simétrico a otro seleccionado a través de ese eje.

- Automatización del proceso de introducir piezas y modelos en el servidor: El proceso de introducir datos en la base de datos del servidor se hace mediante la herramienta “phpMyAdmin” y la colocación de archivos en el servidor se tiene que hacer a través de un cliente FTP, este proceso es costoso temporalmente y complicado de realizar para personas no acostumbradas a trabajar con medios informáticos complejos. Por este motivo la realización de otra aplicación o herramienta que automatizara o al menos hiciera más intuitivo todo este proceso sería una gran mejora para la aplicación global. Se podría realizar una interfaz gráfica que permitiese cargar un modelo 3D y a través de una serie de controles, de manera similar a la herramienta de inserción, permitiera introducir los datos de la pieza y las asociaciones correspondientes con otros objetos de manera más intuitiva y gráfica, sería bueno que incluyera algún tipo de visor de la pieza con la que se está trabajando en el momento.

- Mejora de la interfaz gráfica: Las herramientas programadas utilizan los recursos y características propias de Unity para la creación y diseño de interfaces gráficas. Aunque útiles y completamente funcionales carecen de ciertas propiedades gráficas que harían la aplicación visualmente más atractiva. Unity dispone de funciones para cambiar los estilos y propiedades de los controles y ventanas y hacer estos totalmente personalizados aunque esto aumenta ligeramente la programación de la aplicación. Aun así mejorar la interfaz gráfica supondría una mejora considerable respecto a la opinión de los futuros usuarios y le daría un aspecto más profesional a la aplicación.

- Inclusión de otro tipo de conexiones: Las uniones consideradas son las uniones básicas de tipo “macho-hembra”, no obstante podrían introducirse otro tipo de conexiones para aumentar el dominio de objetos posibles a construir, como por ejemplo conexiones de tipo rail, como las que hay en los

cajones por ejemplo, o conexiones de tipo angular para tener en cuenta cosas como las bisagras de las puertas.

- Control de la topología del objeto: La herramienta de diseño permite crear los objetos sin ningún tipo de restricción por lo que se permite crear modelos que serían físicamente posibles pero no coherentes, por ejemplo una silla no puede estar compuesta por el asiento y una pata, en cambio la aplicación sí que permite diseñar este objeto. Una mejora notable sería la inclusión de restricciones que obligaran o comprobaran que los objetos creados son topológicamente correctos.

El uso de la aplicación está enfocado principalmente a al diseño muebles por lo que tendría su posible uso en esta industria. Este configurador podría usarse como muestra al público para que generase muebles a su gusto con las partes que ofreciera el proveedor, sería un elemento básico que ayudaría al cliente a tener una mejor visión de cómo quedaría el resultado final.

De otra manera teniendo en cuenta que aunque se han diseñado las herramientas partiendo de la idea de construir muebles, la aplicación en si trabaja con “partes” y las “conexiones” entre ellas. Desde esta perspectiva se podría crear una gran cantidad de objetos posibles y teniendo en cuenta el cómo funciona la herramienta de diseño podría utilizarse a modo de guía interactiva de construcción de estos objetos. Empezaríamos de una pieza inicial y las siguientes piezas a colocar se indicarían de manera semitransparente como vimos en el apartado de “Ejemplo de uso” (ver figura 33), una vez colocadas seguiríamos el proceso hasta tener montado el objeto real completo.

En conclusión se ha realizado un proyecto que ha resuelto el problema planteado proponiendo un modelo de conexión para las piezas que como se ha demostrado ha permitido definir de manera correcta la construcción incremental de un objeto. El programa desarrollado puede tener aplicaciones en la vida real que serían útiles en diversos aspectos y con una sencilla interfaz adaptada al

uso de todos los usuarios, estas propiedades podrían mejorarse aún más haciendo inclusión de las mejoras propuestas en esta sección.

8. BIBLIOGRAFIA

-http://webdiis.unizar.es/~SANDRA/MasterIG/ModGeometrico13-14.pdf

-http://www.fing.edu.uy/inco/cursos/compgraf/Clases/2012/10-Modelado%20de%20Solidos.pdf

-http://lsi.ugr.es/~cad/teoria/Tema5/RESUMENTEMA5.PDF

-http://es.wikipedia.org/wiki/Autodesk_3ds_Max

-http://es.wikipedia.org/wiki/Blender

-http://es.wikipedia.org/wiki/Maya_(programa)

-http://www.autodesk.es/products/autodesk-3ds-max/overview

-http://www.arquitectura.com/cad/artic/elcad.asp

-http://www.slideshare.net/armintilano/modelo-de-base-de-datos-orientados-a-objetos-9861470#

-http://www.slideshare.net/PabloTlv/sgbdoo-13572292

-http://blearning.itmina.edu.mx/dep/sada/carreras/Ingenieria%20en%20Sistemas%20Computacionales/4to%20Semestre/Fundamentos%20de%20Bases%20de%20Datos/fundamentos_bd/BasesDeDatosOrientadasAObjetos.pdf

-http://docencia.lbd.udc.es/bd3/teoria/t1/bdoo.pdf

-http://docs.unity3d.com/Manual/

-http://answers.unity3d.com/

-http://stackoverflow.com/

-http://es.wikipedia.org/wiki/C_Sharp

-http://es.wikipedia.org/wiki/PhpMyAdmin

-http://es.wikipedia.org/wiki/WAMP

-http://es.wikipedia.org/wiki/Extensible_Markup_Language

-http://es.wikipedia.org/wiki/PHP

- http://es.wikipedia.org/wiki/SQL