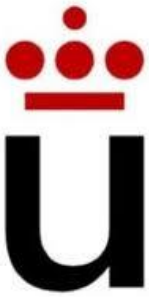

UNIVERSIDAD REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2011/2012

Proyecto de Fin de Carrera

DISEÑO, ANIMACIÓN Y VISUALIZACIÓN INMERSIVA DE UN LAMBORGHINI GALLARDO EN UNA CUEVA DE REALIDAD VIRTUAL

Autor: Ángel Pareja León

Tutores: Álvaro Pérez Molero

Luis Miguel Serrano Gómez

Agradecimientos

Una larga senda ya recorrida, tanto en el ámbito profesional como el personal. Hace unos años comencé el proyecto con ilusión y hambre, hambre que se fue desvaneciendo poco a poco tras mi incorporación en el mundo laboral. Han sido bastantes circunstancias que me ha tocado sortear pero por suerte no he tenido que sortearlas solo.

Quiero dar gracias a todos aquellos que me han dado los pequeños empujones que necesitaba cada cierto tiempo, a los que comprendieron las situaciones personales y profesionales que han ido asaltándome durante este empedrado camino y a los que me han estado prestando la atención necesaria y la ayuda técnica para la consecución completa de este proyecto.

Gracias a todos los que han estado a mi lado en esta larga e importante etapa de mi vida.

Resumen

En los últimos años, y gracias al avance tecnológico en realidad virtual, el interés de la utilización de la misma en el campo de la docencia para apoyar el aprendizaje ha aumentado considerablemente. La utilización de juegos y aplicaciones interactivas para captar la atención del alumno se ha convertido en una nueva rama de usos y utilidad de la informática.

Este Proyecto de Fin de Carrera pretende realizar una aplicación en tiempo real para su visualización en la Cueva de Realidad Virtual de la Universidad Rey Juan Carlos, enfocada a mostrar un vehículo de gran espectacularidad, junto con las principales piezas que forman parte de su motor. De este modo, se plantea una solución que concuerda con las bases fundamentales para favorecer el aprendizaje mediante un sistema multimedia atractivo y sencillo para el usuario.

La aplicación permite la visualización individual de cada una de estas piezas y su posterior montaje, así como la navegación por un entorno simulado donde aparece el modelo completo del vehículo, tanto en su interior como en el exterior. Mediante esta aplicación se proporciona al usuario o alumno un sistema de entretenimiento a la par que instructivo aumentando sustancialmente el grado de aprendizaje durante su utilización.

Este aprendizaje se ve reforzado a través de una animación en alta definición del vehículo completo y del funcionamiento básico de las distintas partes del motor dentro del denominado Ciclo de Otto.

Índice

Tabla de ilustraciones

CAPÍTULO 1 - Introducción y objetivos

Para la realización del proyecto, se ha optado por realizar un modelo virtual del automóvil Lamborghini Gallardo. Se ha de mencionar que, sobre el mismo modelo, existen diferentes versiones de la carrocería. Este proyecto está basado principalmente en la primera versión de dicho modelo, que apareció en el año 2002, sustituyendo en el mercado de los superdeportivos al mitificado Lamborghini Diablo y esperando competir cara a cara con otros grandes superdeportivos como el Ferrari F430 o Porsche 911 Carrera Turbo. El vehículo posee dos plazas con un motor central trasero longitudinal, disponible en carrocerías cupé y descapotable. Aunque la marca es históricamente de origen italiano y su fábrica se encuentra en Italia, pertenece al grupo alemán ***V****olks****w****agenwerk* ***A****ktien****g****esellschaft*, más conocido como grupo VAG, dueño de marcas tan conocidas como Volkswagen, Audi, Seat o Skoda, o tan exclusivas como Bentley y Bugatti.

El motor del modelo se corresponde con un motor de diez cilindros en V fabricado en aluminio desarrollado por el grupo VAG. Mide 4961 cm^3, por lo que la velocidad media del pistón llega a casi 25 m/s a 8000 rpm. La culata tiene doble árbol de levas movidos por cadena y cuatro válvulas por cilindro. Alcanza una potencia de 500 CV (367 kW) a 7800rpm. El par máximo es 510 Nm a 4500 rpm, un régimen bajo teniendo en cuenta que es un motor que puede alcanzar los 8000 rpm en primera, es decir, puede alcanzar los 104 kilómetros en sólo el primer cambio. Da 408 Nm a 1500 rpm y conserva 449 Nm a régimen de potencia máxima.

El motor es el mismo utilizado en otros modelos del resto de marcas del grupo, como el reciente Audi R8 o la mítica berlina de representación Audi RS8, con los que comparte bastidor entre otros elementos.

El objetivo fundamental del proyecto consiste en la realización de un modelado tridimensional sobre un vehículo a motor y, a partir de éste, plantear dos soluciones distintas, como son una aplicación tridimensional en tiempo real y una animación, para

ofrecer un método educativo, vistoso y lúdico para comprender las principales piezas que forman parte de un motor de combustión interna.

Con el fin de conseguir una aplicación en tiempo real lo suficientemente fluida, se ha tenido que realizar un proceso de optimización poligonal, disminuyendo, en la medida de lo posible, el número de polígonos presentes en la escena tridimensional y cuidando que no se pierda el atractivo visual que genera un vehículo de semejantes características. Todo este proceso de optimización se ve nuevamente apoyado mediante diversos algoritmos que ocultan polígonos dinámicamente en la aplicación.

Por otro lado, atendiendo a la animación tridimensional, se ha procedido con el estudio y composición de las diferentes tipologías de animación y conexión entre las piezas para facilitar y automatizar el proceso en todo momento, bastando con definir el movimiento de un pequeño número de piezas para que se muevan en concordancia el resto de piezas del motor. De forma análoga, se ha realizado el estudio de las diferentes técnicas para el aumento de realismo de la escena, vigilando en todo momento mantener unos tiempos de renderizado contenidos. Para este cometido se ha hecho mucho hincapié en la definición de materiales y texturas para aplicar sobre los objetos de la escena, así como los diferentes sistemas de iluminación (fuentes de luz, reflexión y refracción) en ésta.

CAPÍTULO 2 - Estado del arte. Modelos tridimensionales y virtualización

En este apartado se explicarán, de forma teórica, los conocimientos necesarios para la comprensión y realización de las diferentes partes del proyecto, haciendo hincapié en las técnicas de modelado y renderizado del entorno virtual, así como en la implantación del modelo en entorno virtual.

2.1 Introducción a la informática gráfica. Representación de objetos tridimensionales

Un modelo tridimensional es la representación de un objeto, cuerpo o sistema que se desea visualizar o estudiar a través de una simulación de su geometría. Por lo tanto, los modelos tridimensionales se pueden definir como la representación esquemática y visible de un conjunto de cuerpos que, una vez procesados, genera una imagen.

El proceso de la creación de gráficos tridimensionales parte de una serie de fórmulas matemáticas, las cuales generan gráficos en 3D. Estas fórmulas se corresponden con multitud de operaciones sobre puntos, vectores, ecuaciones lineales y matrices, como pueden ser sumas vectoriales, productos escalares, transposición de matrices, multiplicación de matrices, etc. El conjunto de fórmulas y operandos matemáticos sobre los que se aplican, describen las propiedades de tonalidad, textura, sombras, reflejos, transparencias, translucidez, reflexión, iluminación (directa, indirecta y global), profundidad de campo, desenfoques por movimiento, ambiente, punto de vista, etc. Por ejemplo, un vector de puntos representa un objeto tridimensional o malla y, sobre éste, se pueden realizar operaciones de rotación o traslación, multiplicando las matrices de rotación o traslación por el vector (x, y, z) que representa un punto en 3D.

La complejidad de la escena se define por el nivel de fidelidad a la realidad que se desea obtener: un aumento del nivel de realismo implica una mayor complejidad en la

construcción de la escena. Debido a ello, es necesario conocer a priori la estructura y el comportamiento de la escena real a modelar y los objetivos de estudio, análisis y simulación que se vayan a realizar sobre ella.

Existen diferentes técnicas de modelado según su funcionalidad, pudiendo dividirse en modelado de apariencia, cinemático o geométrico/poligonal. En una representación de una escena tridimensional es frecuente pasar por cada uno de los tipos de modelado.

2.2 Tipos de modelos

Existen a su vez diferentes tipos de modelos para representar los objetos de una escena tridimensional. El método de renderizado también es un factor que influye en la decisión del tipo de modelado a utilizar, puesto que según el método se determinan las características y efectos en el renderizado final. Un ejemplo concreto es la utilización de un modelado mediante curvas parametrizables para la construcción de un modelo poligonal de un vehículo. Este tipo de modelado puede generar la superficie del vehículo con gran detalle y permite el estudio de las deformaciones de ésta con mayor fidelidad que un modelado poligonal. Sin embargo, cualquier tipo de transformación en la superficie provoca que se deban realizar una cantidad mayor de cálculos numéricos, por lo que su uso no sería nada recomendable para aplicaciones en las que el tiempo de renderizado deba ser una de las principales virtudes.

2.3 Modelado poligonal

Una malla poligonal es la representación más común de un objeto tridimensional y, a su vez, una de las más tediosas. La popularidad de este método proviene de la facilidad de modelado y que la complejidad del objeto que se desea representar en raras ocasiones imposibilita utilizarlo, aunque no siempre sea el más idóneo.

Los objetos se representan por una red o malla de caras poligonales, formadas normalmente por triángulos o rectángulos planos (llamados *quads*). A su vez, los polígonos pueden ser clasificados en polígonos convexos o cóncavos. Intuitivamente se define un polígono como convexo cuando, trazando una línea recta entre dos puntos cualesquiera del contorno del polígono, todos los puntos del segmento que los une son puntos interiores del polígono. Por último, un polígono puede contener o no agujeros en su interior.

Esta especificación admite alguna variante en función del número de vértices del polígono, pero estas variantes suponen un problema a la hora de la manipulación y cálculo de propiedades del objeto. Las tarjetas gráficas actuales se encuentran optimizadas para la utilización de polígonos triangulares convexos.

Mediante esta representación, se puede representar con fidelidad cualquier objeto. El número final de polígonos que componen la malla poligonal del objeto define el grado de realismo y fidelidad del modelo. Por otro lado, otros factores como la resolución de visualización de la escena y el tamaño del objeto en la pantalla definen el nivel de detalle adecuado para representar en un momento dado un objeto. A mayor resolución de pantalla, se vuelve estrictamente necesario aumentar el número de polígonos del objeto. Lo mismo ocurre con el tamaño del objeto, carece de sentido utilizar un objeto formado por un gran número de polígonos y con gran detalle si en la escena este objeto posee un tamaño que imposibilita la visualización de ese detalle, viéndose mermado el rendimiento computacional de la escena sin que realmente se aporte nada a ella.

Uno de los desarrollos más importantes en los gráficos computacionales fue la aparición de los algoritmos de sombreado durante de la década de los 70. Estos algoritmos permiten que, a partir de modelos poligonales relativamente simples, se obtenga una interpolación visual en el objeto, generando que su representación se vea mucho más suavizada obteniendo multitud de nuevas aristas entres las diferentes caras.

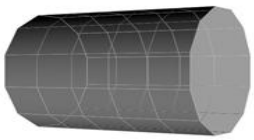

Ilustración 1 Aproximación cilindro mediante planos

Esta forma de aproximación de las superficies curvas deriva muchas complicaciones a la hora de conseguir una aproximación suficiente, siendo el criterio de aproximación una propiedad normalmente arbitraria. El tamaño de los diferentes polígonos es crítico a la hora de realizar la aproximación: en los lugares donde la curva cambia rápidamente es necesaria la utilización de un mayor número de polígonos por unidad de área.

En el caso más simple, una malla poligonal es una estructura de polígonos representados por una lista de coordenadas espaciales que forman los vértices del polígono, los cuales se unen entre sí mediante aristas. Aparte de esta información, se almacenan otros datos para su correcto procesado como son los vectores normales a los vértices, lo que facilita los cálculos posteriores, como las transformaciones lineales que se aplican sobre el objeto.

Por lo tanto, se ha de considerar que los polígonos se almacenan como una estructura jerárquica de información:

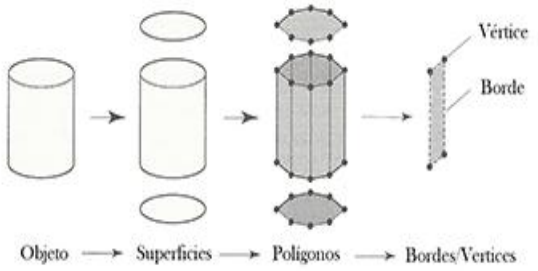

Ilustración 2 Estructura de un objeto poligonal

Los polígonos se agrupan en superficies y las superficies se agrupan en objetos. Por ejemplo, un cilindro se compone normalmente de tres superficies, la cara superior, la inferior y una cara curva que une ambas. La razón de este tipo de agrupación se debe a que

se ha de distinguir entre los bordes que existen entre las caras superiores y la aproximación que se hace para la superficie curva que se establece entre éstas. A su vez, esta aproximación se compone de una serie de polígonos planos definidos por vértices y aristas.

Esta información anterior tan solo describe la geometría básica del objeto. Para el correcto renderizado o imagen final, se deben almacenar otros datos como punteros, números reales, flags y otra información práctica para su representación dentro de las aplicaciones. Esta información usualmente son: atributos poligonales (forzado triangular para facilitar el cálculo, área, vector normal de las caras del polígono, vértices convexos, existencia de agujeros), atributos de borde (longitud, si el borde se sitúa entre dos vértices o dos superficies, vértices que generan el borde) y atributos de vértice (polígonos de los que forma parte el vértice, vector normal del vértice, posición relativa del vértice en una imagen bidimensional).

2.3.1 Modelado manual de objetos poligonales

La forma más sencilla de modelar un objeto real simple es realizar un modelado manual tridimensional. Este método de modelado no requiere de dispositivos externos de elevado coste y alta complejidad. Un diseñador sitúa en el espacio los vértices del objeto para modelarlo. Normalmente, se coloca una foto de frente del objeto a modelar en la vista *front* y una de perfil en la vista *right*, para empezar a situar vértices, aristas y facetas. Las texturas tridimensionales pueden ser introducidas en el sistema a través de un digitalizador, añadiéndolas en un orden para generar las aristas que unen los vértices de forma correcta. Una forma muy efectiva de asegurarse que el objeto modelado es correcto consiste en representar líneas que definen visualmente cada una de las aristas del polígono, lo que comúnmente se conoce como modelo de alambre o *wireframe*.

El modelo de alambre no es el más adecuado para mostrar la representación de un objeto en la pantalla. Aunque su coste computacional es muy escaso, no es fácil hacerse una idea de cómo es el modelo, al no mostrar las superficies del objeto.

2.3.2 Generación automática de objetos poligonales

Este método de digitalización se nutre de un dispositivo que es capaz de modelar, con mucha precisión y/o a muy alta resolución, un modelo poligonal utilizando el rastreo mediante láser o escáneres tridimensionales de luz LED. El objeto se sitúa sobre una plataforma rotativa y es analizado con el láser de forma vertical u horizontal, obteniendo el contorno del objeto. Junto a un algoritmo de mapeado, el sistema genera la malla de puntos necesarios para obtener los polígonos externos del objeto. Como se ha comentado, este sistema tiene la peculiaridad de que sólo genera la superficie externa del objeto, por lo que si se desea modelar el interior se deberá utilizar complementariamente con otros algoritmos de modelado como la tetraedralización. Otro problema que presenta es que se debe atender con atención el nivel de detalle que se desea obtener, dependiendo éste de las capacidades del sistema y de la finalidad del modelo que se genera. Por último, y como nuevo inconveniente, para realizar este tipo de modelado, se necesita tener una superficie real y el objeto real a analizar, por lo que en muchas ocasiones se requiere obtener un modelo a escala real para poder realizar su escaneo.

2.3.3 Generación matemática de objetos poligonales

Otra forma de generar objetos poligonales es mediante una descripción numérica de curvas, funciones y posiciones espaciales. El ejemplo más común de esta metodología son las herramientas CAD.

El usuario trabaja con una idea sobre la forma que desea generar y construye un objeto individual de caras poligonales. En lugar de generar la forma final del objeto, se realizan una serie de transformaciones sobre este elemento simple.

2.3.4 Geometría constructiva de sólidos (CSG)

Esta técnica se basa en que la construcción de los objetos tridimensionales puede ser producida por combinaciones de formas elementales o primitivas geométricas. Por lo tanto, las primitivas simples se combinan a través de operadores booleanos, y cada objeto se almacena como un árbol binario, donde los nodos internos contienen operadores geométricos o booleanos y las hojas contienen las primitivas simples. Por ejemplo, generar una pieza de metal con un agujero en medio, se compone de la sustracción tridimensional entre un sólido rectangular y un cilindro. Este método es una representación volumétrica de primitivas, a diferencia de otros métodos donde sólo se definen las superficies externas del objeto.

El modelador construye la forma usando un árbol tridimensional de bloques que describe cómo éstos deben ser combinados. El interés de utilización de este tipo de representación es facilitar la interacción con el modelado de sólidos. La idea es que los objetos son normalmente partes de otros objetos fabricados posteriormente y de mayor complejidad.

Las primitivas mencionadas con anterioridad son por ejemplo esferas, conos, cilindros o sólidos rectangulares que se combinan, no solo de forma booleana, sino que también pueden sufrir transformaciones lineales para generar nuevas formas. Las combinaciones posibles pueden ser tanto de extracción, sustracción, unión, intersección, etc.

A continuación se muestran algunas formas simples generadas mediante CSG. La figura de la izquierda muestra la unión de dos posibles sólidos. Si consideramos los objetos como nubes de puntos, la unión de éstos engloba a todos los puntos que formaban parte de los dos sólidos iniciales. La figura de la derecha muestra el efecto surgido a través de una operación de sustracción. El operador sustraendo elimina del primer elemento toda la nube de puntos que coinciden entre ellos.

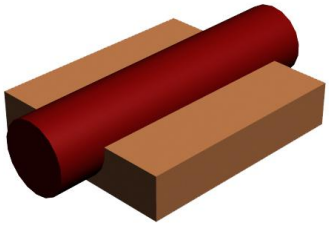

Ilustración 3 Unión de dos sólidos

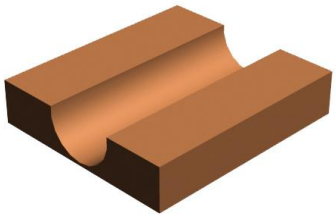

Ilustración 4 Sustracción de dos sólidos

Por último, se puede realizar sobre la misma figura una intersección obteniendo como resultado el sólido que contiene los elementos de los dos cuerpos.

Ilustración 5 Intersección de dos sólidos

El poder de esta forma de diseño queda demostrado por su simplicidad, pero presenta un problema computacional debido a que las transformaciones que se realicen en el modelo resultante se han de realizar de la misma manera en cada uno de todos los objetos que lo componen, y una modificación local sobre uno de los objetos implica el cálculo nuevamente del objeto entero.

2.3.5 Generación por barrido

La generación por barrido se considera la técnica derivada más potente a la hora de modelar objetos tridimensionales. La idea básica del barrido consiste en definir una forma plana y barrerla a lo largo de un recorrido. El objeto tridimensional resultante depende encarecidamente de la forma inicial y la complejidad de la línea de recorrido. Las formas de generación por barrido más famosas son la extrusión, la revolución y los barridos de recorrido libres.

2.3.5.1 Extrusión

En lenguaje de diseño industrial la extrusión es el proceso de barrido de un material (tanto de plástico como de metal) a la fuerza, con una energía suficiente hasta romperlo. El proceso de la extrusión industrial está basado en la ruptura por los límites del material.

En el campo del modelado tridimensional, consiste en la creación de nuevas formas partiendo de una línea bidimensional y extendiéndola a través del espacio.

La mayoría de los programas de modelado tridimensional ofrecen las herramientas para realizar extrusiones simples. La extrusión simple siempre se realiza en uno de los ejes.

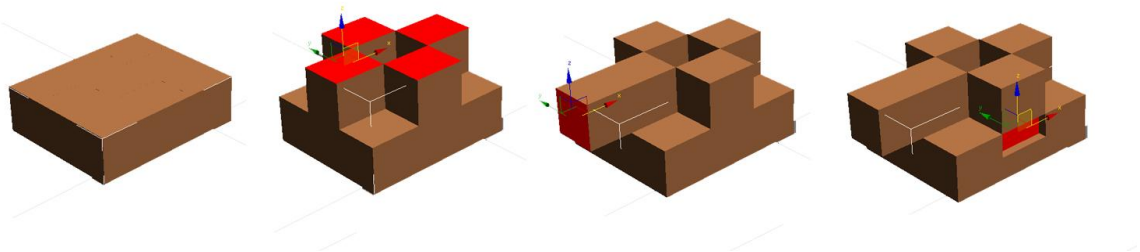

Ilustración 6 Modificación de un sólido mediante extrusión.

2.3.5.2 Barrido libre

Una extrusión que tiene lugar a lo largo de varios de los ejes cartesianos se llama extrusión compleja o barrido libre. El resultado de la extrusión a lo largo de un recorrido es similar a las mallas que son obtenidas mediante las técnicas de modelado matemático explicadas con anterioridad. El modelado por barrido es una de las formas de generación de objetos más utilizada durante siglos en columnas y ornamentos.

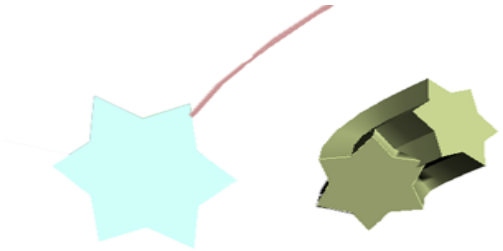

Ilustración 7 Generación de un sólido mediante barrido libre

2.3.5.3 Sólidos de revolución

Una variación muy popular de la construcción de objetos por barrido son los sólidos de revolución. Las herramientas de creación de estos objetos están basadas en una base rotativa, en la cual se coloca un cilindro de algún material moldeable y se hace girar para darle uniformemente un perfil. Esta técnica ha sido extrapolada al software de generación de objetos tridimensionales, siguiendo las mismas pautas. En generación de gráficos por ordenador, se define una forma libre plana y se realiza una rotación, normalmente de 360 grados, sobre sí misma, generando un sólido tridimensional.

La cantidad de secciones queda definida a la hora de realizar la rotación. Los cuerpos generados no tienen porqué ser completamente sólidos, y pueden presentar agujeros en el sentido de la revolución pero, en caso de que se desee que aparezca en una zona concreta, se deberá combinar con otra técnica.

Esta técnica, además, es la técnica utilizada para crear algunas primitivas geométricas como el cilindro o el cono pudiendo, si se desea, utilizar esta misma técnica para generar las primitivas tridimensionales con un cierto grado de personalización.

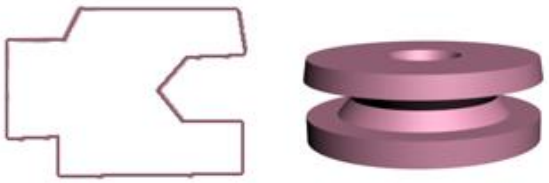

Ilustración 8 Generación de un sólido mediante revolución de una forma

2.4 Modelado mediante curvas parametrizables

También llamados cuadriláteros curvados. Las mallas resultantes son superficies curvadas de puntos. Cada conjunto de estas superficies está definido por una fórmula matemática que establece el espacio tridimensional, con la que se genera cada uno de los puntos del cuadrilátero. Dado que la superficie está definida mediante una serie de fórmulas o vectores explícitos, se pueden realizar modificaciones de la forma o curvatura de la pieza mediante la modificación de algún parámetro de la misma. El tipo de curvas más común en el diseño de gráficos tridimensionales son las curvas de Bézier, los B-Splines y las NURBS.

La potencia de este tipo de representación es muy alta, pero también los problemas de utilizarla. Cuando se modifica uno de los parámetros que definen parte del objeto, se debe calcular nuevamente la forma generada en su totalidad o, en el mejor de los casos, de los vértices adyacentes.

Este método es una ayuda para el modelado, pero en el renderizado la GPU siempre va a utilizar triángulos. El diseñador gráfico puede modelar formas curvas con facilidad y de forma intuitiva, pero a la hora de renderizar se aplicará un algoritmo para convertir todo en triángulos.

2.4.1 Curvas de Bézier

Se denomina curva de Bézier a un método de definición de una curva mediante polinomios. El grado del polinomio que define la curva corresponde con el número de puntos de control menos uno. Normalmente, en gráficos por computador se utilizan polinomios de grado tres. El método consiste en definir algunos puntos de control, a partir de los cuales se calculan los puntos de la curva siguiendo la siguiente fórmula:

$$(x(t), y(t)) = \sum_{i=0}^{n} B_{i,n}(t) P_i$$

Donde P_i son los puntos de control, y $B_{i,n}$ los llamados Polinomios de Bernstein. Dada una curva con dos puntos de control, tendríamos una recta; a mayor grado de la curva, mayor es su suavidad y mayor es el control en la forma de la curva.

La modificación de un punto de control afecta a toda la curva, puesto que la curva es una suma ponderada de los puntos de control. Para el desarrollo de formas complejas, es posible definir un conjunto de curvas en las que los puntos de control primeros o finales coinciden en el espacio. Puesto que un punto de control sólo pertenece a una curva, la modificación de un punto de control sólo afecta a una única curva, quedando el resto de curvas adyacentes sin modificar. En función de la definición de las curvas tenemos los siguientes tipos de continuidad entre éstas:

- **Continuidad C^0:** basta con que el último punto de control de una curva S_i sea igual al primer punto de control de la siguiente curva S_{i+1}. Genera uniones bruscas y con picos.
- **Continuidad C^1:** en este caso no sólo deben coincidir los puntos de los extremos, sino también sus derivadas laterales. Genera uniones suaves.
- **Continuidad G^1**: en este caso no es estrictamente necesario que las derivadas coincidan, sino sólo que sean proporcionales. Menos restrictivo que C^1 y, en muchos casos, suficiente.

- **Continuidad C^2:** deben coincidir también las derivadas segundas. A veces es deseable en 3D.

2.4.2 B-Splines

Se denomina curva B-Spline a la curva parametrizada por otras funciones spline. Las curvas de Bézier son un tipo de B-Splines, en realidad una curva B-Spline es simplemente una generalización de una curva de Bézier. La curva B-Spline se define de la siguiente manera:

$$(x(t), y(t)) = \sum_{i=0}^{n} N_i^k(t) P_i$$

Donde P_i son los puntos de control y $N_i^k(t)$ las funciones de base o funciones de Bernstein, las cuales definen la curva de la sección entre dos puntos de control. De igual modo que en las curvas de Bézier, el grado (k) de las funciones de base define la precisión y suavidad de la curva.

Existen dos características muy importantes en este tipo de curvas:

- Control local: como cada $N_i^k(t)$ tiene soporte local, al modificar un vértice de control tan solo se modifica una porción de la curva.
- La curva es una combinación convexa de sus vértices de control, así es que respeta transformaciones afines.

A consecuencia de estas dos propiedades, las funciones de base son todas positivas y suman 1 en el dominio de la función P (t). Este dominio es invariante a transformaciones afines, por ser una combinación baricéntrica de sus vértices de control.

Una de sus ventajas prácticas en diseño tridimensional sobre las curvas de Bézier es que los puntos de control no se alejan tanto de la forma curva que se desea modelar, por lo que es más sencillo obtener el modelo deseado con menos puntos de control.

En comparación con las curvas de Bézier:

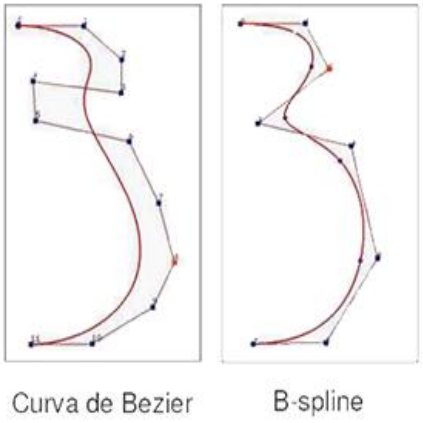

Ilustración 9 Comparación entre curva de Bézier y B-spline

2.4.3 NURBS

Las *Non Uniform Rational B-Splines* o NURBS corresponden con un caso más general de las curvas anteriores. Los B-Splines y las curvas de Bézier son casos particulares de NURBS. No son simples polinomios, sino cocientes de polinomios. Este tipo de curva surge del problema de que con curvas polinómicas no se pueden cubrir todo tipo de curvas, como sucede por ejemplo con una circunferencia. Su función de generación es:

$$B(t) = \sum_{i=0}^{n} \frac{w_i N_i^k(t)}{\sum_{i=0}^{n} w_i N_i^k(t)} b_i$$

Donde $N_i^k(t)$ son las funciones de base y w_i los coeficientes de pesos. Si aumentamos el valor de un peso w_i la función de base $R_i^k(t)$ aumenta, lo que implica que la NURBS "se curva" hacia el punto b_i. Si disminuimos el valor del peso, la curva "se aleja"

del punto b_i. Si utilizamos un peso $w_i=0$, la función de base $N_i^k(t)=0$ y el punto b_i no interviene en la fórmula de la NURBS.

2.5 Técnicas de subdivisión espacial

La técnica de subdivisión espacial es un método que considera absolutamente todo el espacio virtual del objeto y asigna para cada punto de éste una etiqueta de acuerdo con su ocupación en el espacio. La forma más común de organizar esta estructura de datos es situarlos en árboles octogenarios recursivos, llamados *octrees*, identificando el nivel de precisión del vóxel en función de la profundidad de éste en el árbol.

Un vóxel es la representación tridimensional análoga a los píxeles de una representación bidimensional. Definir cada uno de los cubos del espacio es una tarea muy tediosa aunque existen aplicaciones para realizarlo. Un claro ejemplo es el renderizado por traza de rayos donde un eficiente algoritmo se encarga de calcular el color de cada uno de los vóxeles.

El consumo de memoria de este proceso es elevado puesto que, por cada vóxel, se almacenan los diferentes objetos de la escena que se encuentran en su interior pero, a su vez, es muy utilizado porque simplifica el procesamiento computacional al aplicar diferentes transformaciones y cálculos sobre la escena completa.

Este método consiste en dividir el volumen total del escenario en cubos elementales o vóxeles y definir cada uno de estos vóxeles como contenedores de parte del objeto o vacío. La forma de definir estos vóxeles es lo que marca la diferencia entre las técnicas de subdivisión espacial.

Atendiendo a las características de los vóxeles, las técnicas de subdivisión espacial se clasifican en dos grandes grupos: técnicas de subdivisión espacial adaptativa, donde el espacio se divide en vóxeles de diferentes tamaños, y técnicas de subdivisión espacial uniformes, donde todos los vóxeles tienen el mismo tamaño.

Como se ha comentado anteriormente, esta técnica es muy utilizada en los algoritmos de traza de rayos. En estos casos, se analizan las intersecciones del rayo emitido desde el centro de la cámara con los diferentes vóxeles en los que se divide el espacio. Si el vóxel se encuentra afectado por el rayo, se realiza el cálculo para obtener los efectos lumínicos sobre los objetos que se encuentran dentro del primer vóxel que coincida en la trayectoria del rayo y los vóxeles adyacentes, no teniendo que continuar con el resto de la escena.

2.6 Representaciones implícitas

En algunas ocasiones, para la representación de objetos, se utilizan funciones matemáticas implícitas. Por ejemplo, la función:

$$(x-a)^2+(y-b)^2+(z-c)^2=r^2$$

Define de forma implícita una esfera con centro en el punto *(a, b, c)* y radio *r*. El uso de este tipo de representación es frecuente en algoritmos de detección de colisiones y algoritmos de traza de rayos sobre esferas y objetos simples.

Para algunos objetos y superficies podemos definir una fórmula que describe su volumen, como ocurre por ejemplo en esferas. Las funciones implícitas son superficies formadas por funciones matemáticas que asocian un punto con su vecino.

La utilización de esta representación en animaciones conlleva algunos problemas que radican en la dificultad para resolver movimientos, desplazamientos y deformaciones modificando cada uno de los generadores y su influencia en el sólido.

2.7 Técnicas de modelado: Texturizado, mapeado e iluminación

Uno de los objetivos de todo modelado, simulación y desarrollo tridimensional es obtener un nivel de realismo lo más alto posible para conseguir la sensación de inmersión en el propio sistema.

Se han explicado con anterioridad todas las cuestiones morfológicas que se deben plantear en el diseño del sistema, detallando el nivel de realismo de los objetos de la escena con respecto a objetos del mundo real.

Otro aspecto muy importante, y que no debe ser infravalorado, es la utilización de texturas, mapas e iluminación sobre el modelo. Por ejemplo, una esfera tan solo nos proporciona la información de un objeto esférico pero, aplicando sobre ella una textura, como podría ser una representación de la superficie terrestre, ese objeto pasa de ser una simple esfera a un planeta. Si, además, definimos una iluminación y reflejos adecuados, podemos obtener el efecto de tener un planeta con atmósfera o no.

2.7.1 Fuentes de luz e iluminación

Se entiende por modelo de iluminación el cálculo de la intensidad de color de cada punto de la escena. En este cálculo de la intensidad de un punto intervienen factores como el tipo e intensidad de la fuente de luz, el material del objeto y la orientación del objeto con respecto a la luz.

Este campo está estrechamente ligado con el campo de la fotografía o el cine, por lo tanto, la correcta disposición y elección de las fuentes de luz dentro de la escena pueden ser críticas a la hora de obtención de resultados. Al igual que sucede en los otros dos artes, la habilidad para controlar la apertura de la lente y la incisión de la luz en el objetivo de cámara es fundamental.

Se pueden definir seis grandes tipos de fuente de luz atendiendo a la tipología del emisor:

1. Punto de luz: Un punto de luz es una luz que fluye en todas direcciones, de ahí a que también sea denominada como luz omnidireccional. Este es el tipo más simple de fuente de luz y puede ser situado en cualquier punto de la escena.

2. Luz de cono: Esta simulación de luz define un tipo de luz que genera un cono de iluminación desde la fuente hasta el plano que se establezca como tope, por lo tanto define un factor de incidencia según la distancia del objeto a ella.
3. Luz direccional: Esta tipología de luz simula la incidencia de rayos desde una fuente de luz situada en el infinito, y de forma paralela al objeto. Se puede ligar al tipo de luz que ofrece por ejemplo una estrella en el firmamento o el propio sol. Esta luz se puede emplazar en cualquier lugar de la escena.
4. Luz de área: La luz de área se puede entender como una luz que se aplica en un área en concreto y que proviene desde un punto de grandes dimensiones. Utilizando un analogismo fotográfico, sería la luz que se visualiza al utilizar una caja de luz.
5. Luz lineal: Las luces lineales tiene longitud pero no anchura. Es la luz típica que ofrece un fluorescente en la vida real. Este tipo de luz debe ser utilizado con sumo cuidado puesto que su coste computacional es muy elevado.
6. Luz de ambiente: La luz radiada desde esta fuente es distribuida por toda la escena completa. Es la luz que, por defecto, tenemos en cualquier escena, determinando el nivel de iluminación y sombreado de ésta.

2.7.2 Sombreado

La apariencia visual del entorno también está definida por el sombreado de las superficies que lo componen. El sombreado es calculado para cada una de estas superficies a través del su vector normal situado en las esquinas, vértices o polígonos mediante diversos algoritmos. Cada técnica de sombreado está basada en diferentes representaciones de la luz y la superficie. La mayoría de las técnicas de sombreado calculan la sombra local, de caras, el suavizado y la sombra o brillo especular de en la superficie. Atendiendo a su tipología tenemos los siguientes tipos de sombreados:

2.7.2.1 Sombreado de caras (Flat shading)

El sombreado de caras asigna un único y constante valor a cada polígono visible de la superficie según el ángulo que forman éste y la fuente de luz. En la mayoría de las ocasiones, se contabilizan los valores en las esquinas o vértices del polígono y se calcula una media de este valor. El sombreado por cara también es llamado en algunas ocasiones sombreado poligonal o valor de sombreado de caras constante.

La mayoría de las técnicas de sombreado de caras sólo tienen en cuenta parámetros de la luz ambiental, pero algunos también incluyen el sombreado de la luz difusa. Este tipo de sombreado es el más simple de todos y, a su vez, el más rápido de calcular puesto que sólo se establece un valor por cada polígono.

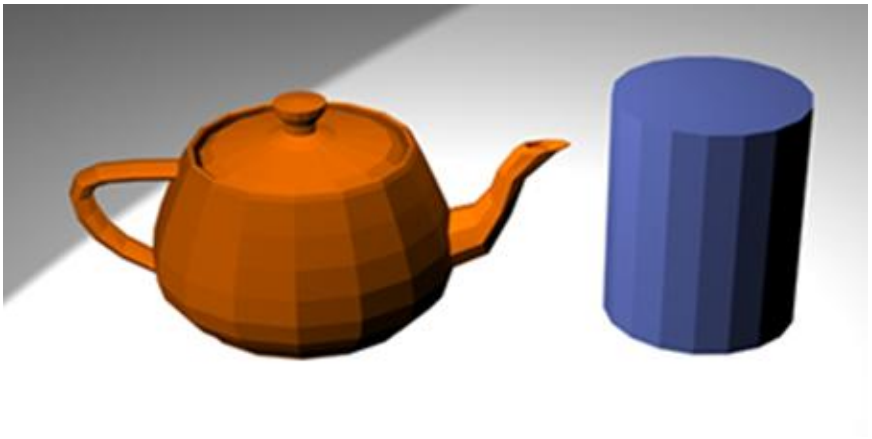

Ilustración 10 Efecto de sombreado de caras

2.7.2.2 Sombreado suavizado (Smooth shading)

El sombreado suavizado es una evolución del sombreado anterior. En este tipo de sombreado se obtienen valores continuos para toda la superficie del objeto. La idea básica de esta técnica es promediar las normales superficiales de los polígonos con sus adyacentes, creando una transición suavizada entre ambos. Normalmente, se calculan en primera instancia los valores de sombreado de todos los vectores normales de las caras del objeto y se interpolan los valores entre ambos para el resto de la superficie. Por esta razón, el

sombreado suavizado genera la apariencia de suavidad en las superficies de los polígonos tridimensionales que tienen poco nivel de detalle.

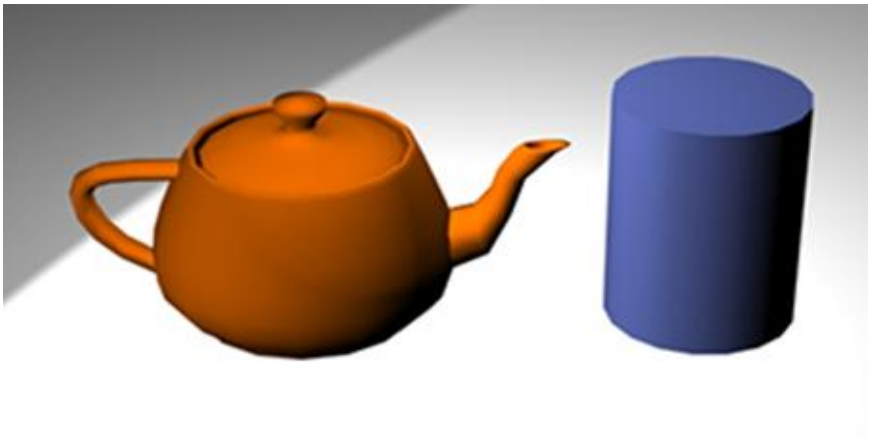

Ilustración 11 Efecto de sombreado suavizado

Algunos programas y algoritmos permiten definir el rango de interpolación entre estos valores. Tan solo cuando un ángulo entre dos normales es mayor al límite establecido, el sombreado genera un nuevo valor y suaviza la superficie. Un tipo de sombreado suavizado muy utilizado es el sombreado de Gouraud.

El sombreado suavizado contempla las propiedades de la luz de ambiente y difusa del objeto y provoca que los objetos con cierta complejidad superficial luzcan realmente bien.

2.7.2.3 Sombreado especular

El sombreado especular genera superficies brillantes que se encuentran en superficies reflectivas, es decir, genera el reflejo de la luz sobre las superficies. Al igual que el método anterior, genera superficies suaves y continuas a través de la interpolación de normales en cada uno de los puntos de la superficie. El proceso se diferencia del sombreado suavizado, ya que el especular calcula el color en cada uno de los vértices del polígono y sus reflejos, calculados a lo largo de varios puntos de la superficie de la malla.

La mayoría de los sombreados de este tipo están basados en el modelo de sombreado de Phong y sus variaciones. Este método de sombreado realiza una interpolación de normales en vez de realizarla sobre intensidades. En cada vértice del polígono se calcula la normal como la media de las normales de los polígonos adyacentes, y la normal de los puntos intermedios se calcula por interpolación lineal. Los atributos que son tenidos en cuenta por este proceso son la luz ambiente, difusa y la luz especular, ofreciendo resultados de sombreados realmente buenos, pero su coste computacional también es bastante más elevado que los métodos anteriores.

Ilustración 12 Efecto de sombreado especular

2.7.3 Texturizado

El texturizado es la forma más habitual de aumentar el detalle de un objeto tridimensional a través de una imagen. Se entiende por textura a la *piel* que envuelve el objeto y nos ofrece características tan dispares como la rugosidad o color. La textura es una imagen (normalmente de dimensiones cuadradas y potencia de 2) que se aplica a través de una función matemática sobre un objeto. La función matemática define la correspondencia entre cada uno de los píxeles o téxeles de la textura y uno o varios puntos del objeto, y se define en el sentido textura-objeto, puesto que un punto del objeto sólo puede almacenar la

información de un único punto de la textura (pudiéndose aplicar varias texturas sobre un objeto) y el mismo punto de la textura puede aplicarse en varios puntos del objeto.

La idea básica consiste en coger una imagen bidimensional y mapearla en la superficie del objeto. Existen muchas técnicas de mapeado, como la proyección (*projecting*) o la envoltura (*wrapping*), cada una con unos resultados diferentes, pero el valor real de esta técnica reside no sólo en simular la textura de la superficie del objeto, sino, como se ha comentado con anterioridad, en la generación de otros atributos como la reflectividad o la rugosidad. Las técnicas de texturizado también son utilizadas como técnicas de simulación de características superficiales, como la textura, color o transparencia, todas ellas desde una única imagen.

Para el proceso de texturizado se pueden emplear la combinación de diferentes imágenes, cada una de ellas definiendo una característica de la superficie. Para la obtención de estas imágenes se pueden utilizar tanto imágenes pintadas, fotografías o incluso patrones sintéticos. No hay tamaño máximo que defina las dimensiones de una textura, cada proyecto tiene sus propias limitaciones. Normalmente las texturas utilizadas para los sistemas de tiempo real son diseñadas como paneles de 256x256 píxeles. En muchas ocasiones, las imágenes son repetidas a lo largo de una superficie para ahorrar coste computacional, lo que se denomina *tiles.*

A continuación se muestra una textura utilizada en el proyecto para representar la consola central del vehículo sobre una malla de bajo nivel de detalle, logrando simplificar el coste computacional del modelo.

Ilustración 13 Detalle de simulación de la consola interior mediante texturas

Existen muchas maneras de proyectar a su vez una textura sobre un objeto o superficie. Algunos métodos de proyección son simples, mientras que otros son más complejos; algunos obtienen efectos muy realistas y otros simplemente dan una apariencia válida. No existe una teoría ni pautas concretas para escoger un tipo de proyección a la hora de aplicar las texturas, pero podemos diferenciar cuatro tipos diferentes:

1. La proyección plana aplica la imagen sobre superficies planas directamente, puesto que sus resultados son totalmente predecibles y la distorsión de la imagen es mínima sobre el objeto. Normalmente se aplica en un plano XX, XZ o bien XY con idénticos resultados en el objeto. Este método también puede ser utilizado en superficies curvas proyectando la imagen de forma paralela a las caras, aunque en estas ocasiones es fácil tener errores de cortes visibles o huecos en la superficie.

2. La proyección cúbica es una pequeña variación de la proyección plana, tan solo que repite la imagen por cada uno de los seis posibles lados del cubo que engloba el objeto 3D. Es muy útil para aplicar texturas sobre objetos cúbicos o paralelepípedos, obteniendo resultados muy dispares en superficies curvas o irregulares.

3. La proyección cilíndrica aplica la imagen de la superficie envolviendo el objeto por el mapa y aplicando por la cara superior e inferior una proyección circular de ésta. Esta técnica es muy útil para el mapeo de texturas alrededor de objetos cilíndricos

como una botella o una pila. Las proyecciones cilíndricas están diseñadas para cubrir toda la superficie del objeto. A través de un parámetro permiten controlar el ángulo del mapeo y posición de las capas utilizadas como tapas del cilindro.

4. La proyección esférica aplica un mapa rectangular alrededor de la superficie hasta que se encuentra nuevamente el comienzo de la textura. Esta técnica es muy útil para aplicarse sobre objetos redondos como puede ser una pelota. Al igual que el método anterior, se puede definir en ella un parámetro indicando el ángulo del mapeo de la textura.
5. La proyección *unwrapped* (extendida) permite definir sobre una superficie plana toda la morfología del objeto tridimensional. Un ejemplo típico es el cubo de papel para montar, obteniendo una figura en forma de T plana con sus 6 caras. Sobre esta forma se aplica la proyección de la textura, envolviendo todo el objeto. Es una forma de proyección muy utilizada, sobre todo en objetos tridimensionales en los que se desea aplicar texturas en forma de pegatinas. Este ha sido el método más utilizado para la aplicación de texturas en este proyecto.

Otro aspecto que se ha de tener en cuenta a la hora de texturizar un objeto 3D es el posicionamiento inicial de la textura. Toda textura tiene forma rectangular o cuadrada, por lo tanto, y de forma normalizada, se define en esa estructura cada punto de la superficie con un par de coordenadas que van desde 0 hasta 1. El punto superior izquierdo de la textura se corresponde con la coordenada (0,0), el punto inferior de la izquierda con la coordenada (0,1), el punto superior derecho con la coordenada (1,0) y el punto inferior derecho con la coordenada (1,1), respectivamente.

Por defecto, la textura se comienza a aplicar en el origen de la superficie, el cual se puede establecer en cualquier punto de la superficie. Según la aplicación que se utilice para realizar el modelo, estos orígenes se establecen en unas posiciones u otras. En todo caso, se pueden definir una coordenada XY y un desplazamiento sobre ésta. En objetos simples no entraña excesiva dificultad establecer las posiciones de aplicación de la textura, pero para

estructuras tridimensionales complejas se requiere un proceso de ajuste fino que puede ser muy costoso. En estructuras poligonales curvas, se utiliza el concepto de coordenadas UV para tal propósito. El espacio curvo está definido por un valor horizontal (U) dentro del rango (0,1) y un valor vertical (V) de la misma manera. El parámetro U se sitúa normalmente en la posición más a la izquierda del volumen contenedor del objeto y el parámetro V en el superior. En superficies cuadráticas, sin embargo, el parámetro U se asocia con la longitud y el parámetro V con la latitud de la curva que define la superficie.

A su vez, existen otra serie de parámetros para posicionar correctamente la textura sobre el sólido, estos parámetros son el escalado (*scaling)* y la repetición (*tiling*). El escalado es utilizado para cubrir mayor o menor parte de la superficie, con una misma textura. Un objeto con imagen muy pequeña podría cubrir una superficie muy amplia, pero podría provocar imprecisión a la hora de visualizar la textura. Por otro lado, el *tiling* o repetición, establece los patrones de repetición, tanto en sentido horizontal como vertical.

2.7.4 Reflexión de la luz

La luz reflejada por los materiales también nos ofrece información visual del tipo de material que es, aparte de ser una de las principales fuentes de realismo en imágenes virtuales. Los tres tipos básicos de luces de reflexión de una superficie son la luz de ambiente, difusa y especular.

Un objeto tridimensional puede adquirir un gran realismo, por ejemplo, reflejando el resto de objetos del modelado sobre su superficie. Aparte, otorga a las superficies una semántica visual del tipo de material simulado a través de sus reflejos, teniendo tanto superficies metálicas, como curvas o plásticas.

Otra técnica muy utilizada para simular la reflexión de un objeto es la utilización de texturas. Con ellas, se puede simular tanto el reflejo de otros objetos (*Enviroment Maps)* o reflejos de luz (*Reflection Maps).*

2.8 Otras técnicas de simulación de materiales y superficies

Un problema muy común a la hora de dotar a los objetos tridimensionales de propiedades físicas de material es el denominado *aliasing*. Este proceso consiste en una visualización de los bordes del objeto de forma escalonada. Es un error que proviene de la dificultad de calcular con exactitud qué propiedad visual se ha de aplicar en estas zonas. Si el téxel no es el adecuado en un punto, se podrá visualizar un salto en la superficie del objeto, perdiendo la continuidad de éste en sus fronteras. Para solventar esos problemas se pueden utilizar los métodos de *supersamplig*, los cuales, para definir el color de un píxel, utilizan la media ponderada de éste con los 4 puntos de su alrededor, o bien el método de *prefiltering*, donde se realiza la ponderación sobre la textura que se aplicará en el punto.

Otra técnica para evitar problemas y errores de visualización es la utilización de los filtros *antialasing*. Este filtro realiza un suavizado de la imagen generada de baja resolución para obtener una imagen "suavizada" de alta resolución.

Por otro lado, también se pueden utilizar las técnicas de *mipmaps* o mapas piramidales. Esta técnica ajusta la textura que se aplica sobre el objeto en función del tamaño de visualización, escogiendo texturas de mayor resolución y detalle en los objetos más visibles.

2.9 Animación

Las primeras secuencias de animación fueron creadas a finales del siglo XIX, pero no fueron fuertemente impulsadas hasta entrado el siglo XX. Una animación consiste en una secuencia de imágenes estáticas que generan la sensación de una secuencia continua.

A lo largo de todo este siglo de antigüedad, se han desarrollado múltiples técnicas para generar dichas secuencias, desde el dibujado a mano de cada una de las imágenes, a la toma de imágenes estáticas de miniaturas (*stop-motion*), técnicas de captura de

movimientos reales (*animatronics*), animación de caracteres mediante robots o la animación 3D o por computador.

Una animación 3D es la animación, creada por computador, en la cual se simula una escena dentro de un entorno tridimensional. Esta animación puede generarse tanto en tiempo real, en la que las imágenes se generan mientras se visualizan o bien una animación prerenderizada, es decir, se trata de una secuencia generada previamente. Por lo tanto, la principal diferencia entre ambas radica en el momento de renderizado de la animación, es decir, el proceso de convertir las fórmulas matemáticas en imágenes digitales. En la primera, la animación se renderiza en tiempo real, consumiendo gran cantidad de memoria y procesamiento. Para que este tipo de animación sea posible, en la mayoría de las ocasiones es necesaria la utilización de una tarjeta especializada en este tipo de cálculos matemáticos y liberar de carga el procesador gráfico. Este tipo de animación se utiliza especialmente en juegos en 3D, donde el dinamismo, la velocidad y la necesidad de diferentes ángulos y movimientos, es fundamental. En la segunda, la animación del modelo 3D se realiza previamente. El trabajo pesado se realiza una sola vez y luego se puede ejecutar como vídeo, lo cual no consume mucho procesamiento. Este tipo de animaciones son las que se visualizan, por ejemplo, en cine.

2.10 La Cueva de Realidad Virtual

La Universidad Rey Juan Carlos está dotada de una sala de simulación tridimensional inmersiva denominada Cueva de Realidad Virtual. Esta sala se encuentra situada en el Centro de Apoyo Tecnológico y permite al usuario adentrarse en un sistema de realidad aumentada donde puede moverse y desplazarse por la escena que esté siendo simulada, enriqueciendo la experiencia de la simulación y permitiendo "navegar" a través de la escena o mundo que se desee.

La Cueva de Realidad Virtual de la Universidad Rey Juan Carlos es una de las cuevas de realidad virtual más grandes del sur de Europa con unas dimensiones de 3,20 m x 3,20 m

de superficie de planta y tres metros de altura, lo que permite alojar en su interior a doce personas de forma simultánea.

La Cueva está formada por una habitación en cuyas paredes son proyectadas consecuentemente las imágenes que forman el entorno, de forma que el usuario se sitúa en el interior de la escena literalmente. La habitación es totalmente reconfigurable, permitiendo ajustar las pantallas a los requisitos de la simulación, como forma de anfiteatro, cueva, pantalla única, etc. También posee diferentes dispositivos que permiten recoger información de las acciones del usuario en todo momento, como su posición o rotación. El *feedback* es conseguido mediante cascos o gafas estereoscópicas, guantes y otros periféricos diseñados para tal fin.

Además de ofrecer interacción física mediante los guantes y sensación inmersiva en el entorno, es posible utilizar el sistema de sonido de la Cueva, el cual se puede configurar de forma totalmente direccional, siempre teniendo en cuenta la posición del sujeto que se encuentre en su interior.

CAPÍTULO 3 - Descripción informática

Para la realización del proyecto, ha sido necesario, aparte del estudio de las técnicas de modelado, animación tridimensional y los distintos sistemas existentes de aprendizaje interactivo guiado, la utilización de diferentes herramientas y programas informáticos especializados. En esta sección, se explicará con detalle la utilización y el desarrollo del proyecto mediante estas herramientas.

3.1 Aplicaciones utilizadas y entornos de desarrollo

En la ejecución del proyecto podemos diferenciar cuatro etapas distintas: una primera fase de modelado, una segunda de texturizado, una tercera de animación y la última de desarrollo de la aplicación para la Cueva de Realidad Virtual. Las diferentes fases se han desarrollado gracias a diversas aplicaciones informáticas, cada una con un propósito diferente. A continuación se explicará cada una de las aplicaciones utilizadas.

El modelado tridimensional puro se ha realizado a través de la aplicación 3D Studio Max 8. Este software es una de las herramientas más famosas y utilizadas para los trabajos de modelado y animación tridimensional. Pertenece a la empresa Autodesk y dispone gran cantidad de extensiones y documentación externa. La aplicación por sí sola dispone de las herramientas necesarias para efectuar casi todo el flujo de trabajo, desde el modelado poligonal a la animación y renderizado. También dispone de herramientas para el desarrollo de scripts y funcionalidad, pero ese proceso se ha decidido realizar mediante la aplicación EON Studio 5.5.

El uso de 3D Studio Max está principalmente orientado al modelado en videojuegos, aunque también es utilizado profesionalmente para el desarrollo de anuncios, televisión, infografía o efectos especiales. Se ha utilizado tanto la versión 8 como la versión 2011, la primera de ellas para modelar y la segunda para realizar la animación.

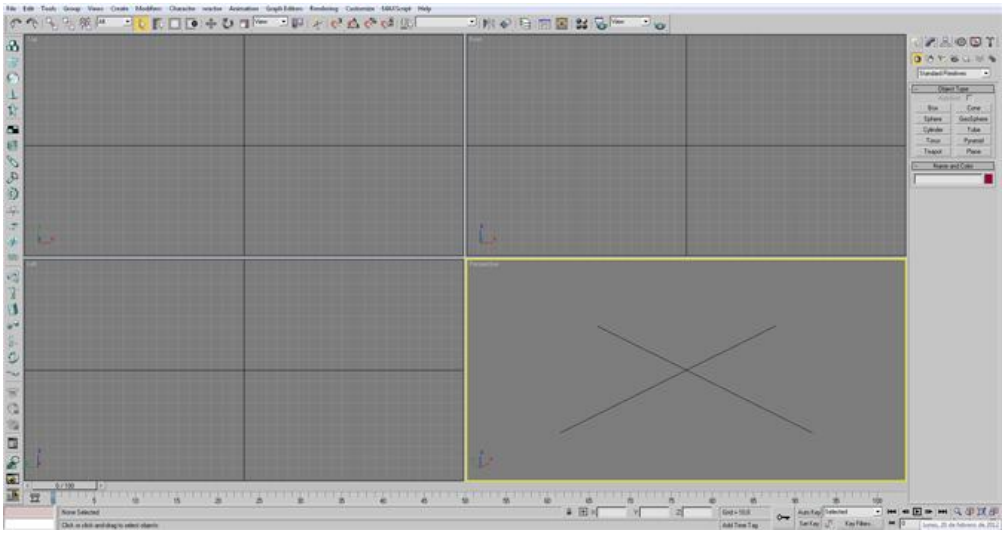

Ilustración 14 Visualización pantalla principal de 3dsMax 8

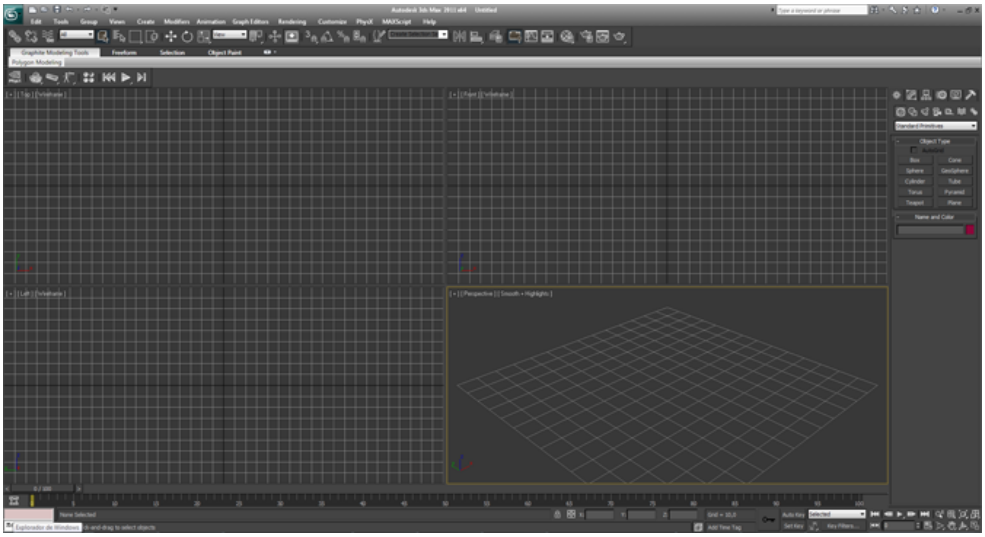

Ilustración 15 Visualización pantalla principal de 3dsMax 2011

Por otro lado, para poder realizar el texturizado y mapeado de texturas del modelo tridimensional, ha sido necesaria la utilización de una aplicación de retoque fotográfico, concretamente Gimp. Esta herramienta forma parte del proyecto GNU y está disponible bajo licencia GNU y comenzó su desarrollo como proyecto universitario en el año 1997

como una herramienta simple de edición fotográfica para Linux. Desde entonces, y con el apoyo incondicional de la comunidad, ha evolucionado hasta expandirse por los principales sistemas operativos y ha ido añadiendo nuevas funcionalidades cada vez más complejas, llegando a ser una alternativa a la herramienta Adobe Photoshop para usuarios avanzados no profesionales.

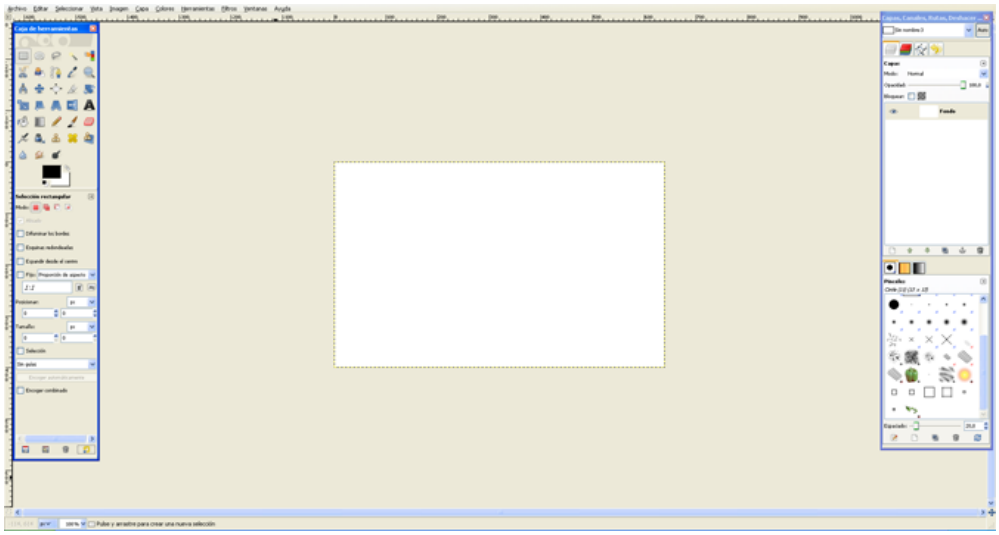

Ilustración 16 Visualización pantalla principal del editor de imágenes GIMP

Por último, y para la generación del sistema en la cueva, se ha utilizado el software EON Professional Studio junto al plugin para 3d Studio Max 8 llamado EON Raptor. Estas herramientas permiten generar exportaciones del modelado tridimensional para visualizarse tanto en navegadores web como en otras herramientas de modelado y realidad virtual.

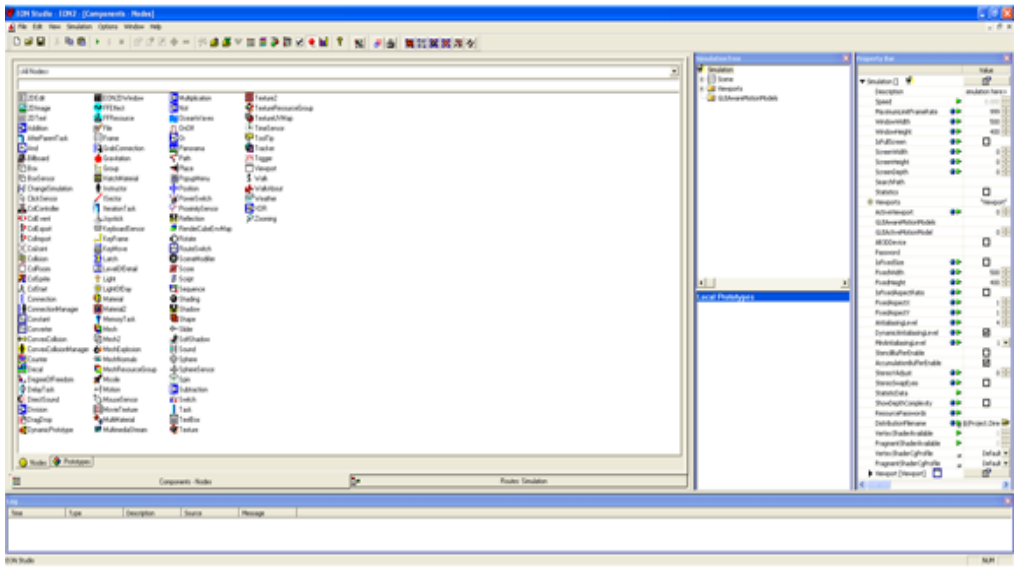

Ilustración 17 Visualización de la pantalla principal de EON Studio 5.5

3.1.1 Elección de aplicaciones y entorno de desarrollo

Una vez conocidas las aplicaciones utilizadas en el proyecto, se debe explicar el porqué de ello. Esta elección es de vital importancia pues va a marcar el flujo de trabajo del proyecto además de marcar los límites de desarrollo de éste.

La elección de la aplicación 3ds Max queda totalmente legitimada y condicionada por la exportación del modelo a la Cueva de Realidad Virtual de la Universidad Rey Juan Carlos, donde será visualizado posteriormente. La existencia de un plugin como EON Raptor para 3ds Max 8 y encontrarse totalmente integrado con la cueva exige la utilización de esta suite de modelado. Aun así, existen otras razones del porqué del uso de la herramienta. A la hora de usabilidad, para un usuario nuevo, 3ds Max ofrece al usuario una interfaz más intuitiva y sencilla que sus competidores directos, Maya o Blender, los cuales requieren a priori un esfuerzo mayor al comienzo de la curva de aprendizaje. Por otro lado, atendiendo al enfoque de estas suites de modelado, Maya y Blender se encuentran enfocadas al modelado orgánico, es decir, personajes, fluidos, animales, etc. y 3ds Max se encuentra más enfocada a un modelado arquitectónico, estructural e industrial. Sin llegar a

los extremos que podría permitir una aplicación como AutoCAD, permite la definición de piezas artificiales de múltiples formas, logrando generar mallas poligonales regulares y simétricas con mayor facilidad. Por problemas encontrados a la hora de realizar correctamente el renderizado de la animación, se ha debido de utilizar también una versión más actual de 3ds Max, concretamente la versión 2011. Aparte, esta versión ofrece significativas mejoras como son la dedicación a sistemas operativos y CPU´s de 64 bits, mejorando sustancialmente el rendimiento general (no se debe olvidar la gran cantidad de procesamiento requerido para el renderizado), así como la inclusión automática de materiales estructurales por defecto, los llamados 3ds Material, que ofrecen un gran nivel de realismo. En última instancia, la aplicación permite la utilización de la tecnología CUDA de NVIDIA. CUDA es una arquitectura de cálculo paralelo que aprovecha la gran potencia de la GPU (unidad de procesamiento gráfico) para proporcionar un incremento extraordinario del rendimiento del sistema. Esta tecnología se encuentra disponible en las tarjetas gráficas NVIDIA, permitiendo al usuario doméstico mejorar sus tiempos de renderizado sin tener que recurrir a las caras tarjetas profesionales disponibles en el mercado.

La utilización de la aplicación EON Professional radica principalmente en su compatibilidad con la Cueva de Realidad Virtual, siendo bastante complejo encontrar cualquier otra alternativa que se adapte a las necesidades y a especificaciones de dicha cueva. A favor de esta aplicación, comentar que es una herramienta sencilla aunque con algunas peculiaridades de funcionamiento, pero permite realizar, con relativa facilidad, aplicaciones 3D a partir de una malla tridimensional dada.

En último lugar, la elección de la utilización de Gimp como herramienta de tratamiento de imágenes se basa en que, sin necesitar grandes funcionalidades técnicas, es una gran aplicación de edición de imágenes, gratuita, multiplataforma y de software libre.

3.1.2 Lenguaje de programación

Para dotar al sistema de ciertas funcionalidades de cara al usuario, se ha utilizado el lenguaje de programación JScript. Este lenguaje es la versión de Microsoft del lenguaje de programación Javascript. Ambos lenguajes son muy similares, las principales diferencias se establecen en que JScript permite el manejo de objetos propios de Windows como el manejo de ActiveX y otros objetos de este sistema operativo.

EON Professional Studio permite la utilización y la interpretación de código ActionScript, enfocado mayoritariamente a aplicaciones Flash, pero se ha optado por la utilización de JScript por la amplia experiencia previa en el lenguaje JavaScript.

3.2 Recopilación de datos y estudio del proyecto

Para realizar una simulación tridimensional realista se requiere de un trabajo previo de documentación sobre lo que se desea modelar, simular o animar. A continuación, se describe el trabajo previo necesario correspondiente con esta tarea.

3.2.1 Documentación

Como se acaba de comentar, es fundamental el entendimiento y comprensión del sistema que se desea modelar. Ciñéndose al proyecto, la primera actividad necesaria previa al modelado es la obtención de diferentes imágenes desde todos los ángulos posibles del automóvil Lamborghini Gallardo. Por la exclusividad propia del modelo y de la marca (se ha de recordar que se trata de un superdeportivo), el trabajo de campo se ha centrado principalmente en la recopilación de imágenes del modelo desde todos los ángulos posibles. Debido a las limitaciones de imágenes existentes por la red, y para encontrar un mayor detalle de alguna zona confusa, se procedió a la visita a un concesionario especializado en vehículos de alta gama pero, desafortunadamente, no se encontró en los concesionarios de la zona ninguna unidad del modelo mencionado.

Otras imágenes necesarias para el correcto modelado son las imágenes guías del *viewport*, es decir, una imagen simple que muestre el vehículo desde la parte frontal, la trasera, vista lateral y vista superior. Estas imágenes se han obtenido de un libro de promoción del propio vehículo conseguido a través de internet.

Ilustración 18 Fotos del modelo real a modelar

Por último, en referencia a la carrocería del vehículo, también se decidió adquirir un modelo a escala 1:18 del mismo, aunque por limitaciones en el mercado de modelismo en ese momento se tuvo que utilizar una maqueta del modelo descapotable.

A su vez, para realizar tanto el modelado como la animación del motor se requirió la recopilación de información sobre las principales partes de un motor y las piezas correspondientes a la transmisión.

Para poder realizar correctamente la animación del motor se ha de comprender en qué consiste el ciclo de combustión interna de encendido provocado o ciclo de Otto. Estos motores basan su funcionalidad en que existen cuatro tiempos (existe una variante de dos tiempos) técnicos de funcionamiento, estos son:

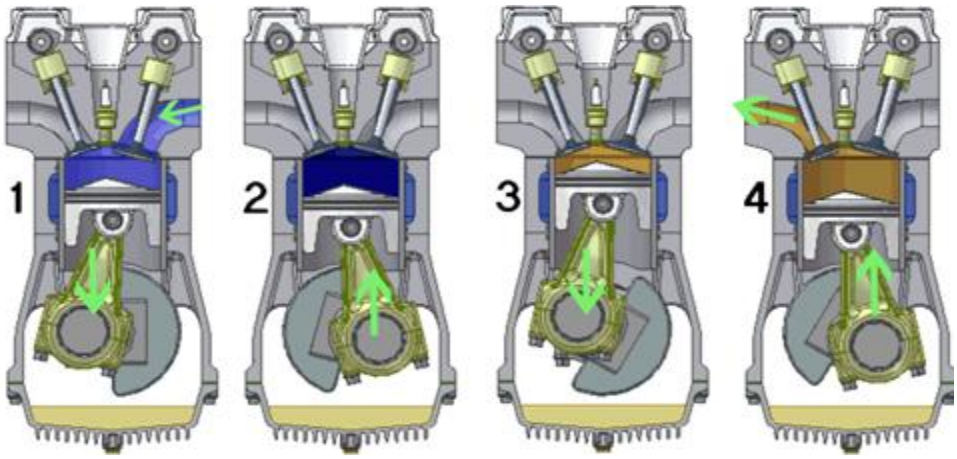

Ilustración 19 Fases del ciclo de Otto

Admisión (Fase 1): Las válvulas permiten la inclusión en el cilindro de gasolina y aire a la vez que el pistón desciende por el cilindro debido al movimiento del cigüeñal y la biela hasta llegar al punto mínimo interno.

Compresión (Fase 2): Se cierran las válvulas de admisión y se comprime el oxígeno y la gasolina dentro del cilindro debido a que el pistón vuelve al punto alto del cilindro.

Explosión (Fase 3): Cuando el pistón alcanza el punto más alto en su recorrido por el cilindro, la bujía prende la mezcla aire-gasolina que se encuentra en el cilindro a través de un chispazo, provocando una explosión que desplaza con fuerza nuevamente el pistón hasta el punto mínimo interno.

Escape (Fase 4): En esta última fase, se abren las válvulas de escape y el pistón vuelve a ascender hasta el punto más alto del cilindro comenzando nuevamente el ciclo de Otto.

El movimiento lineal del pistón por el cilindro se convierte en movimiento circular gracias a la utilización de la biela y el cigüeñal. Posteriormente este movimiento, mediante el volante de inercia, la caja de cambios y el resto de la transmisión es desplazado hasta las ruedas motrices.

3.3 Modelado

El proceso de modelado concierne a la construcción estructural del modelo poligonal. La diversidad morfológica de los objetos que pueden formar parte de un vehículo provoca que se tengan que utilizar diferentes metodologías de modelado para las distintas partes o piezas.

En primer lugar, es necesario situar en la escena las texturas utilizadas para guiarse a la hora del modelado del vehículo. Para ello, se generan cuatro planos y se aplican sobre ellos la textura correspondiente; vista superior, vista frontal, vista trasera y vista cenital.

Es fundamental, antes de comenzar a modelar, escoger concienzudamente el método de modelado para la pieza. Inicialmente, se barajó la posibilidad de modelar utilizando la técnica de modelado poligonal libre, por extrusión, métodos de revolución, geometría constructiva de sólidos o la utilización de splines y superficies parametrizables. Debido a la complejidad de las piezas en la unión de ángulos y superficies, sobre todo, se descartó la opción de definición de splines y superficies parametrizables. Resumiendo muy brevemente las técnicas de modelado utilizadas en el vehículo, la mayor parte está modelado utilizando extrusión y técnicas de barrido, mientras que otras partes, como llantas, ruedas y algunas piezas del motor, se han modelado utilizando geometría construcción de sólidos (CSG).

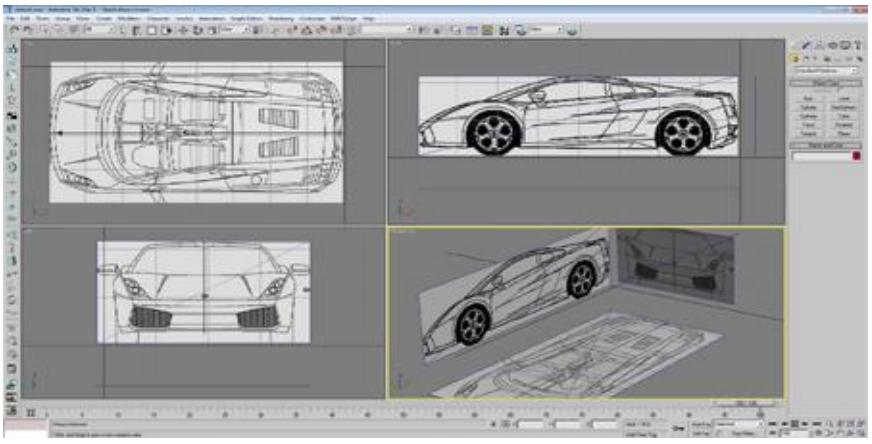

Ilustración 20 Configuración de los viewports con las imágenes de guía

El modelado del vehículo en sí parte desde una de las vistas, por ejemplo la vista lateral, y sobre esa vista se genera en 2D, y utilizando el sistema de extrusión, todo el contorno de la pieza correspondiente. Una vez ajustada la pieza en una de las vistas, utilizando el sistema de modelado libre, se ajusta cada una de las piezas a la nueva vista seleccionada, pasando de tener una estructura bidimensional, a una malla tridimensional. Una vez que se tiene una estructura tridimensional, se deben ir definiendo nuevos polígonos, bien mediante la extrusión de diferentes vértices o bien realizando cortes en los polígonos existentes.

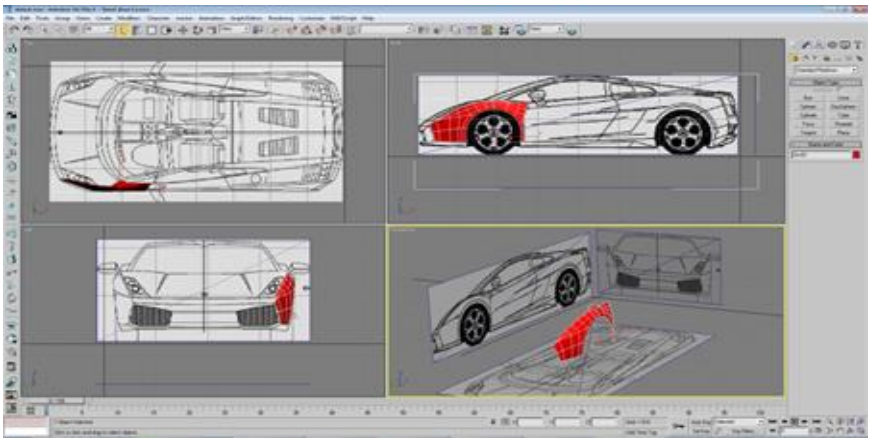

Ilustración 21 Comienzo del modelado tridimensional

Dado que el vehículo es totalmente simétrico, basta con modelar tan solo uno de los lados de éste y aplicar posteriormente un operador de simetría. Esto también se traduce en un ahorro del coste computacional y poligonal en el modelo. Todas las piezas correspondientes a la carrocería del automóvil se han modelado siguiendo este proceso.

Para la realización de las ruedas se ha utilizado una técnica totalmente diferente. El conjunto total de la rueda está formado por el neumático, la llanta, el disco de freno y la pinza.

El neumático está construido a partir de un cilindro hueco. Sobre este objeto se realizan diversas modificaciones y se suavizan los bordes. Posteriormente se le aplica un operador de suavizado, obteniendo finalmente el aspecto deseado.

Para los frenos y las pinzas se siguen un proceso de modelado libre a partir de un cuadrilátero sencillo.

La llanta, en cambio, ha requerido un proceso más laborioso. Analizando la llanta a simple vista se puede comprobar que está formada por cuatro partes iguales que, una vez vistas individualmente, no es complicado diferenciar que en realidad esa cuarta parte se puede dividir de nuevo, por lo que la llanta entera se compone por ocho secciones idénticas entre sí. Teniendo esto en cuenta, se ha generado mediante técnicas de geometría constructiva de sólidos (GCS), modelado libre y modelado por extrusión cada una de las secciones. Posteriormente se aplican varios modificadores simétricos sobre la sección hasta obtener toda la superficie de la llanta.

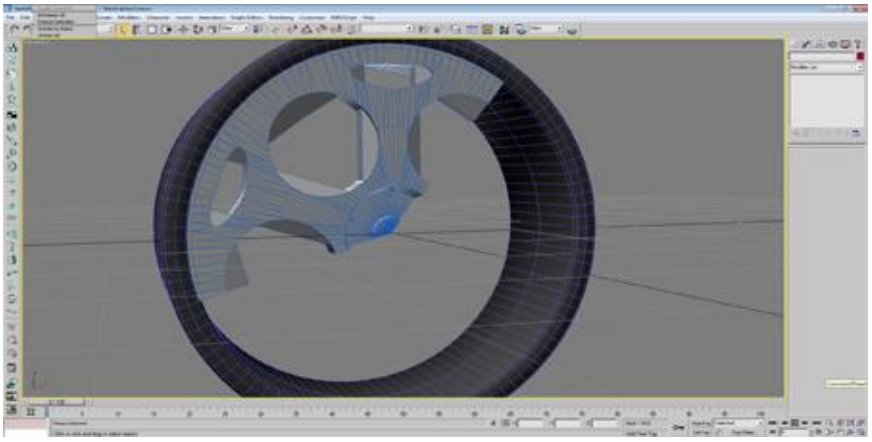

Ilustración 22 Proceso de modelado de la rueda

En última instancia, se agrupan todas las piezas en una sola mediante un operador de agrupación, teniendo la rueda completamente formada.

Otra parte crítica es el modelado del motor. El motor de un automóvil está formado por cientos de partes, tanto fijas como móviles, pero el proyecto se ha centrado en generar las partes imprescindibles para poder realizar una animación completa del funcionamiento de éste.

Puesto que hay que realizar una animación, es indispensable generar de una forma lógica cada una de las piezas que se ven comprometidas. La mayoría de las piezas están

generadas mediante la combinación de las técnicas de geometría constructiva de sólidos, modelado libre y modelado por extrusión.

Posteriormente, y una vez generadas todas las piezas necesarias, es imprescindible generar las estructuras de control de animación.

La animación del motor se ha realizado mediante cinemática inversa (*The Art of 3D Computer Animation and Effects*). Existen dos partes móviles principales en la animación, las piezas correspondientes al árbol de transmisión y el conjunto de pistón, cilindro y biela. Gracias a la utilización de la cinemática inversa, la rotación de la biela provoca que el conjunto se mueva consecuentemente dando una apariencia bastante real del sistema o del Ciclo de Otto comentado con anterioridad.

Por último, y para definir el espacio que ocupa la escena como un espacio finito, se han creado una serie de objetos a modo contenedor. La idea principal es la de ofrecer en la escena la apariencia de una situación real, en este caso, un garaje. La generación del garaje consta de seis planos ajustados entre sí, cuatro columnas generadas con sendos paralelogramos y, en un lateral, y mediante composición de objetos, unas rejillas a modo de ventana.

3.3.1 Texturizado y definición de materiales

Para dotar de realismo a la escena es necesario definir diferentes materiales y texturas sobre el modelo. El 3ds Max posee un completo editor de materiales accesible desde el menú contextual (Material Editor) o pulsando la tecla M.

Ilustración 23 Ventana de configuración de materiales en 3dsMax

Para cada uno de los materiales se puede definir su tipología (metal, plástico…), color, propiedades de la luz y reflejos, rugosidad o transparencia. También se pueden definir los materiales como texturas e incluso realizar combinaciones entre todos ellos para obtener materiales más complejos.

Hay que señalar que para la realización del modelo y para la inclusión de éste en EON se han definido uno a uno una serie de materiales metalizados del tipo de material estándar que ofrece 3ds Max, dejando a un lado los materiales propios de los motores de render debido a su incompatibilidad con EON. Para el renderizado de la animación se ha generado una librería de materiales complementaria que mejora mucho la apariencia de los objetos, estos materiales son los *MentalRay Materials* o los nuevos materiales disponibles en 3ds Max 2011, *3ds Max Material*.

Ilustración 24 Comparación visual entre un material 3ds Max Material (izquierda) y un material estándar de 3ds Max (derecha)

Para la ejecución completa de la escena se han definido un total de 58 materiales, contando los materiales texturizados.

Por otro lado, también ha sido necesaria la inclusión de texturas en el propio modelo. Dentro de la escena tridimensional se pueden observar dos formas de aplicación de texturas; la primera, aplicación de texturas de ambiente (Ilustración 25) y la otra, las texturas Unwrapped o sobre mallas desplegadas (Ilustración 26).

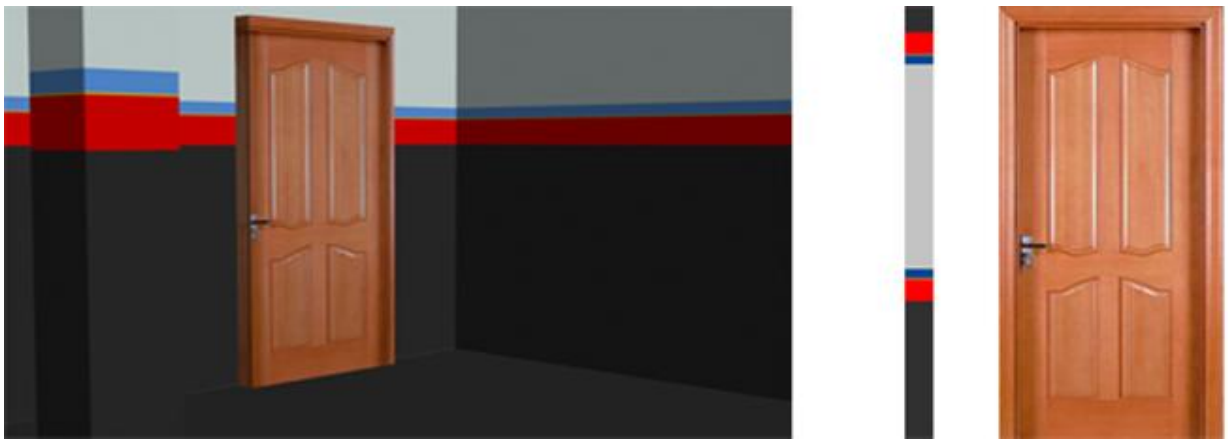

Ilustración 25 Aplicación de texturas de ambiente

También se ha utilizado en las tapas de balancines del motor los mapas de relieve para dar forma a las tapas con la inscripción de la marca tal y como sucede en el modelo real.

Ilustración 26. Aplicación de textura Bump Mapping sobre la tapa de balancines

3.4 Animación y simulación

La animación del vehículo se ha enfocado para mostrar con el mayor realismo posible la apariencia de un superdeportivo como es el Lamborghini Gallardo y poder mostrar al mismo tiempo, de una forma sencilla y elemental, el funcionamiento de un motor gasolina de cuatro tiempos.

Para producir la animación se han generado las partes más importantes y básicas de un motor. Por un lado, se pueden distinguir las partes pertenecientes al "bloque motor" y las piezas móviles del motor. Las partes del bloque motor tan solo ofrecen la parte externa del motor completo así como las partes estáticas para la animación. Como partes móviles se han realizado las piezas pertenecientes a la distribución (árbol de levas, poleas) y las piezas móviles del Ciclo de Otto (cigüeñal, bielas, pistones…). Aparte también aparece el conjunto de admisión y escape.

Ilustración 27 Vista exterior del motor

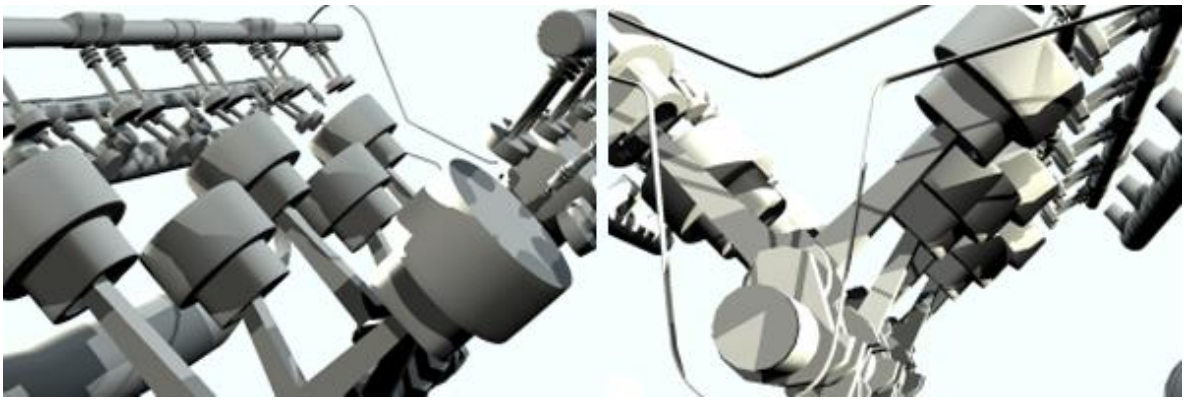

Ilustración 28 Detalle del conjunto de piezas interno del motor

Debido a que en algún momento de la animación ciertas piezas aparecen en primer plano, ha sido necesario modelar estas pequeñas piezas en tamaño con un gran detalle, provocando considerablemente el aumento del número de polígonos del conjunto.

Ilustración 29 Comparación del tamaño de las válvulas y la rueda

Para la realización de la animación de forma lógica y controlada, se han definido los cigüeñales como punto de control de sí misma. A partir de ellos, se asignan las bielas como enlaces fijos hacia los huesos del esqueleto de la animación, que se corresponde con los pistones. El movimiento del pistón se define rectilíneo en la medida de lo posible, logrando finalmente que una operación de rotación sobre el cigüeñal genere de forma automática un movimiento rectilíneo de sube-baja en los cilindros, tal y como ocurre en un motor real.

La apertura de las válvulas, así como la sincronización con los árboles de levas y las poleas de la distribución se ha tenido que realizar de forma manual, teniendo que definirse movimientos circulares en estos elementos de forma sincronizada para obtener un movimiento aparentemente real.

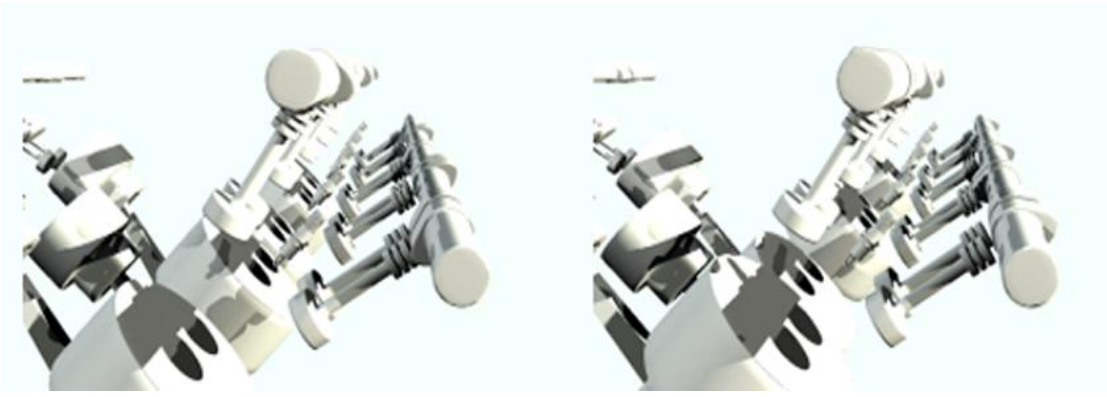

Ilustración 30 Detalle del árbol de levas y movimiento de las válvulas

3.5 Elaboración del modelo en EON Studio

Para el visionado del modelado en la Cueva de Realidad Virtual se ha tenido que realizar el trabajo de exportación y adecuación el modelo tridimensional del 3ds Max a EON Studio 5.

Este proceso se realiza en primera instancia mediante el plugin de EON para 3ds Max, el cual genera un fichero compatible con EON.

El EON Studio 5 utiliza para las escenas un árbol de nodos de diversas tipologías. Dentro de este árbol se localizan los diferentes objetos que forman la escena, cada uno de los materiales definidos, nodos operacionales básicos y los nodos de control. A la hora de abrir el fichero resultante de la exportación de 3ds Max, aparece un árbol con todos los elementos previos de la escena, mallas poligonales, cámaras, etc. Tras la carga inicial, el árbol no es del todo correcto puesto que la importación del fichero genera elementos duplicados en el árbol. Tras eliminar los elementos duplicados, es recomendable agrupar los nodos por tipologías o secciones, lo que facilita enormemente el trabajo.

Por otro lado, para cumplir los requisitos funcionales, se ha tenido que implementar una serie de algoritmos. Estos algoritmos se encuentran implementados mediante JScript. EON ofrece al desarrollador unas librerías de acceso a los diferentes objetos que forman la escena, permitiendo dotarlos de diversas funcionalidades.

3.6 Proceso de diseño

En esta sección se tratará y explicará los diferentes módulos técnicos que han sido necesarios desarrollar para la visualización y utilización de la aplicación en tiempo real.

3.6.1 Diseño de la aplicación

La aplicación se ha diseñado desde el punto de vista demostrativo pero sin dejar a un lado los intereses educativos. Para ello, se ha propuesto un minijuego de montaje de las

diferentes piezas de las que consta un motor térmico de cuatro tiempos de cierta complejidad como puede ser un motor de ocho cilindros en V. Del mismo modo, se ha utilizado un vehículo lo suficientemente atractivo para atraer la atención del usuario, lo que deriva en un aumento del interés y de las capacidades de enseñanza del propio sistema. La aplicación también demuestra las posibilidades y virtudes de la utilización de un sistema virtual para la demostración de productos un tanto inaccesibles por el ciudadano medio.

Debido a la inclusión de funcionalidad mediante teclado, y para facilitar la navegación por el entorno, se ha incluido en el mismo un pequeño menú situado en la parte superior de la pantalla, el cual informa rápidamente del funcionamiento del sistema y que ofrece la accesibilidad necesaria para realizar la tarea de cambio de color.

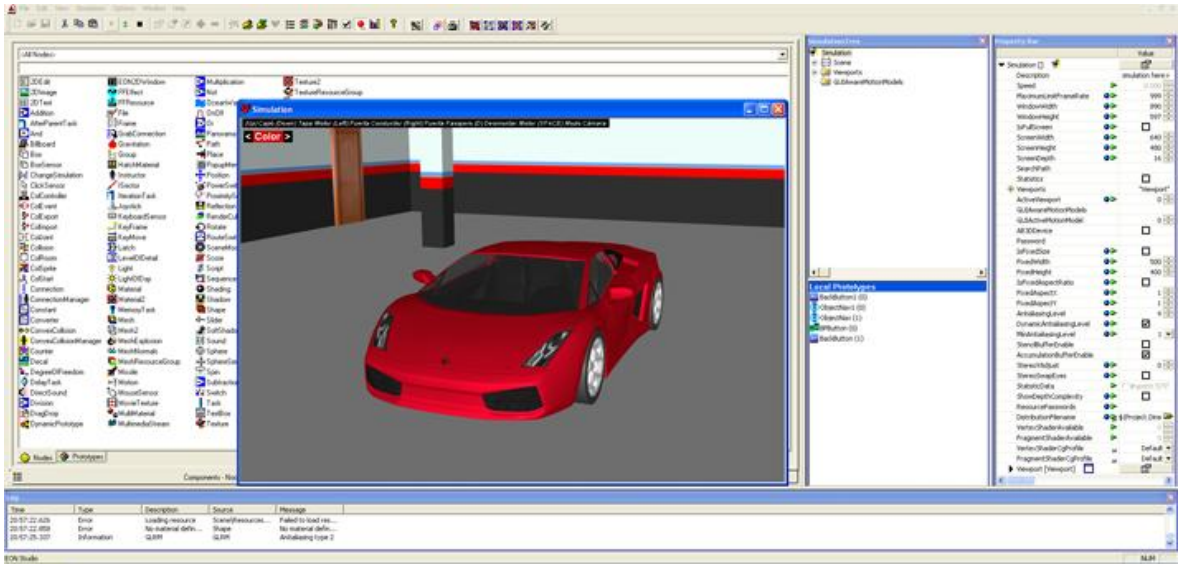

Ilustración 31 Visualización del Lamborghini Gallardo en EON Studio

Para poder completar toda la aplicación en EON, en primer lugar se debe generar un nuevo nodo de código en el árbol de navegación del proyecto y conectarlo a los nodos en los que se deseen agregar las nuevas funcionalidades que se implementan. Tras generar el nodo, aparece disponible el editor de código, pudiendo escoger si se desea realizar en ActionScript o JScript, como es el caso de este trabajo.

Se han implementado dos algoritmos diferentes, que se describen a continuación.

El primero de ellos consiste en dotar al modelo tridimensional de un pequeño panel para poder modificar, en tiempo real, los colores principales de la carrocería del vehículo. Así pues, se ha implementado un sistema que, mediante la pulsación de botones, aplica diferentes tonalidades de color, definidas previamente en el modelo estructural. Para ello, se debe realizar, mediante un algoritmo de recorrido de árboles, la modificación de los atributos correspondientes de color en cada uno de los nodos del árbol pertenecientes al conjunto de la carrocería del vehículo.

Por otro lado, se ha implementado también un pequeño sistema para el aprendizaje de la colocación de las piezas principales de un motor. Se ha definido sobre la tecla "D" un evento que se encarga de desmotar por completo el motor del vehículo y ocultar el resto de partes del mismo. Nuevamente, se debe realizar un recorrido por todo el árbol de elementos de EON y actuar atendiendo a la tipología de cada uno de los nodos de forma consecuente. A partir de ese momento, el usuario puede ir realizando *clic* en las distintas partes en las que se ha divido el motor, pasando éstas a montarse en la posición que le corresponde. Del mismo modo, se muestra en una ventana emergente el nombre de cada una de las partes en las que se pulsa, permitiendo la compresión y aprendizaje de qué elementos forman parte de un motor típico de combustión interna.

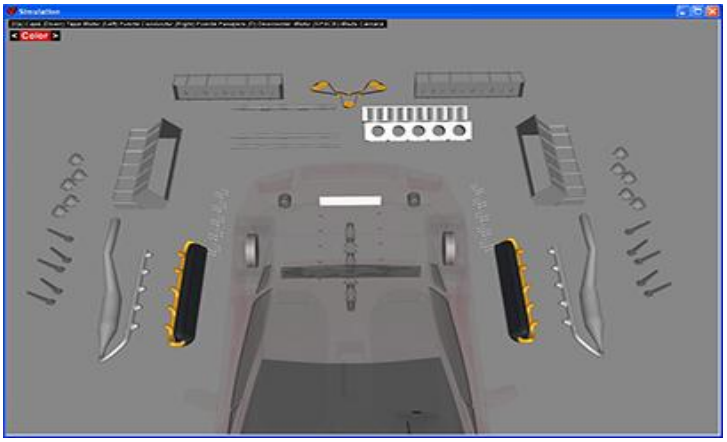

Ilustración 32 Comienzo del proceso de montaje del motor

Una vez reconstruido completamente el motor, éste vuelve a colocarse en su posición correcta dentro de la carrocería del vehículo, la cual vuelve a aparecer en la escena.

Para poder ejecutar estos métodos implementados y aplicarlos a las diferentes partes del modelo, ha sido necesaria la inclusión de diferentes entidades dentro del *Routes Simulation* del proyecto.

En primera instancia, se debe incluir todo el grupo de objetos del modelo poligonal sobre los que se quiere aplicar una implementación en el esquema. Posteriormente, y para cada uno de ellos, se debe definir el evento por el que se lanzarán las diferentes rutinas implementadas y mencionadas con anterioridad. Para realizar este paso, es necesario incluir en el árbol de navegación del modelo el tipo de manejador de eventos que se desea utilizar. Concretamente, para cada uno de los nodos poligonales en los que se desea poder efectuar las acciones diseñadas, se ha tenido que incluir, al mismo nivel, los nodos de movimiento necesarios para desplazar la malla por el espacio virtual y un nodo controlador de eventos de ratón a partir de un *clic* simple.

Posteriormente, se deben conectar los nodos entre sí en el *Routes Simulation*. También ha sido necesario aplicar ciertos operadores booleanos para poder controlar las acciones de colocar en el origen o desplazar a la zona de visibilidad cada una de las piezas.

Por otro lado, también se ha dotado al modelado de la funcionalidad de desplazarse por la escena de una forma sencilla mediante el ratón o el teclado y la posibilidad de abrir, mediante eventos de teclado o *clic* de ratón, las puertas del vehículo, el capó, el maletero y subir y bajar las ventanillas del vehículo.

Para el desarrollo y monitorización de la aplicación, se ha implementado un sistema de traza de *logs* más cómodo y completo que el que viene por defecto, el cual sólo permite escribir en una consola de errores como si se tratase de la salida estándar, sin poder asociar ninguna semántica a las diferentes trazas. En este sistema, se puede definir el nivel de trazas que se desea tener en la ejecución de la aplicación y mostrar consecuentemente sólo

las trazas correspondientes a este nivel. Del mismo modo, se ha agilizado la forma en la que se accede a los logs, disminuyendo considerablemente el tiempo de prueba de la aplicación.

3.7 Algoritmos de mejora de visualización y rendimiento:

Debido al alto número de piezas que contiene el modelo en su totalidad, se realiza un algoritmo de *culling* en la carga del mismo, mejorando sustancialmente el rendimiento de la aplicación en ordenadores de potencia más limitada.

El conjunto de piezas (y, por lo tanto, de polígonos) pertenecientes al motor es bastante elevado. Como la mayoría de las piezas son internas al bloque motor, en la carga de la simulación se ocultan automáticamente todas las piezas internas, dejando únicamente a la vista el bloque contenedor de éstas. En el momento de ejecutar la rutina de montaje del motor, se muestra al usuario todo el conjunto de piezas internas y se realiza un proceso de *Fade-Off* u ocultación del resto de piezas del motor. Tras finalizar el montaje del motor y mostrar nuevamente el vehículo, se ocultan automáticamente todas las piezas internas del motor, obteniendo nuevamente un aumento de rendimiento en la aplicación.

Del mismo modo, para esta aplicación *real-time* se han tenido que modelar nuevamente algunas de las piezas en mallas de bajo nivel puesto que las cotas de detalle de algunas de las piezas no llegan a los niveles de la animación.

En los sistemas de realidad virtual, como en cualquier película, historia o videojuego, el sonido se convierte en un gran aliciente y complemento para introducir al usuario en un ambiente inmersivo. En la aplicación en tiempo real se ha optado por introducir en el sistema el sonido del vehículo a ralentí, dotando a la simulación de la sensación de estar realmente en una habitación con el vehículo. EON Professional permite introducir cualquier sonido de ambiente introduciendo el nodo DirectSound dentro del árbol de visualización. Este nodo debe ser configurado y asociado a un archivo en formato WAV, siendo este formato el único permitido. A su vez, se pueden definir parámetros como el momento de reproducción/pausa o bien la reproducción cíclica constante.

CAPÍTULO 4 - Resultados

A continuación se muestran los resultados obtenidos en el presente trabajo.

4.1 Modelado del Lamborghini Gallardo

El modelo final del vehículo consta de un total de 217000 polígonos en total, sin aplicar ningún filtro de suavizado previo, de los cuales 140000 corresponden únicamente al modelado del motor, cifras que dan una idea del nivel de detalle alcanzado en esta zona para su correcta visualización en la animación.

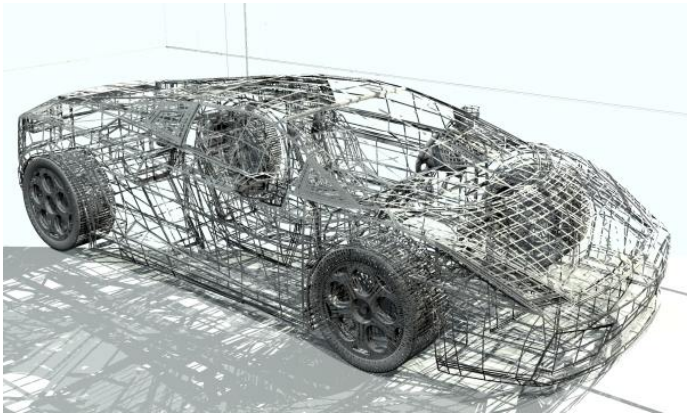

Ilustración 33 Modelo de alambre del Lamborghini Gallardo (vista perspectiva)

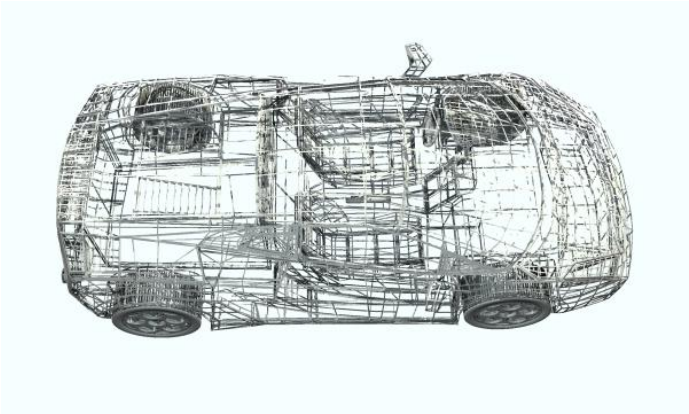

Ilustración 34 Modelo de alambre del Lamborghini Gallardo (vista superior)

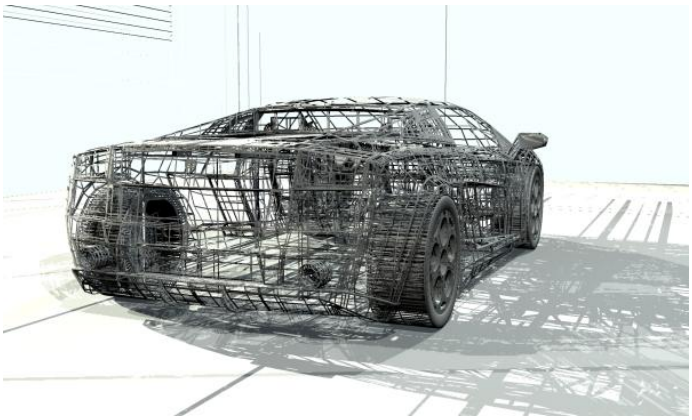

Ilustración 35 Modelo de alambre del Lamborghini Gallardo (vista trasera)

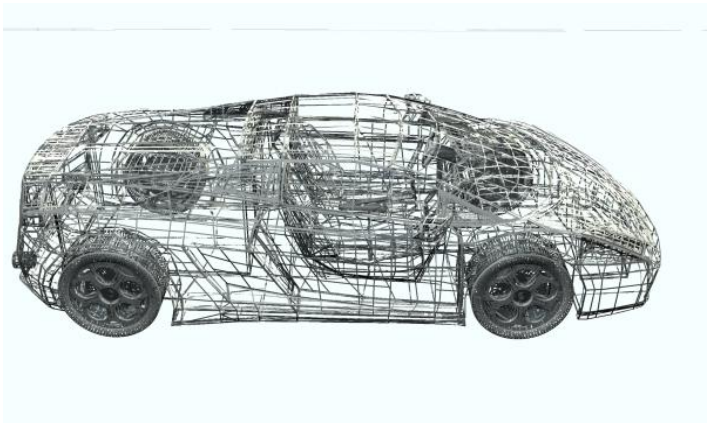

Ilustración 36 Modelo de alambre del Lamborghini Gallardo (vista lateral)

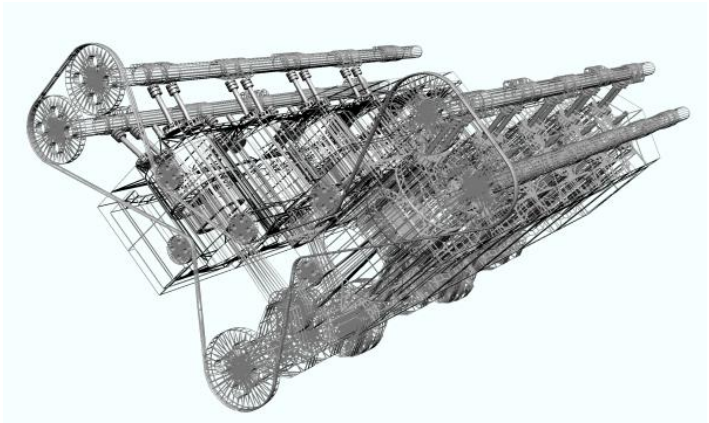

Ilustración 35 Modelo de alambre de las piezas internas del motor

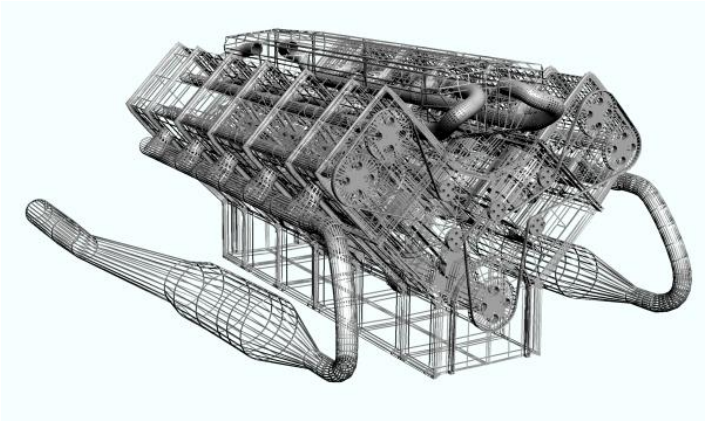

Ilustración 36 Modelo de alambre del motor en su totalidad

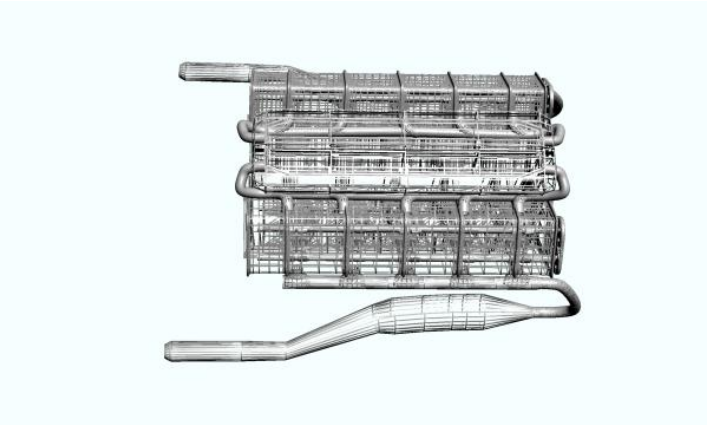

Ilustración 37 Modelo de alambre del motor (vista lateral)

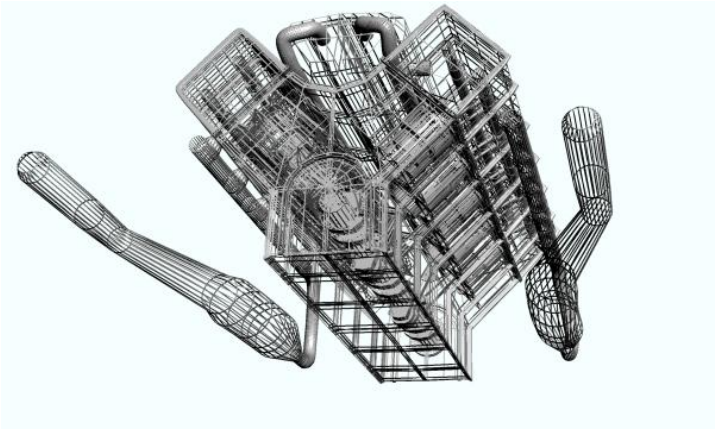

Ilustración 38 Modelo de alambre del motor (vista trasera)

4.2 Animación

Tras todo el proceso de modelado, se ha renderizado una breve animación en alta resolución (720p) donde se muestra todo el modelo tridimensional realizado. La animación consta principalmente de dos fases:

La primera, muestra el modelado exterior del vehículo dentro de una habitación o garaje. También se pueden observar en diversos momentos de la animación los efectos de la utilización de un sistema lumínico complejo, con un total de seis luces repartidas por la escena. Una de ellas, se ha configurado de tal forma que simula la entrada de luz solar por unas rendijas desde el exterior, así como el efecto visual que produce esta clase de iluminación sobre los diferentes materiales que componen el vehículo junto con el estudio del coste computacional en el renderizado de estos sistemas de iluminación mediante traza de rayos o *raytracing*.

Por otro lado, la animación consta de una sección más orientada al entendimiento y aprendizaje básico del funcionamiento de un motor térmico de cuatro tiempos con sus principales piezas móviles. La animación realiza un pequeño recorrido por los diferentes componentes del motor y muestra una imagen general del sistema, de cómo un movimiento rectilíneo se transforma en movimiento circular para luego ser transferido a las ruedas.

Ilustración 39 Imagen final del Lamborghini Gallardo

Ilustración 40 Imagen final de la parte trasera del Lamborghini Gallardo

Ilustración 41 Imagen final del interior del Lamborghini Gallardo

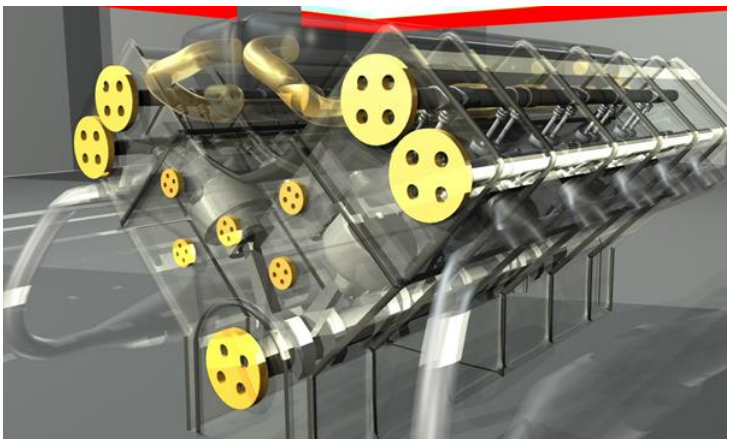

Ilustración 42 Imagen final del motor del Lamborghini Gallardo (vista perspectiva)

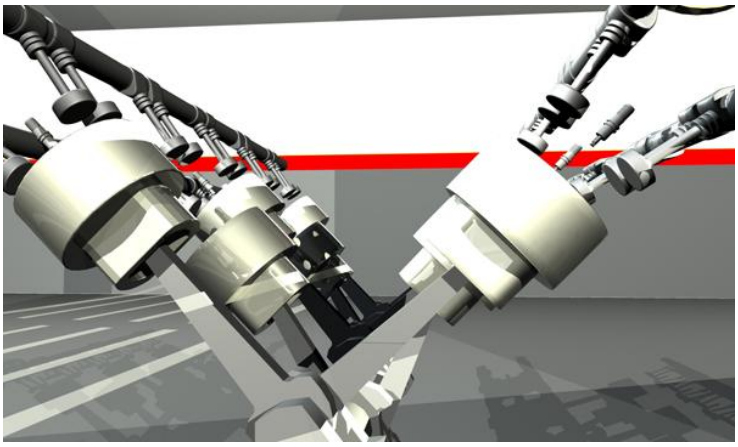

Ilustración 43 Imagen final del interior del motor del Lamborghini Gallardo (vista frontal)

4.3 Aplicación en tiempo real y Cueva de Realidad Virtual

Se ha desarrollado una aplicación que sirve a la vez de demostración y visualización del vehículo modelado, así como de pequeña aplicación de aprendizaje.

Esta aplicación consta de tres partes principales: La primera de ellas permite explorar y recorrer la escena tridimensional para poder visualizar desde diferentes ángulos tanto el interior como el exterior del vehículo. Todo ello se realiza desplazando la cámara en primera persona por la escena, utilizando el teclado o el ratón. La segunda corresponde con ciertas funcionalidades para dotar al modelo de mayor dinamismo. Es posible abrir y cerrar las puertas (nuevamente, mediante ratón o teclado), bajar las ventanillas, levantar el capó o abrir el maletero que, curiosamente, se encuentra en la parte anterior de la carrocería. Además, mediante un pequeño menú de opciones en la pantalla, se permite modificar el color de la carrocería del coche entre un pequeño abanico definido. Por último, se ha desarrollado un módulo de aprendizaje en el cual se desmonta la totalidad del motor y se activa una rutina de ensamblado del mismo. Mientras se ensamblan las piezas, se muestra en pantalla la pieza que se está ensamblando, permitiendo al usuario conocer las diferentes partes que componen el motor. En todo momento el desplazamiento está permitido por la escena, pudiendo, si se desea, visualizar las partes más pequeñas con mayor detalle.

4.4 Mejoras de rendimiento

Como se ha comentado, las pruebas de rendimiento se han realizado en dos apartados diferentes, el renderizado para la generación de la animación y la aplicación de tiempo real. La realización de un sistema virtual lo más realista posible conlleva varios problemas técnicos con los que se ha de lidiar siempre: la complejidad de la malla del modelo, la utilización de un sistema de iluminación y sombreado que genere unas imágenes lo más realistas posibles y la contención de los tiempos de renderizado en la medida de lo posible.

El renderizado de la animación permite la utilización de modelos mucho más complejos que cualquier sistema en tiempo real, lo que se traduce a posteriori en poder

obtener una calidad de imagen muy superior, a costa del empleo de un mayor tiempo de renderizado pero, aun así, se tienen que establecer dentro de unos límites lógicos.

La animación consta de dos mil cuatrocientas imágenes en alta resolución, las cuales han tardado en renderizarse un tiempo aproximado de veinticinco horas. Para este proceso se ha utilizado un ordenador personal de cuatro núcleos y cuatro gigabytes de memoria RAM de alta velocidad 1333Mhz trabajando en dual-chanel. Aparte, para la mejora de tiempos de renderizado, se dispuso de una tarjeta NVIDIA dotada de tecnología CUDA, pudiéndose utilizar los núcleos de la tarjeta para realizar parte del cálculo computacional.

Las primeras pruebas, con un sistema lumínico sin estudio, tardaban cuatro minutos y medio en realizar un simple render de la parte exterior del coche. Tras la modificación de diferentes valores para el renderizado, como son la cantidad de rayos trazados de origen por cada luz, la cantidad de rayos emitidos en cada reflexión de ésta, la calidad de los filtros de suavizado (antialiasing) y la optimización del sistema de iluminación, se ha podido reducir el tiempo para la misma imagen hasta los veinte segundos. En otros, casos como el proceso de transparencia del vehículo para mostrar el motor y sus partes, la diferencia es más evidente, pasando a tardar en cada una de las imágenes de veinte minutos o media hora a tres o cuatro minutos.

Por otro lado, para la aplicación en tiempo real, fue necesaria la reducción de polígonos total del modelo, teniendo que sacrificar en gran medida la calidad gráfica del entorno. Así mismo, como ya se ha comentado, ha sido necesaria la utilización de un algoritmo de *culling* para reducir el número de polígonos cargados en la escena. En el sistema de iluminación de la aplicación en tiempo real se han limitado el número de fuentes de luz y su complejidad, primando en todo momento la fluidez del sistema frente a la calidad visual.

CAPÍTULO 5 - Conclusiones

Una vez finalizado el proyecto es importante recapitular sobre los objetivos cumplidos y analizar los resultados obtenidos.

Tal y como se deseaba, se ha generado completamente un modelado tridimensional sobre un vehículo lo suficientemente atractivo y exclusivo que logre generar el interés del observador tan solo con su presencia. Esta peculiaridad del vehículo ha provocado que el modelado tridimensional se tuviese que hacer sin poder visualizar realmente lo que se modela, basándose en unas pocas imágenes y vídeos de Internet, además de una maqueta a pequeña escala.

La necesidad de reducir el coste computacional de la aplicación en tiempo real ha provocado el estudio de las diferentes técnicas para la mejora de rendimiento en gráficos y visualización 3D, tales como la implementación de un algoritmo de *culling* o la sustitución de polígonos mediante el uso de texturas.

Se han generado los principales componentes del motor y se ha realizado una animación sobre éstos para demostrar la utilidad e importancia de la realidad virtual en el campo educativo, mostrando de una forma entretenida el funcionamiento de un motor de cuatro tiempos.

Todo este proceso ha servido para poder asimilar de una forma más profunda el trabajo necesario para realizar un entorno tridimensional aparente, teniendo que ajustarse a las peculiaridades de un sistema tan específico como es la Cueva de Realidad Virtual de la Universidad Rey Juan Carlos. Por supuesto, los conocimientos adquiridos en modelado y animación me permiten abrir mi campo de desarrollo profesional a un sector que factura cada año más que la propia industria cinematográfica.

Ha sido muy positivo, dado mi interés en el desarrollo de videojuegos para plataformas móviles, entender el proceso de modelado en su totalidad y tener que adecuarse a las limitaciones de hardware de sistemas tan cerrados como son estos dispositivos.

La utilización de diferentes herramientas de ámbito profesional como son 3ds Max y EON Studio 5.5 ha provocado que me enfrente realmente a la necesidad de encontrar información muy concreta y específica sobre cuestiones poco frecuentes. En muchas de las ocasiones la única información disponible era la propia documentación de la aplicación, lo que limitaba mis posibilidades de reacción, pero provocaba que fuera necesario el desarrollo de soluciones propias, basadas únicamente en mi observación de los resultados obtenidos de una acción u otra y basándome únicamente en mi propio criterio.

Por otro lado, se ha logrado comprender realmente la utilizad de los entornos de realidad virtual con fines educativos y no solo lúdicos. La importancia de estos sistemas de aprendizaje está en auge en este comienzo del siglo XXI dados sus resultados satisfactorios. Se ha probado a mostrar la animación y la aplicación a personas con muy bajos o nulos conocimientos sobre mecánica y, en la mayoría de ellos, se ha observado la facilidad de entendimiento del funcionamiento del motor, aparte de poder observar cómo su reacción a esta forma de aprendizaje es mucho más positiva que, por ejemplo, estudiar el funcionamiento en un libro. La mayoría de los usuarios, tras la primera visualización, no dudaban en desear realizar una segunda visualización o ejecución de la aplicación aún tratándose de temas que a priori no son de su interés.

Aunque todo el proceso ha sido muy costoso y ha requerido bastante tiempo por mi parte, estoy satisfecho por haber adquirido una serie de conocimientos sobre un tema que es tan poco o nulamente tratado dentro de la propia carrera, ampliando mis conocimientos en un campo en el que considero que las posibilidades de evolución, desarrollo y compensación tanto profesional como personal son muy grandes.

CAPÍTULO 6 - Líneas de trabajo futuras

Como todo trabajo realizado, siempre es posible realizar modificaciones y ampliaciones. En primer lugar, y centrando en la aplicación de tiempo real, se podría realizar una simulación con mayores funcionalidades, atendiendo a las mejoras de los efectos visuales. Mediante el uso de técnicas de simulación más avanzas, se puede aumentar considerablemente el realismo del entorno gráfico, lo que conlleva una mejora significativa en la sensación de inmersión en la escena por parte del usuario. La iluminación en la aplicación de EON es de los sistemas de iluminación más básicos posibles. Evidentemente, la mejora en este aspecto provoca el estudio del impacto computacional de un sistema más complejo, por lo que se decidió dejar a un lado todas estas mejoras en el proyecto.

Atendiendo a la misma idea de mejora de la aplicación en tiempo real, se podrían desarrollar nuevas rutinas o minijuegos destinados al aprendizaje mediante entornos de realidad aumentada. En estos momentos, dejado a un lado la funcionalidad de movimiento implementada, existe una única rutina con carácter educacional en la que se muestran las principales piezas del motor, así como su nombre y posición de montaje, al ir realizado *clics* sobre ellas. Con el desarrollo de nuevas funcionalidades es posible profundizar más en los diversos aspectos educativos que derivan de un vehículo y un motor.

Otra línea de trabajo futuro se encuentra claramente en la utilización del modelo tridimensional para el desarrollo de una aplicación avanzada. Las posibilidades de esta rama son muy amplias, desde el desarrollo y utilización del sistema en un videojuego o simulador de conducción y carreras con fines educativos.

Siguiendo con el hilo anterior, podría realizarse la aplicación para su ejecución en un entorno que no fuera únicamente la Cueva de Realidad Virtual de la Universidad. Con el auge de los dispositivos móviles por parte de la población, puede ser interesante la adecuación y desarrollo del modelo en una plataforma como iOS o Android, desarrollando

una aplicación que fuese fácilmente utilizada por cualquiera y aprovechándose de los medios de distribución que poseen estas plataformas.

Por otra parte, el proyecto es susceptible en verse ampliado en la profundidad y detalle del conjunto del motor y transmisión. Un motor de cuatro tiempos como éste posee cientos de piezas ensambladas de forma milimétrica y trabajando al unísono, que transforman el simple movimiento producido por la explosión de la mezcla de aire-gasolina en un cilindro, en un movimiento circular en las ruedas motrices. Todo ese complejo subconjunto de piezas y movimientos puede ser modelado y simulado para su utilización tanto con fines comerciales y promocionales, como con fines educativos para ingenieros y aficionados.

Por último, teniendo una perspectiva más comercial, el desarrollo de aplicaciones de demostración visual de vehículos se encuentra a la orden del día. El interés de las marcas por mostrar a posibles compradores de un vehículo las diferentes configuraciones de catálogo de un mismo modelo, de una forma rápida y atractiva, abre un nuevo abanico de diseño y desarrollo de nuevas piezas para mostrarse según el rango de opciones. Estos cambios podrían ir desde el simple cambio de color del vehículo, como ya se está realizando, a cambiar la configuración de llantas y acabados opcionales.

CAPÍTULO 7 - Bibliografía

- Ingo Wald and Philipp Slusallek; Computer Graphics Group (2001). Saarland University. ***State of the Art in Interactive Ray Tracing***

- Cyrille Damez, Kirill Dmitriev and Karol Myszkowski; Max-Plank-Institut für Informatik (2003). ***Global Illumination for Interactive Applications and High-quality Animations***

- Ingo Wald; University of Utah (2007). ***State of the Art in Ray Tracing Animated Scenes***

- Juan Torres Arjona and José Manel Menéndez; Universidad Politécnica de Madrid. ***Virtual reality devices in driving simulator.***

- Ruth Aylett and Marc Cavazza; University of Salford; University of Teesside. ***Intelligent Virtual Enviroments***

- Isaac V.Kerlow; John Wiley & Sons, Inc. **The Art of 3D *Computer Animation And Efects.*** Third Edition

- Leonardo Fernánndez Jambrina; Universidad Politécnica de Madrid. ***Curvas de Bézier***

- Universidad Rey Juan Carlos (2011). ***Apuntes del Máster en Informática Grafica, Juegos y Realidad Virtual***.

- Alan Watt; Addison-Wesley. ***3D Computer Graphics.*** Third Edition

- Thomas Akenine-Möller, Eric Haines, Naty Hoffman; A.K. Peters Ltd. (2008). ***Real-Time Rendering***. Third Edition.

- ***Autodesk – Home***. www.autodesk.es
- ***Wikimedia Commons***. *commons*.wikimedia.org
- ***Fundación Wikimedia***, Inc Wikipedia.org
- **EON Reality, Inc.** – ***The world's leading interactive 3D visual content management and Virtual Reality software provider***. www.eonreality.com.
- ***MSDN Home Page***. msdn.microsoft.com
- ***Gamer's Graphics & Display Settings Guide.*** TweakGuides.com

Anexo A: Dispositivos de realidad aumentada y su utilización como herramientas de aprendizaje

El aprendizaje digital ha superado en los últimos años todas las expectativas en su utilidad para la enseñanza. Los entornos virtuales y los juegos educativos han mejorado el nivel de aprendizaje, entendimiento y motivación de los alumnos. Atendiendo a este objetivo, se han promovido múltiples estudios sobre la manera en la que los propios alumnos desean realizar su aprendizaje.

La introducción de simuladores espaciales y de vuelo en Estados Unidos ha sido el comienzo de los entornos virtuales de aprendizaje y los juegos educativos. En 1952, los primeros videojuegos hacen su aparición, algunos de ellos basados en la simulación táctica-militar, desarrollados por el laboratorio de defensa aérea de Santa Mónica. En los años 80 y 90, la agencia de investigación de proyectos avanzados de defensa (DARPA) comienza el programa de investigación de simulación en red (DARPA). Sus avances establecieron las bases del protocolo de intercambio de datos para la simulación, los cuales fueron predecesores del protocolo de simulación interactiva distribuida (DIS, Kincaid 2006). En el año 2002, el Centro internacional Woodrow Wilson funda la iniciativa Serious Video Games en 2002, la cual establece las nuevas iniciativas en el desarrollo educativo, las herramientas de exploración y mantenimiento del "State-Of-The-Art" para el desarrollo de videojuegos (SGI, 2008). Desde entonces, los entornos de aprendizaje virtuales y los juegos educativos se encuentran en continuo proceso de evolución,

Un juego educativo supone un reto que combina las metas de aprendizaje y entretenimiento, utilizando conceptos como puntuación, victorias y derrotas. Una cualidad, conocimiento o aptitud puede ser desarrollada a partir de juegos educativos. Los entornos virtuales de aprendizaje han dado origen a multitud de combinaciones de sistemas de aprendizaje inteligente (ITSs) con entornos virtuales.

Tradicionalmente, los entornos virtuales están formados por diferentes módulos, el módulo de dominio, el módulo del estudiante, el modelo de aprendizaje y la interfaz gráfica del usuario (GUI, *Graphical User Interface*, por sus siglas en inglés).

Estos módulos pueden contener diferentes técnicas de realizad virtual, las cuales utilizan las representaciones gráficas del dominio de conocimiento, ayudando al estudiante a entender y desarrollar una respuesta inteligente (Freedman, 2000). Estos entornos otorgan al estudiante la ventaja de realizar un aprendizaje y seguir un plan de resultados instantáneos ofreciendo una respuesta de aprendizaje.

Actualmente, muchos grupos de investigación están desarrollando nuevos sistemas de aprendizaje inteligente (Conati & Maccleren, 2009; D´Mello et al., 2008; Sarrafzadeh et a1., 2008; Chaffar & Fransson, 2004; Chalfoun/Frasson, 2008). Estos grupos de investigación sugieren un aprendizaje sin recompensas al conocimiento del estudiante, entendiendo sus propias circunstancias personales y su disposición emocional, ofreciendo para ello diferentes líneas de aprendizaje para cada alumno.

En comparación con la enseñanza tradicional, la enseñanza a través de entornos virtuales tiene ventajas y desventajas. En la enseñanza tradicional, el conocimiento depende del tutor que importa las clases, la experiencia del lector y sus conocimientos previos en el campo. Por otro lado, los entornos virtuales educativos pueden llegar a ofrecer todo el conocimiento que los estudiantes necesitan aprender, sin depender en tanta medida del profesor que los imparta. Por otro lado, en la enseñanza tradicional, tan solo se tiene un tiempo específico para cada una de las partes, pudiendo en el caso de la enseñanza virtual ser el propio alumno el que decida cuánto tiempo y cuando emplearlo para cada una de las partes, siendo capaz además de adaptarse al ritmo y forma de aprendizaje de cada estudiante. Otro factor importante a tener en cuenta es que esta clase de enseñanza siempre está disponible para el estudiante, aunque depende de la disponibilidad de los recursos y de la implementación de diversos mecanismos de inteligencia (Bergeron, 2005; Hsiao, 2007). En resumen, el uso de la realidad virtual aumenta la interacción con el propio estudiante y

ofrece un sistema personalizado de enseñanza y forma de adquirir los conocimientos, adecuándose a las propias habilidades del estudiante. Sin embargo, en la enseñanza tradicional, la relación humana con el tutor es capaz de adaptar el estado emocional a la enseñanza, pudiendo seguir diferentes caminos.

1 Sistemas de enseñanza inteligente (Inteligent Tutoring Systems)

Los sistemas de enseñanza inteligentes (SEI) forman parte de la metodología de aprendizaje mediante sistemas de enseñanza de realidad aumentada y juegos educacionales, promoviendo a los estudiantes a alcanzar diferentes metas. Actualmente, las investigaciones se centran en la representación del dominio de conocimiento, del aprendizaje y en la adaptación de los sistemas pedagógicos clásicos. Para ello, se construyen modelos abstractos, los cuales deben aprender cómo el alumno va aprendiendo y analizar cómo es su proceso de respuesta al sistema, pudiendo responder de una forma dinámica. La definición del modelo de aprendizaje se refiere a las diferentes estrategias pedagógicas que deben ser implementadas, asumiendo que los errores y los aciertos del estudiante marcan el camino a seguir para facilitar el aprendizaje.

La enseñanza necesita diferentes formas de comunicación, tales como la explicación, respuesta, persuasión, demostración, etc. En un principio, en los sistemas virtuales sólo se atiende a la superación de unos hitos aislados, sin atender a todas las complicaciones y circunstancias que han surgido en este proceso.

Existen varias técnicas a seguir por estos “tutores virtuales”, como pueden ser el tutor LISP, GUIDON o MENO.

El tutor LISP fue diseñado por Anderson & Skwarecki en 1986 y pretende implementar todas las posibles respuestas por parte del tutor para cada pregunta del estudiante, dando lugar a una multitud de reglas lógicas. Posteriormente, y atendiendo a esas reglas, se genera una respuesta en un lenguaje natural, formando una metodología de

Model-Tracing, donde el tutor debe analizar constantemente las acciones del estudiante, guiándolo por el proceso mientras éste va resolviendo diferentes problemas.

El tutor virtual GUIDON y el tutor virtual MENO, por el contrario, incorporan habilidades de comunicación. El tutor GUIDON se centra en encontrar los caminos que representan la enseñanza y la resolución de problemas, manejándolos por separado. El tutor GUIDON está basado en la enseñanza socrática, la cual propone que el proceso de aprendizaje debe ser realizado a través de las cuestiones que se plantee el propio alumno, en lugar de verter sobre él una pila de datos y conocimientos esperando a que éste sea capaz de asimilarlos. Por otro lado, el tutor MENO utiliza el lenguaje natural para ofrecer al alumno los diferentes temas o dominio del conocimiento y atiende a sus cuestiones e inquietudes gracias a nuevas explicaciones. Para obtener una comunicación más efectiva, es necesario registrar el significado semántico de multitud de palabras que el estudiante puede no ser capaz de comprender. A su vez estos problemas plantean el reto de determinar qué conocimientos son correctamente entendidos y cuáles son las cuestiones que pueden ser formuladas.

2 Sistemas de enseñanza inteligente actuales

Autotutor

Este sistema es utilizado en los entornos virtuales de aprendizaje para enseñar Física newtoniana y los pensamientos en la enseñanza primaria. Estudia los tipos de humor, aburrimiento, compromiso y dificultad, analizando los diálogos de comunicación, el lenguaje corporal y los gestos. El autotutor interacciona con el estudiante mediante lenguaje natural escrito, nutriéndose de una serie de preguntas directas, evaluaciones escritas del alumno y sus propias preguntas, además de analizar los conceptos erróneos adquiridos de éste.

El autotutor se centra en cambiar los estados emocionales negativos, como el aburrimiento, frustración y confusión, promoviendo la comunicación e interacción continua. El desafío es conseguir la intervención por parte del tutor sin que los estudiantes sean capaces de percibirla, dejando de ser un impedimento para su avance.

Las conclusiones obtenidas por medio de los diálogos no aseguran por sí mismas los estados en los que se encuentra el alumno, y el uso de hardware externo para el reconocimiento de estas sensaciones puede ser demasiado invasivo, provocando que el alumno no se encuentre cómodo y que el uso de este hardware no sea efectivo. Es muy importante distinguir entre estados de ánimo y emociones. Desde un punto de vista psicológico, las emociones conllevan decisiones en los sistemas, al contrario que los sentimientos, los cuales pueden condicionar su intensidad por muchos factores externos.

PrimeClimb

PrimeClimb es un juego diseñado para la enseñanza de matemáticas para chicos de entre diez y doce años. En este sistema, las emociones del estudiante son captadas usando un modelo de estudiantes DBN (Dynamic Bayesian Networks), el cual establece que las emociones son dependientes del trato personal y el contexto. También defiende que se ha de intervenir en el proceso de aprendizaje aumentando la motivación, utilizando métodos intrusivos si hiciera falta. Así mismo, se debe ir considerando en todo el proceso cuáles son las metas que el propio alumno debe conseguir, dando a los propios alumnos una gran libertad de opción en su aprendizaje.

Easy with Eve (Aprende con Eva)

Este sistema de aprendizaje está pensado para la enseñanza de Matemáticas en primaria, analizando los estados cognitivos y afectivos del alumno para adaptar estrategias pedagógicas en función de estos dos condicionantes, basándose en análisis facial. Para

analizar estos posibles estados se utiliza una webcam y una red neuronal que, posteriormente, aplicará un algoritmo de reconocimiento. Este algoritmo identifica multitud de imágenes por alumno y las compara entre sí, intentando identificar para dicho alumno expresiones de sorpresa, tristeza miedo, etc., con diferente tasa de éxito.

Esta estrategia de pedagogía fue ideada a partir de la observación de vídeos donde profesores enseñan a sus alumnos diversos conceptos sobre la audición. En los vídeos se analizan las expresiones faciales de cada uno de los alumnos, sus intensidades y frecuencias, en función de la clase del profesor, buscando patrones de interacción entre ambos. Simulando a la acción del profesor, que toma diferentes patrones según la situación del alumnado, se diseña un algoritmo que busca en los alumnos los mismos gestos, actuando consecuentemente con ellos. Si no es capaz de encontrar el gesto deseado, prosigue con la enseñanza y espera a encontrar el patrón buscado.

Como problema al sistema, existen problemas de rendimiento graves por falta de memoria, así como la falta de respuesta en situaciones en las que el estado del alumnado no se encuentra registrado. Sin embargo, las investigaciones siguen la línea de encontrar diferentes técnicas de inteligencia artificial para aprender las nuevas situaciones.

EMASPEL (Sistema PeerToPeer de agentes multiemocionales para el aprendizaje)

El sistema EMASPEL fue creado en el año 2007 para enseñar teoría de la telecomunicación para alumnos de secundaria. El sistema reconoce el estado afectivo del estudiante utilizando una webcam y analizando los rasgos faciales, los cuales son clasificados con un sistema de distancias.

3 Dispositivos de realidad virtual

Los avances en realidad virtual siempre van de la mano con el incremento de la potencia computacional y las mejoras en renderizado, así como con el cálculo de las operaciones de gráficos. Simular una escena está siendo poco a poco mucho más sencillo, obteniendo resultados cada vez más realistas. Otro factor a tener en cuenta, y que favorece el desarrollo, es la caída de precios en el equipamiento y hardware específico para realizar dichas tareas.

Uno de los campos que han favorecido todo esto es la gran importancia de desarrollo de sistemas de conducción virtuales. Esos sistemas ofrecen una genial herramienta para estudiar las reacciones de los conductores, sin necesitar un coche real y sin provocar una situación real de peligro. Por supuesto, este espectro de utilidades no se queda tan solo aquí, teniendo en los propios sistemas de entretenimiento un área de aplicación muy conocida por todo el público en general.

En estos sistemas, es muy importante tener una sensación de realidad muy elevada, sirviéndose en ocasiones de entornos totalmente virtuales, con sistemas sobre un coche real, donde se aplican diferentes efectos de movimiento y fuerzas, aumentando considerablemente la sensación de inmersión en el sistema. Recíprocamente, los datos de los diferentes usuarios son analizados con el propósito de obtener diversa información sobre sus reacciones y su comportamiento. La importancia de esta información es muy elevada para la mejora del sistema y poder averiguar cuál es su estado y analizar las próximas mejoras a realizar.

Hoy en día, el renderizado de los entornos virtuales se realiza mediante diferentes algoritmos de rasterización que se ejecutan sobre un hardware muy específico. La optimización ha aumentado considerablemente en los últimos años, siendo la mayoría de los ordenadores actuales equipos suficientes para desempeñar estas tareas. Este considerable aumento está siendo suficiente para muchas aplicaciones, pero otras como los

videojuegos y las aplicaciones tridimensionales siguen teniendo grandes restricciones para el renderizado en tiempo real.

El hardware gráfico actual conlleva numerosas dificultades en este aspecto. La visualización en tiempo real necesita un preprocesado bastante sofisticado para reducir el número de polígonos a visualizar por cada frame, o cuadro de la imagen a representar por pantalla. Esto se consigue mediante técnicas como la utilización de diferentes niveles de detalle, el ocultado de polígonos no visibles, la simplificación métrica, etc. Existen, por lo tanto, otras cuestiones como el procesado en paralelo de estos aspectos, la comunicación entre sistemas y el cálculo redundante.

Otro aspecto a tener en cuenta a la hora del desarrollo tridimensional es el compromiso entre realismo y rendimiento. Una escena con una calidad gráfica muy alta pero con un rendimiento muy pobre ofrece un nivel de inmersión bajo, igual ocurre al contrario, una aplicación que se ejecuta con fluidez y con muchas funcionalidades pero que presenta un aspecto muy poco real, no ofrece una experiencia positiva.

Andexo B: Código JScript implementado

La funcionalidad JScript implementada se puede dividir en cuatro partes funcionales; definición de variables e inicialización, rutinas de montaje del motor, cambio de color y logs o mensajería.

1. Inicialización y definición de constantes

En esta parte se definen todas las constantes necesarias para la definición de colores y los niveles de trazas, además de las variables de control de las distintas funcionalidades. Del mismo modo se configura la activación de trazas y el nivel de éstas, pudiendo definirse a nivel de *debug*, *info* o *error*.

```
//Log variables & constants
var MOTOR = "-motor";
var traceActive=true; //sets true for trace
var DEBUG = 2;
var INFO = 1;
var ERROR = 0;

var RED=0;
var BLUE=1;
var BLACK=2;
var GOLD=3;
var GREEN=4;
var WHITE=5;

var traceMode = DEBUG;
var enableMaterialMod = 1; //enable material fadeoff

//Script variables
var parts;
var partsCount;
var unmountNode;
```

```
var partsList;
var colorNode;
var colorMaterial;
var colorCounter;
/**
* Inicialice function call on load
**/
function initialize()
{
	// Called when simulation starts
	// Add initialization code
	print.info("--- initialize");
	parts = new Array();
	partsList = new Array();
	materialsOpacities=new Array();
	colorCounter = -1;
	partsCount = 0;
	unmountNode = eon.FindNode("PulsarD");
	unmountNode.GetFieldByName("enabled").value = 1;
	colorNode = eon.FindNode("Msg Color");
	colorMaterial = eon.FindNode("-Coche-Carroceria");
	msgNode = eon.FindNode("Msg");
	resetMessage();
	On_NextColor();
	initMaterialsOpacity();
	saveMaterialsOpacity();
	getParts("Motor");
	setClickParts(0);
	print.info("--- initialize END")
}
```

2. Rutinas de montaje del motor

Esta parte incluye las funciones necesarias para ejecutar la funcionalidad de montaje y desmontaje del motor en su totalidad.

El método *On_ClickNode()* se ejecuta en el evento *OnClick* de las distintas piezas del motor. Este se encarga de controlar las diferentes piezas que se han montado e ir mostrando el nombre de la pieza sobre la que se realiza click. En caso de que se complete el montaje del motor, restaura el aspecto visual del vehículo.

```
/**
* Function call on click motor node
**/
function On_ClickNode()
{
	print.info("On_ClickNode");
	//PulsarD nodo de desmontaje
	if (partsCount==0){
		unmountNode.GetFieldByName("enabled").value = 0;
		print.debug("Bloqueo desmontar");
	}

	//No se usa GetNodeName debido a que el string contiene mas datos que controlar
	var path = eon.GetNodePath(ClickNode.value);
	path = path.substring(0,path.indexOf(MOTOR)-1);
	path = path.substring(path.lastIndexOf("\\")+1);
	insertPartName(path);
	if (path.indexOf("Tapa explosiones")!=-1){
		path = path.substring((path.indexOf("explosiones")+"explosiones".length));
		path="CarterSuperior"+path;
		print.debug("<"+path+">");
	}
	var clickerNodeName = "Click"+path.replace(/ /g,"");
	print.debug("ClickerNodeName >"+clickerNodeName);
	var clickerNode = eon.FindNode(clickerNodeName);
	clickerNode.GetFieldByName("changeCursor").value=0;
	clickerNode.GetFieldByName("Button").value=-1; //Disable
	if (isEngineMount()){
		eon.FindNode("-interior-Caja Motor").GetFieldByName("Hidden").value = 0;
		unmountNode.GetFieldByName("enabled").value = 1;
		restoreMaterialsOpacity();
		eon.FindNode("Motor Interior").GetFieldByName("Hidden").value = 1;
```

```
                print.debug("Desbloqueo montar y muestro el coche");
                resetMessage();
        }
        print.info("On_ClickNode END");
}


/**
* Function call after motor node movement
**/
function On_MoveEnd()
{
        resetMessage();
}
```

El método *On_MotorMode()* activa el sistema de montaje del motor, desbloqueando la funcionalidad sobre cada una de las piezas y oculta o transparenta el resto de piezas del vehículo.

```
/**
* Function call on run motors parts mode
**/
function On_MotorMode()
{
        print.info("On_MotorMode");
        eon.FindNode("Motor Interior").GetFieldByName("Hidden").value = 0;
        eon.FindNode("-interior-Caja Motor").GetFieldByName("Hidden").value = 1;
        transparenceMaterial(2);
        setClickParts(1);
        print.info("On_MotorMode END");
}

/*************************************************************************
**********************************ENGINE FUNCTIONS************************
*************************************************************************/
```

El método *insertPartName()* registra la pieza que esta siendo montada y muestra un mensaje indicando su nombre.

```
/**
* Save the motor parts inserted
**/
function insertPartName(part){
	print.info("--- insertPartName");
	if (part!=null && part!=""){
		showMessage(part);
		partsCount++;
		parts[partsCount]=part;
	}
	print.info("--- insertPartName END");
}

/**
* Checks if all the motor parts are inserted
**/
function isEngineMount(){
	return partsCount > 14;
}
```

El método *transparenceMaterial(opacity)* recorre el árbol de nodos de los que se compone la escena. En caso de pertenecer el material a una pieza no correspondiente con el motor se transparenta el material en función del parámetro dado.

```
/**
* Set the opacity field from all materials from a given value (0-100)
**/
function transparenceMaterial(opacity){
	print.info("---transparenceMaterial");
	var materialsNode = eon.FindNode("Materials");
	var childrenNodes = materialsNode.GetFieldByName("TreeChildren");
	var materialsNumber = childrenNodes.GetMFCount();
	var i=0;
	var material;
	var name
```

```
	print.debug("Hay un total de " + materialsNumber + " materiales.");
	for (i=0; i<materialsNumber; i++){
		material = childrenNodes.GetMFElement(i);
		name = eon.GetNodeName(material);
		if(name.indexOf("-Coche-")==0 || name.indexOf("-Interior-")==0){
			print.debug("Comprobando material " + name );
			if (material.GetFieldCount()==21) { //simplex material
				if (enableMaterialMod) material.GetFieldByName("Opacity").value=opacity/100;
				print.debug("material modificado!!");
			}
		}
	}
	print.info("---transparenceMaterial END");
}
```

Dado que en la escena se incluyen materiales con diferente grado de opacidad, el método *saveMaterialsOpacity()* almacena el nivel de opacidad de cada uno de los materiales de la escena que se van a modificar al lanzar la rutina de montaje del motor.

```
/**
* Save the opacity value for materials
**/
function saveMaterialsOpacity(){
	print.info("---saveMaterialsOpacity");
	var materialsNode = eon.FindNode("Materials");
	var childrenNodes = materialsNode.GetFieldByName("TreeChildren");
	var materialsNumber = childrenNodes.GetMFCount();
	print.debug("Hay un total de " + materialsNumber + " materiales.");
	var i=0;
	var material;
	var name;
	var opacity;
	for (i=0; i<materialsNumber; i++){
		material = childrenNodes.GetMFElement(i);
		name = eon.GetNodeName(material);
		print.debug("Probando a salvar el material " + name);
```

```
            if(name.indexOf("-Coche-")==0 || name.indexOf("-Interior-")==0){
                if (material.GetFieldCount()==21) { //simplex material
                    print.debug("Guardando el material " + name);
                    opacity = material.GetFieldByName("Opacity").value;
                    materialsOpacities[materialsOpacities.length]=name+"#"+opacity;
                }
            } else{
                print.debug("Material del motor");
            }
        }
        print.debug("Se han almacenado un total de " + materialsOpacities.length + " materiales.");
        print.info("---saveMaterialsOpacity END");
}
```

El método *restoreMaterialsOpacity()* recupera el listado de materiales almacenado por el método *saveMaterialsOpacity()* restaurando el nivel de opacidad de cada uno de los materiales para así recuperar el aspecto visual inicial de la escena.

```
/**
* Restore the opacity field from modfied materials
**/
function restoreMaterialsOpacity(){
        print.info("---restoreMaterialsOpacity");
        var i=0;
        var material;
        var name;
        var opacity;
        var entry;
        print.debug("materialsOpacities.length " + materialsOpacities.length);
        for (i=0; i<materialsOpacities.length; i++){
                entry = materialsOpacities[i];
                name = entry.substring(0, entry.indexOf("#"));
                opacity = entry.substring(entry.indexOf("#")+1,entry.lenght);
                print.debug("Restoring material " + name + " to " + opacity);
                material = eon.findNode(name);
```

```
                if(enableMaterialMod) material.GetFieldByName("Opacity").value = opacity;
        }
        print.info("---restoreMaterialsOpacity END");
}
```

El método *initMaterialsOpacity()* recorre el árbol de materiales de la escena y los restaura con la configuración inicial.

```
function initMaterialsOpacity(){
        print.info("---initMaterialsOpacity");
        var materialsNode = eon.FindNode("Materials");
        var childrenNodes = materialsNode.GetFieldByName("TreeChildren");
        var materialsNumber = childrenNodes.GetMFCount();
        print.debug("Hay un total de " + materialsNumber + " materiales.");
        var i=0;
        var material;
        var name;
        var opacity;
        for (i=0; i<materialsNumber; i++){
                material = childrenNodes.GetMFElement(i);
                name = eon.GetNodeName(material);
                if (material.GetFieldCount()==21) {
                        if (name=="-Coche-Ventana") {
                                material.GetFieldByName("Opacity").value = 0.5;
                        } else {
                                material.GetFieldByName("Opacity").value = 1;
                        }
          }
        }
        print.info("---initMaterialsOpacity END");
}
```

El método *getParts(nodeName)* obtiene dinámicamente el listado de objetos que se van a utilizar en la funcionalidad de montaje del motor. De cada uno de estos, almacena lso nombres para luego poder mostrarlos correctamente.

```
/**
*       Gets click nodes from Motor
**/
```

```
function getParts(nodeName){
	var node = eon.FindNode(nodeName);
	var children = node.GetFieldByName("TreeChildren");
	var childrenNode;
	if (children!=null && children.GetMFCount()>0){
		//has children
		for (var i=0; i<children.GetMFCount(); i++) {
			childrenNode = children.GetMFElement(i);
			getParts(eon.GetNodeName(childrenNode));
		}
	} else{
		if (nodeName.indexOf("Click")==0) {
			partsList[partsList.length]=nodeName;
		}
	}
}
```

El método *setClickParts()* bloquea la funcionalidad de click en las piezas que ya han sido montadas para así evitar posibles errores.

```
/**
*	Enable or disable click event on motor parts
**/
function setClickParts(status){
	var node;
	for (var i=0; i<partsList.length; i++){
		node = eon.findNode(partsList[i]);
		node.GetFieldByName("changeCursor").value=status;
		if(status){
			node.GetFieldByName("Button").value=0; //Left
		} else{
			node.GetFieldByName("Button").value=-1; //Disable
		}
	}
}
```

3. Cambio de color

Las funcionalidad de cambio de color se recoge en tres métodos; *On_NextColor()* y *On_PrevColor()* que se ejecutan automáticamente al pulsar sobre los botones de cambio de color para cambiar el color actual y el método *getColorMaterial()* el cual devuelve las propiedades que definen un color u otro.

```
/**************************************************************************
*****************************COLOR FUNCTIONS*****************************
**************************************************************************/
function On_NextColor(){
	print.info("--- On_NextColor");
	colorCounter++;
	var color = getColorMaterial(colorCounter % 6);
	print.debug("Color " + color);
	colorMaterial.GetFieldByName("Ambient").value=
		eon.MakeSFVec3f(color.ambient[0], color.ambient[1], color.ambient[2]);
	colorMaterial.GetFieldByName("Diffuse").value=
		eon.MakeSFVec3f(color.diffuse[0], color.diffuse[1], color.diffuse[2]);
	colorMaterial.GetFieldByName("Specular").value=
		eon.MakeSFVec3f(color.specular[0], color.specular[1], color.specular[2]);
	colorNode.GetFieldByName("TextColor").value=
		eon.MakeSFVec3f(color.txt[0], color.txt[1], color.txt[2]);
	colorNode.GetFieldByName("BoxColor").value=
		eon.MakeSFVec3f(color.bg[0], color.bg[1], color.bg[2]);
	print.info("--- On_NextColor END");
}
function On_PrevColor(){
	print.info("--- On_PrevColor");
	colorCounter--;
	var color = getColorMaterial(colorCounter % 6);
	print.debug("Color " + color);
	print.debug("color.ambient[0] " + color.ambient[0]);
	colorMaterial.GetFieldByName("Ambient").value=
		eon.MakeSFVec3f(color.ambient[0], color.ambient[1], color.ambient[2]);
	colorMaterial.GetFieldByName("Diffuse").value=
		eon.MakeSFVec3f(color.diffuse[0], color.diffuse[1], color.diffuse[2]);
```

```
        colorMaterial.GetFieldByName("Specular").value=
                eon.MakeSFVec3f(color.specular[0], color.specular[1], color.specular[2]);
        colorNode.GetFieldByName("TextColor").value=
                eon.MakeSFVec3f(color.txt[0], color.txt[1], color.txt[2]);
        colorNode.GetFieldByName("BoxColor").value=
                eon.MakeSFVec3f(color.bg[0], color.bg[1], color.bg[2]);
        print.info("--- On_PrevColor END");
}

function getColorMaterial(colorId){
        var returnColor = {}; //JSON OBJECT
        if (colorId==RED){
                print.debug("RED");
                returnColor.ambient=[0.816, 0.027, 0.027];
                returnColor.diffuse=[0.816, 0.027, 0.223];
                returnColor.specular=[0.74, 0.223, 0.223];
                returnColor.txt=[1,1,1];
                returnColor.bg=[0.816, 0.027, 0.027];
        } else if (colorId==BLACK){
                print.debug("BLACK");
                returnColor.ambient=[0.22, 0.22, 0.22];
                returnColor.diffuse=[0.212, 0.212, 0.212 ];
                returnColor.specular=[0.749, 0.749, 0.749 ];
                returnColor.txt=[1,1,1];
                returnColor.bg=[0.212, 0.212, 0.212 ];
        } else if (colorId==GOLD){
                print.debug("GOLD");
                returnColor.ambient=[0.929, 0.733, 0.204];
                returnColor.diffuse=[0.929, 0.733, 0.204];
                returnColor.specular=[0.705, 0.25, 0];
                returnColor.txt=[0,0,0];
                returnColor.bg=[0.929, 0.733, 0.204];
        } else if (colorId==BLUE){
                print.debug("BLUE");
                returnColor.ambient=[0.133, 0.133, 0.651];
                returnColor.diffuse=[0.133, 0.133, 0.651];
                returnColor.specular=[0.203, 0.203, 0.638];
```

```
            returnColor.txt=[1, 1, 1];
            returnColor.bg=[0.133, 0.133, 0.651];
    } else if (colorId==WHITE){
            print.debug("WHITE");
            returnColor.ambient=[0.910, 0.910, 0.910];
            returnColor.diffuse=[0.910, 0.910, 0.910];
            returnColor.specular=[0.590, 0.590, 0.590];
            returnColor.txt=[0,0,0];
            returnColor.bg=[0.910, 0.910, 0.910];
    } else if (colorId==GREEN){
            print.debug("GREEN");
            returnColor.ambient=[0.263, 0.859, 0.027];
            returnColor.diffuse=[0.263, 0.859, 0.027];
            returnColor.specular=[0.223, 0.705, 0.223];
            returnColor.txt=[1,1,1];
            returnColor.bg=[0.263, 0.859, 0.027];
    }
    return returnColor;
}
```

4. Logs y mensajería

En esta parte se definen los métodos para mostrar un mensaje tanto por consola de EON como por el cuadro de mensajes de la aplicación. El primero de ellos esta destinado a la traza y depuración de la aplicación. Las funciones *showMessage(msg)* y *resetMessage()* muestran el mensaje pertinente u oculta el panel de mensajes. Este panel es el utilizado para mostrar el nombre de la pieza del motor que se esté montando.

```
/**********************************************************************
*****************************UTILS FUNCTIONS*****************************
**********************************************************************/
/**
* Logger object
**/
var print = {
```

```
        info : function(printText) {
                if (traceActive && traceMode>=INFO){
                        eon.trace("INFO  > " + printText);
                }
        },
        debug : function(printText ) {
                if (traceActive && traceMode==DEBUG){
                        eon.trace("DEBUG  > " + printText);
                }
        },
        error : function(printText) {
                if (traceActive){
                        eon.trace("ERROR  > " + printText);
                }
        }
};
/**
* Show message in the viewport messages board
**/
function showMessage(msg) {
        print.info("---showMessage");
        msgNode.GetFieldByName("Text").value=msg;
        msgNode.GetFieldByName("IsActive").value=1;
        print.info("---showMessage END");
}
/**
* Reset viewport messages board
**/
function resetMessage() {
        print.info("---resetMessage");
        msgNode.GetFieldByName("Text").value="";
        msgNode.GetFieldByName("IsActive").value=0;
        print.info("---resetMessage END");
}
```