UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

AMBIENTE VIRTUAL PARA UBICACIÓN DE AULAS

TESIS

PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTAN

MIGUEL ANGEL RAMÍREZ LUNA

RODRIGO FAVILA ZAVALZA

DIRECTOR: ING. LUIS SERGIO VALENCIA CASTRO

CIUDAD UNIVERSITARIA, MÉXICO D.F. A 22 DE ABRIL DE 2013

Dedicatorias

A mis padres Miguel y María, por su constante e incondicional apoyo que me ha permitido hoy alcanzar esta meta. Por ser mis primeros y mejores maestros quienes no solo con palabras, sino mediante el ejemplo, me han formado en la persona que soy el día de hoy. Todo lo bueno en mi es gracias a ustedes.

A mi hermana Carmen, quien siempre ha creído en mí en los momentos más difíciles, por todos sus buenos consejos y la gran motivación que me ha dado para superarme día a día.

A mi tía Beatriz, por todo el cariño y apoyo que me ha brindado durante toda mi vida y quien es otra madre para mí.

A todos ustedes dedico este trabajo. No podría imaginar mejores personas en mi vida

Miguel Angel Ramírez Luna

Agradecimientos

Al M.I. Rodrigo Guillermo Tintor Pérez, por darnos la oportunidad de desarrollar nuestras habilidades en Computación Grafica junto a su equipo de trabajo en el Palacio de Minería.

Al Ing. Luis Sergio Valencia Castro, por su interés en este proyecto y por su ayuda en cubrir la dirección de esta Tesis.

A Catalina Ferat Toscano, por orientarnos en la redacción de este trabajo.

A Hugo Aguilera Marín, por su colaboración en el diseño y la elaboración de las imágenes de referencia del personaje.

A nuestros sinodales, por su interés en el proyecto y el tiempo dedicado a la revisión de este trabajo.

Contenido

Tabla de Figuras

1. Introducción

El ser humano se ha dado a la tarea de tratar de reproducir la realidad desde tiempos remotos, podemos tomar como evidencia de esto las pinturas rupestres encontradas en diversas cuevas a lo largo del planeta. Los avances tecnológicos han permitido al humano representar la realidad de forma cada vez más fiel; lo que antes eran dibujos plasmados en roca han llegado a transformarse hoy en ambientes virtuales creados en una computadora, con los cuales se nos permite no solo observarlos, también se nos permite interactuar con ellos de diferentes maneras.

Los ambientes virtuales son algo cada vez más común en nuestro tiempo y tienen un número diverso de utilidades. Gracias a estos las personas pueden conocer diversos lugares del planeta sin la necesidad de viajar físicamente a dicho lugar, también pueden visitar mundos salidos de la imaginación de otras personas, sirven para desarrollar habilidades que podemos extrapolar al mundo real (entrenamiento en simulaciones), inclusive pueden ayudar en el estudio y tratamiento de diferentes afecciones; como en el caso del tratamiento de las fobias, ya que al sentirse la persona en un lugar seguro que puede controlar, es más fácil que enfrente sus miedos y logre superarlos, todo esto entre muchas otras aplicaciones.

Con esto se puede observar que la utilidad de los ambientes virtuales es muy amplia, y el campo de desarrollo de estos no está limitado a un sector en específico. Lo cual está provocando una demanda que crece cada día más, y con el surgimiento de nuevas tecnologías, así como el refinamiento de las ya existentes, se espera poder lograr un nivel de realismo tan preciso, que seamos incapaces de distinguir el ambiente virtual, de un ambiente real.

1.1 Definición del Problema

Durante el periodo en que realizamos nuestros estudios en la Facultad de Ingeniería se observaron los inconvenientes que se generan cuando los alumnos deben localizar algún salón de clases o laboratorio. Estos inconvenientes son resultado de la falta de información que hay acerca de la localización de los salones y laboratorios de la facultad. Si bien, hay salones y laboratorios cuya ubicación es señalada de forma general en los mapas que se encuentran en el sistema de inscripciones, estos mapas carecen de información específica acerca de la correcta ubicación de todos los salones y laboratorios de la Facultad de Ingeniería.

Teniendo esta problemática en mente, se empieza a gestar la idea de buscar un método mediante el cual los alumnos obtengan la información precisa de la ubicación de los salones y extender esta información no sólo a alumnos, también al personal de la facultad y a gente ajena a la institución. De esta forma se decide crear un ambiente virtual en el cual, el usuario, pueda navegar libremente a través de la institución.

1.2. Objetivo

La finalidad de éste trabajo es reproducir fielmente el edificio principal de la Facultad de Ingeniería, para elaborar un ambiente virtual, en el que las personas puedan localizar los salones y laboratorios de forma precisa; esto se lograra implementando un sistema que permita ingresar el nombre del salón o laboratorio buscado, y regrese la información con la cual la persona pueda navegar con el avatar en el ambiente virtual desde un punto cualquiera, hasta el salón seleccionado.

1.3. Motivación

La idea de realizar este proyecto surgió de la experiencia personal que se tuvo al momento de ser alumno ya fuera de nuevo ingreso, o simplemente se cursara un nuevo semestre, y las dificultades que se tenían para localizar los salones, especialmente laboratorios. En el caso de los laboratorios cuando uno imprimía sus horarios, el lugar donde iba a tomar uno los laboratorios solo aparecía marcado como Lab. Pero no se indicaba en qué lugar, o cual era el nombre de ese laboratorio en específico, así que la solución era recorrer todo el edificio de la facultad o preguntarle a alguien y esperar que supiera donde se encontraba dicho laboratorio. Se decidió realizar la búsqueda de salones, porque también nos encontramos con varios casos en los que algún alumno nos llegó a preguntar en donde se encontraba algún salón en específico, y con el antiguo sistema de numeración de salones ocurría que por ejemplo el salón 108 podía encontrarse en un edificio, pero el salón 208 que debería ser simplemente en el piso siguiente se podía encontrar en otro edificio.

1.4. Contribución

Con este proyecto se busca reducir los problemas y las pérdidas de tiempo al buscar los salones, empezando dicho proyecto con el edificio principal, y esperamos que al ver esta iniciativa, otros alumnos se aventuren a realizar la misma actividad de este proyecto pero que se extienda al anexo de la facultad, al edificio de posgrados; e incluso se pueda llevar esta idea a otras facultades y eventualmente contemos con toda Ciudad Universitaria de forma virtual.

1.5 Resumen de Capítulos

Capítulo 2. Realidad Virtual: Campos en los que se puede aplicar la realidad virtual, evolución desde una forma de entretenimiento hasta su uso para el entrenamiento de las personas en diferentes disciplinas, como el entrenamiento que reciben los pilotos de aviadores o astronautas.

Capítulo 3.Modelado: Técnicas de modelado, características, ventajas y desventajas. Texturizado, tipos de texturas, mapas y canales. Iluminación, tipos y lightmaps. Animación de personajes, animación esquelética y keyframes.

Capítulo 4. Motores de Juego: Tipos de motores de juego, ventajas y desventajas. Motores de propósito general y propósito específico. Ejemplos de motores de juego y los diferentes juegos o aplicaciones construidas con cada uno de ellos.

Capítulo 5. Desarrollo de la aplicación: Establecimiento de una metodología de software. Fase de modelado, explicación de las técnicas que se usaron para cada sección de la facultad, tipos de mapas que se usaron para el texturizado. Fase de programación, creación de scripts en C# desarrollo del sistema de cámara en tercera persona, menú principal, menú de pausa y de búsqueda. Integración con el sistema de Pathfinding de Unity. Validación de la aplicación, desarrollo de las soluciones a los diferentes problemas.

Capítulo 6. Conclusiones: Recapitulación de los objetivos iniciales y comparación con los resultados obtenidos al término del desarrollo. Posibilidades de expansión de la aplicación hacia otros edificios de la Facultad de Ingeniería u otras facultades.

Capítulo 7. Bibliografía. Bibliografía y mesografía.

Capítulo 8. Apéndice: Muestras del código utilizado, manual de usuario de la aplicación, tabla de figuras.

2. Realidad Virtual

La realidad virtual se ha vuelto un concepto con el cual muchas personas están familiarizadas, si bien puede que no entiendan la definición del concepto como tal, si tienen una comprensión empírica de este concepto, esto se debe a la exposición que ha tenido en medios masivos, entre ellos uno de los más importantes el internet, y que de una u otra forma ha permitido que la gente esté en contacto directo con productos como videojuegos, simuladores, recorridos virtuales inmersivos y no inmersivos, por mencionar algunos cuantos ejemplos.

En este capítulo se empezará por definir el concepto de realidad virtual, se verán cuáles son los tipos en los que se divide la realidad virtual, así como un poco de su historia a través de los años y algunas aplicaciones actuales que están ayudando a cambiar la forma en como vemos al mundo y a nosotros mismos.

2.1. Definición

Para definir de forma correcta el concepto de realidad virtual, primero se tiene que analizar la definición de cada una de sus palabras, “realidad” es todo lo que constituye el mundo real y “virtual”, lo que tiene existencia aparente y no física. De éste análisis se podría definir la realidad virtual como una existencia aparente de todo lo que constituye el mundo real. Aunque esta definición un poco burda nos da una idea de lo que es la realidad virtual, no nos da idea de su potencial, ya que en la realidad virtual no solo se puede reproducir la realidad, también se pueden reproducir cosas que solo existen en la imaginación de las personas.

Aunque ya se tiene una definición sencilla de realidad virtual, falta mencionar la parte que involucra a las computadoras, ya que realidad virtual es un término relativamente moderno, el cual surgió a partir del desarrollo de la computación. Las computadoras permitieron crear simulaciones, al principio primitivas, hechas solo con vectores, y con el tiempo, cada vez más avanzadas del entorno que nos rodea. El avance de la tecnología no solo ha permitido que las simulaciones sean más realistas, también han permitido que se experimenten de una manera totalmente diferente, esto gracias a los diferentes periféricos que se han desarrollado; los cuales logran sumergirnos en estas simulaciones de tal manera que pareciera que nos encontramos dentro de estos mundos, al permitirnos controlarlos con el simple movimiento de nuestras manos, en lugar de usar teclados, mouse, u otros métodos tradicionales, nosotros nos convertimos en el propio control, con lo cual es más fácil creer que nos encontramos dentro de esta simulación,

permitiendo que se convierta en nuestra realidad, durante el tiempo en el que permanezcamos dentro de ella.

Como la realidad virtual es un concepto relativamente nuevo, no hay por el momento una definición única que sea aceptada como correcta. Sin embargo, esto no quiere decir que las definiciones que se han propuesto sean incorrectas, simplemente nos dice que, no hay una definición estándar que podamos tomar de cualquier libro y con ella digamos “ésta es la definición universal de realidad virtual”, por lo cual, lo mejor que se puede hacer es elaborar nuestra propia definición, tratando de que abarque todo el concepto de realidad virtual, de forma sencilla y concreta. Tomando esto en cuenta, y basándonos en lo que se dijo anteriormente, se puede elaborar nuestra definición de realidad virtual como: “La simulación de medios ambientes, tanto reales como imaginarios, y de los mecanismos sensoriales del hombre por computadora, de tal manera que se busca proporcionar al usuario la sensación de inmersión y la capacidad de interacción con medios ambientes artificiales, y de acuerdo al grado de inmersión esta se puede clasificar como inmersiva y no inmersiva”.

Ahora que ya tenemos una definición de realidad virtual y por lo tanto conocemos el concepto, se hablará de su historia, lo cual permitirá ver la evolución que ha tenido en un periodo relativamente corto, y nos dará una idea de lo que podemos lograr en la actualidad y lo que podremos lograr en el futuro.

2.2 Historia

El inicio de la realidad virtual en la historia es algo ambiguo, depende mucho de que conceptos se tomen en cuenta para marcarlos como antecesores de lo que es la realidad virtual actualmente. Tomando esto en cuenta podemos tomar como inicio la creación del primer estereoscopio, el cual fue realizado por Charles Wheatstone, un inventor británico, el estereoscopio consistía en una especie de gafas en las que se situaban 2 fotografías distintas en cada ojo, creando de este modo una imagen 3d en el interior del cerebro, lo cual permitía experimentar una sensación de profundidad. De esta forma daba la sensación a las personas de estar en un lugar diferente al lugar donde se encontraban aunque fuera solo de forma visual, y claro, usando un poco la imaginación. De ahí pasamos hasta el año de 1930, año en el que se empezó a usar el Link Trainer (Ej. Fig. 1.1), aunque fue desarrollado en 1929 fue hasta 1930 cuando se implementó de forma masiva. El Link Trainer fue un simulador de vuelo usado durante la segunda guerra mundial, este funcionaba mediante válvulas y bombas mecánicos, y permitía que el simulador respondiera al control ejercido por el piloto y proporcionaba una lectura apropiada de los instrumentos que incluía el

simulador, de esta manera el piloto aprendía a leer de forma correcta los instrumentos, lo que evitó más fatalidades en los pilotos que tenían que depender de los instrumentos para volar en situaciones de nula visibilidad. En los años 30's en Estados Unidos crearon simuladores mecánicos para estudiar las crecidas de los ríos y las presas.

Fig. 1.1. Piloto a bordo de un Link Trainer. Con este simulador de vuelo. Se entrenó a cientos de pilotos durante la guerra.

A pesar de que estos dos antecedentes no tenían nada que ver con las computadoras, su objetivo era representar ya sea un lugar mediante las imágenes tridimensionales, u obtener la respuesta que tendría un objeto en el mundo real, para así poder predecir su comportamiento. Hasta el año de 1946 vendría la era de las computadoras, con la creación de ENIAC, la cual fue la primera computadora electrónica de propósito general, se menciona la parte electrónica porque en esa misma época surgieron algunas computadoras anteriores a ENIAC pero el funcionamiento de éstas era mecánico. Con ENIAC se realizan las primeras simulaciones de las trayectorias de misiles, así como las simulaciones de las explosiones durante el proyecto Manhattan. En 1956 Morton Heilig comienza el diseño de la primera experiencia virtual multisensorial, asemejándose a una de las máquinas de arcade actuales el Sensorama combinaba una proyección de películas, audio, vibración, viento y olores, todos diseñados para hacer sentir al usuario como si de verdad estuviera en la película en lugar de solo estarla observando. El Sensorama (Ver Fig. 1.2) se patento en 1961, y este colocaba al espectador en un cine para una persona y permitía al usuario experimentar una de cinco películas de duración de dos minutos en 3D a color con el complemento de sensación de movimiento, sonido, viento y olores. Las cinco diferentes experiencias incluían, un viaje en motocicleta a través de Nueva York, un viaje en bicicleta, un viaje en un buggy a través de las dunas, un viaje en helicóptero sobre la ciudad en 1960 y un baile por una bailarina de danza árabe.

Heilig también patento una idea para un mecanismo que algunos consideran el primer Head-Mounted Display (HMD) (Ver Fig. 1.3). El HMD es un dispositivo de visualización similar a un casco, que permite reproducir imágenes creadas por una computadora sobre un "display" muy cercano a

los ojos o directamente sobre la retina de los ojos. En este segundo caso el HMD recibe el nombre de monitor virtual de retina. También propuso la idea para un teatro inmersivo que permitiría la proyección de imágenes tridimensionales sin que el espectador tuviera que usar lentes especiales o cualquier otro dispositivo, la audiencia estaría sentada en gradas, y los asientos estarían conectados con la pista de la película, para proveer no solo sonido estero pero también la sensación de movimiento; los olores serian creados inyectando diferentes tipos de aromas en el sistema de aire acondicionado. Desafortunadamente este mecanismo nunca fue construido.

Fig. 1.2. El Sensorama de Heilig.

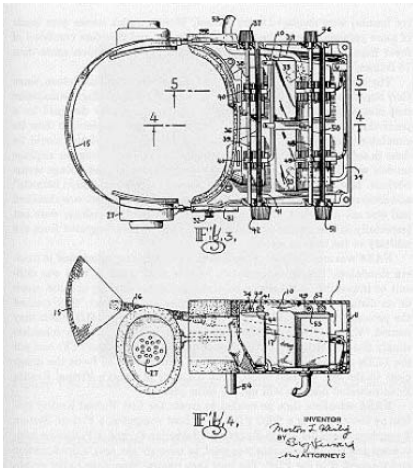


Fig. 1.3. La patente del Head Mounted Display de Heilig.

A pesar de que Heilig patento el mecanismo de lo que se considera el primer Head-Mounted Display, Comeau y Bryan empleados de Philco Corporation, construyeron el primer Head-Mounted Display en 1961. Este mecanismo llamado Headsight contenía un único elemento CRT unido al casco y un sistema de localización magnética para determinar la posición en que se encontraba la cabeza. El HMD fue diseñado para ser usado con un circuito cerrado de video a control remoto, para observar situaciones peligrosas de forma remota. A pesar de que en esta época ya se contaba con algunos de los mecanismos que utiliza la realidad virtual, faltaba la integración de la computadora y de las imágenes generadas por computadora.

El campo conocido como realidad virtual, surgió de la investigación en graficas interactivas tridimensionales y la simulación de los vehículos en los años de 1960 y 1970. El desarrollo de esta disciplina se puede rastrear a los trabajos de Ivan Sutherland y Larry Roberts. Las primeras contribuciones practicas incluyeron la investigación y el desarrollo que permitió que el CRT (tubo de rayos catódicos) sirviera como un dispositivo efectivo y costeable en el cual se crearan imágenes generadas por computadora, y las interfaces interactivas que mostraron que un usuario podía interactuar con la imagen del CRT para completar alguna tarea deseada. Roberts, también

escribió el primer algoritmo para eliminar superficies ocultas u obscurecidas de una imagen en perspectiva en 1963. Sus soluciones para este y otros problemas similares generaron intentos durante la siguiente década para desarrollar algoritmos más rápidos para generar superficies ocultas. Entre las actividades de Sutherland y sus colegas en la Universidad de Utah se incluían esfuerzos para desarrollar algoritmos de mayor rapidez para remover las superficies ocultas de los gráficos en 3D, un problema identificado como un punto clave en el cuello de botella computacional generado.

En el año de 1965 Ivan Sutherland describió el concepto de realidad virtual en un artículo titulado "The Ultimate Display", en este artículo escribió:

- El ultimate display seria, un cuarto en el cual la computadora pueda controlar la existencia de la materia. Una silla mostrada en dicho cuarto podría ser lo suficientemente real como para sentarse en ella. Unas esposas mostradas en dicho cuarto proporcionarían confinamiento, y una bala mostrada en dicho cuarto sería fatal. Con la programación apropiada dicho cuarto podría ser literalmente el país de las maravillas en el cual Alicia caminó

A pesar de ésta descripción, en el artículo no se llegó a usar el término realidad virtual como tal.

Los estudiantes del programa Utah hicieron dos importantes contribuciones a este campo, incluyendo un método de búsqueda de áreas por Warnock (1969) y un algoritmo scan-line que fue desarrollado por Watkins (1970) y el cual se desarrolló en un sistema de hardware. Uno de los avances más importantes fue el desarrollo de Henri Gouraud de un simple esquema para un shading (sombreado) continuo (1971). A diferencia del shading poligonal, en el que el polígono entero (una representación estándar de una superficie) era de una sola tonalidad de gris, el esquema de Gouraud involucra la interpolación entre los puntos de una superficie para describir un shading continuo a través de un solo polígono, de esta forma logrando una mejor aproximación a la realidad. Este efecto hace que una superficie compuesta por polígonos discretos, parezca continua. Esta habilidad es esencial para generar la calidad de imágenes visuales necesarias para presentar un ambiente virtual creíble.

Durante los años 80´s surgieron otros dos proyectos que cabe mencionar. Andy Lippman y un grupo de investigadores del MIT desarrollaron lo que se puede considerar el primer sistema hipermedia. El Aspen Movie Map (Fig. 1.4), una aplicación que permitía al usuario disfrutar de un viaje simulado a través de la ciudad de Aspen, Colorado. Este sistema utilizaba un juego de discos de video que contenían fotografías de todas las calles de Aspen. Esta grabación se realizó con cuatro cámaras, cada una apuntando en diferente dirección y colocada en una camioneta, lo podríamos considerar el antecesor del google streetview, las fotografías se tomaban cada 3

metros. El usuario podía navegar avanzando ya sea a la izquierda, a la derecha, avanzar o retroceder. Cada foto estaba enlazada a otra para soportar estos movimientos. En teoría el sistema podía desplegar 30 imágenes por segundo, simulando una velocidad de 330 km/h. El sistema fue alentado a 10 imágenes por segundo, el equivalente a 110 km/h. Para hacer más inmersivo este programa, el usuario podía detenerse en frente de algunos de los edificios principales de Aspen y entrar en ellos. El sistema usaba dos pantallas una vertical para el video y una horizontal para mostrar el mapa de Aspen, el usuario podía seleccionar un punto en el mapa y saltar directo a él, en lugar de recorrer la ciudad para encontrarlo.

Fig. 1.4. Una imagen del Aspen Movie Map. Podemos notar su gran parecido con su predecesor Google Streetview.

El otro logro que cabe destacar fue el llamado VIDEOPLACE (Ver Figs. 1.5.1 y 1.5.2). Creado por Myron Krueger, en este sistema la computadora tenía el control sobre la relación entre la imagen del participante y los objetos en la escena desplegada, y podía coordinar el movimiento de un objeto gráfico con las acciones del participante. Mientras que la gravedad afectaba el cuerpo físico, no controlaba o confinaba la imagen, la cual podía flotar si era necesario. Una seria de simulaciones podía ser programada, basadas en cualquier acción. El VIDEOPLACE ofrecía más de 50 composiciones e interacciones incluyendo animales, fractales, pintura con los dedos, dibujo digital, repeticiones entre otros.

En la instalación del programa, el participante se colocaba frente a una pantalla de proyección de video mientras que la pantalla detrás de él era iluminada para producir imágenes de alto contraste para la cámara colocada enfrente de la pantalla de proyección, permitiéndole a la computadora distinguir al participante, del fondo. La imagen del participante era digitalizada para crear siluetas, las cuales eran analizadas por procesadores especializados. Los procesadores podían analizar la postura de la imagen, el rango de movimiento y su relación con los demás objetos gráficos en el

sistema. Estos podían reaccionar al movimiento del participante y crear una serie de respuestas, ya fueran visuales o auditivas, también se podían enlazar dos o más ambientes.

Fig. 1.5.1. Arreglo de cámaras del VIDEOPLACE.

Fig. 1.5.2. VIDEOPLACE en acción.

En 1983 se crea el primer instrumento que permite medir la posición de las manos, desarrollado por el Dr. Gary Grimes de los laboratorios Bell. Este instrumento contenía sensores para medir la flexión de los dedos, sensores táctiles en las puntas de los dedos, sensores de orientación y sensores para determinar la posición de la muñeca. Las posiciones de los sensores como tal, eran intercambiables. El propósito de éste era crear caracteres alfanuméricos de acuerdo a la posición de las manos. Su diseño principal fue como una alternativa al teclado, pero mostro su efectividad permitiendo a los usuarios incapaces de hablar, deletrear con los dedos las palabras usando dicho sistema (Fig. 1.6).

A este guante le siguió un guante óptico, este guante se llamó DataGlove (Ver Fig. 1.7.1). Este guante era de neopreno y contenía dos bucles de fibra óptica en cada dedo. Cada bucle estaba diseñado para un nudillo, lo cual ocasionalmente causaba problemas, si el usuario tenia manos grandes o muy pequeñas los bucles no embonaban de forma correcta con la posición verdadera de los nudillos y el usuario perdía la habilidad de producir gestos adecuados. Al final de cada bucle había un LED y al lado contrario había un foto sensor. El cable de fibra óptica tenía pequeños cortes en toda su longitud, cuando el usuario doblaba el dedo, había una fuga de luz del cable de fibra óptica. La cantidad de luz que alcanzaba al foto sensor era medida y convertida en una medida que determinaba cuanto se había flexionado el dedo. Este guante permitió que el desplazamiento en los mundos virtuales fuera realizado a través de las manos, sumergiendo más al usuario en este mundo virtual (Fig. 1.7.2).

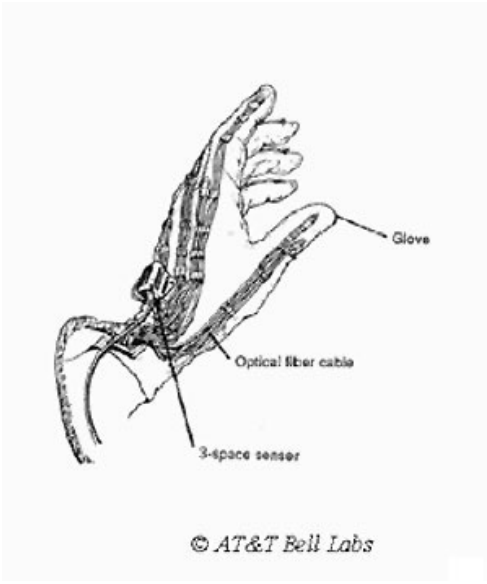


Fig. 1.6. Guante Digital de Grimes.

Fig. 1.7.1. Imagen del DataGlove.

Fig. 1.7.2. DataGlove siendo usado en conjunto con un HMD, permitiendo una inmersión mayor en los ambientes virtuales.

En 1983 Krueger publica su ahora famoso libro *Aritificial Reality,* realidad artificial, en donde uso este término para describir sus ambientes inmersivos e interactivos, basado en técnicas de reconocimiento de video, que colocan al usuario en un contacto completo y sin trabas con el mundo digital. En el lenguaje moderno, el término realidad artificial, se usa normalmente para describir una ambiente virtual que es indistinguible de la realidad. Esto en contraste con el término realidad virtual, el cual se aplica a la tecnología que emula la realidad pero que puede ser fácilmente reconocido como una simulación.

En 1984 William Gibson publica su novela llamada *Neuromancer* donde se utiliza el término ciberespacio, en el cual hace referencia a un mundo virtual. El termino ciberespacio lo acuño en su

novela corta *Burning Chrome*, y su uso a través de la novela de *Neuromancer* provoco que el término ganara reconocimiento, convirtiéndose en el término para la World Wide Web (Internet). En 1992 se estrena la película *The Lawnmower Man* (El cortador de césped) En la cual se hace uso del concepto de realidad virtual y de ambientes virtuales, ayudando a popularizar la realidad virtual.

En 1994 se presenta el VRML un formato estándar para representar graficas vectoriales en 3D interactivas, diseñadas teniendo en menta particularmente el internet. VRML por sus siglas en inglés que significan Lenguaje para modelado de realidad virtual. En 1997 se presentó un nuevo formato actualizado conocido como VRML97 o VRML 2.0 y fue reconocido por ISO como estándar. VRML generó mucho interés a su alrededor, pero nunca se vio un uso más amplio del mismo. Una de las razones para esto puede haber sido la falta de un buen ancho de banda. En la época de la popularidad del VRML, la mayoría de los usuarios, tanto negocios como personales, estaban usando conexiones de dial-up para tener acceso a internet. Esto provocaba que no se pudiera ver de forma correcta lo que se hiciera con el VRML debido a los largos tiempos de espera de ese tipo de conexiones y la baja velocidad que se manejaba.

En los años 70´s, 80´s y 90´s se dio el surgimiento y la popularización de las consolas de videojuegos así como el uso de las PC´s como entretenimiento. En los 70´s surge la Magnavox Odyssey, creada por Ralph Baer, considerado el padre de los videojuegos. Surgieron en esas épocas, el Atari, el Nintendo Entertainment System (NES) entre otras. Gracias a los videojuegos se popularizo más el concepto de realidad virtual, ya que estos nos presentaban mundos virtuales los cuales podíamos recorrer a nuestro gusto, a pesar de que se puede considerar realidad virtual, les faltaba el grado de inmersión necesario para que fueran considerados una experiencia de realidad virtual. No fue sino hasta el año de 1995 con el surgimiento del Virtual boy, cuando se tuvo una consola que permitiera al usuario experimentar la realidad virtual, por desgracia dicha consola no tuvo éxito, debido a problemas en su diseño y al costo de la misma. Seria hasta tiempos recientes cuando volveríamos a ver nuevos intentos por traer la realidad virtual a las consolas y al mercado en general. Con el desarrollo del Nintendo Wii, del PS Move y el Eye Toy de Sony y del Kinect por parte de Microsoft, han aparecido nuevas herramientas para tratar de llevar la realidad virtual al público en general (Ej. Fig. 1.8).

Fig. 1.8. De izquierda a derecha en sentido de las manecillas del reloj: Eyepet de Sony, Kinectimals de Microsoft y el Wii de Nintendo. La interacción con ambientes virtuales se ha vuelto más popular gracias a estos acercamientos.

2.3 Tipos

La realidad virtual tiene diferentes clasificaciones, algunos hacen mención de solo dos de éstas y algunos otros hacen mención de una tercera. Los tipos de realidad virtual se clasifican tomando en cuenta los diferentes grados de inmersión que se pueden obtener mediante el uso de diferentes dispositivos, nosotros vamos a mencionar los tres diferentes tipos. De acuerdo a esta clasificación los tipos son:

- Realidad virtual no Inmersiva
- Realidad virtual semi-Inmersiva
- Realidad virtual inmersiva

A continuación se explicaran las diferencias que hay en estas clasificaciones, mostrando algunos ejemplos de las mismas

2.3.1 Realidad virtual no Inmersiva

Los sistemas no inmersivos (también llamados sistemas de escritorio) son aquellos en los que el usuario solo tiene la imagen desplegada para acceder al mundo virtual que tiene frente a él. De esta manera, el monitor (o cualquiera que sea el método de proyección) es la ventana hacia el mundo virtual. La interacción con el mundo virtual se realiza mediante diversos periféricos como son el mouse, el teclado, el joystick, etc. Este tipo de sistemas son muy útiles para el público en general, para las universidades, laboratorios de investigación y para los medios de entretenimiento, por mencionar algunos [9]. Aunque no sean los más idóneos y no ofrezcan una inmersión total, son una excelente alternativa debido a su bajo costo, tanto del producto, como del mantenimiento.

Algunos ejemplos de estos sistemas son: las simulaciones por computadora, las cuales nos presentan un ambiente virtual que podemos controlar para obtener predicciones del comportamiento de los objetos en el mundo real (Fig. 2.1); los recorridos virtuales, si bien son generados por computadora, el objetivo de estos es diferente al de las simulaciones, ya que en el recorrido, tenemos un mundo creado en la computadora, el cual solo nos dedicamos a recorrerlo, ya sea para obtener localizaciones de diferentes lugares de ese mundo, o para conocer el aspecto de estos lugares (Fig. 2.2). Los videojuegos, en los cuales podemos recorrer mundos virtuales, ya sean copias de lugares reales, o mundos salidos de la imaginación de otras personas (Fig. 2.3).

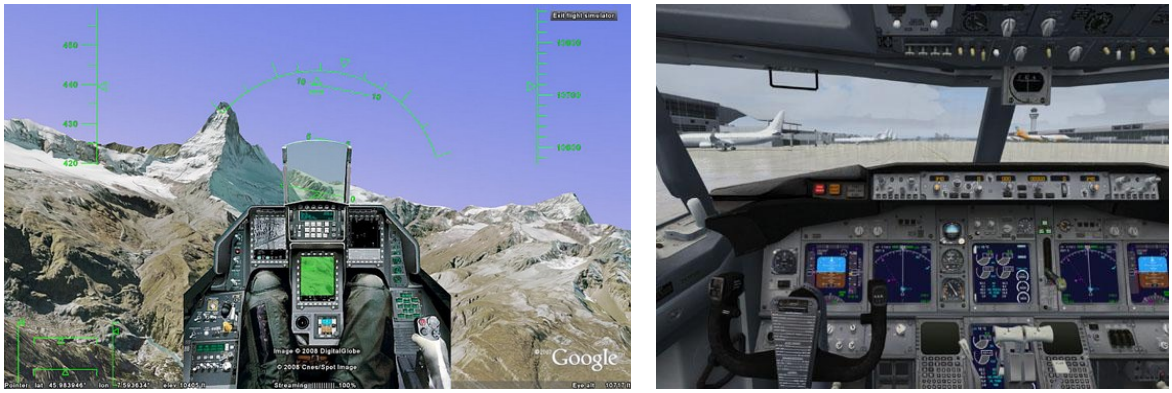


Fig. 2.1. Simuladores de vuelo

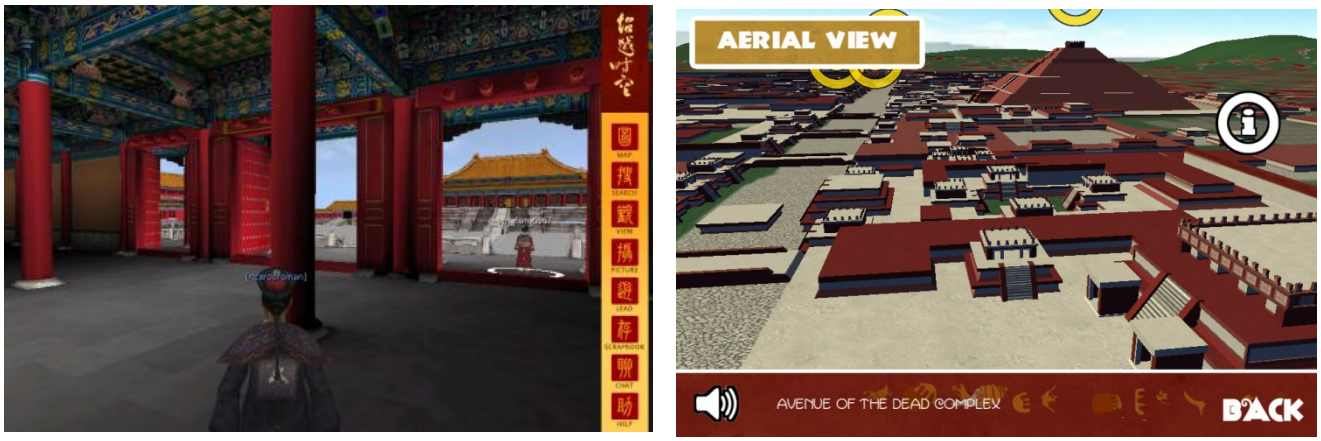


Fig. 2.2. Recorridos virtuales. Izquierda: Ciudad prohibida de china, derecha: Teotihuacan.

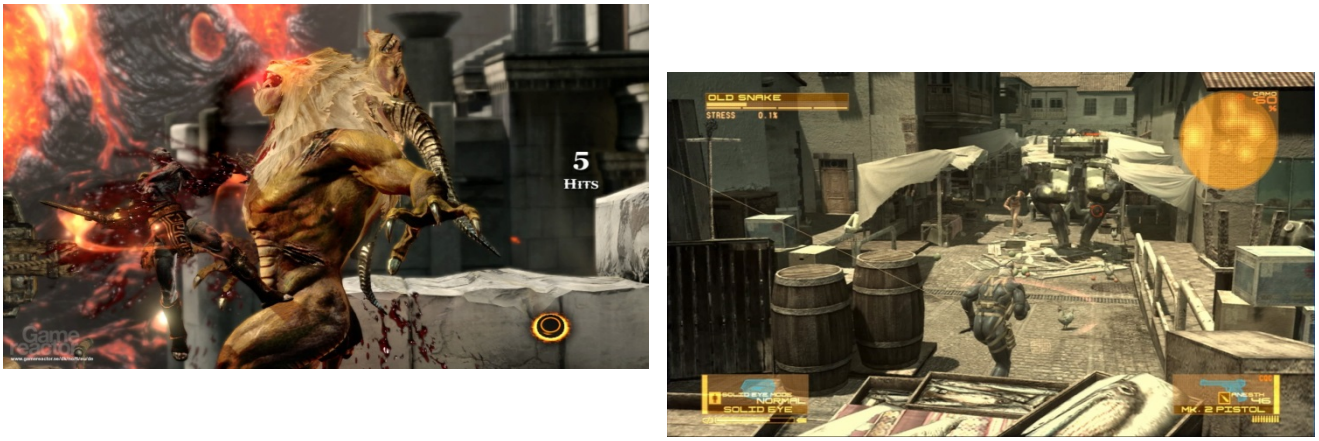


Fig. 2.3. Videojuegos. Izquierda God of War 3, derecha: Metal Gear Solid 4

2.3.2 Realidad virtual Semi-Inmersiva

Los sistemas semi-inmersivos (también llamados inmersivos de proyección), se caracterizan por tener 4 pantallas en forma de cubo (tres pantallas forman las paredes y una el piso), las cuales rodean al observador, el usuario usa lentes y un dispositivo de seguimiento de movimientos de la cabeza, de esta manera cuando el usuario se mueve, las proyecciones de las perspectivas son calculadas por el motor de realidad virtual para cada pared y se despliegan en proyectores que están conectados a la computadora (Fig. 3). Este tipo de sistemas son usados principalmente para visualizaciones donde se requiere que el usuario se mantenga en contacto con diversos elementos del mundo real, pero que tenga una buena inmersión en el ambiente virtual [9].

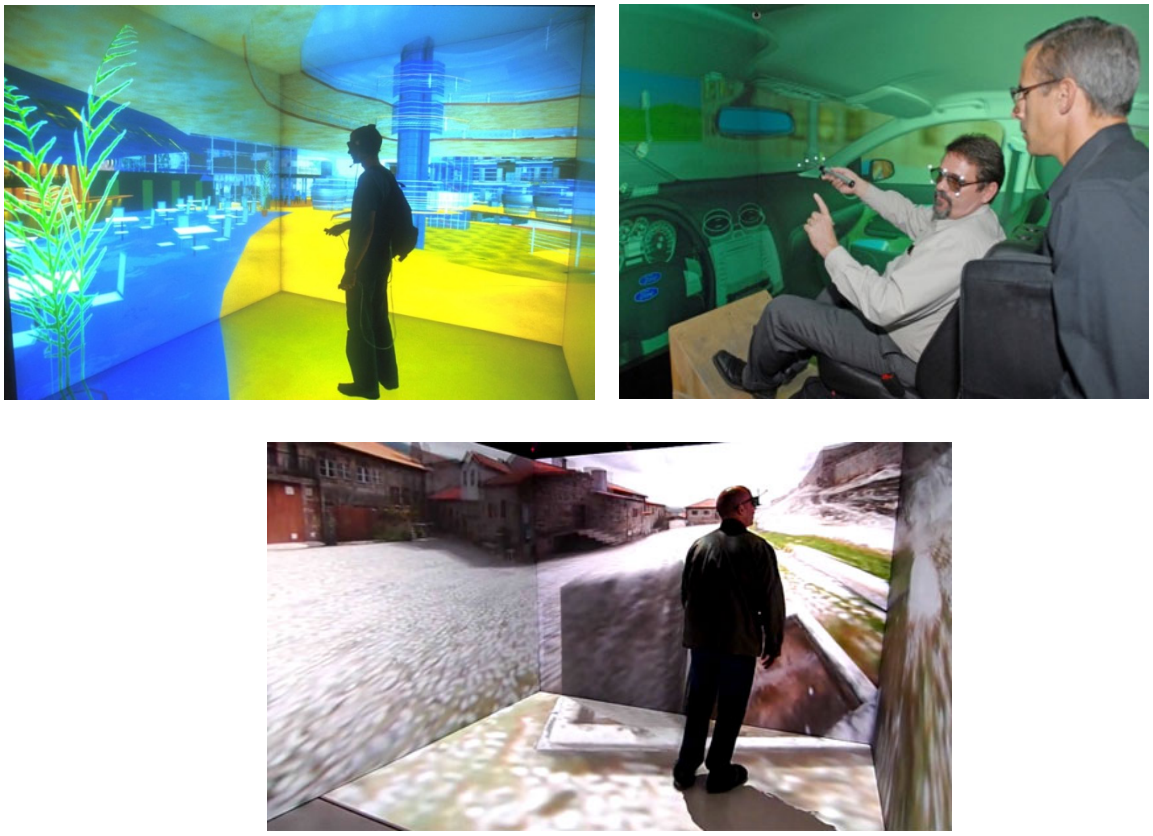


Fig. 3. Algunos ejemplos de sistemas virtuales semi-inmersivos: una plaza comercial, el interior de un coche y las calles de una ciudad.

2.3.3 Realidad virtual Inmersiva

Los sistemas Inmersivos son aquellos donde el usuario tiene la sensación de estar dentro del mundo virtual que está explorando. Este tipo de sistemas utiliza diferentes dispositivos, denominados accesorios (Fig. 4), los cuales pueden consistir de guantes, trajes especiales, visores o cascos, estos últimos le permiten al usuario visualizar los mundos virtuales a través de ellos, y precisamente estos son el principal elemento que le permiten al usuario sentirse inmerso dentro de estos ambientes [9]. Con la combinación de estos accesorios, se logra sumergir al usuario en estos ambientes, y tal vez en un futuro, gracias al avance de los motores gráficos podríamos lograr llegar al foto realismo y con el desarrollo de nuevos accesorios, lograr que el usuario no distinga el mundo real, del ambiente virtual en el que se encuentra. Estos sistemas son ideales para aplicaciones de entrenamiento o capacitaciones, así como para el manejo de diferentes situaciones que se faciliten de forma virtual para el usuario, para que luego las realice en el mundo real, como por ejemplo el manejo de las fobias usando realidad virtual inmersiva.

.

Fig. 4. Podemos observar los diferentes accesorios usados para la realidad virtual inmersiva, desde cascos o visores, hasta guantes y otras herramientas usadas para interactuar en el ambiente virtual.

2.4. Aplicaciones de la realidad virtual

La Realidad Virtual es una tecnología que puede ser aplicada en cualquier campo, como la educación, gestión, telecomunicaciones, juegos, entrenamiento militar, procesos industriales, medicina, trabajo a distancia, consulta de información, marketing, turismo, etc. Una de las principales aplicaciones es la tele robótica, que consiste en el manejo de robots a distancia, pero con la salvedad de que el operador ve lo que el robot está viendo e incluso tiene el tacto de la máquina. En la industria se utiliza también la Realidad Virtual para mostrar a los clientes aquellos productos que sería demasiado caro enseñar de otra manera o simplemente no están construidos porque se realizan a medida. Se están utilizando sistemas de este tipo, por ejemplo, para el diseño de calzado deportivo, permitiendo acortar los tiempos de diseño de un producto de vida muy

corta en cuanto a la permanencia de un modelo en el mercado. La Realidad Virtual también se utiliza para tratar sistemas que no pueden ser manejados en el mundo real. Por ejemplo, simulaciones de enfrentamientos bélicos, o simuladores de vuelo.

Otro campo de aplicación es el de la construcción de edificios. Entre otras posibilidades, la realidad virtual permite el diseño del interior y exterior de una vivienda antes de construirla, de forma que el cliente pueda participar en el mismo realizando una visita virtual de la vivienda que se va a construir.

En el ámbito de la medicina, además de facilitar la manipulación de órganos internos del cuerpo en intervenciones quirúrgicas (Véase la Fig. 5), la realidad virtual permite, entre otras posibilidades, la creación, para los estudiantes de medicina, de pacientes virtuales que adolecen de diversas enfermedades y presentan los síntomas característicos para poner en práctica las habilidades terapéuticas del futuro médico. En el tratamiento de fobias también se ha comprobado la utilidad de los sistemas de realidad virtual, donde el paciente tiene el control de la "realidad" y puede ir manejando su experiencia dentro de la misma.

Otras aplicaciones científicas de la Realidad Virtual consisten en el estudio de tormentas eléctricas, los impactos geológicos de un volcán en erupción, el diseño de compuestos químicos, el análisis molecular, la investigación en ingeniería genética, etc.

Fig. 5. Izquierda: microcirugía realizada de forma remota, derecha: modelo tridimensional del cuerpo humano

En resumen podemos ver que los campos en los que se puede aplicar la realidad virtual son muy extensos, si bien como vimos su historia la realidad virtual empezó como una forma de entretenimiento, actualmente se extiende más allá del simple entretenimiento. Esto la ha convertido en indispensable, como se observó en su uso para el entrenamiento de las personas en diferentes disciplinas, como por ejemplo el entrenamiento que reciben los pilotos de avión, hasta el entrenamiento que realizan los astronautas. Y en un futuro podríamos llegar a ser sumergidos en una realidad virtual absoluta en la cual no se podrá distinguir la realidad virtual de la realidad.

2.5 Desarrollo de Ambientes Virtuales en México

Actualmente se desarrollan ambientes virtuales enfocados en una gran variedad de campos a nivel mundial. Aunque en México todavía es un área que está en crecimiento, ya se cuentan con aplicaciones de ésta en diversos campos, entre estas aplicaciones, algunas de ellas desarrolladas en colaboración con la UNAM, podemos mencionar:

- El desarrollo de una terapia para personas que sufren de estrés postraumático porque han sido víctimas de algún hecho violento o testigos de uno. El cual consiste en exponer al paciente a imágenes de realidad virtual que recrean una situación violenta.
- En la Facultad de Psicología se ofrece terapia de inmersión en ambientes de realidad virtual a personas con trastorno obsesivo compulsivo enfocados en la contaminación y el orden. Los ambientes de realidad virtual incluyen cuatro escenarios: un baño, un autobús, un restaurante y una habitación. Cada uno de ellos tiene niveles de exposición que permiten evaluar el avance de los pacientes.
- El modelo tridimensional Sarah del laboratorio de enseñanza virtual del idioma español que opera en el observatorio de realidad virtual Ixtli, desarrollado en conjunto con el Centro de Estudios Para Extranjeros (CEPE El laboratorio). En el modelo se muestra la forma y el modo en el que se articulan las palabras en el idioma español.

3. Modelado

Un modelo 3D puede "verse" de dos formas distintas. Desde un punto de vista técnico, es un grupo de fórmulas matemáticas que describen un "mundo" en tres dimensiones. Desde un punto de vista visual, un modelo en 3D es una representación esquemática visible a través de un conjunto de objetos, elementos y propiedades que, una vez procesados (renderización), se convertirán en una imagen en 3D o una animación 3d. Por lo general, el modelo visual suele ser el modelo 3D que los diseñadores manejan, dejando las fórmulas a procesos computacionales. Esto es así, porque lo que el modelo en 3D visual representa se acerca más a la imagen en 3D final que se mostrará al renderizarse.

Existen aplicaciones de modelado en 3D, que permiten una fácil creación y modificación de objetos en tres dimensiones. Estas herramientas suelen tener objetos básicos poligonales (esferas, triángulos, cuadrados, etc.) para ir armando el modelo. Además suelen contar con herramientas para la generación de efectos de iluminación, texturizado, animación, transparencias, etc. Algunas aplicaciones de modelado son 3D Studio Max, Alias, Blender, Cheetah3D, Cinema 4D, Generative Components, Houdini, LightWave, Maya, MilkShape 3D, Rhinoceros 3D, Softimage|XSI, trueSpace, ZBrush, etc.

El modelo en 3D describe un conjunto de características que, en conjunto, resultarán en una imagen en 3D. Este conjunto de características suele estar formado por objetos poligonales, tonalidades, texturas, sombras, reflejos, transparencias, translucidez, refracciones, iluminación (directa, indirecta y global), profundidad de campo, desenfoques por movimiento, ambiente, punto de vista, etc. [2]

El modelado consiste en ir dando forma a objetos individuales que luego serán usados en la escena. Existen diversas técnicas de modelado las cuales veremos a detalle más adelante. Los procesos de modelado pueden incluir la edición de la superficie del objeto o las propiedades del material (por ejemplo, color, luminosidad, difusión, especularidad, características de reflexión, transparencia u opacidad, o el índice de refracción), agregar texturas, mapas de relieve (bump-maps) y otras características. El proceso de modelado puede incluir algunas actividades relacionadas con la preparación del modelo 3D para su posterior animación. A los objetos se les puede asignar un esqueleto, una estructura central con la capacidad de afectar la forma y movimientos de ese objeto. Esto ayuda al proceso de animación, en el cual el movimiento del esqueleto automáticamente afectara las porciones correspondientes del modelo. [2]

3.1. Tipos de modelado

El 3D es una mera representación de coordenadas, que conforman estructuras envueltas por una textura. Imaginémoslo, como estructuras de alambre, recubiertas de papel de colores. El truco, es realizar la malla de manera simple, para luego crear el material por el cual le daremos sus características tales como metal, barro, agua, etc [3]. Por tanto, primero se deben construir un modelo, para ello hay técnicas de modelo comunes las cuales son:

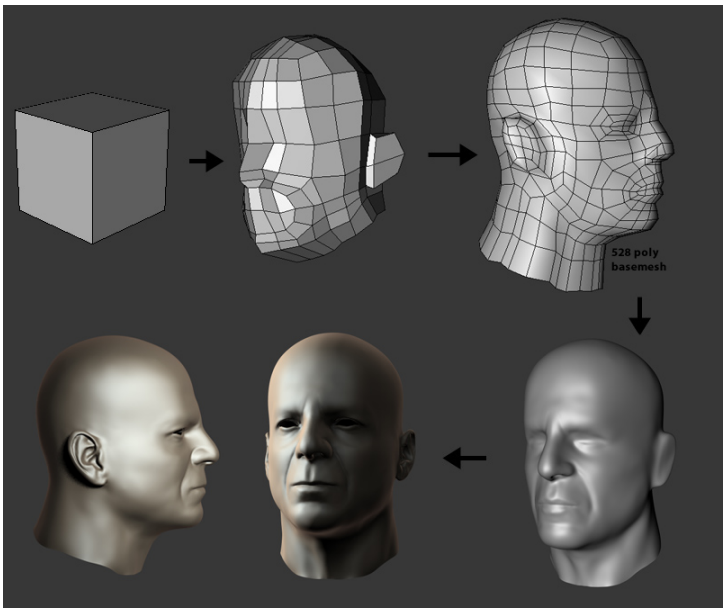
- Box Modeling.
- NURBS Modeling.
- Operaciones Booleanas.
- Extrude, Lathe.
- Loft.
- Sistema de Partículas.
- Modelos por Texturas.

A continuación procederemos a ver estas técnicas de modelado más a detalle y proporcionaremos ejemplos para complementar su explicación.

3.1.1. Box Modeling

Es una técnica de modelado poligonal en la que se comienza con una primitiva geométrica (cubo, esfera, cilindro, etc) y luego refina su forma hasta la apariencia deseada (Ver Fig. 6). A menudo en esta técnica se trabaja por etapas, comenzando con una malla de baja resolución, refinando la forma, a continuación, se va subdividiendo la malla para suavizar los bordes duros y agregar detalles. El proceso de subdivisión y refinar se repite hasta que la malla contiene suficientes detalles poligonales para transmitir adecuadamente el concepto deseado. Es de las técnicas más populares actualmente.

El elemento fundamental en la técnica de box modeling son las caras de cuatro lados, comúnmente llamadas quads, las cuales permiten resultados predecibles y consistentes, ya que pueden subdividirse en 2 o 4 triángulos (trazando una o dos diagonales) que poseen aproximadamente la misma dirección de la normal (recordemos que la gran mayoría de los motores de videojuegos interpreta las geometrías como tiras de triángulos).



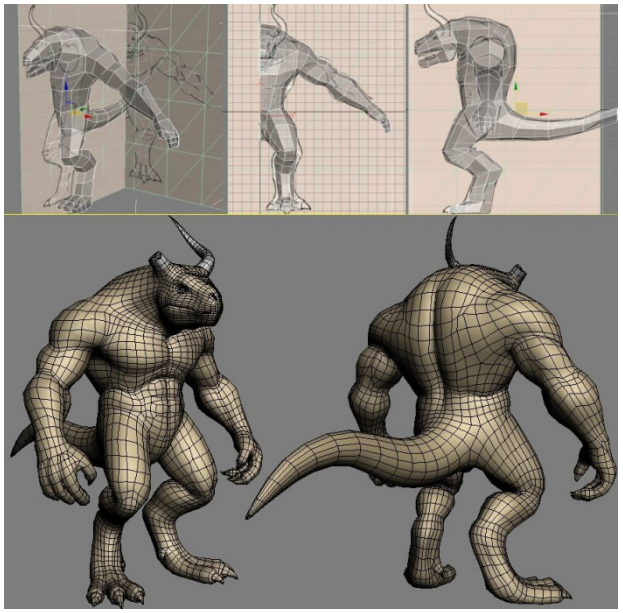


Fig. 6. Observamos cómo se puede obtener una malla compleja, usando como base un cubo, usando la técnica de box modeling

3.1.2. NURBS Modeling

Una curva NURBS se define por su grado, un conjunto de puntos de control, y un vector de nodos (Fig. 7). Las curvas y superficies NURBS son generalizaciones de las curvas de Bézier[1], así como de superficies. Mientras que las curvas de Bézier se desarrollan en una sola dirección paramétrica, normalmente llamada "s" o "u", las superficies NURBS evolucionan en dos direcciones paramétricas, llamada "s" y "t", o "u" y "v". [5]

Las curvas y superficies NURBS son útiles por varias razones:

- Son invariantes bajo transformaciones afines, así como de perspectiva.
- Ofrecen una estructura matemática común para figuras analíticas estándar (por ejemplo, cónicas) y figuras de forma libre.
- Proporcionan flexibilidad para diseñar una gran variedad de figuras.
- Reducen el consumo de memoria al almacenar figuras (en comparación con métodos más sencillos).
- Pueden ser evaluados rápidamente por algoritmos numéricamente estables y precisos.

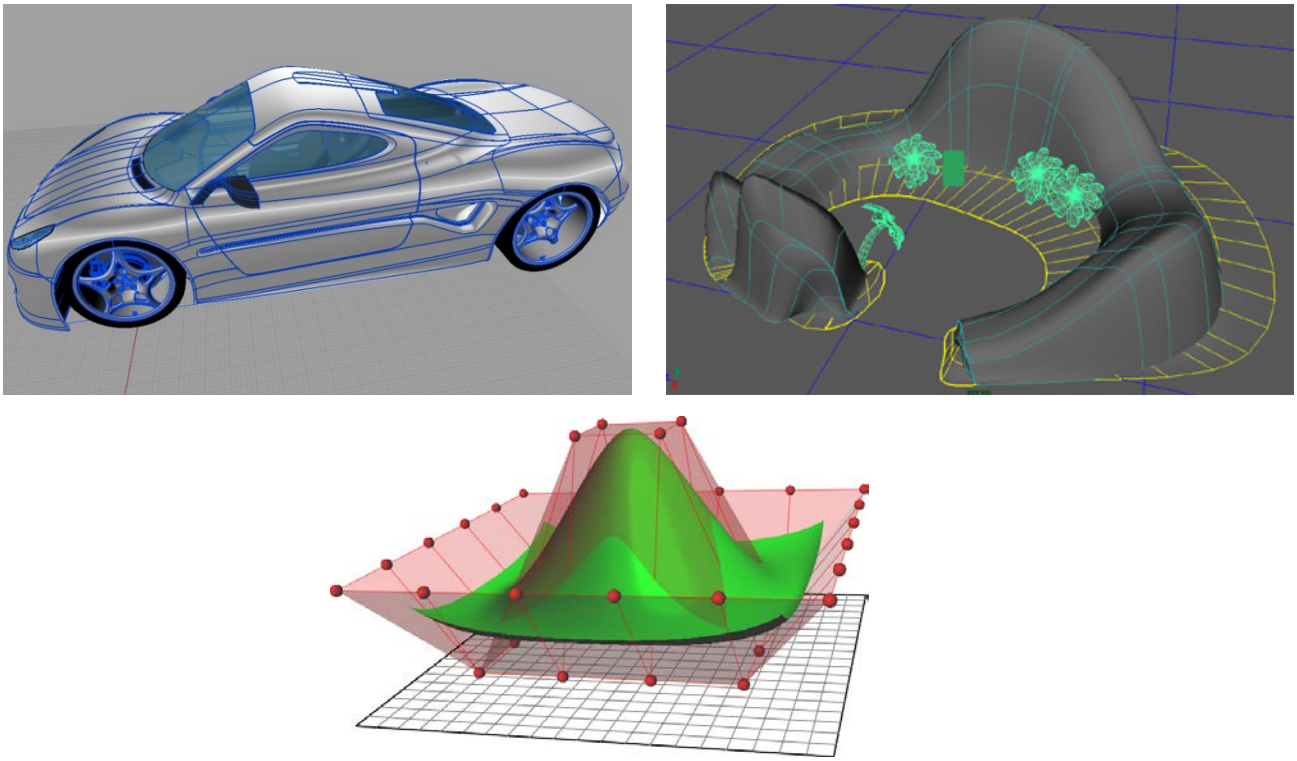


Fig. 7. Ejemplos de Superficies y objetos generados con NURBS

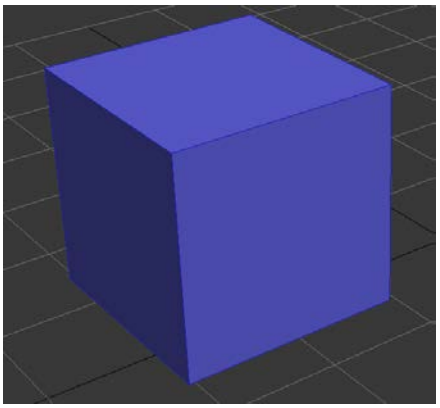
[1] Curva de Bézier.- La curva queda definida por la posición de los puntos extremos del segmento, y emplea otros dos puntos, generalmente fuera de la curva para definir las pendientes en los extremos [6]

3.1.3. Operaciones booleanas

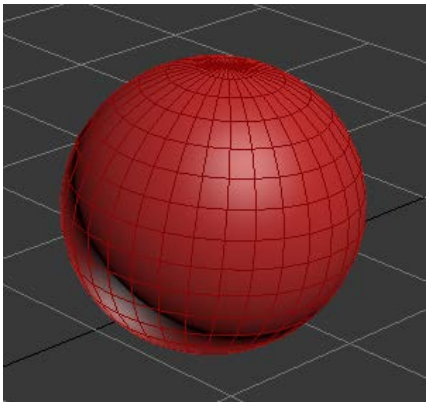
Las operaciones booleanas se basan en el sistema de reglas desarrollado por George Boole, las cuales establecen la forma en la que dos argumentos interactúan entre sí para obtener un tercero. Esto llevado a un modelo 3D se interpreta como: un objeto C será el resultado de la interacción entre dos objetos (A y B) definida por la operación que se haya realizado. Las operaciones que se pueden realizar son: Unión, Intersección y Diferencia. Por ejemplo:

-Si se tiene el objeto A que es una caja y el objeto B que es una esfera, el resultado de acuerdo a las operaciones será:

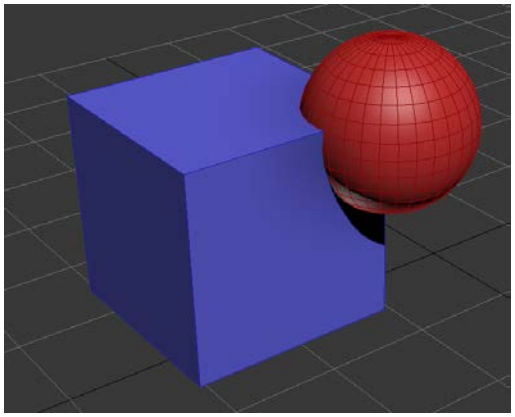
Objeto A



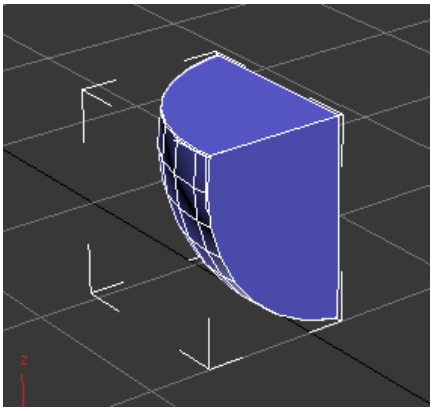
Objeto B



Unión

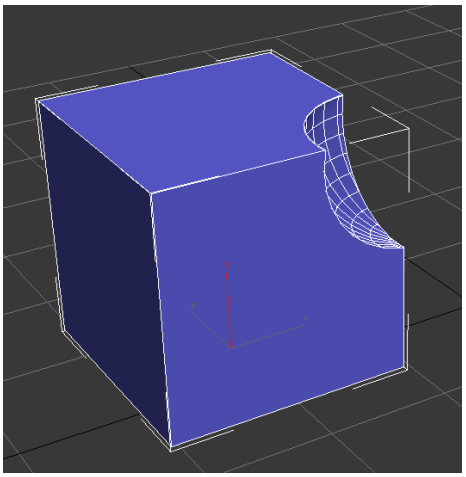


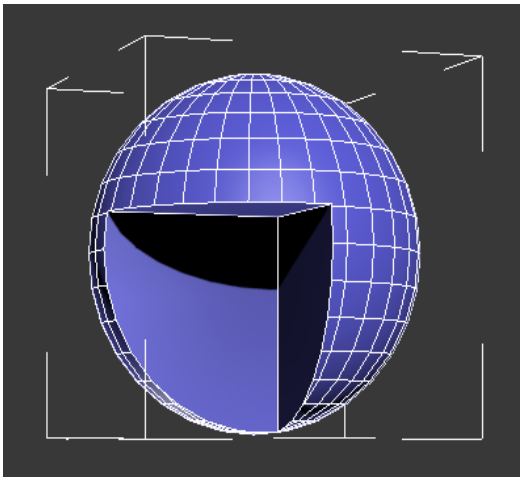
Intersección



Diferencia A-B

Diferencia B-A





3.1.4. Extrude, Lathe.

Son dos técnicas que a partir, de una figura 2d (spline) crea el volumen [6]. El extrude, da profundidad a un objeto 2D (Fig. 8.1). El Lathe, genera geometría 3D a partir de la revolución de un objeto 2D sobre un eje. Ideal para botellas, copas, y demás objetos sin diferencia en sus costados (Fig. 8.2).

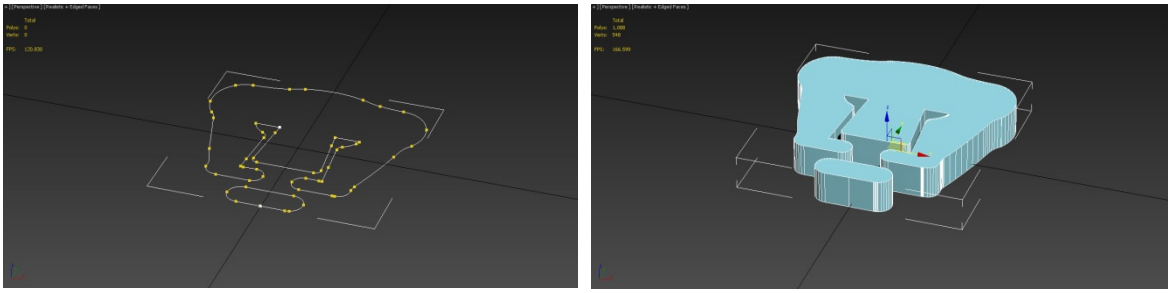


Fig. 8.1. Izquierda. Splines 2D, Derecha. Objeto generado con Extrude a partir de los splines 2D

Fig. 8.2. Izquierda. Spline 2D. Derecha Copa generada con Lathe

3.1.5. Loft

Se utilizan 2 ó más splines, para crear una malla 3D continua (Fig. 9). El primer spline, funciona como el camino a seguir (path) mientras que los demás, dan forma, extendiéndose, a través del path. Ideal para crear cables, botellas, etc. Este método de creación es muy poderoso y flexible, y permite generar objetos complejos fácilmente modificables.

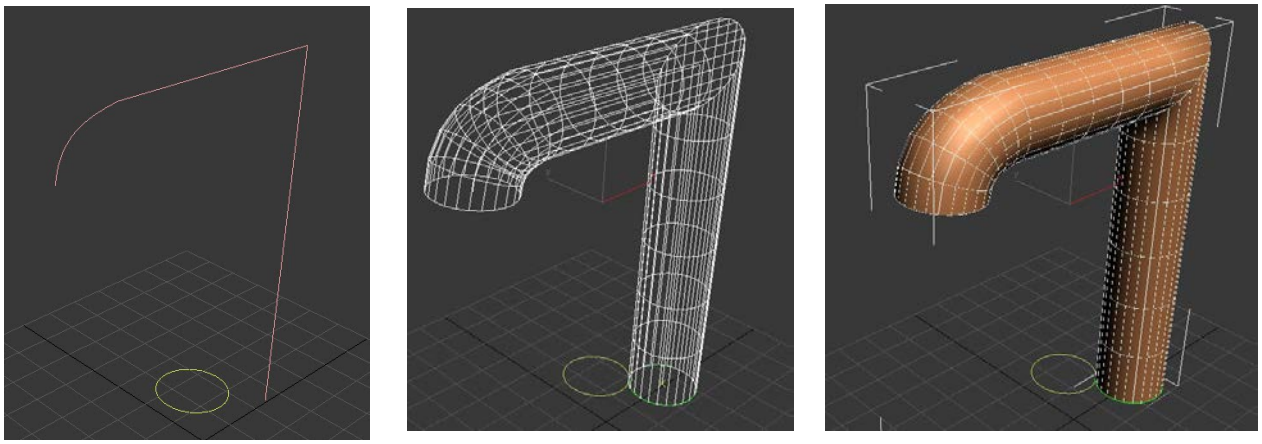


Fig. 9. En este ejemplo podemos observar la creación del path y la creación de los demás splines que siguen el path.

3.1.6. Sistema de Partículas

Podemos describir un sistema de partículas como una colección de muchos objetos diminutos (partículas), que al considerarse como un conjunto, representan un fenómeno cuya complejidad puede ser bastante elevada, pero también nos puede llevar a obtener resultados mucho más realistas que utilizando técnicas tradicionales (Fig. 10). En estos sistemas, cada partícula posee unas propiedades o atributos, que en la mayor parte de los casos coincide con atributos físicos del sistema modelado.

La principal característica de los sistemas de partículas es que evolucionan en el tiempo. Esta evolución temporal puede ser controlada de diversas formas, aunque lo usual es que venga determinada por reglas probabilísticas.

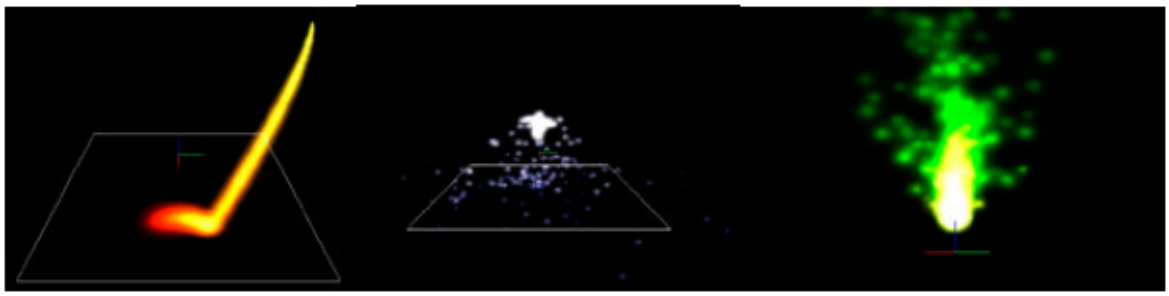


Fig. 10. Sistemas de partículas representando diferentes efectos.

3.2. Modelado de bajo conteo poligonal (Low poly)

El modelado de bajo conteo poligonal (low poly) es diferente al modelado para otras producciones 3D. El proceso normal se usa para secuencias no-interactivas, como películas, cortometrajes y series de animación, donde la imagen desplegada es siempre la misma. Los fotogramas se generan anticipadamente, y se editan en secuencias no-lineales. Ya que las imágenes son pre-generadas, se puede usar cualquier método para generarlas. Usualmente una red de computadoras procesan de forma paralela el conjunto de fotogramas, a este método de procesamiento se le conoce como render farm. El proceso es costoso y tardado, un fotograma de película actual puede tardar horas en procesarse en cientos de computadoras.

Los modelos low poly son usados en aplicaciones donde se hace un render en tiempo real, es decir, tantas veces por segundo como se necesite para una visualización fluida. Por ejemplo, si se generan imágenes para un televisor, la computadora genera 30 fotogramas por segundo, y los despliega a medida que son procesados. Las imágenes no están pre-procesadas, ni almacenadas en alguna parte. Las imágenes se generan al vuelo porque deben responder a los comandos del usuario. Si la computadora está mostrando un personaje de pie, y el usuario que controla al personaje decide moverlo usando un dispositivo de entrada, el programa recibe un comando y decide en el siguiente fotograma empezar a mover al personaje. Las imágenes que se despliegan en pantalla no pueden estar pre-procesadas, porque las acciones del usuario son impredecibles.

Por el hecho de tener que procesar las imágenes al vuelo, la calidad de imagen depende de las capacidades del hardware de video. Los primeros gráficos 3D en tiempo real eran de muy baja calidad, mientras que los más recientes se parecen cada vez más a las películas pre-procesadas (Fig. 11). El modelado poligonal de personajes y escenarios es distinto en películas y videojuegos por obvias razones. Las limitaciones de las tarjetas de video no permiten tener personajes con millones de polígonos rendereándose en tiempo real.

Un personaje de película puede usar modificadores como TurboSmooth para suavizar la apariencia de las superficies, usar biselados para mejorar los bordes de los objetos, y también tener detalles geométricos pequeños modelados, como tornillos y remaches. Todo esto agrega millones de polígonos a un modelo. En el rendereado de una película esto no es tan grave, porque se procesa antes de tiempo, pero para un videojuego, los objetos no pueden tener un nivel tan alto de detalle. Se usan modelos simples y de bajo conteo de polígonos (Fig. 11). Un personaje de esta generación puede tener de 8.000 a 40.000 polígonos aproximadamente, dependiendo del hardware y el tipo de programa [4]. Los detalles más pequeños deben ser simulados usando texturas con los detalles "dibujados" o usando mapas de normales.

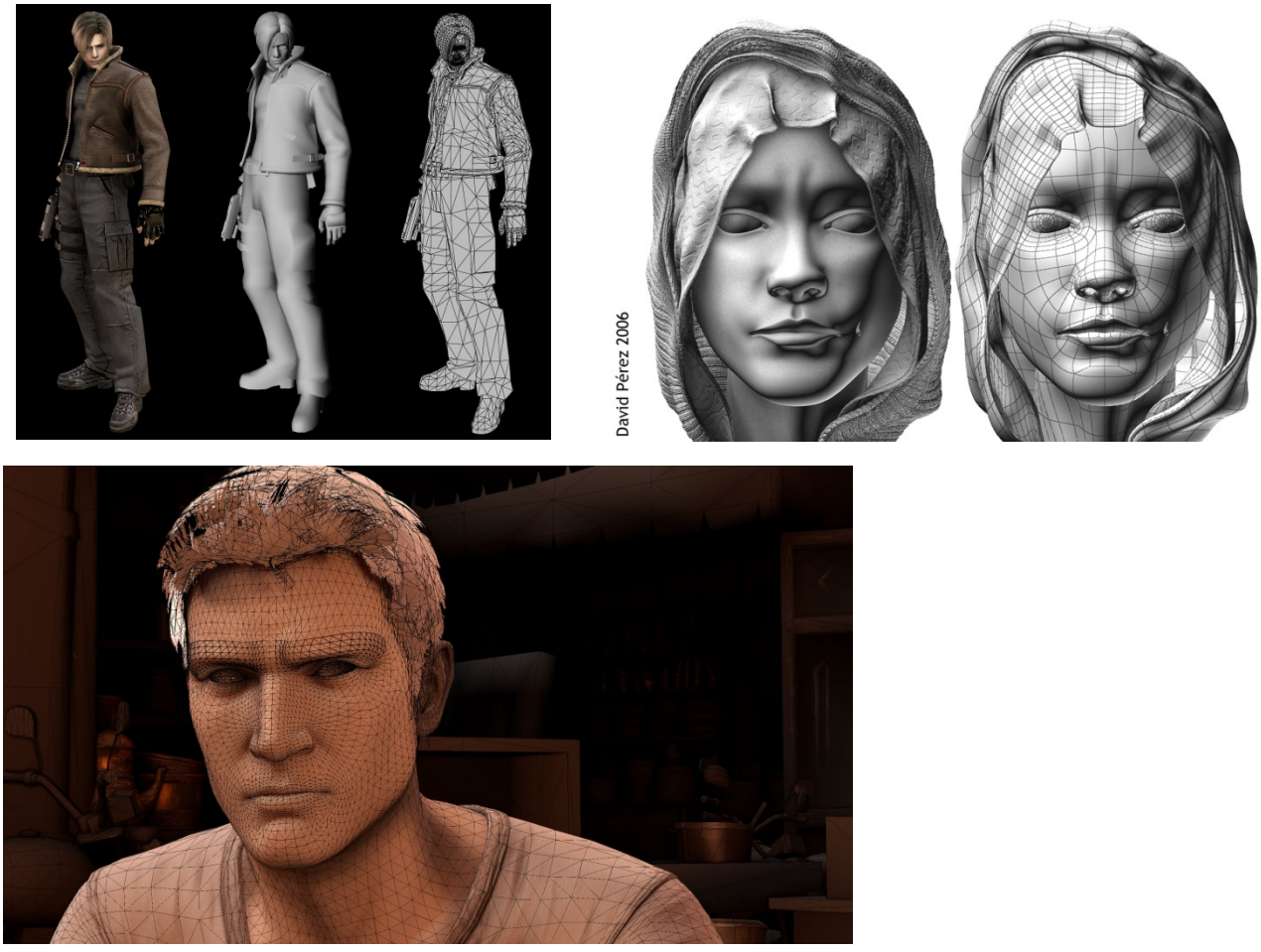


Fig. 11. Modelos de diferentes personajes de videojuegos, y con diferente conteo de polígonos, gracias al avance de las tecnologías.

3.3. Texturizado.

Ya que un modelo para videojuegos no puede tener todos sus detalles modelados independientemente, se usan modelos más simples con los detalles pintados sobre las texturas. En los primeros juegos 3D era común ver personajes que parecían traer guantes en lugar de manos, con las separaciones de los dedos dibujadas en las texturas. Hoy con los avances en hardware se pueden tener más libertades, pero aún existen limitaciones.

Los motores de videojuegos[2] no tienen capacidades de iluminación tan avanzadas como en el rendereado offline. Por lo mismo muchas texturas tienen algo de información de iluminación "cocinada" sobre el color. Es común que una textura tenga una capa de oclusión ambiental generada desde un modelo de mayor detalle, para agregar fidelidad visual a una textura.

El texturizado consiste en aplicar a las caras de los distintos objetos bitmaps o cualquier otro tipo de gráfico (como colores planos, etc.) dándole además características ópticas a esos objetos, como por ejemplo, opacidad, reflexión, refracción, etc. Además, mediante el texturizado también podemos dar algunas características físicas como un cierto relieve.

Estos detalles se logran crear, aplicando los diferentes tipos de canales y de mapas que se pueden utilizar para darle mayor calidad a las texturas. Éstos son:

- Canales Alfa
- Mapa de Normales
- Mapa de Relieves
- Cube Maps

Y estos se aplican con un método de mapeo, como el mapeo UV.

3.3.1. Canal Alfa

El concepto de canal alfa fue presentado en el artículo Compositing Digital Images (1984) de Thomas Porter y Tom Duff. El sistema de color tricromático, se conforma por tres canales que nos indican la saturación de cada color. El canal alfa se refiere a un cuarto canal en el que se almacena un valor entre 0 y 255 para cada pixel. En este canal podemos incluir mapas de 8 bits o de escala

[2] Para más información de los motores de juegos ir al capítulo 4

de grises en lugar de generar una imagen por separado como pueden ser los mapas de transparencia, mapas especulares, etc.

Por ejemplo en el caso de los mapas de transparencia, un valor de 255 indica que el pixel es completamente opaco. Un valor de 0 significa transparencia absoluta. Utilizando valores intermedios en este rango, podemos determinar el nivel de transparencia del objeto (Fig. 12).

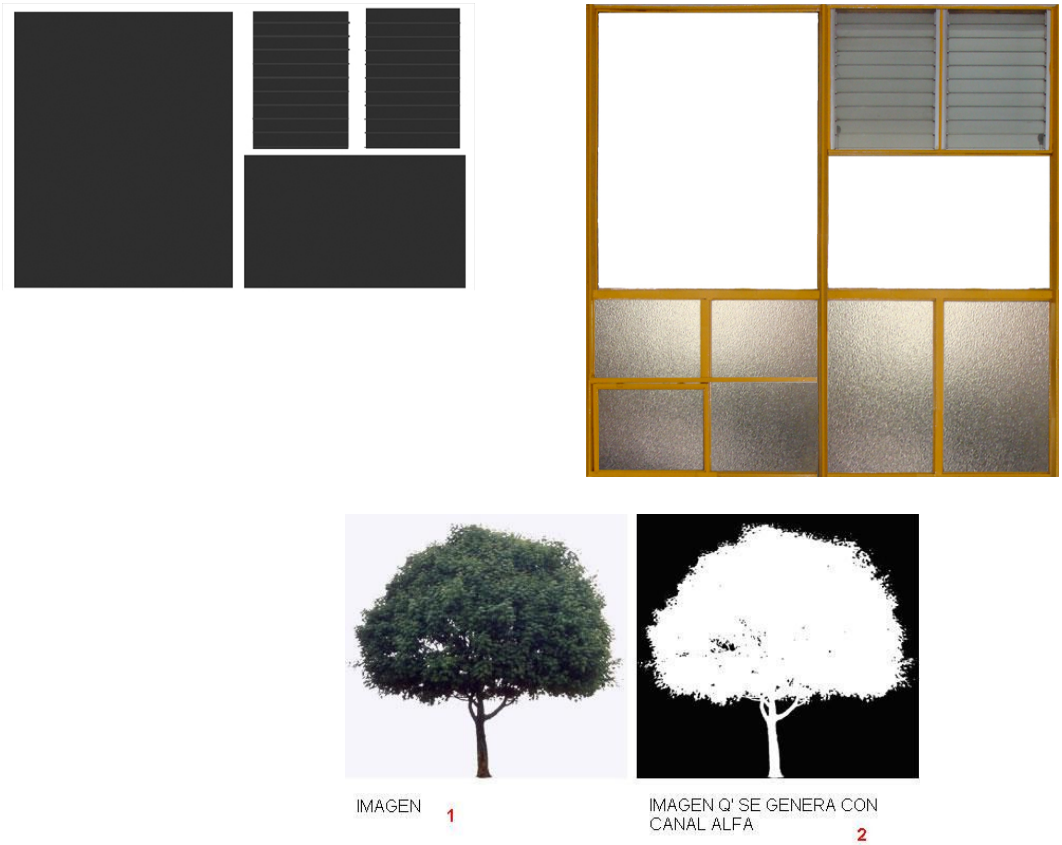


Fig. 12. Podemos observar los Canales alfa con sus respectivas texturas. Arriba Textura de las ventanas de la facultad de Ingeniería. Abajo Imagen de un árbol

3.3.2. Mapa de Relieves

Los mapas de relieves son un recurso empleado en la generación de gráficos por computadora creado por James F. Blinn en 1978. Consiste en dar un aspecto rugoso a las superficies de los objetos [8]. Esta técnica modifica las normales de la superficie sin cambiar su geometría. Las normales originales de la superficie seguirán perpendiculares a la misma. El mapeado de relieves cambia la perpendicularidad por otras normales para lograr el efecto deseado, todo ello sin modificar la topología ni la geometría del objeto (Fig. 13). El resultado es razonablemente rico y detallado, y pueden lograrse grandes parecidos a elementos naturales (como la textura de una naranja).

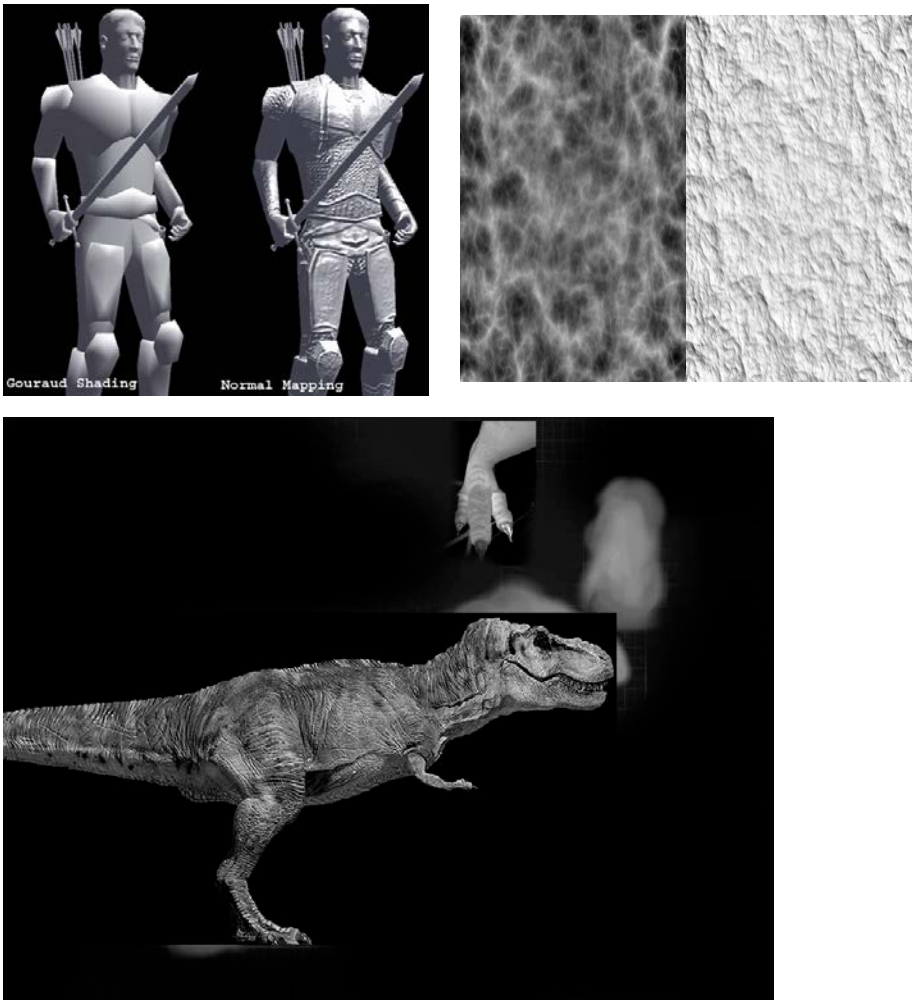


Fig. 13. Los efectos del bump mapping, en diversos objetos.

3.3.3. Mapas de Normales.

Los mapas de normales o “normal bump”, al igual que los mapas de relieve (bump), se aplican para simular detalles en los objetos que no lo tienen (Fig. 14). Este es un efecto de cámara, que simula el desplazamiento de los píxeles – solamente en el render -, según sea la posición de la cámara. Estos mapas cumplen la misma función que un mapa de relieves, pero almacenan mucha más información, pues poseen tres canales de colores (RGB), al contrario de los mapas de relieves, que trabajan en función de una escala de grises. [8]

El objetivo de utilizar mapas de normales es reducir el número de polígonos (y tiempo) que se necesitan para modelar un objeto complejo. Normalmente se utilizan en aplicaciones que se ejecutan en tiempo real pues aliviana la carga del procesador. Sin embargo, puede ser bastante útil, en el caso de escenas estáticas o animaciones.

Para comprender los mapas de normales, tenemos que definir el concepto de normal. La normal de la cara de un objeto, corresponde a un vector tridimensional que determina la dirección en que ella apunta, siendo este, perpendicular a la cara. En el caso de un punto (o píxel), la normal corresponde al vector perpendicular a un plano imaginario tangente a él. Como el mapa de bits de normales corresponde a una gradiente de 3 colores (RGB), es posible almacenar información tridimensional de la normal de cada píxel. Los colores que se almacenan en la imagen corresponden a la gama de los rojos, verdes y azules. Sus rangos varían entre 0 y 255, y corresponden a valores entre -1.0 y 1.0, los cuales son interpretados para saber el sentido de las normales de cada píxel. El color rojo representa al eje X, el verde representa al Y, y el azul al Z. Aunque en la realidad todos los píxeles de una cara plana apuntan hacia el mismo lado, el mapa de normales define hacia donde apuntan (supuestamente) cada uno de los píxeles. Esto permite simular relieve, pues cada píxel recibe luz, dependiendo de su orientación, por lo tanto, si el objeto es iluminado de izquierda a derecha, recibirán luz solamente los píxeles cuyas normales apunten hacia la izquierda, generando la sensación de relieve.

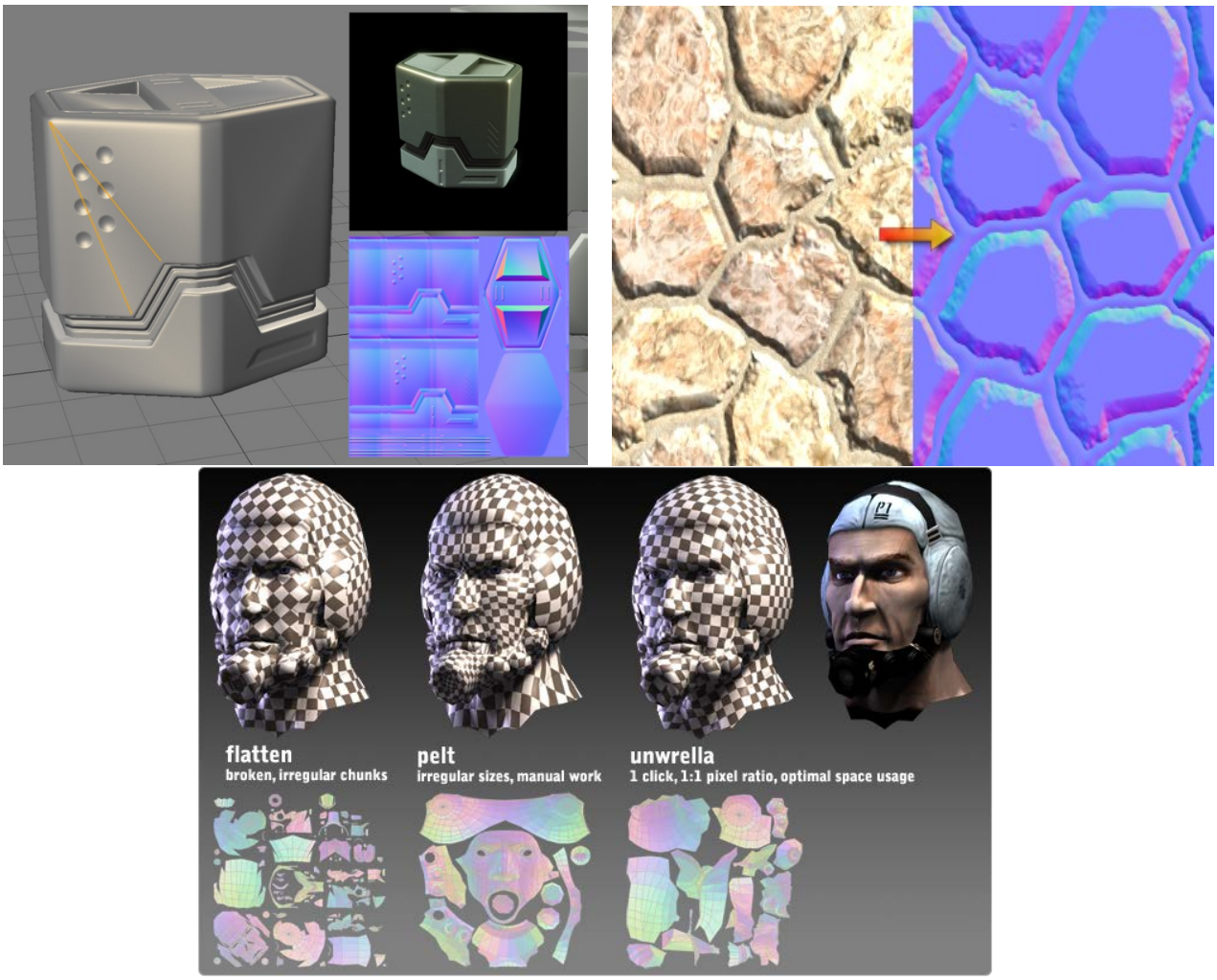


Fig. 14. Podemos observar los cambios que sufren los objetos, una vez que se les agrega el mapa de normales

3.3.4. Cube Maps

Mediante el uso de imágenes predefinidas creadas con forma de mapa geométrico que rodea a un objeto, se pueden definir los reflejos para un solo punto en el espacio. El cube map (Fig. 15) representa una imagen de textura del mapa de entorno basada en el vector reflejado para cada punto del objeto. Al proyectar el entorno 3D sobre un mapa de entorno 2D que rodea a un objeto, se pueden crear reflejos bastante precisos. Los cube maps permiten reducir en gran medida el número de cálculos necesarios en la simulación de reflejos a la vez que genera resultados de gran calidad de imagen.

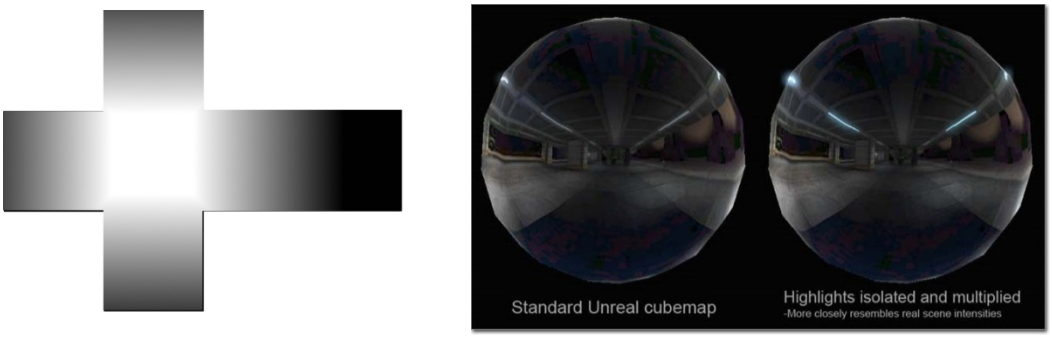
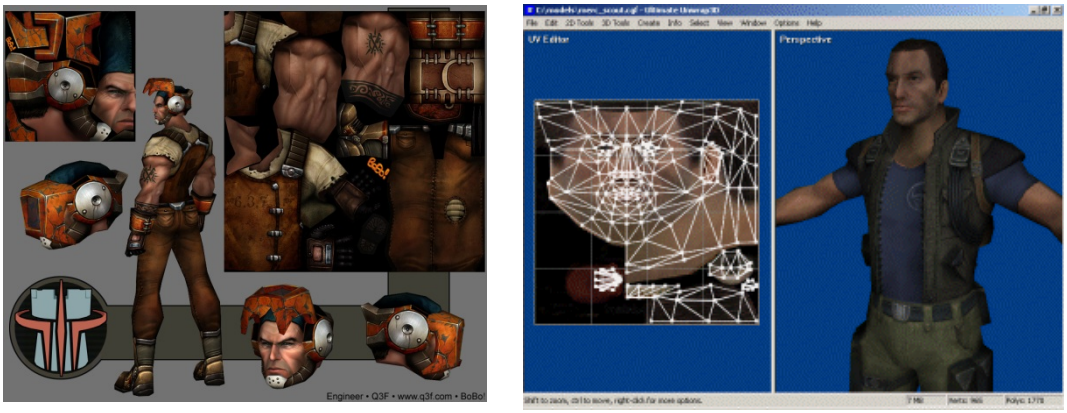


Fig. 15. Cube Maps de ejemplo.

3.3.5. Mapeo UV

El mapeo UV (Fig. 16) es una manera de asignar texturas de tipo Imagen sobre modelos tridimensionales. Se puede usar para aplicar texturas a formas arbitrarias y complejas como cabezas humanas o animales [2]. A menudo estas texturas son imágenes dibujadas, creadas con programas como Gimp, Photoshop, o cualquier otro programa de dibujo. Un mapeo UV es una forma de asignar la parte de una imagen a un polígono en el modelo. Cada vértice del polígono es asignado a un par de coordenadas 2D que definen que parte de la imagen es mapeada. Estas coordenadas 2D se llaman UVs. La operación de crear estos mapas UV se conoce también como "despliegue" ("unwrap" en inglés), debido a que todo ocurre como si la malla fuera desenvuelta o desplegada sobre un plano 2D.

A diferencia de los vértices, las UV pueden estar separadas de sus caras, lo que permite crear patrones para aplicar las texturas de forma posterior. Las UV determinan cómo va a proyectarse una textura sobre el modelo. Al crear primitivas como cubos, cilindros y esferas, estas vienen con sus UVs predefinidas. Si se aplica una textura directamente sobre una primitiva, no va a haber gran deformación de la imagen. Por el contrario, con un modelo complejo, las UV tienen que estar definidas manualmente, para evitar deformaciones de la textura.



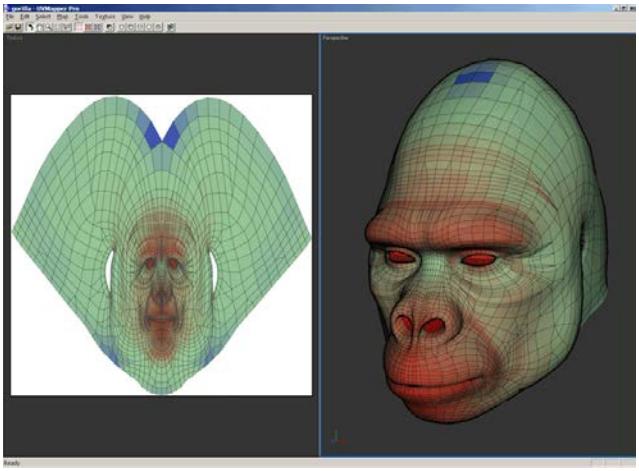


Fig. 16. En la parte superior, podemos ver unas texturas desplegadas para un fácil mapeo, en la parte inferior podemos ver el mapeo de las UV´s.

3.4. Iluminación

Las técnicas de iluminación pretenden simular o aproximar los fenómenos físicos de distribución de la luz (reflexión, refracción, absorción, etc.) sobre una escena 3D, con el objetivo de calcular la intensidad lumínica en cualquier punto del modelo. Las técnicas de iluminación directa son aquellas que tienen en cuenta solamente la luz que llega a las superficies en forma directa desde las fuentes. En cambio, las de iluminación global, permiten capturar la iluminación indirecta (la que proviene de reflexiones y refracciones en otras superficies), lo cual aporta una gran cuota de realismo a las imágenes generadas aunque tienen un mayor tiempo de procesamiento. Sin ella, las imágenes obtenidas tienen un aspecto plano y sintético.

Estos algoritmos se alimentan de datos como:

- Descripción geométrica de la escena (vértices, caras, normales, etc.).
- Descripción de materiales de las superficies (coeficientes de brillo, transparencia, color, rugosidad, etc.).
- Descripción de fuentes de luz (color, dirección, potencia, etc.).

3.4.1. Iluminación Global

Un objeto en la vida real, a no ser que se encuentre dentro de un recipiente totalmente negro es iluminado por luz directa e indirecta o solo por luz indirecta. Esto se debe a que los fotones emitidos por la fuente de luz rebotan en todas las superficies con las que se encuentran entonces un objeto en un lugar físico es irradiado por fotones directos de la fuente de luz y por fotones que han rebotado en las superficies circundantes (iluminación indirecta). Cada superficie circundante al objeto producirá una intensidad de iluminación distinta desde distintas direcciones y con distintas tonalidades de color sobre el objeto dependiendo del color de cada superficie y a su vez la luz también rebotará en el objeto de atención (Fig. 17).

Existen numerosas técnicas de iluminación global, pero sin duda las más famosas son Ray Tracing y Radiosity. Analizándolas en detalle la mayoría se pueden incluir en uno de estos dos grupos:

- Métodos de muestreo de puntos: por ejemplo Ray Tracing, donde se toman gran cantidad de muestras de la iluminación en diferentes puntos de la escena.
- Métodos de elementos finitos: por ejemplo Radiosity. La distribución de luz se calcula resolviendo un sistema de ecuaciones lineales que representa el intercambio de luz entre parches de las superficies.

Además hay técnicas híbridas que combinan aspectos de ambos métodos.

Fig. 17. Ejemplos de Iluminación global aplicados a una escena

3.4.2. LightMaps

Dado que el cálculo de la iluminación en tiempo real suele ser muy costoso a nivel computacional, se desarrolló un "truco" que consiste en pre-calcular la iluminación que deben tener las partes de una escena y aplicar esa iluminación pre-calculada en forma de una segunda textura añadida a la textura normal de la escena (Fig. 18), de forma que se ahorra tiempo de cálculo (es mucho más rápido aplicar una textura que calcular la iluminación frame por frame) y además no es excluyente de la utilización de luces dinámicas.

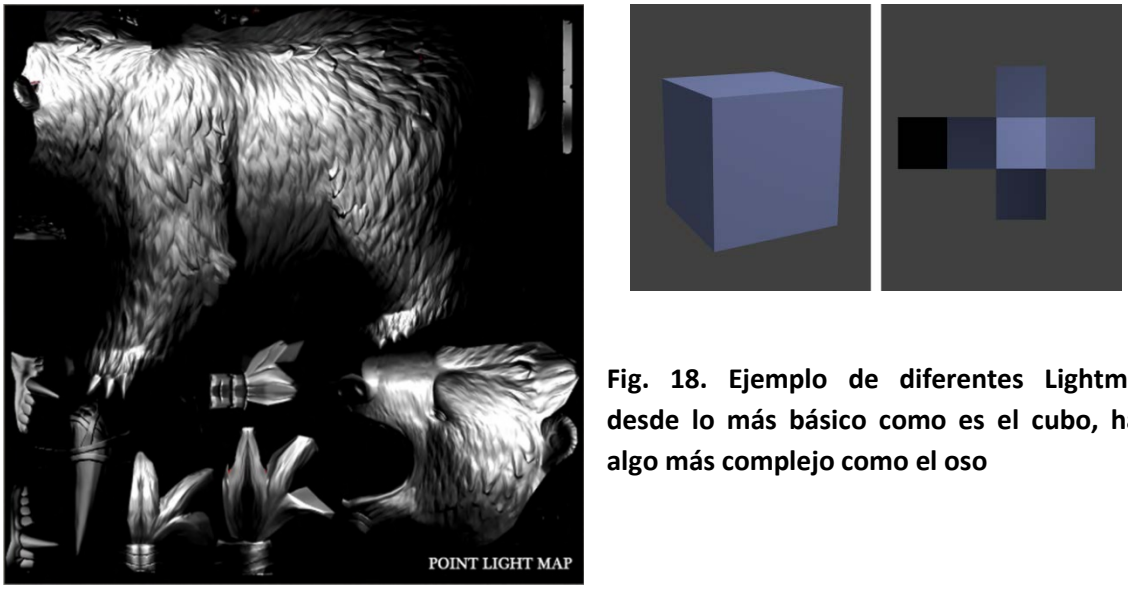
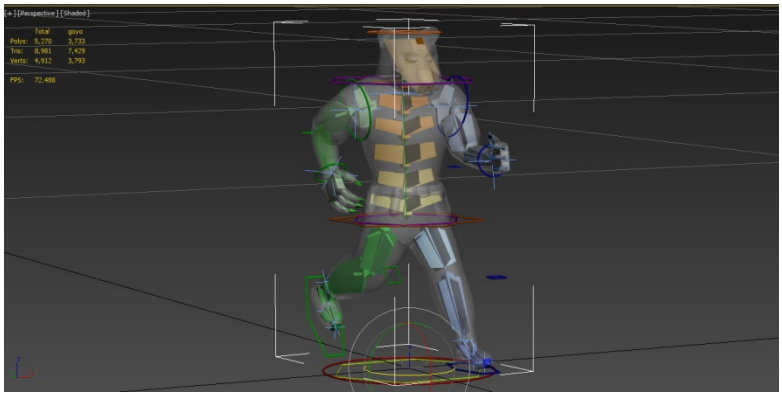


Fig. 18. Ejemplo de diferentes Lightmaps, desde lo más básico como es el cubo, hasta algo más complejo como el oso

3.5. Animación de Personajes.

Animar una malla o un objeto 3D mediante transformaciones como, rotación, traslación y escalamiento, consume mucho tiempo y los resultados son de baja calidad. Para esto, de igual forma en la que un cuerpo humano tiene un esqueleto y una serie de músculos que le proporcionan estabilidad y movimiento; a un objeto 3D se le crea una geometría interna enlazada entre si la cual se encarga de proporcionar movimiento para la geometría externa. Al proceso de crear el sistema de huesos, de controles que faciliten el movimiento de los mismos y a la programación de funciones que limiten ciertos movimientos no deseados para la animación se le llama Rigging (Fig. 19). Una vez que se tiene el rigg, se deben especificar la partes de la geometría interna que van a proporcionar movimiento a la geometría externa, a este proceso de enlace entre las dos geometrías se le llama Skinning.

Para hacer una comparación, un rigger es como un "creador de marionetas digitales", es decir existe un modelo 3D que no se puede mover en ninguna forma, entonces un rigger crea un sistema de huesos, controles y scripts para que dicho modelo se mueva en la forma en que debe hacerlo de acuerdo a lo que se requiere [1].



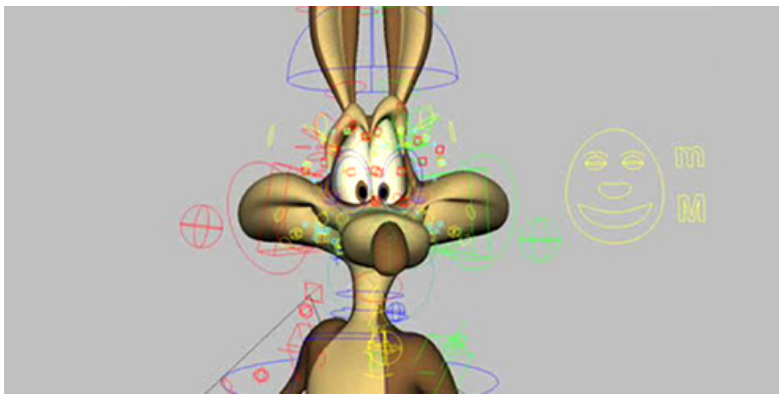


Fig. 19. Observamos diferentes tipos de Rigging.

En la parte superior, vemos un Rigging enfocado al movimiento del cuerpo. Mientras que en el Rigging inferior está enfocado a las expresiones faciales del personaje

Como se pudo observar, hay diferentes técnicas de modelado, las cuales se deben escoger de acuerdo al proyecto a realizar. Pero para tener un modelo de mayor impacto visual no basta con solo modelarlo, también se requiere de su texturizado asi como de la iluminación de la escena para permitir un ambiente más real. Y no podemos olvidarnos de la animación del personaje, de todo su proceso de rigging si lo que se quiere es realizar un videojuego, una animación o un recorrido virtual, ya que si solo se quiere realizar una foto, no hay necesidad de realizar este paso.

4. Motor de juegos

Un motor de juegos es un conjunto de sistemas de software que facilita el desarrollo de un videojuego proporcionando herramientas que de otra forma tendrían que ser construidos o creados cada vez que se tuviera que realizar un nuevo proyecto, como son los sistemas de iluminación, la inteligencia artificial, soporte para sonido, la física de los objetos, el cargador de modelos entre otros [9]. Los motores de juego pueden ser:

- Motores de juego de propósito específico: Se desarrollan para un género de videojuegos de forma específica.
- Motores de juego de propósito general: Su desarrollo tiene una mayor complejidad esto se debe a que abarcan un amplio género de videojuegos (Fig. 20).
-

A pesar de que existan motores de juego de propósito general, desafortunadamente no existe un motor que pueda desarrollar cualquier tipo de juego, ya que se realizan optimizaciones específicas a los géneros y a las diferentes plataformas de hardware.

La creación de un motor de juegos es un proceso complicado y largo, en la actualidad existen motores comerciales que venden licencias para permitir crear juegos con componentes de software ya existentes.

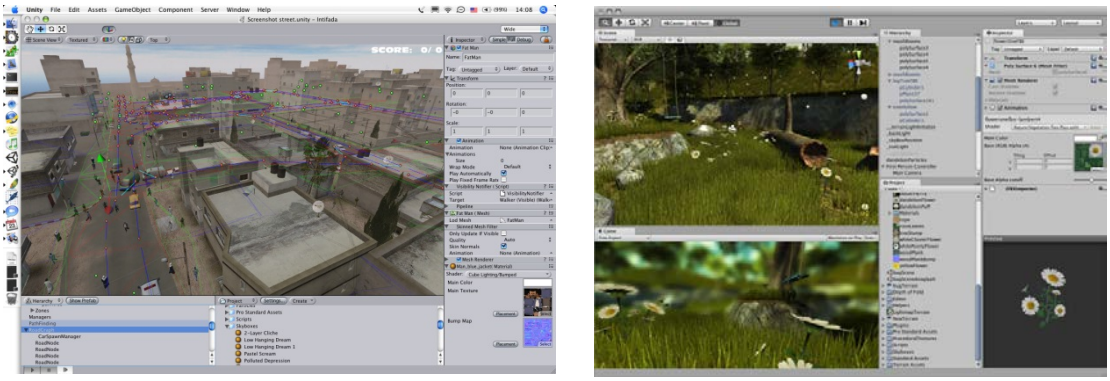


Fig. 20. Unity, ejemplo de un motor de propósito general.

4.1. Introducción a los Motores de juegos.

El desarrollo de un motor de juegos es equivalente al desarrollo de un sistema informático. Un motor de juegos bien diseñado cuenta con lo siguiente:

- Mantenibilidad: El motor debe escribirse de tal manera que pueda evolucionar para cumplir con las necesidades de los desarrolladores de videojuegos. Es inevitable un cambio, debido al uso de nuevas tecnologías y las crecientes expectativas por parte de los jugadores.
- Portabilidad: Debe ser fácil de usar el motor en diferentes computadoras.
- Confiabilidad: La confiabilidad tiene un gran número de características, incluyendo la fiabilidad, protección y seguridad. El software seguro no debe causar daños en caso de una falla del sistema.
- Eficiencia: El motor no debe hacer que se malgasten los recursos del sistema, como la memoria y ciclos de procesamiento. Por lo tanto, la eficiencia incluye tiempos de respuesta y de procesamiento, utilización de la memoria, etcétera.
- Usabilidad: El motor de juegos debe ser fácil de utilizar para el desarrollo de juegos. Esto significa que debe tener una interfaz de usuario apropiada y una documentación adecuada.
- Facilidad de pruebas: El motor de juegos debe buscar mantener sus componentes aislados, de esta manera se facilitan las pruebas de software.

4.2. Historia de los Motores de Juegos.

Un motor de juegos es un conjunto de sentencias de programación que necesita una computadora para crear entornos visuales y representar ambientes que sean lo más similar a la realidad posible, aunque no siempre se logró alcanzar niveles de realismo creíbles a causa de las obvias limitaciones tecnológicas de hace unos años. Hoy en día, tampoco se alcanzó el realismo absoluto; sin embargo, las tarjetas gráficas actuales tienen un poder de procesamiento enorme (incluso mayor que el de un microprocesador), lo que indica que hasta una computadora hogareña le pisa los talones a las grandes estaciones gráficas para representar entornos en tres dimensiones. Los motores de juegos no sólo se utilizan como base para videojuegos, son de vital importancia en simulaciones de todo tipo (entrenamientos militares, quirúrgicos y policiales). También existen los motores de juegos 2D: basta con recordar el término SCUMM en el que se basaron una gran cantidad de aventuras gráficas de la casa LucasArts; pero por obvias razones los motores bidimensionales son de mucha menor complejidad.

Un mismo motor de juegos puede ser utilizado en más de un videojuego (como el caso de CryEngine2 que fue usado para juegos como Crysis, Blue Mars, Entropia Universe), aunque hay empresas que han concebido un motor dedicado y exclusivo para un juego en particular (por ejemplo: el Iron Engine en el juego Sins of a Solar Empire).

Existen motores desarrollados por importantes empresas durante años. La inversión de tiempo y dinero para su creación es tal, que la licencia para que otras empresas puedan usarlos asciende a costos muy altos, aún más en los motores multiplataforma. Por su parte, en menor medida que los motores propietarios, existen motores gratuitos y de código abierto, siendo Unity uno de los más famosos al ser compatible con Windows, GNU/Linux y Mac OS X (aunque su licencia és paga para plataformas como Android, PS3, XBOX 360 o Wii). No debemos confundir los motores de juegos con las API's o librerías de abstracción, como DirectX o OpenGL. La cadena de dependencias comienza con el juego o aplicación software, desarrollado partiendo de un motor gráfico que, a su vez, se basa en una capa de abstracción de hardware. Una capa de abstracción suele ser parte de los sistemas operativos modernos y se encarga de interactuar en forma directa con el hardware: tarjetas gráficas (ya sea una nVidia GeForce GT240 o una ATI Radeon HD5850), tarjetas de sonido (ya sea una X-Fi 5.1 o una Realtek genérica), dispositivos de entrada (diversos modelos de joysticks o volantes) y de comunicación (tarjetas de red o módems).

4.3. Tipos de Motores de Juegos.

Los motores de Juegos, se clasifican de acuerdo a la acción especializada de la que se encargan, tenemos motores generales, que no tienen especialización pero como tal forman parte de la clasificación. De acuerdo a su grado de especialización los dividimos en:

- Motores Generales.
- Motores Gráficos.
- Motores de colisiones y física.
- Motores de Animación.

4.3.1 Motores Generales

Los motores generales, son desarrollados para cubrir un amplio género de videojuegos, evitan el enfocarse en un solo género. Aunque a pesar de esto, cada motor tiende a favorecer el género del juego con el que se desarrolló en mente. Como ejemplos tenemos los siguientes:

RAGE Engine.- Motor comercial desarrollado por Rockstar San Diego. Se comenzó a desarrollar RAGE (Rockstar Advanced Game Engine) en 2004. Permite utilizar mundos extensos y tiene una compleja Inteligencia Artificial. Este motor está orientado a la creación de escenarios de gran tamaño por lo cual favorece al género sandbox.

Naughty Dog Game Engine.- Motor comercial desarrollado específicamente para el PS3. Maneja una infinidad de objetos dinámicos con física independiente, interacción ambiente-animación, efectos de iluminación e Inteligencia Artificial. Tiene transiciones entre cinemáticos y juego casi imperceptibles por lo que favorece la creación de juegos con un estilo cinematográfico.

Unreal Engine.- Motor comercial desarrollado por Epic Games, su núcleo fue desarrollado en C++ y funciona en las plataformas: Dreamcast, Xbox, Xbox 360, PlayStation 2 y PlayStation 3. La última versión del Unreal Engine es la Unreal Engine 4 usa DirectX 11 y funciona en una amplia gama de plataformas. Utiliza PhysX para la física y colisiones. Es el motor de juegos más popular que existe, y cuenta con una gran versatilidad ya que en él se han desarrollado juegos pertenecientes a casi todos los géneros existentes.

Unity 3D.- Es un motor multiplataforma desarrollado por Unity Technologies, está diseñado para el desarrollo de juegos en plataformas web, computadoras, consolas y dispositivos móviles. Este motor fue desarrollado en los lenguajes C/C++, y soporta scripts escritos en lenguajes como C#, Javascript y BOO. Su última versión es la 4.1 liberada en marzo del 2013. Cuenta con una versión gratuita que cuenta con las herramientas necesarias para desarrollar un proyecto, y con una versión Pro que incorpora herramientas más completas que permiten proyectos de alta calidad. Este motor usa Direct3D y OpenGL en su versión para Windows, y OpenGL en su versión para Mac y Linux.

4.3.2 Motores Gráficos

Los motores Gráficos se encargan del cálculo de todos los polígonos de la escena, de los cálculos de luces y sombras, de todo el aspecto relacionado a las gráficas, tanto su cálculo como de su despliegue. Como ejemplos tenemos los siguientes:

id Tech 1.- Conocido como motor de Doom. Creado por John Carmack y fue un motor que revolucionó la industria en su época.

OGRE.- Motor de visualización de gráficos, es uno de los motores de gráficos más prominentes. Soporta las API de gráficos Direct3D y OpenGL, y se ejecuta en plataformas Windows, Linux y Mac. Se desarrolló en C++ y existen muchos complementos que permiten integrar motores de sonido, física, colisiones, red, inteligencia artificial, etcétera.

Irrlicht Engine.- Motor de gráficos 3D escrito en C++. Funciona en diferentes plataformas como son Mac OS X, Linux y Windows además Xbox, PSP, SymbianOS y el iPhone. Soporta OpenGL, DirectX 8 y 9, OpenGL ES. La comunidad desarrollo interfaces para el SDL, iPhone y SymbianOS.

Aleph One.- Motor de juegos para shooters 3D. Desarrollado por Bungie antes de ser comprador por Microsoft. Se desarrolló con C y su principal plataforma es Mac, Windows y Linux.

Axiom Engine .- Motor de visualización de gráficos en 3D. Desarrollada con C# para ser utilizada con .NET y Mono. Provee una abstracción completa del API 3D, Soporta DirectX y OpenGL, contiene un modelo scene graph y soporta shaders complejos.

4.3.3 Motores de Colisiones y Física

Los motores de colisiones y física, calculan las colisiones que se presenten en el juego, desde las que se dan entre el jugador y los demás objetos, como las que se dan entre los objetos en la escena. Y la reacción de estas colisiones es calculada y mostrada con la parte del motor de física, los motores han llegado a tal punto que antes se calculaban trayectorias predefinidas de acuerdo a las diferentes colisiones, ahora todas las reacciones se calculan al momento, y obtenemos una reacción física acorde a lo que se podría esperar en el mundo real. Como ejemplo tenemos a los siguientes:

Havok Physics.- Es un motor comercial muy popular que se desarrolló en C/C++. Dependiendo del producto que se desea desarrollar, se puede conseguir la versión gratuita o la versión comercial. La versión actual 7.1 funciona en Xbox y Xbox 360; Wii; Sony's PlayStation 2, PlayStation 3 y PlayStation Portable; Linux; y en Mac OS X. Se usó en los siguientes juegos: Too Human, Alone in the Dark, Assassin’s Creed, Bio Shock, Halo, Starcraft 2 y muchos más.

PhysX.- Conocido anteriormente como NovodeX. Actualmente es propiedad de Nvidia y se distribuye de manera gratuita o comercial dependiendo del producto que se desea desarrollar. Funciona en Windows 7, Windows Vista, Windows XP, Mac OS X, Linux, Wii, PlayStation 3, Xbox 360. Permite el acelerar el procesamiento de la física pasando algunos cálculos al GPU, permitiendo al CPU realizar otros cálculos. Se usó en los siguientes juegos: Batman: Arkham Asylum, Mirror's Edge, Unreal Tournament 3, Mafia II y muchos más.

Bullet.- Motor de física de código abierto para objetos 3D, tiene una licencia gratuita zlib. Desarrollado por Erwin Coumans, un ex trabajador de Havok. Se ha utilizado en juegos como: Grand Theft Auto IV, Madagascar Kartz, Regnum Online, etcétera. Además es utilizado en películas.

4.3.4 Motores de Animación

El motor de Animación se encarga de calcular y reproducir todas las animaciones del juego en tiempo real, en lugar de utilizar animaciones predefinidas; esto quiere decir que los personajes, las acciones y reacciones de síntesis son en tiempo real, que son diferentes cada vez, incluso cuando se reproduce la misma escena [1]. Como ejemplo tenemos:

Euphoria.- Euphoria es un motor de animación creado por NaturalMotion basado en la Dynamic Motion Synthesis (Síntesis de Movimiento Dinámico), la tecnología genera animaciones sobre la marcha usando una completa simulación del cuerpo, músculos y sistema nervioso. En lugar de utilizar animaciones predefinidas, los personajes, las acciones y reacciones se generan en tiempo real (Fig. 21).

Havok Animation.- Es un motor de animación desarrollado por Havok. Dependiendo del producto que se desea desarrollar, se puede conseguir la versión gratuita o la versión comercial. Facilita la integración de animaciones con Havok Physics.

Granny.- Incluye exportadores de modelos y animaciones de los paquetes de modelado 3D y animación comerciales, como son: Maya, 3D Studio Max, etc.

Fig. 21. Motor de Animación Euphoria

Como observamos, los motores de juego están compuestos por una gran variedad de elementos, los cuales nos permiten tener un gran número de herramientas en un mismo paquete. También se pudo ver que hay diferentes tipos de motores de juegos, y estas divisiones se hacen de acuerdo a su especialización, a pesar de esta especialización también cuentan con las demás características de un motor de juego estándar en la mayoría de las ocasiones. El motor de juegos nos permite el desarrollo de diferentes aplicaciones, no solo videojuegos, permitiéndonos concentrarnos en otros aspectos de la aplicación sin preocuparnos por desarrollar nuestro propio motor de juegos para que funcione la aplicación.

5. Desarrollo

En este capítulo se propone el desarrollo de una aplicación la cual permitirá al usuario recorrer el edificio principal de la Facultad de Ingeniería de Ciudad Universitaria. Este recorrido virtual estará montado en el motor de juego Unity, e incluirá la opción de localizar el punto de interés deseado, para esto uno introducirá el nombre del punto de interés al que se quiere ir, y una guía le indicara el camino a seguir para llegar a su destino.

Los modelos del edificio principal así como el personaje y el mobiliario se realizaron en el programa 3ds Max. Algunos otros detalles como los correspondientes a las áreas verdes dentro de la facultad se realizaron dentro del Motor de juego Unity, esto debido a que Unity cuenta con un una serie de herramientas para la construcción de terrenos y follaje. Por lo tanto construirlos en un programa externo y para que posteriormente fueran exportados a Unity presentaba un trabajo innecesario. Para la creación de las texturas utilizadas en todo el proyecto se utilizó Adobe Photoshop, debido a la flexibilidad y a las herramientas que proporciona el programa para la elaboración y manipulación de imágenes.

El objetivo de este ambiente virtual es que sea de utilidad para el sistema de inscripciones y como tal, considere la inclusión de la aplicación en su página para que tanto alumnos, académicos e invitados puedan conocer el edificio, previo a su visita a la facultad y por lo mismo no tengan problemas localizando las aulas, o cualquier otro punto de interés para ellos. Cabe mencionar, que el alcance de este proyecto es el desarrollo de la aplicación, queda fuera del alcance del proyecto las respectivas modificaciones que el sistema de inscripciones considere necesarios para su ajuste e incorporación al mismo.

5.1. Análisis de requerimientos

Se plantaron varios requerimientos para la elaboración de este proyecto los cuales fueron:

a) Obtención de información detallada del edificio principal: se tuvo que obtener los planos arquitectónicos para obtener de estos las medidas de la facultad, y con ello construir un modelo virtual a escala real. Para esto se consideró hacer una petición a la facultad de Ingeniería con el propósito de que nos proporcionaran los planos arquitectónicos de los diferentes edificios, pero nos comentaron que era una petición en la que tardaríamos un

tiempo considerable en obtener respuesta, sin garantía de que nos proporcionaran los planos. Así que se optó por nuestros propios medios tomar las medidas utilizando un medidor de distancia laser y flexómetros. Con estas medidas, construimos nuestros propios planos.

También fue necesaria la creación de un archivo fotográfico el cual sirvió para obtener referencias visuales y texturas. Como en la Facultad de Ingeniería hay una política que prohíbe la toma de fotografías de las instalaciones, se tramitó un permiso a la Secretaría de Servicios Académicos de la facultad.

Una vez que nos proporcionó este permiso, se buscó el mejor momento que nos permitiera realizar las mediciones y tomas de fotografías. Se determinó que la mejor opción eran los días sábado, debido a la poca afluencia de gente en la facultad.

b) **Software a utilizar:** Se necesitó un conjunto de programas para la elaboración del proyecto, los cuales consistieron en: un programa de diseño 3D, un programa de manipulación de imágenes y un motor de juegos. Entre las diferentes opciones que existen en el mercado actualmente se eligieron:

- 3ds Max y Adobe Photoshop por la experiencia que adquirimos en este programa durante nuestro servicio social
- Unity: porque cuenta con una gran cantidad de documentación, es fácil de obtener y presenta una gran flexibilidad para desarrollar aplicaciones en múltiples plataformas.

c) **Presentación de la aplicación:** Se decidió que la aplicación fuera un recorrido virtual 3D, debido al atractivo que generan este tipo de aplicaciones. Para complementar el atractivo visual optamos por que el recorrido fuera realizado en tercera persona. El personaje a utilizar fue Goyo, la mascota de la UNAM, debido a que es un icono muy popular en la comunidad universitaria.

d) **Programación:** El recorrido virtual necesitó de una programación que se encargue de manejar los eventos que se quieran realizar dentro de él. En Unity esta programación se hace por medio de scripts. La programación se dividió en dos apartados, que fueron:

- La cámara y el control del personaje
- El algoritmo de búsqueda de las aulas y puntos de interés.

Para la programación de los scripts se utilizó el lenguaje C#. Esto se debe a que Unity da un mejor soporte a este lenguaje, ya que aunque maneja javascript, la versión que utiliza

Unity es un caso particular de javascript; es decir no se usa una versión estándar de este lenguaje.

Con este conjunto de requisitos en mente procedimos a diseñar los elementos que conforman la aplicación.

5.2. Diseño de la aplicación

Para el desarrollo de este proyecto se identificaron dos divisiones, la división correspondiente al modelado, y la correspondiente a la programación. Cada división se conformó por diversos módulos.

La división de modelado estuvo conformada por:

- Módulo del escenario.
- Módulo del personaje.

El modulo del escenario comprende los edificios A, B, C, D y E que conforman el complejo del edificio principal. Para llevar un buen flujo de trabajo se modelaron dos conjuntos de edificios de forma independiente; una persona se encargó de modelar los edificios A y D, mientras que otra persona modeló los edificios B, C y E. Se diseñó de esta forma porque dichos edificios se encuentran conectados entre sí, por lo tanto no seguir este diseño (es decir que una persona modelara el edificio D y otra el A) traería como consecuencia incoherencias al momento de conjuntar los modelos en un solo archivo, porque es común que haya variaciones al tomar medidas y al construir los modelos; estas variaciones pueden ser por errores de exactitud y precisión en los instrumentos de medición y error humano.

Se procedió a analizar que partes eran requeridas para la aplicación. Tomando en cuenta que el objetivo es la localización de las aulas, determinamos los puntos a modelar. Con base en este requerimiento se determinó los detalles que no eran necesarios, y que ahorrarían tiempo de cálculo al motor de juegos, lo cual permitiría que el framerate[3] del recorrido virtual no sufriera bajas notables, y permitiera transiciones fluidas.

[3] Término que se refiere a la cantidad de cuadros o imágenes que son desplegados en pantalla, cada segundo.

De las partes que no se modelaron, las más notables son los laboratorios del Edificio D, si bien, las entradas a los laboratorios con sus respectivos letreros están presentes, no está presente la maquinaria de dichos laboratorios. El hecho de modelar el equipo de laboratorio requería de una gran cantidad de tiempo dedicado al proyecto. Representaba también una carga significativa para el motor de juegos, debido a la cantidad de geometría que se tendría que haber usado. Aun manteniendo un contenido bajo de polígonos, al ser varios objetos el conteo poligonal se elevaba, y esto provocaba que el framerate bajara en esta área. Y teniendo en cuenta que el requerimiento es mostrar la localización de las aulas, y no es como tal mostrar las aulas y su interior, se decidió que era innecesario el modelado de las mismas.

El módulo del personaje consiste en todo el procedimiento que se tiene que seguir para obtener un personaje que va desde un modelo estático hasta uno completamente animado. Para la creación del personaje se debe llevar un orden específico, este orden es:

- Creación de la geometría del personaje.
- Texturizado del personaje
- Rigging del personaje
- Skinning del personaje
- Animaciones del personaje
- Cocinado de las animaciones

La división de programación se conformó por:

- Módulo de control de cámaras y personaje.
- Módulo de la Interfaz Gráfica (GUI).
- Módulo de búsqueda.

El módulo de control de cámaras y personaje consiste en el conjunto de algoritmos que van a permitir manipular la cámara y al personaje con el mouse y el teclado respectivamente. Este módulo tuvo un desarrollo por etapas, cada etapa complementando a la anterior, sus etapas fueron:

a) Funcionalidad Básica: en esta etapa se implementó el movimiento de una primitiva mediante el uso del teclado y una cámara que tuviera la capacidad de orbitar alrededor de esta.

b) Funcionalidad Mejorada: se agregaron aspectos de mayor complejidad como lo fue la gravedad, la colisión y oclusión de la cámara con el escenario

c) Incorporación del personaje: en esta última fase se agrega el personaje riggeado y animado y se crea el código necesario para que la animación correspondiente al comando recibido sea reproducida.

En el módulo de la interfaz gráfica se crearon los algoritmos correspondientes para que los diferentes cuadros de dialogo sean desplegados según la opción seleccionada por el usuario. Estos cuadros de dialogo consistieron en:

- Menú de Inicio.
- Menú de pausa.
- Menú de búsqueda.

En el módulo de búsqueda se desarrolló el algoritmo que se encargara de encontrar el aula que el usuario seleccione. Una vez obtenida la localización del aula, el algoritmo pone en marcha el pathfinding de Unity, con el cual se traza la ruta desde donde está el personaje hasta donde se encuentra el aula deseada.

Finalmente se tendría una aplicación en la cual el usuario pueda realizar cualquiera de las acciones mostradas en el siguiente diagrama:

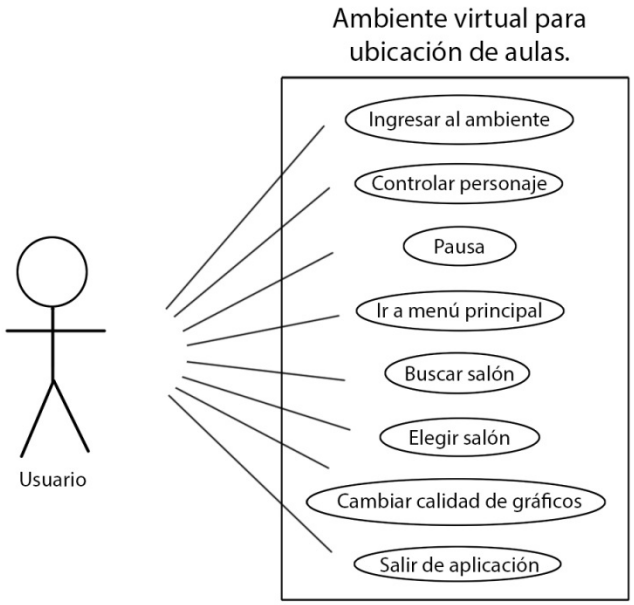


Fig. 22. Diagrama de Casos de Uso

5.2.1 Metodología de Ingeniería de Software

Durante el proceso de desarrollo del proyecto se usó el método de desarrollo en cascada. Este se usó después de una extensiva investigación acerca de los métodos usados en el ámbito de los recorridos virtuales y los videojuegos. Si bien lo que normalmente se usa es una variación del método en cascada, al no encontrar más información específica acerca de esta variación, se decidió usar el método tradicional.

Si bien el método en cascada nos impide un avance rápido en el proyecto, este método fue el que más se acopló a las necesidades del proyecto; ya que el método en cascada nos impide seguir a la siguiente fase, si la fase anterior no se ha terminado por completo. Y el desarrollo de esta aplicación se realizó de esa manera, esto se debe a que en el proyecto no podíamos empezar una fase nueva sin antes tener todo lo de la fase anterior terminado. No podíamos hacer el modelo, sin antes tener los planos y las fotografías; no podíamos realizar las animaciones del personaje sin antes tener el modelo y el rigging de todo el personaje; no podíamos hacer la importación de los objetos a Unity, sin que estos estuvieran terminados por completo, la razón de esto es que causaría muchos problemas cargarlos por separado porque podría haber inconsistencias en los objetos, que nos obligarían a regresar a la etapa anterior.

Por estos motivos se optó por el método de cascada, ya que nos presentaba mayores ventajas y la obtención de una aplicación de mayor calidad, que en el caso de los otros modelos. En los puntos anteriores tratados en este capítulo pudimos observar el proceso de la metodología en cascada

5.3 Construcción de la aplicación

5.3.1 Etapa de Modelado

Para la estructura en general, se usó la técnica de box modeling. Se escogió esta técnica por el hecho de que las estructuras tienen formas geométricas definidas, y es relativamente fácil obtener estas formas a partir de una figura geométrica estándar (Ver Fig. 22.1). Y en algunos casos, no se tuvo que modificar la figura, como en el caso de las columnas simplemente se utilizó un cilindro, y solo se le aplicó la textura necesaria.

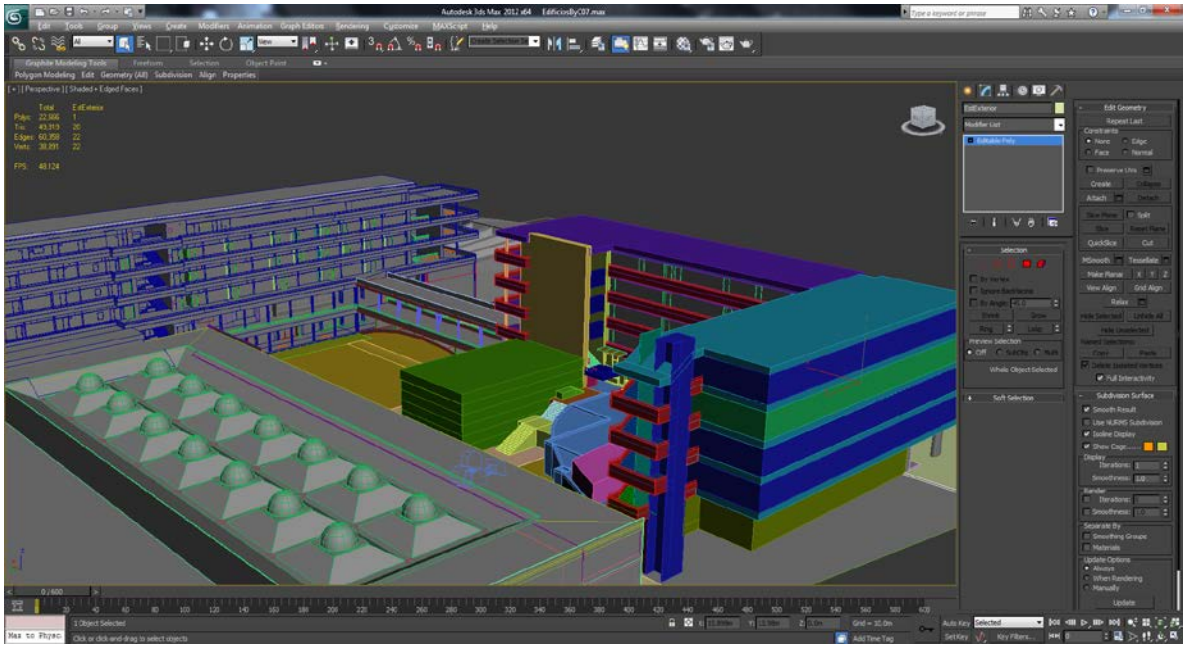


Fig. 23.1 Podemos observar la forma del edificio principal, al estar compuesto de formas geométricas básicas, el box modeling fue la mejor técnica para el modelado.

El box modeling fue la técnica que se utilizó en la mayor parte del proyecto, no solo se usó para la estructura del edificio, también se usó para el modelado de los bustos de las estatuas, de las bancas y de algunos otros detalles, tomando en cuenta que las normales son importantes para el motor de juego y como se explicó en el apartado de box modeling esta técnica permite resultados consistentes que poseen aproximadamente la misma dirección de las normales, de esta forma no hubo necesidad de correcciones mayores en los modelos al cargarlos en Unity y que tuviera problemas interpretando las normales.

Para modelar el estacionamiento interior junto con las jardineras, se usó la técnica de Extrude (Fig. 22.2), tomando en cuenta la forma del estacionamiento, fue mejor trazar con splines[4] el modelo y luego aplicar el Extrude en lugar de tratar de modelarlo a partir de un cubo o plano y manipular vértice por vértice para llegar a la forma deseada.

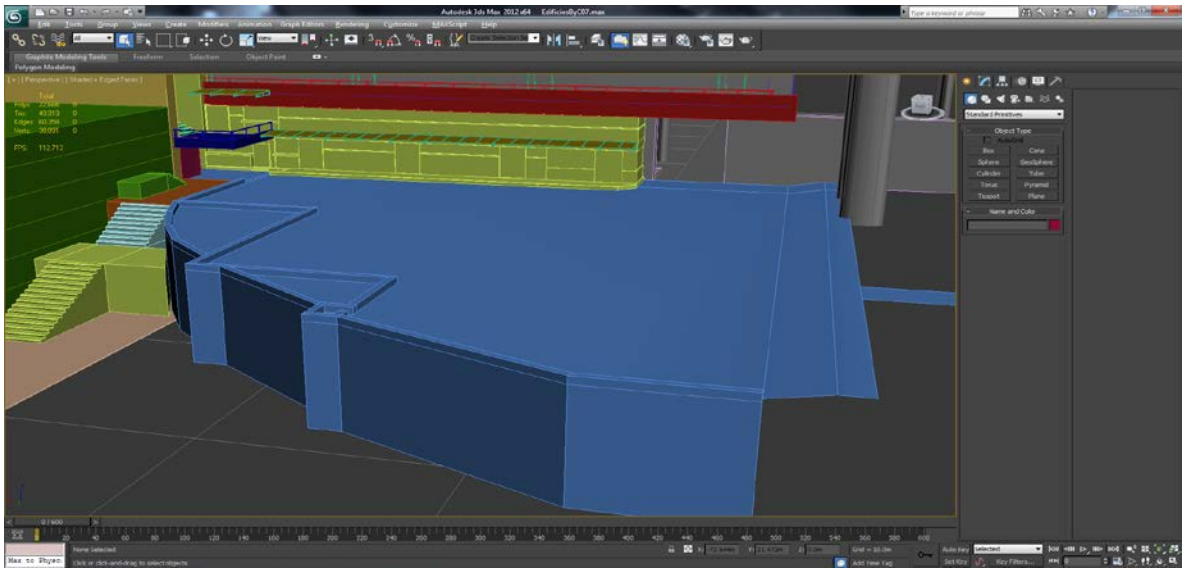


Fig. 23.2 Observamos la forma del estacionamiento con sus jardineras

Las operaciones booleanas, se usaron con muy pocos objetos, debido a que incrementa el conteo de polígonos, entonces se evitó su uso a menos que fuera absolutamente necesario (Fig. 22.3).

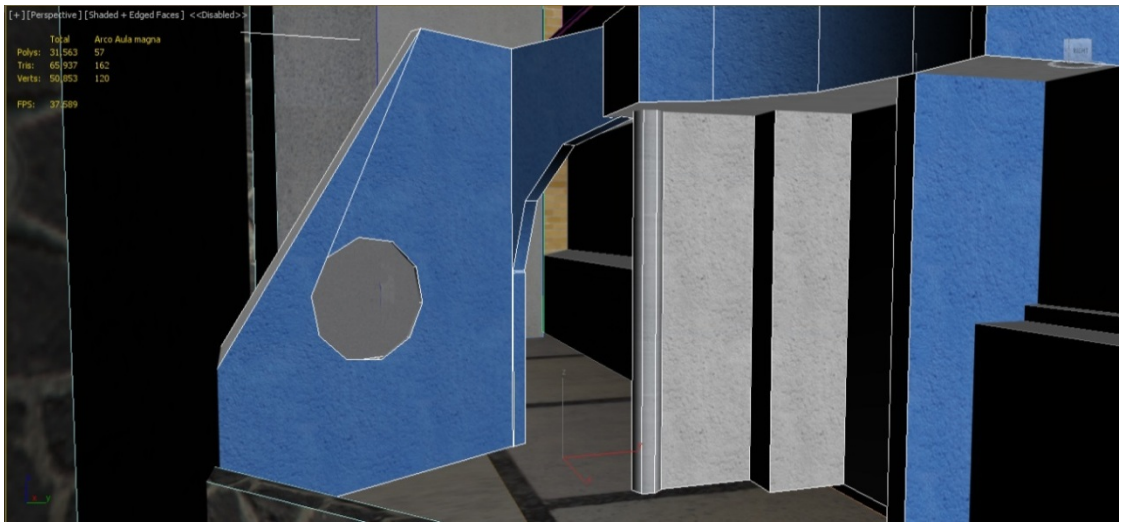


Fig. 23.3. La entrada del Aula Magna es un ejemplo de un objeto modelado con operaciones booleanas.

[4] Un spline es una curva que pasa a través de un conjunto de puntos definidos, esto la convierte en una curva definida en porciones. Teniendo una curva independiente entre punto y punto [6].

Una vez que se tuvieron los objetos modelados, se crearon las texturas para dichos objetos. Las texturas fueron obtenidas de las diferentes fotos que se tomaron de la facultad, esto con el objetivo de lograr que la textura fuera lo más realista posible. Teniendo en mente que se iban a obtener las texturas de las fotos, cuando fueron tomadas las fotos se buscaron las condiciones más adecuadas que nos permitieran obtener texturas de buena calidad. Estas condiciones fueron:

- Evitar áreas sombreadas en la foto
- Evitar reflejos en superficies reflectantes
- Evitar distorsión en las fotos, ya fuera el ángulo en el que se tomó la foto, o disminuir el efecto que en ocasiones genera el lente de la cámara
- En caso de texturas que puedan generar mosaicos para su fácil repetición, buscar la parte que tuviera mejor vista para hacer éste mosaico.
- Capturar imperfecciones como grietas, o partes desgastadas, debido a que las imperfecciones proveen un mayor realismo al objeto una vez texturizado

Mientras que algunas texturas se obtuvieron sin problemas de las fotos, otras presentaron un poco más de dificultad. Para estas texturas se obtenían diferentes partes de diferentes fotos, una vez en Photoshop pasamos a aplicar la edición necesaria. Algunas solo requerían retoques básicos, otras se les necesito pasar algunos filtros o algunos otros detalles para obtener texturas de alta calidad. En algunas no solo bastó con los filtros y las múltiples fotos, en algunos casos se tuvo que dibujar la textura, ya fuera porque la obtención de esos detalles probaba ser de mucha dificultad en las fotos tomadas, o al dibujar la textura se obtenía un mayor detalle del que se podía obtener de cualquier foto.

Una vez que se obtuvieron las texturas, se crearon a partir de ellas los diferentes mapas para darle mayor detalle a los objetos (Fig. 22.4). Algunos mapas de normales obtenidos se reusaron para diferentes texturas, que si bien las texturas podían ser algo diferentes, las bases de la textura eran las mismas, como en el caso de las puertas.

Finalmente se tuvo las siguientes características en el modelo:

Polígonos: 32 063

Triángulos: 67 937

Texturas: 330

Fig. 23.4. Del lado izquierdo tenemos la textura difusa del hidrante, la parte rosa es un canal alfa. Del lado derecho tenemos su mapa de normales.

Para algunas texturas se usaron canales alfa, de esta forma, evitamos hacer más geometría. Un ejemplo de esta situación es con las puertas de los salones que tienen ventana (Ver Fig. 22.5). Si se hubiera quitado esa parte de la puerta en la geometría, se tendrían más polígonos, la puerta es un plano, como tal tiene solo 2 triángulos, si se hubiera hecho el hueco de la ventana, se tendrían más triángulos ya que tendríamos un cuadro dentro de otro; aunque no estuvieran presentes los triángulos de la ventana, si estarían presentes los vértices. Si tomamos esto en cuenta y el hecho de que son muchas puertas las que se modelaron, el conteo de triángulos y por ende de polígonos se incrementaría.

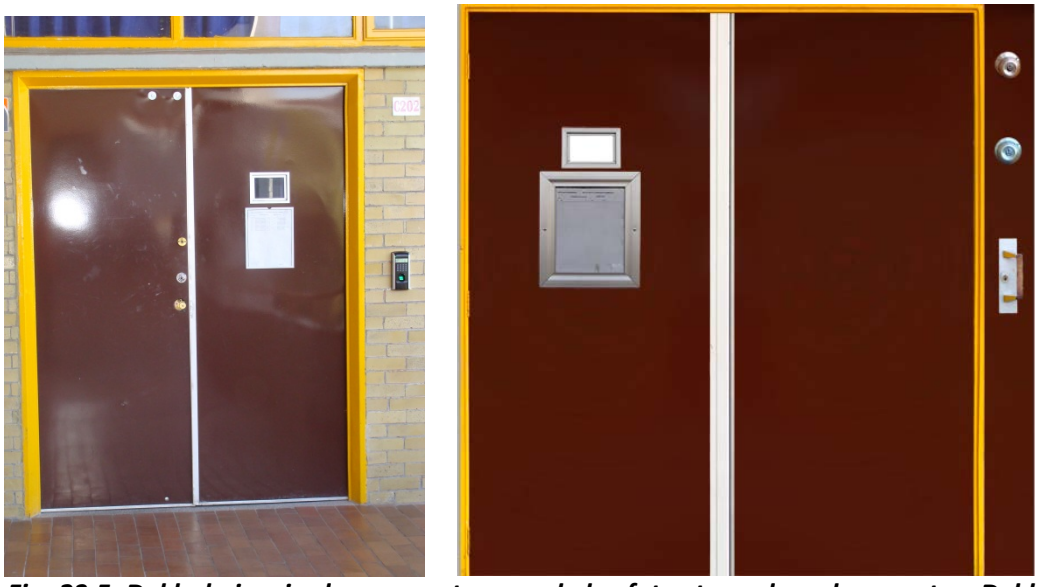


Fig. 23.5. Del lado izquierdo se muestra una de las fotos tomadas a las puertas. Del lado derecho la textura que se obtuvo para las puertas.

Teniendo las texturas y sus respectivos modelos, comenzó el proceso de texturizado. Para este proceso se usó el método de UV mapping. En algunos de los objetos la parte de la texturizada fue sencilla, debido a las formas básicas de las que estaban hechos, en el caso de las columnas simplemente se hizo un mapeo para un cilindro (una opción predefinida en 3Ds Max), y de esta forma el objeto quedo texturizado. Para otros objetos no resulto tan sencillo, y se usó el comando Unwrap UVW (Figs. 22.6 y 22.7), de esta forma se desenvuelve el polígono y uno ajusta los vértices a la textura, que es el equivalente para el programa del método UV mapping.

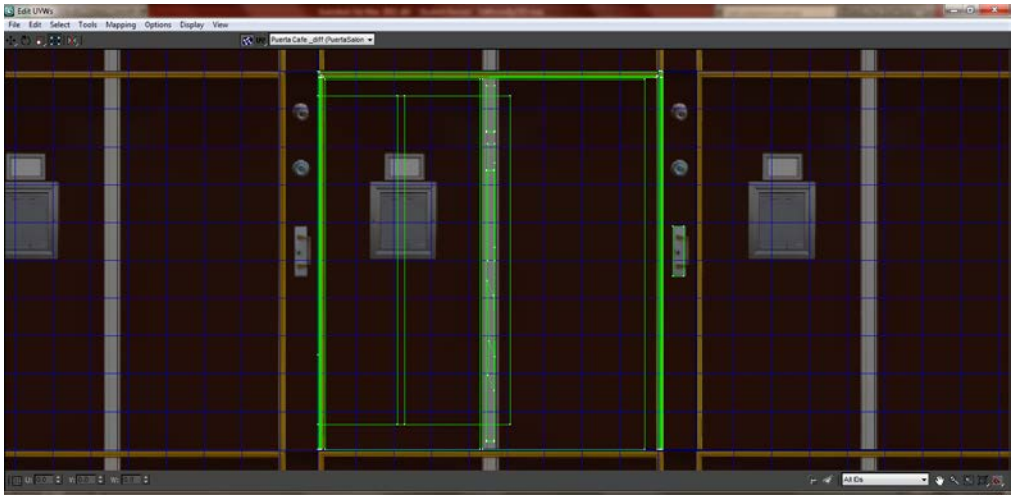


Fig. 23.6. En la figura las líneas verdes son los polígonos, los cuales se acomodaron a la textura que se le aplico al objeto

Fig. 23.7. Aquí podemos observar la puerta ya texturizada

De ésta manera se realizó todo el modelado del edificio principal de la Facultad de Ingeniería. Antes de realizar su carga en el motor de videojuegos, se pasó a la etapa de optimización del modelo, la cual no tuvo mayor problema, ya que desde el principio se tomó en cuenta el número de polígonos que tenían los objetos. Pero aun teniendo esto en cuenta, se volvió a revisar el modelo buscando zonas que presentaran un alto conteo de polígonos, y una vez localizadas, se procedió a eliminar polígonos considerados innecesarios, los cuales pudiera ser que se pasaran por alto en la primera etapa, o simplemente se consideraron necesarios en las primeras etapas del proyecto, y al tener el modelo terminado se descubrió que o eran innecesarios esos polígonos, o que se reemplazaban con una textura y su respectivo mapa de normales, logrando un menor conteo de polígonos pero manteniendo la misma calidad del modelo.

El personaje para la aplicación se modelo de forma independiente al edificio principal. A lo largo del desarrollo se realizaron dos versiones del mismo personaje. La primera versión fue un modelo muy básico, con un muy bajo conteo de polígonos, el cuál se utilizó solamente para propósitos de referencia. Y la segunda versión fue el modelo definitivo que se utilizó en la versión final del proyecto. (Fig. 22.8)

El rigging y la animación del personaje se realizó en 3Ds Max. Mediante el uso de animación esqueletal y keyframes[5] se hicieron 9 animaciones para acompañar a los 9 estados con los que cuenta el personaje, y que son:

- Estado inactivo 1
- Estado inactivo 2
- Correr
- Saltar
- Caer
- Aterrizar
- Movimiento a la derecha
- Movimiento a la izquierda
- Movimiento hacia atras

[5] Keyframe se refiere a los fotogramas clave usados en la animación mediante keyframes. Estos fotogramas clave nos proporcionan los atributos de un objeto en un instante de tiempo (posición, tamaño, rotación, etc.); en la animación por keyframes se definen un conjunto de keyframes en el tiempo y el software encargado de la animación se encargara de “rellenar” el espacio de tiempo entre los keyframes definidos para dar la sensación de movimiento.

Una vez que se tuvieron los diferentes clips de animación, se exportó el personaje en formato Fbx, en este proceso las animaciones se juntan de modo que quede un solo clip que contiene todas las animaciones; estas se aplican o “cocinan” al modelo, de tal forma que quedan predefinidas como parte de la geometría, es decir, el modelo no puede realizar otra animación que no sean las que se incluyeron en el clip.

Fig. 23.8. El personaje del recorrido tuvo dos versiones diferentes, como podemos observar la versión 2.0 es la más reciente.

Con el escenario finalizado se procedió a cargarlo en Unity. Todo el modelo del Edificio Principal esta medido en metros, sin embargo Unity, al momento de importar objetos maneja un factor de escala de 0.01 por unidad. Esto significa que Unity multiplicará por 0.01 cada unidad del modelo importado, lo cual nos dará un objeto 100 veces más pequeño que el que se modelo en 3ds Max.

Existen 2 soluciones para este problema:

1. Cambiar el factor de escala dentro de Unity de 0.01 a 1.0
2. Exportar el objeto cambiando el factor de escala a 100.0

En el caso de objetos estáticos, no influye el hecho de cambiar el factor de escala dentro de Unity. Sin embargo esto si influye de manera importante cuando se importa un objeto animado, como lo es el personaje, causando que sus animaciones se alteren drásticamente, y en varias ocasiones que la geometría colapsara. Por esta razón se optó por seguir la segunda opción.

Una vez que se cargaron todos los modelos (Fig. 22.9), se le agregaron sus cajas de colisiones. Estas cajas son las que calculan la colisión entre objetos, estos cálculos los realiza el motor de juego y despliega los efectos de la colisión en tiempo real. Debido a la naturaleza del recorrido virtual, no hay cálculos complicados que tenga que realizar el motor de física, solo necesita

calcular la colisión con el personaje, y así evitar que el personaje atraviese las paredes, o atraviese el piso en el que está parado.

Para el personaje, una vez dentro de Unity, se le asignó una capsula de colisión y su clip de animaciones se dividió en pequeños clips que contenían cada movimiento por separado. Mediante un script se asocian estas animaciones con las acciones que se le indiquen al personaje de acuerdo a las teclas que el usuario presione, creando así la sensación de que el personaje camina y salta, cuando en realidad el personaje solo se desplaza de un lado a otro y lo demás sucede gracias a la repetición de los clips de animación

Para la iluminación del escenario se utilizó una luz direccional. Una luz direccional es aquella que simula una fuente luminosa muy grande y que se encuentra increíblemente lejos. Es por esta razón que su posición en el escenario es irrelevante. Sólo cuando se le aplica una rotación, es que la luz emitida por esta fuente cambia según el ángulo en que se encuentre. El uso más común que se le da a una luz direccional es para simular al Sol, y este caso no fue la excepción.

Como el escenario se construyó teniendo en mente que sería de día y no habría un ciclo día/noche, no fueron necesarias otras fuentes de luz. La única excepción fue el Edificio D y la sección de oficinas del Edificio A. Ya que estas son zonas donde hay muy poca luz, se usaron luces omnidireccionales para simular la iluminación emitida por las lámparas de estos edificios.

Una vez que se colocaron las luces y se ajustaron adecuadamente, se procedió a generar los light maps correspondientes. La construcción de light maps es una muy buena forma de optimizar el rendimiento de la aplicación ya que, las luces y sombras quedan dibujadas en la textura de los objetos y así no es necesario calcularla en tiempo real. Este método sólo funciona para objetos estáticos por lo tanto para objetos dinámicos, como el personaje, se requiere que la iluminación sea calculada en tiempo real.

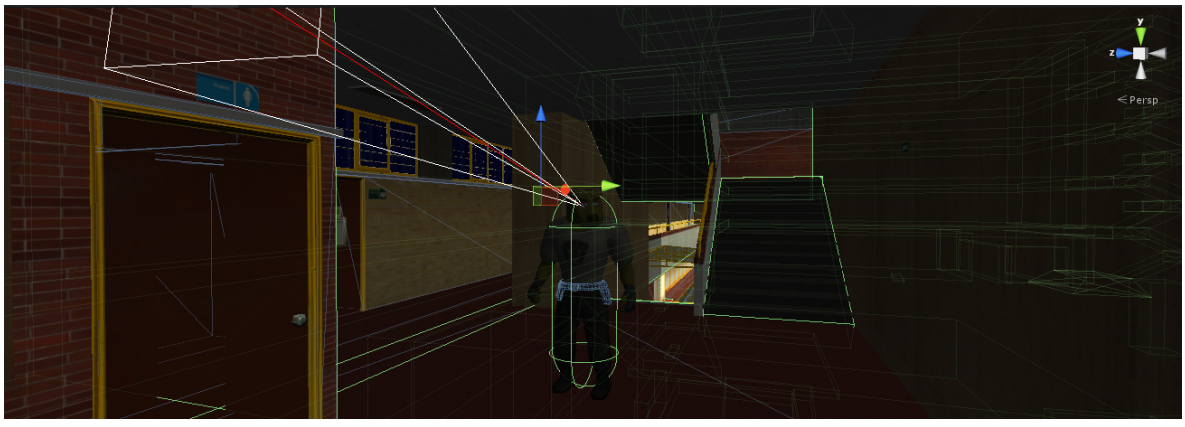


Fig. 23.9. Ejemplo de las cajas de colisión del escenario y del personaje.

5.3.2 Etapa de Programación

El motor de juegos nos permitió un buen manejo de los objetos que no integrados a la estructura principal del edificio (Props). Por ejemplo, las bancas de concreto del edificio, el busto de Barros Sierra, y los letreros de las puertas se consideran props. Y no solo un buen manejo de estos, también un manejo de todos los elementos que forman la escena, incluyendo luces, cámaras, y los modelos.

Se realizó el scripting en C#. Se construyó un sistema de cámara en tercera persona con movimientos de seguimiento fluidos.

Primero se elaboró la funcionalidad básica de la cámara en tercera persona. Esta incluye: mover a un personaje no animado o un objeto y tener una cámara que lo siga y pueda girar alrededor de este.

La funcionalidad básica del sistema de cámaras se conformó de 4 clases:

TP_Controller: Esta es la clase principal. Se encarga de registrar los comandos introducidos por el usuario. Convierte las instrucciones de movimiento a un vector en tres dimensiones. También se encarga de revisar constantemente si existe una cámara en el escenario, si por alguna razón la cámara deja de existir, el programa se detendrá.

TP_Motor: Procesa la información de movimiento que brinda TP_Controller y con ello mueve al personaje por el escenario. Asimismo rota al personaje con relación a la posición de la cámara, es decir el personaje siempre se moverá alejándose de la cámara.

TP_Camera: Registra la información mandada por el usuario desde el mouse, y con ella moverá la cámara alrededor del personaje. La razón por la cual no lo hace TP_Controller es para que, si se desea, se pueda asignar esta clase a una cámara sin personaje; y se pueda manipular dicha cámara independientemente. Se encargará de crear una cámara, si TP_Controller no detecta una, o de tomar el control de una ya existente.

Helper: Esta fue una clase de apoyo. Para la funcionalidad básica esta clase sólo se encargó de acotar los ángulos de rotación, para manejar un rango entre 0 y 360 grados respecto al eje Y . Además de acotar la rotación de la cámara respecto al eje X para que no pueda dar una vuelta completa y evitar errores de orientación en el control del personaje.

En la funcionalidad mejorada se agregaron aspectos como la gravedad, habilidad de saltar, colisiones de la cámara con el escenario y evitar la oclusión de la cámara con los objetos.

Se añadieron métodos a nuestras clases :

TP_Controller: Se añadió el reconocimiento de salto para que cuando se presione la tecla indicada esta clase llame a todos los elementos involucrados en el salto. Como el movimiento vertical, la animación de salto, la aplicación de la gravedad a lo largo del salto.

TP_Motor: Se añadió el método que aplica la gravedad al personaje y que implementara el movimiento de salto

TP_Animator: Se creó esta nueva clase la cual definirá un estado según la acción que se está realizando. De acuerdo con cada estado se define una velocidad de movimiento diferente, por lo que la velocidad no será la misma si nos estamos moviendo hacia adelante, que la velocidad hacia atrás o hacia los lados.

TP_Camera: Se añadió la capacidad de evitar obstáculos que interfirieran entre el personaje y la cámara. Esto se logró viendo el problema de tal forma que la detección de colisiones y la detección de la oclusión de la cámara se trataran como un solo problema en lugar de tratarlos como 2 problemas por separado. La solución fue plantear el problema bajo la siguiente condición: ¿Puede el jugador ver a la cámara?

Bajo esta premisa tenemos un solo problema, ya que si el jugador no puede ver al a cámara, lo que se tendrá que hacer es acercar esta última y así lo resolvemos parcialmente. La otra mitad del problema consiste en que se debe de tomar en cuenta el “Near Clipping Plane” de la cámara ya que si no se hace esto en ocasiones este plano penetrará la geometría y dará una imagen indeseada a pesar de que la cámara no esté traspasando la geometría. Para resolver este problema simplemente aplicamos la solución anterior a los 4 puntos que conforman el Near Clipping Plane.

En la última parte se añadió la última funcionalidad a la clase TP_Animator , esta se encarga de ejecutar las animaciones según el estado en el que se encuentre el personaje. Por ejemplo, si el personaje está caminando, se reproducirá la animación de caminar. Para que los cambios entre una animación y otra se vuelvan imperceptibles, se hace uso de una transición entre animaciones o “blending” que ofrece Unity.

En la figura 22.10 se muestra un diagrama de las diferentes clases utilizadas para la construcción del sistema de cámara en tercera persona. Asimismo en el Apéndice 8.1x de esta tesis se da una muestra del código utilizado.

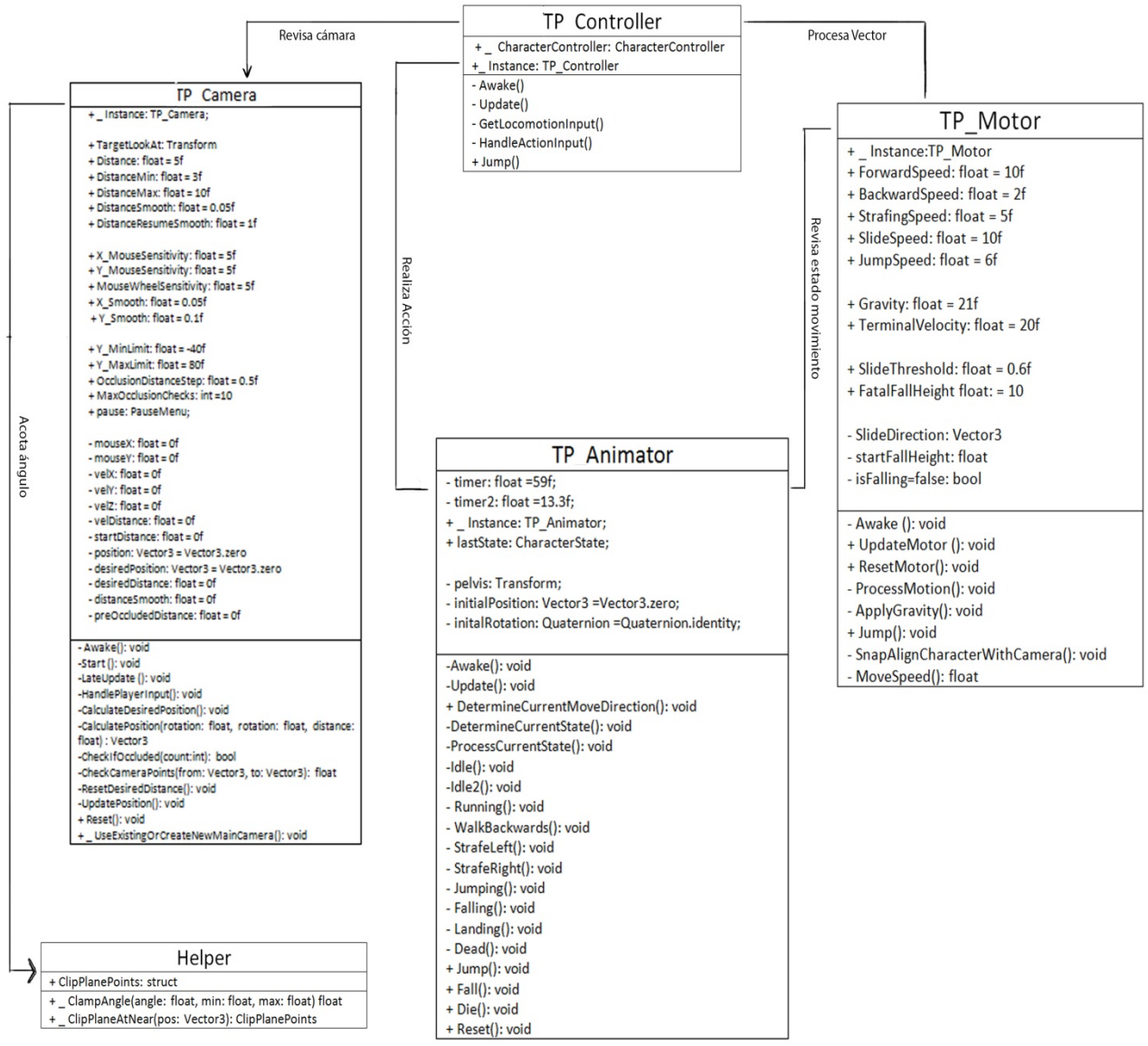


Fig. 23.10. Diagrama de clases para el módulo de control de cámara y personaje

El control para el menú principal constó de tres clases (Ver fig. 22.11)

MenuScript: La clase principal que se encarga de revisar si se ha hecho click sobre algún objeto con el cuál se pueda interactuar en la escena. De ser así ejecutará la acción necesaria, como reproducir alguna animación, o indicar que se debe cargar el nivel.

CameraFade: Esta clase recibe información de MenuScript y realiza la carga del nivel además se encarga de hacer un oscurecimiento en la cámara para hacer menos evidente el tiempo de carga del nivel.

Spawner: Esta clase recibe información de MenuScript y su única tarea es mover al personaje a uno de los dos puntos de inicio en la escena, cuando esta cargue, dependiendo el que se haya escogido en el menú principal.

En el Apéndice 8.2 de esta tesis se da una muestra del código utilizado.

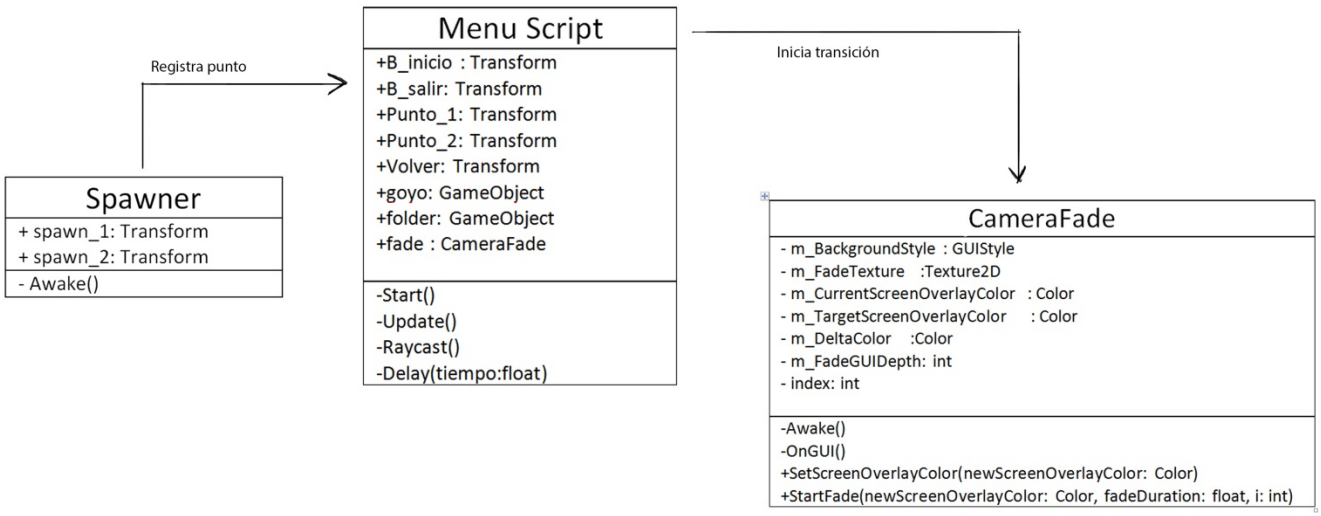


Fig. 23.11. Diagrama de clases para el módulo del menú principal

Por último el sistema de búsqueda se conformó por dos clases como se muestra en la figura 22.12.

PauseMenu: Esta clase es la encargada de desplegar el menú de pausa y el menú de búsqueda según la tecla que se haya presionado. Para el caso del Menú de pausa se definieron las siguiente opciones:

- Regresar al menú Principal
- Cambiar el nivel de detalle de los gráficos
- Salir de la aplicación

Para el caso del menú de búsqueda, se registrara la cadena de caracteres ingresada por el usuario se enviara a la clase Búsqueda.

Búsqueda: Esta clase recibe la cadena de caracteres enviada por la clase PauseMenu y , como su nombre lo indica, buscará esta cadena entre todos los objetos de la escena, marcados como "letrero". Si hay coincidencias se mostrarán y se llamara al Pathfinding de Unity para que trace la ruta correspondiente a la opción seleccionada por el usuario. De no haber coincidencias se mostrara un mensaje de error y se esperara a que el usuario ingrese otro nombre.

En el Apéndice 8.3 de esta tesis se da una muestra del código utilizado.

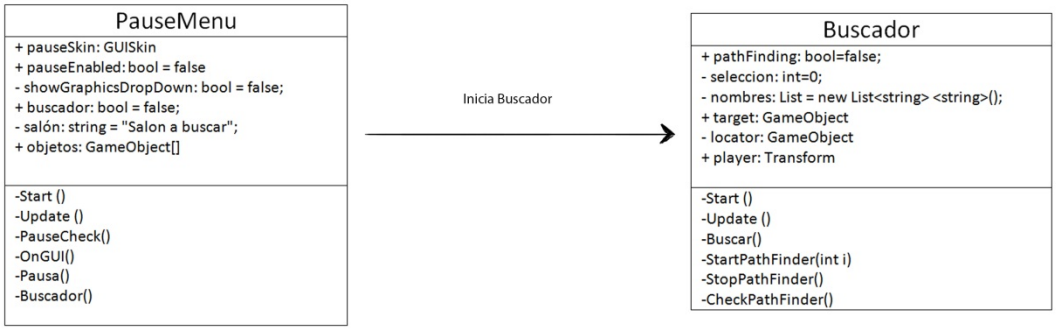


Fig. 23.12. Diagrama de clases para el módulo del menú de pausa y búsqueda

5.4 Fase de pruebas y solución de problemas

Una vez que se cargó todo el modelo y se escribieron todos los scripts, se pasó a la etapa de pruebas de la aplicación, se anotaron los errores que se obtuvieron y se observó en que partes se podía optimizar la aplicación, ya fuera que necesitara optimización en la parte de modelado, o que se pudiera optimizar con la programación de más scripts.

Se realizaron pruebas de caja negra, pruebas unitarias y de integración sobre un equipo portátil Asus con las siguientes especificaciones:

- CPU: Intel Core i7 a 1.66 GHz
- 6GB de memoria RAM

- GPU: ATI Mobility Radeon HD 5870
- Sistema operativo: Windows 7 Home Premium 64 bits con actualización KB958488

Se realizaron pruebas de caja negra a los siguientes módulos:

Colisiones con el escenario: Esta prueba consistió simplemente en recorrer todo el escenario, cada rincón, pasillo, escalera, muro etc. En busca de un sitio en el que no hubiera colisión ya fuera porque se omitió o porque hubiera algún problema con las primitivas. Se encontraron pequeños problemas como puntos donde el personaje ya no podía acceder debido a una colisión que accidentalmente se había puesto. No hubo mayor problema que ajustar las primitivas para corregir estos errores.

Pathfinding: Se verificó que se trazara un camino adecuado que el personaje pudiera seguir. Se creó un objeto de prueba y se colocó en diferentes partes del escenario. En todos los casos se obtuvo una ruta adecuada para el personaje, siempre y cuando el objetivo se encontrara cerca de la malla de navegación.

Menú principal: Se comprobó que al hacer click en los objetos correctos de la escena se ejecutaran los eventos asignados. Sobre todo en las partes en las que un objeto se encontraba encima del otro. Se probó seleccionando objetos que no tuvieran ninguna acción asignada, se realizaron pruebas haciendo click derecho y central. En ningún caso se obtuvieron resultados que no fueran activados con el izquierdo, obedeciendo así las reglas establecidas

Se realizaron pruebas unitarias a los siguientes módulos:

Sistema de cámaras y control de personaje: Como éste módulo se fue mejorando, era necesario identificar los errores que pudieran existir en la fase actual antes de implementar la siguiente.

En primera instancia se probó este sistema en una escena aparte, la cuál era mucho más simple. Se probó que el personaje realizara la acción asignada a cada tecla del teclado, también se probó la correcta orientación del personaje al moverse por el escenario; la cuál debería de ser siempre alejándose de la cámara, según se definió. En cuanto al control de la cámara se revisó que esta respondiera a los movimientos del mouse y que obedeciera a las restricciones que se le asignaron, como fue no poder girar sobre el eje X más allá del ángulo definido.

Las pruebas para la segunda fase del sistema de cámaras se realizaron dentro del escenario principal. Se revisó la cámara detectara las colisiones con el escenario, de la misma manera que al detectar una colisión la esta se moviera para no perder de vista al personaje. Se colocó al personaje en una serie de ángulos muy cerrados, donde la cámara tuviera muy poco espacio para moverse, como en pasillos pequeños o las escaleras de los edificios. Se obtuvieron buenos resultados ya que no se pudo obtener una situación donde la cámara atravesara partes del escenario que no debiera.

En la fase final se probó que las animaciones del personaje se ejecutaran correctamente y que su transición fuera lo más imperceptible posible. Se movió al personaje por todo el escenario, realizando todas las acciones posibles (correr, saltar, caer, caminar hacia atrás y en diagonal) de manera aleatoria para ver todas las posibles combinaciones en la transición de los movimientos. Fue en esta prueba donde se obtuvieron los siguientes problemas:

- Problema con la fluidez en el movimiento del personaje: El problema ocurría cuando el personaje bajaba las escaleras del edificio o cuando caía de una distancia muy pequeña, como una irregularidad en el terreno o una pequeña plataforma. Al hacerlo se ejecutaba la animación de caída y arruinaba la fluidez en el movimiento del personaje.
  La solución fue agregar una condición en el código encargado de ejecutar la animación de caída. Cuando la velocidad de caída del personaje fuera muy pequeña, se ignoraría el comando que activa la animación y se seguiría ejecutando la animación previa.

- Problema en la transición entre animaciones del personaje: Este error ocasionaba una mala transición entre las animaciones del personaje cuando había un cambio brusco entre las acciones del mismo, tales como el levantarse de una caída, un cambio brusco de dirección o el cambio entre la animación de saltar y caer.
  Para resolver este problema se acotaron los tiempos de las animaciones en un par de cuadros/segundo. Así se tuvieron animaciones ligeramente más cortas cuyo cambio era prácticamente imperceptible a la vista, pero la transición entre una y otra mejoraba notablemente.

Menús de Pausa y Búsqueda: Se hicieron pruebas para asegurar que, sin importar lo que estuviera pasando en la escena, cuando uno presionara la tecla asignada al menú de pausa o al menú de búsqueda toda la acción se detuviera y se desplegara el menú correspondiente. También se verificó que un menú no permitiera el despliegue de otro sin que antes se cerrara el primero, es decir si el menú de pausa estaba activado, no se pudiera desplegar el menú de búsqueda hasta que se hubiera quitado la pausa.

En el caso del menú de búsqueda se verificó que realmente encontrara el nombre del salón ingresado por el usuario, y que regresara un mensaje de error si dicho nombre no fuera encontrado. Para esto se probó con varias combinaciones de nombres en diferente orden. Al final se obtuvo un resultado satisfactorio, ya que el buscador regresó una lista de aulas que contienen por lo menos una de las palabras del nombre del salón a buscar, lo cual es bueno ya que no hay garantía de que el usuario introduzca el nombre exacto del aula. Para comprobar que realmente el objeto encontrado fuera el objeto deseado, se obtuvo su coordenada dentro del escenario para que se mostrara en la consola de depuración de Unity, y se comparó con su coordenada dentro del editor.

Las pruebas de integración se realizaron sobre los siguientes módulos:

Buscador y Pathfinding: Se buscó encontrar fallas en el posicionamiento del objeto buscador cuando el personaje se encontrara en una ubicación poco probable, como pudiera ser dentro de las áreas verdes de la facultad o debajo de un puente. También se buscó encontrar posibles fallas en la translación del objetivo a la posición del salón que se escogiera en el menú de búsqueda.

Se realizaron varias búsquedas para cada nivel de cada edificio y se encontraron los siguientes problemas.

- El mal posicionamiento del objetivo al momento de iniciar la búsqueda: Este error consistía en que cuando se elegía un aula y se comenzaba la búsqueda, el objetivo no se movía hacia la coordenada deseada, quedando muchas veces dentro de las paredes o dentro de los pisos o techos. (Fig. 23.1)
  Se identificó el problema, que tenía que ver con la orientación en la que Unity importa los objetos provenientes de programas de diseño que definen la altura con el eje Z, mientras que en Unity la altura está definida por el eje Y.
  Para mover el objetivo al lugar indicado se tenía que proporcionar la coordenada de un objeto que se encontrara en la posición del aula a buscar. En este caso se utilizaron los letreros de los salones Por lo tanto, como la orientación de las normales tenían definida la altura por el eje Z, cuando el objetivo se movía a esa coordenada y se desplazaba en dirección de la normal de la cara del letrero quedaba en la posición incorrecta.
  Para solucionar este problema, simplemente mediante código se aplicó una rotación de 90 grados a la normal sobre el eje X para que ahora la altura fuera definida por el eje Y.

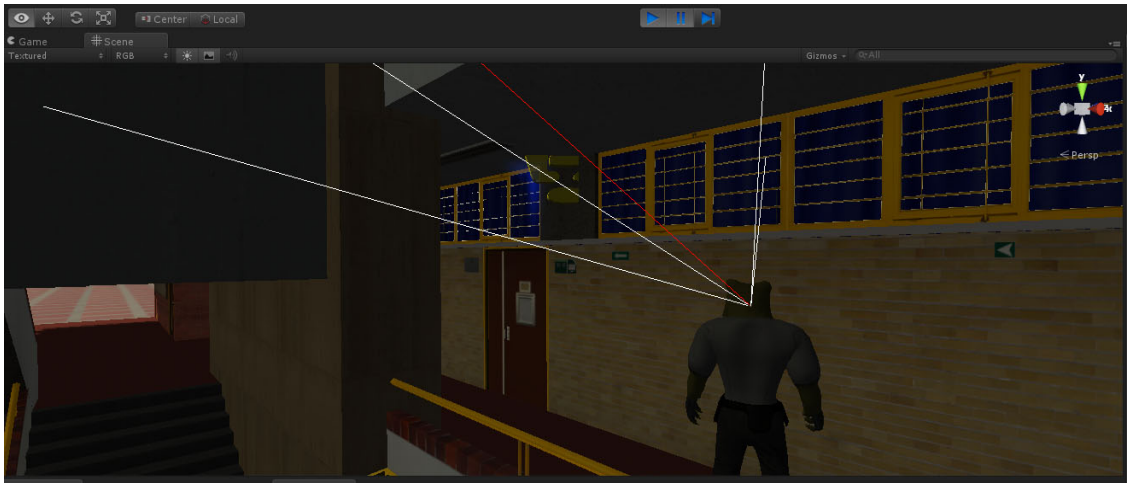


Fig. 24.1. Se puede apreciar que el objetivo quedaba dentro de las paredes del edificio.

- Problema con la inicialización del objeto guía: Este problema ocurría la segunda vez que se iniciaba una búsqueda. El objeto buscador se desplazaba hasta el lugar desde donde se había realizado la primera búsqueda. Sin importar que el personaje se encontrara en otro lado del escenario. (Fig. 23.2)

  El problema era causado por el Pathfinding de Unity, el cuál no estaba registrando el fin o la cancelación de la búsqueda por parte del usuario. Así que, cada vez que se iniciaba una nueva búsqueda, el objeto buscador se movía a la posición del personaje pero inmediatamente cambiaba a la su posición original y de ahí trazaba el camino al aula indicada.

  Como solución se decidió que, mediante código, se generaría al objeto buscador sí y sólo sí se iniciaba una búsqueda y de la misma forma debería ser destruido cuando esta finalizara o se cancelara. De esta forma no sólo se corrigió el error, también hubo una pequeña mejora en el rendimiento al no tener un objeto presente cuando no se le necesitaba.

Fig. 24.2. Se observa que a pesar de haber iniciado la búsqueda en la parte de abajo el objeto buscador inmediatamente se movió hacia arriba

Menú de Pausa y Menú principal: Aquí simplemente se comprobó que se pudiera ir y regresar al menú principal las veces que el usuario considere necesario. Las pruebas se centraron específicamente en la transición del menú principal al escenario principal. Se corroboró que el personaje siempre apareciera en uno de los dos puntos de entrada que se definieron. Por último se verificó que se oscureciera la pantalla cada vez que se cargara el escenario, esto se realizó exitosamente por lo que no hubo necesidad de hacer más pruebas.

Después de realizar las pruebas, y aplicar las correcciones necesarias, se finalizó el proceso del desarrollo de la aplicación (Figs. 23.3, 23.4 y 23.5). El producto final que se obtuvo fue una aplicación con la cual podemos realizar un recorrido virtual a través del edificio principal de la Facultad de Ingeniería en la que el usuario controla a un personaje y puede moverse libremente por la Facultad y si necesita saber la localización de un salón, simplemente teclea el salón que desea encontrar y una guía lo llevara hasta él.

.

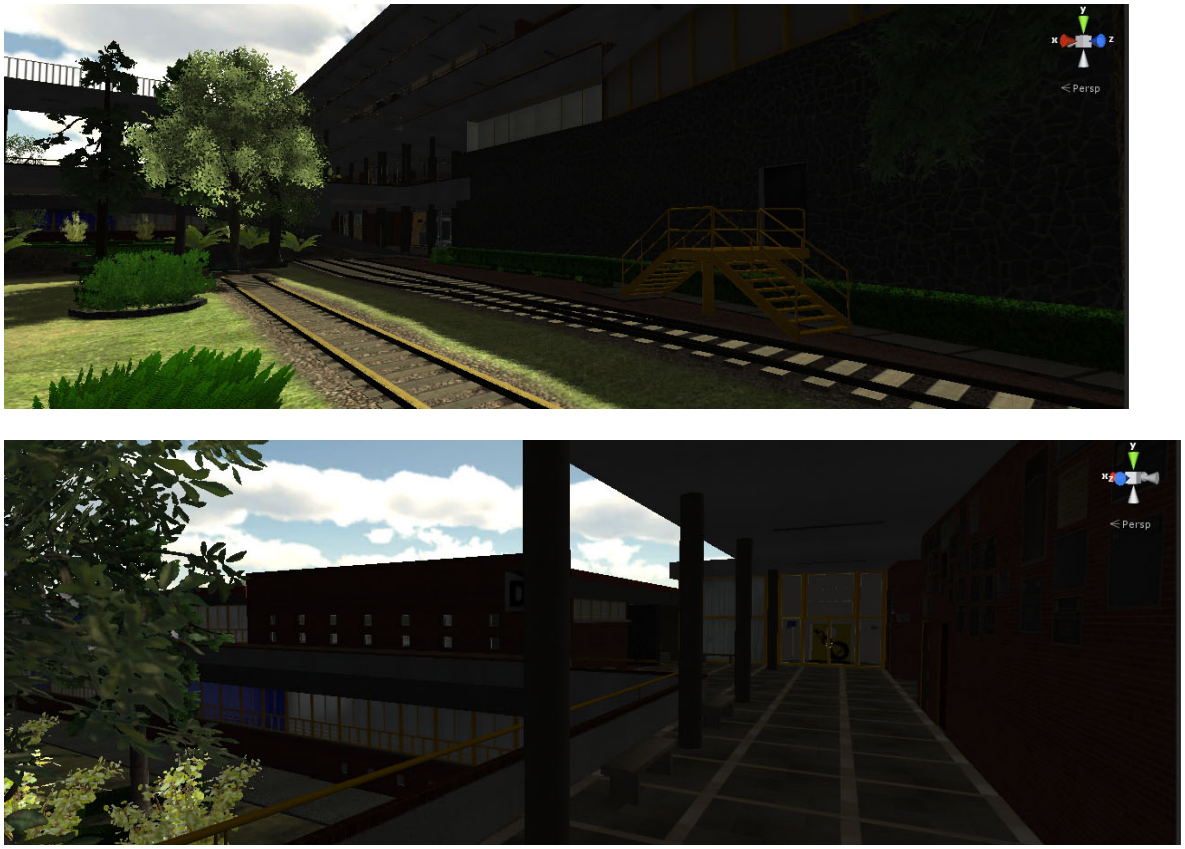


Fig. 24.3. Edificio A de la Facultad de Ingeniería cargado en el motor de juegos Unity. Parte exterior y biblioteca

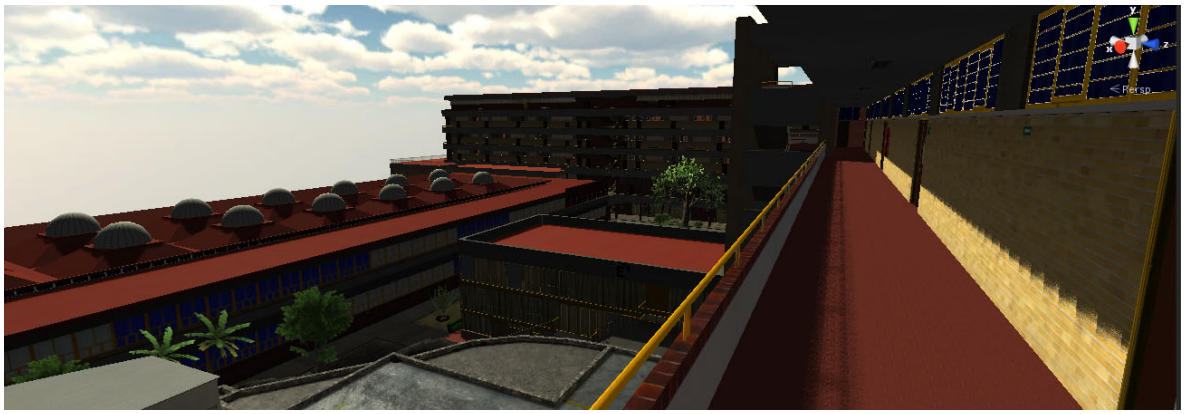


Fig. 24.4. Edificio B de la Facultad de Ingeniería

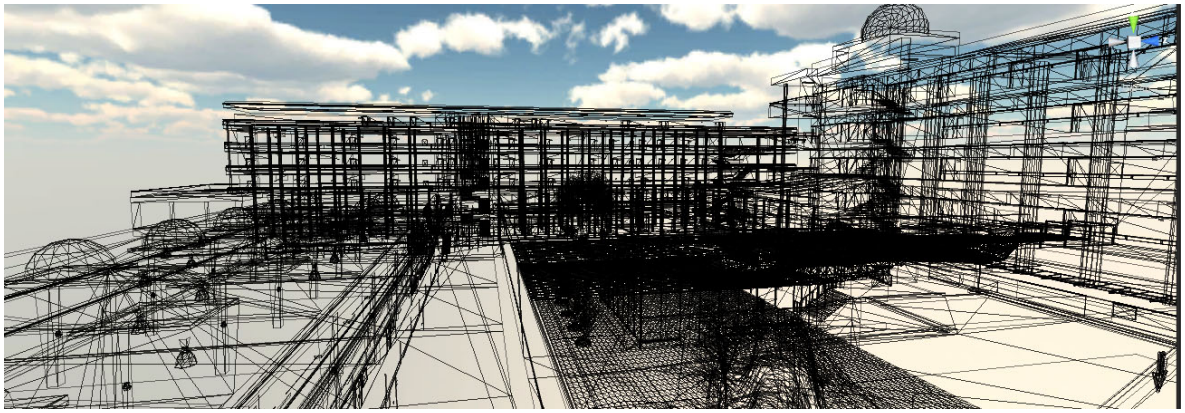


Fig. 24.5. Estacionamiento interior de la Facultad de Ingeniería en modo wireframe, aquí podemos observar los polígonos de los que se compone la facultad

5.5 Liberación del proyecto

Como último paso en este proyecto se generó el archivo ejecutable correspondiente. Se quitaron recursos innecesarios, como herramientas, texturas, modelos y scripts utilizados en el desarrollo del proyecto, ya que su inclusión en el archivo ejecutable sólo hubieran generado un archivo de mayor tamaño.

Dicho archivo ejecutable se encuentra en el CD anexo con esta Tesis y en el apéndice 8.5 se tiene el manual de usuario donde se muestra la aplicación ejecutándose.

5.6 Mantenimiento

El mantenimiento de esta aplicación se deberá de realizar cuando sea incorporado al Sistema de Inscripciones de la facultad, con la finalidad de corregir los errores que pudieran generarse al hacerlo.

En el alcance de este proyecto no se consideró la inclusión de esta aplicación en el sistema de inscripciones, quedará a decisión de la Dirección encargada de dicha área el aceptar la incorporación de esta aplicación a su sistema. Por esta razón no es posible abordar esta etapa en este momento.

Sin embargo algunos de los problemas que podrían surgir serian:

- Adaptar la aplicación a otras plataformas: Si fuera necesario adaptar a otras plataformas este proyecto, como móviles u otros sistemas operativos, se deberá ajustar el correspondiente, para que sea reconocido por la nueva plataforma.

- Crear un archivo ejecutable de menor tamaño: Se deberá cambiar el tamaño de muchas de las texturas, las cuales son las que ocupan el mayor porcentaje de espacio en un proyecto de este tipo, a un tamaño más pequeño. Considerando que al hacerlo se perderá calidad en los gráficos.
- Introducción de hardware más avanzado: Aunque sea poco probable que un hardware más avanzado no tenga el poder computacional para ejecutar esta aplicación, siempre existe la posibilidad de que no sea compatible con ciertos aspectos como lo son los shaders[6] de los materiales. En este caso se tendría que ajustar el código de los mismos para ser reconocidos por el nuevo hardware. Los tipos de shader más utilizados en este proyecto fueron:
  a) Diffuse.
  b) Normal mapped.
  c) Normal Mapped Specular.
  d) Parallax Normal Mapped Specular.
  e) Transparent
  f) Reflective
  g) Transparent/Reflective

  De acuerdo con documentación de Unity, se necesita de una tarjeta aceleradora, capaz de soportar shader model 4.0

- Expandir la aplicación a otros edificios de la facultad: Para esto sería necesario dividir en varias escenas cada nuevo complejo que se decidiera agregar para tener un buen diseño de nivel. A pesar de que Unity es muy eficiente para procesar objetos con un considerable número de polígonos, querer incluir varias secciones de la Facultad de Ingeniería en una solo escena sería muy ineficiente.
- Agregar funcionalidad: Un proyecto de este tipo es bastante flexible al momento de querer incorporar nueva funcionalidad. Por ejemplo si se quisiera usar para una aula virtual para la educación a distancia, o para la construcción de un ambiente de interacción a distancia entre usuarios como Second Life.
  Para esto se deberá considerar el modelado de nuevas secciones como interior de las aulas, y revisar el diseño de niveles para obtener un rendimiento óptimo. De la misma forma se deberán de implementar funciones de red para la interacción de varios usuarios vía Internet.

Para la solución de estos u otros problemas que puedan surgir se deberá tener conocimiento de las herramientas utilizadas por la gente encargada del futuro manejo de la aplicación y de las que se usaron en el desarrollo de este proyecto.

[6] Los shaders son programas utilizados para realizar transformaciones y crear efectos especiales, como por ejemplo iluminación, fuego o niebla; que se escriben en un lenguaje estándar, para procesar tanto píxeles (Píxel Shader) como vértices (Vertex Shader), e incluso permiten generar nuevas primitivas de forma dinámica así como la modificación de primitivas existentes (Geomtry shader).

6. Conclusiones.

Gracias a los avances que se han dado en el campo de la realidad virtual, el público ha podido experimentar la realidad virtual de diferentes formas. Aunque la realidad virtual inmersiva y semi-inmersiva no son tan comunes, por el alto costo y la cantidad de equipo especializado que se necesita para reproducirla. La realidad virtual no inmersiva le ha permitido al público tener un acercamiento a estas tecnologías, a través de los videojuegos y los recorridos virtuales.

Tomando en cuenta la popularidad de estos, fue por lo que se decidió crear un recorrido virtual, al usar tecnologías de realidad virtual no inmersiva, se garantizó que un gran número de personas pueda tener acceso a nuestro proyecto. Una de las ideas por la cual se gestó este proyecto fue la escasa información con la que se cuenta para la localización de los salones.

Si bien durante el desarrollo de este proyecto se pudo observar una mejora en la forma de ubicar los salones en la facultad, se cambió la numeración de los salones eliminando los salones con designación B, se agregaron los letreros identificando a los edificios, por lo que se hizo más fácil localizar las aulas de esta forma y algunas otras pequeñas mejoras. Nos pareció que nuestra aplicación podría complementar esas mejoras, y no solo eso, también presentar esta idea de una forma llamativa para cualquier persona, y que mejor forma de captar la atención de las personas que usar un recorrido virtual.

Durante el proceso que conllevo el diseño de la aplicación se adquirió una gran experiencia en los diferentes procesos que se realizaron, como se pudo ver con el cambio que presento el primer modelo del personaje del recorrido, al diseño final del personaje; hubo también mejoras en la programación de los diferentes scripts que se usaron en el motor de juegos, si bien estos no se aprecian a simple vista en el proyecto final, esas mejoras están ahí.

También se aprendió mucho sobre los procesos necesarios para realizar un videojuego, si bien, lo que se presenta en el trabajo es un recorrido virtual, las bases para realizar este proyecto son las mismas que se siguen para desarrollar un videojuego; como tomar en cuenta el número de polígonos en un modelo para que el motor de juegos no tenga problemas al cargar los modelos, y los framerates se mantengan constantes, y de esta forma obtener movimientos fluidos dentro del juego.

Una vez terminada la aplicación, se comparó el producto obtenido con los objetivos con los cuales se realizó el proyecto, y se observó que se cumplieron estos objetivos. Se tenía en mente que la aplicación iba a permitir al usuario navegar libremente por la Facultad de Ingeniería, objetivo que se logró, y el otro objetivo importante era que el usuario pudiera localizar las aulas de la Facultad; esto se logró con la implementación del pathfinding, de esta manera el usuario coloca el nombre del salón que quiere localizar, y se calcula una ruta desde donde este colocado el usuario, y el programa le muestra cómo llegar a su destino. Otro de los requerimientos era que fuera de fácil acceso al público, al usar Unity la aplicación se puede publicar en modo web, por lo cual si el sistema de inscripciones decide colocarlo en su página, la aplicación estaría disponible a cualquier persona que tenga acceso a internet.

Si bien la aplicación ya está terminada, aún queda mucho por hacer. Para este caso en particular se escogió el edificio principal de la Facultad de Ingeniería, pero no es una limitante para un proyecto a futuro, se podría agregar a esta aplicación el anexo de Ingeniería, la Torre de Ingeniería, y si el proyecto gana aceptación no se tendría que limitar a la Facultad de Ingeniería, se podría llevar a las demás facultades, y eventualmente, tener un modelo virtual de todo Ciudad Universitaria.

Partiendo de esta idea, el hacer el modelo completo de Ciudad Universitaria requiere de mucho trabajo, y de gente especializada en diferentes campos para lograr un recorrido de primera calidad, inclusive se podría cambiar de motor de juegos a otro que ofrezca una mayor capacidad de manejo de polígonos, o que ofrezca mejor manejo de detalles.

Por el momento al contar con el modelo en Unity, se tiene la capacidad de portar este modelo a plataformas Android, e iOS, de esta forma teniendo un recorrido virtual de la Facultad de Ingeniería en la palma de tu mano. A pesar de que se tenga esta capacidad no quiere decir que sea un trabajo sencillo, ya que se deben optimizar los modelos y texturas para que funcionen de forma óptima en los celulares, pero el trabajo es menor al de realizar todo el proyecto desde cero para tener esta opción.

7. Bibliografía y Mesografía.

[1] Avgerakis George, Digital Animation Bible, 1era edición, McGraw Hill , 2004

[2] Franson D., 2D Artwork and 3D Modeling for Game Artist. Premier Press Game Development Series, pp. 186-191 2003.

[3] Lee Kim, Inside 3D Studio Max 4, 2da edición, New Riders, 2002.

[4] Mitchell W. J. y McCullogh M, Digital Design Media, 2da Edición, Van Nostrand Reinhold, 1994.

[5] Piegl Les & Tiller Wayne: The NURBS Book, 2da edición, Springer-Verla, 1995–1997.

[6] Ramshaw Lyle Dr. Blossoming: A connect-the-dots approach to splines, Palo Alto, CA, Junio, Research Report 19, Compaq Systems Research Center, 1987.

[7] Roger David F.: An Introduction to NURBS with Historical Perspective, Morgan Kaufmann Publishers 2001.

[8] Søren Dreijer, Bump Mapping Using CG, 3era edición.

[9] Van Dam Foley, Feiner & Hughes: Computer Graphics - Principles and Practice, 2da edición, Addison Wesley 1996.

Adobe Photoshop CS, Manual de usuario, 2003.

Sederberg Thomas Dr., BYU NURBS, http://cagd.cs.byu.edu/~557/text/ch5.pdf

¿Qué es la Realidad Virtual? : http://www.realidadvirtual.com/que-es-la-realidad-virtual.htm (9 diciembre 2012)

How reality works? : http://electronics.howstuffworks.com/gadgets/other-gadgets/virtual-reality.htm (1 Enero 2013)

Virtual Reality: http://www.vrs.org.uk/#virtual-reality (10 Noviembre 2012)

Partes de una Tesis: http://profesores.fib.unam.mx/jlfl/Seminario_IEE/ Seminario_IEE_Tema_3a.pdf (12 Diciembre 2012)

Definición (Definición de Virtual): http://definicion.de/virtual/ (11 noviembre 2012)

Enter@te en línea, David Gustavo Alberto Gutiérrez Ramírez: http://www.enterate.unam.mx/Articulos/2007/abril/sarahi.html (5 Octubre 2012)

Creación de ambientes virtuales inmersivos con sofware libre, María del C. Ramos: http://www.oei.es/noticias/spip.php?article823 (2 Noviembre 2012)

Redalyc: http://redalyc.uaemex.mx/redalyc/pdf/768/76820304.pdf (3 Febrero 2013)

Hiperbolia City: http://hiperbolia.com/indexesp.php (1 Agosto 2012)

El Sonido 13, Georgina Cárdenas: http://www.elsonido13.com/detalle-noticia.asp?id=543 (12 Diciembre 2012)

Crónica: http://www.cronica.com.mx/nota.php?id_nota=531486 (11 Diciembre 2012)

Revista Digital Universitaria, Georgina Cárdenas: http://www.revista.unam.mx/vol.13/num3/art34/art34.pdf (2 Febrero 2013)

RT: http://actualidad.rt.com/actualidad/view/26946-Realidad-virtual-al-servicio-de-mexicanos-para-ayudar-a-recuperarse-de-sucesos-traum%C3%A1ticos (7 Mayo 2012)

La Jornada Zacatecas: http://www.ljz.mx/secciones/ciencia-y-tecnologia/59-ciencia-y-tecnologia/17450-utiliza-unam-ambientes-virtuales-para-tratamiento-de-obesidad.html (5 Mayo 2012)

Realidad Virtual.com: http://www.realidadvirtual.com/info/origenes-de-la-realidad-virtual.htm (3 Marzo 2012)

A Critical History of Computer Graphics and Animation: http://design.osu.edu/carlson/history/lesson17.html (9 Octubre 2012)

Wikipedia: http://en.wikipedia.org/wiki/Link_Trainer (8 Noviembre 2012)

Informática Hoy: http://www.informatica-hoy.com.ar/realidad-virtual/Realidad-Virtual-Su-historia-y-sus-variantes.php (8 Agosto 2012)

Historia de la Realidad Virtual: http://usuarios.multimania.es/artofmusic/the_matrix_vr/historia_vr.html (9 Diciembre 2012)

Realidad Virtual: http://www.inei.gob.pe/biblioineipub/bancopub/inf/lib5047/c03.HTM (8 Septiembre 2012)

Fundamentos de Realidad Virtual: http://telematica.cicese.mx/computo/super/cicese2000/realvirtual/Part2.html (9 Enero 2013)

Polycount Wiki: http://wiki.polycount.net/CubeMap (11 Marzo 2013)

8. Apéndice

8.1 Muestra del código correspondiente al módulo del sistema de cámara en tercera persona

Sección de la clase TP_Controller donde se obtiene la información del usuario:

```
void GetLocomotionInput()
   {
      var deadZone = 0.1f;

      if (Input.GetAxis("Vertical") > deadZone || Input.GetAxis("Vertical") < -deadZone)
         TP_Motor.Instance.MoveVector += new Vector3(0, 0, Input.GetAxis("Vertical"));

      if (Input.GetAxis("Horizontal") > deadZone || Input.GetAxis("Horizontal") < -deadZone)
         TP_Motor.Instance.MoveVector += new Vector3(Input.GetAxis("Horizontal"),0,0);

      //Despues de calcular nuestro vector de movimiento llamamos a TP_Animator para que con el vector
      //recien calculado determine la direccion en la que nos movemos
      TP_Animator.Instance.DetermineCurrentMoveDirection();

   }

   //El siguiente metodo controlara todas la acciones que haga nuestro personaje cuando se presione la tecla correspondiente , como
saltar
   void HandleActionInput()
   {
      //Revisamos si se ha presionado la tecla de salto que tiene por defecto Unity.
      if(Input.GetButton("Jump"))
      {
         //Si es asi llamamos al metodo Jump()
         Jump();
      }
}
```

Sección de la clase TP_Motor donde se procesa el vector obtenido de TP_Controller:

```
void ProcessMotion()
     MoveVector=new Vector3(0,MoveVector.y,0);

      // Se normaliza el vector si magnitud >1
      if (MoveVector.magnitude > 1)
         MoveVector = Vector3.Normalize(MoveVector);

      //Aplicamos deslizamiento si aplica el caso
      ApplySlide();

       MoveVector *= MoveSpeed();

      //Volvemos a aplicar la velocidad terminal, que guardamos previamente en el metodo GetLocomotionInput()
      //en la clase TP_Controller, a MoveVector.y
      MoveVector = new Vector3(MoveVector.x, VerticalVelocity, MoveVector.z);

      //Aplicamos gravedad
      ApplyGravity();

      //MoveVector *= Time.deltaTime;
// Se mueve el personaje en el espacio
```

```
    TP_Controller.CharacterController.Move(MoveVector*Time.deltaTime);
  }

  // Metodo para aplicar la gravedad
  void ApplyGravity()
  {
    // Primero asegurarnos de que no nos salgamos de  la velocidad terminal
    if (MoveVector.y > -TerminalVelocity)
      // Para aplicar la gravedad afectamos solo la componente Y de nuestro vector , tomamos su alor acuat y le restamos
      //el valor de la gravedad multiplicado por el delta del tiempo para hacer consistente la gravedad en segundos y no en frames
      MoveVector = new Vector3(MoveVector.x,MoveVector.y - Gravity*Time.deltaTime,MoveVector.z);

    //Revisamos si nuestro personaje esta sobre la tierra y si su velocidad en y es menor a -1 (esto quiere decir si está bajando una
distancia
    //despreciable como un pequeño escalón)
    if (TP_Controller.CharacterController.isGrounded && MoveVector.y < -1)
      // si es asi su vel en Y sera siempre -1
      MoveVector = new Vector3(MoveVector.x, -1, MoveVector.z);
  }
```

Sección de la clase TP_Camera donde se obtiene la información del mouse:

```
void HandlePlayerInput()
    //Registra los datos de entrada que manda el usuario
{
    //Este valor define el rango en el que se podra mover
    //el scroll sin que pase algo
    var deadZone = 0.01f;

    if (Input.GetMouseButton(1))
    {
      // Si el boton derecho del mouse está Presionado
      //Get Mouse Axis
      mouseX += Input.GetAxis("Mouse X") * X_MouseSensitivity;
      mouseY -= Input.GetAxis("Mouse Y") * Y_MouseSensitivity;
    }

    //Aqui limitaremos el movimiento en Y del mouse
    mouseY = Helper.ClampAngle(mouseY, Y_MinLimit, Y_MaxLimit);


    //En esta parte se obtiene la distancia que se alejara o acercara la camara cuando
    // giremos el scroll

    //Checamos que este fuera de la zona neutral
    if ((Input.GetAxis("Mouse ScrollWheel") < -deadZone || Input.GetAxis("Mouse ScrollWheel") > deadZone) &&
                                    pause.pauseEnabled==false &&
                                    pause.buscador==false)
    {
      //La distancia que nos moveremos sera la distancia actual menos lo dado por el scroll por la sensibilidad
      // y recortaremos ese valor si sobrepasa nuestra distancia minima o maxima
      desiredDistance = Mathf.Clamp(Distance - Input.GetAxis("Mouse ScrollWheel") * MouseWheelSensitivity,
      DistanceMin, DistanceMax);

       //Adicionalmente cada vez que giremos la rueda del mouse igualaremos preOccludedDistance con desiredDistance, con esto
       indicamos que a pesar de que hubiera una distancia previa
       //producto de  chocar la camara con el escenario al mover el scroll le estamos diciento a la máquina que está bien , que no
       queremos regresar a la distancia anterior,
       //cuando esta deje de estar bloqueada.Mismo caso con el distanceSmooth , indicamos que use la velocidad normal , no la
       definida para cuand deba regresar a la posicion anterior

       preOccludedDistance = desiredDistance;
       distanceSmooth = DistanceSmooth;
```

```
    } }
```

Sección de la clase TP_Animator donde se determina el estado del personaje:

```
public void DetermineCurrentMoveDirection()
  {
    //iniciamos esta serie de variables como falsas porque inicialmente no hay una direccion
    //hacia donde nos estemos moviendo
    var forward = false;
    var backward = false;
    var left = false;
    var right = false;

    //Revisamos hacia donde nos movemos , si z>0 hacia adelante, z<0 hacia atras
    // si x>0 hacia la derecha y si x<0 hacia la izquierda
    if (TP_Motor.Instance.MoveVector.z > 0)
       forward = true;
    if (TP_Motor.Instance.MoveVector.z < 0)
       backward = true;
   if (TP_Motor.Instance.MoveVector.x > 0)
       right = true;
    if (TP_Motor.Instance.MoveVector.x < 0)
       left = true;

    //Despues de saber cuál de las 4 direcciones principales se encuentra el personaje
    //Revisamos las diagonales
    //Si se está presionando hacia adelante revisamos si además se está presionando la izquierda o la derecha

    if (forward)
    {
       if (left)
          MoveDirection = Direction.LeftForward;
       else if (right)
          MoveDirection = Direction.RightForward;
       else
          MoveDirection = Direction.Forward;
    }

    //Hacemos lo mismo pero en direccion hacia atras
    else if (backward)
    {
       if (left)
          MoveDirection = Direction.LeftBackward;
       else if (right)
          MoveDirection = Direction.RightBackward;
       else
          MoveDirection = Direction.Backward;
    }

    else if (left)
       MoveDirection=Direction.Left;

    else if (right)
        MoveDirection=Direction.Right;

    else
                        MoveDirection = Direction.Stationary;


  }
            //Aqui se determina en qué estado se encuentra el personaje (corriendo,caminando,saltando,etc)
            void DetermineCurrentState()
            {
                        if(State==CharacterState.Dead)
                                    return;
```

```
if(!TP_Controller.CharacterController.isGrounded)
{
	if(State!=CharacterState.Falling&&
		State!=CharacterState.Jumping&&
		State!=CharacterState.Landing)
	{
		//Deberiamos estar en caida si se da esta situacion
		Fall();
	}
}
//Si llegamos a esta condicion significa que estamos en el suelo, entonces revisamos:
if(State!=CharacterState.Falling&&
	State!=CharacterState.Jumping&&
	State!=CharacterState.Landing)
{
//Si se llega a este punto significa que solo sería posible que el personaje se estuviera desplazando en el suelo
//asi que con el siguiente switch se designara el estado actual


	switch(MoveDirection)
	{
		case Direction.Stationary:
			if(timer>0)
			{
			timer-=Time.deltaTime;

				State=CharacterState.Idle;
			}

			else if(timer<=0&&timer2>0)
			{
				timer2-=Time.deltaTime;

				State=CharacterState.Idle2;
			}
			else{timer=59;timer2=13.3f;}

			break;
		case Direction.Forward:
			State=CharacterState.Running;
			break;
		case Direction.Backward:
			State=CharacterState.WalkingBackwards;
			break;
		case Direction.Left:
			State=CharacterState.StrafingLeft;
			break;
		case Direction.Right:
			State=CharacterState.StrafingRight;
			break;
		case Direction.LeftForward:
			State=CharacterState.Running;
			break;
		case Direction.RightForward:

			State=CharacterState.Running;
			break;
		case Direction.LeftBackward:

			State=CharacterState.WalkingBackwards;
			break;
		case Direction.RightBackward:
			State=CharacterState.WalkingBackwards;
			break;
		}			}}
```

8.2 Muestra del código correspondiente al módulo del menú principal

Sección de la clase MenuScript donde se detectan la opciones del usuario:

```
void Raycasting()
{
                        if  MouseButton(0)
                                Ray ray =Input.mousePosition
                                RaycastHit hit
                                if (ray)
                                        if (hit=B_inicio)
                                                folder.animation.Play("Open")
                                                goyo.animation.Play("TurnPage")

                                        else if (hit = volver)
                                                goyo.animation.Play("TurnBackPage");
                                                folder.animation.Play("Close");
                                        else if (hit = punto_1)
                                                fade.SetScreenOverlayColor(new Color(0,0,0,0));
                                                fade.StartFade(new Color(0,0,0,1), 3,1);
                                        else if (hit= punto_2)
                                                fade.SetScreenOverlayColor(new Color(0,0,0,0));
                                                fade.StartFade(new Color(0,0,0,1), 3,2);
                                else
                                        No hacer nada
                        else
                                No hacer nada
                        if (!animacion)
                        goyo.animation.CrossFade("Idle");
```

8.3 Muestra del código correspondiente al módulo del menú de pausa y buscador

Sección de la clase Buscador donde se realiza la búsqueda del nombre del aula introducida por el usuario:

```
void Buscador(){

bool search;            //Indica si se ha presionado el boton buscar
bool anadeElemento=false;       //Evita que se incluya 2 o mas veces el mismo resultado en la lista

/* Construimos un Cuadro que contendra a todo el menu , luego se construye un text field para que el usuario ingrese
 * la palabra a buscar  y eso se iguala a "salon" . Se construye el boton buscar y se iguala a search*/

GUI.Box(new Rect(Screen.width /2 - 350,Screen.height /2 - 200,700,350), "Busca tu Salon");
salon = GUI.TextField(new Rect(Screen.width /2 - 330, Screen.height /2 - 180, 400, 20), salon, 40);
search=GUI.Button(new Rect (Screen.width /2 +80, Screen.height /2 - 180, 100, 30), "Buscar");

if (search){
            nombres.Clear();
            salon=salon.ToUpper();
            string [] salonDiv=salon.Split(' '); //Arreglo temporal donde se divide la frase contenida en salon en palabras
                        for(int i=0;i<objetos.Length;i++){
```

```
                    for(int j=0;j<salonDiv.Length;j++)
                        {
                        if(objetos[i].name.Contains(salonDiv[j])&&anadeElemento==false)
                            {
                            nombres.Add(objetos[i].name);
                            anadeElemento=true;
                            }
                        }
                        anadeElemento=false;
                    }
                    nombres.Insert(0,"Elige un Boton");

                }
                //Si la lista no esta vacia es decir que  se encontraron coincidencias
                if(nombres.Count>1){
                    //Paso el contenido de la lista a un arreglo para que lo acepte la funcion GUI.SelectionGrid()
                    string [] temp=nombres.ToArray();
                    GUILayout.BeginArea (new Rect (Screen.width /2 - 325,Screen.height /2-100, 670, 400));


                    seleccion = GUILayout.SelectionGrid(seleccion, temp, 2);
                    GUILayout.EndArea ();
                }
                else{
                    //Si la lista esta vacia manda un mensaje de que no hubo resultados para esa busqueda
                     GUI.Label (new Rect (Screen.width /2 - 250,Screen.height /2-50 ,200, 100), "No encontre resultados");

                }



/*Se usa la funcion GUI.changed para que los siguientes calculos solo se efectúen si se hace alguna interaccion con la interfaz, esto para evitar el gasto innecesario de recursos que se genererian al hacerlos cada cuadro*/
                if (GUI.changed)
                {
                    if(seleccion!=0){       //se revisa si se ha seleccionado uno de los botones
                        for(int i=0;i<objetos.Length;i++)
                        {
                            if(objetos[i].name==nombres[seleccion]){

                                StartPathFinder(i);
                                break;
                            }
                        }
                    }
                }
            }
```

8.5 Manual de usuario

8.5.1 Requerimientos Mínimos

- Procesador : 2.4 GHz. O doble núcleo a 1.66 GHz o superior
- Memoria: 1GB RAM para Windows XP , 2GB Para Windows Vista, Windows 7
- Tarjeta de Video: Tarjeta compatible con DirectX 9.0 256MB en memoria de video
- Sistema Operativo: Windows XP Service Pack 2, Windows Vista, Windows 7

8.5.2 Iniciando la aplicación

Para iniciar la aplicación se deberá ingresar a ella desde Windows, haciendo doble clic sobre el icono de la misma. Al hacer esto, se empezara a cargar la aplicación e ingresaremos al Menú Principal.

8.5.3 Controles

La aplicación cuenta con los siguientes controles:

Acción	Control
Avanzar	W
Retroceder	S
Izquierda	A
Derecha	D
Mirar	Mouse
Presionar Botones/Seleccionar	Botón izquierdo del mouse
Acercar / Alejar la cámara	Rueda del mouse
Menú de Búsqueda	F
Pausa	Escape
Saltar	Barra Espaciadora

8.5.4 Menú Principal

Al iniciar la aplicación apareceremos en el menú principal en el cual podremos hacer las siguientes acciones:

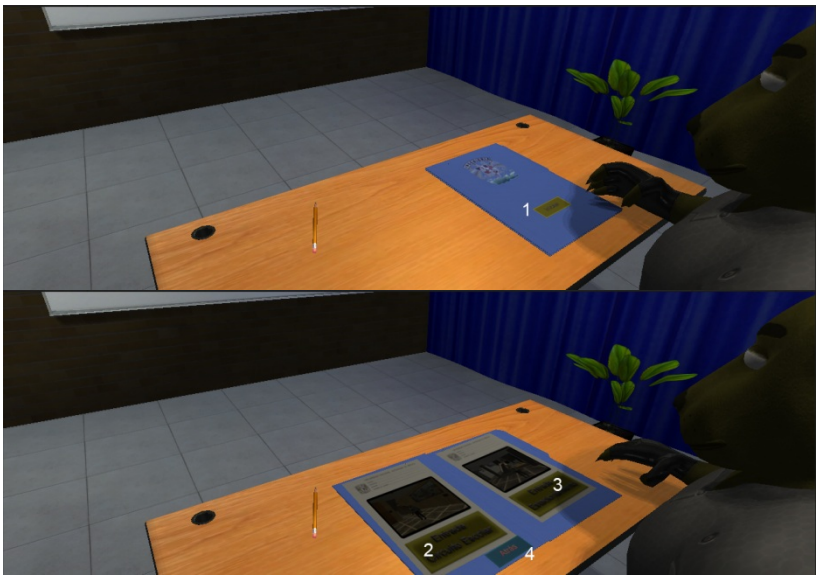


Fig. 25: Menú Principal

1. Iniciar: Abrir el folder de selección de entrada
2. Elegir la entrada ubicada en Circuito Escolar
3. Elegir la entrada ubicada en el estacionamiento de la Facultad de Ingeniería
4. Cerrar el folder de selección

8.5.5 Recorrido Virtual

Una vez seleccionado el punto de entrada, se puede recorrer todo el escenario a voluntad. Se usan las teclas W, A, S, D para mover al personaje y el mouse para dar dirección. También se puede usar la Barra Espaciadora para saltar.

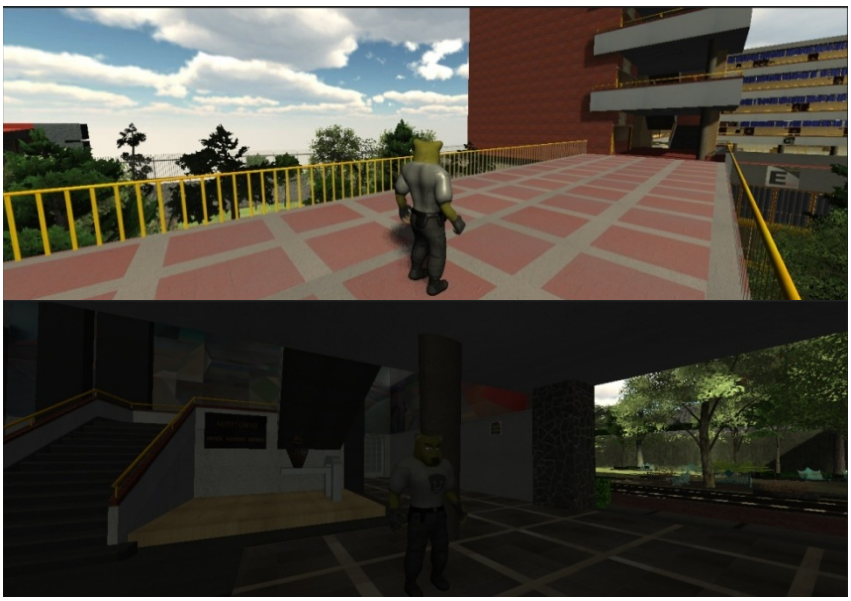


Fig. 26 Escenario

8.5.6 Buscar Aula

En cualquier momento podremos iniciar una búsqueda para encontrar un aula. Para esto presionamos F para desplegar el menú de búsqueda

Fig. 27: Menú de Búsqueda

1. Cuadro de texto donde se escribe el aula a buscar
2. Botón para inicial la búsqueda.

Una vez que se escribe el nombre del aula se mostrará una lista con todas las coincidencias y se podrá elegir uno de los botones correspondientes al aula que se quiere llegar, si no hay coincidencias se deberá de hacer otra búsqueda.

Fig. 28: Búsqueda de aulas

3. Lista de botones con los resultados de la búsqueda

Una vez que se elige uno de los botones se tiene que seguir al objeto guía, el cual llegará hasta el destino. Si el usuario se aleja del objeto guía, este se detendrá y esperará a que el usuario vuelva a acercarse para continuar el recorrido. Al llegar al destino el objeto desaparecerá.

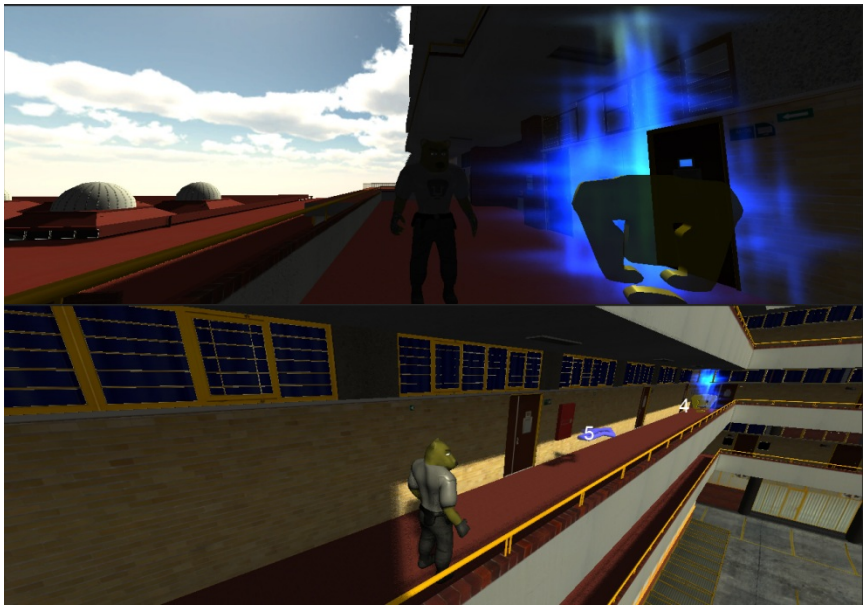


Fig. 29: Objeto guía y objetivo

4. El destino se aparece resaltado por un objeto iluminado
5. Objeto guía

8.5.7 Menú de Pausa

En cualquier momento se puede accesar al Menú de Pausa presionando la tecla ESC y se tienen las siguientes opciones:

Fig. 30 Menú de Pausa

1. Botón Menú Principal: Regresa al usuario al menú principal
2. Botón de Gráficos: Despliega una lista donde podremos elegir la calidad gráfica de la aplicación
3. Lista de Calidad de Gráficos: Lista de los diferentes niveles de calidad gráfica. El nivel más bajo es “Fastest” y el nivel más alto es “Fantastic”. Si se tiene un equipo con bajas especificaciones, se deberá elegir uno de los niveles más bajos.
4. Botón Salir: Sale de la aplicación y nos regresa a Windows.