UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

Sistema de Modelado Tridimensional basado en Trazos Libres: MOSKIS 3D

Autor: Inmaculada Gijón Cardos
Director: Carlos González Morcillo

Julio, 2006.

TRIBUNAL:

Presidente:
Vocal:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	**Fdo.:**	**Fdo.:**

Este documento fue compuesto con LaTeX. Imágenes generadas con OpenOffice.

Resumen

El proceso de síntesis 3D ha cobrado gran importancia en los últimos años. Su difusión ha sido muy amplia en distintos campos, como simulación de prototipos para su posterior construcción, proyectos artísticos y/o comerciales como producciones publicitarias y en áreas de entretenimiento como cine y videojuegos.

Una fase determinante en este proceso es la fase de modelado, la cual es muy costosa, compleja y requiere un gran número de recursos.

Con MOSKIS 3D se pretende reducir la complejidad de la fase de modelado, consiguiendo rápidamente un prototipado inicial, de forma que se obtiene un modelo 3D básico a partir de un trazo 2D que indica el contorno de la figura que se desea obtener.

Este proyecto está dedicado a
mi familia, por hacerlo posible
gracias a su constante apoyo
y confianza.

Agradecimientos

A Carlos González Morcillo, por su gran idea y su contínuo apoyo.
A Daniel y a mi hermana, María, por la revisión de este documento.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El proceso de síntesis 3D es aquel por el cual se obtiene una imagen 2D (raster) que representa la descripción de una escena tridimensional.

El proceso de síntesis 3D se puede dividir en tres fases: [GM05a] [Ram05b]

- **Pre-producción**. En esta etapa se establecerán los pasos previos para poder desarrollar la escena.

  - **Escritura del guión.**

    Se procederá a la escritura del guión sobre el cual se trabajará. Esta fase se desarrollará normalmente al principio en proyectos pequeños y, en proyectos grandes, se trabajará en ello también durante el proyecto, teniendo en cuenta las limitaciones técnicas que puedan surgir.

  - **Desarrollo visual.**

    En esta fase se establecerá el estilo de la escena, la atmósfera, que vendrá determinada por la luz, y se seleccionará también la paleta de colores básica sobre la que se trabajará.

  - **Diseño de personajes.**

El diseño de personajes normalmente se realizará mediante bocetos, aunque en grandes producciones se utilizan también, en la etapa de preproducción, esculturas o modelos 3D básicos.

El diseño se realizará ajustándose al papel que desempeñan los personajes en el guión, dándo mayor detalle en los papeles principales y menor detalle en los secundarios.

De este diseño se obtienen hojas de personajes con toda la información.

- **Storyboard.**

  El storyboard suele ser de gran ayuda, porque muestra mediante una serie de viñetas la secuencia del guión y proporciona una disposición visual de los acontecimientos tal y como deben ser vistos por el objetivo de la cámara.

  El storyboard puede ser tan extenso que complementa y en algunas ocasiones sustituye al anteriormente citado guión.

- **Producción**.

  - **Modelado.**

    En esta fase se llevará a cabo la construcción tridimensional de objetos y personajes.

    En la fase de modelado se obtiene la geometría del objeto o personaje a modelar, es decir, la malla 3D. Esta fase es muy costosa y compleja, sobre todo en tiempo.

    Dependiendo de los recursos y del proyecto, se pueden emplear diferentes técnicas, como escáneres 3D, que obtienen mallas tridimensionales muy densas (véase figura 1.1 y 1.2), para proyectos de alto presupuesto, cuyos resultados pueden ser completados posteriormente mediante técnicas tradicionales de modelado.

    Como se puede observar en la figura 1.2, la malla 3D obtenida es muy densa. En este caso el modelo consta de 291307 vértices y 580237 caras.

    MOSKIS 3D se desarrolla con el objetivo de facilitar en gran medida esta fase, obteniendo rápidamente modelos iniciales, que posteriormente deberán ser detallados.

Figura 1.1: Modelo en escayola.

- **Texturizado.**

  En la fase de texturizado, se establecen materiales y texturas, para los objetos y personajes, que van a definir la ”piel”de la malla 3D. Se pueden utilizar imágenes como textura, o también procedurales, que a diferencia de las imágenes, en vez de almacenar mapas de bits, lo que almacenan es el algoritmo o procedimiento que las genera.

- **Iluminación.**

  En esta fase se procede a decidir cuántos puntos de luz van a ser necesarios, sus características y su situación para cada una de las escenas. Es el resultado directo de la fase de **desarrollo visual**.

- **Esqueletos y animación.**

  La animación procedural es la que se define de forma algorítmica, como efectos especiales de explosiones, humo, pelo, o bien, simulaciones de leyes físicas como sistemas de partículas.

  Para animación de personajes se suelen utilizar esqueletos para controlar el movimiento de la malla, y cadenas de cinemática inversa (IK Chains). Para las articulaciones se utilizan restricciones para disminuir los grados de libertad y conseguir

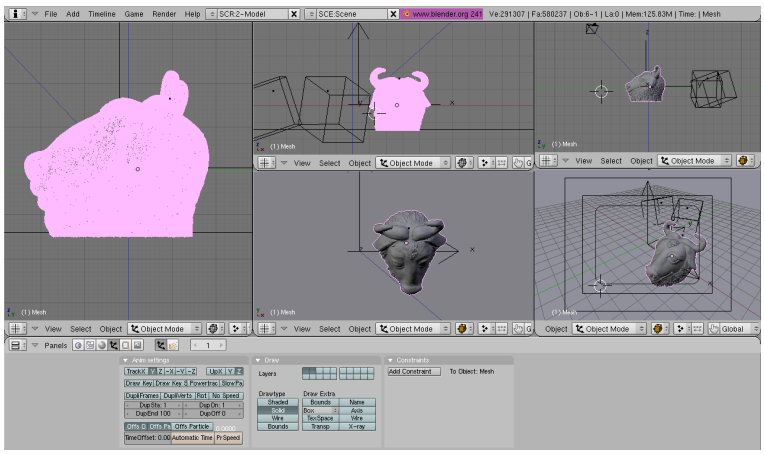

Figura 1.2: Modelo en Blender obtenido a partir de un escáner 3D.

movimientos coordinados.

Cabe destacar también como técnica de animación, la captura de movimiento, muy utilizada tanto en cine como en juegos. De forma general, se registra la posición de unas marcas y se genera la posición de los elementos del esqueleto asociado.

- **Render.**

  Es el proceso por el cual se calcula la imagen 2D asociada a una escena 3D.

  Para pruebas sucesivas, se suelen realizar renderizados de baja calidad y resolución para reducir el tiempo de cálculo. Se suelen emplear capas, para dividir la generación y obtener partes de la escena por separado, para disminuir el coste computacional.

  Se aumenta la calidad en renderizados finales ya que el tiempo empleado será más elevado.

- **Post-producción**.

  - **Retoque y efectos.**

    En esta fase se corrigen los errores de la etapa de render y se incorporan efectos menos costosos de incorporar que en fases anteriores.

- **Composición.**

  En esta fase se unen las diferentes capas generadas en la fase de render, mediante el uso de transparencias, etc.

- **Montaje.**

  Esta fase se realiza mediante editores externos y se aplican efectos de postproducción para el formato de distribución de salida.

1.1. Enfoque.

La fase de modelado es una fase muy costosa, sobre todo en tiempo.

En grandes producciones, una parte importante de los recursos utilizados, son destinados a la realización de esta fase, como queda demostrado en los estudios de Isaac Victor Kerlow. [Ker00]

Un ejemplo se puede ver en la película **Bichos** (Disney-Pixar) (véase figura 1.3).

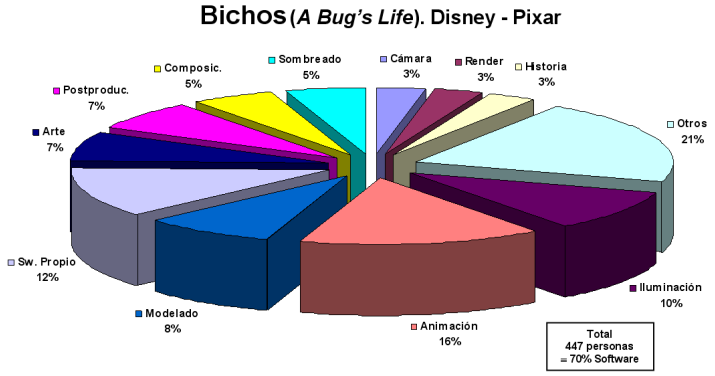

Figura 1.3: Recursos para cada fase.

Se puede comprobar la importancia de esta fase en la película, ya que este proceso ocupa un alto porcentaje con respecto a todo el proyecto. Un total de 447 personas trabajaron durante dos años para llevar a cabo el rodaje de la película. El 8 % del personal trabajó en la fase de modelado, lo que supone alrededor de 36 personas, que trabajando a jornada completa durante dos años que duró el rodaje, supone una duración aproximada de 3800 horas dedicadas a la tarea de modelado. [Ker00]

Estos datos justifican la investigación y construcción de herramientas y métodos para facilitar esta tarea.

MOSKIS 3D tiene como objetivo facilitar y reducir el tiempo de generación de modelos básicos para el prototipado inicial de un objeto o personaje 3D.

1.2. Estructura del trabajo.

Este documento se divide en siete capítulos y cinco anexos:

- **Capítulo 1:** El capítulo 1 muestra la introducción general al proceso de síntesis 3D, el enfoque general del proyecto y la estructura de este documento.
- **Capítulo 2:** Objetivos del proyecto. Enumeración y descripción de cada uno de los objetivos.
- **Capítulo 3:** Antecedentes, estado de la cuestión. En este apartado se hace un estudio del estado del arte de la fase de modelado. También se han realizado estudios sobre diferentes tecnologías y técnicas, y la situación actual de todas ellas.
- **Capítulo 4:** Métodos y fases de trabajo. En este capítulo se hace un estudio de los pasos seguidos para la realización de este proyecto.
- **Capítulo 5:** Resultados obtenidos tras la realización del proyecto y un análisis de los datos.
- **Capítulo 6:** Conclusiones y propuestas futuras. En este capítulo se hace un análisis de las conclusiones del proyecto y de las propuestas sobre futuras ampliaciones o modificaciones que se pretenden realizar.
- **Bibliografía.**
- **Anexos.** En este apartado se añaden cinco anexos:
  - Instalación de la aplicación.
  - Manual de usuario.

- Diagramas de clases.
- Diagramas de casos de uso.
- Diagramas de componentes.
- CD adjunto.

Capítulo 2

Objetivos del proyecto.

El objetivo principal del proyecto es construir una herramienta que permita la generación automática de figuras 3D a partir de bocetos 2D creados por el usuario.

Este objetivo principal, puede dividirse en una serie de subobjetivos que se detallan a continuación:

- **Generación instantánea y automática.**

  El usuario es capaz de construir objetos, que empleando otras técnicas de modelado, como superficies de subdivisión, tardaría como mínimo 10 veces más.

- **Compatible con operadores de modelado.**

  Los objetos generados con MOSKIS pueden completarse, modificarse y refinarse empleando los operadores habituales de modelado (extrusión, ...).

- **Independencia en la resolución final**.

  El uso de superficies implícitas basadas en campos de energía continuos, permite la elección de la resolución en la geometría final mediante un proceso de discretización de la información (por ejemplo, utilizando el método de Marching Cubes).

- **Rápido aprendizaje**.

  El periodo de aprendizaje en herramientas de modelado suele ser elevado.

MOSKIS 3D por el contrario, es un programa cuyo periodo de aprendizaje es muy reducido, el usuario simplemente tendrá que dibujar una silueta, ayudándose de un ratón o tableta digitalizadora y el sistema le proporcionará la malla 3D deseada.

- **Usable.**

  MOSKIS 3D proporciona una interfaz muy intuitiva que ofrece dos posibilidades:

  - Modo automático.

    El usuario realizará un trazo indicando la silueta del modelo que desea obtener, y el sistema decidirá en función de esta silueta, los parámetros adecuados y se obtendrá la malla poligonal 3D automáticamente.

  - Modo detallado.

    El usuario realizará un contorno y podrá decidir paso a paso los parámetros que le proporcionarán un nivel más o menos detallado de la figura que desea conseguir.

    En el modo detallado, el sistema sigue siendo muy intuitivo y sencillo de utilizar, ya que el número de parámetros es muy reducido y evidente en cada paso.

- **Modelos orgánicos realistas**.

  El usuario podrá realizar un trazo, definiendo el contorno de la figura que desea modelar, de esta forma se obtendrá la malla 3D asociada.

  La malla obtenida, tendrá un aspecto irregular propio de los objetos orgánicos, lo que aumentará su realismo.

  El sistema proporcionará además de la obtención de mallas 3D, la posibilidad de generar contornos y planos a partir del trazo que define la silueta del objeto.

- **Integrado en una herramienta de producción 3D profesional**.

  MOSKIS 3D estará integrado como script para la herramienta Blender, facilitando la obtención de un modelo inicial que posteriormente será detallado mediante los operadores de ésta herramienta.

Blender es una herramienta de diseño 3D que permite modelado 3D, animación, renderizado, postproducción, creación interactiva y playback y que además está disponible para múltiples plataformas.

En la actualidad, ninguna de las herramientas integradas de producción 3D como, por ejemplo, 3D Studio MAX, Lightwave o Maya, incorporan ningún script o utilidad de modelado libre basado en bocetos.

- **Multiplataforma**.

  El sistema podrá ser utilizado tanto en sistemas operativos Windows como Linux.

- **Libre**.

  No existe ninguna herramienta de las características de MOSKIS 3D distribuida con licencia libre o de código abierto. MOSKIS se distribuirá bajo licencia GPL de GNU, facilitando de esta forma el estudio e investigación en este campo de la informática gráfica.

Capítulo 3

Antecedentes, estado de la cuestión

3.1. Modelado

Dentro de la etapa de producción, en el proceso de síntesis 3D anteriormente descrito, una de las fases más importantes es la fase de modelado, en la que se basa este proyecto.

3.1.1. Definición.

El modelado se encarga de la creación de modelos consistentes mediante la definición de su geometría para que puedan ser manejados algorítmicamente mediante un computador. [GM05b]

3.1.2. Evolución histórica.

A comienzos de los años 60 aparecieron los primeros sistemas gráficos, que fueron desarrollados con propósitos CAM (Modelado Asistido por Computador). Esto dio lugar a la revolución industrial de la maquinaria numéricamente controlada (NC).

En 1967 se desarrolló en el MIT (Massachusetts Institute of Technology) el primer compilador para un lenguaje de descripción gráfica. Simultáneamente, las principales multinacionales automovilísticas y de aviación desarrollaron sus primeros sistemas gráficos de diseño.

A mediados de los 70 se produjeron avances significativos en el Modelado Geométrico. Las limitaciones iniciales que presentaban los primeros sistemas gráficos fueron superadas y las técnicas fueron evolucionando dando lugar a lo que hoy se conoce como superficies esculpidas, superficies paramétricas, de Bezier, etc.

También se desarrollaron los primeros modeladores alámbricos (wireframe) y los esquemas poligonales. Estos sistemas comenzaron siendo bidimensionales e intentaban facilitar el dibujo técnico por computador.

A continuación surgieron los esquemas de Modelado Sólido en 1973, desarrollandose los primeros sistemas gráficos en las universidades de Hokkaido y Cambridge. Los sistemas fueron presentados en Budapest como TIPS y BUILD, demostrando su superioridad ante cualquier otro tipo de modelado geométrico.

TIPS utiliza primitivas básicas y operaciones booleanas para definir sólidos, es decir, utiliza las técnicas de modelado basadas en lo que hoy día se denomina Geometría Sólido Constructiva (CSG).

BUILD define los objetos como un conjunto de superficies más la información topológica que las relaciona, que será la información de caras, aristas y vértices. Esta técnica se conoce hoy en día por Representación de Fronteras (Boundary Representation o B-Rep).

Desde entonces han surgido muchos modeladores que utilizan alguna de las técnicas anteriores, o ambas en el caso de los Modeladores Híbridos. En 1977 se desarrolló en la universidad de Rochester (EEUU) el sistema gráfico PADL-1, que utiliza CSG para la definición de objetos, pero que puede convertirlos automáticamente en B-Rep, aunque el paso contrario presenta varios problemas. [Ram05c]

3.1.3. Taxonomía de métodos.

Un modelo común, normalmente es tan complejo que pocas veces es posible definirlo mediante una única ecuación.

Para poder modelar objetos complejos, se hace uso de la combinación de objetos simples, también llamados primitivas. Dependiendo de qué primitivas se utilizen y de la combinación que se establezca se presentan distintos esquemas de modelado. [Ram05a]

- **CSG vs B-Rep.**
  - **Modelado booleano:** también llamado CSG (Constructive Solid Geometry), combina primitivas tridimensionales. Los modelos se representan mediante un conjunto de expresiones lógicas que relacionan dichas primitivas.

    El sólido queda descrito mediante un árbol binario:
    - La raíz es el modelo resultante.
    - Las hojas son las primitivas.
    - Los nodos internos son los operadores booleanos.
  - **Modelado de fronteras:** también llamado B-Rep (Boundary Representation), utiliza primitivas bidimensionales. Los sólidos quedan definidos mediante estructuras de datos. Dependiendo de las estructuras de datos de este tipo de modelos podemos considerar: [GM05b]
    - **Poligonal:** en el modelado poligonal las estructuras de datos almacenan información sobre vértices, aristas, caras y sus relaciones topológicas, es decir, cómo se conectan.

      Esta es la representación más empleada, y la única en muchos motores de render.

      Se consigue disminuir la linealidad con determinados métodos de sombreado.

      Este método es muy consistente y rápido ya que el hardware está preparado para trabajar con polígonos.

      La conversión de curva a polígono es muy fácil y rápida.

Hay que tener en cuenta que su almacenamiento es muy costoso y que hay algoritmos de sombreado y de texturizado que suelen ser más eficientes con curvas.

- **Superficies curvas:** En el modelado mediante superficies (parches) curvas, las estructuras de datos almacenan información sobre puntos de control, puntos de tangente, etc.

  Un tipo de curva utilizado son las curvas Spline, formadas por secciones polinómicas que satisfacen ciertas condiciones de continuidad en la frontera de cada intervalo.

  En la ecuación 3.1 se pueden ver las ecuaciones parámetricas de las curvas Spline.

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y \tag{3.1}$$

  En 1973, Riesen descubrió otro tipo de curva llamado B-Spline, de naturaleza no global, es decir, cada vértice tiene influencia sobre la forma de la curva en un intervalo limitado. Las curvas de Bezier son un tipo de B-Spline.

$$P(t) = \sum_{i=0}^{n} P_i N_{i,j}(t) \tag{3.2}$$

  En la ecuación 3.2, muestra la ecuación de la curva B-Spline donde P_i son los $n+1$ vértices del polígono, y $N_{i,j}(t)$ son las funciones mezcla.

  En el modelado mediante superficies curvas, la generación de estas superficies se realiza mediante redes de parches de curvas. De esta forma se consiguen suaves cambios en la malla moviendo únicamente puntos de control y el coste de almacenamiento de estos puntos es muy bajo.

- **Comparativa.**

  A continuación, se muestra una comparativa del modelado booleano con respecto del modelado de fronteras y viceversa. [Ram05a]

Esquema	Ventajas	Inconvenientes
CSG	La estructura de datos es simple y la cantidad de datos reducida, por lo tanto la gestión interna de los datos es fácil.	Las operaciones disponibles sobre sólidos están limitadas y generalmente no es fácil implementar otras operaciones.
	Si las primitivas son válidas, queda garantizada la integridad de los modelos.	La visualización por el método estándar es costosa.
	Las modificaciones globales de los modelos booleanos son fáciles, basta con cambiar los parámetros asociados a las primitivas (tamaño, forma, situación y orientación).	Las transformaciones locales son difíciles de implantar en los entornos interactivos.
	Mayor facilidad de descripción que en los modelos B-Rep.	El poder expresivo de los esquemas de modelado booleano es inferior al de los B-Rep.
	Cualquier sistema que utilice CSG como esquema de representación interno, puede suministrar un modelo B-Rep ya que la conversión es sencilla. El sistema puede soportar, por tanto, una amplia variedad de aplicaciones.	Al ser modelos no evaluados, se precisan los evaluadores de fronteras, que requieren mucho tiempo de cálculo. Sin embargo, como los pasos básicos son muy simples, son buenos candidatos para ser implantados en VLSI.

Cuadro 3.1: Ventajas e inconvenientes de CSG con respecto a B-Rep

Esquema	Ventajas	Inconvenientes
B-Rep	Mayor poder expresivo que el CSG, ya que el rango de primitivas es más amplio.	Estructura compleja y gran cantidad de información.
	Fáciles de visualizar.	La comprobación de la integridad de los modelos es complicada.
	Facilidad en la ejecución de modificadores locales. El usuario dispone de más operaciones que en el CSG.	Algunos operadores, como los booleanos, pueden generar modelos abiertos (no íntegros), al operar sobre modelos cerrados.
	Los poligonales son modelos evaluados, por lo que se tiene fácil acceso a la información geométrica y topológica, aunque los algoritmos se complican mucho cuando se trabaja con superficies curvas.	La descripción directa de los modelos es muy tediosa, por lo que se suelen utilizar conversores desde otros esquemas a B-Rep.
	La visualización en modelo alámbrico es inmediata.	Es posible la pérdida de modelos originales al operar incorrectamente con ellos.

Cuadro 3.2: Ventajas e inconvenientes de B-Rep con respecto a CSG

- **Métodos avanzados.**

  - **Superficies de subdivisión.**

    Es una representación intermedia entre las superficies poligonales y las superficies generadas mediante curvas.

    Toda superficie de subdivisión empieza con una superficie poligonal llamada red de control, la cual se subdivide en más polígonos, y así sucesivamente, empleando una aproximación adaptativa que añade únicamente más resolución en las zonas que lo requieren.

    Las superficies de subdivisión proporcionan la flexibilidad de las superficies poligonales y la resolución de los parches.

    No son superficies paramétricas, es decir, no se definen mediante una fórmula, sino de forma algorítmica y su topología es irregular.

    La generación de superficies de subdivisión tiene principalmente dos fases:

    - Refinamiento: partiendo de la red de control, se crean nuevos vértices que se reconectan para crear nuevos triángulos.
    - Suavizado: calcula la posición de algunos (o todos) los vértices de la nueva malla. Las reglas de suavizado determinan diferentes esquemas y la superficie final, dónde el más conocido es el de Catmull-Clark.

    El algoritmo de Catmull-Clark se divide en tres pasos: [CC78]

    - Para cada una de las caras de la malla, se generan los nuevos ”vértices de cara” como la media de todos los puntos de la cara original. Cada cara tendrá como mínimo tres vértices.
    - Se generan los nuevos ”vértices de arista”, como la media de los puntos medios de las aristas originales junto con los ”vértices de cara” nuevos, de las caras adyacentes a la arista.
    - Se mueven los vértices originales de la red de control utilizando los vértices vecinos (véase la ecuación 3.3). Se conecta a continuación, cada vértice

nuevo con sus nuevos vértices de arista adyacentes.

$$v^{i+1} = \frac{N-2}{N}v^i + \frac{1}{N^2}(\sum_{j=0}^{N-1}(e_j^i + f_j^{i+1})) \tag{3.3}$$

- ◇ N es el número de aristas que salen de un vértice, también llamado valencia.
- ◇ v^i es el vértice v en el nivel de recursión i.
- ◇ e_j^i son los vértices vecinos de v, en el nivel de recursión i, estando j en el rango [0,N-1].
- ◇ f_j^i son los vértices de cara vecinos de v en el nivel de recursión i.

- **Superficies implícitas (Metasuperficies).**

  Una superficie implícita es aquella definida mediante funciones que se evalúan en un campo contínuo.

  Un punto p formará parte de la superficie implícita si el resultado es U (valor límite) cuando el punto se evalúa en la función f (ver ecuación 3.4).

$$f(x, y, z) = f(p) = U \tag{3.4}$$

  Pueden mezclarse obteniendo puntos que cumplan las condiciones de cada función (normalmente U = 0) (ver ecuación 3.5).

$$f(p) = \sum_{i=0}^{n-1} h(p) \tag{3.5}$$

  En el caso concreto de metasuperficies, cada elemento viene definido, al menos, con su posición, su fuerza y su radio de acción.

  Una vez conseguido el cálculo de la fuerza para cada punto P, se tiene un campo contínuo, mientras lo que se pretende conseguir es una superficie, para lo cual se tiene que discretizar ese campo contínuo.

  La discretización se puede realizar por ejemplo, mediante la generación de polígonos Marching Cubes, aunque existe otro algoritmo que intenta mejorarlo, llamado Marching Tetrahedroms.

El algoritmo de Marching Cubes es uno de los más famosos. Consiste en describir la superficie en forma de polígonos, descritos por las intersecciones de la superficie a evaluar, con una descomposición del espacio en formas geométricas base. El algoritmo geométrico comienza realizando una descomposición del espacio afectado por la superficie, en cubos de un tamaño determinado (geometría base), estos cubos también son denominados voxels. [Fer05]

Dada la descomposición en voxels, cada uno de ellos puede encontrarse en una de estas tres situaciones:

- Completamente fuera de la superficie.
- Completamente dentro de la superficie.
- Parcialmente dentro, o sea, intersectado por la superficie.

El algoritmo, después de realizar esta clasificación, se dedicará a determinar la forma del polígono de intersección entre la ecuación de la superficie y el cubo intersectado.

El resultado de esta representación como en toda discretización, dependerá de la precisión elegida, que en este caso vendrá determinada por el tamaño de las celdas (voxels).

- **Modelado procedural.**

  Este tipo de modelado se aplica especialmente a objetos encontrados en la naturaleza, ya que simula el proceso de crecimiento natural y se describe en forma de procedimientos. [Rem05]

  - **Geometría fractal.** Especialmente efectivo en la creación de figuras aleatorias irregulares que simulan las formas encontradas en la naturaleza. El procedimiento consiste en la división de los polígonos del objeto de forma recursiva y aleatoria en muchas formas irregulares. La cantidad de división vendrá delimitada por un parámetro que indique el nivel de recursión.

    Cabe destacar el modelado de plantas, que se realiza codificando sus características mediante una serie de reglas usadas como base de su construcción. Se suelen utilizar dos técnicas:

- ◇ Técnicas orientadas al espacio.
- ◇ Técnicas orientadas a la estructura: Sistemas-L (Lyndenmayer Systems).

○ **Sistemas de partículas.** Se emplean figuras simples, normalmente esferas o puntos tridimensionales, a los cuales se les aplica atributos de crecimiento. Cuando estos atributos se simulan, las partículas tienen un comportamiento específico que da lugar a trayectorias de partículas. Debido al crecimiento de las partículas con el paso del tiempo, sus trayectorias definen una figura que se convierte en un modelo tridimensional. Este crecimiento puede ser aleatorio o controlado.

Los sistemas de partículas se utilizan generalmente para crear fenómenos naturales, cuya figura cambia constantemente, como las tormentas de nieve, las nubes, el fuego, etc.

Para crear un sistema de partículas los pasos a seguir son:

- ◇ Crear un emisor de partículas.
- ◇ Determinar el número de partículas.
- ◇ Definir la forma y el tamaño de las partículas.
- ◇ Definir el movimiento inicial de las partículas. Se suele definir la velocidad, dirección, rotación y aletoriedad con que las partículas salen del emisor.
- ◇ Modificar el movimiento de las partículas.

○ **Modelado por simulación física.** La forma de modelar materiales naturales cuya forma está en constante cambio es simularla.

Este método se utiliza para fenómenos como el fuego, humo, viento, nubes, etc.

- **Modelado basado en trazos.** "*Sketch Modelling*".

Este tipo de modelado consiste en partir de trazos realizados por el usuario (trazos 2D), tanto para la generación de un objeto, como para la modificación del mismo. Esto proporciona una representación homogénea ya que todo el modelado se realiza intuitivamente de la misma forma.

Para la creación de un modelo nuevo, inicialmente se procederá al dibujado del contorno 2D y se obtendrá la malla poligonal asociada a ese contorno. A la hora de modificar la malla obtenida, se realizarán de la misma forma trazos 2D para realizar las operaciones correspondientes sobre ésta.

Se profundiza sobre este tipo de modelado más adelante.

3.1.4. Operadores.

- Los sistemas de modelado CSG se caracterizan por la generación de modelos mediante el uso de operadores booleanos, que permiten (al usuario) realizar una serie de combinaciones para obtener el modelo final.

  Podemos definir tres operaciones booleanas básicas:

  - Unión.
  - Intersección.
  - Diferencia.

- En sistemas de modelado B-Rep se pueden aplicar una serie de operadores, además de los operadores booleanos citados anteriormente: [GM05b]

  - **Extrusión (Extrusion o lofting):** Consiste en generar una superficie 3D extendiendo una forma 2D a lo largo de un eje.
  - **Barrido (Sweeping):** Es una generalización de la extrusión, ya que consiste en generar un objeto 3D extendiendo una forma a lo largo de un camino, si el camino es recto estamos aplicando el operador de extrusión.
  - **Revolución (Revolve o lathe):** Es una variación de las superficies de barrido. Se desplaza una forma 2D alrededor de un eje.
  - **Skinning:** Se genera una superficie curva que cubre secciones cerradas, también llamadas ”rebanadas”. Es similar al barrido, sólo que la curva puede cambiar su geometría mientras avanza por el camino.

- **Mezclado (Blending):** Es un método especial para combinar dos superficies, que no tienen por qué estar en contacto. La nueva superficie curva que se genera se puede controlar mediante puntos de control.
- **Truncado (Beveling):** Consiste en recortar un vértice situado entre dos superficies adyacentes reemplazándolo por un plano inclinado.
- **Redondeado (Rounding):** Es una versión del truncado que suaviza los vértices y aristas del objeto.

- Además de todos los operadores comentados anteriormente, se van construyendo herramientas y aplicaciones que permiten facilitar la modificación de un modelo para conseguir el objetivo final. Estos operadores pueden denominarse operadores avanzados:

  - **Sculpter:** Este tipo de operador ha tenido bastante relevancia en los últimos años ya que permite mediante trazos 2D, simulando un pincel o un dedo, modificar la malla 3D como si se estuviese esculpiendo en ella, simulando la arcilla.

    Existen algunas herramientas que implementan ya esta técnica, que son ZBrush [ZBr97], herramienta que además de exculpir un objeto 3D permite pintar la malla generando de esta forma su material o textura.

    Otra herramienta de este tipo en el mercado es BBrush, que es un script integrado en la herramienta de síntesis 3D Blender [BBr97].

3.1.5. Modelado basado en trazos.

3.1.5.1. Estado del arte.

El problema del modelado de figuras complejas surge al ser un proceso muy costoso en el tiempo y por lo general poco intuitivo. Para la realización de esta tarea existen técnicas de alta calidad, pero de alto coste, como pueden ser el modelado en arcilla y la utilización de escáneres 3D para la digitalización del modelo. Estas técnicas son utilizadas en producciones profesionales de alto presupuesto, que posteriormente se mejoran utilizando técnicas software.

El modelado se realiza habitualmente partiendo de primitivas básicas como pueden ser esferas, cilindros, conos, ... y modificando éstas mediante la combinación de múltiples operaciones y/o la modificación directa de sus vértices o puntos de control.

Actualmente existen herramientas, aún en investigación, cuyo objetivo es facilitar esta tarea dando una alternativa distinta al modelado a partir de primitivas. Este tipo de herramientas tienen como objetivo hacer más rápido y cómodo el prototipado inicial del modelo para su posterior tratamiento mediante las técnicas clásicas.

La idea principal es el uso de trazos libres (bocetos) generados por el usuario mediante un ratón o tabla digitalizadora, es decir, el usuario dibuja en pantalla una silueta 2D, y el sistema genera automáticamente su malla poligonal correspondiente, proporcionando mediante la misma técnica la posibilidad de realizar determinadas operaciones sobre la malla 3D [GG04].

Algunos trabajos sobre este tipo de modelado, usan trazos 2D para trabajar sobre la interfaz de usuario, y por ejemplo para manejar curvas de interpolación [T.94].

Algunas aplicaciones que utilizan estas técnicas para modelado 3D están empezando a cobrar importancia, una de ellas es SKETCH [RZK96], que utiliza una interfaz basada en gestos para modelar rápidamente sólidos, utilizando primitivas simples. Este sistema trabaja bien con el uso de primitivas, ya que permite modelar sólidos de forma intuitiva, pero no puede ser utilizado para figuras libres como modelos orgánicos.

Otra aplicación que trabaja para que la interfaz de un modelador sea más intuitiva mediante el uso de trazos 2D es Mod3D [NR00]. En esta herramienta, cuando el usuario así lo indica, se produce una interpretación del trazo o trazos llevados a cabo. La validación del trazo se realiza mediante dos patrones básicos que son líneas y círculos, y su interpretación se realiza mediante la comparación del trazo con los trazos almacenados en una tabla de trazos válidos. Esta herramienta es muy limitada en cuanto a la generación de componentes, ya que se obtienen primitivas básicas mediante trazos comparándolas con los previamente almacenados en una tabla. Mod3D también está limitado a la generación de sólidos a partir de primitivas, ya que no permite creación de formas libres.

En 1996, Takeo Igarashi creó una herramienta, Teddy [Iga95], que realmente consigue el ”Modelado 3D basado en bocetos”. Esta herramienta sí nos permite la generación de una malla 3D poligonal partiendo del contorno dibujado por el usuario en la interfaz. Además

incluye operaciones como extrusión y bending siguiendo el mismo planteamiento, es decir, las operaciones modifican la malla también mediante trazos realizados por el usuario. Los resultados obtenidos tienen un aspecto muy natural, se obtiene una malla poligonal en todos los casos y la resolución de la malla es fija, es decir, es independiente del tipo de figura (ver figura 3.1).

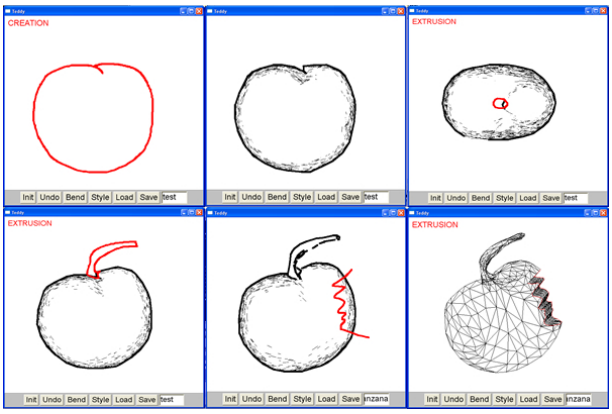

Figura 3.1: Modelado en Teddy.

3.1.5.2. Pasos en la generación de un objeto en Teddy.

En Teddy, los pasos que se siguen para la generación de una malla 3D son los siguientes [IM99], [GG04]:

- El usuario realiza un trazo libre dibujando un contorno, este trazo debe ser cerrado y no debe producir intersecciones consigo mismo, ya que en este caso el sistema fallaría.
- A partir del contorno dibujado, se crea un polígono mediante la conexión del punto de comienzo y del punto final del trazo. Se procede en este paso a la identificación de las aristas del polígono, el cuál será la silueta de la malla 3D que se obtenga.

  Para la generación del polígono, algunos parámetros son decididos por el programador para controlar como se realiza este proceso:

  - Mínima y máxima longitud de cada arista del polígono.
  - Mínima longitud del trazo de entrada que es considerada como válida.

- Ángulo mínimo que es requerido entre aristas consecutivas para crear una nueva arista.

Una vez obtenido el polígono, los ejes obtenidos en este paso van a ser llamados ejes externos, y se llamarán ejes internos a los creados posteriormente como resultado de la triangulación (ver figura 3.2).

- Se procede a la Triangulación de Delaunay que se puede enunciar como sigue [el06]:

  Una red de triángulos, es una triangulación de Delaunay si todas las circunferencias circunscritas de todos los triángulos de la red son vacías.

  Esta sería la definición para espacios bidimensionales, para espacios tridimensionales se podría ampliar utilizando una esfera circunscrita en vez de una circunferencia circunscrita.

  La circunferencia circunscrita de un triángulo es la circunferencia que contiene los tres vértices del triángulo.

  Según la definición de Delaunay la circunferencia circunscrita es vacía, si no contiene otros vértices aparte de los tres que la definen.

  Las triangulaciones de Delaunay tienen las siguientes propiedades:

  - La triangulación forma la envolvente convexa del conjunto de puntos.
  - El ángulo mínimo dentro de todos los triángulos está maximizado.
  - La triangulación es unívoca si en ningún borde de la circunferencia circunscrita hay más de tres vértices.

  Una vez triangulado el polígono, el conjunto de triángulos forman un plano delimitado por el polígono de la silueta pintada por el usuario.

  Se clasifican los triángulos obtenidos después de esta triangulación como (ver figura 3.2):

  - Triángulos T: triángulos con dos aristas externas y una interna.
  - Triángulos S: triángulos con dos aristas internas y una externa.

- Triángulos J: triángulos con todas sus aristas internas.

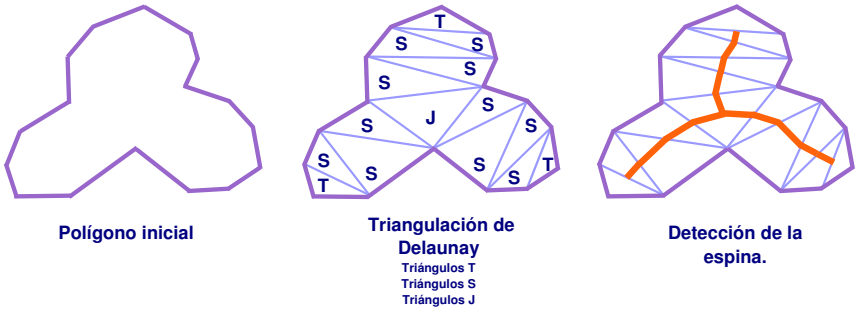

Figura 3.2: Polígono inicial y generación del plano.

- Una vez obtenido el plano formando la figura deseada, se procede a la detección de la espina dorsal del objeto (spine), como el eje central del objeto. La espina dorsal se obtiene del siguiente modo (ver figura 3.2):
  - Para cada triángulo S, se calcula el punto medio de sus dos aristas internas y se conectan.
  - Para cada triángulo J, se calcula el punto medio de cada arista (todas son internas) y el punto medio del triángulo, quedando conectado con cada uno de los anteriores calculados.
- En este instante se tiene un conjunto de triángulos donde algunos son innecesarios, con lo cual se realiza un podado de aristas insignificantes y retriangulación de la malla.

  Para el podado de aristas irrelevantes se examina cada triángulo T, expandiéndolo en regiones progresivamente más grandes, combinándolo con triángulos adyacentes.

  El algoritmo de poda que se implementa en Teddy se describe a continuación.

  Para cada triángulo T, al que llamaremos X:

  1. Construir un semicírculo que tenga de diámetro la longitud de la arista interior de X, y cuyo centro sea el centro de dicha arista (ver figura 3.3).

2. Si los vértices de X se encuentran en o dentro del semicírculo, quitamos la arista interna de X y fusionamos X con el triángulo que se encuentre al otro lado de la arista (ver figura 3.3).

   Si algún vértice de X se encuentra fuera, ir al paso 4 (ver figura 3.4).

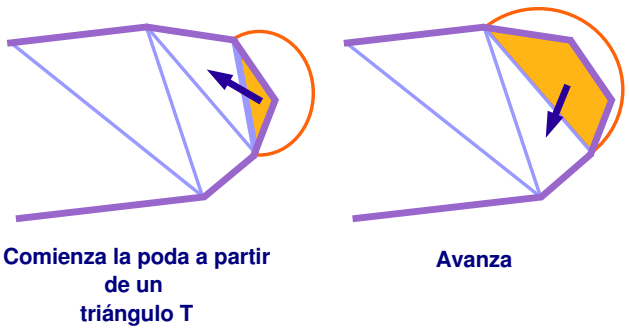

Figura 3.3: Algoritmo de poda.

3. Si el triángulo adyacente nuevo es un triángulo S, X tiene ahora una arista externa más y una nueva arista interior, volver al paso 1.

   Si el triángulo adyacente nuevo es un triángulo J ir al paso 4 (ver figura 3.5).

4. Si los vértices de X se encuentran fuera del semicírculo, se triangula X con un abanico de triángulos que irradian en el punto medio de la arista interior (ver figura 3.4).

   Si por el contrario, el triángulo adyadente nuevo es un triángulo J, se triangula X con un abanico desde el punto medio del triángulo J (ver figura 3.5).

   FIN.

- La espina dorsal podada es generada mediante la conexión de los puntos medios de las aristas internas de los triángulos S y triángulos J (exceptuando los triángulos nuevos generados en la poda) (ver figura 3.6).

- El siguiente paso es subdividir los triángulos S y triángulos J para prepararlos para la elevación. Estos triángulos se dividen en la espina dorsal de modo que ahora se tiene

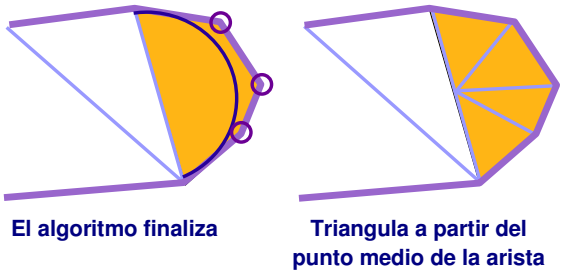

Figura 3.4: Finaliza el algoritmo de poda. Vértices fuera del semicírculo.

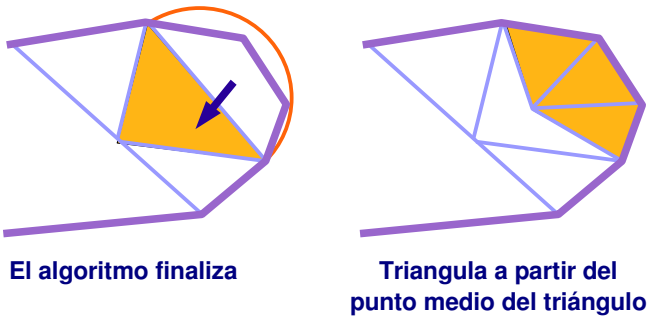

Figura 3.5: Finaliza el algoritmo de poda. Encontrado triángulo J.

una malla triangular 2D completa entre la espina dorsal y el perímetro del polígono inicial (ver figura 3.6).

- Lo que se tiene hasta ahora es una triangulación 2D con una espina dorsal que representa el eje mediano de la forma inicial. Para producir una forma 3D, se utiliza la espina dorsal para crear un perfil de alturas, es decir, para cada vértice de la espina dorsal, se crean dos nuevos vértices que tengan las mismas coordenadas X e Y, con una coordenada Z positiva y otra negativa, igual a la distancia en valor absoluto entre el vértice y la silueta del objeto.

  El problema que surge es cómo computar la distancia de la espina dorsal a la frontera del

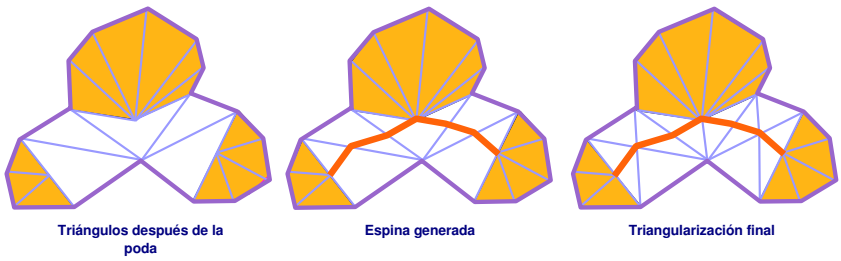

Figura 3.6: Malla triangulada 2D completa.

polígono. Lo ideal sería construir secciones representativas exactas de la forma en cada vértice de la espina dorsal. Como esto es demasiado costoso, se hace un promedio de las distancias entre el vértice y los vértices externos que están conectados directamente a ese vértice (ver figura 3.7).

- Una vez elevada la espina dorsal, se procede a la construcción de la malla. Cada arista interna de cada triángulo del abanico, excluyendo las aristas de la espina dorsal, es convertida a un cuarto de óvalo. El sistema construye la malla poligonal apropiada mirando las aristas elevadas vecinas. El sistema elimina el plano generado inicialmente a partir de la silueta.

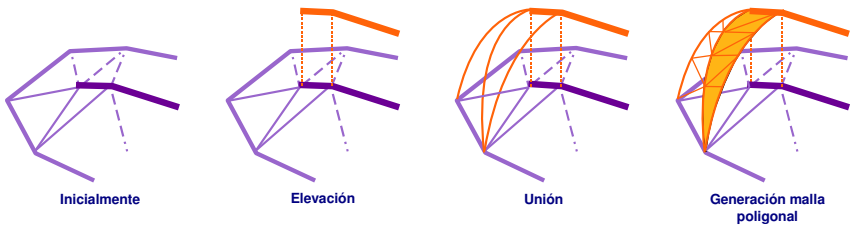

Figura 3.7: Elevación de la espina dorsal.

- En este momento se tiene la mitad de la figura. La malla obtenida es copiada al otro lado para hacer la malla cerrada y simétrica, y el sistema, posteriormente, aplica algoritmos de refinamiento de mallas para eliminar pequeñas aristas y pequeños triángulos.

Tabla de polígonos		
Caras	Vértices	Coordenadas
C1	V1	X1 Y1 Z1
C1	V2	X2 Y2 Z2
C1	V3	X3 Y3 Z3
C2	V1	X1 Y1 Z1
⋮		
C4	V1	X1 Y1 Z1
C4	V2	X2 Y2 Z2

Cuadro 3.3: Estructura de polígonos.

3.2. Técnicas.

3.2.1. Estructuras de datos.

Todos los esquemas de modelado poligonal representan las caras de modo explícito en sus estructuras de datos. El resto de información geométrica y topológica puede quedar registrada de muchas formas diferentes, con mayor o menor grado de redundancia. Algunas de las estructuras de datos más utilizadas se comentan a continuación.

- **Estructuras basadas en polígonos.**

  En esta representación los objetos son una colección de polígonos y, a su vez, cada polígono es una secuencia de coordenadas (x,y,z). Esta secuencia puede almacenarse tanto en una tabla, donde cada polígono tendrá un descriptor que lo identifique, como en una lista encadenada.

  Aunque estas estructuras son muy sencillas, generan mucha información redundante, ya que las coordenadas de un mismo vértice se han de almacenar tantas veces como caras convergen en él. Por lo tanto, en los modelos con muchos polígonos (caso normal), no es la solución idónea.

- **Estructuras basadas en vértices.**

Tabla de vértices		Tabla de caras	
Vértices	Coordenadas	Caras	Vértices
V1	X1 Y1 Z1	C1	V1 V2 V3 V4
V2	X2 Y2 Z2	C2	V6 V2 V1 V5
V3	X3 Y3 Z3	C3	V7 V3 V2 V6
V4	X4 Y4 Z4	C4	V8 V4 V3 V7

Cuadro 3.4: Estructura de vértices.

Tabla de aristas		Tabla de vértices		Tabla de caras	
Aristas	Vértices	Vértices	Coordenadas	Caras	Aristas
A1	V1 V2	V1	X1 Y1 Z1	C1	A1 A2 A3 A4
A2	V2 V3	V2	X2 Y2 Z2	C2	A9 A6 A1 A5
A3	V3 V4	V3	X3 Y3 Z3	C3	A10 A7 A2 A6
A4	V4 V1	V4	X4 Y4 Z4	C4	A11 A8 A3 A7

Cuadro 3.5: Estructura de aristas.

La redundancia del caso anterior puede ser eliminada si se consideran los vértices como elementos independientes de los objetos, y luego son asociados a las caras.

La lista de vértices de cada cara viene dada en un órden consistente, de esta forma se obtiene la orientación, también llamado vector normal, de las caras, información muy útil a la hora de eliminar aristas y caras ocultas.

La información que se deja implícita se calcula cuando se necesita, lo que supone un aumento del tiempo, mientras que registrar toda la información de manera explícita produce un consumo mayor de espacio. Se debe buscar un equilibrio entre espacio y tiempo en función de la velocidad de cálculo y de la memoria disponible.

- **Estructuras basadas en aristas.**

  Con este método, las caras de los objetos se representan como secuencias de aristas que forman un camino cerrado.

En la tabla de aristas se indica la orientación de cada arista, por ejemplo, la arista A1 se considera orientada desde el vértice V1 hasta el V2.

- **Estructura winged-edge.**

  Las estructuras simples hacen que el número de cálculos sea muy elevado. Por ejemplo, calcular qué dos caras comparten una arista para verificar la integridad del modelo requiere obtener la lista de aristas de todas las caras.

  Las estructuras sofisticadas, intentan incluir la mayor cantidad de información posible dentro de un volumen de datos razonable y de rápido acceso, para disminuir el costo de los cálculos. Entre las estructuras más elaboradas se encuentra la conocida como winged-edge, desarrollada por Baumgart en 1972.

  Además de registrar información sobre las aristas y las caras, también quedan indicados los bucles (caminos cerrados) que forman las aristas al ser recorridas en sentido positivo y negativo.

  - Creación de la estructura winged-edge.

    Cada arista separa dos caras. Dada una arista a, y siendo C', C'' las caras que separa; si se recorre la arista en el sentido de orientación positivo, al llegar al vértice se encuentra la arista a' que pertenece a la cara C'; si se recorre a en el sentido negativo, se encuentra a'', que pertenece a C''.

    El sentido de recorrido y las aristas localizadas al llegar a los vértices constituyen la información sobre los bucles que se almacenan.

    Para crear la tabla de aristas:

    1. Se establece un sentido de giro consistente, por ejemplo, el sentido de giro de las agujas del reloj considerando los objetos desde el exterior.
    2. Por cada arista a_i, se busca el par de caras C_a,C_b que separa y se establece el sentido de giro positivo de la arista. Es indiferente el sentido elegido, aunque una vez establecido debe permanecer fijo. En este caso, imaginemos que el recorrido positivo viene dado por (V_a, V_b), que indica que la arista se recorre en sentido positivo cuando va del vértice V_a al V_b.

Tabla de vértices		Tabla de aristas				Tabla de caras	
Vértices	Coordenadas	Aristas	Vértices	R+	R-	Caras	Inicio
V1	X1 Y1 Z1	A1	V1 V2	A2	A5	C1	A1
V2	X2 Y2 Z2	A2	V2 V3	A6	A3	C2	⋮
V3	⋮	A3	⋮	⋮	⋮	⋮	

Cuadro 3.6: Estructura de winged-edge.

3. Se recorre cada arista a_i en sentido positivo hasta llegar al vértice. Entonces se elige una arista de la cara C_a o de la cara C_b, de forma que el recorrido por la nueva arista sea en el sentido de recorrido consistente de la cara (en este caso el sentido de las agujas del reloj). Sólo una de las aristas cumple esa condición, que será la que se almacene en la estructura (puntero R+).
4. Se vuelve a recorrer la arista a_i, pero esta vez en sentido negativo. Cuando se llega al vértice opuesto, se aplica el mismo criterio que anteriormente, y la arista seleccionada quedará indicada por el puntero R-.

Se puede ver un ejemplo de esta estructura en la figura 3.8 y en la tabla de la estructura de winged-edge, que se presentan los datos.

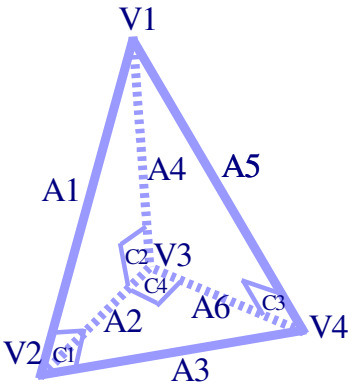

Figura 3.8: Ejemplo de estructura winged-edge.

- **Cuantización.**

En estas representaciones se almacenan grandes volúmenes de datos que a veces llegan a ser inmanejables, por lo tanto hay alternativas de compresión de datos como la cuantización.

La cuantización es una técnica de almacenamiento con pérdida de información que consiste en el uso de tipos de datos con menor precisión.

En casos extremos se hacen uso de ”compresiones salvajes” como por ejemplo pasar de float o doble a unsigned short (16 bits típicamente) o unsigned bytes (8 bits).

Para la cuantización se utiliza:

$Vcomprimido = TamTipoComp * ((Voriginal - Min)/(Max - Min))$

$Prec = (Max - Min)/TamTipoComp$

$Vdescompr = Prec * Vcomprimido + Min$

Se puede ver con este ejemplo:

Altura de un individuo con codificación de 1 byte.

$Vcomprimido = 255 * ((X - 1{,}0)/(2{,}20 - 1{,}0))$

$Prec = (2{,}20 - 1{,}0)/255 = 0{,}0047m$

$X = 0{,}0047 * Vcomprimido$

Si por ejemplo $X = 1{,}82...$:

$Vcomprimido = 255 * (0{,}82/1{,}20) = 174$

$X = 0{,}0047 * 174 = 1{,}818$

3.2.2. Transformaciones lineales en 3D.

Las características geométricas que definen un cuerpo en el espacio tridimensional, pueden ser alteradas mediante transformaciones 3D. Un tipo particular de éstas, son las transformaciones lineales que, una vez aplicadas, mantienen el paralelismo entre las aristas que forman cada objetoy que se estudiarán a continuación.

Se comenzará con el estudio de algunos conceptos previos para su correcta definición:

- Sistemas de referencia.

  Para visualizar los objetos, se deben situar en un sistema universal de referencia (SUR), en el cual el eje Z puede tener dos orientaciones, la situada hacia el observador y la que no. Todas las coordenadas de los distintos objetos han de estar dadas en uno de estos sistemas de referencia.

- Transformaciones lineales y matrices.

  En la Informática Gráfica suele utilizarse notación matricial para escribir las transformaciones lineales de los objetos. La convención más utilizada es que el vértice que se va a transformar se exprese mediante el vector horizontal multiplicado por la matriz de transformación.

  Por ejemplo, en la expresión $(x', y') = (x, y) * M$, la matriz correspondiente a la transformación lineal sería M, el punto inicial sería (x, y) y el resultado, es decir, la ubicación del punto finalmente en el sistema de referencia sería (x', y').

  Si se aplica un conjunto de transformaciones a un modelo poligonal de n vértices, tendrá que aplicarse el conjunto de transformaciones lineales a los n vértices del polígono. En general, se aplican las transformaciones a todos los puntos significativos de los objetos.

- Sistemas homogéneos.

  Se tiene el vector de un punto inicial $V = (x, y)$, el vector de translación $T = (t_x, t_y)$ y las coordenadas resultantes después de la traslación se calculan como se ve en la ecuación 3.6:

$$V' = V + T(x', y') = (x, y) + (t_x, t_y) \tag{3.6}$$

  Esto permite ser extendido a cualquier dimensión.

  Si S y R se suponen matrices de escalado y giro, respectivamente, vemos en la ecuación 3.7:

$$V' = V * S$$

$$V' = V * R \tag{3.7}$$

Las traslaciones lineales de los puntos en el espacio se efectúan sumando, mientras que los giros y los cambios de escala se consiguen multiplicando. Esta heterogeneidad en los operadores, supone un problema a la hora de generalizar los procesos de transformación, para evitarlo, se utilizan sistemas de referencia homogéneos.

Un sistema de coordenadas homogéneo es el resultante de añadir una dimensión extra a un sistema de referencia dado. De esta forma los vectores homogéneos de los puntos inicial y final serán $V = (x, y, w)$ y $V' = (x', y', w)$. Por sencillez $w = 1$.

En un sistema homogéneo, las traslaciones lineales de los puntos del plano quedarían expresadas como en la ecuación 3.8, si se utiliza una matriz de traslación T apropiada como se ve en la ecuación 3.9.

$$V' = V * T \tag{3.8}$$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{3.9}$$

De esta forma es fácil comprobar viendo las ecuaciones 3.8 y 3.9, que $(x', y', 1) = (x, y, 1) * T$.

- Composición de matrices.

  La composición de matrices, matemáticamente, es la multiplicación de matrices en un orden determinado.

  Se tiene el punto $V = (x, y, z, 1)$ del espacio, expresado en un sistema homogéneo. Al hacer:

$$(x', y', z', 1) = (x, y, z, 1) * T$$

$$(x'', y'', z'', 1) = (x', y', z', 1) * R \tag{3.10}$$

Se consigue mover primero el vértice V y luego girarlo. Se puede llegar al mismo resultado final multiplicando el vector V por la matriz resultante de componer T y R, siendo $M = T * R$ se tiene que:

$$(x'', y'', z'', 1) = (x, y, z, 1) * M \tag{3.11}$$

El orden en el que se multiplican las matrices es importante ya que el producto entre matrices no es conmutativo aunque sí es asociativo.

Siendo M_n la matriz compuesta o neta resultante de la composición de las matrices T_1, R, T_2 y S, es decir $M_n = T_1 * R * T_2 * S$, al multiplicar un punto por esta matriz se obtiene el mismo resultado que si se multiplicase sucesivamente por las matrices que componen M_n.

En 3D, la expresión general de la matriz neta sería:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ d_x & d_y & d_z & 0 \end{bmatrix} \tag{3.12}$$

Donde la submatriz $A_{i,j}$ representa el cambio de escala y la rotación neta y D_i el vector de desplazamiento neto.

- **Transformaciones lineales tridimensionales.**

  - Traslación.

    La traslación de un objeto consiste en moverlo en una dirección determinada.

En 3D, el sistema de referencia homogéneo tendrá cuatro dimensiones, por lo que la traslación del punto $V = (x, y, z, 1)$ quedará indicada como $V = (x', y', z', 1) = (x, y, z, 1) * T$ siendo la matriz de traslación:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 0 \end{bmatrix} \tag{3.13}$$

Para realizar la traslación inversa a la efectuada mediante la matriz T, se ha de aplicar la matriz inversa T^{-1}, que se obtiene cambiando el signo del vector de traslación.

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 0 \end{bmatrix} \tag{3.14}$$

- Escalado.

  Dentro del espacio de referencia, los objetos pueden modificar su tamaño relativo en un eje, en dos o en los tres. Para ello se ha de aplicar la matriz de escalado, que viene dada por:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

  De esta forma, el cambio de escala del punto $V = (x, y, z, 1)$ en el sistema homogéneo quedará indicado por $V' = (x', y', z', 1) = (x, y, z, 1) * S$.

Cuando $s_x = s_y = s_z$, el cambio de escala es uniforme; en cualquier otro caso el cambio de escala será no uniforme.

Para obtener el cambio de escala inverso al realizado aplicando la matriz S, basta con multiplicar los puntos finales por la matriz inversa de S, es decir S^{-1}, que sería:

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.16}$$

Los cambios de escala no uniformes deforman los objetos, por lo que pueden resultar interesantes (siempre que se realicen de forma controlada); por el contrario, el escalado uniforme no deforma los objetos, por lo que suele utilizarse con más frecuencia.

- Rotación.

  Se quiere rotar el punto $V = (x, y, z)$ un ángulo α alrededor del eje Z para obtener V' como $V' = (x', y', z', 1) = V * R_z$, siendo R_z:

$$R_z = \begin{bmatrix} cos(\alpha) & sen(\alpha) & 0 & 0 \\ -sen(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

  Como se puede ver, la coordenada Z permanece constante ya que se rota el vértice alrededor de este eje.

  De la misma manera se derivan las ecuaciones de giro alrededor del eje X e Y.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & sen(\alpha) & 0 \\ 0 & -sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$R_y = \begin{bmatrix} cos(\alpha) & 0 & -sen(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ sen(\alpha) & 0 & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

Al girar con R_x la coordenada permanece constante en X, de igual forma, utilizando R_y, el giro no varía en Y.

Para deshacer el giro, se ha de girar el punto un ángulo $-\alpha$. La matriz de giro inversa es la misma, pero con el ángulo cambiado de signo. Otra forma rápida de obtener R^{-1} es encontrando la matriz traspuesta de R (intercambiando filas por columnas), es decir, $R^{-1} = R^T$.

3.2.3. Tecnologías.

En este apartado, se estudiarán las tecnologías utilizadas en MOSKIS 3D, que son:

- Librería gráfica OpenGL.
- Herramienta de producción 3D Blender.
- Lenguaje de programación Python.
- API Blender-Python.

3.2.3.1. OpenGL.

- **Introducción.**

  OpenGL es una biblioteca gráfica desarrollada por Silicon Graphics Incorporated (SGI). Sus siglas significan ”Open Graphics Library”.

  - Es el estándar más extendido.

    OpenGL fue el primer entorno de desarrollo portable para aplicaciones gráficas 2D y 3D. Desde su aparición en 1992, se ha ido extendiendo rápidamente y en una gran variedad de plataformas.

OpenGL incorpora un amplio sistema de renderizado, mapeado de texturas y efectos especiales entre otras funciones.

- Alta calidad visual y funcional.

  Cualquier aplicación 3D requiere una representación visual de alta calidad y un alto rendimiento que es proporcionado por OpenGL. OpenGL está difundido en gran variedad de mercados, como pueden ser CAD/CAM/CAE, entretenimiento, medicina, realidad virtual, etc.

- Ventajas orientadas al desarrollo.
  - Estándar: Un consorcio independiente, la OpenGL Architecture Review Board, revisa sus especificaciones. Es la única biblioteca gráfica verdadermente abierta y un estándar multiplataforma.
  - Estable: Las nuevas incorporaciones en la especificación están muy controladas y las actualizaciones propuestas se anuncian para poner en sobreaviso a los desarrolladores. Los requisitos de compatibilidad con versiones anteriores se aseguran de que las utilidades anteriores no lleguen a ser obsoletas.
  - Fiabilidad y portabilidad: Todas las aplicaciones OpenGL producen resultados visuales consistentes en cualquier software que soporte OpenGL independientemente del sistema operativo y el sistema de ventanas.
  - Contínuo desarrollo: Compatible con los últimos avances en hardware debido al contínuo crecimiento del API.
  - Escalable: OpenGL está destinado tanto para PC's, estaciones de trabajo o supercomputadores. Como consecuencia, las aplicaciones son escalables a cualquier clase de máquina que el desarrollador elija.
  - Fácil de utilizar: OpenGL está bien estructurado con un diseño intuitivo. Normalmente son necesarias pocas líneas de código. Se han desarrollado una serie de extensiones que facilitan el uso y añaden funcionalidades a las básicas, como GLU, GLUT y GUI. El programador por otro lado, no tendrá que preocuparse de las especificaciones del hardware ya que OpenGL se encarga de esto.

- Bien documentado: Una amplia documentación y ejemplos están disponibles.

- **Modelo conceptual de OpenGL: ”cámara sintética”**

  El concepto de ”cámara sintética” que utiliza OpenGL, significa imaginar un objeto en una determinada posición y filmarlo con una cámara.

  En la figura 3.9 aparecen los elementos descritos a continuación:

  - Centro de proyección: es el punto desde el que el observador mira el mundo, o desde el que una cámara lo está filmando.
  - Plano de proyección: proyección del mundo. Se pasa de coordenadas del mundo (coordenadas 3D) a coordenadas del plano (coordenadas 2D).

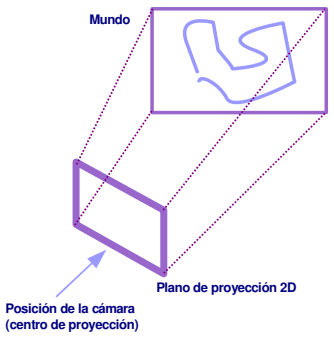

Figura 3.9: Modelo conceptual de OpenGL.

Este modelo conceptual se compone de tres elementos principalmente:

- Luces: para poder ver el mundo. Las luces se definen principalmente por su localización, intensidad y color.
- Cámara: es nuestro ”punto de vista” del mundo en cada momento. La cámara se caracteriza por su posición, orientación y apertura o ”campo visual”, es decir, la cantidad de mundo que la cámara podrá ver y por tanto, proyectar.
- Objetos: que formen parte de nuestro mundo, y que por tanto sean filmados por la cámara. Los objetos tendrán como características el color, material, etc.

- **Pipeline gráfico.**

  El pipeline gráfico es el proceso que siguen los elementos del modelo conceptual hasta que se muestra un resultado por pantalla (véase figura 3.10).

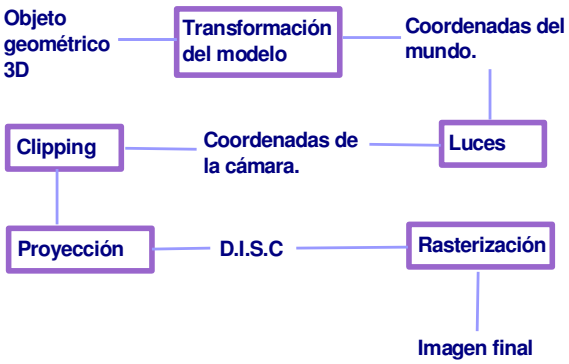

Figura 3.10: Pipeline gráfico.

- Objeto geométrico: el mundo 3D se va a componer de primitivas geométricas, como puntos, líneas, polígonos, etc.
- Transformación del modelo: se encarga de rotar, trasladar y escalar cualquier objeto para que sea dibujado por pantalla. OpenGL realiza las funciones, multiplicando la geometría por varias matrices, cada una para cada proceso.
- Coordenadas del mundo: la cámara es independiente del mundo. Las coordenadas de la geometría del mundo 3D (vértices) son independientes de las coordenadas de la cámara.
- Luces: se iluminan los objetos para ser vistos desde la cámara.
- Coordenadas de cámara: después de aplicar luces, ya vemos la escena, con lo cual ya sabemos la localización de los objetos con respecto de la cámara.
- Clipping: oculta aquello que está en el mundo pero que no se desea en ese momento. Es como recortar parte de la escena.
- Proyección: paso de coordenadas 3D del mundo a las coordenadas 2D del plano de proyección.

- D.I.S.C: Device Independent Screen Coordinates. Tras proyectar la imagen, tenemos coordenadas independientes del dispositivo. Antes de asociar la imagen con algún tipo de pantalla, se encuentran los datos de la imagen en el frame buffer, que es la zona de memoria destinada a almacenarla. Según la resolución de la pantalla sea mayor o menor, un punto del mundo 3D ocupará más o menos píxeles.
- Rasterizacion: consiste en asociar los puntos a píxeles en la pantalla.
- Imagen: ya se tiene la imagen final y concluye el proceso.

El pipeline gráfico puede ser implementado de forma hardware o software. En máquinas en las que se realiza via software es más lento que en máquinas dedicadas.

OpenGL tiene como ventaja, que funciona independientemente de la implementación del pipeline, únicamente se conseguirá más o menos velocidad dependiendo del sistema.

- **Tipos de funciones gráficas.**

  - Funciones primitivas: definen objetos como puntos, líneas o polígonos.
  - Funciones atributivas: definen características de lo que se dibuja, como por ejemplo el color.
  - Funciones de visualizacion: posición de la cámara, proyección de la geometría, clipping, etc.
  - Funciones de transformación: girar, rotar, escalar, etc.
  - Funciones de entrada: generación de aplicaciones interactivas con uso típico de teclado y ratón por parte del usuario.
  - Funciones de control: para interactuar en red, en aplicaciones cliente-servidor, o manejar un sistema operativo multitarea.

- **Transformaciones: multiplicación y postmultiplicación.**

  A partir de lo que se veía en el apartado de **Transformaciones lineales 3D**, existen dos convenciones en cuanto al uso de transformaciones geométricas:

- Robótica/Ingeniería: se utilizan los vectores de tipo fila, que se multiplican por la izquierda. Las matrices se ordenan de izquierda a derecha según el orden de transformaciones (véase ecuación 3.20). A esto se le llama premultiplicación de matrices.

$$(Pf) = (Pi)\,[T1]\,[T2]\,[T3]\,[T4] \tag{3.20}$$

  Dónde Pf es el punto de transformación final y Pi el inicial.

- Gráficos: los puntos se toman como vectores en columna que se multiplican a las matrices por la derecha, y además, el orden de las transformaciones, de primera a última a aplicar, es de derecha a izquierda (véase ecuación 3.21). A esto se le llama postmultiplicación de matrices.

$$[Pf] = [T4]\,[T3]\,[T2]\,[T1]\,[Pi] \tag{3.21}$$

  Dónde Pf es el punto de transformación final y Pi el inicial.

  Cabe recordar la importancia del orden de multiplicación de las matrices, ya que como se comentaba anteriormente, el producto de matrices no es conmutativo.

OpenGL, y cualquier paquete gráfico, contiene lo que se llama CTM, que es la matriz de transformación actual. Esta matriz guarda la información sobre todas las matrices que se han ido acumulando. Cualquier vértice que pase por el pipeline gráfico, será multiplicado por esta matriz para su transformación.

En OpenGL, la CTM se compone de dos matrices:

- Matriz de transformación.
- Matriz de modelado.

  Ambas son concatenadas y su producto crea la CTM.

En primer lugar, lo primero que debe hacerse en OpenGL es cargar la matriz identidad, que es el elemento neutro de la multiplicación de matrices, y con esto me aseguro que comienzo sin ninguna transformación anterior. A continuación, se pueden ir acumulando transformaciones sucesivas (véase código fuente 3.1).

Código fuente 3.1: Transformaciones en OpenGL.

```
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()
glScalef(sx,sy,sz)
glTranslatef(tx,ty,tz)
glRotatef(angulo,vx,vy,vz)
```

En OpenGL, se utiliza la convención de gráficos, por lo que la mprimera transformación que se aplicará será la última que se ha definido.

- **Cámara: proyecciones.**

  Los tipos de proyecciones planares, son las proyecciones en las que se define una dirección de visión, que va desde el observador hasta el objeto a proyectar. La dirección se establece por medio de proyectores (líneas) que cortan el plano de proyección generando así la imagen 2D final.

  Dentro de las proyecciones planares existen distintos tipos:

  - Paralelas
    - Oblicua
    - Ortográfica
  - Perspectiva
    - 1 punto
    - 2 puntos
    - 3 puntos
    - Proyección ortográfica.

El centro de proyección, COP, se encuentra en el infinito y las líneas de proyección son perpendiculares al plano de proyección (véase figura 3.11).

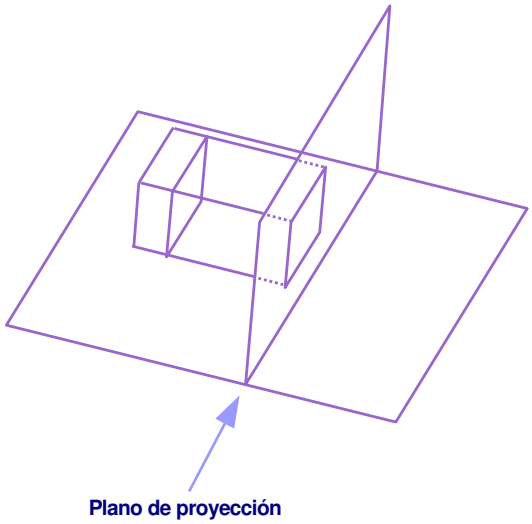

Figura 3.11: Proyección ortográfica.

Este tipo de proyección, no mantiene las dimensiones reales de los objetos según a la distancia que estén de la cámara, por lo tanto el realismo no es total. Se utiliza sobre todo en proyectos de ingeniería con programas CAD/CAM. Los parámetros que se especifican son las dimensiones de la caja (véase figura 3.11 (Xmin,Xmax,Ymin,Ymax,Zmin,Zmax). A los valores max y min se les denomina far o back y near o front.
En openGL queda definida como se ve en el código fuente 3.2.

Código fuente 3.2: Proyección ortográfica.

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glOrtho(xmin, xmax, ymin, ymax, zmin, zmax)
```

- Proyección perspectiva de un punto.

  Las proyecciones perspectiva mantienen las dimensiones reales de los objetos según su distancia a la cámara. El efecto visual es mucho más realista que en

la proyección ortográfica.

En la figura 3.12, se tiene un sólo centro de proyección (COP) y todas las líneas de proyección parten de él. En este caso, las líneas de proyección no son paralelas.

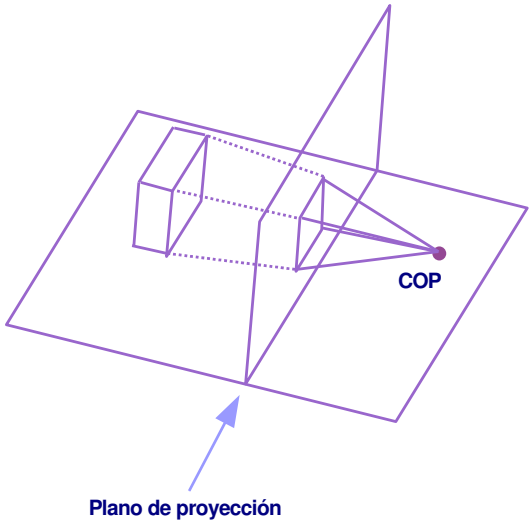

Figura 3.12: Proyección perspectiva.

Código fuente 3.3: Proyección perspectiva.

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
gluPerspective(FOV en grados, Relación de aspecto,
    znear, zfar)
```

Según el código fuente 3.3, los parámetros que se tienen que especificar son:

- ◇ FOV en grados: es el campo visual. Se refiere al ángulo de abertura vertical.
- ◇ Relación de aspecto: cociente entre la anchura y la altura del plano de proyección deseado.
- ◇ Valores near y far del volúmen de visualización: mismo significado que en la ortográfica.

- **Características de la cámara.**

  Los parámetros que se definen son:

  - Posición XYZ: en el mundo 3D.
  - Orientación: una vez situada debe orientarse para dónde mirar.
  - Dirección ”AT”: define hacia dónde mira la cámara, a qué punto concrétamente.

  En openGL se haría como se muestra en el código fuente 3.4. Se debe tener en cuenta que la posición de la cámara no tiene nada que ver con las proyecciones, con lo que la matriz sobre la que trabajamos es la de modelado o transformaciones.

  Código fuente 3.4: Situación de la cámara.

```
glMatrixMode(GL_MODELVIEW)
gluLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX,
      upY, upZ)
```

  En el código fuente 3.4, los parámetros a tener en cuenta son los siguientes (véase figura 3.13):

  - Coordenadas del ”eye”: posición XYZ dónde colocar la cámara dentro del mundo 3D.
  - Coordenadas del ”at”: valor de XYZ del punto al que queremos que mire la cámara.
  - Coordenadas del vector ”up”: es un vector que regula la orientación. Ha de mirar hacia arriba, si el vector que mira hacie adelante es el que va del ”eye” al ”at”.

3.2.3.2. Blender.

- Origen: Inicialmente Blender fue desarrollado como una aplicación propietaria de un estudio de animación holandés llamado NeoGeo; Ton Roosendaal, su principal autor, fundó la empresa Not a Number Technologies (NaN) en junio de 1998 para continuar con su desarrollo y distribución.

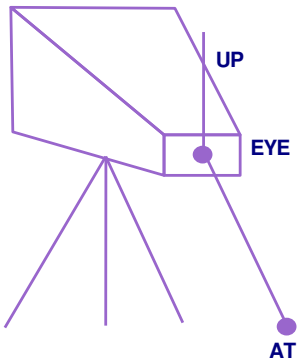

Figura 3.13: Situación de la cámara.

La compañía llegó a bancarrota en el año 2002 y se acordó ofrecer Blender como un producto de libre bajo licencia GNU GPL a cambio de 100.000 €.

El 18 de julio de 2003, Ton Roosendaal creó una fundación sin ánimo de lucro para recoger donaciones; el 7 de septiembre se anunció la recaudación completa y el código fuente se hizo público el 13 de octubre.

- Blender en la industria: Blender ha avanzado mucho en muy poco tiempo, la industria de generación de gráficos. Aunque las superproducciones no lo han utilizado aún para generar imágenes por computador (CGI), hay proyectos actuales que han comenzado a utilizarlo profesionalmente:

  - Spiderman 2 en esta película se utilizó para hacer una previsualización de escenas (Screen-board test).
  - Otros proyectos hechos con la participación de diversos usuarios de Blender, incluido Toon Roosendaal como el cortometraje Elephants Dream, con experimentos de sus capacidades, extendidas gracias a la posibilidad de poder editar su código fuente, aportando esta experiencia a los demás usuarios con innovacio-

nes fundamentales: un sistema de control de gestos (Morph system), un sistema de composición de textura y post producción (Composite), entre otros.

- Características:
  - Multiplataforma, libre, gratuito y con un tamaño de origen realmente pequeño comparado con otros paquetes 3D, dependiendo del sistema operativo en el que se ejecute.
  - Proporciona un elevado número de tipos de objetos 3D como mallas poligonales, superficies NURBS, curvas de Bezier y B-Splines, metasuperficies, fuentes vectoriales (TrueType, PostScript, OpenType).
  - Proporciona superficies de subdivisión de Catmull-Clark.
  - Permite modelado de mallas a nivel de vértice, aristas o caras.
  - Proporciona operaciones booleanas sobre las mallas y herramientas de edición como extrusión, biselado, cortado, subdivisión, suavizado, etc.
  - Junto con las herramientas de animación se incluye cinemática inversa, deformaciones por armadura o cuadrícula, vértices de carga y partículas estáticas y dinámicas.
  - También proporciona edición de audio y sincronización de vídeo.
  - Características interactivas para juegos como detección de colisiones, recreaciones dinámicas y lógica.
  - Posibilidades de renderizado interno versátil e integración externa con el potente trazador de rayos de YafRay.
  - Blender acepta varios formatos gráficos como TGA, JPG, Iris, SGI, TIFF, etc.
  - Tiene un motor de juegos 3D integrado, con un sistema lógico y, para más control, se usa programación en Python.
  - Permite simulaciones dinámicas para softbodies (orgánicos), partículas y fluídos.
  - Se pueden utilizar modificadores apilables, para la aplicación de transformación no destructiva sobre mallas.

- También contiene un sistema de partículas estáticas para simular cabellos o pelajes, al que se han agregado propiedades para lograr texturas realistas.

3.2.3.3. Python.

Python es un lenguaje de programación interpretado e interactivo, capaz de ejecutarse en una gran cantidad de plataformas. Fue creado por Guido Van Rossum en 1990. El nombre proviene de la afición de su creador original por los humoristas británicos Monty Python.

Actualmente Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. La última versión estable del lenguaje es la 2.4.3 (Marzo, 2006).

Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa. También es una calculadora muy útil.

Las ventajas más destacadas de Python son:

- Es un lenguaje de alto nivel.
- Es un lenguaje interpretado.
- Proporciona Orientación a Objetos no estricta, es decir, permite programación estructurada.
- El código resultante es mucho más reducido.
- Extensible y encastrable (lenguaje pegamento).
- Escribir programas en Python requiere mucho menos tiempo que en otros lenguajes.
- No hay diferencias en la fiabilidad de los programas.
- Libre (Licencia GPL)
- Multiplataforma a nivel Software y Hardware.

3.2.3.4. API Blender-Python.

Desde la versión 1.67 de Blender, se permite la utilización del lenguaje de programación Python, para tener acceso a una gran variedad de los objetos del software. Proporciona acceso a gran parte de los datos internos y funciones del programa.

Esto abre muchas posibilidades interesantes, dando la posibilidad de automatizar tareas repetitivas y agregar nuevas funcionalidades a Blender.

Las clases que ofrece el API, más utilizadas en MOSKIS 3D han sido:

- NMesh: esta clase permite la creación de mallas 3D en la ventana 3D de Blender y proporciona funciones para su construcción.
- Mesh: esta clase permite la selección de una malla 3D ya creada en Blender, y su modificación mediante la mayor parte de operadores incluidos en Blender.
- Metaball: esta clase permite la generación de metasuperficies y la obtención y modificación de sus características.

Capítulo 4

Método de trabajo

4.1. Generación de la interfaz de usuario.

4.1.1. Diseño.

MOSKIS 3D es un sistema muy fácil de usar gracias a su sencilla interfaz de usuario.

La interfaz de Blender presenta un área de trabajo personalizable, que puede ser dividida en distintas secciones para tener distintas vistas 3D o herramientas según convenga.

MOSKIS 3D es un script en Python para Blender, por lo tanto, se puede ejecutar en cualquiera de las secciones de la pantalla de Blender, incluso en varias a la vez.

La interfaz de MOSKIS 3D trata de ajustarse en la medida de lo posible a la interfaz de Blender, en cuanto a estilos se refiere, a pesar de que el API sólo dispone de un pequeño conjunto de controles como botones, sliders, etc, para configurarla.

Como se puede ver en la figura 4.1, la pantalla de Blender ha sido dividida en dos partes, en la parte superior se ha cargado MOSKIS 3D y en la parte inferior de la aplicación está la ventana 3D del modelador.

Una vez ejecutado MOSKIS 3D, se puede ver que todas las pantallas siguen la misma estructura de diseño, con 3 partes bien diferenciadas:

- Área de dibujo: Ocupa la mayor parte de la ventana de MOSKIS (sección de Blender en la que se está ejecutando nuestro sistema), y a su vez es redimensionable, según el tamaño de la ventana, el área de dibujo se redimensiona para proporcionar al usuario más espacio de trabajo. Contiene una cuadrícula con los ejes de coordenadas X e Y marcados en rojo y verde respectivamente, para facilitar la situación al usuario.

- Paneles: A la derecha de la pantalla, aparecerán unos paneles que variarán en función del estado de la ejecución del sistema, mostrando unas opciones u otras según corresponda.

  Esta parte de la interfaz se divide a su vez en dos partes:

  - Superior: Aparecerán los controles necesarios para modificar las características del objeto y para interactuar con la ventana 3D de Blender, es decir, añadir los objetos finales a la ventana de Blender para su posterior tratamiento.
  - Inferior: Aparecerán los botones que controlan la vuelta atrás o el paso a la siguiente pantalla y siempre en la parte más inferior aparecerá la opción de abandonar la aplicación.

- Opciones de visualización: En la parte inferior de la aplicación, se puede observar un panel que proporciona dos características principalmente:

  - Acceder a la vista detallada en la ejecución de la aplicación, para ver paso a paso la creación del objeto y poder modificar algunos parámetros. A lo largo de este documento siempre se supondrá una ejecución en modo vista detallada para poder acceder a todas las posibilidades que ofrece el sistema.

- Poder visualizar detalles del objeto durante su creación de diversas formas, como por ejemplo, visualizar sus vértices, visualizar los ejes de coordenadas situados en el centro del objeto, etc.

Como se puede ver en la figura 4.1, está marcada la opción ”Obj Axes” que muestra los ejes de coordenadas y el centro del objeto (el objeto dibujado intenta parecerse al logo de Blender).

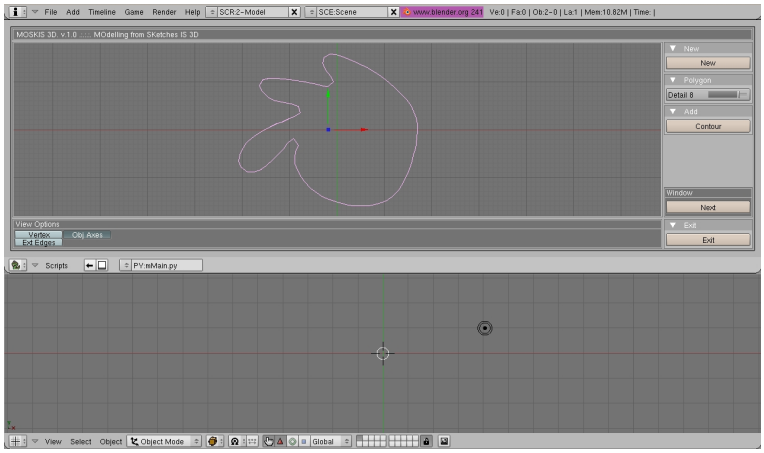

Figura 4.1: Interfaz de MOSKIS 3D.

4.1.2. Construcción.

- Área de dibujo: El API de Blender tiene capacidades suficientes para diseñar interfaces que interactúen con los objetos y funciones de Blender, proporcionando elementos tales como botones, menús, etc. En este caso esto no es suficiente, ya que se requiere un área de dibujo en la propia ventana del sistema, independiente de la ventana 3D que ofrece Blender, y no se dispone de ningún elemento que pueda asemejarse a un área de dibujo con lo cual se tomó la determinación de crearla.

  Para ello se siguieron dos pasos:

  - Dibujado del área: Se utilizó OpenGL, en su versión del API, es decir, *Blender.BGL*. Para su dibujado se utilizaron primitivas tan básicas como el dibujado de

un polígono, para dibujar el rectángulo que enmarca el área, y líneas, para dibujar la cuadrícula y los ejes de coordenadas.

En el código fuente 4.1, se puede ver un ejemplo de las primitivas que se usan en OpenGL (*Blender.BGL*) para el dibujado de un rectángulo con el color y coordenadas que recibe por parámetros.

Código fuente 4.1: Rectángulo en Blender.BGL

```
def drawRectangle(lColor, lCoordinates):
    glColor3f(lColor[0],lColor[1],lColor[2]) #R,G,B
    #lCoordinates = [x1,y1,x2,y2]
    #Esquina superior izquierda y esquina
    #inferior derecha.
    glBegin(GL_POLYGON) #Dibujar un polígono
    glVertex2f(lCoordinates[0], lCoordinates[1])
    glVertex2f(lCoordinates[2], lCoordinates[1])
    glVertex2f(lCoordinates[2], lCoordinates[3])
    glVertex2f(lCoordinates[0], lCoordinates[3])
    glEnd()
```

- Funcionalidad: El área de dibujo tiene dos funcionalidades principalmente:
  - Redimensionable: Se ajusta al espacio que tenga la ventana en la que se carga el sistema y se redimensiona automáticamente, si esta sección varía a lo largo de la ejecución. Para ello el API proporciona la función *GetScreenInfo()*, del paquete *Blender.Window* que devuelve la información de las ventanas de Blender en un diccionario, e identifica a cada ventana mediante un *id*. La función *GetAreaId()*, devuelve el identificador de la ventana en la que se ha cargado el sistema, con lo cual se puede acceder a la información de dicha ventana, como por ejemplo, a las esquinas que la limitan.

    Las esquinas límite se obtendrían como (Xmin,Ymin,Xmax,Ymax), permiten la localización de la ventana de MOSKIS 3D y su tamaño, para poder redimensionar el área de dibujo.

- Interactiva: El usuario dibujará inicialmente la forma 2D que desee modelar, mediante un ratón o tabla digitalizadora, y también podrá, mediante el uso del ratón, escalar la figura (desplazando la rueda central), desplazarla (pulsando el botón derecho) y rotarla (pulsando la rueda central).
  En el código fuente 4.2, se puede ver un ejemplo de las primitivas de *Blender.BGL* que permiten realizar las operaciones de desplazamiento, rotación y escalado.

Código fuente 4.2: Desplazamiento,rotación y escalado.

```
#Traslada una distancia dx en el eje x, etc.
glTranslatef(dx, dy, dz)
#Rota un ángulo rx en el eje x, etc.
glRotatef(rx,1,0,0)
glRotatef(ry,0,1,0)
glRotatef(rz,0,0,1)
#Escala sx en el eje x, etc.
glScaled(sx, sy, sz)
```

- Paneles: Los paneles situados a la derecha en todas las pantallas del sistema están siempre compuestos de dos elementos:
  - Contenedor: Los contenedores son dibujados mediante primitivas de *Blender.BGL* para el dibujado de polígonos, además de la función *Text(string,fontsize)* del módulo *Blender.Draw* que dibuja el texto de los contenedores.
  - Controles: Los controles que aparecen delimitados por los contenedores comentados anteriormente, son proporcionados por el API de Blender-Python y se encuentran en el módulo *Blender.Draw*. Los tipos de controles utilizados han sido:
    - *Button*: botón normal de Blender.
    - *Slider*: barra deslizante para selección de valores.
    - *Toggle*: botón booleano.

- Opciones de visualización: El panel de opciones de visualización situado en la parte inferior de la ventana, está compuesto, al igual que en el caso de los paneles anteriormente descritos, de un panel contenedor y de controles, que en este caso siempre son de tipo *Toggle*, para activar y desactivar opciones de visualización a lo largo de la ejecución de la aplicación MOSKIS 3D.

4.2. Creación de un objeto nuevo en MOSKIS 3D.

4.2.1. Obtención del polígono.

4.2.1.1. Creación del polígono.

- El usuario, con ayuda del ratón, realiza una silueta mediante un trazo libre sobre el área de dibujo. El trazo debe ser cerrado y no producir intersecciones consigo mismo, ya que en este caso el sistema fallaría.

- A partir de este trazo, el sistema genera un polígono cerrado mediante la conexión del punto de comienzo y del punto final del trazo (véase figura 4.2).

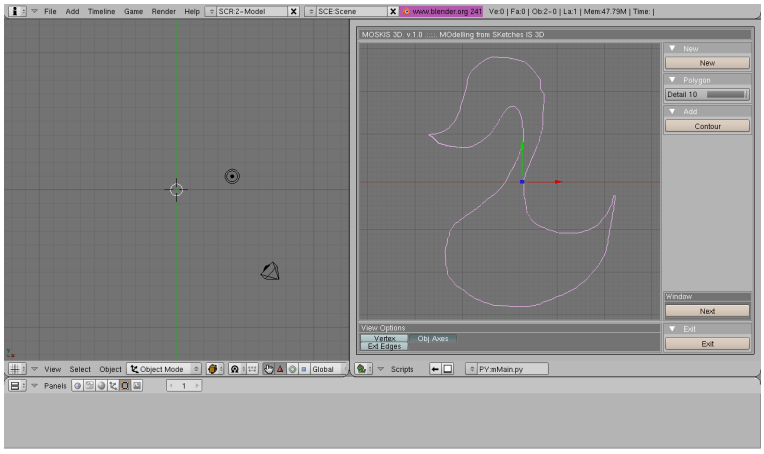

Figura 4.2: Contorno creado por el usuario.

El sistema genera el polígono con un nivel de detalle determinado, que no es el máximo (el máximo sería con todos los puntos que detecta de las coordenadas del usuario mien-

tras realiza el contorno), pero ofrece la posibilidad al usuario de modificar este nivel de detalle mediante una barra de deslizamiento (véase figura 4.3).

El nivel de detalle indica un umbral de distancia entre un vértice y otro, si se sobrepasa esa distancia se formará una nueva arista entre esos dos vértices, si no se sobrepasa, ese vértice será ignorado y se procederá a evaluar el siguiente. Evita la creación de vértices innecesarios.

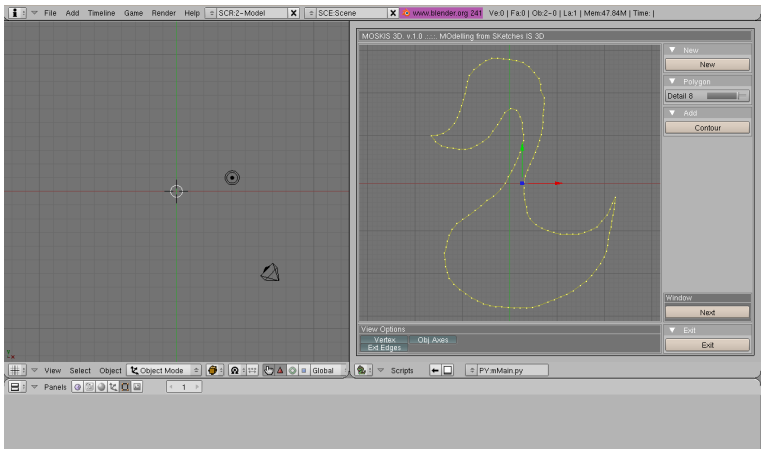

Figura 4.3: Creación del polígono.

En la figura 4.3, se puede observar que está seleccionado por defecto un nivel de detalle de 8 unidades, en un intervalo permitido de 1 a 10 unidades, siendo 10, el máximo nivel de detalle. Siempre que se aumenta el detalle, se tendrán más vértices y aumentará el coste computacional en fases posteriores, pero también se tendrá más exactitud en el objeto obtenido. El nivel de detalle que aparece por defecto es un equilibrio entre detalle y coste.

Se puede observar en las figuras 4.3 y 4.4, resaltados en amarillo el número de vértices de la silueta. En la figura 4.4, se muestra la diferencia en número de vértices entre los niveles de detalle.

En algunas ocasiones, disminuir el número de vértices aumenta el rendimiento y la disminuición de detalle no es relevante.

El polígono generado servirá como silueta del objeto 3D que se quiere obtener.

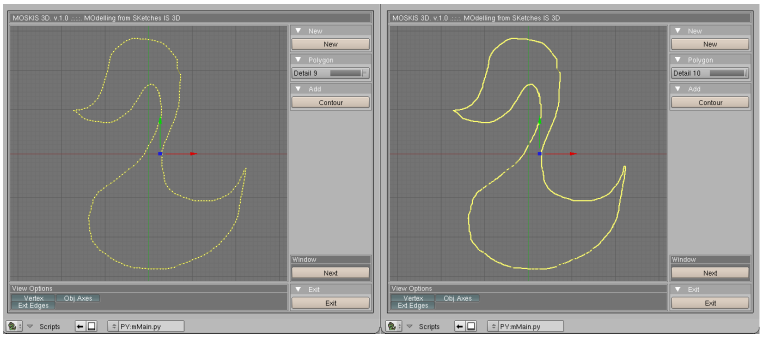

Figura 4.4: Polígonos con niveles de detalle 9 y 10.

4.2.1.2. Añadir el polígono a la ventana 3D de Blender.

El sistema permite, una vez creado el polígono añadir éste a la ventana 3D de Blender, permitiendo así el dibujado de contornos y siluetas 2D libres desde MOSKIS 3D (véase código fuente 4.3 y figura 4.5).

Código fuente 4.3: Añadir contorno

```
#Este método añade el contorno pintado por el usuario a la
#ventana 3D de blender.
def addContour(oPolygon):
    lEdgeList = oPolygon.getEdges()
    fReduct = 20.0
    oMesh = Blender.NMesh.New() #Se crea la malla

    #Se añaden los vértices, sólo se añaden una
    #vez, es decir, sólo se añade el vértice origen de cada
    #arista, porque si se añade el vértice destino también,
    #será el mismo que el vértice origen de la arista siguiente
    #con lo cual los dos vértices estarán duplicados.
    for i in lEdgeList:
        oOriginVertex = i.getVO()
        oBlenderOriginVertex =
```

```
            Blender.NMesh.Vert(float(oOriginVertex.getX()/
                fReduct),
                float(oOriginVertex.getY()/fReduct),
                float(oOriginVertex.getZ()/fReduct))
        oMesh.verts.append(oBlenderOriginVertex)

    #Se añaden las aristas a la malla
    iEdge = 0
    while iEdge < len(oMesh.verts)-1:
        oMesh.addEdge(oMesh.verts[iEdge],oMesh.verts[iEdge+1])
        iEdge+=1
    oMesh.addEdge(oMesh.verts[0],
        oMesh.verts[len(oMesh.verts)-1])
    #Se añade la malla a la ventana 3D de Blender
    PutRaw(oMesh, "Contour Object")
```

E objeto *NMesh* permite la creación de mallas, en este caso mediante vértices y aristas tenemos construido nuestro polígono en la ventana 3D de Blender.

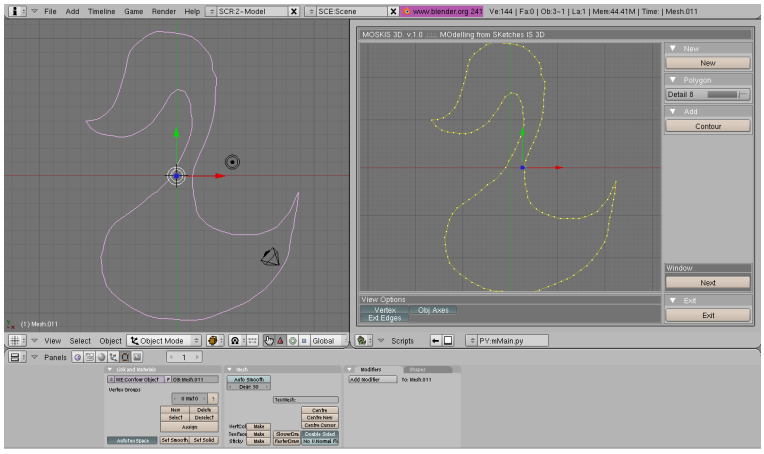

Figura 4.5: Polígono añadido a la ventana 3D de Blender.

Se hace una conversión desde las coordenadas del sistema MOSKIS 3D, al de Blender, que se realiza dividiendo cada coordenada de MOSKIS 3D entre *fReduct = 20.0*. Esta conversión es realizada para igualar el tamaño ya que las coordenadas utilizadas en el área de dibujo del script, equivalen aproximadamente a 1/fReduct en la pantalla 3D de Blender, al iniciar la aplicación.

4.2.1.3. Estructuras de datos.

Para esta parte de la ejecución se tienen los objetos mostrados en la figura 4.6.

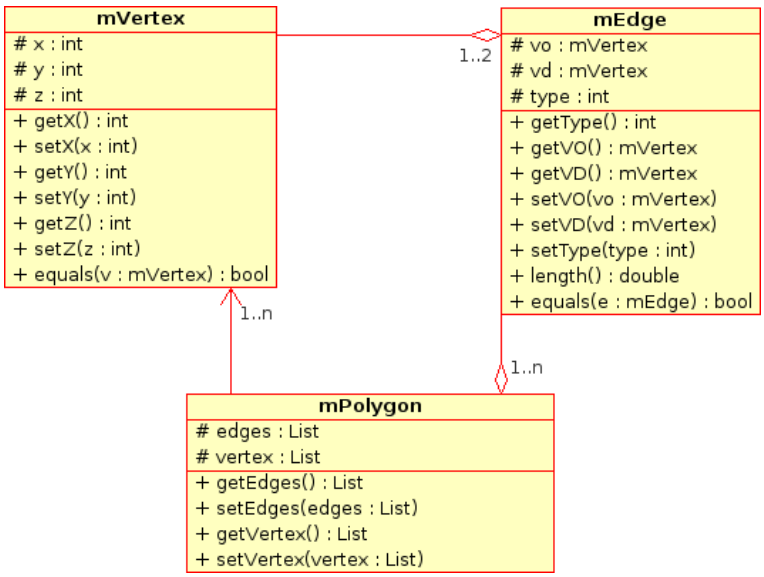

Figura 4.6: Clases utilizadas para la implementación del polígono.

- mVertex: un vértice es un punto definido por las coordenadas en los ejes X, Y y Z.
- mEdge: una arista es un segmento delimitado por dos vértices, por el vértice origen y el vértice destino, y caracterizada por un tipo. Una arista puede ser externa (arista del polígono) o interna (arista generada posteriormente en la triangularización).
- mPolygon: un polígono es un conjunto de aristas. En el diagrama de clases de la figura 4.6, se ve que la clase polígono tiene además, una lista de vértices. Esta decisión provoca información redundante, ya que las aristas ya contienen los vértices que las forman, pero de esta forma se agilizan los cálculos.

Como se estudiaba en el apartado de Técnicas: Estructuras de datos, la estructura aplicada en MOSKIS 3D ha sido una estructura basada en aristas.

No se ha aplicado la estructura Winged-Edge porque es muy costosa su construcción, y al ser necesarias modificaciones muy rápidas en los datos de la malla, no es conveniente su uso en este caso.

4.2.2. Generación del plano.

4.2.2.1. Triangulación básica.

Para la generación del plano a partir del polígono, se lleva a cabo inicialmente un algoritmo de triangulación básica, que consiste en la generación de triángulos para cubrir la superficie delimitada por el polígono (véase algoritmo 1).

Algoritmo 1 Triangulación básica.

```
while número de aristas del plano > 3 do
    Identificar vértices convexos
    for cada vértice convexo do
        Evaluar posible arista interna
        if arista válida then
            Crear triángulo
            Añadir triángulo al plano
            Reducir polígono
        else
            Dividir el polígono en dos.
            Aplicar triangulación a cada subpolígono
        end if
    end for
end while
```

Inicialmente se tiene una lista de triángulos vacía (un plano es una lista de triángulos).

El algoritmo se ejecuta hasta que el polígono quede reducido a un triángulo, que será el último triángulo que se añada a la lista y quedará concluida la triangulación básica.

En cada iteración del algoritmo, se realizan las tareas que se describen a continuación:

- Identificar vértices convexos.

  Se procede a la identificación de cuatro vértices convexos en el polígono, que serán los situados en los extremos de éste.

  Es fácil demostrar que estos vértices son convexos. Por ejemplo, el vértice situado más a la derecha es convexo, porque si no lo fuese, habría otro vértice situado más a la derecha que él. De esta forma queda asegurado, al igual que en este caso con los demás, que los vértices de los extremos son convexos.

  Para la identificación de estos vértices se hace un análisis de coordenadas de cada uno y se seleccionan cuatro:

  - El vértice con coordenada (X, Y_{min}, Z), será el vértice inferior.
  - El vértice con coordenada (X, Y_{max}, Z), será el vértice superior.
  - El vértice con coordenada (X_{min}, Y, Z), será el vértice situado más a la izquierda.
  - El vértice con coordenada (X_{max}, Y, Z), será el vértice situado más a la derecha.

- Evaluar posible arista interna.

  Para cada vértice convexo, se evalúa si sería posible trazar una arista que formase un triángulo con las dos aristas que comparte el vértice convexo, es decir, si sería posible formar un triángulo con este vértice y los dos vértices que le rodean.

  Para que el trazado de la arista interna sea válido, se tiene que comprobar que esta arista interna no cortará a ninguna otra arista del polígono, es decir, no existirá ningún vértice del polígono dentro del área del triángulo a construir, ya que en este caso la triangulación sería errónea (véase figura 4.8).

  El área de un triángulo se calcula como resultado del determinante que se muestra en la ecuación 4.1.

$$A = \frac{1}{2} \begin{vmatrix} 1 & 1 & 1 \\ X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \end{vmatrix} \tag{4.1}$$

La función que calcula el área de un triángulo se ve en el código fuente 4.4.

Código fuente 4.4: Cálculo del área de un triángulo.

```
#Área del triángulo de vértices oV1,oV2,oV3
def triangleArea(oV1,oV2,oV3):
    iX1 = oV1.getX()
    iX2 = oV2.getX()
    iX3 = oV3.getX()
    iY1 = oV1.getY()
    iY2 = oV2.getY()
    iY3 = oV3.getY()
    fArea = float(abs((iX2*iY3+iX3*iY1+iX1*iY2-
            iY1*iX2-iX1*iY3-iY2*iX3)/2))
    return fArea
```

Para calcular si existen vértices dentro del triángulo que se quiere formar se sigue el algoritmo 2.

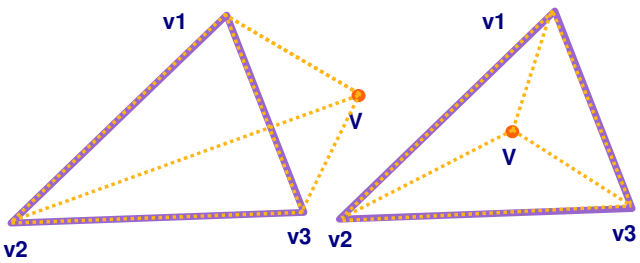

Figura 4.7: Subtriángulos formados.

Algoritmo 2 Vértices dentro del triángulo.

for all vértices del polígono **do**

Calcular el área del triángulo.

Calcular el área de los subtriángulos formados por el vértice a evaluar con cada dos vértices del triángulo (véase figura 4.7).

if Suma de áreas de los subtriángulos $\leq$ Área del triángulo **then**

El vértice está dentro del triángulo.

else

El vértice no está dentro.

end if

end for

- Dividir el polígono en dos.

  En el caso en el que existan vértices del polígono dentro del triángulo que se pretende formar, se calcula el vértice interno al triángulo que esté más cerca del vértice convexo, y se traza una arista interna desde el vértice convexo hasta el vértice interno. El polígono queda dividido en dos (ver figura 4.8).

  Una vez dividido se aplica a cada subpolígono el algoritmo de triangulación.

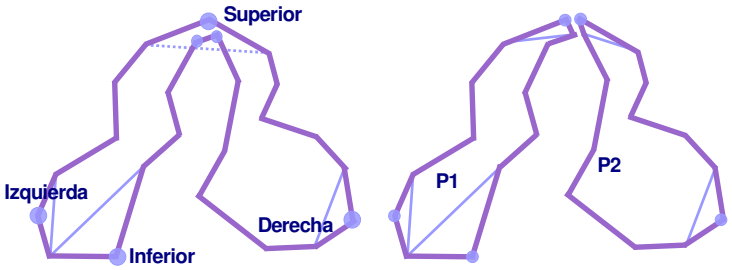

Figura 4.8: División del polígono.

- Crear triángulo y reducir polígono.

  Si la arista interna es válida, se añade el triángulo a la lista de triángulos que van configurando el plano.

Se elimina el vértice convexo con el que hemos formado el triángulo y se genera un nuevo polígono a partir de los vértices de los extremos de las aristas que comparte. Como se ve en la figura 4.9, el triángulo formado es el que se ve en color amarillo y el polígono resultante de la reducción se ve en color blanco.

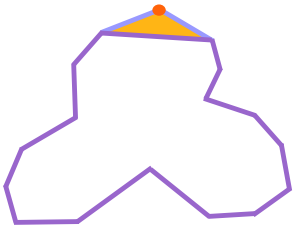

Figura 4.9: Creación del triángulo y reducción del polígono.

En la figura 4.10 se puede ver el resultado de la triangulación básica para nuestro ejemplo con MOSKIS 3D.

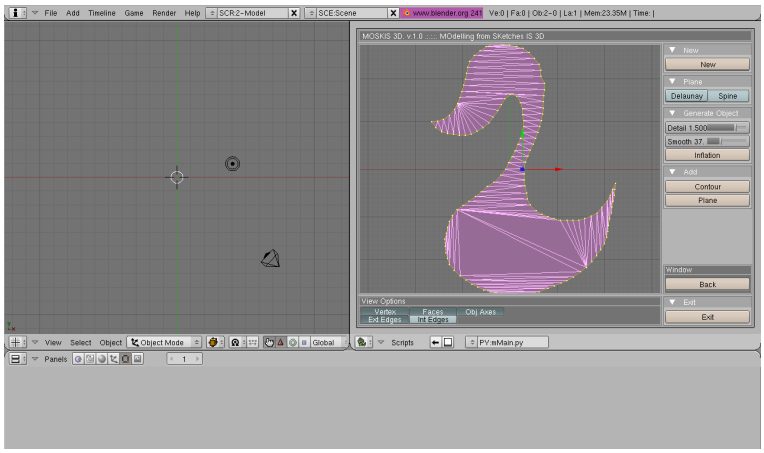

Figura 4.10: Triangulación básica.

4.2.2.2. Optimización basada en Delaunay.

Para asegurar que la superficie del polígono quede repartida mediante triángulos, de la forma más homogénea posible, se realiza una optimización sobre la triangulación básica realizada (véase figura 4.11).

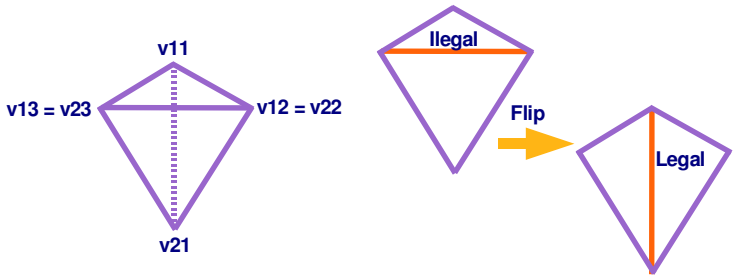

Figura 4.11: Evaluación de arista y flip de arista ilegal.

Dados dos triángulos t_1 y t_2 que comparten una arista.

Sean (v_{11}, v_{12}, v_{13}) y (v_{21}, v_{22}, v_{23}) los vértices de t_1 y t_2 respectivamente (véase figura 4.11).

Sean $(\alpha_1, \beta_1, \gamma_1)$ y $(\alpha_2, \beta_2, \gamma_2)$, los ángulos de t_1 y t_2 respectivamente.

Los triángulos resultantes de girar la arista interna, por la opuesta en el cuadrilátero formado por los dos triángulos que comparten dicha arista, serían t'_1 y t'_2, con vértices $(v'_{11}, v'_{12}, v'_{13})$ y $(v'_{21}, v'_{22}, v'_{23})$ que en este caso se corresponderían con (v_{11}, v_{13}, v_{21}) y (v_{21}, v_{22}, v_{11}) (ver figura 4.11), y con ángulos $(\alpha'_1, \beta'_1, \gamma'_1)$ y $(\alpha'_2, \beta'_2, \gamma'_2)$.

Una arista es legal, si se cumple la ecuación 4.2, en caso contrario es ilegal.

$$(min(\alpha_1, \beta_1, \gamma_1) \geq min(\alpha'_1, \beta'_1, \gamma'_1)) \bigwedge (min(\alpha_2, \beta_2, \gamma_2) \geq min(\alpha'_2, \beta'_2, \gamma'_2)) \tag{4.2}$$

Si una arista es ilegal, se produce un giro, también llamado flip de la arista interna, como se ve en la figura 4.11, sustituyendo los triángulos t_1 y t_2 del plano, por t'_1 y t'_2.

Los triángulos generados a partir de la triangulación, se pueden catalogar como:

- Triángulos T: compuestos de dos aristas externas.
- Triángulos S: compuestos de una arista externa.
- Triángulos J: compuestos de tres aristas internas.

En la figura 4.12 se puede ver el resultado de la optimización en nuestro ejemplo con MOSKIS 3D.

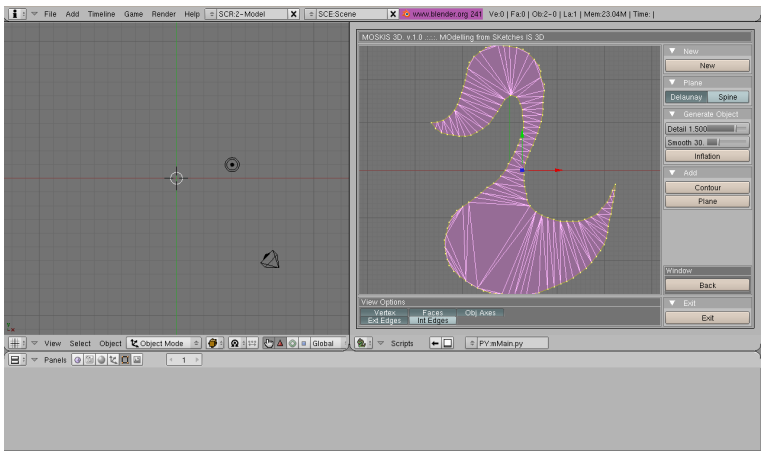

Figura 4.12: Optimización por Delaunay.

4.2.2.3. Añadir el plano a la ventana 3D de Blender.

El sistema permite, una vez creado el plano, añadir éste a la ventana 3D de Blender, permitiendo así obtener planos con siluetas de formas libres automáticamente (véase figura 4.13) y pudiendo darles desde Blender un tratamiento posterior (véase figura 4.14).

En este caso, al igual que en el caso en el que se añadía un polígono, se crea un objeto *NMesh*, pero a diferencia del anterior, no se añaden aristas, sino los vértices de cada triángulo (sólo una vez, para no crear vértices repetidos). Posteriormente se crea, una cara con los vértices de cada triángulo, que se añade posteriormente a la malla.

En la porción de código 4.5, se puede ver como se agrega una cara a la malla.

Código fuente 4.5: Creación de una cara.

```
#Para cada triángulo, se crea una cara, con los vértices
#añadidos de la malla, ya que se tienen sus posiciones en
#trianglesMesh, y una vez añadidos los tres vértices a la
#cara, se añade la cara a la malla
for i in lTrianglesMesh:
    oFace = Face()
    for j in i:
        oFace.v.append(oMesh.verts[j])
```

```
oMesh.faces.append(oFace)
```

En la figura 4.13 se puede ver el resultado de añadir un plano a la ventana 3D de Blender y en la figura 4.14 se ve el resultado después de aplicar la operación de extrusión.

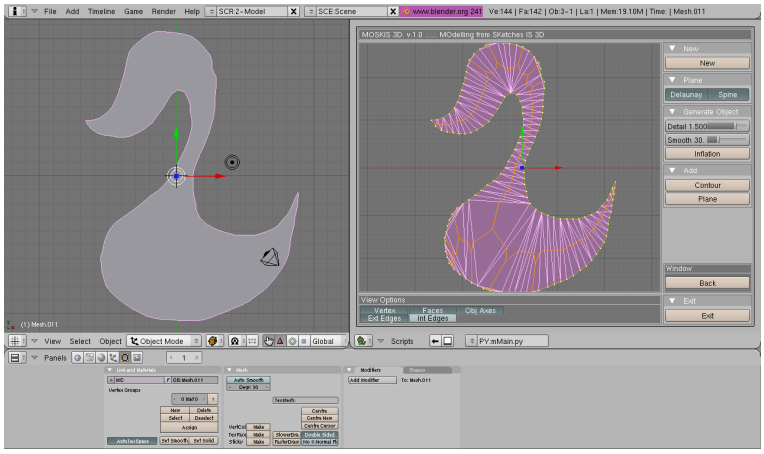

Figura 4.13: Plano añadido a la ventana 3D.

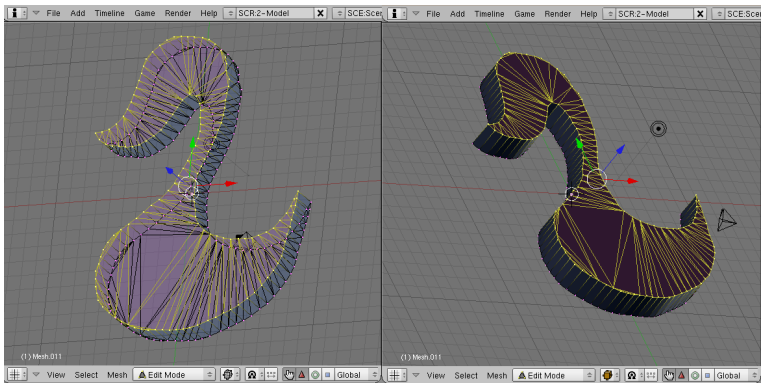

Figura 4.14: Plano añadido a la ventana 3D y extruido posteriormente.

4.2.2.4. Estructuras de datos.

En esta parte de la ejecución participan los objetos que se muestran en la figura 4.15

- mTriangle: un triángulo es un polígono de tres lados que a su vez tiene un atributo tipo, que indica si el polígono es de tipo T, S o J.

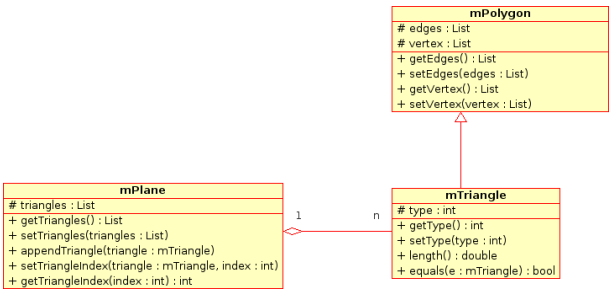

Figura 4.15: Clases utilizadas para la implementación del plano.

- mPlane: un plano es una lista de triángulos.

4.2.3. Cálculo del perfil de alturas.

4.2.3.1. Detección de la espina dorsal.

Una vez generado el plano se procede a detectar la espina dorsal de éste. Para ello se siguen los siguientes pasos:

1. Se detectan los puntos de conexión de la espina dorsal, que serán los puntos principales del eje mediano de la forma inicial.

2. La unión mediante aristas de estos puntos de conexión me dan la espina dorsal del plano que representa la silueta del objeto.

El algoritmo 3 describe este proceso.

El punto central de un triángulo se calcula como el punto de intersección de dos de sus bisectrices (véase figura 4.16 y algoritmo 4).

Una bisectriz de un triángulo es el segmento que une uno de sus vértices con el punto medio de la arista opuesta a este vértice. Para calcular el punto de intersección de dos bisectrices del triángulo, es decir, calcular el centro del triángulo véase el algoritmo 4.

En la figura 4.17 se puede ver la detección de la espina dorsal para cada tipo de triángulo.

Algoritmo 3 Algoritmo de generación de la espina dorsal.

for all triángulo del plano **do**

if triángulo de tipo S **then**

Se detectan 2 puntos de conexión y 1 arista de la espina dorsal.

$p_1 \longleftarrow$ centro de la arista interna 1

$p_2 \longleftarrow$ centro de la arista interna 2

$a_1 \longleftarrow$ arista formada por p_1 y p_2

else if triángulo de tipo J **then**

Se detectan 4 puntos de conexión y 3 aristas de la espina dorsal.

$p_1 \longleftarrow$ centro de la arista interna 1

$p_2 \longleftarrow$ centro de la arista interna 2

$p_3 \longleftarrow$ centro de la arista interna 3

$p_4 \longleftarrow$ centro del triángulo

$a_1 \longleftarrow$ arista formada por p_1 y p_4

$a_2 \longleftarrow$ arista formada por p_2 y p_4

$a_3 \longleftarrow$ arista formada por p_3 y p_4

end if

end for

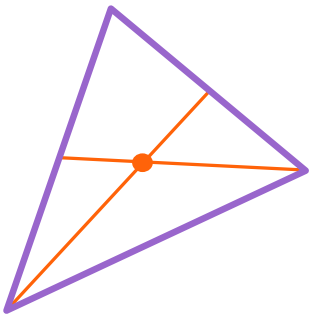

Figura 4.16: Centro de un triángulo.

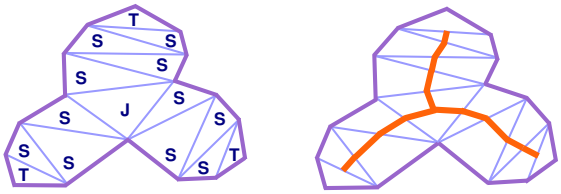

Figura 4.17: Detección de la espina.

4.2.3.2. Elevación de la espina dorsal.

Una vez que todos los puntos de conexión han sido unidos mediante aristas, formando la espina dorsal del plano, se procede a la elevación de la espina, es decir, a darle coordenada Z a cada uno de los vértices que la forman.

El problema es computar la distancia desde la espina dorsal al límite del polígono. Como el construir secciones representativas exactas de la forma, para cada vértice de la espina dorsal, tiene un coste muy elevado, su cálculo se realiza como puede verse en el algoritmo 4.2.3.2.

Como se ve en el algoritmo, para cada punto de conexión p se da un tratamiento dependiendo de si es el centro de una arista interna o el centro de un triángulo de tipo J.

En la figura 4.18, se puede ver el resultado de la detección y elevación de la espina sobre el objeto en el sistema en la triangulación básica (figura de la izquierda) y en la triangulación de Delaunay (figura de la derecha).

El resultado de la elevación de la espina dorsal en la triangulación de Delaunay es mucho más realista como eje central del objeto que en la triangulación básica. Es recomendable la

Algoritmo 4 Punto central de un triángulo

1: Se obtienen las coordenadas de los extremos de las bisectrices (un vértice del triángulo y el punto medio de la arista opuesta a éste).

2: Se halla la ecuación implícita de cada bisectriz de la forma:

$\frac{Y-Y_1}{X-X_1} = \frac{Y_2-Y_1}{X_2-X_1}$

Poniéndolo en función de X e Y de la forma $AX + BY + C = 0$ quedaría:

$A = \frac{Y_2-Y_1}{X_2-X_1}$

$B = -1$

$C = Y_1 - (\frac{Y_2-Y_1}{X_2-X_1})X_1$

3: Se resuelve el sistema de ecuaciones formado por las dos ecuaciones correspondientes a dos de las bisectrices del triángulo.

$A_1X + B_1Y + C_1 = 0$

$A_2X + B_2Y + C_2 = 0$

4: El punto (X, Y) formado por la solución del sistema de ecuaciones, será el punto central del triángulo.

Algoritmo 5 Algoritmo de elevación de la espina dorsal.

for all p: puntos de conexión **do**
  if p asociado a arista interna **then**
    Distancias entre p y todos los vértices del plano directamente conectados.
    distancia de elevacion de p ⟵ promedio de las distancias.
  else if p es punto medio de un triángulo J **then**
    Se detectan 4 puntos de conexión y 3 aristas de la espina dorsal.
    Distancias de elevación de los puntos de conexión directamente conectados a p.
    distancia de elevación de p ⟵ promedio de las distancias de elevación.
  end if
end for

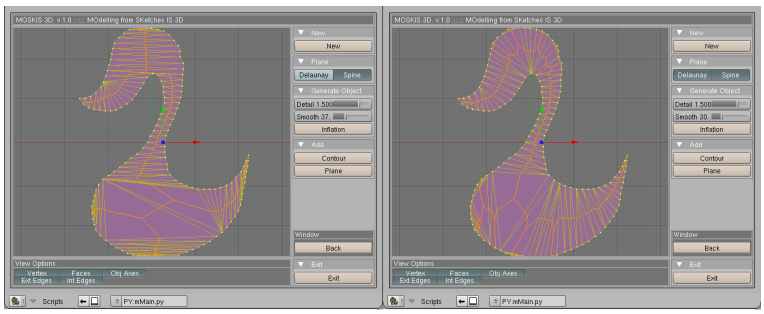

Figura 4.18: Detección y elevación de la espina dorsal en la triangulación Básica y Delaunay.

optimización de Delaunay ya que redistribuye los triángulos en la malla de forma homogénea.

4.2.3.3. Estructuras de datos.

En esta parte de la implementación participan los objetos de la figura 4.19.

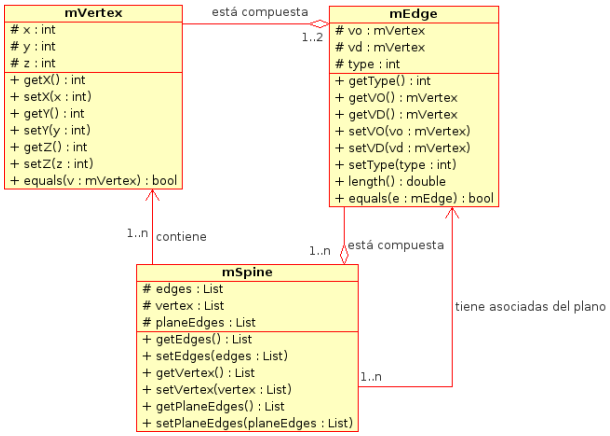

Figura 4.19: Clases utilizadas para la implementación de la espina.

- mSpine: una espina está compuesta de tres elementos:
  - edges: es la lista de aristas de la espina, es decir, la lista de aristas que unen los puntos de conexión de la espina dorsal. La estructura se almacenará como una lista de listas, es decir, en cada posición contendrá una lista de aristas que se corresponderá con las aristas de la espina para cada triángulo.

- vertex: es la lista de vértices de la espina, es decir, la lista de puntos de conexión. Esta información es redundante, ya que es contenida en el atributo edges, pero es una forma de agilizar cálculos.

  Esta lista, es al igual que en el caso anterior, una lista de listas, ya que en cada posición, se almacena una lista con los puntos de conexión de cada triángulo. Por ejemplo, si es un triángulo S, vertex almacenará en una posición una lista con dos puntos de conexión y si es un triángulo J, almacenará una lista con cuatro puntos de conexión.

- planeEdges: es la lista de aristas del plano que corta cada vértice de la espina dorsal. Al igual que en los casos anteriores, es una lista de listas, y consigue en cada posición, la lista de aristas relacionadas con los vértices de la misma posición de la lista vertex.

Un ejemplo aclarativo de este tipo de almacenamiento se puede ver en la figura 4.20.

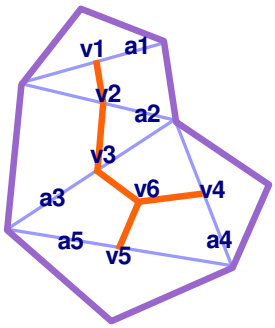

Figura 4.20: Ejemplo de lo que almacena un objeto mSpine.

Según la figura 4.20, lo que se almacena en cada lista es lo siguiente:

vertex = [[v1,v2],[v2,v3],[v3,v4,v5,v6]]

edges = [[edge(v1,v2)],[edge(v2,v3)],[edge(v3,v6),edge(v4,v6),edge(v5,v6)]]

planeEdges = [[a1,a2],[a2,a3],[a3,a4,a5]]

4.2.4. Creación del objeto final.

Las dos aproximaciones que se detallan a continuación, explican como a partir del plano y la espina dorsal de éste, se obtiene la malla 3D asociada.

En ambos casos, la obtención del volumen del objeto se realiza mediante la utilización de superficies implícitas, concretamente metaelementos, que pueden ser definidos como objetos que cambian de figura dependiendo de lo cerca que estén de otro metaelemento.

La magnitud de la fuerza de atracción de un metaelemento, normalmente está definida en función de su volumen, pero su área de influencia, puede ser determinada independientemente de su tamaño.

La elección de este mecanismo para la generación del objeto es debida a que la creación de metasuperficies, está indicada para el modelado de personajes y objetos de la naturaleza, es decir, modelado orgánico, ya que permite generar formas suaves muy parecidas a las de los seres humanos.

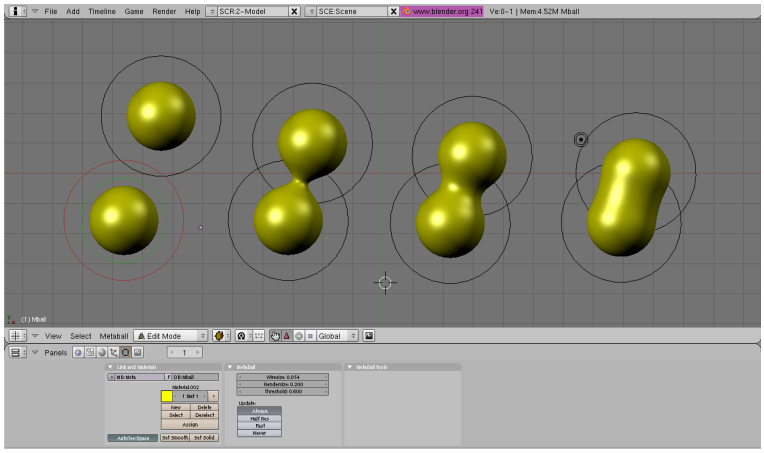

Figura 4.21: Metabolas en Blender.

En la figura 4.21, se ve un ejemplo de como varía la forma de los metaelementos, en este caso metabolas, con la distancia a la que se encuentran unos de otros.

4.2.4.1. Primera aproximación.

- **Algoritmo.**

La obtención de volumen se realiza, añadiendo para cáda vértice o punto de conexión de la espina dorsal, un metaelemento, concrétamente una metaelipse (metaelemento con forma de elipse) con una dimensión y orientación determinada.

Las dimensiones y el ángulo de rotación de la metaelipse van a ser las siguientes:

- Tamaño de la metaelipse (coordenada X): Véase algoritmo 6

Algoritmo 6 Tamaño de la metaelipse en el eje X

if el punto de conexión es el punto medio de una arista interna **then**

$ancho \longleftarrow$ longitud de la arista interna.

else if punto medio de un triángulo de tipo J **then**

$ancho \longleftarrow$ longitud media de las bisectrices del triángulo J.

end if

- Tamaño de la metaelipse (coordenada Y):

  Se corresponde con la distancia media desde ese punto de conexión al resto de puntos de conexión directamente conectados al mismo.

- Tamaño de la metaelipse (coordenada Z):

  Se corresponde con el doble de la altura del punto de conexión de la espina dorsal asociado a la metaelipse. Es el doble, ya que la figura resultante va a ser simétrica y se tiene la altura sólo para la coordenada positiva, pero se necesita también la coordenada negativa, lo cual se soluciona dando el doble de la altura a la metaelipse y situándola en el punto de conexión.

- Rotación:

  Las aristas internas del plano, no tienen todas la misma dirección, con lo cual, la metaelipse debe tener la misma dirección que la arista asociada, de tal forma que se calcula el ángulo de rotación que debe tener esta metaelipse como se detalla en el algoritmo 7.

De esta forma se tiene cada metaelipse situada en el centro de cada punto de conexión y con las dimensiones y ángulo de rotación adecuado (ver figura 4.23).

Algoritmo 7 Rotación de una metaelipse.

Se establece un centro de coordenadas imaginario en el centro de la arista, es decir, en el punto de conexión asociado a esa metabola (ver figura 4.22).

Sea vo el vértice origen de la arista y vd el vértice destino.

if $vo.x < vd.x$ **then**

Construímos el triángulo formado por los lados a, b y c (ver figura 4.22, casos 1 y 2).

Por el *Teorema del Coseno*, se tiene:

$a^2 = b^2 + c^2 - 2bc * cos(A)$

$b^2 = a^2 + c^2 - 2ac * cos(B)$

$c^2 = a^2 + b^2 - 2ab * cos(C)$

Se calcula el ángulo A despejando de la primera fórmula del teorema.

if $vo.y < vd.y$ **then**

Se establece el ángulo A positivo (ver figura 4.22, caso 1).

else

Se establece el ángulo A negativo (ver figura 4.22, caso 2).

end if

else if $vo.x > vd.x$ **then**

Construímos el triángulo formado por los lados a, b y c (ver figura 4.22, casos 3 y 4).

Se calcula el ángulo A por el *Teorema del Coseno*.

if $vo.y > vd.y$ **then**

Se establece el ángulo A positivo (ver figura 4.22, caso 3).

else

Se establece el ángulo A negativo (ver figura 4.22, caso 4).

end if

end if

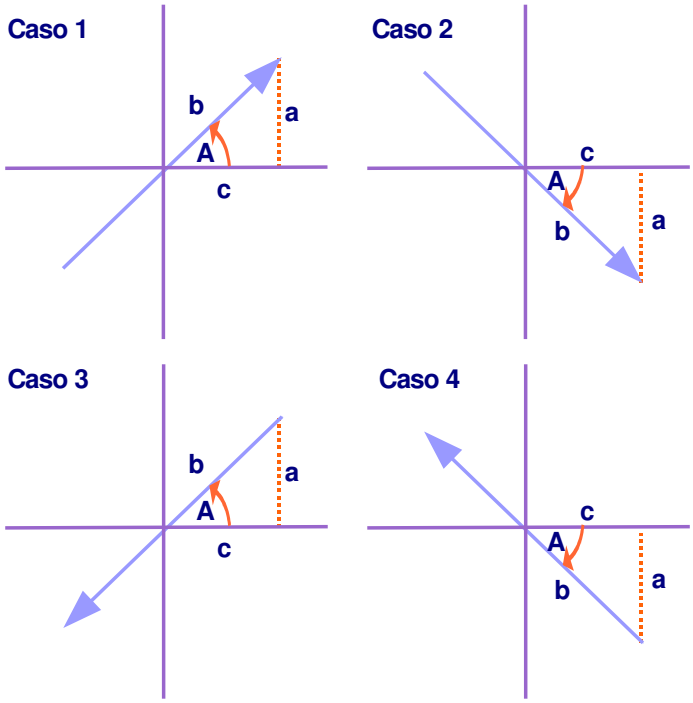

Figura 4.22: Cuatro casos para hallar la rotación de la metaelipse.

- **Problemas.**

  En la generación del objeto 3D según esta aproximación, surgen dos problemas principales:

  - Dados dos puntos de conexión de la espina dorsal, muy lejanos entre sí, como el tamaño en la coordenada y de la metaelipse se establece como un promedio, puede que haya otros puntos conectados a éstos vértices que se encuentren más cerca, y el promedio ser una distancia pequeña.

    Esto tiene como efecto, la aparición de huecos en el objeto 3D por falta de metaelementos en esa zona (ver figura 4.24 y 4.25).

  - En algunas figuras creadas por el usuario (ver figura 4.24 y 4.25), se identifican saltos bruscos entre las metasuperficies, cuando en realidad, no se debería identificar visualmente cada metaelemento, sino que se debería obtener una malla suavizada. Estos saltos se reducen cuando se aumenta el nivel de detalle, porque a

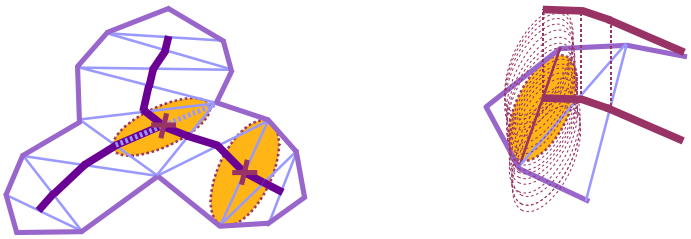

Figura 4.23: dimensiones y rotación de las metaelipses.

mayor nivel de detalle, más vértices se obtienen, por lo tanto son generados más triángulos, y más puntos de conexión en la construcción de la espina.

Cuanto más nivel de detalle, las progresiones en la altura (coordenada z) de los metaelementos, cambia de uno a otro de forma más progresiva.

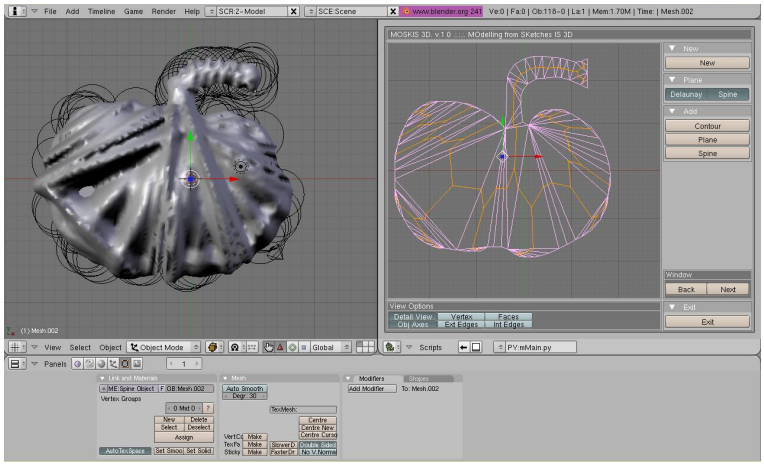

Figura 4.24: Resultado 1 de la primera aproximación.

4.2.4.2. Versión final.

La solución a los problemas anteriores consiste en la creación de metaelipses con dimensiones fijas y reducidas en las coordenadas x e y, y el cálculo de un promedio para la coordenada z.

En vez de crear una metaelipse por cada punto de conexión de la espina dorsal, se crearán las necesarias para cubrir toda la superficie del plano.

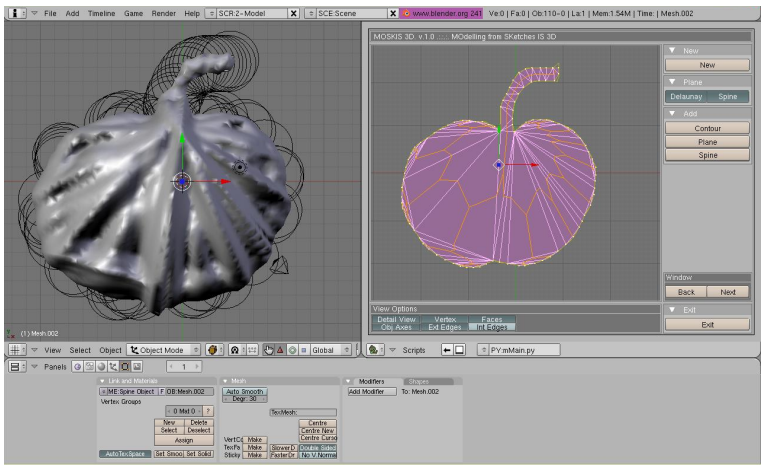

Figura 4.25: Resultado 2 de la primera aproximación.

- **Generación de las metaelipses.**

  Cada arista del plano, asociada a un vértice de la espina dorsal (punto de conexión), es rellenada con metaelipses con unas dimensiones en los ejes X e Y fijas. Se calcula la longitud de la arista, y el número de metaelipses que caben en ella, y se distribuyen metaelipses a lo largo de toda la arista. Si sobra alguna porción de arista, al ser de menor longitud que el tamaño establecido para las metaelipses, se añade una metaelipse del tamaño necesario.

  MOSKIS 3D establece un tamaño para estas metaelipses por defecto, mediante el que se obtiene un equilibrio entre coste computacional y nivel de detalle de la figura generada.

  Se ofrece la posibilidad al usuario de modificar el tamaño de estas metaelipses, mediante una barra de deslizamiento situada en el panel *Generate Object*, que va a indicar el nivel de detalle, a más nivel de detalle seleccionado, menor tamaño de la metaelipse.

  Hay que tener en cuenta que la disminución este tamaño, aumentará el número de metaelipses creadas y este hecho aumentará de forma elevada el coste computacional.

  En la figura 4.26 se puede ver el resultado de situar metaelipses a lo largo de una arista.

  En el algoritmo 8 se muestra la forma de proceder.

  Según el algoritmo 8, el número de aristas imaginarias generadas será el número de metaelipses que quepan en la distancia d.

Algoritmo 8 Generación del número de metaelipses.

for all t: triángulo del plano **do**
  if t es de tipo S **then**
    $d \longleftarrow$ distancia entre las dos aristas (en los extremos distantes evidentemente).
    if $d >$ tamaño de metaelipse **then**
      Generar aristas imaginarias (ver figura 4.27).
      Calcular sus puntos de conexión.
    end if
  else if t es de tipo J **then**
    $d \longleftarrow$ longitud de la arista mínima de t.
    if $d >$ tamaño de metaelipse **then**
      Generar aristas imaginarias (ver figura 4.27).
      Calcular puntos de conexión.
    end if
  end if
end for
Rellenar aristas del plano + aristas imaginarias generadas, con metaelipses.

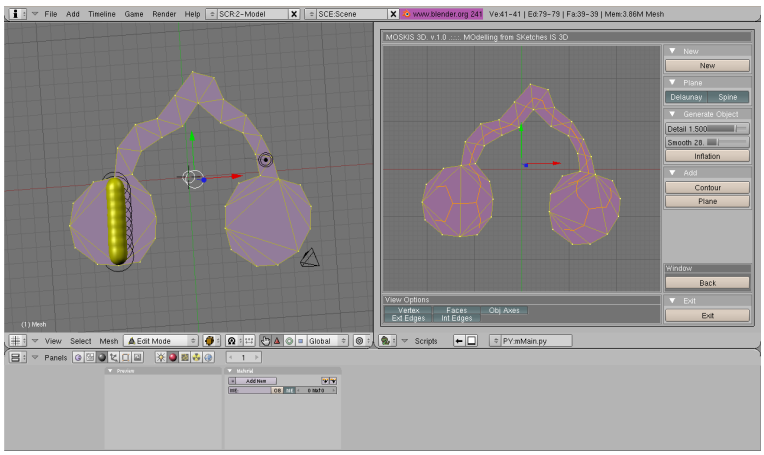

Figura 4.26: Situación de metaelipses a lo largo de una arista.

El cálculo de los puntos de conexión para estas aristas imaginarias generadas se realiza de la forma:

- La posición (x, y) del punto de conexión será el punto medio de la arista imaginaria.
- La posición z se calcula, hallando el incremento o decremento que se produce entre las dos aristas internas (no imaginarias) que las rodean. Una vez calculado este valor, cada arista imaginaria incrementará o decrementará su valor z, para producir una cambio gradual entre un punto y otro (ver figura 4.28).

Una vez calculado todo, se rellena mediante metaelipses toda la longitud de cada arista como se puede ver en la figura 4.26. Este proceso se realiza para todas las aristas, aristas del plano y aristas imaginarias generadas.

De esta forma queda cubierta toda la superficie del plano que define nuestro objeto mediante metaelementos, aún sin altura.

En las figuras 4.29 y 4.30, se puede ver el resultado del algoritmo descrito hasta ahora. La figura presenta como se cubre la superficie del plano mediate metaelipses, quedando la figura definida con bastante detalle y de una forma homogénea.

En la figura 4.29 aparece el objeto generado con una metasuperficie, sin embargo, en la figura 4.30, el modelo obtenido es una malla poligonal. Del paso de metasuperficie a

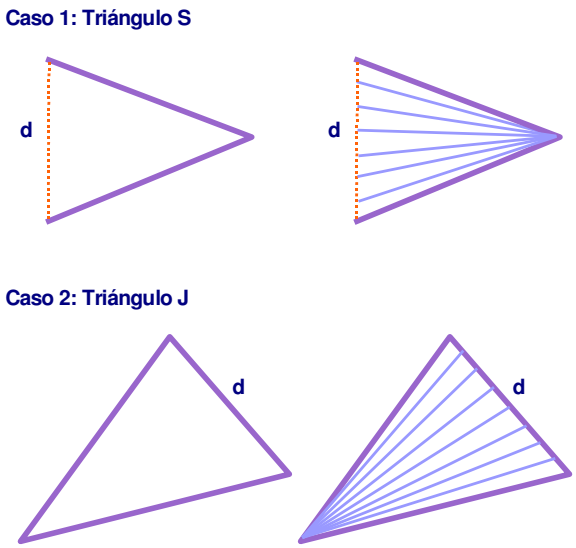

Figura 4.27: Generación de aristas imaginarias.

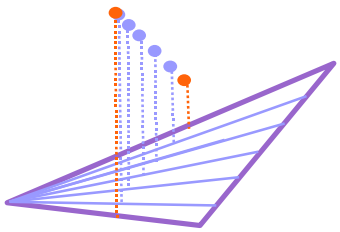

Figura 4.28: Cálculo del punto de conexión de aristas imaginarias.

malla poligonal se hablará más adelante.

- **Tamaño de las metaelipses (coordenada z).**

  El siguiente paso, es dar volumen a la figura, para ello se tiene que dar un valor al tamaño de la metaelipse sobre el eje Z, es decir, la altura.

  - Incremento lineal de alturas.

    Para cada arista (del plano + imaginarias), se tienen las tres coordenadas del punto de conexión asociado a ella.

    Suponiendo $Z = 0$ en las metaelipses de los extremos de una arista, y $Z =$

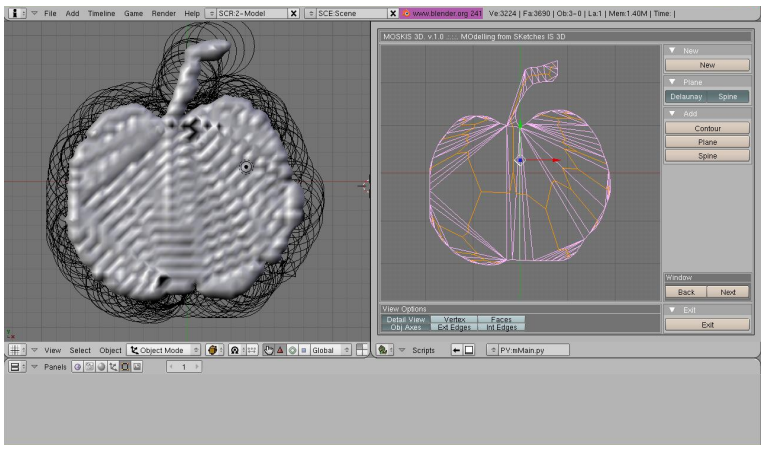

Figura 4.29: Resultado 1 de MOSKIS 3D cubriendo la superficie, sin dimensión en Z.

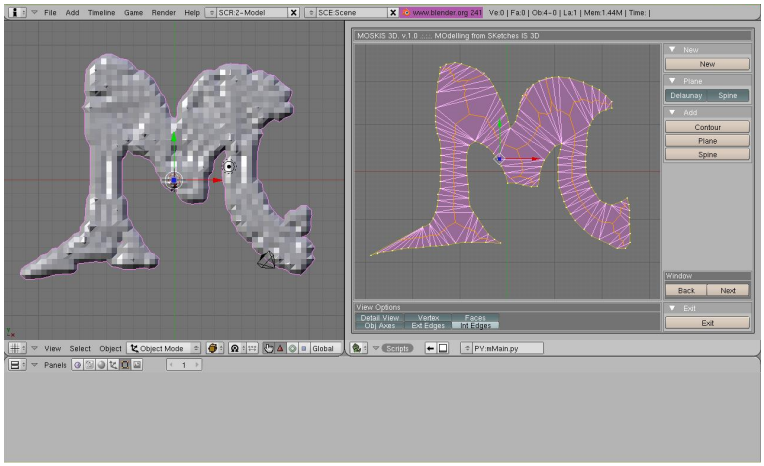

Figura 4.30: Resultado 2 de MOSKIS 3D cubriendo la superficie, sin dimensión en Z.

coordenada Z del punto de conexión, en la metaelipse central de la arista. Se hace un incremento lineal desde cada extremo de la arista, hasta el centro o punto de conexión, dando estos valores incrementales a la coordenada Z de cada metaelipse situada en la arista (ver figura 4.31).

Con la decisión tomada de asignar $Z = 0$ en las metaelipses de los extremos de las aristas, surge un problema reflejado en la 4.32.

Si las aristas imaginarias de los triángulos de tipo J, tienen en los extremos una altura $Z = 0$, en uno de los extremos (el extremo que coincide con la tercera arista

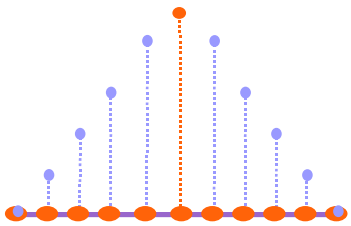

Figura 4.31: Valores de la coordenada Z para cada metaelipse.

interna del triángulo J) se producirá un hundimiento entre el punto de conexión de las aristas imaginarias y el punto de conexión de la arista interna del triángulo J que cortan.

Este efecto se puede ver en la figura 4.33, donde $p1$, $p2$ y $p3$ son los puntos de conexión de las aristas de un triángulo de tipo J, el paso desde los puntos $p1$ y $p2$ hasta la arista interna que tiene como punto de conexión el punto $p3$ debería ser gradual, pero se produce un paso por $Z = 0$ en los extremos de las aristas imaginarias, que provoca el hundimiento.

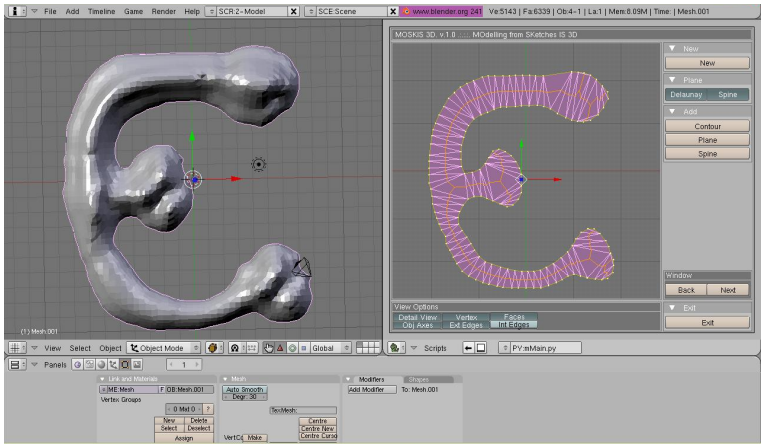

Figura 4.32: Hundimientos en triángulos de tipo J.

Para evitar este hundimiento en la figura, la coordenada Z de uno de los extremos de la arista imaginaria ya no va a valer $Z = 0$, va a tener el valor de la coordenada Z de la metaelipse (de la arista interna del triángulo) coincidente con este extremo. La arista interna del triángulo a la que pertenece la metaelipse de la que se habla,

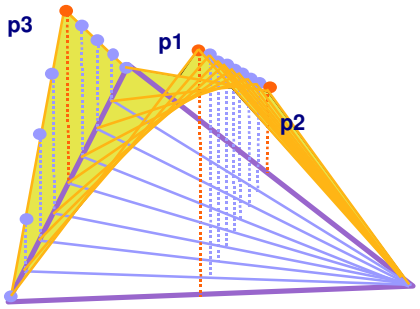

Figura 4.33: Ejemplo de hundimiento en un triángulo de tipo J.

es la que es cortada por la arista imaginaria (ver figura 4.34).

Para ello se haría un incremento (o decremento) lineal gradual dede la coordenada Z de los puntos de conexión de las aristas imaginarias, hasta la coordenada Z de la metaelipse coincidente, perteneciente a la arista interna del triángulo J que corta (ver figura 4.34).

Esto se cumple sólo para aristas imaginarias generadas en un triángulo de tipo J. Para las aristas del plano y para las aristas imaginarias de triángulos de tipo S se sigue el procedimiento anterior.

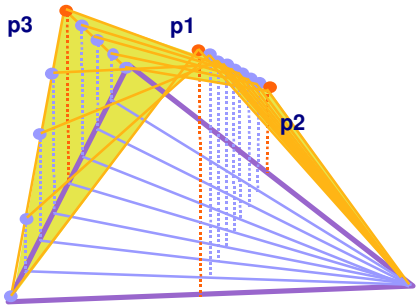

Figura 4.34: Ejemplo de la solución al hundimiento en un triángulo de tipo J.

Mediante esta aproximación, se obtienen resultados bastante realistas (ver figuras 4.35 y 4.36), pero se aprecia que en elevaciones destacadas, es decir, zonas muy elevadas de la figura, se generan porciones piramidales o picos, debido al incremento lineal de la coordenada Z en las metaelipses. Este efecto se aprecia sobre todo en aristas de gran longitud y con un punto de conexión de coordenada Z

elevada (ver figura 4.31).

La solución sería incrementar la coordenada Z en vez de linealmente, circularmente.

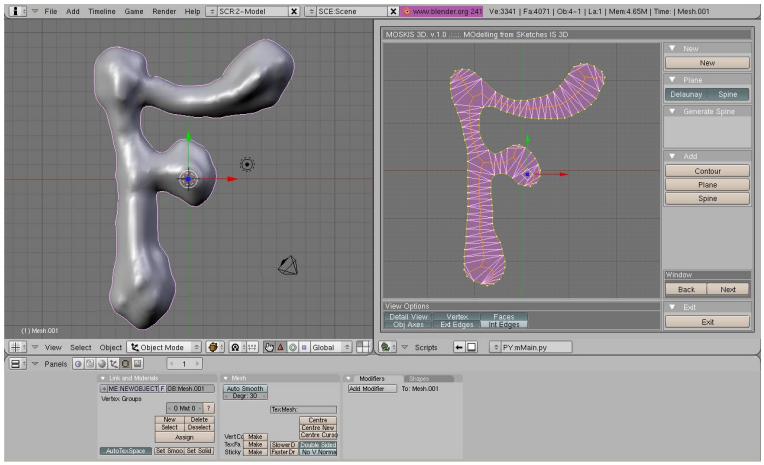

Figura 4.35: Resultado 1 de la versión con incremento lineal.

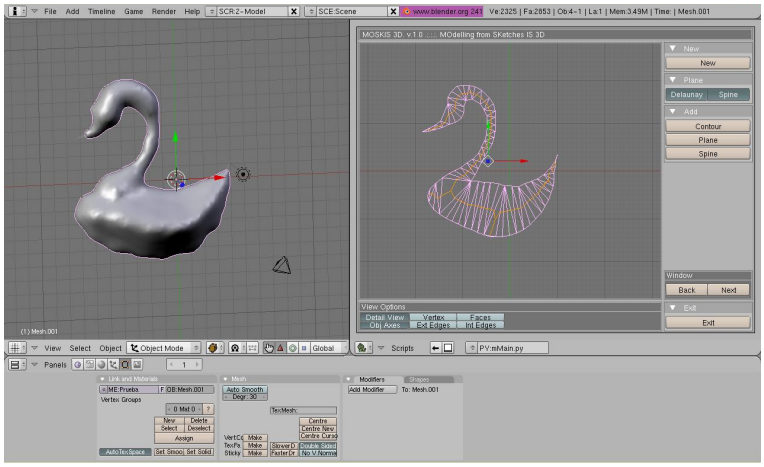

Figura 4.36: Resultado 2 de la versión con incremento lineal.

- Incremento circular de alturas.

  Para evitar los efectos producidos por los incrementos lineales en el eje Z, se toma la determinación de hacer incrementos circulares como se muestra en la figura 4.37.

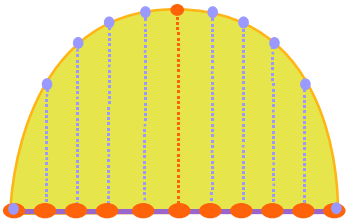

Figura 4.37: Valores de la coordenada Z para cada metaelipse.

Se tienen tres puntos, que son los extremos de la arista y el punto de conexión. Se necesita un arco que pase por esos tres puntos.

En primer lugar se va a obtener la ecuación de la circunferencia que pasa por esos tres puntos. Una vez obtenida la ecuación, se sustituirá la coordenada X de cada metaelipse, para obtener, la coordenada Z.

Se ha tomado como plano para hacer este cálculo el (X, Z), aunque también se podría haber tomado el plano (Y, Z), ya que se tienen como datos, tanto la coordenada X como la Y, y lo que interesa es la coordenada Z.

La ecuación de la circunferencia con centro en el punto (a, b) y radio c es:

$$(x - a)^2 + (z - b)^2 = c^2 \tag{4.3}$$

Se necesita el centro del triángulo y el radio a partir de él.

El algoritmo 9 muestra como hallar centro de una circunferencia que pasa por tres puntos $p1$, $p2$ y $p3$.

Una vez obtenido el centro del triángulo (a, b), se obtiene el radio c, como la distancia entre (a, b) y cualquiera de los puntos $p1$, $p2$ o $p3$. Teniendo ya todos los datos, para calcular cada coordenada Z, basta con sustituir en la ecuación la coordenada X correspondiente.

Una vez obtenido el conjunto de metabolas que conforman el objeto 3D asociado al contorno inicial que pintó el usuario, se realiza un paso mediante el API de

Algoritmo 9 Centro de una circunferencia que pasa por tres puntos

1: Se traza el triángulo que pasa por los puntos $p1$, $p2$ y $p3$.

2: Para dos segmentos del triángulo, se halla su mediatriz. La mediatriz de un segmento corta al segmento por su mitad y perpendicularmente (ver figura 4.38).

3: Se halla la ecuación implícita de cada mediatriz de la forma:

$\frac{Z-Z_1}{X-X_1} = \frac{Z_2-Z_1}{X_2-X_1}$

Poniéndolo en función de X y Z de la forma $AX + BZ + C = 0$ quedaría:

$A = \frac{Z_2-Z_1}{X_2-X_1}$

$B = -1$

$C = Z_1 - (\frac{Z_2-Z_1}{X_2-X_1})X_1$

4: Se resuelve el sistema de ecuaciones formado por las dos ecuaciones correspondientes a las dos mediatrices.

$A_1X + B_1Z + C_1 = 0$

$A_2X + B_2Z + C_2 = 0$

5: El punto (X, Z) formado por la solución del sistema de ecuaciones, será el punto central de la circunferencia, es decir (a, b).

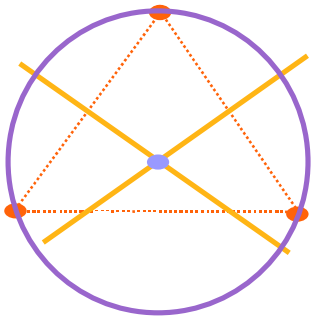

Figura 4.38: Mediatrices de dos segmentos de un triángulo.

Blender-Python, para convertir estos objetos en una malla poligonal que es el resultado deseado.

Mediante esta solución, la malla obtenida adquiere un aspecto mucho más realista, al ser más suave en las elevaciones como se puede ver en las figuras 4.39 y 4.40.

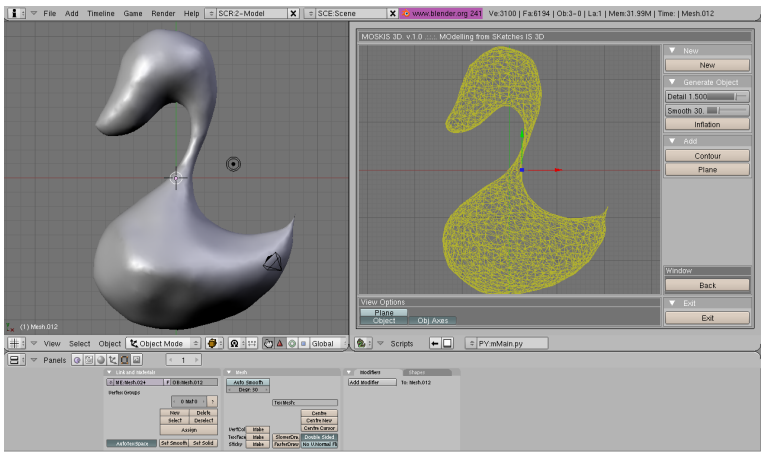

Figura 4.39: Resultado 1 de la versión con incremento circular.

Se puede observar en las figuras comentadas anteriormente, que en la ventana de MOSKIS3D también se muestra un modelo 3D semejante al que se muestra en la ventana 3D de Blender. No es un modelo exactamente igual, ya que tiene una reducción de vértices mediante la función anteriomente citada *remDoubles*, para poder manejarla más fácilmente, pero es una aproximación bastante cercana.

En la figura 4.41, se puede observar el número de vértices generados en la malla 3D de

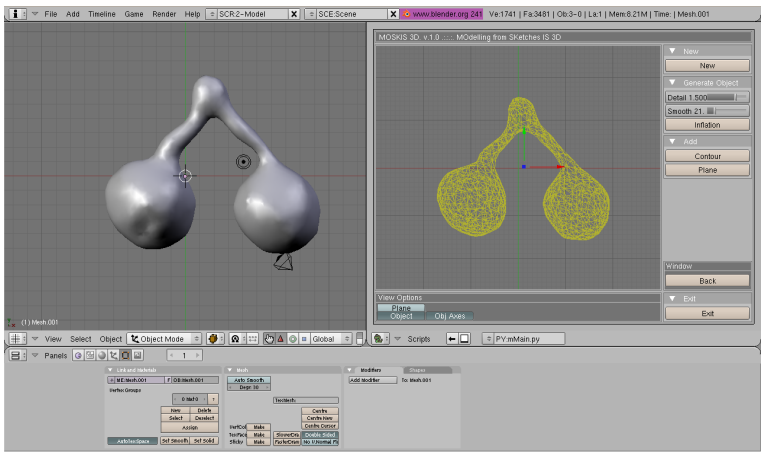

Figura 4.40: Resultado 2 de la versión con incremento circular.

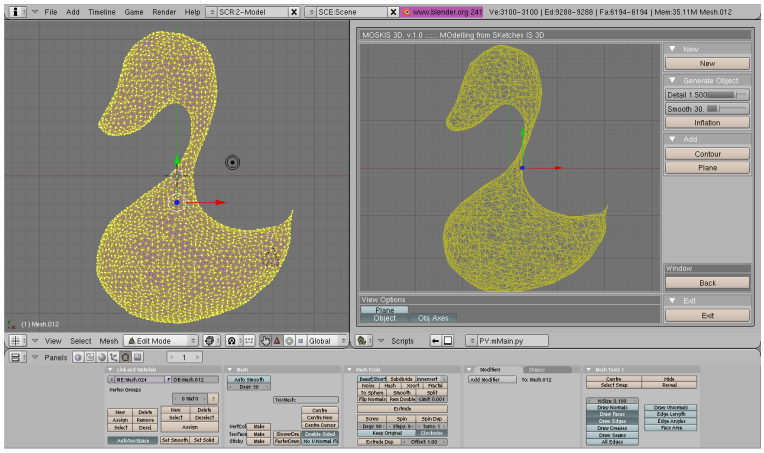

Figura 4.41: Resultado 3 de la versión con incremento circular.

la ventana de Blender. En la siguiente sección se hablará de como se genera la malla 3D en la ventana de Blender.

El dibujado de la malla en la ventana de MOSKIS 3D, se realiza mediante *Blender.BGL*, símplemente mediante la función que se muestra en el código fuente 4.6.

Código fuente 4.6: Dibujar malla.

```
def drawObject(lColor, dGlobalObject, dWindowData,
    dCameraData):
    #Como los ejes de coordenadas, van a sufrir las
```

```
    #modificaciones debidas a las operaciones de rotación
    #y span (no las de escalado) pues obtienen los datos
    #del diccionario cameraData
    glLoadIdentity()
    lCameraPosition = dCameraData["cameraPosition"]
    lSpan = dCameraData["span"]
    lRotationAngle = dCameraData["rotationAngle"]
    iScaled = dCameraData["scaled"]
    #Primero se desplaza el objeto según lo indicado en el
    #span porque es dónde el usuario lo ha dejado colocado
    glTranslatef(lCameraPosition[0]+lSpan[0],
        lCameraPosition[1]+lSpan[1],
        lCameraPosition[2]+lSpan[2])
    #Procedemos a la rotación del objeto según los
    #parámetros indicados
    glRotatef(lRotationAngle[0],1,0,0)
    glRotatef(lRotationAngle[1],0,1,0)
    glRotatef(lRotationAngle[2],0,0,1)
    glScaled(iScaled, iScaled, iScaled)

    lCenterAreaPaint =
        dWindowData["centerAreaPaintCoords"]

    lTriangles = dGlobalObject["object"].getTriangles()

    for i in lTriangles:
        lEdges = i.getEdges()
        for j in lEdges:
            drawEdge(lColor, j.getVO(), j.getVD(),
                dWindowData, dCameraData)
```

```
glLoadIdentity()
```

4.2.4.3. Añadir la malla a la ventana 3D de Blender.

El usuario, a la hora de generar el objeto 3D, tiene la posibilidad de modificar dos parámetros:

- Nivel de detalle: Esta barra de deslizamiento, aumenta o disminuye el nivel de detalle, lo que se traduce, en que disminuye el tamaño fijo en el eje X e Y de las metaelipses que componen el objeto. Cuánto más nivel de detalle, más metaelipses, y por tanto mucho más tiempo de generación.

  El nivel de detalle establecido por defecto, intenta establecer un equilibrio entre calidad y coste computacional.

- Smooth: Esta barra de deslizamiento, indica el número de veces que se aplicará el algoritmo de suavizado (API de Blender-Python) para la malla 3D obtenida.

  Este valor, se calcula automáticamente, como media de las coordenadas Z de todos los puntos de conexión de la espina dorsal, de esta forma, para superficies con más elevación, se aplicará más suavidad y para superficies con menos elevación será menos necesaria.

  Aún así este parámetro puede ser modificado por el usuario, no aumenta de forma significativa el tiempo de generación.

  Su efecto puede ser el limado de detalles importantes si se abusa de él, o mallas demasiado rugosas en algunos casos, si no se aplica.

  La rugosidad en determinadas mallas, puede ser lo deseado en algunas ocasiones, ya que los modelos orgánicos tienen irregularidades y puede dar un aspecto más realista al objeto.

Cuando el usuario pulsa el botón Inflation, se desencadena el algoritmo explicado en el apartado anterior, y se crea la malla en la ventana 3D de Blender siguiendo los siguientes pasos:

1. Se crea un objeto *Metaball* de Blender.

   Mediante este objeto se acceden a los datos de las metasuperficies en Blender.

2. Se añaden los metaelementos (metaelipses) al objeto *Metaball* (véase código fuente 4.7).

   El objeto final va a ser un conjunto de metaelementos que van a formar un objeto.

   Los metaelementos se añaden mediante la función *addMetaelem*, con parámetros destacados como:

   - Tipo de metaelemento: el 6 indica metaelipse.
   - Coordenadas de posición: posición del metaelemento en las coordenadas X, Y y Z.
   - Tamaño del metaelemento: tamaño del metaelemento en las coordenadas X, Y y Z.

Código fuente 4.7: Código de creación del objeto formado por metaelipses.

```
#Creación de un objeto metaball en Blender.
oOb = Blender.Object.New("Mball", "MOSKIS")
oMb = Blender.Metaball.New()
... #Se calculan los datos de los metaelementos
for k in lMetaballPos:
    lPos = k["pos"]
    fTam = k["tam"]
    fElevation = k["elevation"]
    oMb.addMetaelem([6,lPos[0],lPos[1],0,2,1,2,fTam,
        fTam,fElevation])
#Se añaden los metaelementos segun los datos de
#posición y tamaño en cada coordenada.
oOb.link(oMb)
```

3. Se añade el objeto a la escena actual de Blender (véase código fuente 4.8).

   Se obtiene la escena actual de Blender, en la que se está trabajando. Las escenas son la forma de organizar el trabajo en Blender.

   Una vez obtenida la escena actual, se añade el objeto a la escena. Este paso es necesario, porque es justo en ese momento cuando se realizan todos los cálculos de los metaelementos y cuando realmente se obtienen los datos del objeto, y se puede acceder a ellos. Este paso consume gran cantidad de recursos y tiempo.

   Código fuente 4.8: Añadir objeto a la escena actual.

```
#Obtener la escena actual
oSc = Blender.Scene.getCurrent()
#Añadir el objeto y actualizar la escena
oSc.link(oOb)
oSc.update(1)
```

4. Se genera la malla poligonal a partir del objeto de tipo *Mball* como un objeto *NMesh* (véase código fuente 4.9).

   Para ello se obtiene los elementos de la escena que estén seleccionados (cuando un elemento se añade a la escena, ese elemento queda seleccionado por defecto), como se acaba de añadir el objeto de tipo *Mball*, es fácil obtenerlo.

   A continuación mediante la función del API de Blender *GetRawFromObject*, se obtiene la malla poligonal a partir del objeto. Esta función obtiene la malla poligonal de cualquier tipo de objeto de Blender.

   A continuación, se quita el objeto de tipo *Mball* de la escena y se añade la malla poligonal.

   El paso anterior de añadir el objeto de tipo *Mball* a la escena, es un paso intermedio, necesario para realizar los cálculos, ya que hasta que el objeto no se visualiza en Blender, no se calcula. Este paso no se aprecia visualmente en la ejecución, pero como se ha dicho anteriormente tiene un coste computacional muy alto.

Código fuente 4.9: Obtener malla poligonal.

```
#Se obtienen los objetos de la escena actual
selected = oSc.getChildren()
for i in selected:
    if i.getType() == 'MBall':
        #Se obtiene la malla poligonal
        me = Blender.NMesh.GetRawFromObject(i.getName())
oSc.unlink(oOb) #Se quita de la escena el objeto Mball
Blender.NMesh.PutRaw(me) #Se añade la malla
```

5. Suavizado del objeto (véase código fuente 4.10).

   Para obtener la malla final suavizada, en primer lugar se accede a la malla poligonal, mediante el objeto *Mesh* de Blender, que se diferencia del objeto *NMesh*, en que *NMesh* es útil para crear mallas y añadirlas a la ventana 3D de Blender, mientras que *Mesh*, es útil a la hora de modificar mallas ya añadidas a la ventana 3D, consumiendo muy pocos recursos y efectuando todas las operaciones de forma rápida.

   Una vez obtenida la malla, se procede a eliminar vértices repetidos o muy próximos sin alterar la figura. Para ello se utiliza la función *remDoubles*, aunque antes se deben seleccionar todos los vértices de la malla, ya que esta función se aplica a los vértices seleccionados (véase código fuente 4.11).

   A continuación se aplica la función *quadToTriangle*, que convierte todas las caras en triángulos, por si en algunos casos se tienen polígonos de cuatro lados.

   Una vez hecho todo esto, se procede al suavizado del objeto, y para esto se aplica la función *smooth* sobre la malla (véase código fuente 4.12), con todos los vértices seleccionados, ya que se realiza únicamente sobre los vértices seleccionados (véase código fuente 4.11).

Código fuente 4.10: Suavizado del objeto.

```
selected_ob = Blender.Object.GetSelected()
#Se suaviza el objeto
```

```
for ob in selected_ob:
    if ob.getType() == 'Mesh':
        the_mesh = ob.getData(mesh=True)
            selAllVertex(the_mesh)
            the_mesh.remDoubles(0.300)
            the_mesh.quadToTriangle(0)
            setSmooth(the_mesh, dButtons["smoothButtonVal"])
```

Código fuente 4.11: Selección de vértices.

```
def selAllVertex(oMesh):
    for oFace in oMesh.faces:
        oFace.sel = 1
    for oVert in oMesh.verts:
        oVert.sel = 1
    oMesh.update()
```

Código fuente 4.12: Función de smooth.

```
def setSmooth(oMesh, numSmooth):
    for oFace in oMesh.faces:
        oFace.sel = 1
        oFace.smooth = 1
        #Las caras se visualizarán suavizadas
        #en vez de con aspecto sólido.
    i = 0
    while i < numSmooth:
        oMesh.smooth()
        i+=1
    #Se deseleccionan los vértices de la malla
    for oFace in oMesh.faces:
        oFace.sel = 0
```

```
    for oVert in oMesh.verts:
        oVert.sel = 0
    oMesh.update()
```

4.2.4.4. Estructuras de datos.

Para esta parte de la ejecución se tiene el diagrama de clases de la figura 4.42

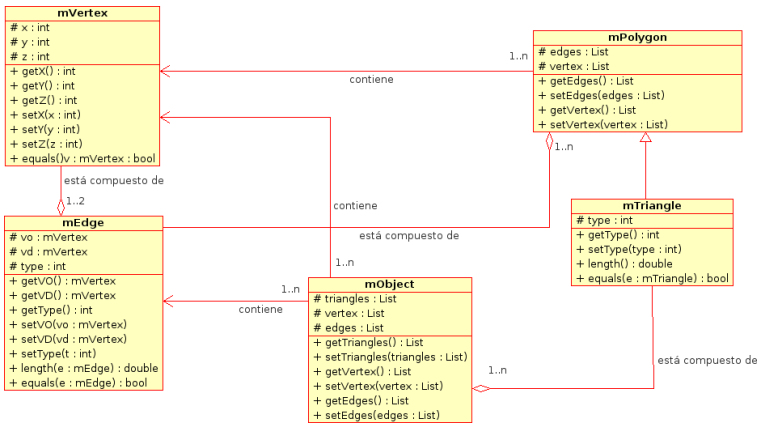

Figura 4.42: Clases utilizadas para la implementación del objeto.

- mObject: un objeto es una lista de triángulos.

  En mObject se almacena también la lista de vértices y aristas, que aunque es mantener información redundante, agiliza los cálculos.

Capítulo 5

Resultados

En este capítulo se muestran algunos de los resultados que se pueden obtener mediante la utilización de MOSKIS 3D y que demuestran su facilidad en la generación de mallas, planos o contornos base, rápidamente.

- **Notas musicales.**

  Este ejemplo, muestra la generación de notas musicales a mano alzada, mediante la opción de añadir planos de la aplicación.

  En las figuras 5.1 y 5.2, se muestran los planos generados por MOSKIS 3D para cada uno de los trazos.

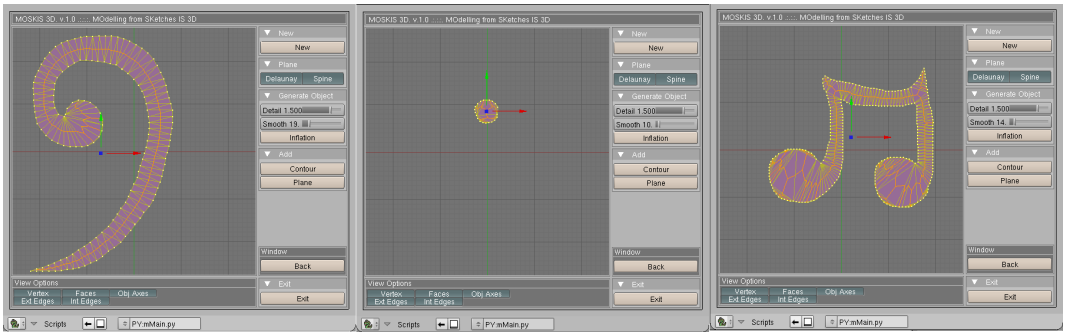

Figura 5.1: Planos 1 generados.

Se ha añadido el plano a la ventana 3D y se ha extruido posteriormente. No se ha aplicado ningún tipo de operador posteriormente, para poder apreciar la malla poligonal

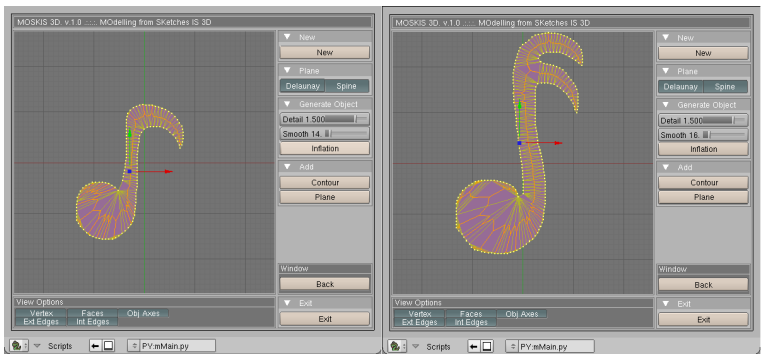

Figura 5.2: Planos 2 generados.

Nota musical	Vértices	Caras
Clave de Fa	154	152
Punto de la clave de Fa	21	19
Dos corcheas	263	261
Corchea	91	89
Semicorchea	225	223

Cuadro 5.1: Tabla de resultados: Notas musicales.

resultante de la extrusión del plano (véase figura 5.3).

- **Cerezas.**

  Este ejemplo, muestra la generación de dos mallas 3D para formar unas cerezas.

  En la figura 5.4, se muestran los trazos realizados para formar las cerezas y en la figura 5.5, los trazos realizados para formar el tronco.

  En la figura 5.6, se muestran las cerezas una vez formadas en la pantalla. Únicamente se les ha aplicado color, es decir, no se ha realizado ninguna operación desde el modelador Blender.

- Pato.

  En el ejemplo, se muestra la generación de la malla 3D asociada al trazo realizado en la figura 5.7 y 5.8.

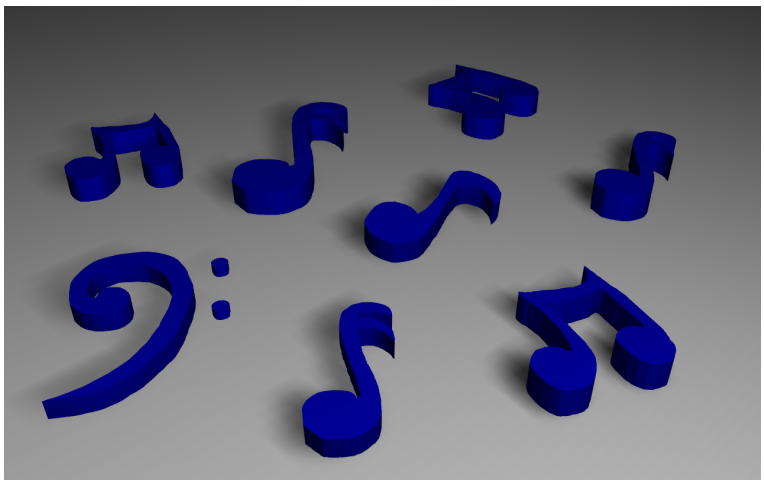

Figura 5.3: Notas musicales. Resultado obtenido.

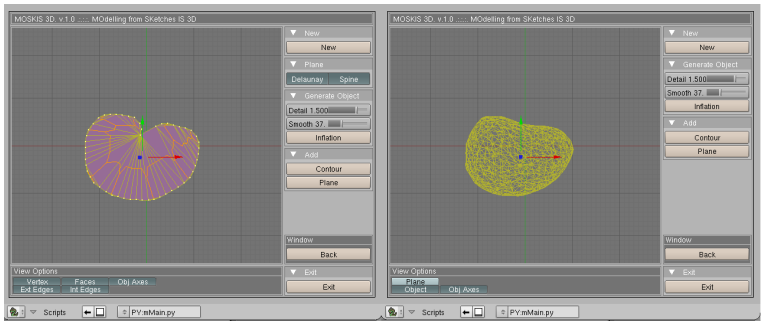

Figura 5.4: Trazo realizado para las cerezas.

En la figura 5.8, se ha hecho un renderizado de la malla sin texturizar y sin ninguna operación posterior a la generación desde MOSKIS 3D.

En la figura 5.9, se muestra el modelo del pato, texturizado pero sin aplicar ninguna operación de Blender, únicamente se han añadido dos esferas negras como ojos.

- **Cara.**

  En el ejemplo, se muestra la generación de la malla 3D asociada al trazo realizado en la figura 5.10 y 5.11.

  En la figura 5.11, se ha hecho un renderizado de la malla sin texturizar y sin ninguna operación posterior a la generación desde MOSKIS 3D.

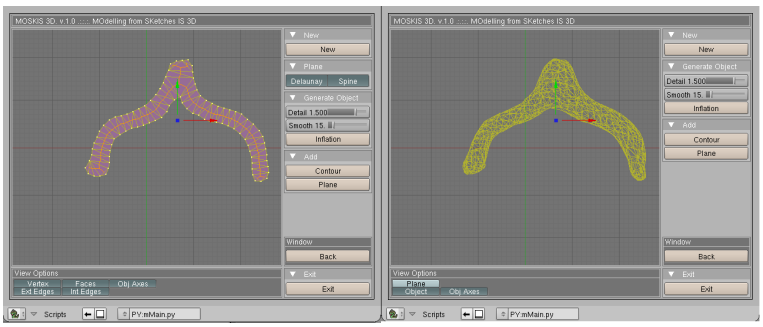

Figura 5.5: Trazo realizado para el tronco.

Objeto	Vértices polígono	Caras plano	Vértices objeto	Caras objeto	Duración
Cereza	51	49	1241	2478	7 seg
Tronco	91	89	1024	2043	11 seg

Cuadro 5.2: Tabla de resultados: Cerezas.

En la figura 5.12, se muestra el modelo de la cara, texturizado y con dos ojos generados en Blender, pero sin ninguna operación aplicada a la malla generada por MOSKIS 3D.

- **Pajarracos.**

  En el ejemplo, se muestra la generación de la malla 3D asociada al trazo realizado en la figura 5.13 y 5.14.

  En la figura 5.15, se ve el resultado de la malla obtenida por MOSKIS 3D.

  En la figura 5.16, se ve la textura que se le va a asignar a la malla, mediante mapeado de texturas UV, proceso que se muestra en la figura 5.17.

  El resultado final obtenido se puede ver en la figura 5.18.

Objeto	Vértices polígono	Caras plano	Vértices objeto	Caras objeto	Duración
Pato	102	100	2345	4679	18 seg

Cuadro 5.3: Tabla de resultados: Pato.

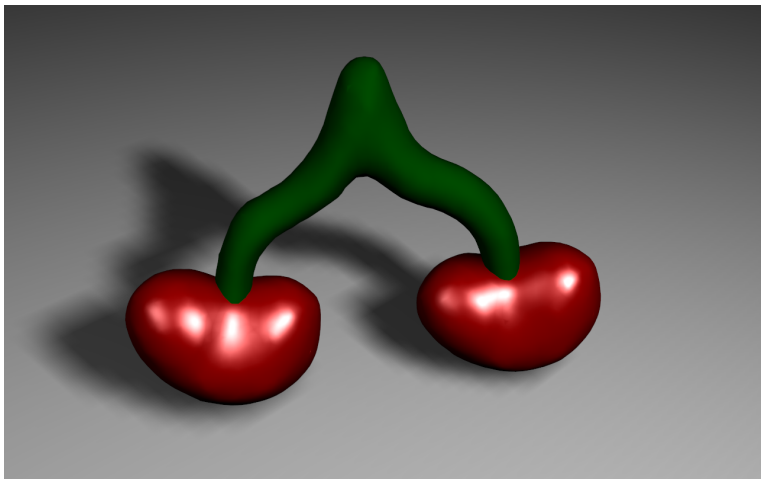

Figura 5.6: Resultado final: Cerezas.

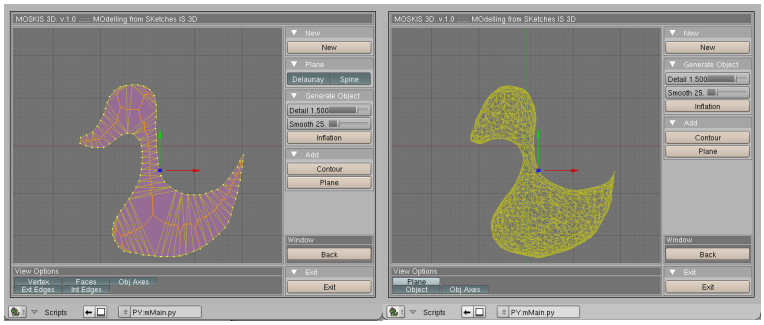

Figura 5.7: Trazo realizado para la generación del pato.

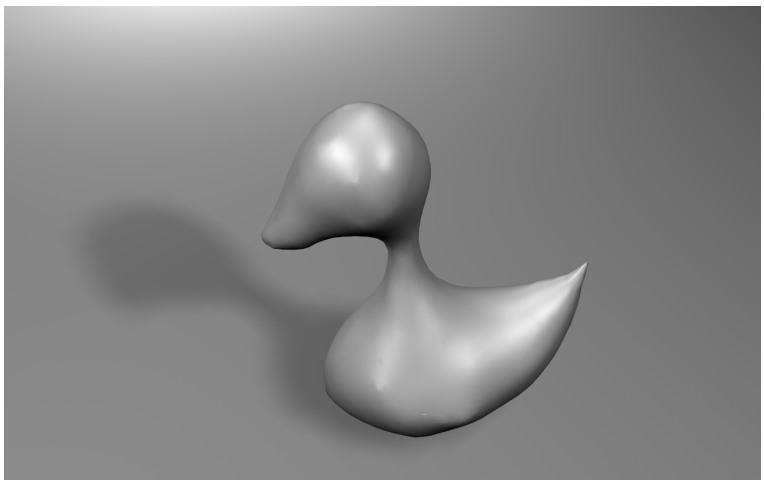

Figura 5.8: Malla generada: Pato.

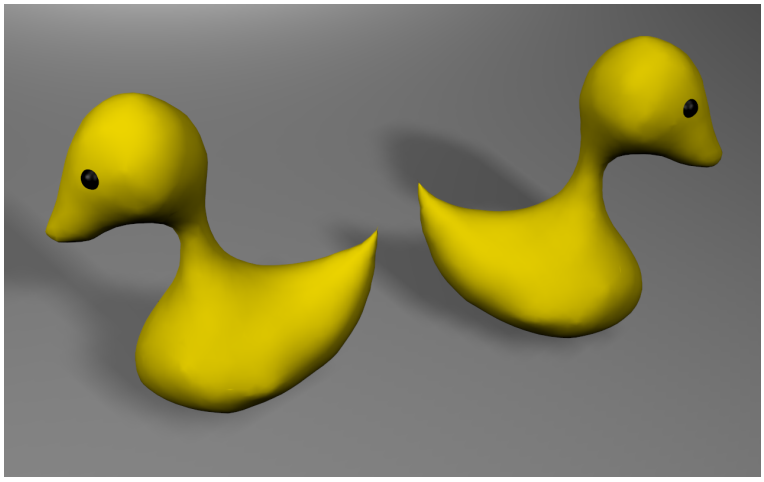

Figura 5.9: Resultado final: Pato.

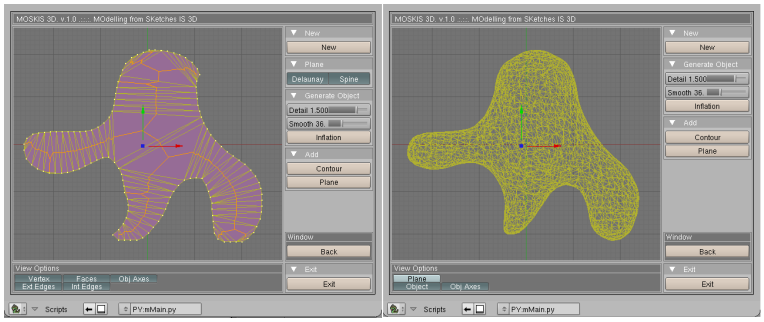

Figura 5.10: Trazo realizado para la generación de la cara.

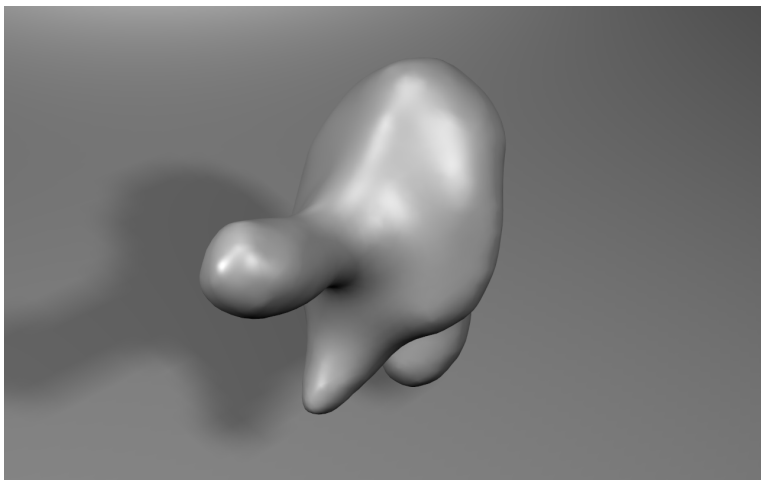

Figura 5.11: Malla generada: Cara.

Objeto	Vértices polígono	Caras plano	Vértices objeto	Caras objeto	Duración
Cara	127	125	3282	6560	35 seg

Cuadro 5.4: Tabla de resultados: Cara.

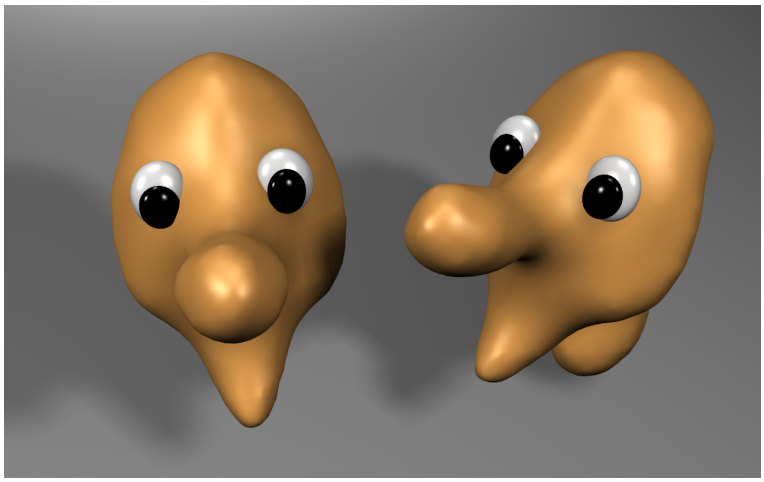

Figura 5.12: Resultado final: Cara.

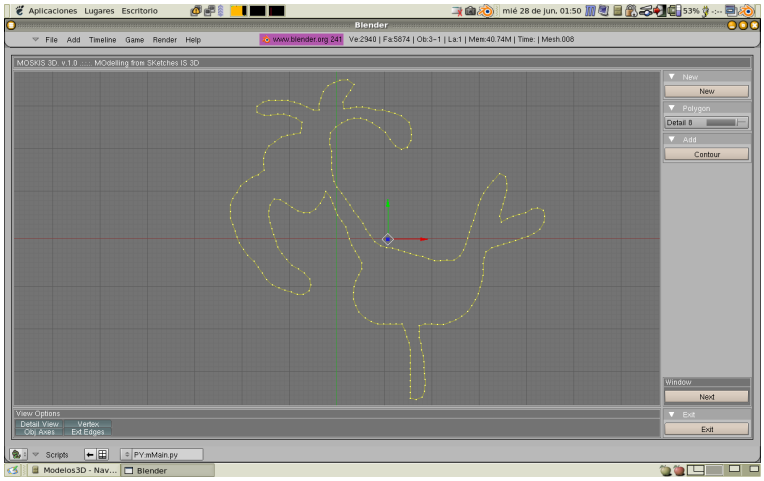

Figura 5.13: Trazo realizado para la generación del pájaro.

Objeto	Vértices polígono	Caras plano	Vértices objeto	Caras objeto	Duración
Pajarraco	192	190	2806	5608	1min, 5 seg

Cuadro 5.5: Tabla de resultados: Pajarracos.

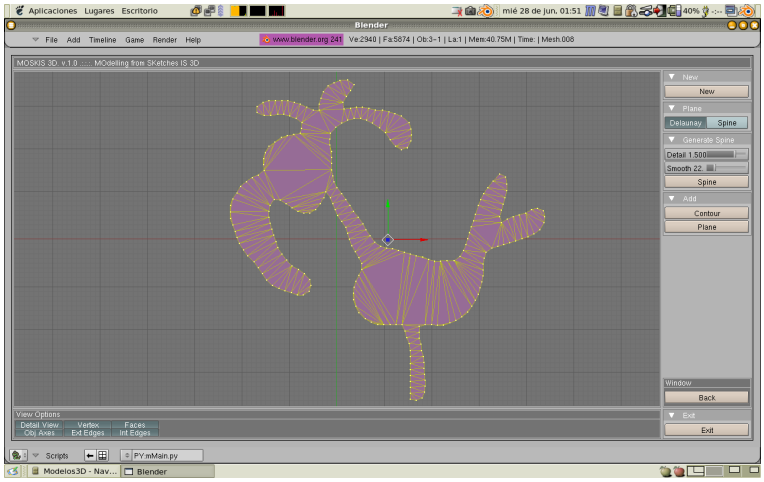

Figura 5.14: Generación de la triangulación para el pájaro.

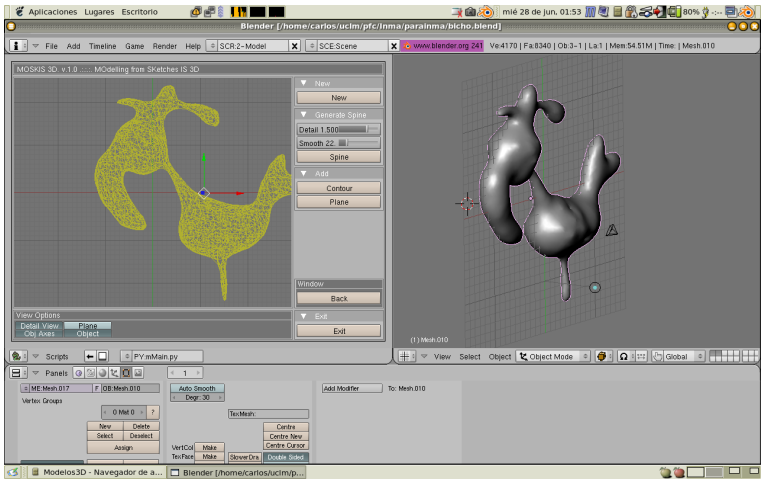

Figura 5.15: Malla generada: Pajarracos.

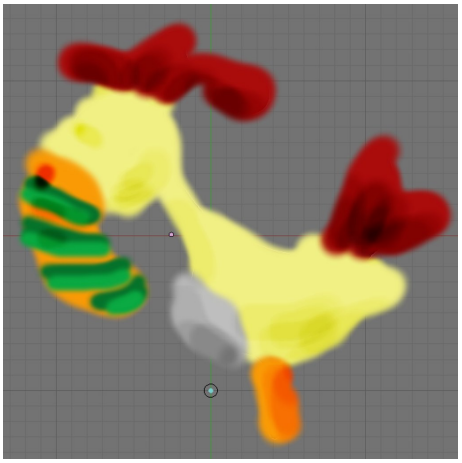

Figura 5.16: Textura a aplicar.

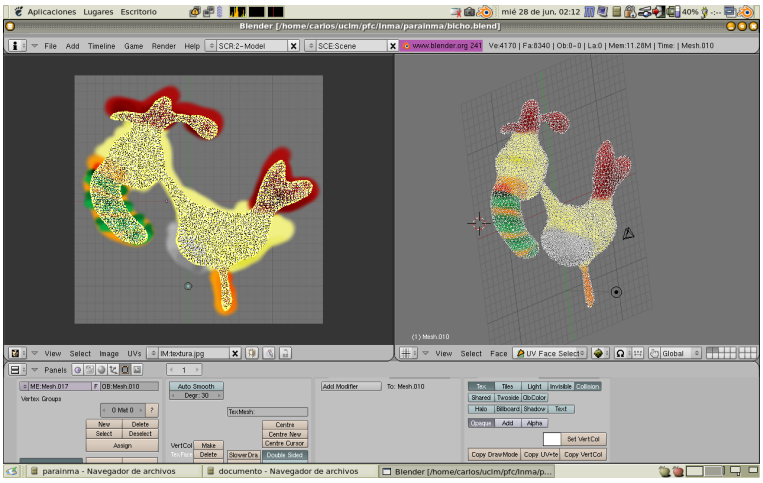

Figura 5.17: Mapeado UV.

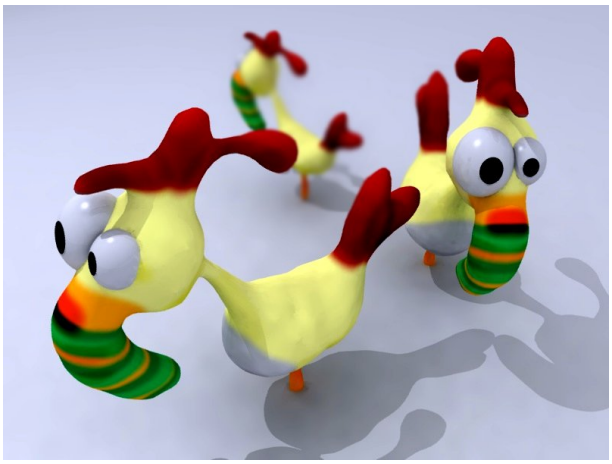

Figura 5.18: Resultado final: Pajarracos.

Capítulo 6

Conclusiones y propuestas futuras.

6.1. Conclusiones.

MOSKIS 3D genera de forma automática mallas 3D a partir de trazos 2D del usuario definiendo un contorno, con lo cual el objetivo principal se ha cumplido, obteniendo una herramienta que:

- Genera de forma instantánea y automática mallas 3D simplemente mediante un trazo, que mediante otras técnicas sería mucho más complejo y costoso.
- A las mallas generadas en MOSKIS 3D, se les pueden aplicar todos los operadores que Blender proporciona, para su modificación.

  Por ejemplo, en la figura 6.1, se muestra la creación del modelo del Quijote en MOSKIS 3D.

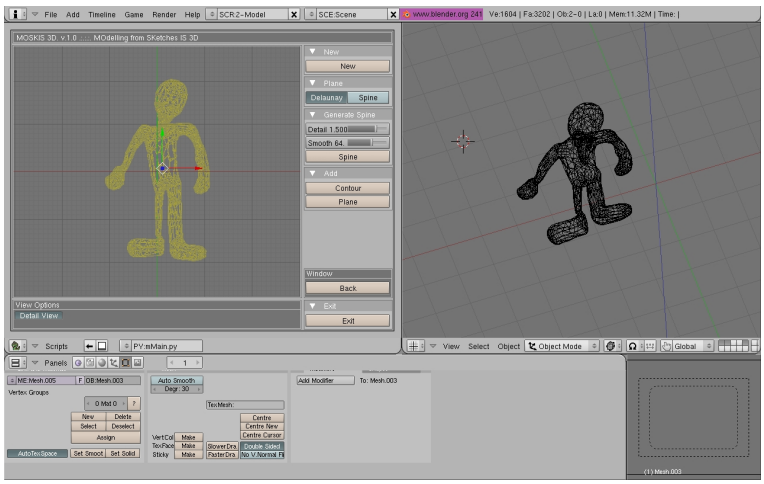

Figura 6.1: Creación del quijote.

En la figura 6.2, se muestra como los modelos pueden variar su pose inicial, empleando esqueletos internos (se ve que el Quijote coge la lanza y cambia la pose de las piernas y del brazo derecho).

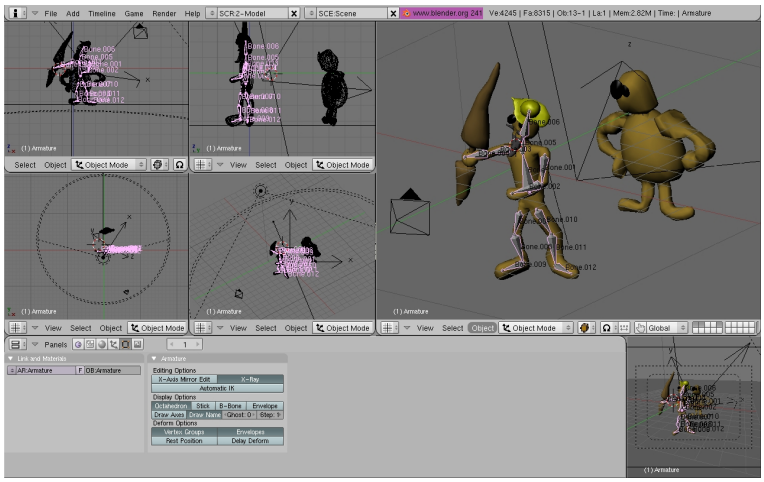

Figura 6.2: Esqueletos internos en el Quijote.

En la figura 6.3, se puede ver el resultado final.

- La resolución final de la malla, se decide en el último paso del algoritmo, decidiento el tamaño de las superficies implícitas de las que se compone la malla en su generación.

Figura 6.3: Resultado del Quijote.

- El periodo de aprendizaje de la herramienta, podría acotarse en minutos, ya que es muy sencilla de utilizar.
- Se obtienen modelos orgánicos realistas, dada la suavidad de la malla obtenida, y en algunos casos, un aspecto irregular que puede dotar la modelo de mayor realismo.
- MOSKIS 3D está desarrollado para una herramienta de producción profesional, este paso es importante, ya que todas las herramientas de este tipo se encuentran en investigación y no existe ninguna herramientas de las características de Blender que incorpore este tipo de modelado.

6.2. Propuestas futuras.

6.2.1. Optimización del tiempo de generación.

La generación de la malla 3D no es inmediata. Requiere cálculos elevados y el tiempo de respuesta puede ser alto, dependiendo del tamaño de la figura y del parámetro de nivel de detalle.

Cuanto mayor es la figura, más metaelipses se generarán para su cálculo y más tiempo de ejecución será necesario.

Cuando mayor es el nivel de detalle, más pequeñas son las metaelipses que generamos, con lo cual, también aumenta el número de metaelipses y el tiempo de ejecución.

Se pretende la disminución del número de metaelipses para una ejecución más rápida, sin el deterioro de la calidad del modelo obtenido.

Se propone un estudio intermedio entre las dos aproximaciones detalladas cuando se hablaba de la creación del objeto mediante metasuperficies, para generar un número intermedio de metaelipses entre las dos aproximaciones.

El algoritmo hasta la obtención del plano permanecería sin variaciones, y se generaría el perfil de alturas de igual forma que se detallaba anteriormente.

Una vez generado el plano, para cada triángulo del plano, se añadirán un conjunto de metaelipses de diferentes tamaños, de forma que las metaelipses de mayor tamaño permaneciesen en el centro de cada triángulo y fuese disminuyendo el tamaño en metaelipses situadas en los vértices, según el ángulo formado. Es una aproximación adaptativa.

Esto tiene sentido, ya que en los extremos de los triángulos del plano, es dónde más se detalla la figura.

6.2.2. Operación de extrusión.

Se pretende la incorporación de la operación de extrusión mediante la misma técnica de trazos, utilizada para la creación.

Blender ya posee extrusión, pero lo que se pretende, es poder seleccionar con un trazo la zona de la malla a extruir, y mediante otro trazo, el camino que va a seguir esa extrusión.

Este método pretende hacer más sencillo, más rápido y de forma homogénea esta operación sobre el objeto que se ha creado.

- Trazo base.

  En primer lugar, el usuario realizará un trazo, seleccionando la zona del objeto 3D a extruir. Este trazo se llamará trazo base (véase figura 6.4).

El trazo se proyectará sobre la figura para ver qué puntos de la figura forman parte de este trazo. Al proyectar el trazo, para cada punto de éste, se obtendrán dos puntos de la figura, uno más cercano (el que nos interesa) y otro más lejano (véase 6.4).

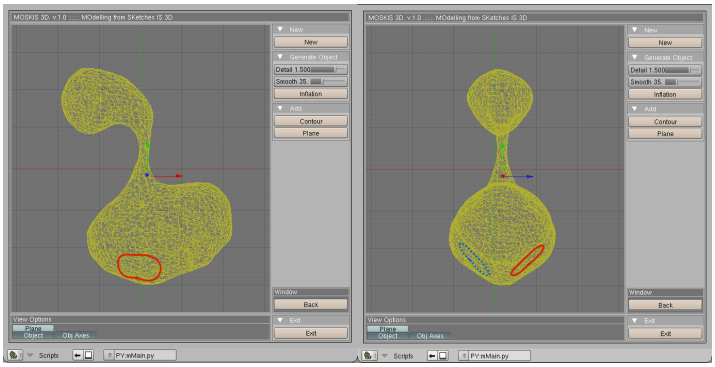

Figura 6.4: Selección del trazo base.

- Trazo de extrusión.

  A continuación, el usuario realizará un trazo, llamado trazo de extrusión, con la forma que desea conseguir con la operación.

  El usuario puede desear una extrusión positiva, es decir, una prolongación del objeto 3D hacia el exterior, siguiendo la forma del trazo realizado; o una extrusión negativa, es decir, una concavidad en el interior del objeto 3D con la forma indicada.

  En la figura 6.5, se observa a la izquierda un trazo de extrusión para una extrusión positiva y a la derecha, el trazo realizado para una extrusión negativa.

- Generación de la extrusión.

  Para la generación de la extrusión se siguen los siguientes pasos:

  1. Se recupera el objeto 3D formado por metaelementos (el que se tenía en la escena antes del paso a malla poligonal).
  2. Se genera un polígono a partir del trazo base realizado por el usuario.

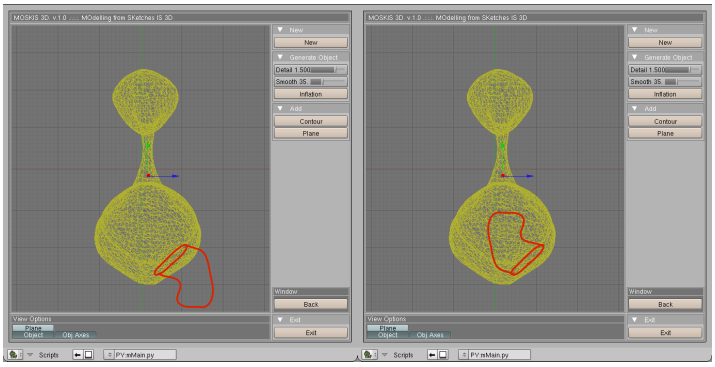

Figura 6.5: Selección del trazo de extrusión.

3. Se procederá a la triangularización básica del polígono resultante.

   A continuación se realizará la optimización basada en Delaunay comentada anteriormente, para repartir la superficie en triángulos de forma homogénea (véase figura 6.6).

   Se continuará detectando la espina dorsal del polígono, obteniendo una serie de puntos conectados, formando el eje central de la superficie seleccionada (véase figura 6.6).

   No se realizarán los cálculos para la elevación de la espina dorsal, porque no se quiere crear un objeto 3D a partir del trazo base, sino extruir la forma del trazo, a lo largo del trazo de extrusión, por lo que no tiene sentido el cálculo del objeto 3D asociado a esa forma, y por tanto, no tiene sentido el paso de elevación de la espina dorsal.

4. Una vez obtenido el plano correspondiente al trazo base, y la espina dorsal de dicho plano, se procederá a rellenar el plano con metaelipses.

   Con este paso, se obtienen un conjunto de metaelementos que definen la superficie delimitada por el trazo base.

   En el caso de una extrusión positiva, las metaelipses generadas tendrán una atracción positiva. En el caso de una extrusión negativa, las metaelipses generadas tendrán una atracción negativa.

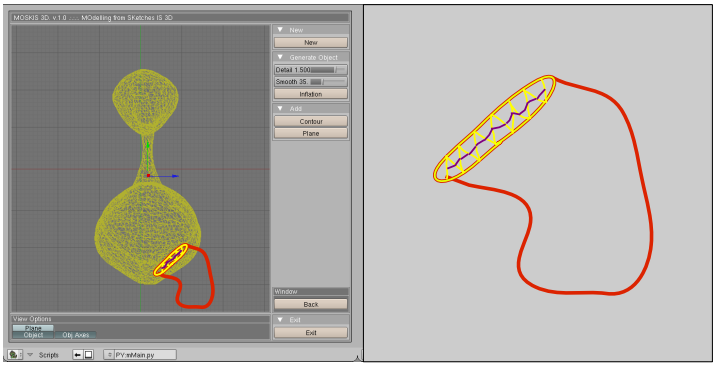

Figura 6.6: Triangulación y espina del trazo base.

5. Se genera un polígono a partir del trazo de extrusión.
6. Se procede a la extrusión del trazo base a lo largo del trazo de extrusión.

   Se realizan proyecciones del conjunto de metaelipses, de forma que cada una de las proyecciones resultantes, queden situadas perpendicularmente con respecto al trazo de extrusión y se redimensionen para ajustarse al tamaño del trazo (véase figura 6.7).

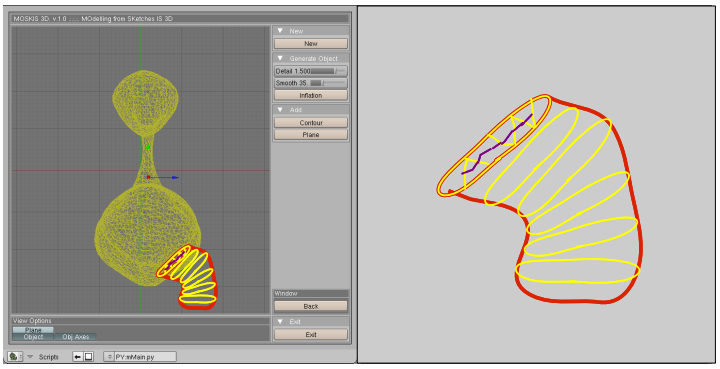

Figura 6.7: Proyecciones.

El número de proyecciones dependerá del nivel de detalle con el cual se haya generado el polígono del trazo de extrusión.

En los casos, en los que la distancia entre los vértices del polígono de extrusión,

sea muy grande, se producirán huecos. Para evitar los huecos se generarán proyecciones intermedias.

7. Se vuelve a generar la malla polígonal a partir del objeto 3D generado (objeto anterior+operación de extrusión).

6.2.3. Carga y almacenamiento.

Sería deseable que el trabajo realizado desde MOSKIS 3D pudiese concluir en varias sesiones.

Para conseguir este objetivo se proponen dos opciones:

- Carga de mallas desde la ventana 3D de Blender.

  El problema de esta proposición, es que si se cargan mallas no generadas con MOSKIS 3D, se tendría que calcular el objeto mediante metasuperficies para poder aplicar después sucesivas extrusiones.

  Pero una implementación que consiguiese esto, permitiría la modificación de cualquier malla, generada en Blender o por MOSKIS 3D.

- Almacenamiento del trabajo.

  Se podría almacenar el trabajo realizado para cargarlo desde MOSKIS 3D y continuarlo hasta su finalización, aunque en el caso en el que se consiguiese cargar cualquier malla desde la ventana 3D de Blender, y posteriormente aplicar la operación de extrusión, la opción de almacenamiento no sería necesaria.

Bibliografía

[BBr97] BBrush. http://grafit.netpositive.hu/content/view/135/26/, 1997.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

[el06] Wikipedia: La enciclopedia libre. http://es.wikipedia.org, 2006.

[Fer05] Marcos Fernández. Modelos de representación de objetos 3d. *Universidad de Valencia*, 2005.

[GG04] I. Gijón and C. González. Sistema de modelado tridimensional basado en trazos libres: Moskis 3d. *Jornadas de Blender*, 2004.

[GM05a] Carlos González Morcillo. Introducción. síntesis de la imagen 3d. *Animación para la Comunicación*, 2005.

[GM05b] Carlos González Morcillo. Modelado. construcción de geometría 3d. *Animación para la Comunicación*, 2005.

[Iga95] Takeo Igarashi. http://www-ui.is.s.u-tokyo.ac.jp/ takeo/index.html, 1995.

[IM99] T. Igarashi and S. Matsuoka. Teddy: A sketching interface for 3d freeform design. *ACM SIGGRAPH'99*, 1999.

[Ker00] I. Victor Kerlow. *The art of 3D computer animation and imaging*. John Wiley & Sons, 2000.

[NR00] L.Ñarváez and I. Rudoming. Mod3d: Modelador 3d basado en trazos 2d. 2000.

[Ram05a] Ricardo Ramos. Esquemas de modelado sólido. *Universidad de Oviedo*, 2005.

[Ram05b] Ricardo Ramos. Informática gráfica: Perspectiva general. *Universidad de Oviedo*, 2005.

[Ram05c] Ricardo Ramos. Introducción al modelado sólido. *Universidad de Oviedo*, 2005.

[Rem05] Inmaculada Remolar. Modelado 3d. *Universidad de Jaume*, 2005.

[RZK96] Hughes J. R.C. Zeleznik K.H. Sketch: An interface for sketching 3d escenes. *ACM SIGGRAPH'99*, pages 163–170, 1996.

[T.94] Baudel T. A mark-based interaction paradigm for free-hand drawing. *Conference UIST'94*, pages 185–192, 1994.

[ZBr97] ZBrush. http://www.pixologic.com/zbrush/home/home.html, 1997.

Anexo A

Instalación de la aplicación.

- Para la instalación del sistema se siguen los siguientes pasos:
  - Instalación de Blender 2.41.

    Es necesaria esta versión de Blender (la última hasta el momento), ya que en la última versión del API Blender-Python, han sido introducidos nuevos módulos utilizados en MOSKIS 3D, que hacen que no sea compatible con versiones anteriores de Blender.
  - Copia de archivos.

    Se copian los archivos de la ruta software/moskis3d del cd adjunto, en la carpeta .blender/scripts de la aplicación Blender.
- Para la ejecución del sistema se siguen los siguientes pasos:
  - Abrir la aplicación Blender.
  - Configurar el área de trabajo de forma cómoda, por ejemplo dividiendo la ventana 3D de Blender que aparece por defecto en dos (véase figura A.1).
  - Pulsar el botón situado a la izquierda de una de las ventanas. Se desplegará un menú para seleccionar el tipo de ventana. Seleccionar *Text Editor* (véase figura A.1).
  - Pulsar la opción *File*, y a continuación *Open*, situada a continuación del menú de selección de tipos de ventanas (véase figura A.2).

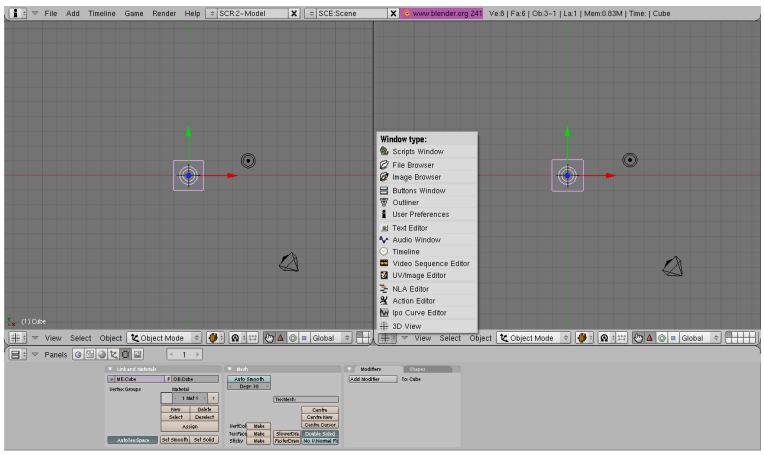

Figura A.1: Selección del tipo de ventana.

Buscar y abrir el archivo *mMain.py* en la ruta .blender/scripts del programa Blender. Aparecerá el código del archivo seleccionado.

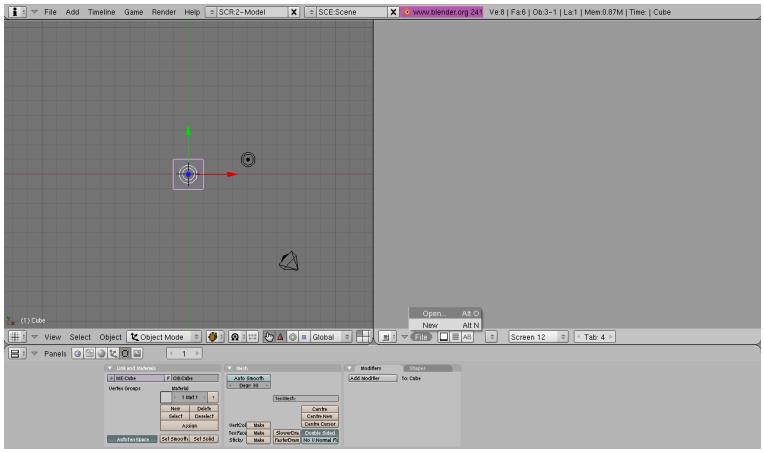

Figura A.2: Abrir archivo.

- Ejecutar la aplicación.

  Para ello, seleccionar del mismo menú *File* citado anteriormente, la opción *Run Python Script*, o ejecutar *Alt+P*.

  En este momento aparecerá en la pantalla la interfaz de MOSKIS 3D.

Anexo B

Manual de usuario.

- Inicio de la aplicación.

  - Detail View: si está activado, se procederá a una ejecución paso a paso, permitiendo la modificación de determinados parámetros. Si no está activado, se generará automáticamente el objeto 3D a partir de un trazo en el área de dibujo (véase figura B.1).
  - Área de dibujo: área en la que se dibuja el contorno 3D.

    Si el botón *Detail View* está desactivado, cuando el usuario termine de generar el contorno, se pasará a la generación del objeto y durante los cálculos, se podrá ver en la parte superior de la ventana, el estado de la ejecución del algoritmo (véase figura B.1).

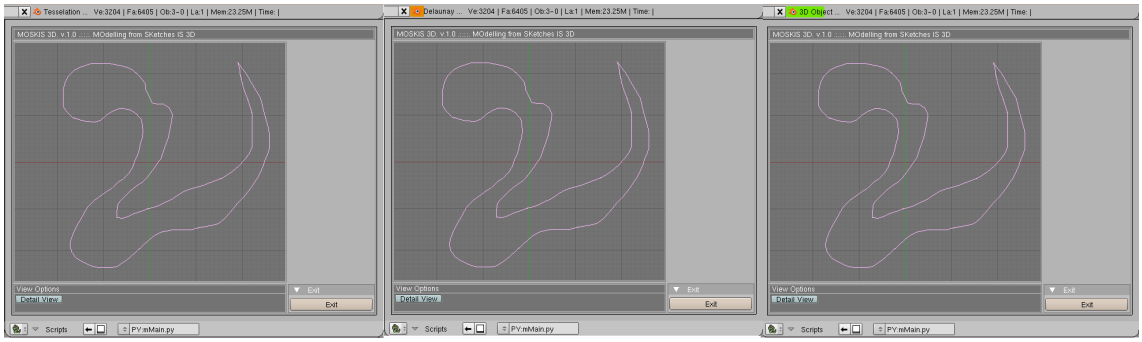

Figura B.1: Generación automática.

En la figura B.1, se muestran los tres estados que se pueden observar según la barra de progreso situada en la parte superior de la aplicación.

Si el botón *Detail View* está activado, se pasará a la pantalla de opciones de un polígono (véase figura B.2) que se detalla a continuación.

- Opciones del polígono.
  - Opciones de visualización.
    - Vertex: visualizar los vértices.
    - Ext. Edges: visualizar las aristas del polígono.
    - Obj. Axes: visualizar los ejes de coordenadas.

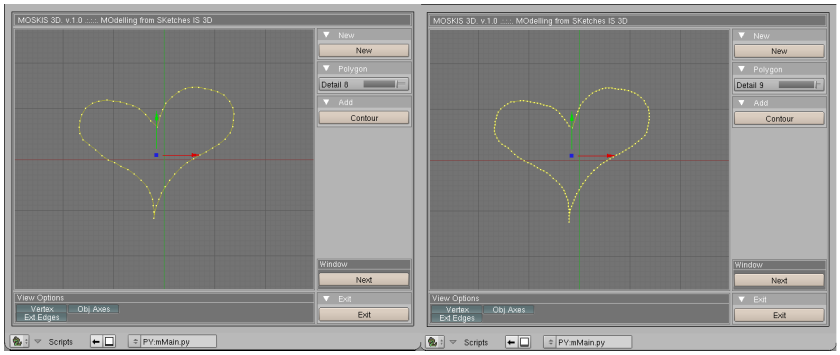

Figura B.2: Opciones Polígono.

  - Paneles.
    - Polygon:
      - Detail: permite la modificación del nivel de detalle del polígono (véase figura B.2).
    - Add:
      - Contour: añade el polígono a la ventana 3D de Blender (véase figura B.3).
    - Window:
      - Next: pasa a la pantalla de opciones del plano.

- Opciones del plano.

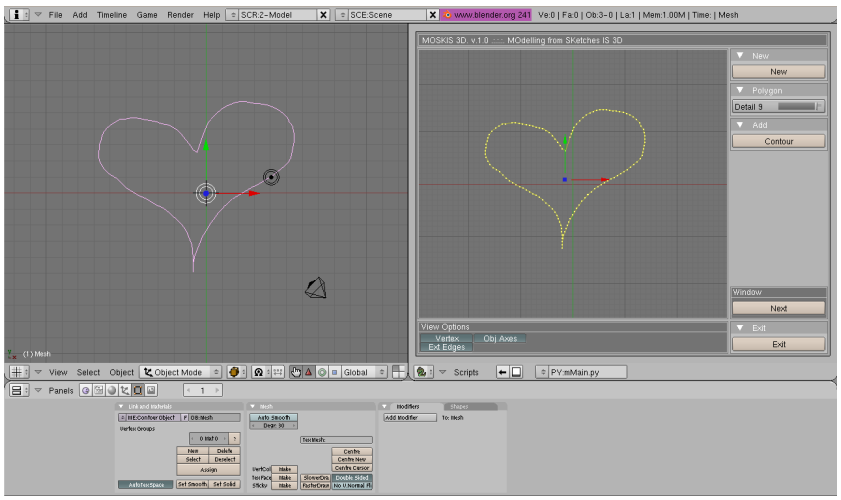

Figura B.3: Opciones Polígono.

- Opciones de visualización.
  - Vertex: mostrar vértices del plano.
  - Ext. Edges: mostrar aristas externas, es decir, las aristas del polígono.
  - Faces: mostrar las caras generadas por la triangulación.
  - Int. Edges: mostrar las aristas internas, es decir, las generadas en el proceso de triangulación.
  - Obj. Axes: mostrar los ejes de coordenadas.
- Área de dibujo:
  - Desplazamiento: mediante el botón derecho del ratón, y su movimiento se puede mover el objeto de un lado a otro.
  - Rotación: mediante el botón central del ratón, se puede rotar el objeto.
  - Escalado: girando la rueda del ratón se puede escalar el objeto.
- Paneles (véase figura B.4).
  - Plane.
    - ◇ Delaunay: si está activado, está calculada la triangulación basada en Delaunay.
    - ◇ Spine: si está activado, se muestra la espina dorsal del plano.

- Generate Object.
  - Detail: permite graduar el nivel de detalle del objeto 3D que se pretende obtener. Aumentarlo puede elevar en gran medida el coste computacional.
  - Smooth: permite graduar el nivel de suavizado de la malla a obtener. Se calcula automáticamente según la figura, pero puede modificarse para obtener otros aspectos.
  - Inflation: genera el objeto 3D.
- Add.
  - Contour: añade el contorno (polígono) a la ventana 3D de Blender.
  - Plane: añade el plano a la ventana 3D de Blender.

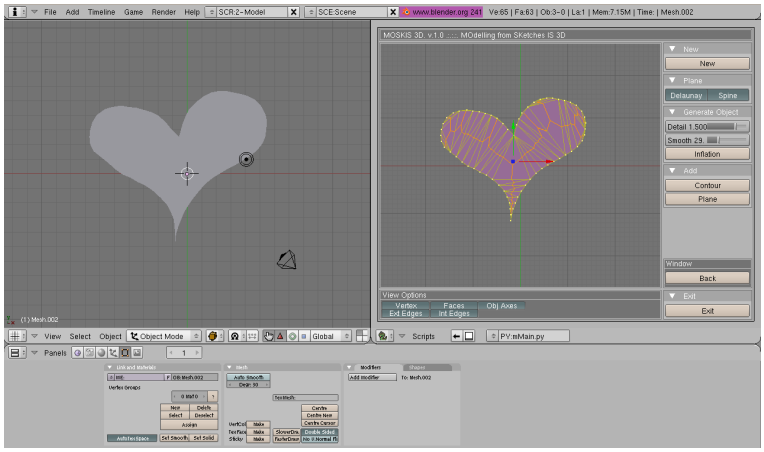

Figura B.4: Opciones Plano.

- Window.
  - Back: vuelve a la pantalla de opciones del polígono.

- Opciones del objeto (véase imagen B.5).
  - Opciones de visualización.
    - Plane: muestra el plano original para comparar el objeto obtenido.
    - Object: muestra el objeto 3D.

- Obj. Axes: muestra los ejes de coordenadas del objeto.

- Área de dibujo: permite las opciones de rotado, escalado y desplazamiento del objeto igual que en el caso anterior.

- Paneles:

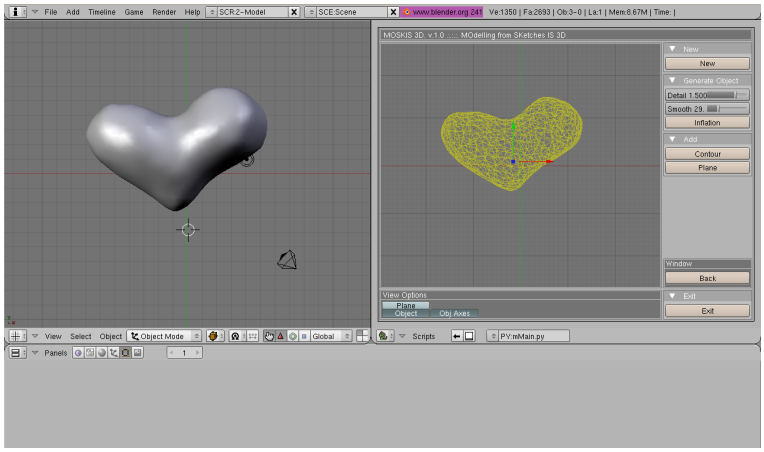

Figura B.5: Opciones Objeto.

  - Generate Object: mismas opciones de Detail, Smooth y Inflation que en la fase de opciones del plano.
  - Add: mismas opciones de Contour y Plane que en la fase de opciones del plano.
  - Window:
    - Back: este botón vuelve a la fase de opciones del plano.

Anexo C

Diagramas de clases.

En la figura C.1 se muestran los diagramas de clases más importantes del sistema.

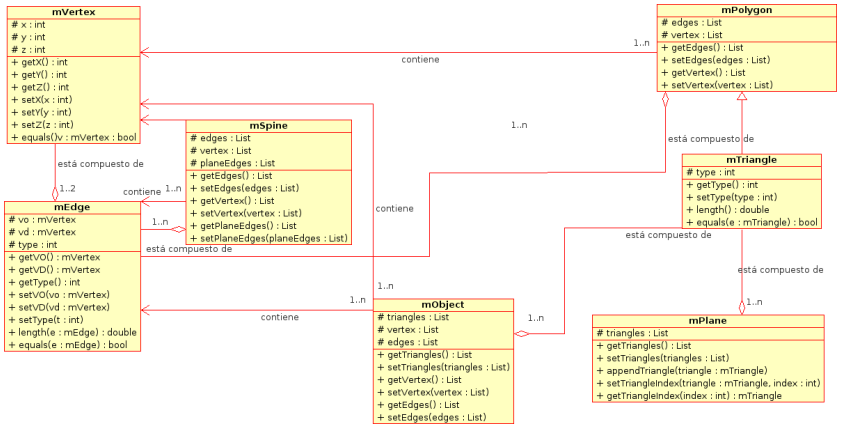

Figura C.1: Diagrama de clases de objetos.

Hay que destacar, que las clases mPolygon, mSpine y mObject, mantienen información redundante para agilizar los cálculos y permitir un fácil acceso.

En la figura C.2 se muestran los diagramas de las clases de utilidades que están implicadas en la creación de un objeto nuevo.

En la figura C.3, se muestran otras clases, incluída la clase principal, mMain.

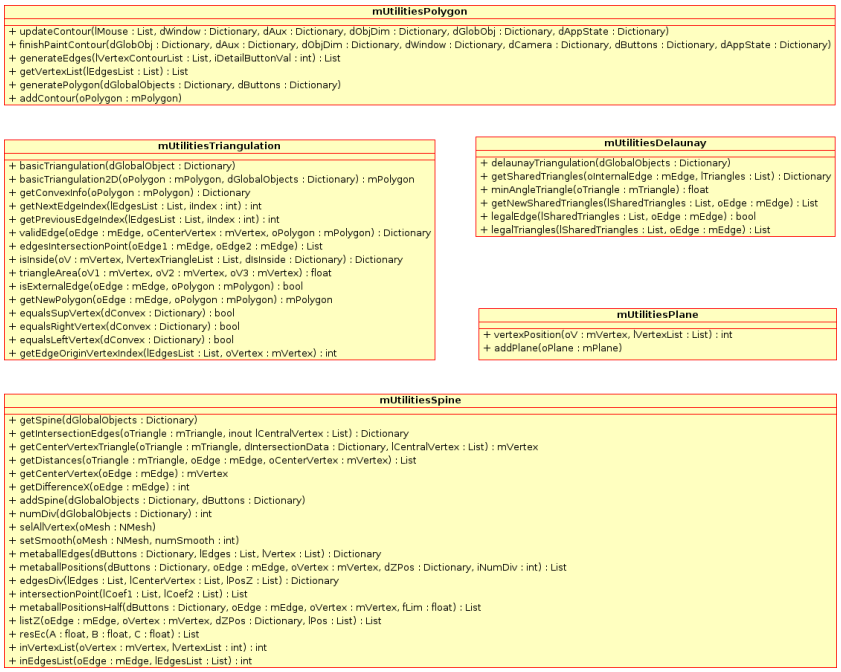

Figura C.2: Diagrama de utilidades.

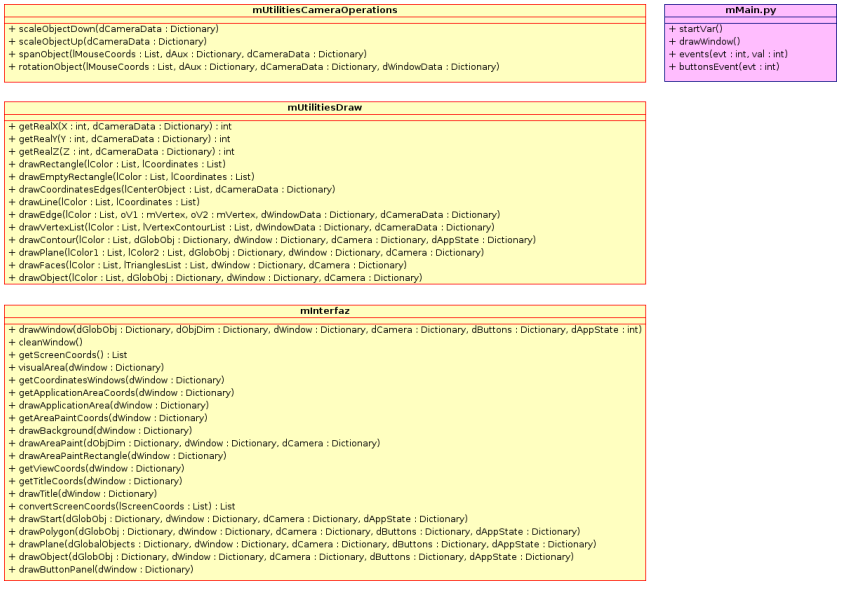

Figura C.3: Otras clases.

Anexo D

Diagramas de casos de uso.

Los diagramas de casos de uso describen de una manera rápida y clara la funcionalidad del sistema y la interacción del usuario con él.

En la figura D.1 se muestra el diagrama de casos de uso para la ejecución de la vista detallada de la aplicación.

Figura D.1: Ejecución vista detallada.

En la figura D.2, se muestra el diagrama de casos de uso para la ejecución de la vista automática.

El usuario dibuja un contorno y el sistema genera el objeto 3D asociado, permitiendo una

vez generado, la modificación de parámetros como el nivel de detalle y el suavizado para modificar la generación obtenida. También se permite añadir el contorno y el plano generado.

Figura D.2: Ejecución vista automática.

Anexo E

Diagramas de componentes.

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, como ficheros y paquetes.

En la figura E.1 se muestra el diagrama de componentes de los ficheros de la aplicación.

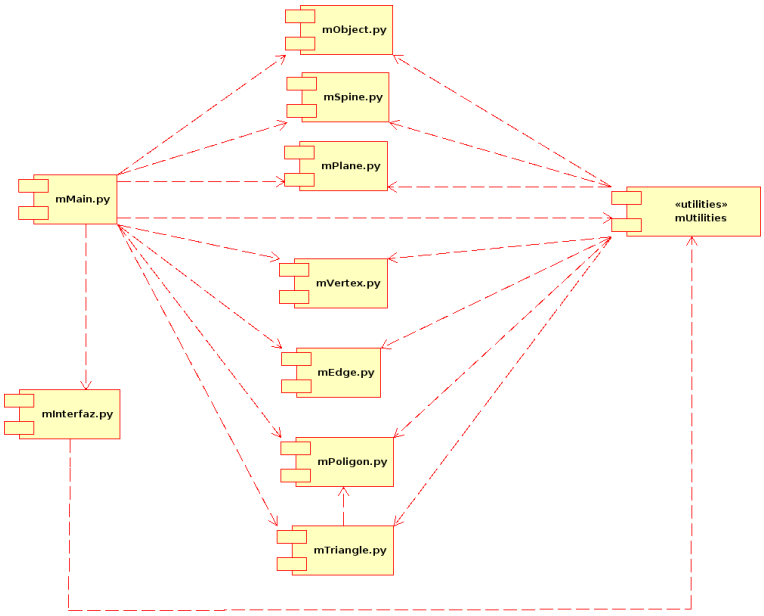

Figura E.1: Diagrama 1 de componentes.

En la figura E.2, se muestra el diagrama de componentes de los ficheros de la aplicación, con paquetes de Blender y Python.

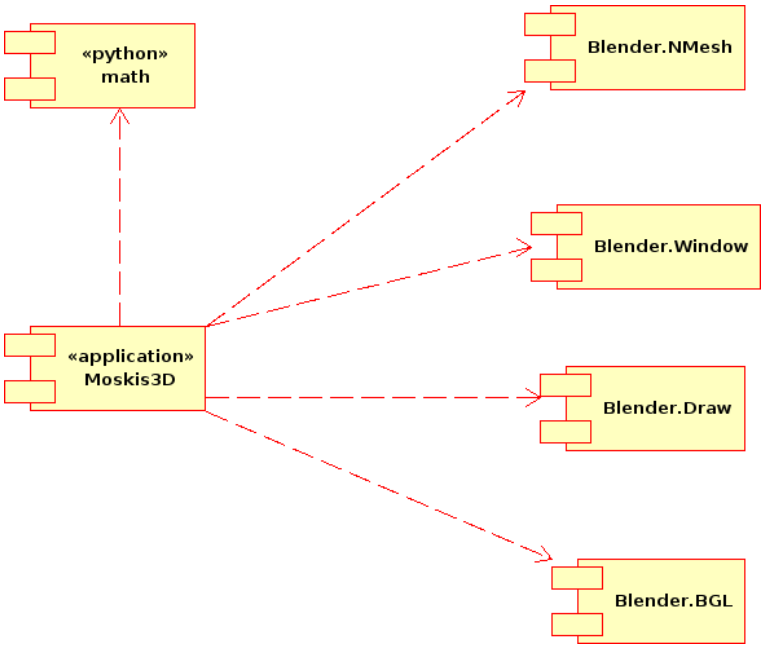

Figura E.2: Diagrama 2 de componentes.

Anexo F

CD adjunto.

El cd adjunto, tendrá la siguiente estructura de directorios:

- **Software**
  - **Blender**: en este directorio se encuentra la versión 2.41 de Blender para distintas plataformas.
  - **Código fuente MOSKIS 3D**: en este directorio se encuentra el código fuente de MOSKIS 3D.
- **Documentación**
  - **Memoria**: en este directorio se encuentra la memoria del proyecto, junto con las imágenes incorporadas en ella.
  - **Modelos resultados**: en este directorio se encuentran los resultados de los modelos realizados con MOSKIS 3D.

Dada la extensión del código fuente (más de 5000 líneas) no se imprimirá en este documento, puede encontrarse en el cd, en la ruta Software/Código fuente MOSKIS 3D.