Sistema de diseño gráfico para la creación de flores

Autores:
Roberto García Alonso
Alberto Maqueda Viñas
José María Martín Blázquez

Profesor Director:
Pedro Jesús Martín de la Calle

Proyecto de Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid

AGRADECIMIENTOS

En primer lugar queremos agradecer la ayuda y la confianza depositada en nosotros por el profesor director, Pedro Jesús Martín de la Calle, sin la cual la realización de este proyecto no hubiese sido posible.

También queremos agradecer la ayuda prestada por el profesor Antonio Gavilanes Franco que tan amablemente ha atendido nuestras dudas.

No nos queremos olvidar de agradecer el apoyo de nuestras familias y amigos, que nos han acompañado a lo largo de estos meses y nos han dado la fuerza necesaria para seguir adelante.

Por último, agradecer a todo aquel que dedique su tiempo a revisar este proyecto, deseándole que le sea útil y le sirva de ayuda.

AUTORIZACIÓN A LA UCM

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Roberto García Alonso

Alberto Maqueda Viñas

José María Martín Blázquez

ÍNDICE

ÍNDICE DE FIGURAS

RESUMEN

Flower Designer 3D es un entorno de diseño gráfico en tres dimensiones que permite crear y editar flores tanto estructural como geométricamente, generando previamente sus componentes a través de una interfaz muy intuitiva.

En el desarrollo del proyecto se han aplicado numerosas técnicas y algoritmos de informática gráfica y geometría computacional, tanto ya existentes como de elaboración propia.

El usuario plasmará con trazos la idea que tiene en la cabeza y el sistema realizará los cálculos y aproximaciones necesarias para obtener cada uno de los componentes. Para dar mayor realismo se pueden añadir colores o texturas pertinentes, así como la introducción de ruido o deformación del componente.

Los modelos creados podrán guardarse como imágenes, archivos de formato propio para su posterior edición, y archivos en formato estándar (ply) que permiten su reutilización desde otras aplicaciones.

Flower Designer 3D is a three dimensional graphic design environment that allows developing and editing flowers both structurally and geometrically, generating its own components through an intuitive interface.

Throughout the project development many techniques and algorithms from computers graphics and computational geometry have been applied, not only existing ones but also new proposals.

The user introduces some strokes of what he has in mind and the system will perform the necessary calculations and approximations to obtain each component. To provide greater realism, colours or textures can be applied as well as introducing component noise or deformation.

The created models will be able to be saved as images, own format files for further editing, and standard format files (ply) that allows its reuse in other applications.

LISTA DE PALABRAS CLAVE

Flor.

Pétalo.

Sépalo.

Estambre.

Receptáculo.

Pistilo.

Jogl.

Open GL.

Java.

Ply.

B – Splines.

1.- INTRODUCCIÓN

1.1.- OBJETIVOS

Desarrollo de un entorno de diseño gráfico de flores en tres dimensiones, proporcionando el mayor realismo posible. La tecnología usada incluye la aplicación de algoritmos y técnicas específicas de la informática gráfica y la geometría computacional, así como la investigación de otras propuestas nuevas para una posible futura aplicación de estas. Se pretende que el proceso sea lo más intuitivo posible y que con unos pocos trazos introducidos el programa muestre el resultado.

1.2.- ALCANCE

El proyecto se quiere alejar del desarrollo de grandes diseños centrándose en un diseño detallado de un conjunto de pequeños elementos cuya unión proporciona otro más complejo. Se permite crear una flor individualmente, no un conjunto de flores ni inflorescencias compuestas por éstas.

La idea de crear un elemento complejo a partir de otros más sencillos es posible gracias a la implementación de un editor geométrico y de un editor estructural. El primero nos permite diseñar componentes elementales, mientras que el segundo los une y los distribuye para componer una flor.

Basándose en esta aplicación y en sus algoritmos se pueden desarrollar otros que diseñen elementos compuestos a partir de otros más sencillos. También es posible el uso para desde otras aplicaciones de los archivos exportados.

1.3.- MOTIVACIÓN

1.3.1.- MOTIVACIÓN Y JUSTIFICACIÓN

La idea inicial fue la simulación de algún elemento de la naturaleza, ya fuese un árbol, un paisaje o finalmente el diseño de una flor, que es lo que se termino haciendo. La idea de realizar un paisaje completo era muy seductora y los resultados que se obtenían en algunos artículos eran muy impresionantes, ya fueran de paisajes naturales o simplemente la renderización de césped (ref. [6]). Sin embargo, los amplios conocimientos que requerían, tanto matemáticos como de informática gráfica, hicieron que optáramos por un modelo que también ofreciera resultados interesantes en cuanto a la calidad y al realismo. Una vez elegido el tema del editor de flores, el objetivo era que simplemente con unos cuantos trazos realizados, el sistema realizara los cálculos y aproximaciones convenientes a nivel geométrico para obtener un componente de aceptable calidad. Éste podría mejorarse añadiendo colores y texturas, y visualizase desde distintos puntos de vista. Finalmente, puesto que Open GL presenta algunas limitaciones como el tema de la iluminación que es básica, la exportación a un formato de archivo estándar (.ply) permitirá que otros programas o técnicas más avanzados puedan introducir mejoras en el componente creado, como es el caso del *ray-tracing*.

1.3.2.- ANÁLISIS DE LO EXISTENTE

Existen multitud de estudios y artículos sobre representación de la naturaleza y numerosas aplicaciones que usan las ideas propuestas, véase por ejemplo los videojuegos o películas de animación. En el caso del diseño y edición de flores, estas aplicaciones también están relacionadas con el diseño basado en trazos (strokes) y en la realización de bocetos. Una interesante herramienta para el diseño libre de objetos usando una interfaz de dibujo es *Teddy* (ref [7]). En cuanto al diseño de flores existen técnicas similares a la papiroflexia para diseñar orquídeas (ref [1] y [2]), que consisten en esbozar y posteriormente realizar pliegos con el ratón hasta obtener la flor deseada. Otras técnicas proponen en dibujar trazos o formas en dos dimensiones y posteriormente darles volumen con técnicas de contorno o extrusión (ref [3]). Más cercano a nuestro proyecto se encuentran los editores que se dedican a la construcción de manera jerárquica de una flor o planta. Unos lo hacen usando reglas similares a la construcción de gramáticas y aplicando la recursión para simular el crecimiento de la planta (ref. [8]) o combinando lo anterior con el dibujo de trazos y el uso de paralelepípedos para dotar de volumen al boceto (ref. [9]). Otras realizan bocetos sobre distintos planos que luego se refinan y se unen en una estructura única que compone la flor (ref. [5]). Finalmente, están los que realizan el proceso de forma jerárquica y por etapas, de manera que se diseñan los componentes por separado, uno detrás de otro, y luego se elige cómo y dónde colocarlos ya sea en una flor o en una inflorescencia (ref. [4]). Este último enfoque ha sido el que más nos ha influido en el desarrollo de nuestra aplicación.

Además en casi todos los artículos anteriores se trata el tema del ruido, ya sea para crearlo o suavizarlo por medio de distintas técnicas. Algunos artículos sobre ruido (ref. [10] y [11]).

1.3.3.- ¿QUÉ APORTA NUESTRO PROYECTO?

Nuestro proyecto ofrece una interfaz muy intuitiva para crear una flor siguiendo una serie de pasos. Primero se editan los componentes individualmente mediante la introducción de los trazos que delimitan su perfil (nivel geométrico), a partir de los cuales el sistema realizará los cálculos y aproximaciones necesarias para obtener un modelo de la calidad deseada. Para dar mayor realismo se pueden añadir colores o textura, así como la introducción de ruido o la deformación del componente. Para facilitar el diseño, el modelo en curso se visualiza desde distintos puntos de vista, y la herramienta permite el refinamiento de la componente cómodamente. Además la aplicación posee un editor estructural que sitúa los componentes en la flor, que también permite modificaciones en tiempo real. También es posible realizar una flor de forma rápida, usando los componentes disponibles en las galerías y ajustando hasta obtener el aspecto deseado.

Los resultados obtenidos son bastante realistas si tenemos en cuenta lo intuitivo del programa, el cual permite exportar las creaciones, ya sea capturando la pantalla en una foto, mediante un formato de archivo propio (.flw) o usando formato para la representación de las mallas (.ply). Esto último permite usar nuestras flores en otras aplicaciones de visualización más potentes.

1.4.- DEFINICIONES Y ACRÓNIMOS

1.4.1.- ACRÓNIMOS

Pixel	acrónimo del inglés picture element, "elemento de imagen"
Texel	texture pixel , pixel de textura.
PLY	Polygon File Format,
JOGL	OpenGL para java.
Open GL	Open Graphics Library, librería gráfica libre.

1.4.2.- DEFINICIONES

Pixel	Es la menor unidad homogénea en color que forma parte de una imagen.
Texel	La unidad mínima de una textura aplicada a una superficie.
PLY	Formato de archivos para definir objetos 3D mediante la descripción de los polígonos que los componen. Almacenando propiedades tales como el color y la transparencia, las superficie normales, coordenadas de textura.
JOGL	Versión de OpenGL para java.
Open GL	Librería para escribir aplicaciones gráficas en 2D y 3D.
.FLW	Fichero propio de la aplicación que almacena cada una de las componentes de la flor creadas con Flower Designer 3D. Almacena la información correspondiente a la estructura de la flor y propiedades de cada una de las componentes.
B-spline	Función matemática que define un tipo de curva.

1.5.- ORGANIZACIÓN DEL DOCUMENTO

El presente documento está organizado en capítulos agrupados por temáticas. Comienza con el capítulo *Planificación y organización* que explica de manera general el proceso que se ha llevado a cabo para el desarrollo del proyecto. El siguiente capítulo, *Funcionalidad,* enumera y explica cada una de las funciones propias de la aplicación. A continuación los capítulos *Diseño de la aplicación* y *Detalles algorítmicos* explican el proceso de diseño de cada uno de los elementos que componen una flor y, los algoritmos y herramientas usados en ello. En el capítulo de tecnologías se enumeran cada una de las tecnologías utilizadas en el desarrollo de la aplicación. Los siguientes cuatro capítulos, proporcionan detalles de nuestra aplicación, incluyendo una descripción de la jerarquía de clases y de las estructuras de datos empleadas, y del formato ply soportado. Por último se proporcionan una serie de apéndices, entre los que se encuentra una galería de Fotos con los diseños creados con la aplicación, otro dedicado a los archivos PLY y un apéndice de Matemáticas y algoritmia, que proporciona nociones básicas que hemos usado.

1.6.- NOCIONES BÁSICAS SOBRE LA ESTRUCTURA DE UNA FLOR

Debido a que el objetivo del programa es diseñar flores, en este apartado daremos una breve explicación de su estructura y de sus componentes, que será útil en lo sucesivo

La flor es una estructura compleja que pertenece al grupo de las angiospermas que representa una organización común a casi todos los miembros del grupo. No obstante, posee una gran diversidad en la morfología y fisiología de todas y cada una de las piezas que componen esta estructura. La flor está unida al tallo por un eje, denominado pedicelo, que se dilata en su parte superior para formar el **receptáculo** en el cual se insertan las diversas piezas florales. Éstas están divididas en dos grupos, las piezas estériles y las piezas fértiles.

Las piezas estériles se encuentran en la parte exterior de la flor y su principal función es la protección de las fértiles. Entre ellas se encuentran el cáliz y la corola. El cáliz está formado por **sépalos** y se encuentra en la parte más exterior. Generalmente son muy reducidos, apareciendo como dientes o crestas de color verde, aunque existe una gran variedad en su forma. La corola se encuentra en la parte interior rodeando a las piezas fértiles y está formada por los **pétalos**. Estos son de mayores tamaños y más coloreados que los sépalos pudiendo adoptar igualmente una gran variedad de formas.

Las piezas fértiles, se encuentran en la parte más interior de la flor y son las encargadas de la reproducción, por ello están protegidas por las estériles. La parte más exterior de estas piezas se denomina androceo cuyos componentes se llaman **estambres**, los cuales están compuestos por el filamento y la antera. El filamento se encarga unir el receptáculo con la antera. En la parte más interior de la flor se encuentra el gineceo también llamado **pistilo** en el cual se encuentra el ovario de la flor.

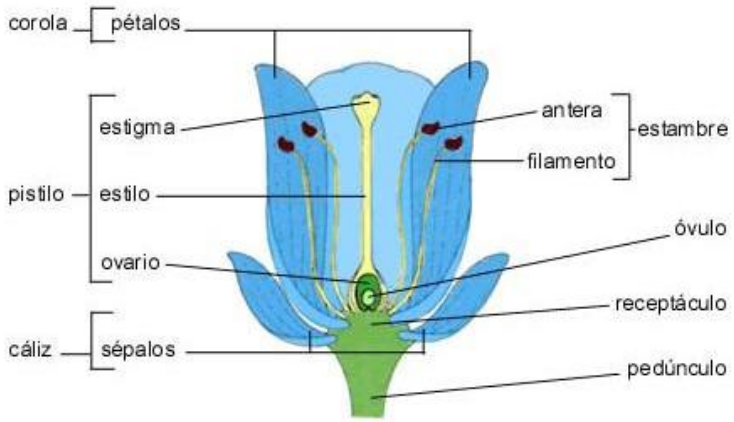


Ilustración 1 : Representación esquemática de las partes de una flor

2.- PLANIFICACIÓN Y ORGANIZACIÓN

En este capítulo se pretende explicar de manera muy general el proceso que se ha llevado a cabo para el desarrollo del proyecto. No se pretende dar una visión de planificación propia de la ingeniería del software o de gestión de proyectos puesto que para este proyecto creemos que tienen más importancia los aspectos algorítmicos y las técnicas que se han usado en la implementación de la herramienta. No obstante sí que creemos conveniente dar una breve explicación de las diferentes fases por la que ha pasado el proyecto.

2.1.- METODOLOGÍA UTILIZADA Y FASES DEL PROYECTO

2.1.1.- ESTUDIO PREVIO

Durante el primer mes aproximadamente y sobre una propuesta de partida por parte del profesor director, consultamos diferentes artículos relacionados con el tema de modelados de flores y plantas. El punto de partida fueron un par de artículos sobre la construcción de orquídeas mediante papiroflexia (ref. [1] y [2]). A partir de las referencias de estos artículos y otros artículos proporcionados por el profesor director, decidimos construir un editor estructural de flores mediante la edición de componentes con trazo libre, a los que se podrá agregar una textura, un editor de inflorescencias para componer varias plores en un mismo pedúnculo. Finalmente el editor de inflorescencias no se ha realizado por problemas de tiempo. Los artículos sobre los que basamos nuestras ideas tratan sobre el modelado basado en bocetos que se definen con trazos (ref. [3]), y el modelado interactivo de flores usando estructuras botánicas constantes (ref. [4]). También tratan sobre la introducción de trazos en distintos planos y refinamiento de los bocetos resultantes (ref. [5]).

2.1.2.- DISEÑO

En la fase de diseño se acordaron las funcionalidades principales y se empezó a implementar con la realización de prototipos individuales para la edición de componente de la flor. Las etapas fueron:

a. Configuración del entorno de desarrollo y de la herramienta JOGL.
b. Diseño en java del esqueleto 2D y 3D.
c. Definición de las funcionalidades básicas del editor estructural.
d. Diseño e implementación de los prototipos de dibujo de receptáculos, estambres y hojas.

2.1.3.- IMPLEMENTACIÓN

Durante esta fase se realizó el desarrollo e implementación de la aplicación. Las fases de esta etapa fueron:

a. Implementación del prototipo del pistilo.
b. Diseño del editor estructural, implementación del *canvas* de los círculos e integración del receptáculo en este último.
c. Deformación de la hoja longitudinal y transversalmente, así como el aumento de su tamaño.

d. Introducción de los visores de cámara desde distintos ángulos a los prototipos de creación de las componentes de la flor.
e. Creación de la ventana de visualización de la flor.
f. Creación de las denticiones de la hoja, introduciendo algoritmos de ruido.
g. Diseño y desarrollo del algoritmo de correspondencia entre 2D y 3D para que los elementos introducidos en la pantalla de círculos en 2D se reflejasen en el receptáculo 3D
h. Aplicación de texturas sobre los componentes.
i. Integración de los prototipos en una misma aplicación organizada en pestañas.
j. Descarte del editor de inflorescencias. En su lugar se refinará la aplicación existente con la creación de un instalador, se permitirá exportar los diseños creados en formato de archivo propio y en formato .ply, y se posibilitará la captura en formato fotográfico de la imagen de cada componente.
k. Creación de una flor con componentes de prueba.
l. Incorporación de galerías útiles para elegir con rapidez los componentes de la flor.
m. Implementación de los menús.
n. Consolidación del sistema de ficheros: exportación a archivos .ply, la exportación/importación en formato de archivo propio (.flw), y la opción de capturar pantalla.

2.1.4.- PRUEBAS Y MEJORAS

Durante esta fase se eliminaron los fallos que se han ido encontrando, se elaboró la documentación y se creó el CD de instalación de la aplicación. Las fases de esta etapa fueron:

a. Se repararon errores y se mejoraron aspectos visuales (creación de una ventana y archivo de configuración para que el usuario personalice la creación de componentes, pantalla de inicio…).
b. Se creó el instalador de la aplicación.
c. Se elaboró la documentación del proyecto con un manual de usuario.
d. Se creó el CD de la aplicación.

3.-FUNCIONALIDAD

La aplicación permite la creación de una flor, pudiendo realizar el diseño de su estructura, así como el de cada uno de sus componentes.

a. **Diseño de una flor**: Nos posibilita diseñar la estructura de la flor, es decir, podemos elegir el número de componentes que queremos que tenga, el tipo (sépalo, pétalo, estambre o pistilo) y el tamaño. El editor de flores nos permite decidir en qué lugar del receptáculo van a estar cada uno de los distintos componentes. Pudiendo colocar todos los de un mismo tipo a la vez, o bien de forma individual.

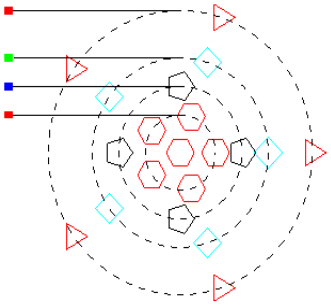


Ilustración 2 : Editor estructural

b. **Diseño de estambres**: Se puede diseñar un estambre dibujando dos trazos, uno correspondiente al filamento y otro a la antera.
c. **Diseño de pistilos**: Se puede diseñar un pistilo dibujando el trazo de su perfil.
d. **Diseño de pétalos y sépalos:** Podemos diseñar un pétalo y un sépalo dibujando tres trazos, dos correspondientes a los laterales y otro a la vena central. Una vez dibujado el componente podemos curvarlo longitudinal y transversalmente mediante trazos sucesivos.
e. **Denticiones:** Existe la posibilidad de añadir denticiones a los sépalos, pudiendo variar su tamaño y el número de éstas. Además, se puede elegir entre dos algoritmos para generarlas.
f. **Diseño de receptáculos:** Podemos diseñar un receptáculo dibujando el trazo de su perfil.
g. **Texturas:** La aplicación nos permite añadir texturas a cada uno de los distintos componentes para darles mayor realismo. Para ello se dispone una galería de texturas que admite la incorporación de otras nuevas.
h. **Blend:** La opción de *blend* permite fusionar la textura con el color del componente para obtener mayor realismo.
i. **Interacción de la escena:** Dispone de una interfaz que nos proporciona comodidad y facilidad a la hora de realizar nuestros diseños. Esta nos ofrece:
    a. Movimiento libre de la escena mediante teclado. Este incluye rotación, translación y escalado.
    b. Tres visores adicionales desde diferentes posiciones configurables para tener una mayor percepción de la escena.
    c. Movimiento de la escena mediante ratón.
    d. Posibilidad de elegir entre cámaras de visión lateral, frontal, cenital o en 3D para la pantalla principal.
j. **Galería:** La aplicación ofrece la posibilidad de guardar los componentes creados con el editor en una galería, para luego poder utilizarlos en la construcción de otras flores.

k. **Exportación .PLY:** Permite la exportación de los elementos generados mediante archivos PLY (ver sección 5.17) para poder reutilizarlos en otras aplicaciones posteriormente.
l. **Exportación e importación .FLW:** Permite la exportación e importación de los elementos generados mediante un formato de archivo propio con extensión .FLW (ver sección 7.2).
m. **Captura de pantalla:** La aplicación ofrece la posibilidad de capturar la imagen de cada uno de los elementos generados y guardarla en disco.
n. **Representación de las mallas:** Posibilidad de elegir la representación de las mallas entre distintitos modos, polígonos, líneas o puntos.
o. **Color:** Permite elegir el color de las mallas creadas.
p. **Configuración:** Posibilidad de cambiar la configuración de la aplicación. Podemos cambiar:
    a. Los visores.
    b. El número de radianes y pasos de revolución.
    c. Los grosores.
    d. El número de lados de los polígonos de los estambres.
    e. El número de puntos de las b-splines.
    f. El número de puntos de filtrado.
q. **Menú:** La barra de menú está personalizada para cada una de las vistas de la aplicación.

4.- DISEÑO DE LA APLICACIÓN

4.1.- DISEÑO DE LA FLOR

El diseño de una flor consiste en ir agregando componentes (pistilos, estambres, pétalos y sépalos) al receptáculo. Para ello, disponemos de cuatro circunferencias concéntricas en las que vamos a ir insertando los distintos componentes. El proceso se gestiona rellenando la estructura de datos DataFlowers (ver sección 10.1). De dentro hacia fuera cada una de estas circunferencias representa a los pistilos, estambres, pétalos y sépalos respectivamente. Los componentes se van colocando a lo largo de su circunferencia formando un polígono imaginario (A) de tantos lados como elementos hemos introducido hasta el momento (ver sección 5.3).

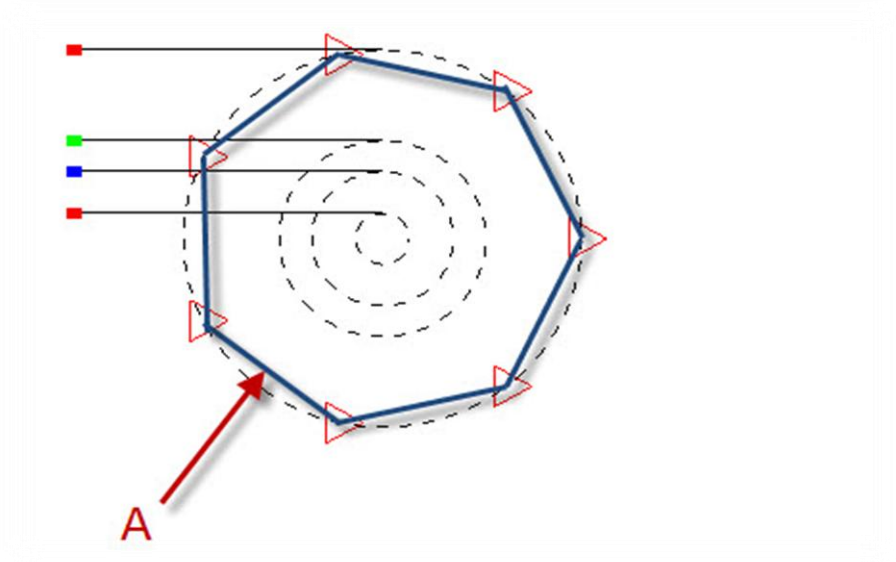


Ilustración 3: Distribución uniforme de los componentes a lo largo de la circunferencia

Podemos modificar la posición de cualquier elemento pinchando sobre él y arrastrándolo hasta el lugar elegido (B), así como el número de elementos por circunferencia. Internamente esto supone cambiar en DataFlowers (que contiene la posición de cada elemento) las antiguas coordenadas del elemento por las nuevas. También es posible cambiar el tamaño de los componentes en una escala de 1 a 10.

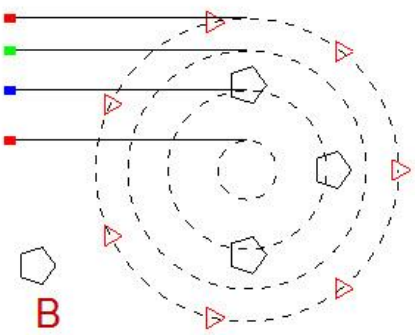


Ilustración 4: Posición del elemento B modificada manualmente

Una vez representada la flor en 2D, es necesario pasarla a 3D. Este paso no es trivial y nos ha supuesto un problema, que se soluciona gracias al desarrollo de un algoritmo propio (ver sección 5.15) que calcula la posición de cada componente en 3D y su inclinación. Para poder tener una mejor visión de cómo está quedando la flor, justo debajo de las circunferencias existe una representación esquematizada en 3D de su estado.

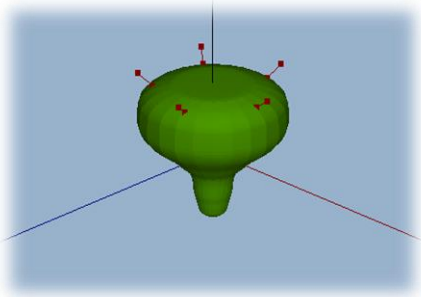


Ilustración 5: Distribución en el receptáculo

Una vez que tenemos las posiciones de inserción en el receptáculo de forma esquemática, procedemos a elegir los componentes de las galerías. Para ello pulsamos *Aceptar* y pasamos a la pantalla que se encarga de seleccionar los componentes. Esta pantalla está formada por dos imágenes, una con el receptáculo, que inicialmente estará vacio y otra con las circunferencias que habíamos editado anteriormente. Haciendo doble-clic con el botón izquierdo del ratón sobre alguno de los componentes de las circunferencias se nos abrirá una pantalla con la galería correspondiente para elegir un elemento. Una vez elegido este se insertará en la posición que le corresponde en el receptáculo (ver sección 5.18). La inclinación con la que aparece en el receptáculo depende de su posición en el eje Y, y se calcula mediante el algoritmo de paso de 2D a 3D anteriormente mencionado.

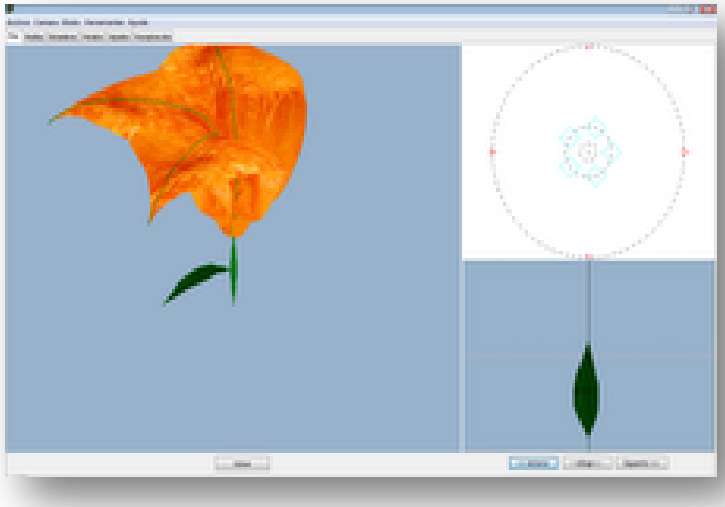


Ilustración 6: Inserción de componentes

Podemos elegir todos los componentes del mismo tipo a la vez si hacemos doble-clic con el botón derecho del ratón dentro de la circunferencia correspondiente. El resto

del proceso para elegir la geometría del componente usado es similar al recién descrito.

El diseño de la flor estará terminado cuando tengamos todos los componentes insertados en el receptáculo.

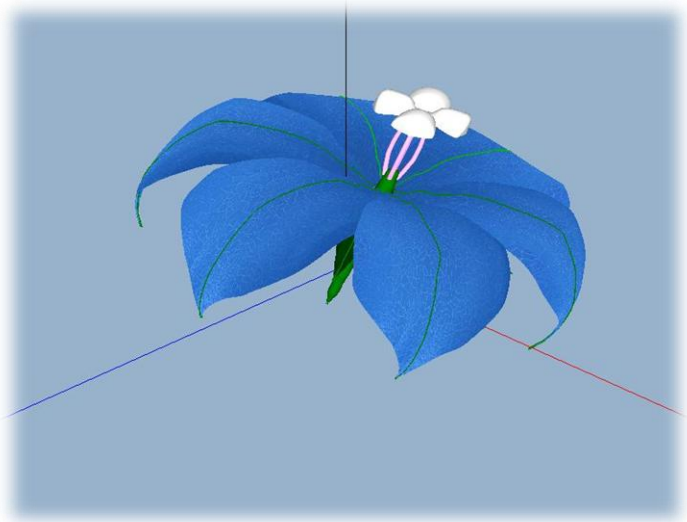


Ilustración 7: Resultado de la aplicación

4.2.- DISEÑO DE UN ESTAMBRE.

Para diseñar un estambre el usuario debe introducir dos trazos en el lienzo. El primero se corresponderá con el filamento y el segundo con la antera.

El primer trazo debe introducirse de abajo a arriba y una vez levantado el ratón se convierte en una B-spline, la cual podrá modificarse antes de introducir el segundo trazo. Este segundo trazo debe introducirse de izquierda a derecha y una vez introducido y transformado a B-Spline también podrá modificarse antes de perfilar para obtener el estambre resultante. Para ver cómo se realiza internamente la transformación del trazo a B-Spline ver sección 5.2.

El proceso de creación del estambre consistirá en crear una malla de contorno alrededor de la base (ver sección 5.6) y colocar la cabeza en la base realizando una media revolución con la curva B-spline (ver sección 5.7).

Una vez creado el estambre, se puede dar color a la base y a la cabeza independientemente, así como asignar una textura a la malla de la base. También podrá exportarse a archivo PLY, guardarlo en la galería, guardarlo con formato .flw, o realizar una captura en jpg.

4.4.- DISEÑO DE UN RECEPTÁCULO

Para el diseño del receptáculo es necesario introducir un trazo de arriba abajo y preferiblemente a la derecha del eje Y, aunque en la mayoría de los casos la herramienta soportará que el trazo lo traspase.

Una vez introducido el trazo, éste automáticamente se transforma a una curva B-Spline, la cual puede modificarse antes de perfilar el receptáculo. El receptáculo

consiste en una figura por revolución y se construye revolucionando la curva sobre el eje Y (ver sección 5.5).

Una vez creado el receptáculo se puede asignar el color deseado así como aplicarle diferentes texturas. También podrá exportarse a archivo PLY, guardarlo en la galería, guardarlo con formato .flw, o realizar una captura en jpg.

4.3.- DISEÑO DE UN PISTILO

El proceso de creación de un pistilo sigue los mismos pasos que el del receptáculo puesto que ambos se realizan por revolución.

4.5.- DISEÑO DE UNA HOJA

Para crear una hoja el usuario introduce tres trazos con el ratón en la interfaz, los trazos serán de arriba a abajo, siendo el primero el límite izquierdo de la hoja, el segundo representará la vena de la hoja, y el tercero será el límite del extremo derecho de la hoja. Estos trazos son transformados en b-splines,

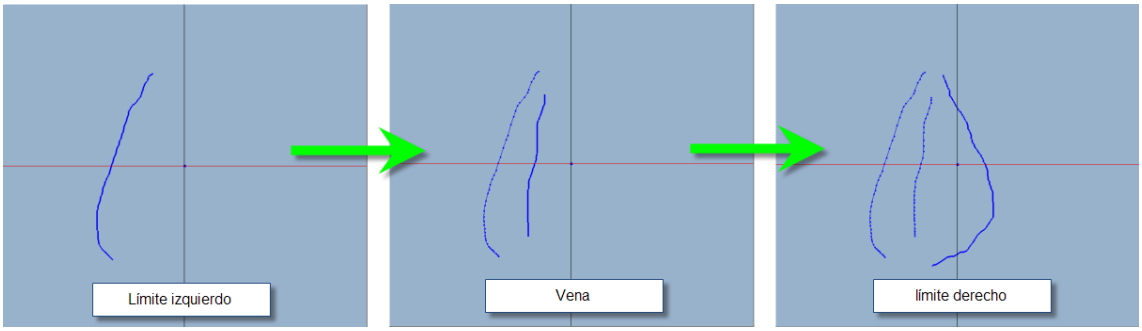


Ilustración 8: Trazos introducidos por el usuario para el diseño de una hoja

Tras introducir los tres trazos se procede a calcular el contorno de la hoja, mediante la unión de las curvas que delimitan los extremos. Para realizar esta acción se crean rectas de regresión lineal (ver sección A3.3) en los extremos de ambas b-splines. Estas rectas se calculan con el 20% de los puntos en los extremos de la curva, tanto superior como inferiormente, de forma que cuantos más puntos se asignen a la b-spline más fiel será la imagen al perfil introducido por el usuario.

Una vez calculadas las cuatro rectas de regresión, se realiza el algoritmo de corte entre las rectas superiores para determinar el extremo superior correspondiente a la punta de la hoja. Lo mismo se realiza sobre las rectas inferiores para determinar el otro extremo de la hoja, que servirá de unión con el receptáculo.

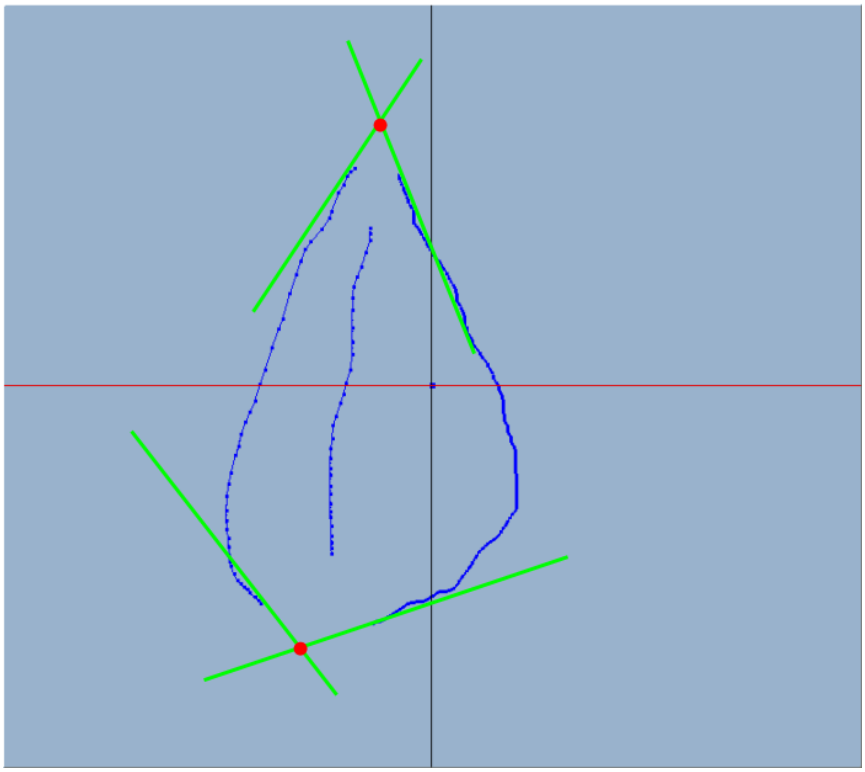


Ilustración 9: Puntos de corte que indican los extremos de la hoja y delimitación de su área

Tras delimitar por completo el área que ocupa la hoja, se pasa a unir el trazo que define a la vena con los puntos extremos de la hoja, para ello se une cada uno de los puntos extremos de la vena con su correspondiente extremo de la hoja mediante un segmento delimitado por esos dos puntos.

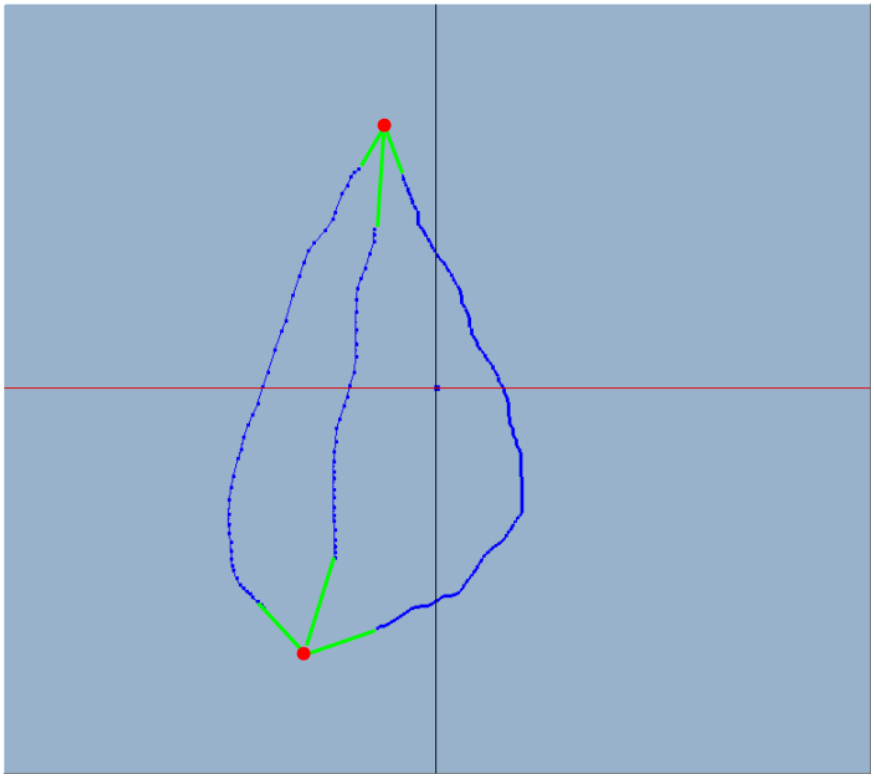


Ilustración 10: Unión de la vena con los extremos de la hoja

Una vez definida la hoja, se cambia de sistema de coordenadas mediante una traslación 2D que sitúa el eje Y sobre la recta que une los dos puntos extremos de la hoja y el eje X perpendicular (en 2D) al eje Y pasando por el extremo inferior. Esta operación se realiza aplicando el algoritmo de cambio de sistema de coordenadas obteniendo como resultado la hoja centrada en la imagen (ver sección 5.11).

4.6.- DISEÑO DE UN PÉTALO

El diseño de un pétalo se realiza a partir de la malla de una hoja a la que se le añade una cierta curvatura tanto longitudinal como transversalmente (ver sección 5.9) mediante trazos introducidos con el ratón. Una vez obtenida la curvatura deseada se puede cambiar el color y añadir texturas dando mayor realismo al resultado

4.7.- DISEÑO DE UN SÉPALO

El diseño de un sépalo sigue exactamente el mismo procedimiento que el de un pétalo. Además hemos incorporado la posibilidad de añadirle ruido en el contorno de la hoja, de forma que se pueden crear denticiones. Para ello se aplican dos algoritmos que deforman la malla (ver sección 5.10).

5.- DETALLES ALGORÍTMICOS

5.1.- DIAGRAMAS DE REQUISITOS ALGORÍTMICOS

Para entender mejor este capítulo, comenzamos presentando una serie de diagramas que incluyen los ingredientes necesarios para crear cada componente. De esta manera si el lector quiere consultar un componente en concreto, puede acudir a la sección correspondiente para ver los fundamentos involucrados en su creación.

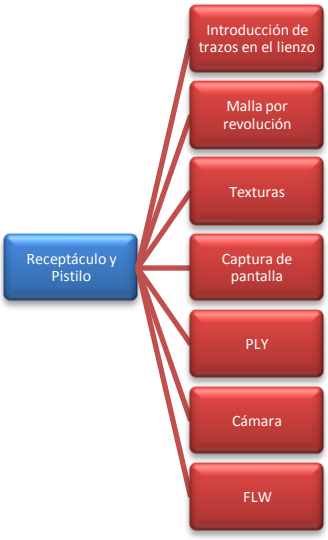


Ilustración 11: Diagrama de dependencias del receptáculo y del pistilo

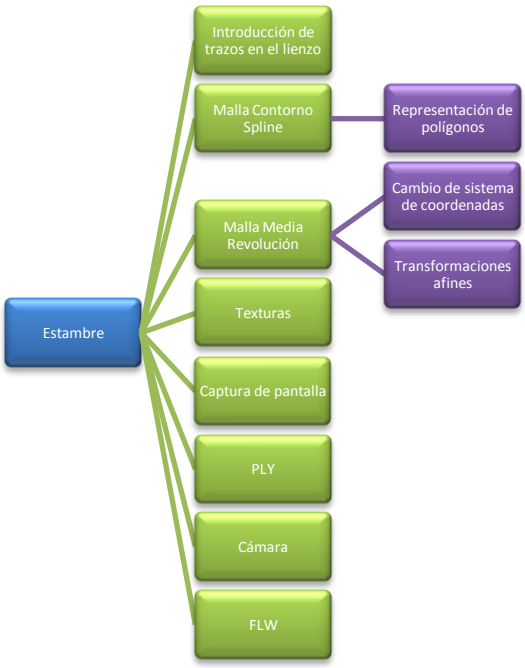


Ilustración 12: Diagrama de dependencias del estambre

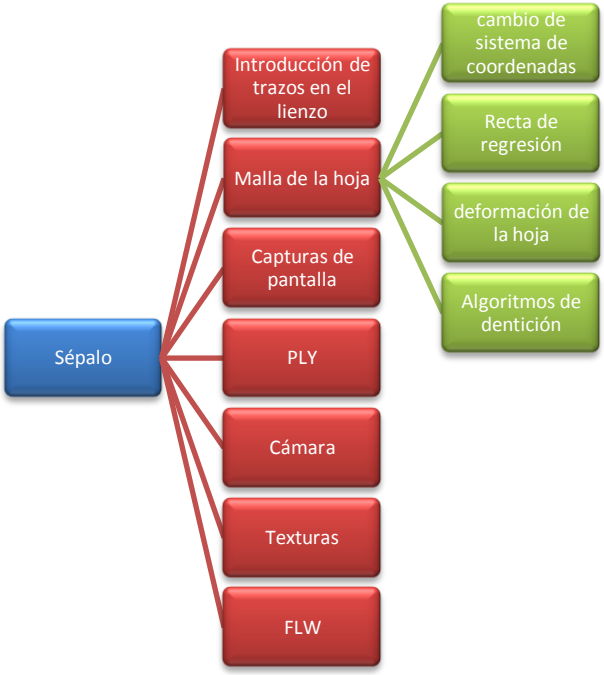


Ilustración 13: Diagrama de dependencias del sépalo

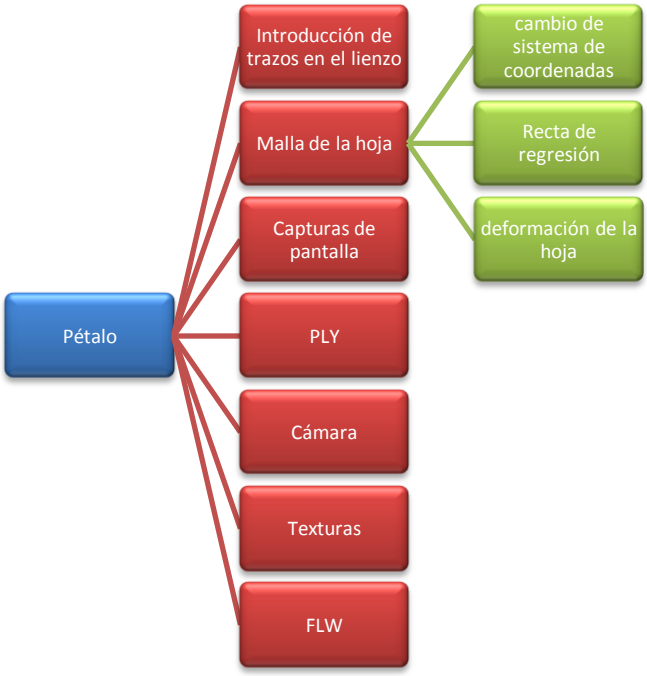


Ilustración 14: Diagrama de dependencias del pétalo

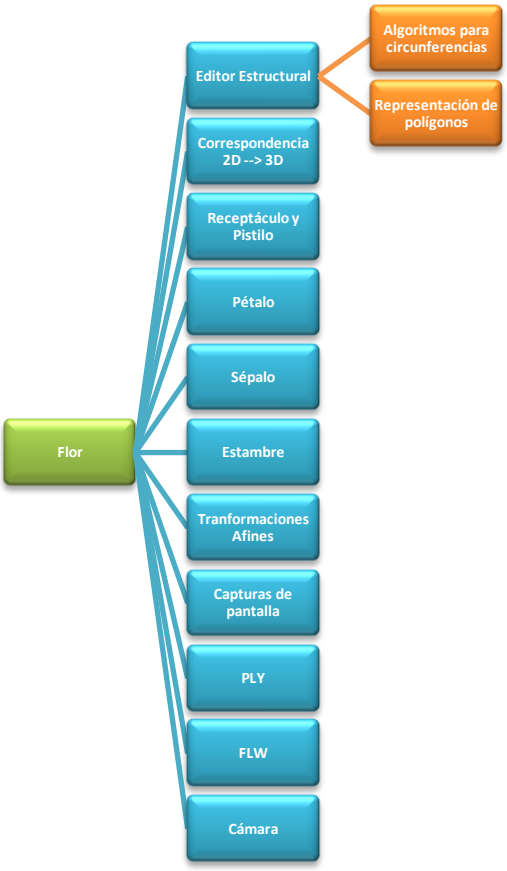


Ilustración 15: Diagrama de dependencias de la flor

5.2.- INTRODUCCIÓN DE TRAZOS EN EL LIENZO

En las pantallas de edición de los componentes de una flor es necesario introducir al menos un trazo mediante el ratón. Este trazo es libre, el usuario pulsa y arrastra el ratón hasta conseguir el trazo (stroke) que desea.

Para capturar el trazo introducido, se capturan todos los puntos por los que ha pasado el ratón, de manera que de cada cierto número de estos puntos (valor configurable) se selecciona uno para formar parte de los puntos de control de una curva B-spline. Así se suaviza el trazo respetando el original al máximo y se aprovechan las ventajas de las B-splines en computación gráfica (ver sección A3.1).

En la siguiente figura podemos ver el trazo introducido por el usuario y como cada ciertos puntos se toma uno para formar parte de los puntos de control de la spline. La spline resultante, que apenas difiere del trazo introducido, está lista para interactuar con ella.

Ilustración 16: Selección de los puntos de control a partir de un trazo

Una vez introducido un trazo y la correspondiente transformación automática a b-spline, ésta se puede modificar pinchando y arrastrando sobre los puntos de control hasta obtener la curva deseada.

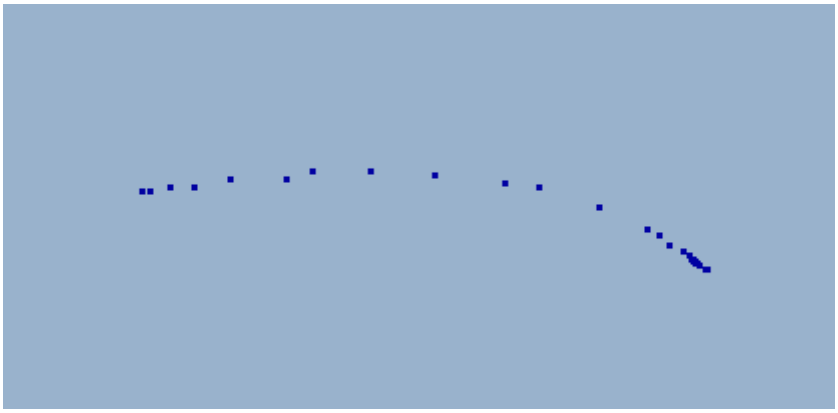


Ilustración 17: Trazo introducido en el lienzo

En la siguiente figura vemos cómo se han recogido puntos de control cada cierto intervalo (depende también de la velocidad del ratón), y se ha construido automáticamente la curva b-spline.

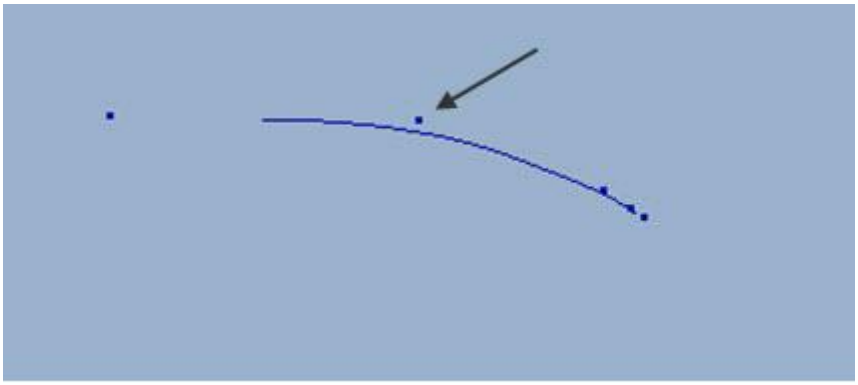


Ilustración 18: Obtención de los puntos de control

Al modificar un punto de control (indicado con una flecha en la figura superior) con el ratón, la curva resultante cambia.

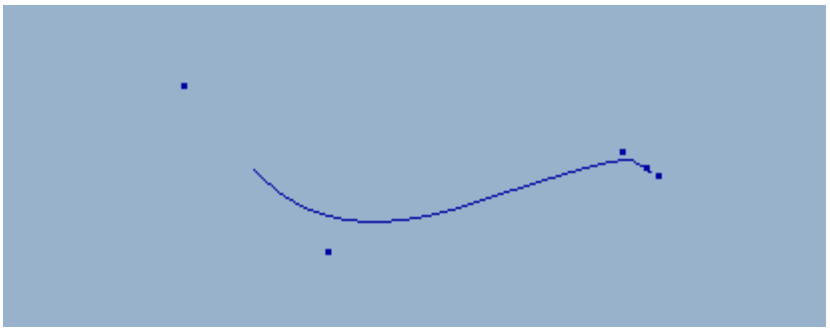


Ilustración 19: Obtención de la spline

Observación: Cuanto mayor sea el cuidado y precisión con que se introduce el trazo (stroke), mayor será el número de puntos de control de la curva y más fiel será al trazo original. Una modificación de un punto de control en este caso provocará un saliente en la curva de manera que su perfil sufrirá un cambio más brusco.

5.3.- REPRESENTACIÓN DE POLÍGONOS

Un polígono regular es un polígono en el que todos los lados tienen la misma longitud y todos los ángulos interiores son de la misma medida (ver sección A3.5). Todas las mallas usadas para crear los componentes se basan en polígonos regulares, así como todos los polígonos que aparecen en la aplicación. También se utilizan para la distribución de componentes en el editor estructural.

Existen dos formas de construir polígonos de manera computacionalmente rápida:

1. Obteniendo los puntos que forman el polígono y uniéndolos. Para ello debemos conocer el radio del polígono y las coordenadas del centro. Dividimos 360/nLados y obtenemos el incremento de ángulo para obtener cada punto. De esta forma empezando con un ángulo de 0 grados y sumando el incremento al ángulo acumulado podemos obtener el i-ésimo punto como sigue:

   X = radio * coseno (ángulo acumulado) + coordenada x del centro

   Y= radio*seno (ángulo acumulado) + coordenada y del centro.

2. Dibujando el polígono mediante movimiento relativo. Esta técnica permite dibujar además de polígonos otras figuras como espirales, rosetas, poli-líneas y en general cualquier figura que repita un cierto patrón o motivo. Esta opción es la que hemos elegido y es la que detallamos a continuación.

5.3.1.- DIBUJO DE POLÍGONOS MEDIANTE MOVIMIENTO RELATIVO

La elección de esta técnica se debe a su mayor versatilidad para dibujar sobre cualquier plano del espacio, basándose en el movimiento relativo de un punto en una dirección determinada. Esta técnica conocida como “Turtle graphics” es conceptualmente similar al dibujo con un lápiz, dejando un trazo desde el punto desde donde se parte avanzando en la dirección del lápiz una longitud dada. El paquete de código llamado “lápiz” así como la clase UtilPoligono del paquete “utils” sirven para implementar estos conceptos.

En la siguiente figura el punto que dibuja, al que llamaremos “tortuga”(turtle), está posicionado en el punto origen “old world CP” preparado para avanzar en una dirección “CD” que se especifica con un número de grados con respecto al eje x. Avanzando la tortuga una distancia “dist” (en el eje x se moverá una distanciaX = dist * coseno(π * CD/180) y en el eje y una distanciaY = dist * seno(π * CD/180)), llega al punto de destino “new world CP” dejando la traza del segmento dibujado.

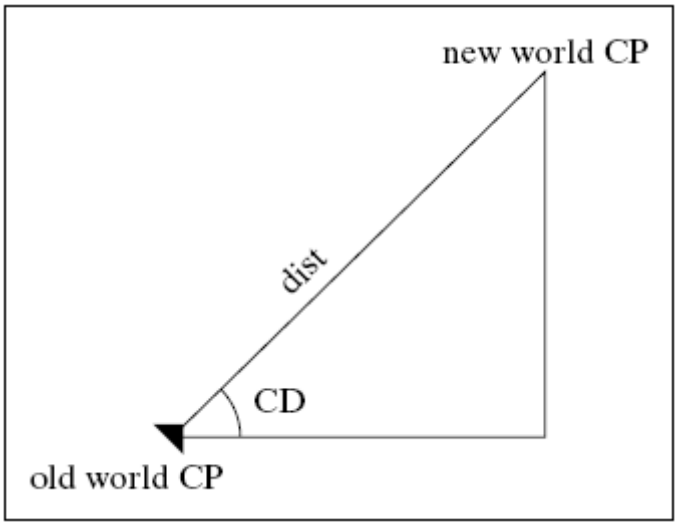


Ilustración 20: Movimiento de la tortuga

Las operaciones básicas de este lápiz son:

- Girar n grados: girar la tortuga n grados en sentido contrario a las agujas del reloj. Este giro es un cambio relativo en la dirección. No se especifica una dirección, sólo un cambio en la dirección.
- Avanzar una longitud: consiste en mover la tortuga en línea recta una cierta longitud desde su posición actual usando su dirección. Se puede indicar si la tortuga deja trazo porque está dibujando, o no lo deja y simplemente está avanzando su posición.

El proceso para dibujar el polígono regular es el siguiente:

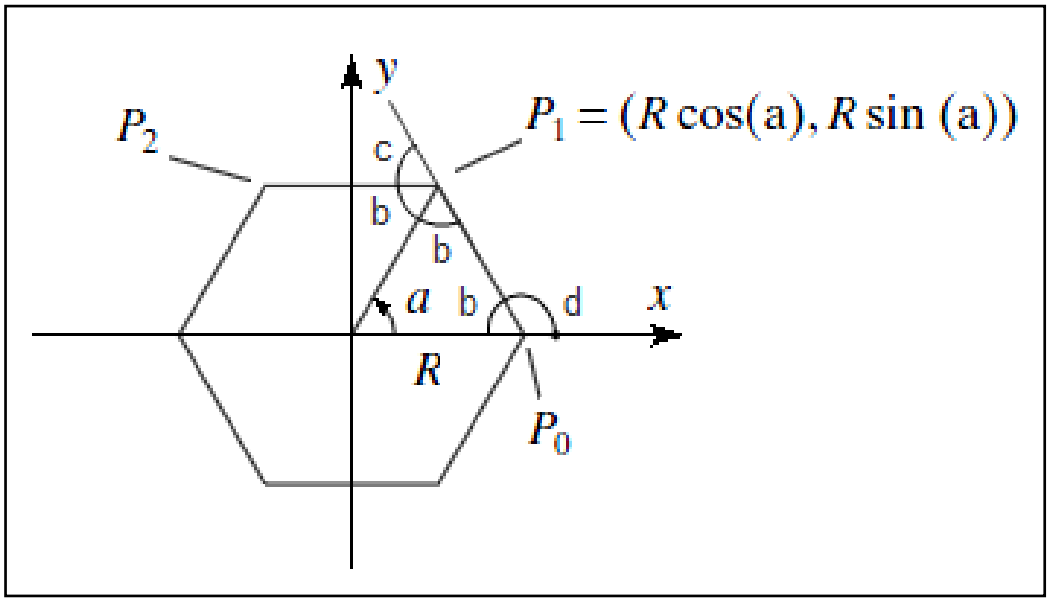


Ilustración 21: Ángulos necesarios para dibujar un polígono

El ángulo a lo obtenemos como a = 360/número de lados.

El ángulo b lo obtenemos como b = (180 – a)/2

El ángulo c lo obtenemos como c = 180 – 2*b = a, ya que c + b + b = 180

El ángulo d lo obtenemos como d = 180 – b, ya que d + b = 180;

Una vez obtenidos los ángulos, colocamos el lápiz (tortuga) en el punto P0 de la figura teniendo como ángulo inicial el ángulo d y sumando el radio R a la coordenada x del centro. A continuación avanzamos una longitud igual al lado del polígono para obtener el primero de los lados. Posteriormente giramos un ángulo c para dirigir el lápiz hacia la dirección del siguiente lado. Este proceso se repite tantas veces como lados tenga el polígono que queremos dibujar.

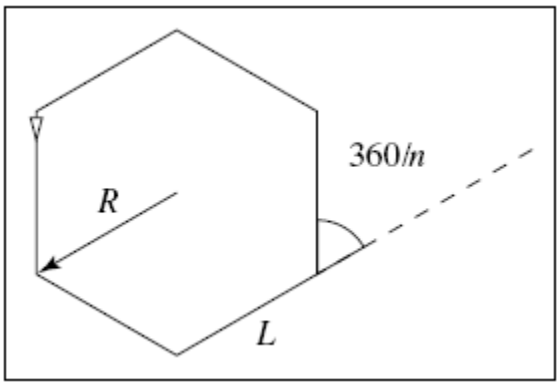


Ilustración 22: Detalle del proceso para dibujar un polígono por medio del lápiz (tortuga)

5.4.- MALLAS. REPRESENTACIÓN DE LOS COMPONENTES EN 3D

5.4.1.- FUNDAMENTOS TEÓRICOS

Las mallas poligonales son conjuntos de polígonos que determinan la superficie de un objeto. Son una forma "estándar" de representación. Entre sus ventajas destacan que son fáciles de manejar y que permiten calcular con comodidad su posición relativa con respecto a un punto o aun rayo.

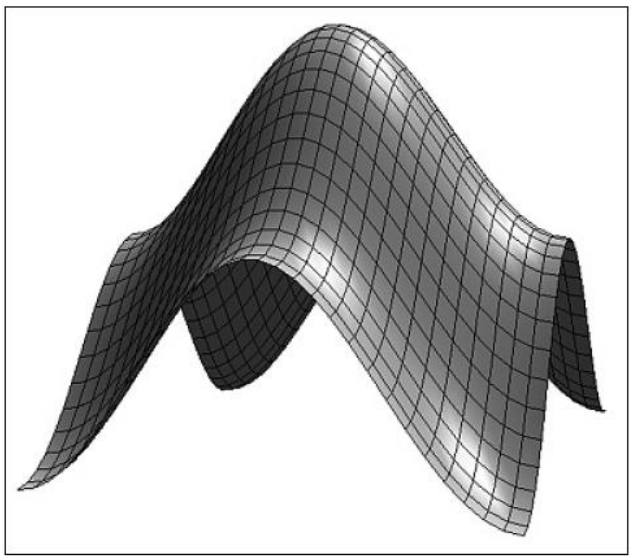


Ilustración 23: Vista de una malla

Información que debe incluir una malla:

- Orientación: indica cuál es el interior, se define mediante vectores normales (pueden ser por caras o por vértices).
- Información geométrica: vértices.
- Información topológica: caras.

Tipos de las mallas:

- Malla sólida: define un volumen.
- Malla conexa: existe un camino de aristas entre dos vértices cualesquiera.
- Malla simple: sólida y sin agujeros.
- Malla plana: formada por polígonos planos.
- Malla convexa: si el volumen que encierra es convexo.

Una malla poligonal viene definida por una lista de polígonos, más la información de hacia dónde se orienta cada uno de ellos. Esta información direccional viene dada por el **vector normal** al plano de la cara y se usa en el proceso de sombreado para determinar cuánta luz refleja la cara.

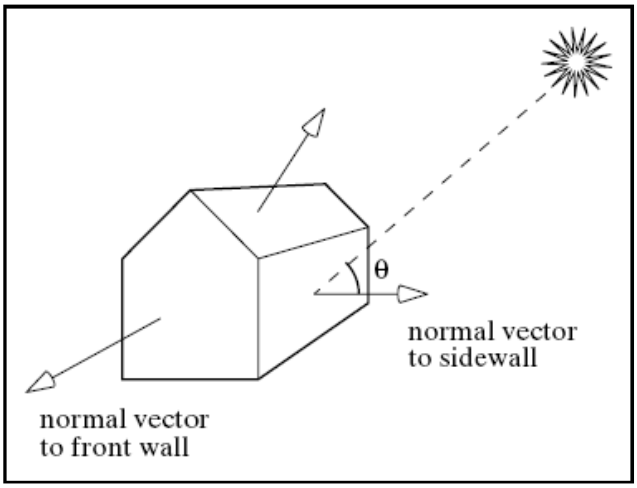


Ilustración 24: La dirección de la normal de una cara determina su iluminación

El cálculo de las normales se realiza a partir del **método de Newell**:

$$m_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i + z_{next(i)})$$

$$m_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i + x_{next(i)})$$

$$m_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y + y_{next(i)})$$

Donde N es el número de vértices de la cara, (x_i, y_i, z_i) es la posición del i-ésimo vértice y next(i) = (i+1) mod n, es el índice del siguiente vértice de la cara después del i-ésimo vértice.

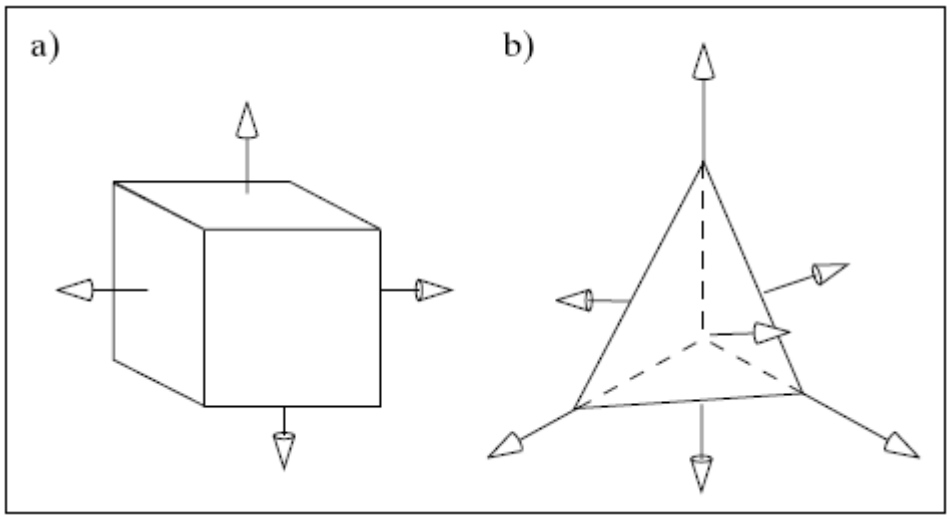


Ilustración 25: Método de Newell

El método de Newell nos permite obtener las normales de las caras de una malla.

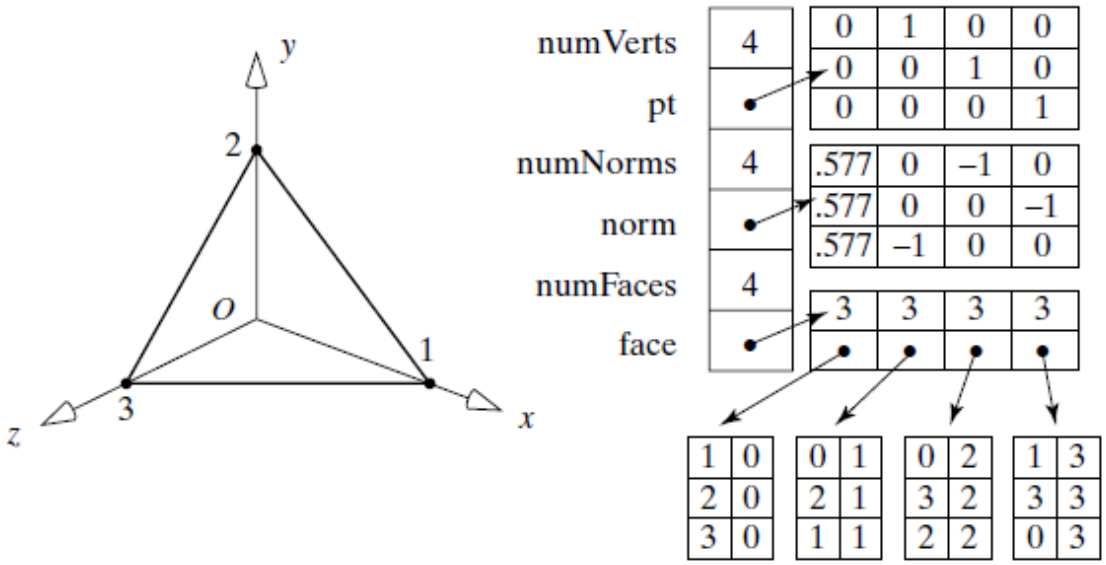
5.4.2.- APLICACIÓN AL PROYECTO:

Para definir mallas usamos las siguientes clases:

- PuntoVector: cuyos objetos son puntos/vectores en tres dimensiones.
- Cara: cuyos objetos contienen información del número de vértices que forman la cara, así como de los índices de los vértices y normales que la forman, esto último mediante un array dinámico de elementos de la clase VerticeNormal.
- VerticeNormal: cuyos objetos son pares de la forma (índice de vértice, índice de normal para ese vértice).
- Malla: cuyos objetos contienen información del número de vértices, número de normales y número de caras, junto con los respectivos arrays dinámicos de vértices, de normales y de caras. Asimismo define el método que permite dibujar una malla. Los diferentes tipos de mallas usados para representar componentes 3D extenderán esta clase y construirán la malla correspondiente.

Los vértices de una cara de la malla se recorren en sentido contrario a las agujas del reloj cuando la cara se mira desde el exterior del objeto.

Ejemplo: Vamos a representar un tetraedro con las estructuras y clases arriba mencionadas. El tetraedro es un poliedro y, por tanto, una malla conexa, sólida y plana. Lo suponemos centrado en el (0, 0, 0) y con lados de longitud 1.



Observaciones: Las caras de las mallas utilizadas en el proyecto son triangulares.

5.5.- MALLA POR REVOLUCIÓN TOMANDO COMO PERFIL UNA B-SPLINE.

El primer paso es almacenar N puntos de control P_1, P_2,..., P_n a partir de un trazo introducido por el usuario como se explica en el apartado de *Fundamento matemáticos y teóricos → Introducción de trazos en el Lienzo.*

Estos puntos definen el perfil de un objeto. Usando la spline definimos la malla por revolución que se obtiene al hacer girar los puntos del perfil alrededor del eje y, tomando puntos nuevos cada K radianes.

Las siguientes figuras muestran el proceso de creación:

Se introduce el stroke para construir la spline que definirá el perfil de la malla por revolución como muestra la siguiente ilustración.

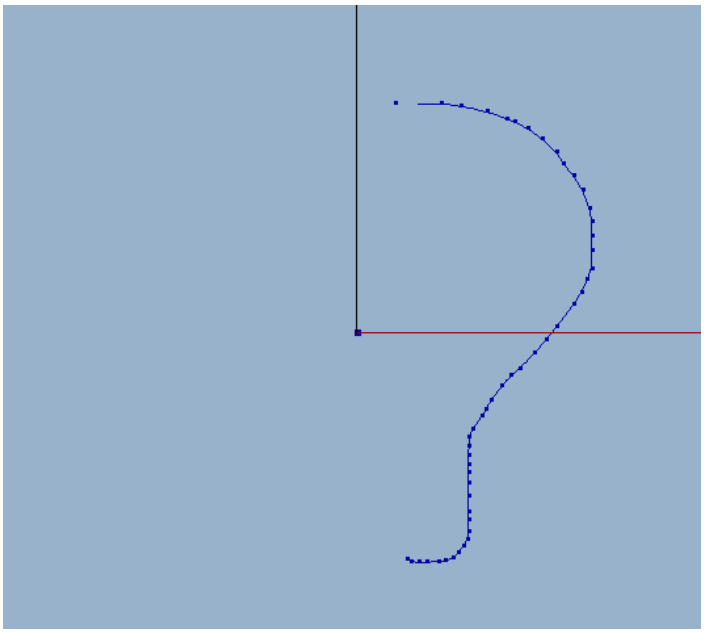


Ilustración 26: Trazo a partir del cual se construye la spline

Hacemos girar cada punto del perfil alrededor del eje Y K grados hasta volver hasta el punto inicial (Se supone que 360 es múltiplo de k). Para obtener mayor precisión el incremento de K debe ser pequeño.

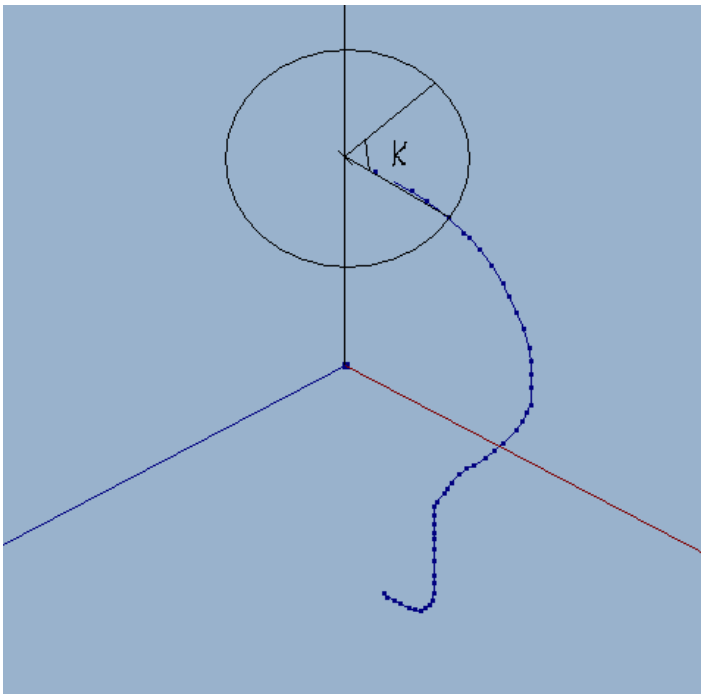


Ilustración 27: Giro del perfil tomando k grados

Al ir uniendo los puntos obtenidos en cada rotación obtenemos una malla por revolución con caras triangulares.

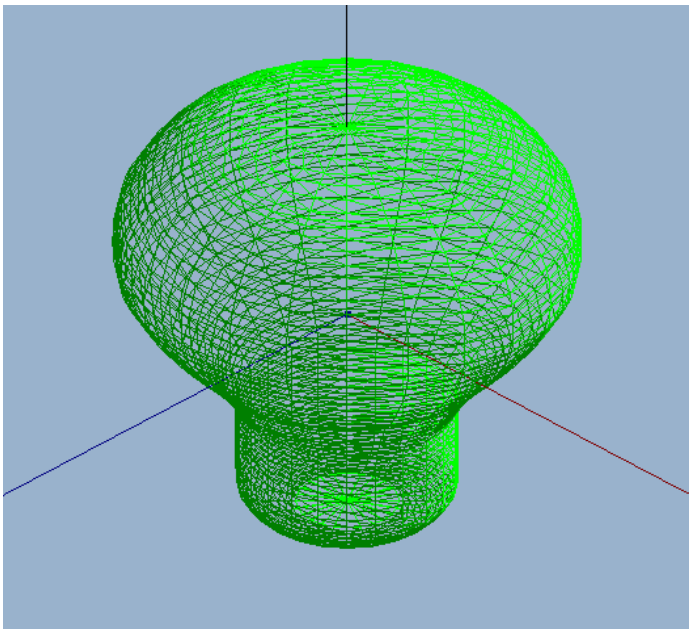


Ilustración 28: Malla por revolución

Al rellenar las caras triangulares que forman la malla obtenemos el resultado final.

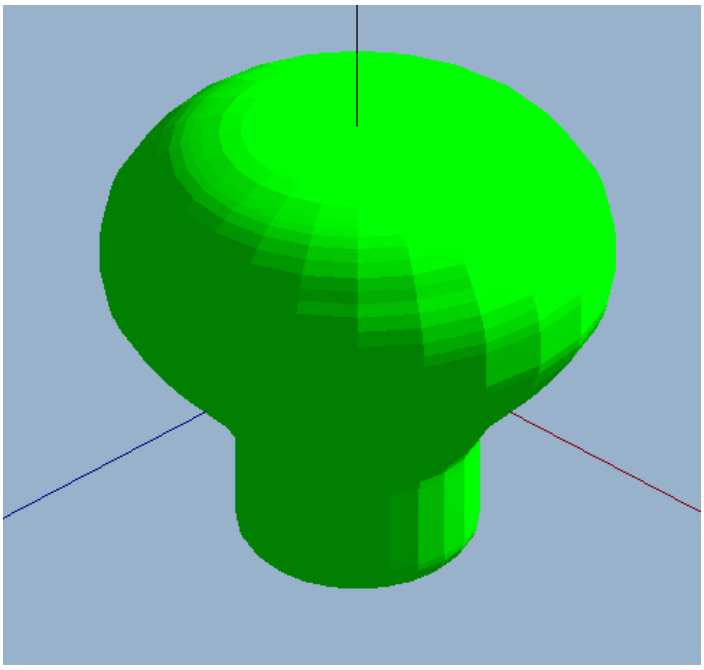


Ilustración 29: Malla formada por polígonos triangulares rellenos

5.6.- MALLA POR EXTRUSIÓN DE LA CURVA B-SPLINE

Esta malla consiste en realizar un "recubrimiento" poligonal sobre una curva B-Spline y su construcción se basa en la creación de polígonos centrados en cada punto que delimita el perfil de la spline y paralelos al plano XZ.

Las siguientes figuras muestran el proceso de creación de la base del estambre:

Introducimos el trazo para el perfil de la base del estambre.

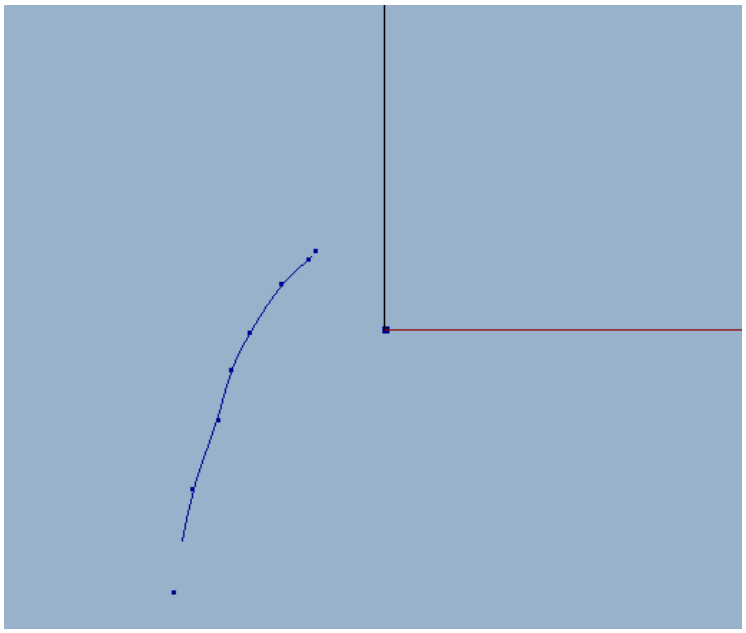


Ilustración 30: Proceso de creación de la base del estambre

A continuación creamos polígonos paralelos al plano XZ que cuyos centros coinciden con los del perfil de la spline. (Obsérvese que un polígono de muchos lados aproxima una circunferencia con fidelidad).

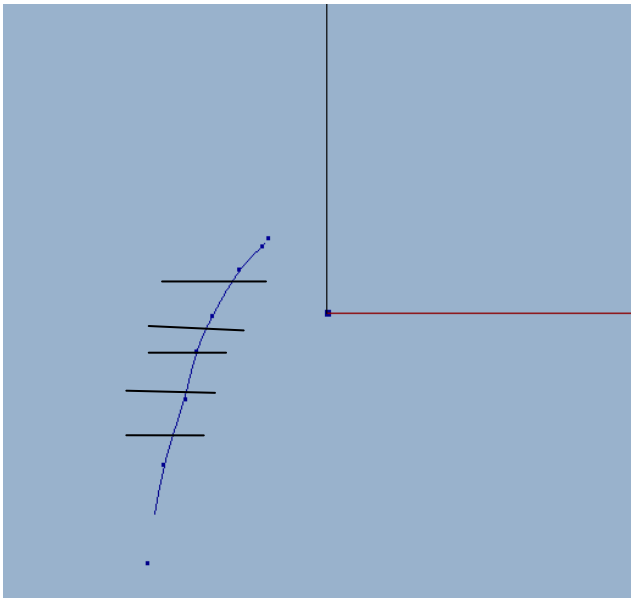


Ilustración 31: Polígonos paralelos al plano XZ

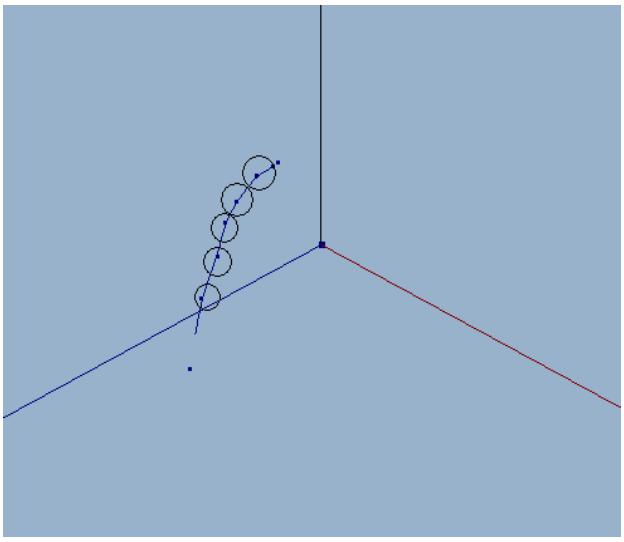


Ilustración 29: Otra imagen de polígonos paralelos al plano XZ

Si unimos los vértices de cada polígono obtenemos una malla que después triangulamos para obtener la base del estambre.

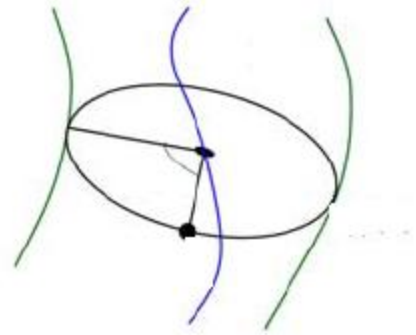


Ilustración 32: Proceso de creación del estambre

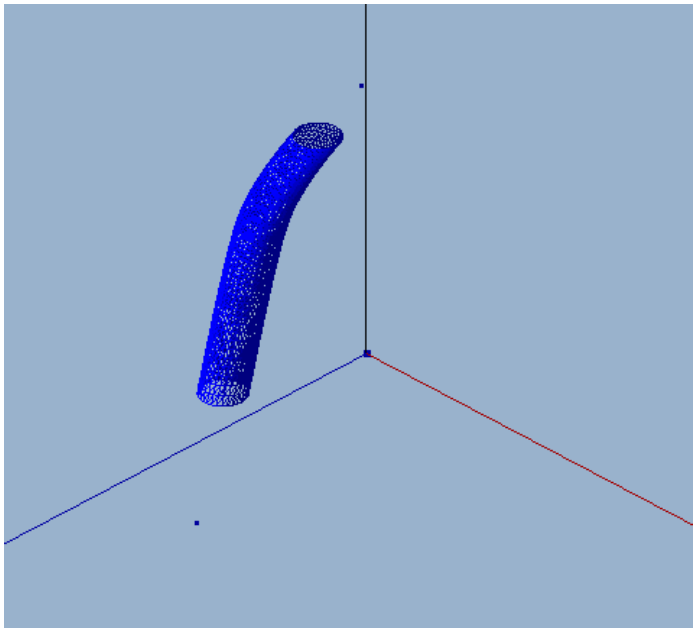


Ilustración 33: Proceso de creación del estambre

5.7.- MALLA POR MEDIA REVOLUCIÓN, ROTACIONES Y TRASLACIONES

Esta malla sigue la misma filosofía y se construye igual que la malla por revolución asociada a una B-spline (véase sección 5.5). La única diferencia es que ahora la rotación se realiza sobre una semicircunferencia y sólo se obtiene media malla.

A continuación en las siguientes figuras se detalla dónde se usa esta malla, mostrando el proceso de creación de la cabeza del estambre. La malla se construye a partir del segundo trazo introducido en la pantalla de creación de estambres, que define una B-spline. Una vez creada esta B-Spline se rota y traslada a la derecha del eje para realizar media revolución. Posteriormente se rota y traslada para colocarla en la cima de la base, usando el cambio de sistema de coordenadas (sección 5.11).

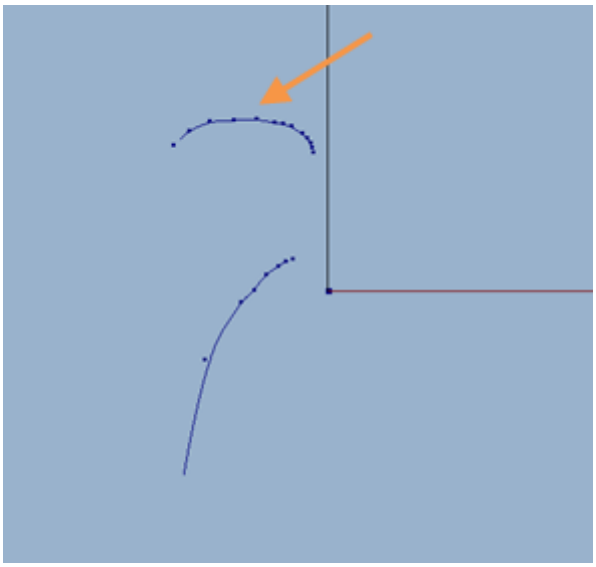


Ilustración 34: Se introduce la spline de la cabeza después de introducir la de la base

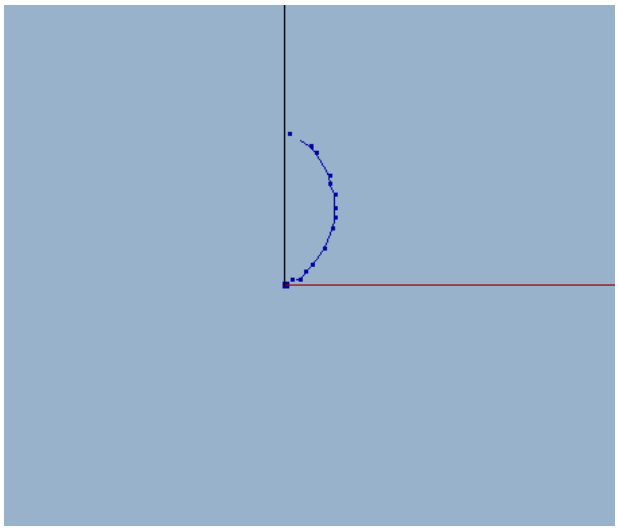


Ilustración 35: La cabeza se traslada y rota al eje y para proceder a la media rotación

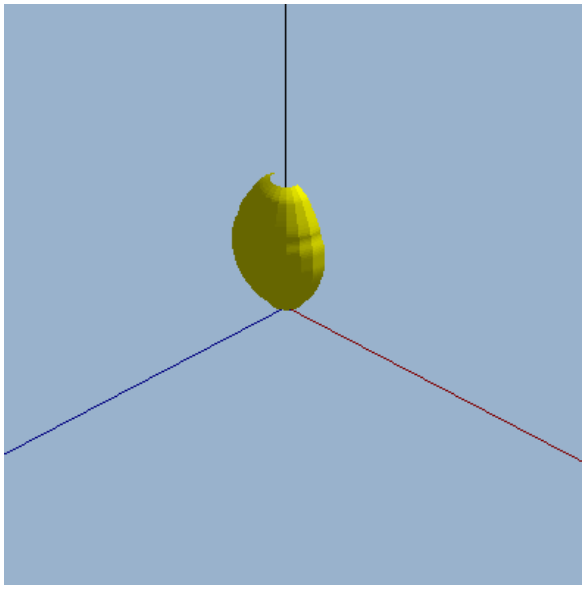


Ilustración 36: Al realizar media revolución obtenemos la cabeza

Al rotar y trasladar la cabeza obtenida por media revolución a la cima de la base, obtenemos el estambre tal y como se puede ver en las siguientes imágenes.

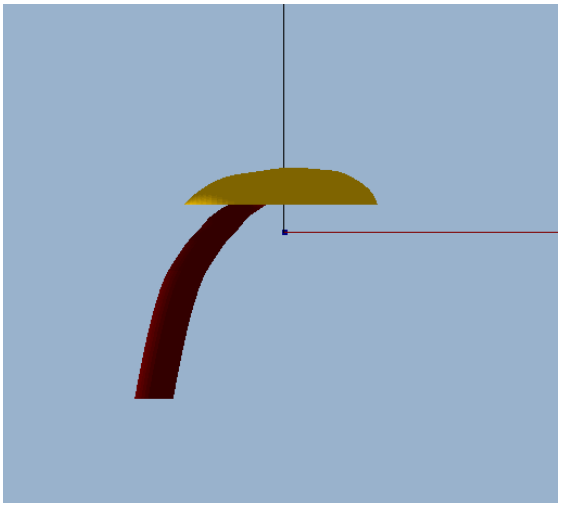


Ilustración 37: Traslado de la cabeza del estambre 1

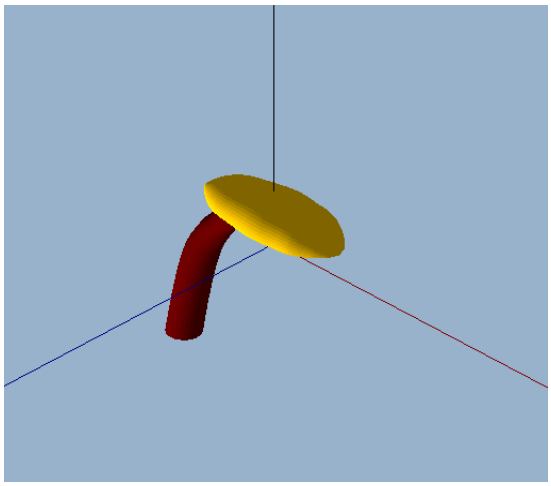


Ilustración 38: Traslado de la cabeza del estambre 2

5.8.- MALLA DE UNA HOJA

La malla de una hoja sirve para representar pétalos y sépalos, ya que ambos componentes tienen una estructura y morfología similar. Debido a esto, hemos optado por realizar el mismo proceso de creación para ambos.

Para obtener la malla que define una hoja es necesario realizar los siguientes pasos:

5.8.1.- CREACIÓN DE LA MALLA CON RESPECTO AL CONTORNO

Una vez que tenemos definido el contorno de la hoja, procedemos a crear su malla. Para ello definimos un numero de divisiones horizontales (divisiones X) y verticales (divisiones Y) en las que vamos a dividir el contorno.

5.8.2.- CREACIÓN DE LA MALLA INICIAL

Inicialmente se calculan puntos uniformemente distribuidos a lo largo del eje Y entre los extremos superior e inferior del contorno (tantos como divisiones Y). Para cada uno de ellos se calcula la recta perpendicular al eje Y que pasa por ese punto, cortando el contorno en dos puntos que definen un segmento. Cada uno de los segmento se divide en puntos uniformemente distribuidos (Tantos como divisiones X) constituyendo los puntos que forman la malla. Dichos puntos se unen en triángulos creando las caras que definen la malla.

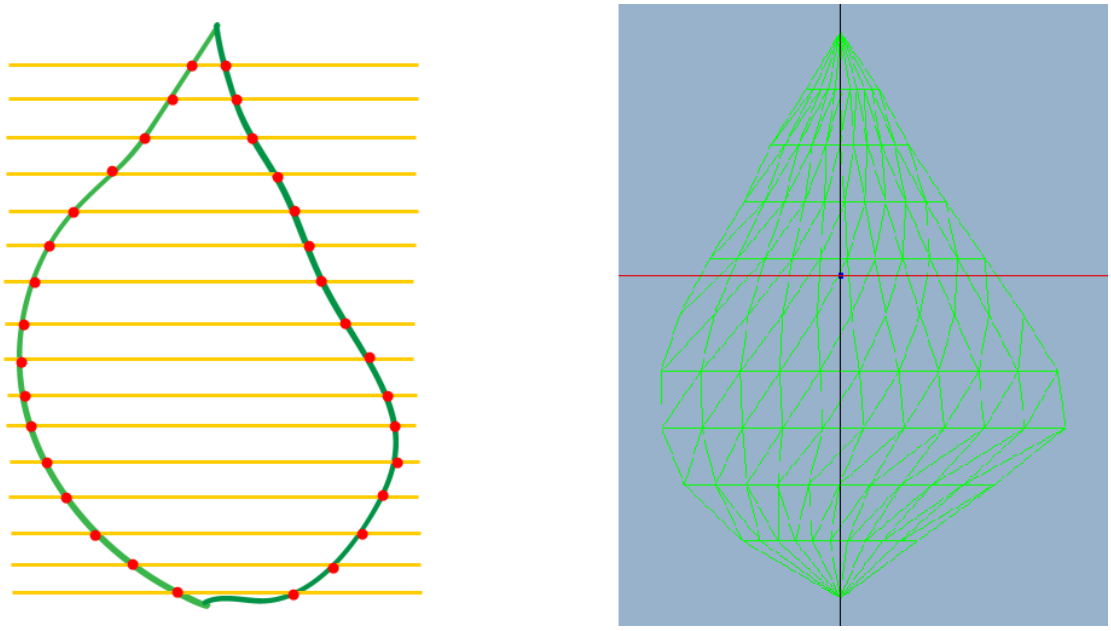


Ilustración 39: Creación de la malla de la hoja

Para llevar a cabo el cálculo de los puntos de corte del contorno de la hoja y de las rectas horizontales perpendiculares al eje Y de la hoja, utilizamos el algoritmo de corte definido en la sección 5.12.

5.8.3.- MODIFICACIÓN DE LA MALLA PARA INCLUIR LA VENA

Para dar mayor realismo a la malla y tener en consideración la b-spline que define la vena, deformamos la malla existente.

El objetivo consiste en que el número de puntos que quedan a la izquierda y derecha de la vena sea el mismo en cada recta perpendicular al eje Y. Para llevar esto a cabo se calculan el punto de corte de cada recta y la b-spline. Cada uno de estos puntos divide la recta en dos segmentos dentro del contorno.

Los puntos (anteriormente distribuidos a los largo de cada recta dentro del contorno) se redistribuirán de la siguiente forma. La mitad de ellos en uno de los segmentos, y la otra mitad en el otro segmento, de forma uniforme en cada segmento. Para ello se asignan las nuevas coordenadas a los puntos ya existentes.

En caso de ser par el número de vértices se quedaría uno más en la izquierda, ya que uno de los vértices se corresponderá con la misma vena.

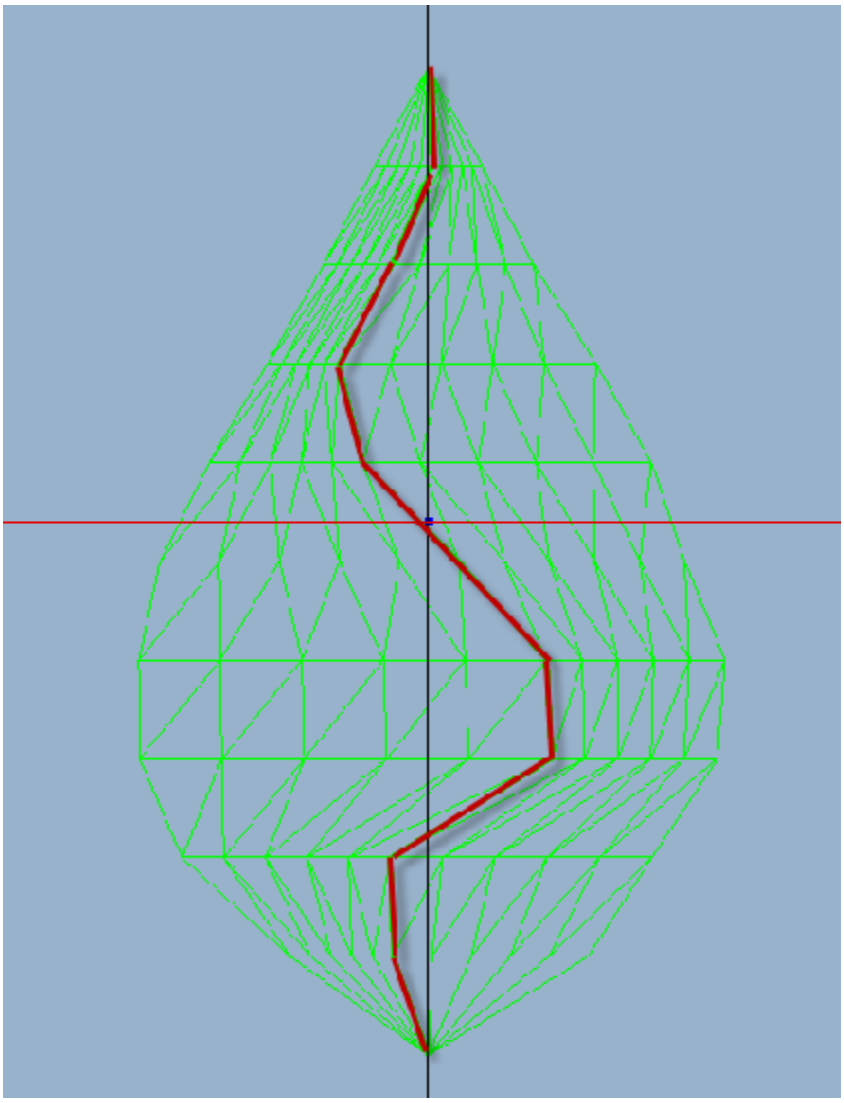


Ilustración 40: Adaptación de la malla a la vena

5.9.- DEFORMACIÓN DE LA HOJA

Una hoja puede deformarse de dos maneras, longitudinalmente o transversalmente. La primera implica una deformación a lo largo de la vena y la segunda a lo ancho de la hoja.

5.9.1.- DEFORMACIÓN LONGITUDINAL

La deformación longitudinal deforma la vena de la hoja longitudinalmente tal y como muestra la imagen.

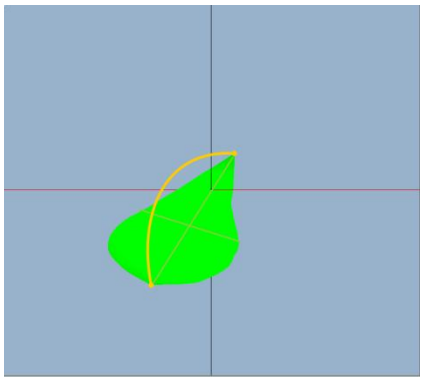


Ilustración 41: Deformación longitudinal de una hoja

El proceso de deformación consiste en trazar una curva desde uno de los vértices que define el contorno de la hoja hasta el otro.

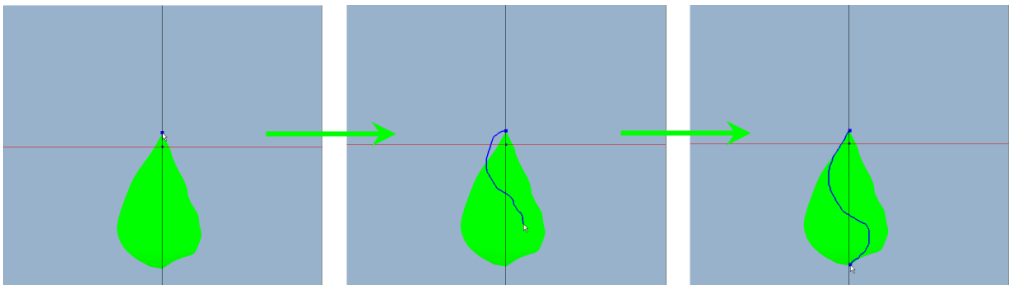


Ilustración 42: Proceso de deformación longitudinal

Para decidir si se trata de una deformación longitudinal se comprueba que el primer y el último punto de la b-spline que representa el trazo dibujado, se encuentran aproximadamente en la misma vertical. Concretamente, calculamos la distancia entre sus coordenadas X y comprobamos si está por debajo de un umbral fijado previamente. Además comprobamos si los vértices elegidos están próximos a los extremos superior e inferior de la malla.

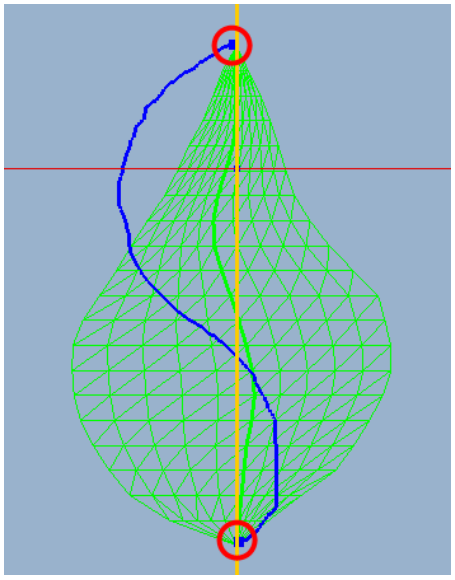


Ilustración 43: Proximidad entre los extremos de la spline y los extremos de la hoja

Una vez superados estos dos tests, el proceso de transformación comienza. Para modificar las coordenadas de los puntos de la malla, usamos la curva que hemos

dibujado en el plano XY (definida mediante una b-spline) y las rectas creadas por el algoritmo de construcción de la malla de la hoja

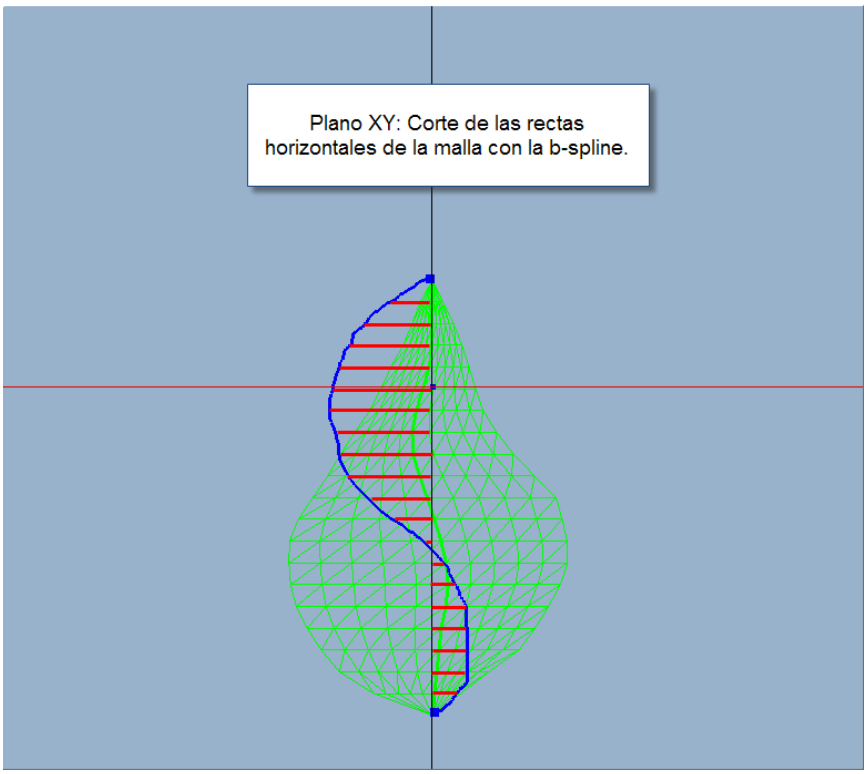


Ilustración 44: Corte de las rectas horizontales de la malla con la b-spline

La transformación se consigue asignado coordenadas Z a los vértices que componen la malla hasta ahora plana (en el plano XY). Concretamente para los vértices de la recta y=cte se calcula el corte de esta recta con la b-spline de la deformación, y se usa la coordenada x resultante en el corte como valor de la coordenada Z.

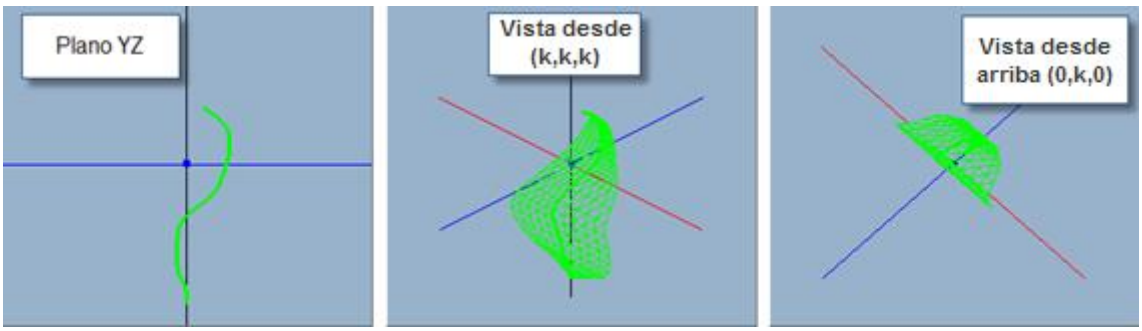


Ilustración 45: Distintas vistas de la malla de la hoja después de aplicar la deformación longitudinal

5.9.2.- DEFORMACIÓN TRANSVERSAL

Para decidir si se trata de una deformación transversal se comprueba que el primer y el último punto de la b-spline que definen la transformación tienen coordenadas próximas. Para ello un umbral que se fija previamente.

Para llevar a cabo la deformación transversal hay que distinguir varios casos posibles, como explicamos a continuación.

5.9.3.- DEFORMACIÓN DESDE UN BORDE AL OTRO

Esta deformación consiste en curvar la hoja de forma transversal, es decir, a lo ancho de la hoja.

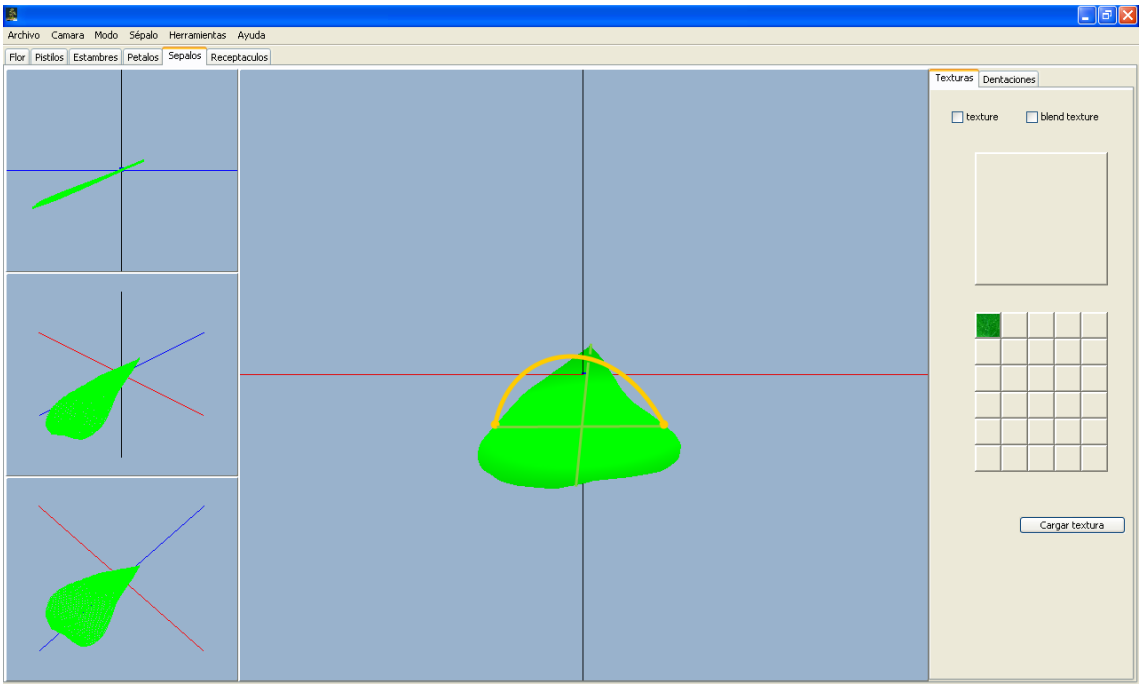
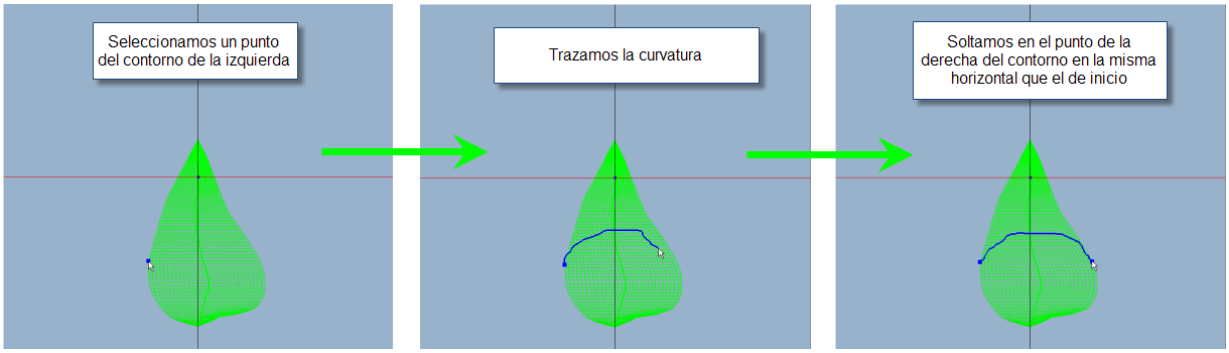


Ilustración 46: Deformación transversal

La curvatura transversal se crea pulsando con el botón izquierdo del ratón en uno de los vértices del borde izquierdo de la hoja, y sin dejar de pulsar se define la forma, arrastrando el ratón hasta otro vértice del borde derecho en la misma horizontal.



Proceso del trazo transversal

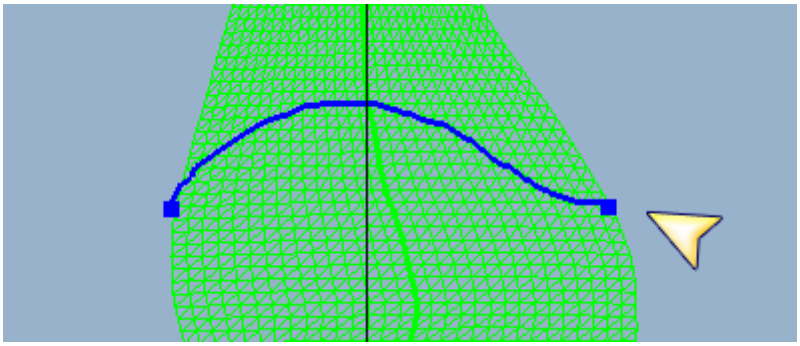


Ilustración 47: Coincidencia de los extremos del trazo con los bordes de la malla

Para decidir si es una deformación válida, comprobamos que el primer vértice del trazo está cerca de alguno de los vértices de la malla que forman el borde izquierdo del contorno de la hoja, y que el segundo se encuentra cerca de la recta horizontal que pasa por el vértice izquierdo escogido. En caso de éxito usamos como vértice derecho de la b-spline de la deformación, el que se encuentra exactamente sobre la recta horizontal.

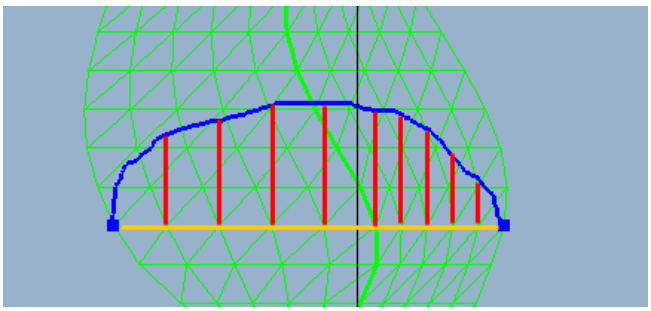


Ilustración 48: Cálculo de los puntos de corte de los vértices de la malla con la b-spline de la deformación

La transformación se aplica asignando coordenadas Z a los vértices de la malla hasta ahora plana (en el plano XY). Concretamente para cada vértice de la recta y=cte se traza una perpendicular, y se calcula el corte que genera con la b-spline de la transformación. La Z que se usa para cada vértice es la distancia que lo separa del punto de corte que le corresponde.

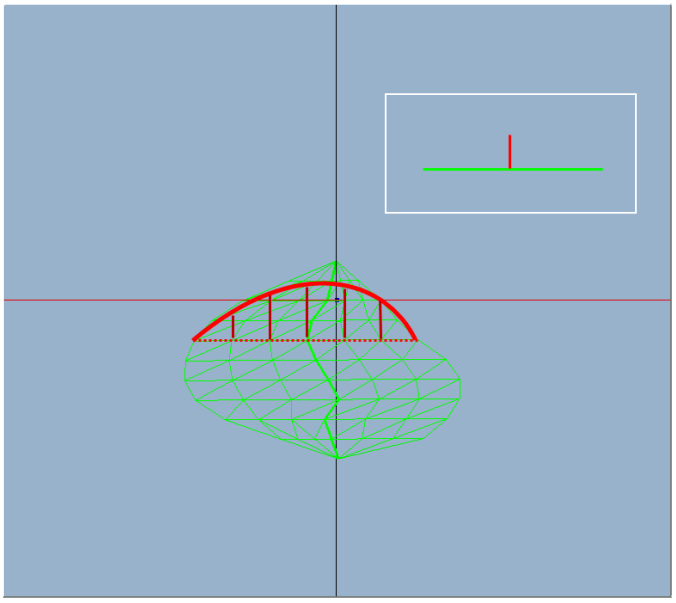


Ilustración 49: Valores Z en la recta dónde se introdujo el trazo de la deformación

Para cada una de las rectas siguientes (anteriores y posteriores a la recta horizontal de la b-spline de la transformación) se calcula un valor Z (para cada uno de los vértices de esas rectas) atenuando el valor Z obtenido para la recta principal. Este valor se calcula proporcionalmente al número de rectas anteriores o posteriores, teniendo en cuenta que al llegar a la última recta la atenuación tiene que ser máxima, es decir, que los valores Z sean 0.

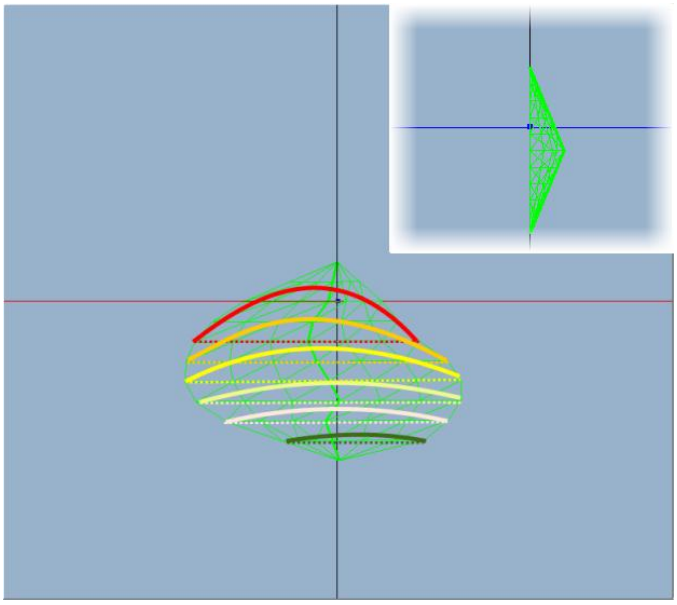


Ilustración 50: Atenuación de las coordenadas Z en las rectas transversales de la malla.

5.9.4.- DEFORMACIÓN DE IZQUIERDA AL CENTRO DE LA VENA

Se procede como en el caso anterior, salvo que en este caso para aceptar un trazo como transformación valida, el último vértice del trazo se comparará con el vértice de la vena que se encuentra sobre la correspondiente recta horizontal.

Para calcular la deformación se procede como en el caso anterior, pero únicamente se tienen en cuenta los vértices de la malla que quedan a la izquierda de la vena.

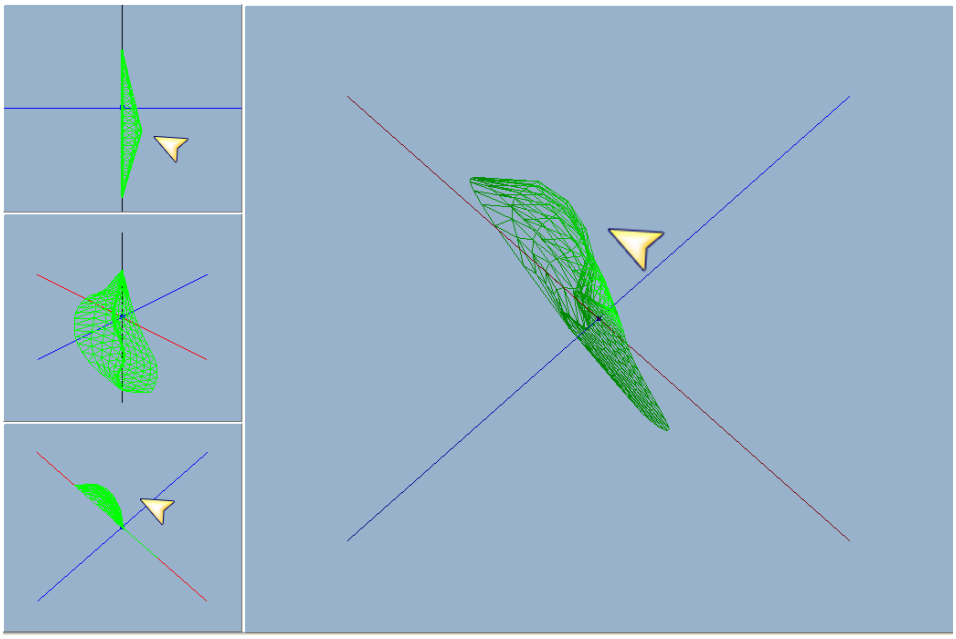


Ilustración 51: Malla de la hoja después de la deformación transversal desde distintos puntos de vista

5.9.5.- DEFORMACIÓN DE DERECHA AL CENTRO DE LA VENA

En este caso en vez del lado izquierdo de la vena se comprueba que el primer vértice está próximo a alguno de los vértices del contorno derecho, y que el segundo se encuentre en la vena sobre la recta horizontal correspondiente.

Se procede de igual manera que en la deformación de izquierda al centro de la vena.

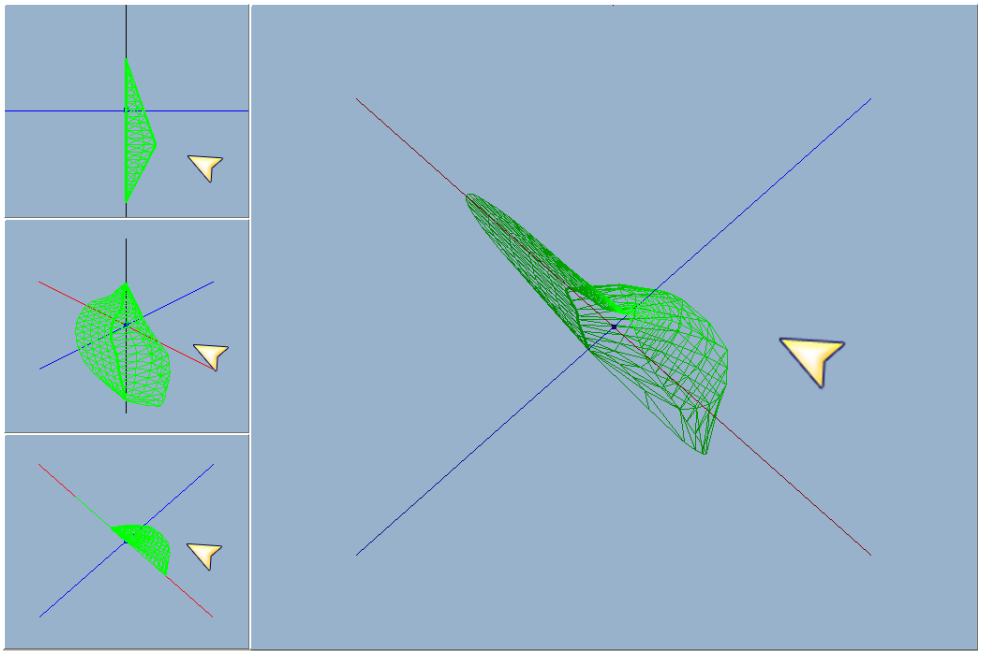


Ilustración 52: Malla de la hoja después de la deformación transversal derecha al centro desde distintos puntos de vista

5.9.6.- DEFORMACIÓN DEL CENTRO A LA DERECHA

Esta deformación supone el ensanchamiento de la malla de la hoja hacia la derecha, redistribuyendo uniformemente los puntos que se encuentran a la derecha de la vena.

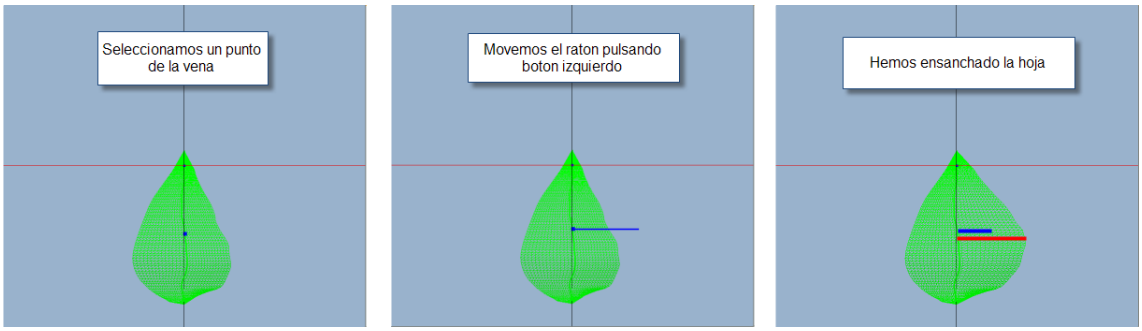


Ilustración 53: Ensanchamiento de la hoja

El proceso de deformación consiste en trazar una curva desde uno de los vértices de la vena de la malla hasta un punto a su derecha.

La recta horizontal que pasa por ese vértice define un segmento entre el contorno y ese vértice. Este segmento se redimensiona sumándole un valor calculado como la distancia entre las coordenadas X del extremo derecho del segmento y el último punto del trazo. Los puntos de la malla del segmento original se redistribuyen uniformemente a los largo de este nuevo segmento.

Para cada una de las rectas siguientes (anteriores y posteriores a la recta de la b-spline de la transformación) se redimensionan los segmentos a la derecha de la vena. Para ello se suma a cada uno de los puntos del borde derecho un porcentaje de la distancia inicialmente calculada. Este porcentaje es proporcional al número de divisiones Y de la

malla, siendo menor cuanto nos alejemos de la recta horizontal inicial usada para definir la deformación. De hecho el incremento desaparece en las rectas próximas a los extremos superior e inferior de la malla.

Una vez desplazado el punto más externo se redistribuyen los puntos intermedios entre la vena y dicho punto de forma equidistante.

De esta forma se ensancha la malla conservando lo máximo posible la forma original.

5.9.7.- DEFORMACIÓN DEL CENTRO A LA IZQUIERDA

Esta deformación supone el ensanchamiento de la hoja hacia la izquierda, de forma idéntica al proceso definido en la sección 5.9.6, teniendo en cuenta que en este caso la deformación es hacia la izquierda en vez de la derecha.

5.10.- ALGORITMO DE DENTICIÓN

Los algoritmos de dentición permiten deformar las hojas añadiendo al contorno de la hoja pequeños triángulos que simulan dientes en forma de pico. El objetivo es aumentar el realismo de las flores creadas, ya que en la naturaleza se presentan estas pequeñas denticiones con mucha frecuencia.

Aunque sólo hemos implementado esta operación en el caso de los sépalos, sería también aplicable a los pétalos ya que ambos componentes se tratan como hojas.

Las opciones disponibles actualmente para configurar las denticiones, con independencia del algoritmo seleccionado, son las siguientes.

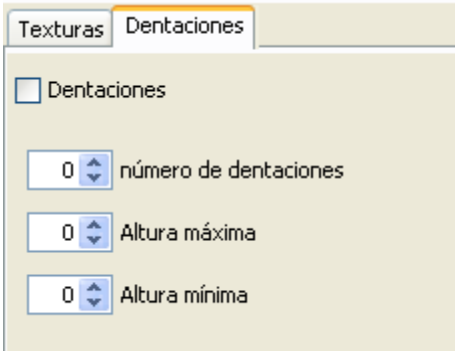


Ilustración 54: Pestaña de denticiones

Número de denticiones: representa el número de dientes presentes en la hoja. En los algoritmos uno y dos, representa el número de dientes por segmento como se explica más adelante.

Altura máxima: valor máximo de la distancia a la que se encuentra el pico del diente del contorno de la hoja.

Altura mínima: valor mínimo de la distancia a la que se encuentra el pico del diente del contorno de la hoja.

Usamos varios algoritmos para generar denticiones en las hojas, a continuación se detalla el funcionamiento de cada uno de ellos.

5.10.1.- ALGORITMO 1. ALGORITMO UNIFORME

Consiste en pegar uno o varios triángulos a cada uno de los segmentos que componen el contorno de la hoja, excepto el primero y el último. El número exacto es el que hemos elegido previamente.

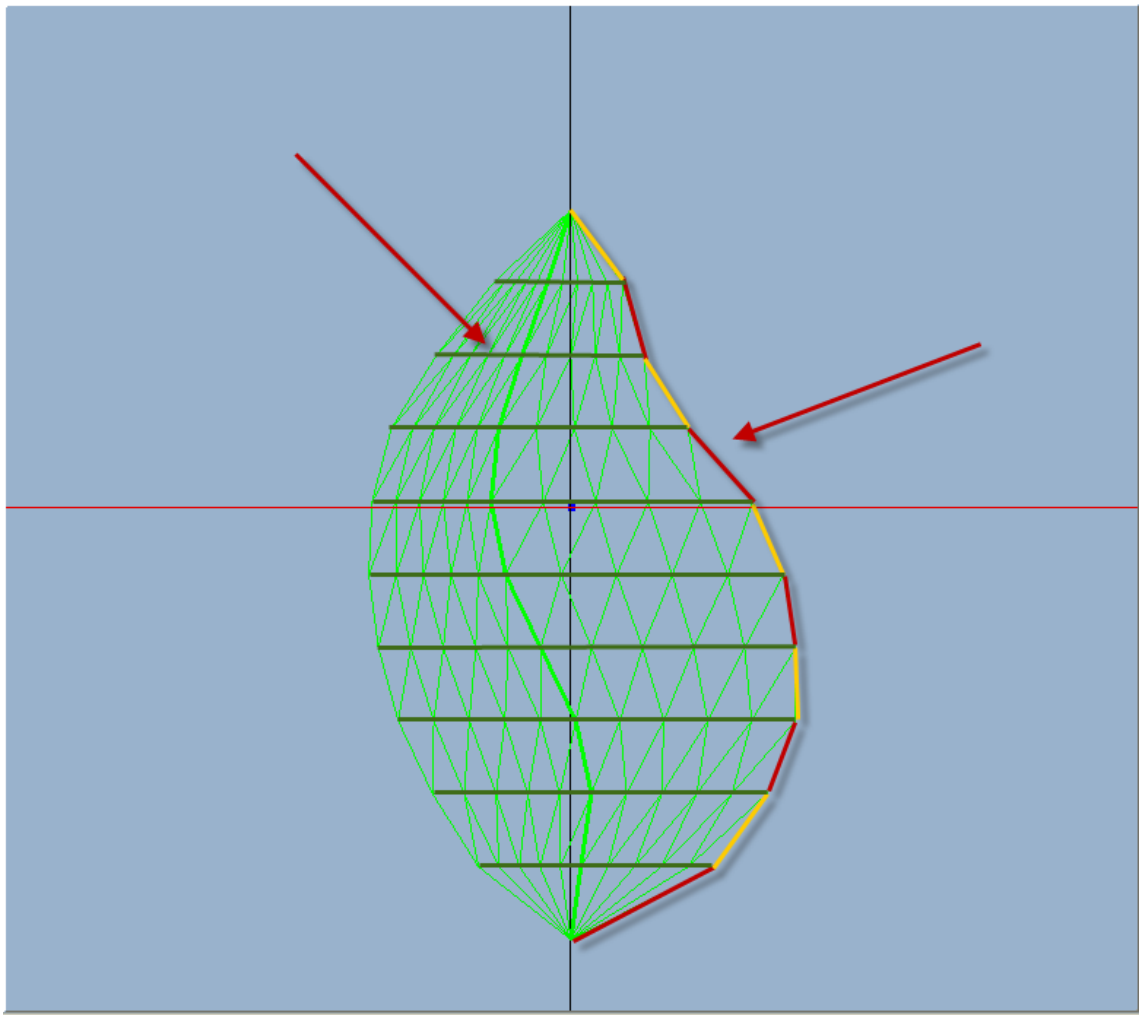


Ilustración 55: Segmentos que componen el contorno

Para calcular estos triángulos dividimos cada uno de los segmentos en tantos sub-segmentos como denticiones hayamos elegido.

Calculando el centro de cada uno de los nuevos segmentos, y el vector unitario director de dicho segmento (calculado mediante la resta de los puntos extremos de dicho segmento y normalizándolo posteriormente), se calcula el vector perpendicular a éste cuya dirección vaya hacia fuera de la malla.

Una vez calculado este vector perpendicular y unitario, lo multiplicamos por la altura. Sumando este nuevo vector al punto medio del segmento obtenemos el tercer vértice del diente, que se corresponde con su pico.

Al unir los tres puntos obtenemos la dentición, tal y como se muestra en la figura.

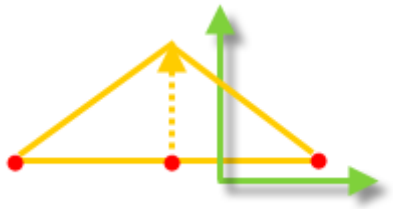


Ilustración 56: Creación de la dentición

Al deformar la malla transversalmente las denticiones quedarían en el plano XY. Para solucionar este problema y conseguir que las denticiones mantengan la curvatura de la malla, se realiza el siguiente proceso.

Se calcula el vector director de uno de los extremos de los segmentos que forman el contorno, con el vértice anterior en la malla, como se muestra en la siguiente imagen.

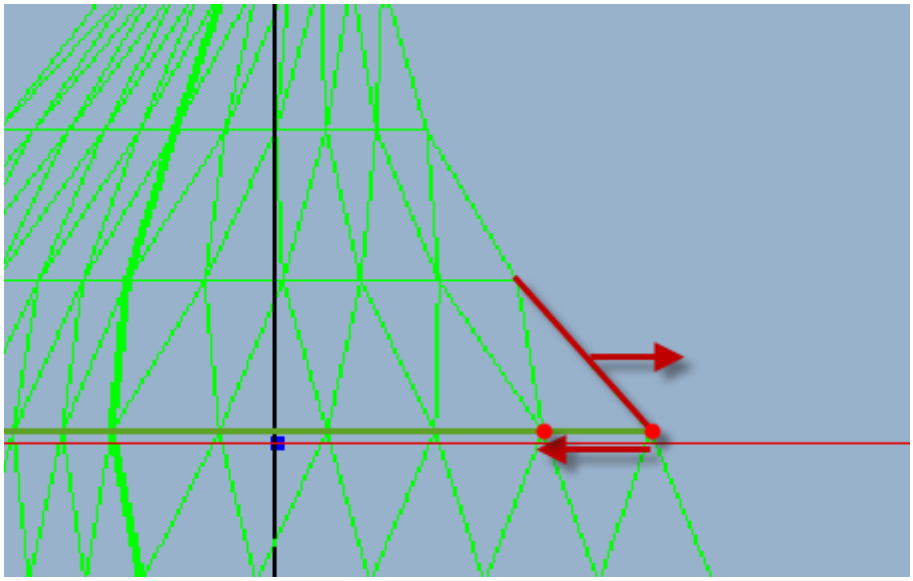


Ilustración 57: Cambio de la orientación de la dentición

Este vector multiplicado por la altura lo sumamos al centro de los segmentos (en cada una de las denticiones) calculados anteriormente. De esta forma las denticiones mantendrán la curvatura con la malla, Ya que el vector usado propaga el efecto que la deformación tuvo sobre la malla.

5.10.2.- ALGORITMO2. ALGORITMO ALEATORIO

Este algoritmo se basa en el mismo proceso de creación que el algoritmo uniforme, la única diferencia es a la hora de crear la altura de las denticiones, ya que estas se calculan de forma aleatoria, calculando para cada dentición un valor entre el máximo y el mínimo fijado como parámetros de entrada.

5.11.- CAMBIO DE SISTEMA DE COORDENADAS

Conocidas las coordenadas cartesianas de un punto, respecto a unos ejes determinados, podemos averiguar las coordenadas de ese punto respecto a otro sistema de coordenadas tal y como se muestra en la siguiente figura. Esto se puede entender como la aplicación de determinadas transformaciones afines.

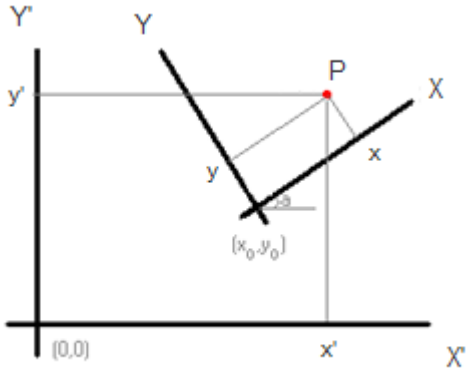


Ilustración 58: Obtención de las coordenadas de un punto respecto a otro eje de coordenadas

En la figura (x, y) son las coordenadas del punto P respecto a los ejes de coordenadas XY y (x_0, y_0) las coordenadas del origen de coordenadas de los ejes XY respecto los ejes X'Y'. Si "a" es el ángulo que se giran los ejes, las coordenadas de P respecto a X'Y' se calculan como sigue:

$$x' = x_0 + x\cos a - y\sin a$$
$$y' = y_0 + y\cos a + x\sin a$$

5.12.- INTERSECCIÓN RECTA/SEGMENTO

Este algoritmo, que también sirve para calcular la intersección entre dos segmentos, se utiliza en la creación de la malla de la hoja y en su deformación.

Las ecuaciones paramétricas de la recta R y el segmento S son:

R (t_1) = A_1 + t_1*$\mathbf{v_1}$, donde t_1 Є [-∞, ∞] , A_1 es un punto de la recta y $\mathbf{v_1}$ es el vector director.

S (t_2) = A_2 + t_2*$\mathbf{v_2}$, donde t_2 Є [0, 1], A_2 es un punto del segmento y $\mathbf{v_2}$ su vector director.

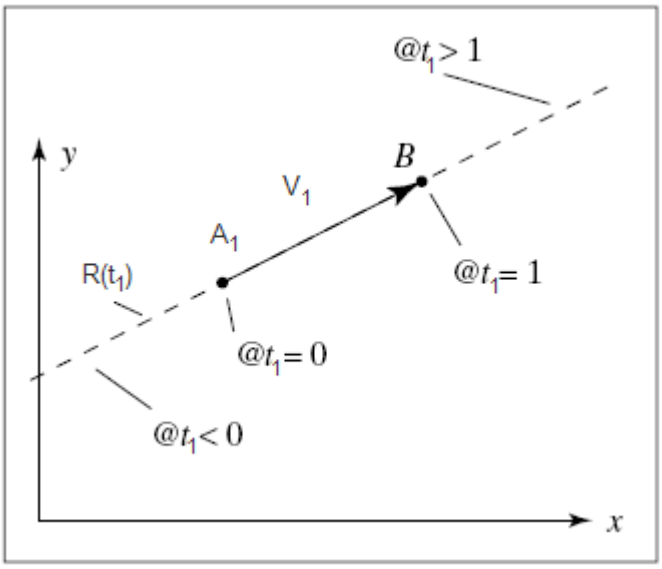


Ilustración 59: Representación paramétrica L(t) de una recta

El problema de la intersección consiste en determinar si existen t_1 y t_2 tal que $A_1 + t_1 * \mathbf{v_1} = A_2 + t_2 * \mathbf{v_2}$. Esto se resuelve como sigue:

1. Si movemos A_2 en la ecuación $A_1 + t_1 * \mathbf{v_1} = A_2 + t_2 * \mathbf{v_2}$ y llamamos $\mathbf{a} = A_1 - A_2$ obtenemos:

$$\mathbf{a} + t_1 * \mathbf{v_1} = t_2 * \mathbf{v_2}$$

2. Para despejar las incógnitas t1 y t2 usamos el vector perpendicular izquierdo a $\mathbf{v_1}$ y $\mathbf{v_2}$. El vector perpendicular a la izquierda o "vector normal izquierda" de un vector **v** (representado por $\mathbf{v}^{\perp}$) es aquel que resulta de girar el vector 90º a la izquierda. Si v = (v_x, v_y), se obtiene como $\mathbf{v}^{\perp} = (v_x, -v_y)$. El producto escalar, por tanto, de $\mathbf{v}^{\perp}$ y **v** será 0 pues son perpendiculares.

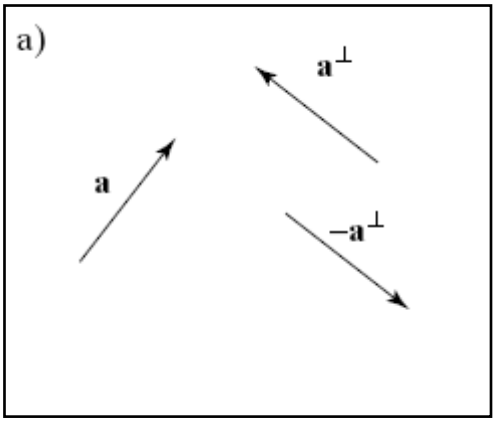


Ilustración 60: El vector a y su perpendicular a⊥

3. Multiplicando la ecuación por $\mathbf{v_2}^{\perp}$ obteniendo:

   $\mathbf{a} * \mathbf{v_2}^{\perp} + t_1(\mathbf{v_1} * \mathbf{v_2}^{\perp}) = 0$, ya que $\mathbf{v_2} * \mathbf{v_2}^{\perp} = 0$.

4. Despejando t_1 tenemos $t_1 = -(\mathbf{a} * \mathbf{v_2}^{\perp})/(\mathbf{v_1} * \mathbf{v_2}^{\perp})$.

   Si en la ecuación (**) multiplicamos por $\mathbf{v_1}^{\perp}$ y despejando obtenemos t_2

   $t_2 = (\mathbf{a} * \mathbf{v_1}^{\perp})/(\mathbf{v_1}^{\perp} * \mathbf{v_2})$.

5. Se pueden dar varios casos a la hora de obtener t_1 y t_2:
   a. Los denominadores son cero, esto es porque $\mathbf{v_1} * \mathbf{v_2}^{\perp} = 0$ y $\mathbf{v_1}^{\perp} * \mathbf{v_2} = 0$, esto quiere decir que la recta y el segmento son paralelos.
      i. Si A_2 pertenece a la recta R entonces el segmento está sobre la recta (Ilustración 44 (e)).
      ii. En otro caso la recta y el segmento son paralelos pero distintos (Ilustración 44 (b).
   b. En otro caso t_1 y t_2 son computables y pueden darse dos subcasos:
      i. Si t_2 no tiene un valor comprendido entre 0 y 1 entonces no hay intersección, se cortarían si fuese una recta en lugar de un segmento (Ilustración 44 (a)).
      ii. En otro caso la recta y el segmento se cortan (Ilustración 44 (c) y (d)).

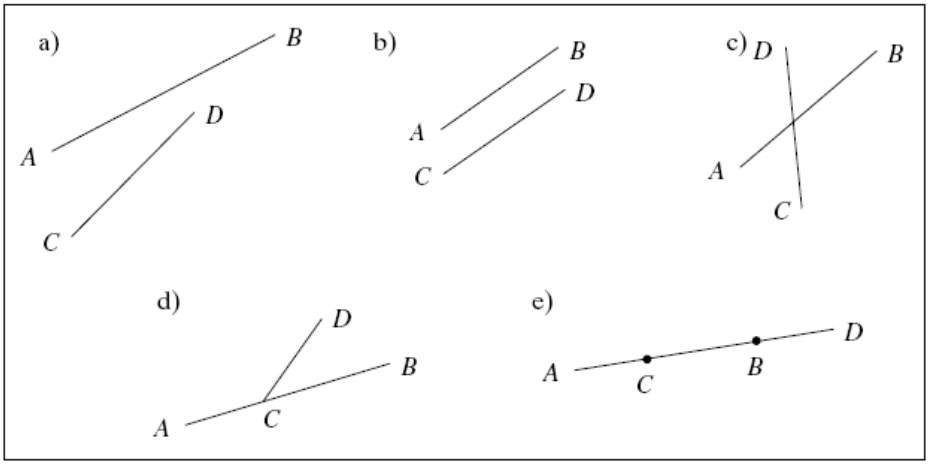


Ilustración 61: Posibles casos entre recta y segmento

6. Una vez que hemos obtenido t_1 y t_2 y comprobado que hay intersección, calculamos el punto exacto de intersección sustituyendo en cualquiera de las ecuaciones el valor correspondiente de t_1 o t_2.

5.13.- AÑADIR TEXTURAS. TEXTURAS EN SUPERFICIES CURVAS

Para añadir las texturas decidimos aplicar el siguiente algoritmo de asignación de coordenadas de texel a pixel, de entre las distintas posibilidades existentes.

La siguiente imagen muestra las coordenadas de los texels de una textura, que van de 0 a 1:

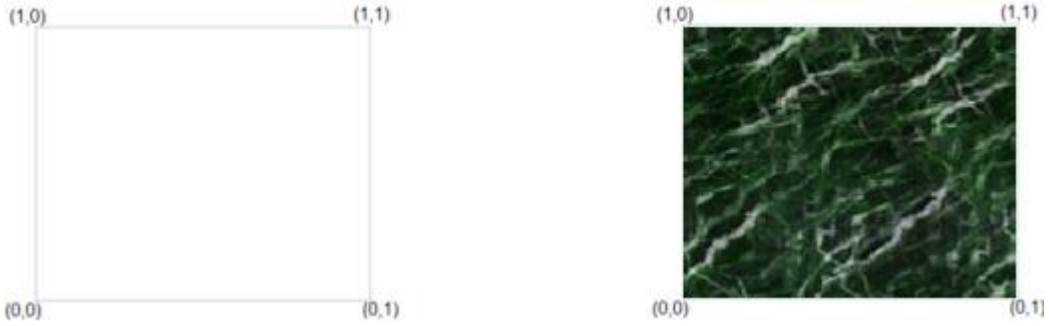


Ilustración 62: Coordenadas los texels de una textura

Calculando los cuatro puntos más externos del objeto dentro del plano XY (los puntos con la menor X, la mayor X, la menor Y y la mayor Y) se crea un rectángulo de lados paralelos a los ejes X e Y que pasa por los puntos anteriormente calculados.

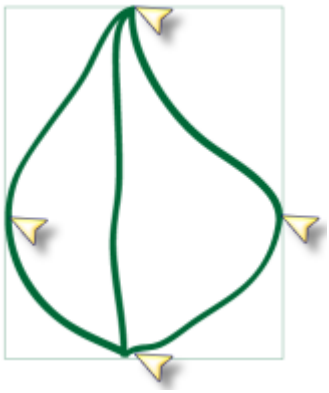


Ilustración 63: Indicación de los cuatro puntos más externos de una hoja

De esta forma los texels de las esquinas de la textura se hacen corresponder con los pixeles definidos por los vértices del rectángulo, asignando a cada uno de los píxeles su correspondiente texel calculado mediante interpolación lineal. Todos estos cálculos se hacen tratando los puntos como si se encontraran sobre el plano XY ya que las coordenadas de textura son sobre dos dimensiones.

Los formatos de textura que se pueden añadir a los objetos son los más habituales: .jpg, .png, .bmp… Para hacer esto posible se han utilizado herramientas de decodificación de archivos de imagen para soportar estos formatos.

Una vez añadida la textura a los objetos se permiten ciertas opciones sobre la misma, como modificar la forma en la que se muestra, ofreciendo la opción de fusionar la imagen de la textura con los colores asignados a los objetos. Esto puede apreciarse en la siguiente imagen.

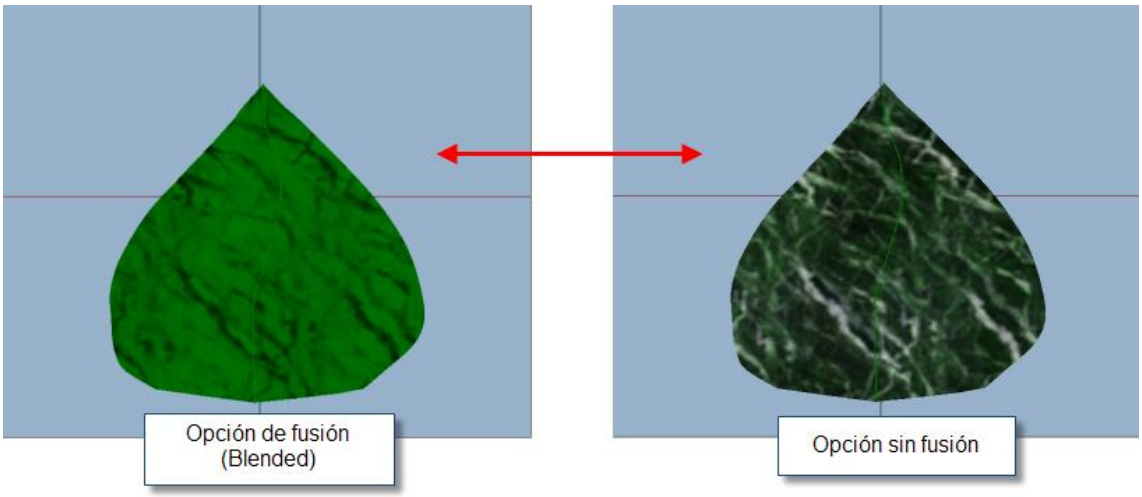


Ilustración 64: Opción de fusión

5.14.- ALGORITMOS PARA CIRCUNFERENCIAS

5.14.1.- CREACIÓN DE UNA CIRCUNFERENCIA

Una circunferencia se simula mediante conjunto de puntos equidistantes cuya distancia a otro punto llamado centro es siempre la misma. Estos puntos se encuentran unidos mediante segmentos dando una sensación de curvatura. También se puede decir que una circunferencia es un polígono regular con infinitos lados. En

consecuencia el procedimiento de generación de circunferencias es el mismo que el de polígonos (Capitulo 5).

5.14.2.- DISTRIBUCIÓN DE LOS COMPONENTES EN UNA CIRCUNFERENCIA

Cuando añadimos componentes de un mismo tipo a la flor, se distribuyen de forma equidistante a lo largo de su correspondiente circunferencia. Para lograrlo, el método que hemos utilizado consiste en generar un polígono regular imaginario de tantos lados como componentes se hayan añadido hasta ese momento. De esta forma tenemos los puntos de inserción 2D de cada componente en el receptáculo. Una vez calculados, usamos otro polígono regular centrado en esos puntos para representar la posición de cada componente en el editor estructural de la flor. El tamaño de estos polígono varía según el tamaño introducido por el usuario para cada componente.

La siguiente figura, muestra como los componentes se distribuyen en un polígono imaginario (A) y usas otro polígono regular (B) para representar cada componente.

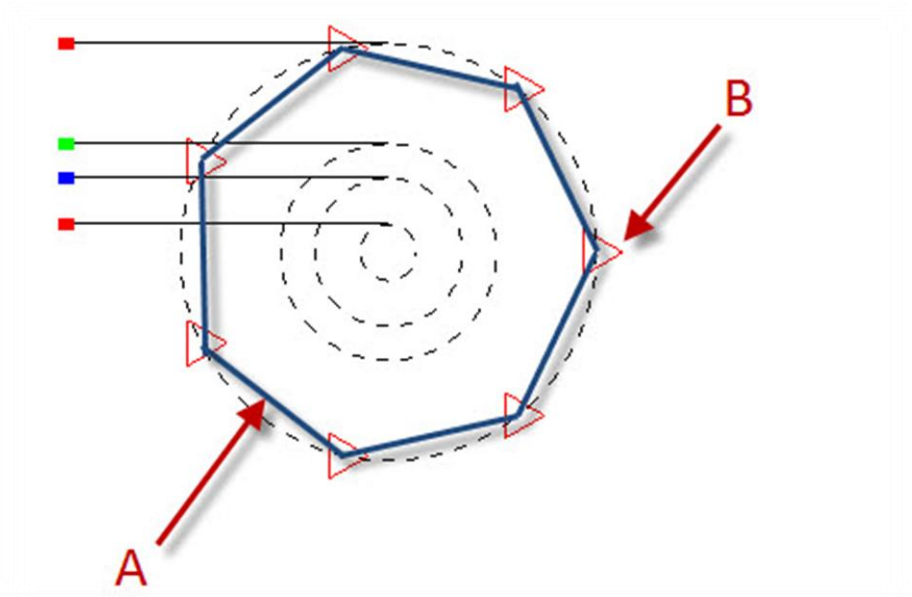


Ilustración 65: Editor estructural: distribución de los componentes en la circunferencia

5.15.- ALGORITMO DE CORRESPONDENCIA ENTRE 2D Y 3D

Una de las tareas más complejas de la aplicación es la de obtener la posición 3D de cada una de las componentes en el receptáculo a partir de su posición 2D en el editor estructural. Para resolverla hemos diseñado un algoritmo que devuelve un elemento del tipo coordenadas 3D que incluye la posición, inclinación y ángulo de rotación en el plano XZ, del lugar de inserción del componente.

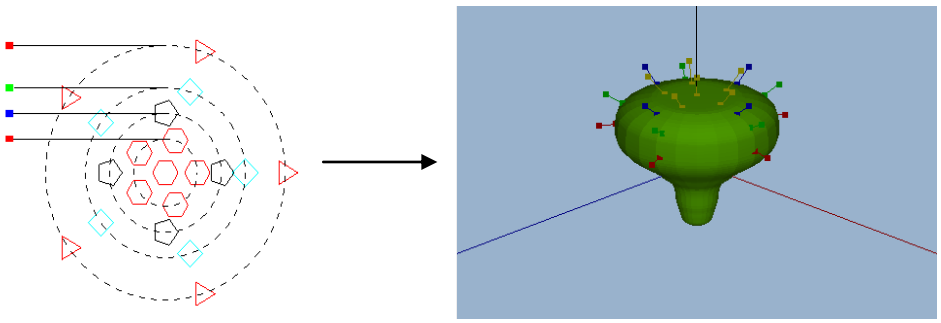
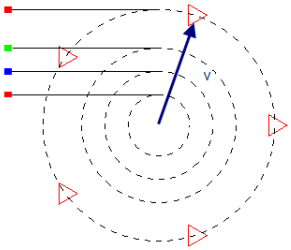


Ilustración 66: Correspondencia entre 2D y 3D

Este algoritmo consta de una serie de pasos que se describen a continuación:

1. Partimos de un punto situado en las circunferencias 2D, a partir del cual pretendemos obtener las coordenadas en 3D que le corresponden en el receptáculo.



2. La distancia respecto al eje central del receptáculo, radio-2D en lo sucesivo, es el resultado del módulo del vector que une el centro de las circunferencias con el punto.
3. Se calcula la longitud de la curva sobre la que se va a revolucionar el perfil del receptáculo (ver sección 5.5). Esta longitud total se calcula acumulando la longitud de cada segmento que forma la curva del perfil. En la siguiente figura observamos un "zoom" del perfil, la longitud sería la suma de cada segmento del perfil = L1 + L2 + L3 +… + Ln.

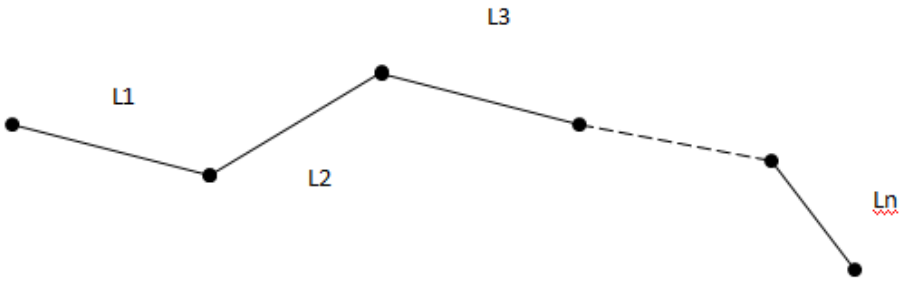


Ilustración 67: Suma de la longitud de cada segmento para obtener la longitud total

4. Se coloca el punto al principio del perfil, superponiendo el radio-2D sobre el perfil del receptáculo. Suponiendo que el perfil está estirado en línea recta, calculamos las coordenadas del punto del perfil donde termina la coincidencia, y usaremos su coordenada Y como la coordenada Y en 3D del punto de inserción. El proceso más detallado es el siguiente:

   a. Para superponer el radio-2D en el perfil partimos de una longitud acumulada con valor cero, deseamos llegar a una longitud que es el radio-2D en la circunferencia.
   b. Vamos acumulando las longitudes de los segmentos hasta que es mayor que la longitud a situar.

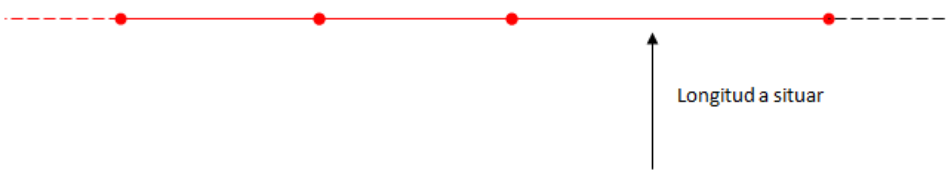


Ilustración 68: Superamos la longitud del radio

   c. Volvemos al punto anterior en el que aún no nos pasamos de la longitud a situar.

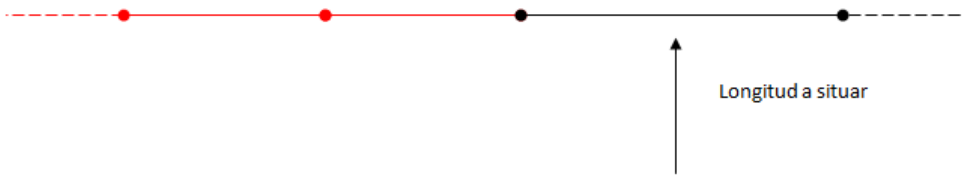


Ilustración 69: Volvemos al último punto antes de pasarnos

   d. Avanzamos una distancia igual a la diferencia entre la longitud a situar y la longitud acumulada, y así obtenemos la situación exacta del punto donde termina el radio-2D.

Ilustración 70: Hemos encontrado el punto que buscábamos

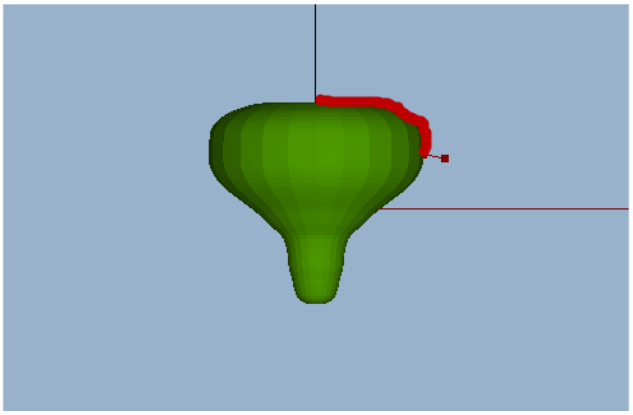


Ilustración 71 : Resultado de superponer la longitud del radio sobre el perfil del receptáculo

5. Una vez hallada la coordenada 3D en el eje Y, hay que calcular el resto de sus coordenadas. Para ello, tenemos que rotar la componente sobre el eje Y un cierto ángulo a lo largo del plano XZ en el que por ahora está situada. Primero necesitamos calcular el ángulo (α) a rotar utilizando la coordenada X del punto en 2D y la longitud del radio-2D, que llamamos *r*.

$$\cos\alpha = \frac{x}{r}$$

Rotamos la componente mediante senos y cosenos.

$$x = r \times \cos\alpha$$

$$z = r \times \sin\alpha\ ,$$

Calculados los valores de x, y, z ya tenemos las coordenadas correspondientes al punto en 3D.

6. Dependiendo de su posición en el perfil, el componente situado en ese punto tendrá una inclinación determinada. Se pueden dar los siguientes casos:
    a. Si un punto está en la longitud cero del perfil o muy próxima, tendrá una inclinación de 90 grados.
    b. Si un punto tiene un radio mayor que la longitud del perfil, tendrá una inclinación de -90 grados.
    c. El punto medio de la longitud del perfil tiene inclinación de cero grados.
    d. Un punto situado entre el del apartado a y el del c tendrá una inclinación positiva, siendo más grande cuanto más se acerque al punto del apartado a.
    e. Un punto situado entre el del apartado c y el del b tendrá una inclinación negativa, siendo más negativa cuanto más se acerque al punto del apartado b.

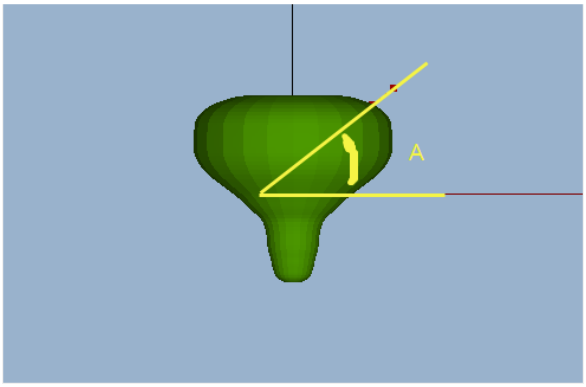


Ilustración 72: Angulo de inclinación (positiva) de un componente

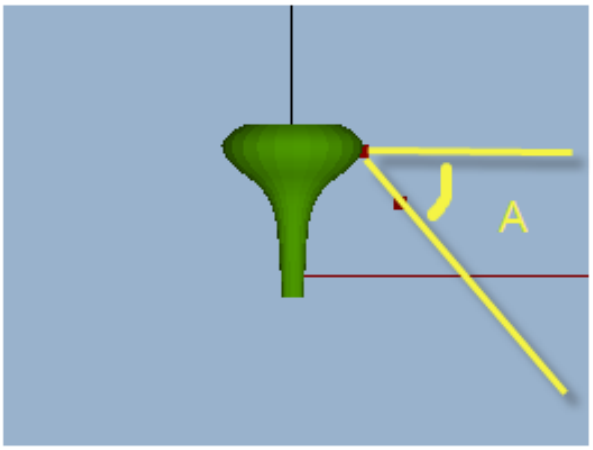


Ilustración 73: Angulo de inclinación (negativa) de un componente

Tras el cálculo del algoritmo ya tenemos la correspondencia en 3D de la componente para así poder situarla en la superfície del receptáculo.

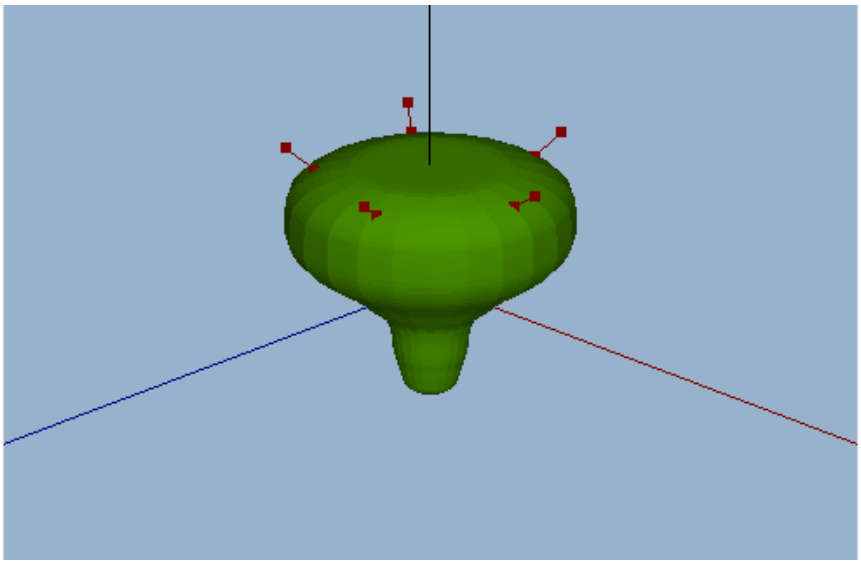


Ilustración 74: Resultado en tres dimensiones de lo definido en dos mediante el editor estructural

5.16.- CAPTURAS DE PANTALLA

Mediante archivos de tratamiento de imagen, codificadores y decodificadores de imagen, permitimos al usuario realizar capturas de la pantalla principal de cada uno de los editores integrados en la herramienta. El formato de estos archivos es .jpg.

Esta operación se ejecuta automáticamente al añadir un elemento a la galería de objetos, pues las capturas se usan para visualizar la galería. La captura en este caso pasa una serie de filtros para obtener una réplica de tamaño reducido.

5.17.- CREACIÓN DE ARCHIVOS PLY

La aplicación permite exportar cada uno de los componentes individuales que componen la flor, así como la propia flor en formato PLY.

Los archivos ply guardan los puntos que forman la malla de los objetos e información adicional que no explicaremos ahora. Para saber más sobre este tipo de formato, se puede consultar el apéndice A2.

Para crear estos archivos, tanto en formato binario como en formato ASCII, tenemos que tener en cuenta la estructura de datos y la organización de la escena tal y como se explica en la sección 5.18. Debido a esto, guardar los puntos que forman la escena no es inmediato.

El valor de los puntos que representan los objetos no se corresponde con las coordenadas del punto que nosotros vemos, ya que son el resultado de aplicar transformaciones afines sobre los objetos, para trasladarlos y rotarlos. Así pues, para obtener el valor de los puntos finales que vemos en la pantalla, debemos multiplicar de forma recursiva las matrices (correspondientes a las transformaciones afines) de cada uno de los componentes que se organizan en la estructura jerárquica de la escena. Es decir, la matriz de una componente afecta a sus hijos en la estructura.

De esta forma obtenemos los puntos finales que guardaremos en el archivo PLY.

Las caras ya las teníamos definidas en las mallas que se utilizan para representar cada uno de los componentes de la flor, por lo que esta información se guarda fácilmente

El formato utilizado para guardar los archivos PLY se puede consultar en el Capítulo *8*.

A continuación se muestra la imagen de uno de los archivos .PLY creados con nuestra aplicación, visualizada con “Headus PLY tool”, una herramienta de visualización de archivos .PLY.

Ilustración 75: Imagen de un archivo PLY con Headus PLY Tool

La siguiente imagen muestra la visualización del mismo ejemplo con Paraview, otra herramienta de visualización y composición.

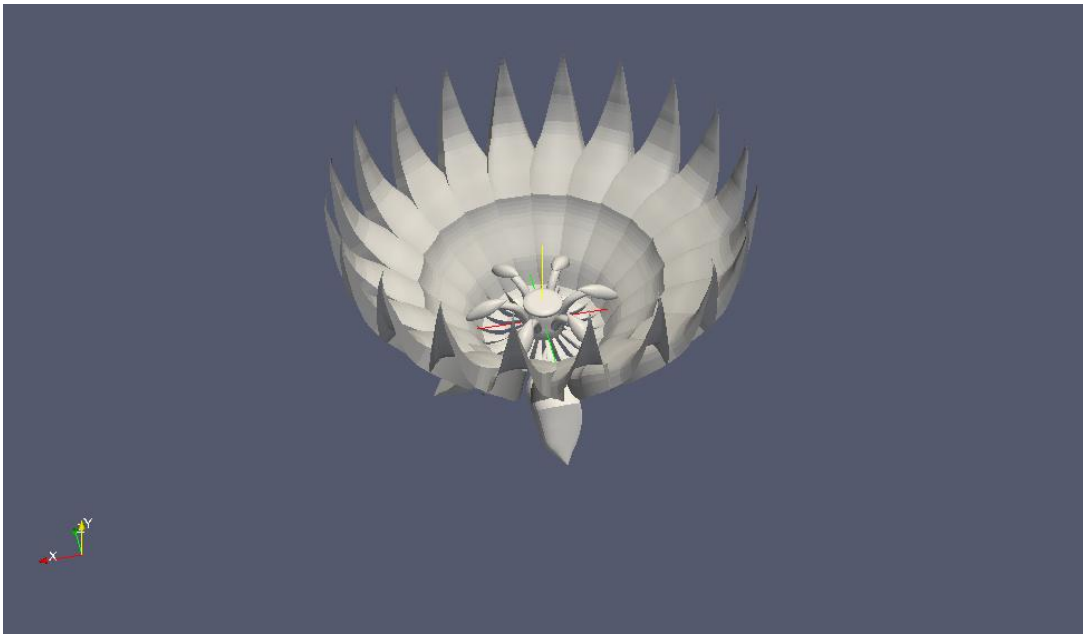


Ilustración 76: Imagen de un archivo PLY con Paraview

5.18.- TRANSFORMACIONES AFINES

Las transformaciones pueden usarse para:

1. Diseñar escenas. Colocación de objetos virtuales. Existen varios tipos:
    a. Escalaciones, que alteran el tamaño.
    b. Traslaciones, que alteran la posición.
    c. Rotaciones, que alteran la orientación.
2. Diseñar objetos. Objetos que repiten un patrón o poseen simetrías.
3. Animaciones. Pintar y borrar imágenes virtuales.
4. Exploración de la escena. Colocar la cámara o los objetos para ver desde todos los ángulos y posiciones.

Como veremos a continuación nuestra implementación usa muchas transformaciones afines. Para más información sobre transformaciones afines ver Apéndice A3.2.1.

5.18.1.- APLICACIÓN AL PROYECTO

OpenGL gestiona las transformaciones afines mediante la matriz de modelado y vista.

- Matriz de modelado: recoge las distintas transformaciones (traslaciones, rotaciones, etc) que ha sufrido el objeto.
- Matriz de vista: recoge el sistema de coordenadas fijado en la cámara.

Para manejar las transformaciones afines en la aplicación se ha creado una jerarquía de los distintos componentes. Cada elemento de una flor es un componente 3D, cada componente 3D tiene una matriz sobre la que se aplican las transformaciones, y esta matriz se post multiplica por la matriz activa de modelado y vista antes de renderizar el componente. Cada componente puede estar formado a su vez por otros componentes de manera que la aplicación de una transformación sobre el debe propagarse a sus hijos. La jerarquía de componentes usada es la siguiente.

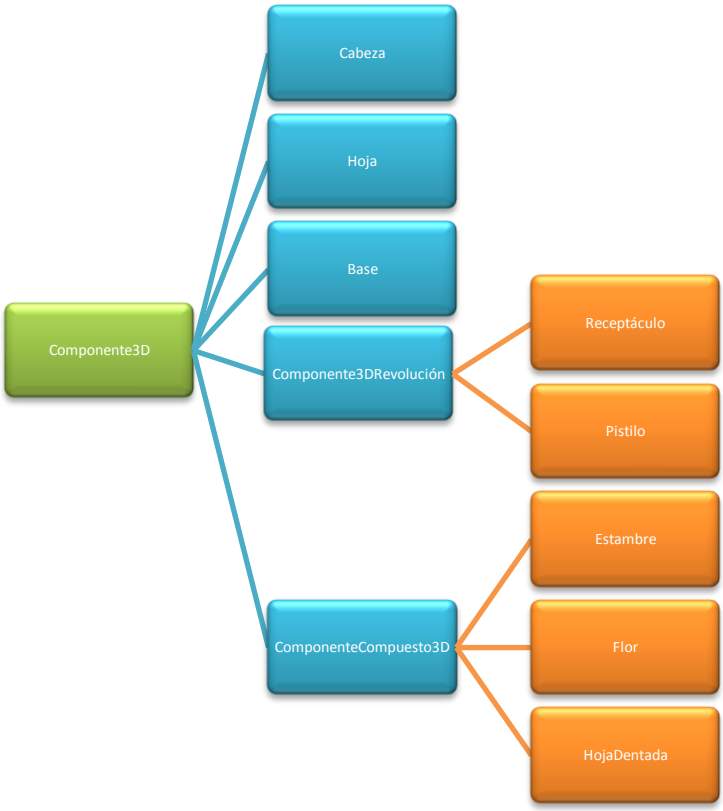


Ilustración 77: Jerarquía de componentes

En el proyecto existen varios casos donde se aplican transformaciones afines. Un ejemplo sencillo es la traslación de componentes al origen de coordenadas. Otras más complejas son:

5.18.2.- TRANSFORMACIONES AFINES EN LA CREACIÓN DEL ESTAMBRE

En la creación del estambre se usa una rotación y una traslación para colocar la cabeza sobre la base. El orden de estas transformaciones es trasladar el punto inferior de la cabeza al punto superior de la base del estambre. Posteriormente se realizan dos rotaciones, la primera sobre el eje Z 90^{o} y después rotar sobre el eje Y otros 90^{o}, ambos giros en sentido de las agujas del reloj. Para más detalles consultar la sección 5.7.

5.18.3.- TRANSFORMACIONES AFINES PARA CONSTRUIR LA FLOR

Las transformaciones se realizan con el objetivo de situar los componentes en el receptáculo. El proceso es el siguiente:

1. Partimos de la información que hemos configurado en el editor estructural. El tamaño se elige por el usuario en el mismo editor.

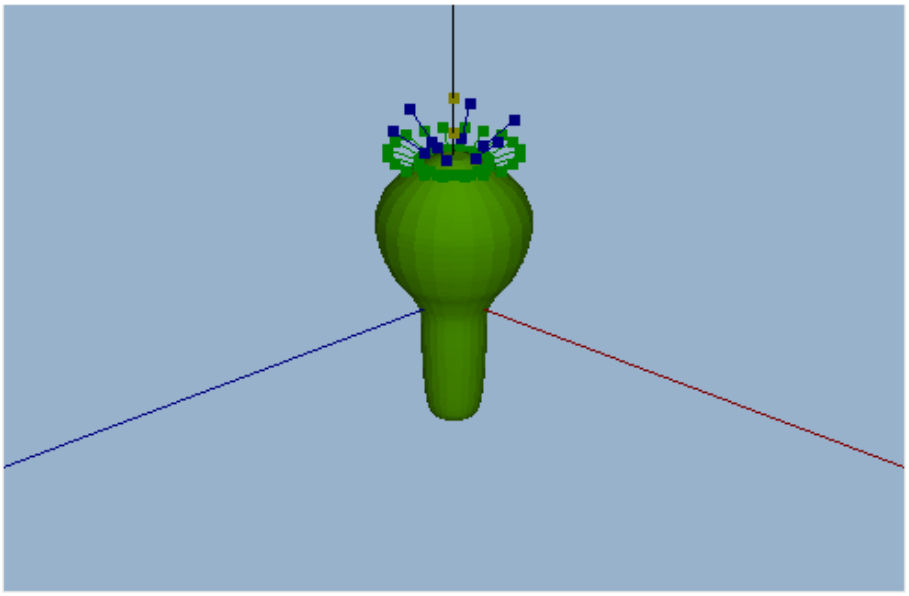


Ilustración 78: El editor estructural se configura de manera que cada línea representa la orientación y la posición de cada componente

Puesto que el proceso es el mismo para colocar cada componente, detallaremos el proceso para uno de ellos. Partimos de la siguiente configuración del editor estructural, suponemos que vamos a añadir tres estambres para el ejemplo.

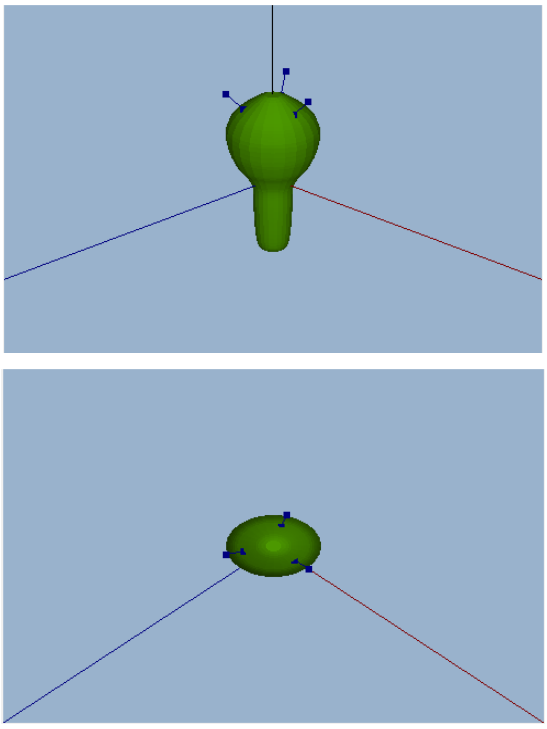


Ilustración 79: Suponemos que vamos a colocar los tres estambres

2. Cargamos el estambre el cual aparece en el origen.

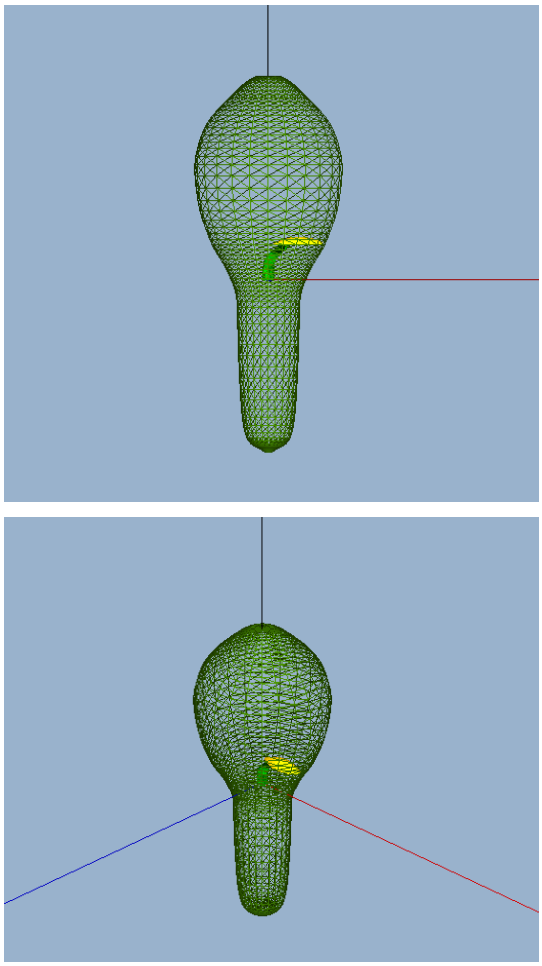


Ilustración 80: Posición original del elemento

El objetivo es trasladarlo al punto indicado en el editor estructural.

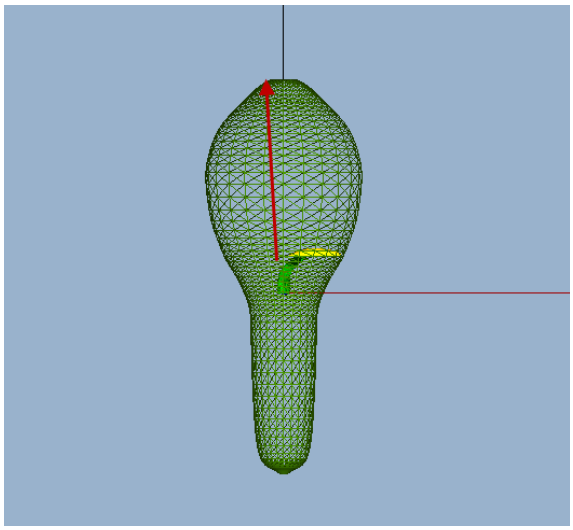


Ilustración 81: Traslación del elemento

3. Una vez trasladado obtenemos el resultado siguiente:

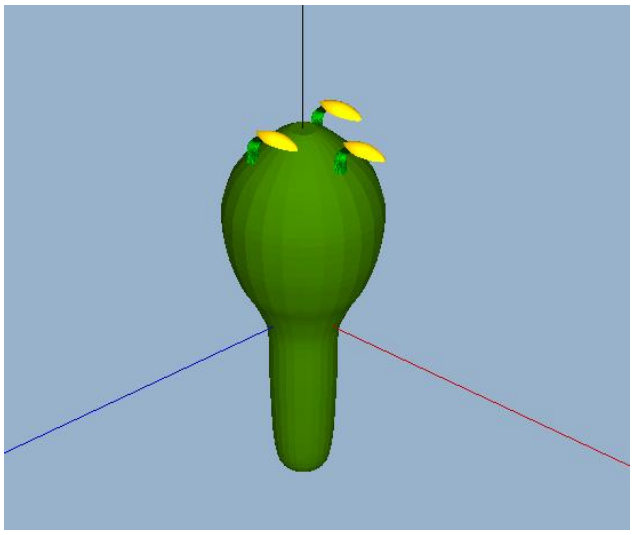


Ilustración 82: Resultado de trasladar los 3 estambres

4. A continuación se escala el objeto para adaptarse al tamaño elegido por el usuario, obteniéndose el siguiente resultado.

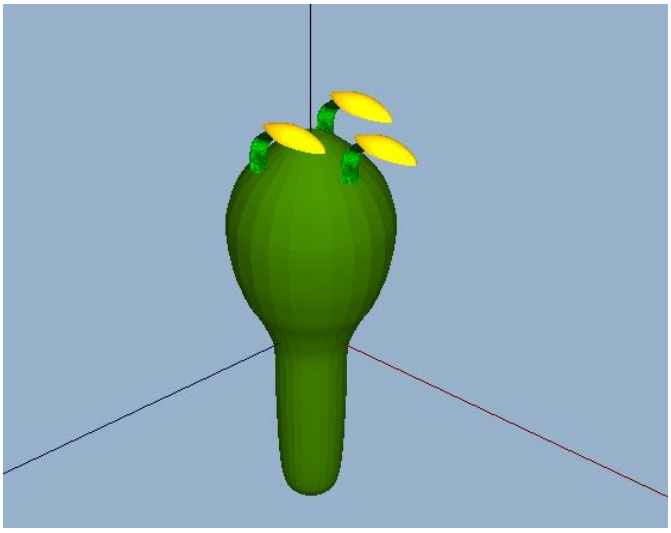


Ilustración 83: Resultado de la escalación

5. Rotamos sobre el eje Y para orientar cada componente.

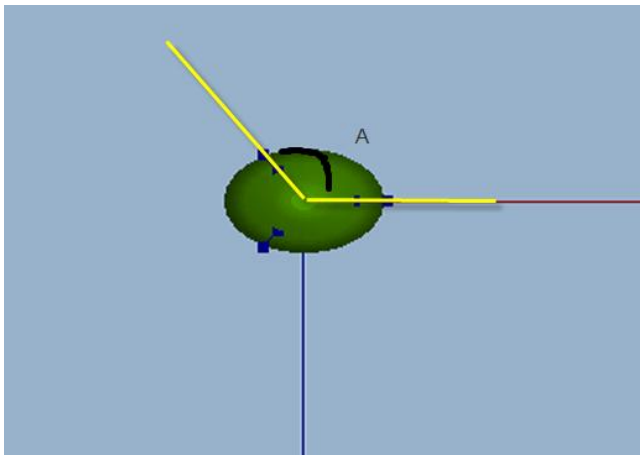


Ilustración 84: Cada componente se rota sobre el eje Y un ángulo A que forma con el eje X en el plano XZ para orientarlo

6. El resultado de la primera rotación es el siguiente:

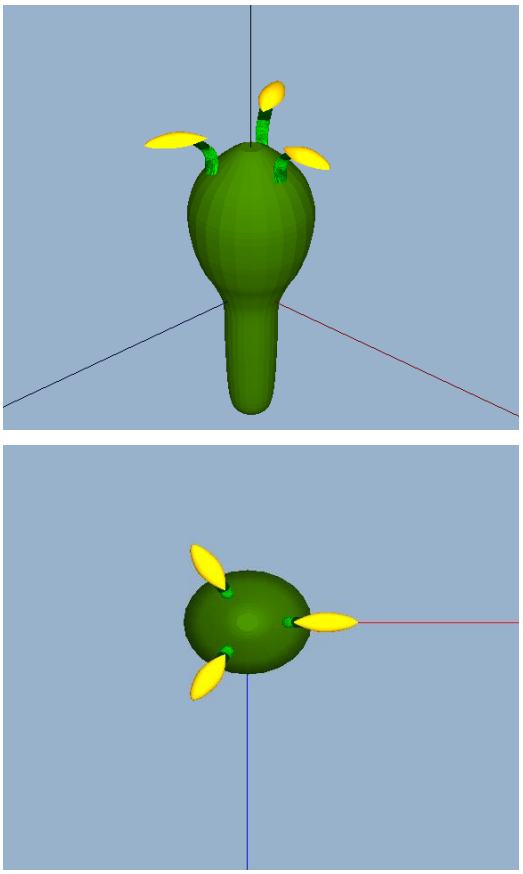


Ilustración 85: Resultado del primer rotado

7. Rotamos sobre el eje Z el ángulo B que forma con el eje X en el plano XY para terminar de colocar el componente.

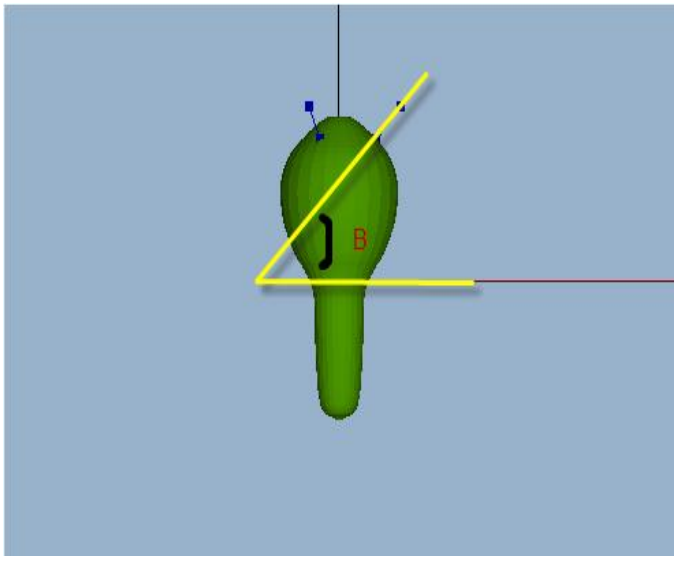


Ilustración 86: Ángulo del segundo rotado

8. El resultado final es:

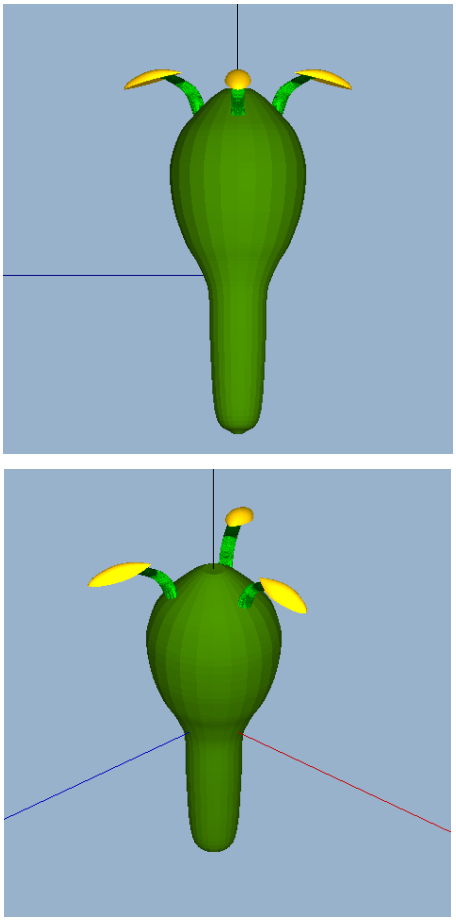


Ilustración 87: Resultado final al aplicar todas las transformaciones afines

9. Si colocamos de la misma manera todos los componentes que aparecen en la ilustración 77 obtendríamos la siguiente flor.

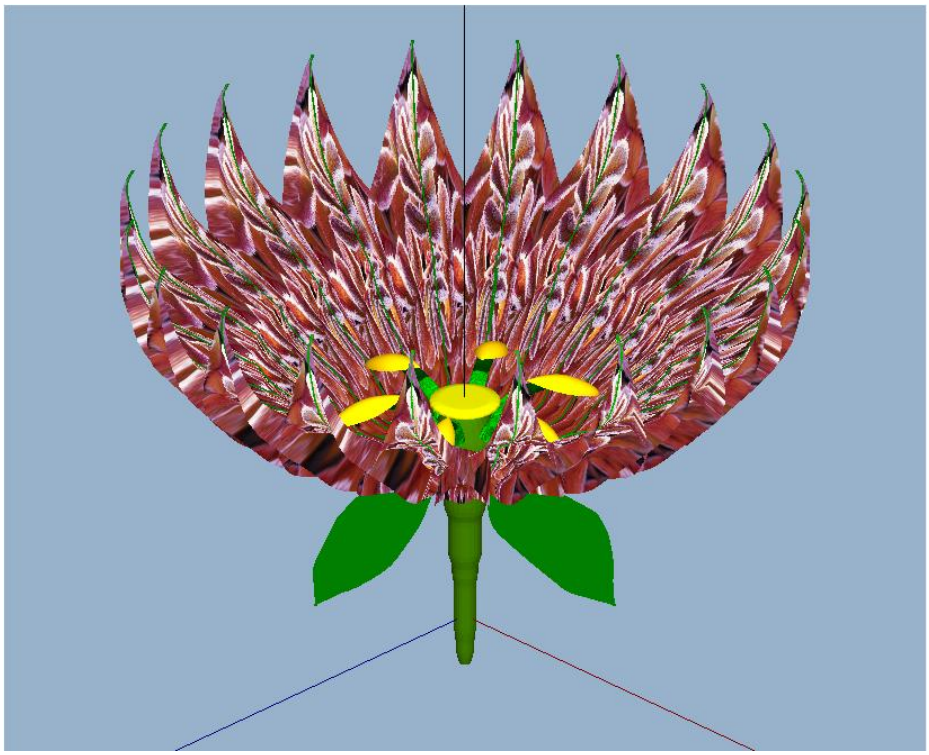


Ilustración 88: Resultado final después de colocar todos los componentes

5.18.4.- ROTADO DEL RATÓN

Con el botón derecho del ratón se puede rotar el elemento como el usuario quiera. Para ello las operaciones se almacenan en una matriz que después se asocia a un componente3D que actúa como raíz y permite ver el objeto completo con los cambios deseados. Para realizar esta rotación hemos utilizado la técnica de "Arcball" que resumimos a continuación.

5.18.4.1.- Fundamentos matemáticos

Cualquier orientación de un cuerpo rígido viene dada por una rotación simple, un giro sobre algún eje respecto a un punto de referencia. Además la combinación de cualquier número de rotaciones puede verse como una rotación simple según el método de Euler.

La ley de combinación depende de los parámetros usados, pero siempre se basa en una geometría esférica simple. Una rotación *R* alrededor de un eje **a,** de ángulo θ puede representarse en una esfera como un arco de longitud ½ θ en el plano perpendicular al vector **a,** con ángulos positivos en sentido contrario a las agujas del reloj alrededor de **a** (Ilustración 88).

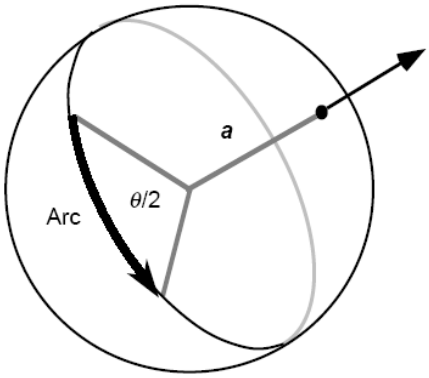


Ilustración 89: Interpretación del arco

Cuando el final del primer arco coincide con el principio del segundo, se forman dos lados de un triángulo esférico. El arco que completa el triángulo, desde el principio del primer arco hasta el fin del segundo, representa una rotación simple (Ilustración 89).

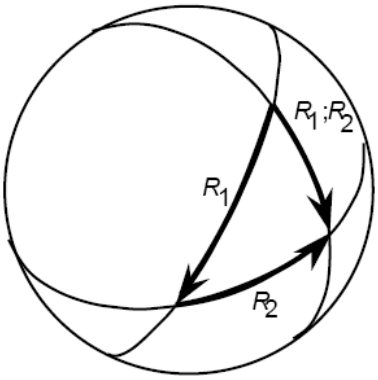


Ilustración 90: Combinación de arcos

5.18.4.1.- Arcball

Supongamos que un objeto es seleccionado en la pantalla. Para cambiar su orientación el usuario dibuja un arco en la proyección de la pantalla sobre una esfera. El arco viene definido por su punto inicial y final, que son dados al pulsar y levantar el ratón (ver figura siguiente).

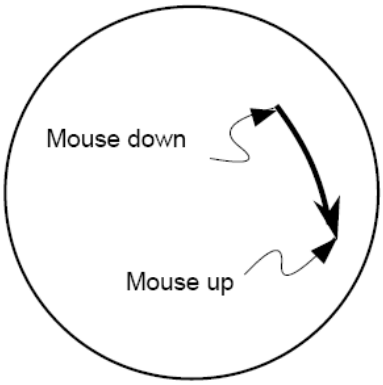


Ilustración 91: Introducción de datos con el ratón

A la vez que movemos el ratón, se toma la posición actual para computar el arco. De esta forma el objeto gira con el ratón. El arco está contenido en un plano definido por los dos puntos y el centro de la esfera. Si los dos puntos caen exactamente uno enfrente del otro, el plano, el plano del arco no está definido. Esto implicaría una rotación de 360 grados, lo que hace que el objeto no cambie, es decir, un punto enfrente de otro en un círculo son "el mismo punto". En consecuencia cuando el ratón se arrastra fuera del círculo en algún punto podemos hacerlo reentrar en su punto de enfrente, lo que se conoce como "wrapping"(ver figura siguiente), es decir no hay límite de rotado.

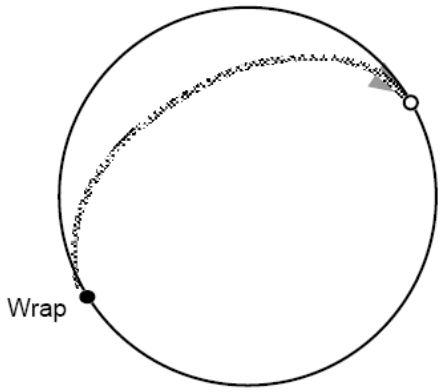


Ilustración 92: Wrapping

5.19.- CÁMARA

5.19.1.- INTRODUCCIÓN

La cámara es una herramienta proporcionada por JOGL, que nos permite ver la escena desde diferentes puntos de vista, poniendo a nuestra disposición una visión en 3D de los objetos.

Esta herramienta necesita ser configurada, indicando la posición y orientación de la cámara, con respecto a la escena, así como el tipo de proyección (ortogonal o perspectiva) que define el volumen de visión. Estos parámetros determinan cómo se proyecta un objeto en la pantalla, y define qué objetos o porciones de ellos se colocan en la imagen final, así como el tamaño de los mismos.

La característica principal de la proyección en perspectiva es que cuanto más alejado está un objeto de la cámara, más pequeño aparece en la imagen final. A diferencia de la proyección en perspectiva, en la ortogonal el tamaño del volumen de visión no cambia desde el plano cercano al lejano, por lo que la distancia a la cámara no afecta al tamaño final de los objetos en la imagen.

La siguiente imagen muestra la posición por defecto de la cámara.

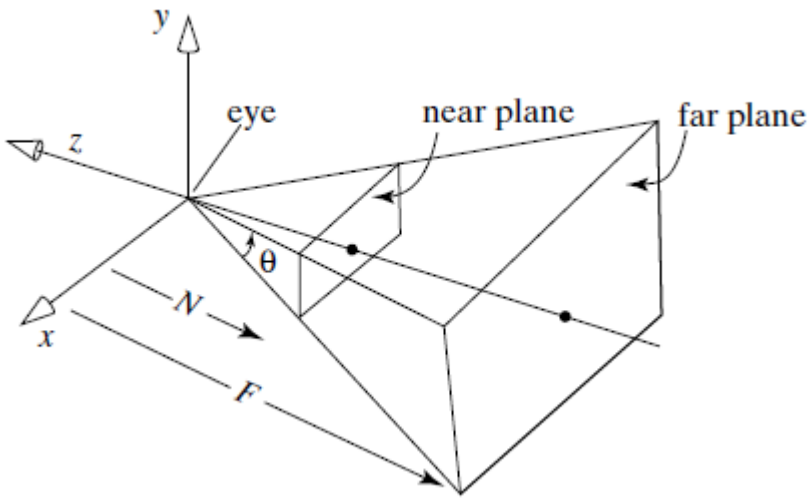


Ilustración 93: Posición por defecto de la cámara

Los siguientes tres parámetros son los que definen la posición y orientación de la cámara, tal y como muestra la siguiente imagen.

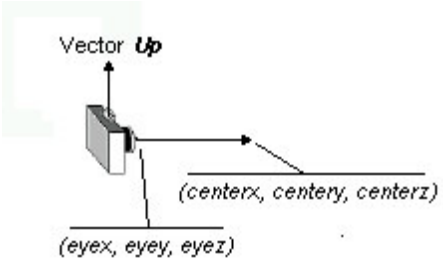


Ilustración 94: Parámetros de la cámara

El punto de vista deseado está especificado por el punto Eye. El punto Center especifica cualquier punto a lo largo de la línea de visión. Y finalmente el vector Up indica qué dirección queda hacia arriba en el volumen de visión.

El sistema de coordenadas que define la cámara tiene su origen en el punto eye, y tres ejes llamados eje-u, eje- v y eje-n que definen su orientación. Las direcciones de estos ejes vienen señaladas por los vectores **u**,**v** y **n** mostrados en la siguiente figura.

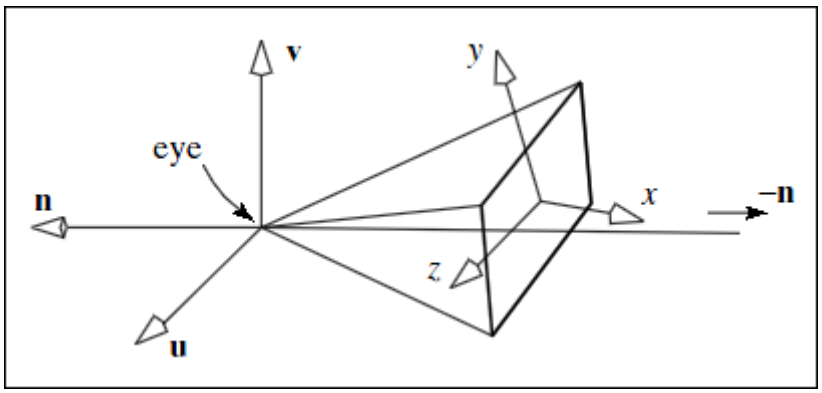


Ilustración 95: Sistema de coordenadas que define la cámara

Podemos pensar en los ejes u, v y n como si fueran los ejes x,y y z de la cámara. Los valores de estos vectores se calculan con las siguientes operaciones:

$$n = eye - look$$

$$u = up \times n$$

$$v = n \times u$$

La siguiente imagen muestra la construcción de estos vectores.

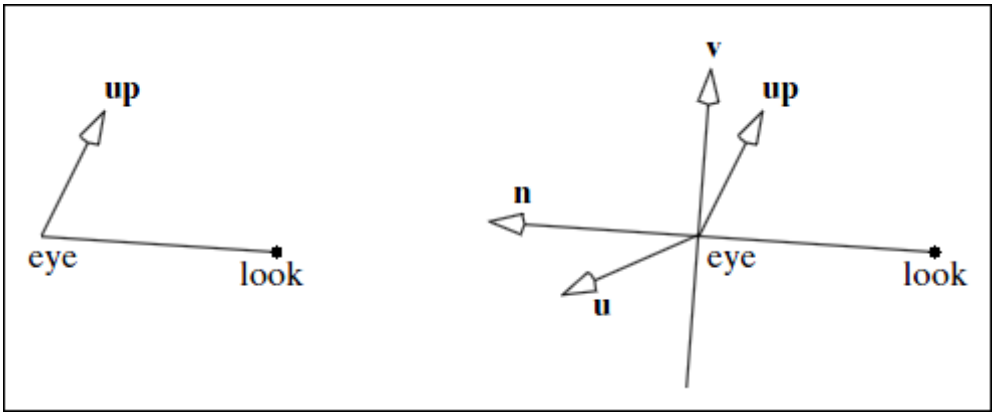


Ilustración 96: Construcción de los vectores

5.19.2.- TIPOS DE CÁMARAS USADAS

Existen 4 tipos de cámaras para cada uno de los editores de los componentes que forman la flor. Estas cámaras tienen proyección ortogonal.

Tres de esas cámaras se corresponden con los visores incorporadas en las pestañas, permitiéndonos ver desde distintos puntos de vista el elemento que estamos diseñando. Estas cámaras son configurables en un archivo de propiedades, permitiendo al usuario modificarlas desde la interfaz de la aplicación.

La cuarta cámara es la que proporciona la vista principal de la escena donde se crean los nuevos elementos (su posición es diferente en cada editor). Esta cámara se puede mover por la escena mediante el teclado.

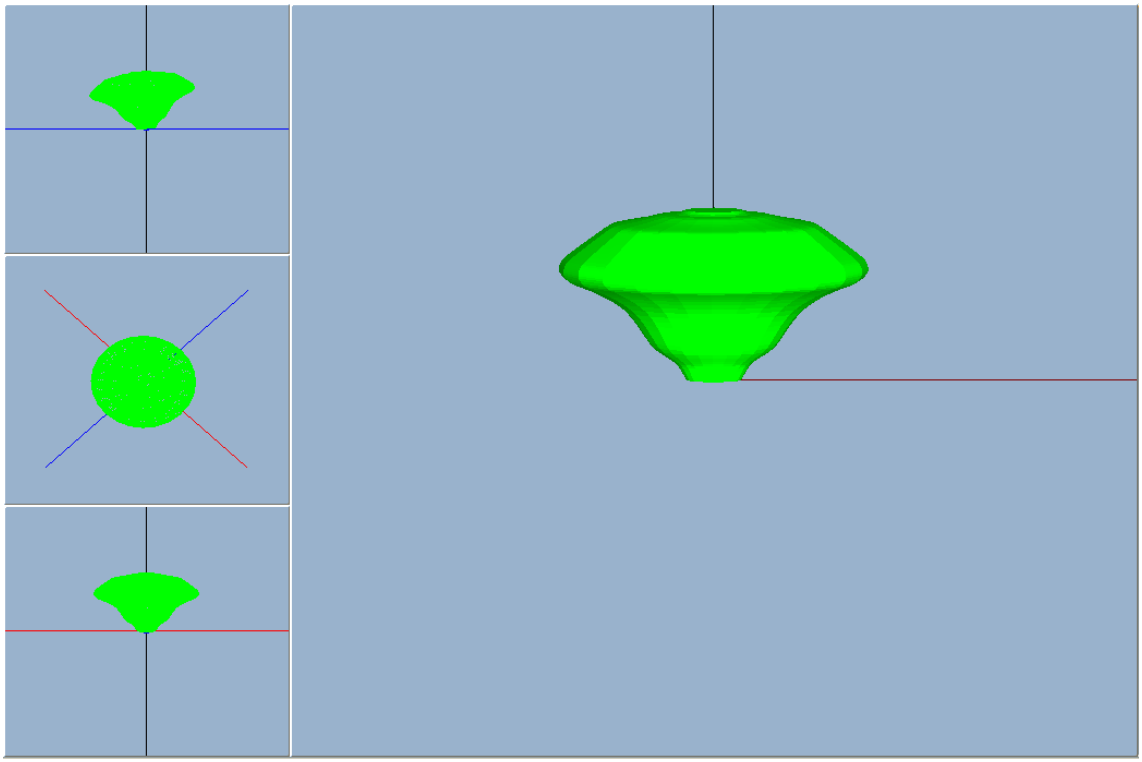


Ilustración 97: Cámaras de la escena

Las cuatro cámaras usadas por defecto están situadas al frente, en una esquina, en un lateral y encima. En las pestañas de edición de pétalos y sépalos la vista principal es la frontal, en el resto de pestañas la vista principal es la cámara situada en una esquina.

5.19.3.- MOVIMIENTOS DE LA CÁMARA

En la aplicación permitimos cualquier tipo de movimiento de la cámara principal de cada uno de los editores, de forma que se pueden distinguir los movimientos de ángulo de la cámara (rotación) y los de traslación. Los movimientos de rotación se llaman pitch, yaw y roll como podemos ver en la siguiente figura.

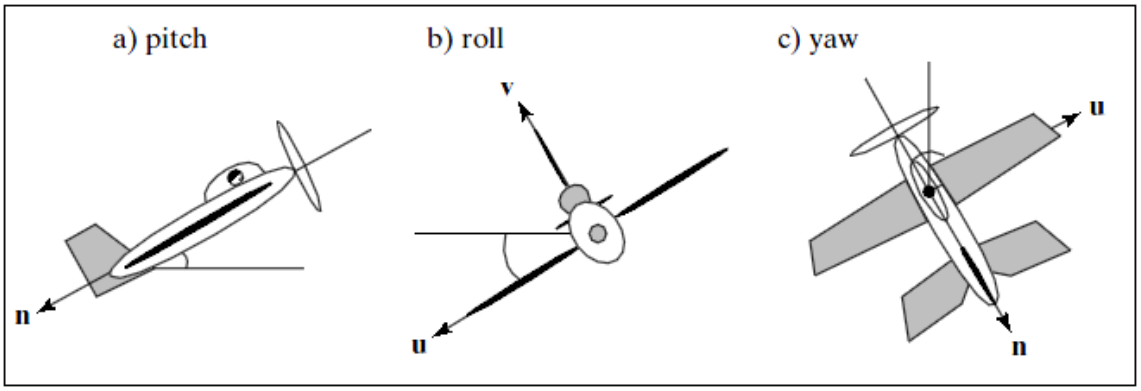


Ilustración 98: Movimientos de rotación de la cámara

De esta forma, dotamos al usuario de un control total a la hora de ver el estado del objeto que esté editando, así como de los distintos puntos de vista que proporcionan los visores.

6.- TECNOLOGÍAS

Para realizar el proyecto hemos usado diversas herramientas. Todas son de libre distribución salvo la utilizada para el autorun que emplea una versión de prueba. El software desarrollado está en JAVA, utilizando el entorno de ECLIPSE. Como API gráfica hemos utilizado la librería JOGL que es una versión de OPENGL para JAVA. A la hora de trabajar con XML hemos utilizado la librería JAXB para JAVA.

Para el instalador hemos utilizamos la herramienta NSIS, mientras que para el autorun hemos elegido AUTO PLAY MENU BUILDER.

Por otra parte, para visualizar archivos PLY hemos utilizado los programas PARAVIEW y HEADUS PLY TOOLS.

El control de versiones tanto del código como de la documentación se ha realizado mediante la herramienta SUBVERSION, utilizando un plugin para ECLIPSE que gestiona el historial de archivos almacenados en un repositorio. El que hemos utilizado concretamente ha sido GOOGLECODE.

Para la comunicación entre los distintos miembros del grupo hemos utilizado GOOGLEGROUPS.

7.- ESTRUCTURA, ORGANIZACIÓN E IMPLEMENTACIÓN.

7.1.- ESTRUCTURA DE LA CARPETA "RESOURCES"

La carpeta "resources" usada por la aplicación es el directorio donde se almacenan los archivos necesarios para su funcionamiento, además de actuar como carpeta temporal en tiempo de ejecución para soportar algunas operaciones con ficheros.

A continuación se detallan cada uno de los subdirectorios y se explica su función.

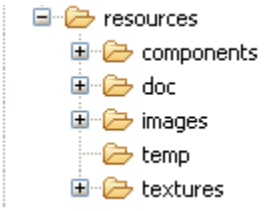


Ilustración 99 : Jerarquía de la carpeta resources

7.1.1 COMPONENTS

En el directorio componentes se guardan los componentes almacenados en la galería para su posterior uso en la edición de una flor. Existe una carpeta para cada tipo de objeto que se puede guardar en la aplicación.

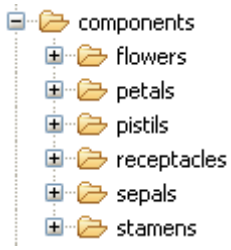


Ilustración 100 : Jerarquía de la carpeta components

En cada uno de estos subdirectorios se almacenarán, por cada elemento guardado en la galería su correspondiente archivo .flw (ver sección 7.2) y un directorio con el mismo nombre que el archivo que contiene la información representada en él. Este directorio es necesario para cargar y visualizar desde la galería en la aplicación. Al añadir un componente a la galería desde la aplicación éste se guardará por defecto con la fecha de creación como nombre. Si se desea darle un nombre al componente recién creado es necesario guardarlo en esta carpeta desde la opción *Guardar Como*.

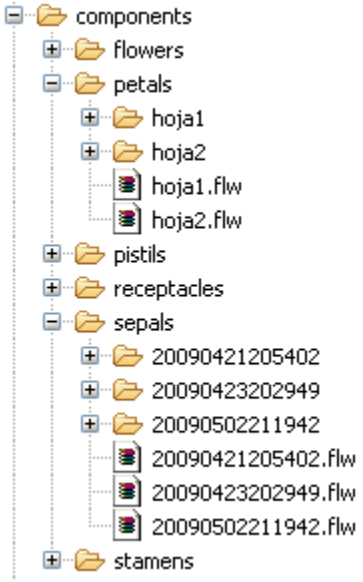


Ilustración 101 : Contenido de cada componente

7.1.2.- DOC

En esta carpeta se guarda documentación del proyecto y los manuales de usuario.

7.1.3.- IMAGES

Este directorio contiene las distintas imágenes usadas por la interfaz de la aplicación.

7.1.4.- TEMP

Es una carpeta temporal que se crea al ejecutar la aplicación y se borra al cerrar la misma. En ella se realizan operaciones con archivos (comprimido/descomprimido…) y se usa como almacenamiento temporal de los archivos abiertos.

7.1.5.- TEXTURES

Tiene una estructura similar a la carpeta “components”, con un subdirectorio para cada uno de los elementos que forman la flor, pero no para la flor en sí. En esta carpeta se almacenan las texturas añadidas por el usuario mediante la interfaz de creación de cada componente, es decir, se guarda cualquier imagen agregada desde la pestaña de texturas.

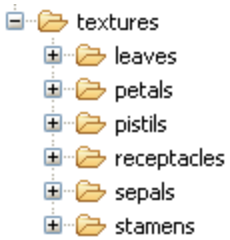


Ilustración 102 : Estructura de la capeta textures

7.2.- GUARDADO DE ARCHIVOS

La aplicación dispone de un formato de archivo propio cuya extensión es .flw. De esta forma los objetos diseñados se pueden guardar en este formato para su posterior reutilización y/o edición.

El formato de archivo .flw es el mismo utilizado para crear los conocidos archivos .zip, de forma que nos permite almacenar en formato comprimido la información de cada uno de los objetos diseñados en la aplicación, haciendo que los datos ocupen poco espacio en un único fichero. Se trata de archivos que pueden ser descomprimidos con cualquier programa específico para esta función.

Estos ficheros contienen todos los datos necesarios para reconstruir el objeto diseñado mediante la aplicación, es decir, guardan la estructura, la forma, el color y las texturas usadas en su creación, permitiendo su reconstrucción exacta. Su contenido concreto depende del objeto almacenado, soportando componentes independientes y la flor completa.

Los ficheros comprimidos correspondientes para un componente de la flor son los siguientes:

- Un **archivo** de datos que contiene la información necesaria para reconstruir la estructura y forma del componente.
- Un **archivo** de propiedades que permite almacenar las distintas opciones usadas en el editor, como la forma en que se aplican las texturas, la textura utilizada...
- La **imagen** en formato .jpg que corresponde con captura de pantalla del editor en el instante en el que se guardó el componente para su incorporación en la galería de componentes. Este archivo se guarda con extensión **.mini** para facilitar su gestión.
- Un **archivo** de imagen correspondiente a la textura aplicada en caso de haber seleccionado esta opción.

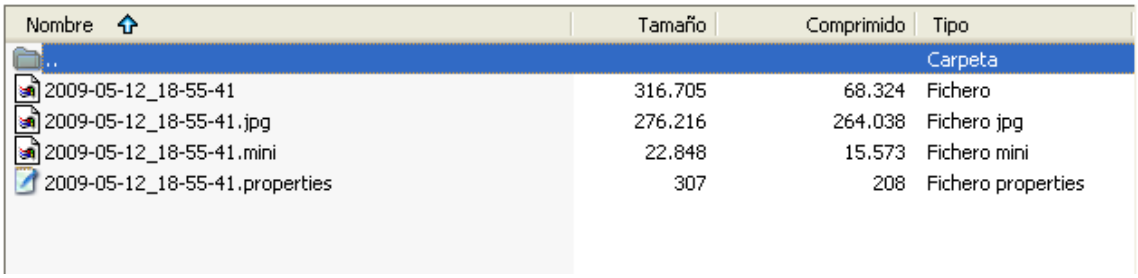


Ilustración 103 : Datos guardados para un componente

Los ficheros comprimidos para la flor son los siguientes:

- Un fichero xml con la estructura de la flor(ver apartado de XML de la flor)
- Para cada uno de los distintos representantes de los componentes que forman la flor, se guardan los mismos ficheros que hemos descrito para los componentes. En caso de existir varios componentes iguales, sólo se guarda la información de uno de ellos, ya que el xml indicará el número de veces que se usa y su localización.

Nombre	Tamaño	Comprimido	Tipo	Modificado	CRC32
..			Folder		
2009-05-20_18-27-38	317.931	62.599	Fichero	10/06/2009 13:00	85C5B025
2009-05-20_18-27-38.jpg	241.400	239.728	Imagen JPEG	10/06/2009 13:00	2A0686D3
2009-05-20_18-27-38.mini	17.114	9.142	Fichero mini	10/06/2009 13:00	5DD681CA
2009-05-20_18-27-38.properties	214	142	Fichero properties	10/06/2009 13:00	D659441F
20090520183035	318.597	67.711	Fichero	10/06/2009 13:00	6E8A0F0E
20090520183035.jpg	123.724	123.706	Imagen JPEG	10/06/2009 13:00	CF054E6E
20090520183035.mini	13.343	4.866	Fichero mini	10/06/2009 13:00	37368215
20090520183035.properties	213	143	Fichero properties	10/06/2009 13:00	9EB1D413
200905201837	592	156	Fichero	10/06/2009 13:00	AA92B600
20090520184957	440	319	Fichero	10/06/2009 13:00	BE0A2BA3
20090520184957.mini	11.725	4.229	Fichero mini	10/06/2009 13:00	5AEA767D
20090520184957.properties	70	63	Fichero properties	10/06/2009 13:00	B3774844
flor azul.properties	118	90	Fichero properties	10/06/2009 13:00	0BA2BA63
flor azul.xml	3.674	676	XML Document	10/06/2009 13:00	42F06DA5

Ilustración 104 : Datos guardados para la flor

7.2.1.- XML DE LA FLOR

Se utiliza un archivo xml como archivo de control para guardar/cargar la estructura de la flor. Este archivo contiene tanto los nombres de los archivos de datos de los componentes que forman la flor como los datos relativos a su posición en la misma (coordenadas en el editor estructural, ángulos de rotación e inclinación…). El archivo xml viene definido por el siguiente esquema.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!-- definition of simple type elements -->
  <xs:element name="tamano" type="xs:int"/>
  <xs:element name="radio" type="xs:double"/>
  <xs:element name="anguloY" type="xs:double"/>
  <xs:element name="anguloZ" type="xs:double"/>
  <xs:element name="file" type="xs:string"/>
  <xs:element name="x" type="xs:double"/>
  <xs:element name="y" type="xs:double"/>
  <xs:element name="z" type="xs:double"/>
  <!-- definition of attributes -->
  <xs:attribute name="receptaculoFile" type="xs:string"/>
  <!-- definition of complex type elements -->
  <xs:element name="dataCirculo-xml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="tamano"/>
        <xs:element ref="radio"/>
        <xs:element ref="coordenadas2D-xml" minOccurs="0" maxOccurs="unbounded"/>
        <!-- the simple type elements are referenced using
        the "ref" attribute                          -->
        <!-- the definition of the cardinality is done
        when the elements are referenced          -->
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="dataFlowers-xml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="dataCirculo-xml" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="receptaculoFile"/>
    </xs:complexType>
  </xs:element>
    <xs:element name="coordenadas2D-xml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="puntoVector-xml"/>
        <xs:element ref="anguloY"/>
        <xs:element ref="anguloZ"/>
        <xs:element ref="file"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
      <xs:element name="puntoVector-xml">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="x"/>
        <xs:element ref="y"/>
        <xs:element ref="z"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ilustración 105 : Esquema que define el archivo xml donde se guarda la flor

Un ejemplo de archivo xml para una flor se muestra en el siguiente fragmento, obsérvese cómo para cada componente de las circunferencias del editor estructural (cada etiqueta <dataCirculo-xml> representa un componente, por este orden: sépalos, pétalos, estambres y pistilos) aparecen datos como su radio, su tamaño, su posición 2D y el nombre del archivo que lo carga. Aunque no son exactamente iguales, la estructura del xml está relacionada con la estructura dataFlowers que se usa en la implementación de la aplicación para guardar en todo momento la correspondencia entre 2D y 3D de los componentes así como su situación en la flor, inclinación, etc. (ver sección 10.1).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataFlowers-xml receptaculoFile="temp">
  <dataCirculo-xml>
    <tamano>5</tamano>
    <radio>56.3636360168457</radio>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>7.105427357601002E-15</x>
        <y>56.3636360168457</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>-157.61966864375268</anguloY>
      <anguloZ>4.71238898038469</anguloZ>
      <file>20090505182454</file>
    </coordenadas2D-xml>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>-56.36363601684571</x>
        <y>7.105427357601002E-15</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>-157.61966864375268</anguloY>
      <anguloZ>3.141592653589793</anguloZ>
      <file>20090505182454</file>
    </coordenadas2D-xml>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>-2.1316282072803006E-14</x>
        <y>-56.36363601684571</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>-157.61966864375268</anguloY>
      <anguloZ>1.570796326794897</anguloZ>
      <file>20090505182454</file>
    </coordenadas2D-xml>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>56.36363601684568</x>
        <y>-1.4210854715202004E-14</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>337.61966864375273</anguloY>
      <anguloZ>0.0</anguloZ>
      <file>20090505182454</file>
    </coordenadas2D-xml>
  </dataCirculo-xml>
  <dataCirculo-xml>
    <tamano>5</tamano>
    <radio>36.969696044921875</radio>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>35.160270328974114</x>
        <y>11.424264354757144</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>16.28817114590801</anguloY>
      <anguloZ>5.969026041820607</anguloZ>
      <file>2009-05-05_18-16-50</file>
    </coordenadas2D-xml>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>29.909112377218072</x>
        <y>21.73024211694046</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>16.28817114590801</anguloY>
      <anguloZ>5.654866776461628</anguloZ>
      <file>2009-05-05_18-16-50</file>
    </coordenadas2D-xml>
    <coordenadas2D-xml>
      <puntoVector-xml>
        <x>21.730242116940445</x>
        <y>29.90911237721809</y>
        <z>0.0</z>
      </puntoVector-xml>
      <anguloY>16.288171145908</anguloY>
      <anguloZ>5.340707511102648</anguloZ>
      <file>2009-05-05_18-16-50</file>
    </coordenadas2D-xml>
```

Ilustración 106 : Archivo .xml que muestra cómo se guarda una flor

8.- FORMATO DE LOS ARCHIVOS PLY

Aunque los archivos PLY permiten guardar mucha información para definir objetos 3D, sólo decidimos usar la que realmente nos resultaba de utilidad. Así elegimos guardar únicamente los puntos que forman la escena y las caras triangulares tal y como se definieron en la creación de las mallas.

La aplicación permite guardar los archivos tanto en formato binario como en ASCII. Esto es posible porque existe una cabecera que indica la información que se encuentra en el archivo la manera de leerla.

Un ejemplo de archivo PLY en formato ASII de nuestra aplicación podría ser el siguiente:

```
ply
format ascii 1.0
comment made by flower designer 3D
comment this file is a class componentes.flor.Flor
element vertex 68226
property float x
property float y
property float z
element face 131952
property list uchar int vertex_index
end_header
0.0 68.75 0.0
3.0612245 68.75 0.0
3.6734693 68.125 0.0
4.285714 67.5 0.0
4.846939 66.796875 0.0
5.3061223 65.9375 0.0
5.714286 64.921875 0.0
6.122449 63.75 0.0
6.5306125 62.578125 0.0
6.9387755 61.5625 0.0
7.295918 60.46875 0.0
7.55102 59.0625 0.0
7.7040815 57.421875 0.0
7.755102 55.625 0.0
7.653061 53.90625 0.0
7.3469386 52.5 0.0
7.0408163 51.171875 0.0
6.9387755 49.6875 0.0
```

Conseguir los puntos que definen la escena no es trivial, el proceso de obtención de estos puntos se ha descrito en la sección *5.17.*

9.- ESTRUCTURA DE PAQUETES Y CLASES

A continuación se detalla la estructura de paquetes y las clases utilizadas para el desarrollo de la aplicación.

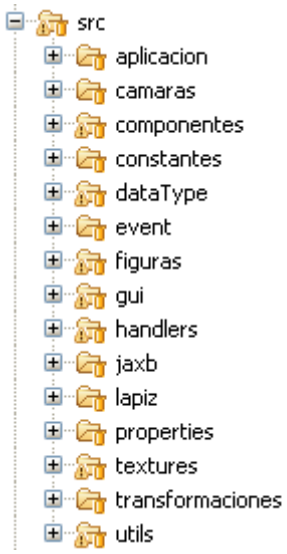


Ilustración 107 : Estructura general de paquetes

9.1.- APLICACIÓN

Contiene las clases necesarias para la ejecución de la aplicación.

- App → es la clase principal de la aplicación, donde se encuentra el método "main".
- Vista → Representa la interfaz gráfica de usuario, sirve para iniciar la parte visual de la aplicación.
- Controlador → Coordina la vista con el modelo de datos de la aplicación.
- Frames → Ventanas de la aplicación.
    - Configuración → Ventana de configuración de la aplicación.
    - GaleríaReceptáculos → Ventana de la galería de Receptáculos que se abre en la pantalla principal en el panel de edición rápida del receptáculo.
    - PantallaInicial → Pantalla inicial que se muestra al iniciar la aplicación.

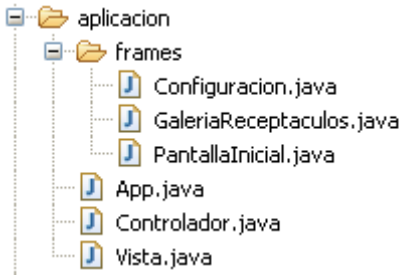


Ilustración 108 : Estructura del paquete aplicación

9.2.- CÁMARAS

- Cámara → Contiene la lógica necesaria para el manejo de las cámaras usadas en la aplicación.

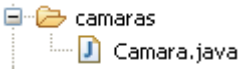


Ilustración 109 : Estructura del paquete cámara

9.3.- COMPONENTES

Modelo de datos para cada uno de los componentes que forman la flor. Posee estructura jerárquica:

- Componente3D → Clase genérica que representa cualquier componente. Reúne las partes comunes a todos ellos (color, matriz de trasformación, malla…).
- ComponenteCompuesto3D → Representa un componente formado por múltiples componentes.
- Componente3DRevolución → Representa a un Componente3D creado por revolución (Receptáculo y Pistilo).
- Flor → Contiene los componentes que forman la flor.
    - Estambre → ComponenteCompuesto3D formado por Base y Cabeza que representa el modelo de datos para el estambre.
    - Pistilo → Componente3DRevolución que representa el modelo de datos para el pistilo.
    - Receptáculo → Componente3DRevolución que representa el modelo de datos para el receptáculo.
    - Hoja → Componente3D que representa modelo de datos para una hoja.
    - HojaDentada → ComponenteCompuesto3D que representa el modelo de datos de una hoja que se puede dentar.
    - Flor → ComponenteCompuesto3D que representa el modelo de datos de la flor.
    - Estambres → Paquete con los componentes que forman el estambre.
        - Base → Componente3D que representa el modelo de datos del filamento del estambre.
        - Cabeza → Componente3D que representa el modelo de datos de la antera del estambre.

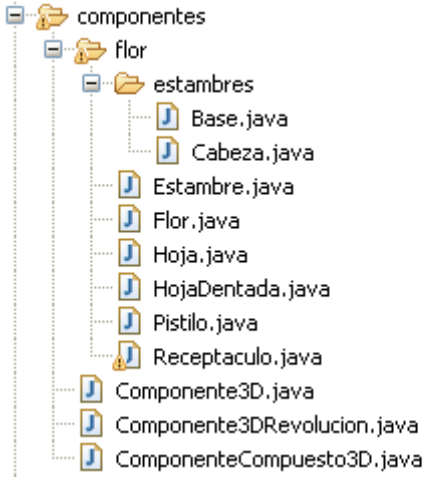


Ilustración 110 : Estructura del paquete componentes

9.4.- CONSTANTES

- Consts → Clase con constantes que se usan en la implementación.

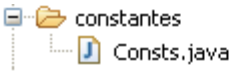


Ilustración 111 : Estructura del paquete constantes

9.5.- DATATYPE

Contiene los tipos de datos propios utilizados en la implementación.

- ContornoHoja → Almacena las listas de puntos que representan los trazos que definen una hoja.
- Coordenadas3D → Contiene la información para situar un punto en el receptáculo en tres dimensiones (ver sección 10.1).
- DataCirculo → Contiene la información relativa a la distribución de los elementos pertenecientes a cada tipo de componente en el editor estructural (ver sección 10.1).
- DataFlower → Contiene el estado en el que se encuentra la creación de la flor (ver sección 10.1).
- GLInt → Tipo propio para representar valores enteros.
- InterseccionRectaSegmento → Sirve para almacenar el resultado obtenido al intersecar una recta y un segmento.
- Situacion3D → Sirve para dibujar en 3D los puntos de inserción de los componentes en el receptáculo.

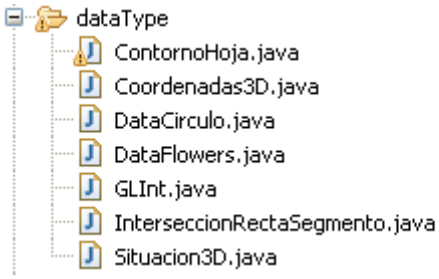


Ilustración 112 : Estructura del paquete dataType

9.6.- EVENT

Contiene clases que representan eventos para cambiar el estado de los componentes.

- ChangeHoja → Eventos para cambiar de edición de hoja a modificación de hoja en el editor geométrico.
  - ChangeHoja
  - ChangeHojaListenerInterface
  - EventChangeHoja
- NotifyDentacionesHoja → Eventos para añadir denticiones a la hoja.
  - EventNotifyDentations
  - NotifyDentations
  - NotifyDentationsListenerInterface
- NotifyTexture → Eventos para añadir texturas a los componentes en el editor geométrico.
  - EventNotifyTexture
  - NotifyTexture
  - NotifyTextureListenerInterface

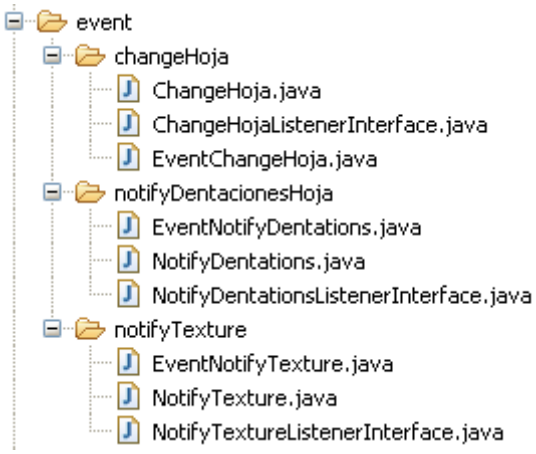


Ilustración 113 : Estructura del paquete event

9.7.- FIGURAS

Contiene clases que representan los elementos geométricos y las mallas.

- Malla → Clases necesarias para la creación de una malla, así como los distintos tipos de mallas.
  - Cara → modelo de datos para la cara de una malla.
  - Malla → modelo de datos genérico para una malla.
  - MallaContornoSpline → representa la malla de la base del estambre (ver sección 5.6).
  - MallaHoja → representa la malla de la hoja (ver sección 5.8).
  - MallaMediaRevolucion → representa la malla de la cabeza del estambre (ver sección 5.7).
  - MallaPorSplineYRevolucion → representa una malla generada por revolución (pistilo y receptáculo). (ver sección 5.5).
  - VerticeNormal → Clase que asocia los vértices que forman una cara con su normal correspondiente.
- BSplines → Clase que representa una curva B-spline y sus operaciones asociadas (ver apéndice A3.1).
- Circulo → Representa una circunferencia.
- Figura → Clase padre de todos los polígonos.
- PuntoVector → Clase que puede representar un punto o un vector en tres dimensiones. Contiene operaciones entre puntos y vectores.
- Recta → Representa una recta.
- RectaRL → Contiene las funciones para crear una recta mediante regresión lineal.
- Segmento → Representa un segmento.

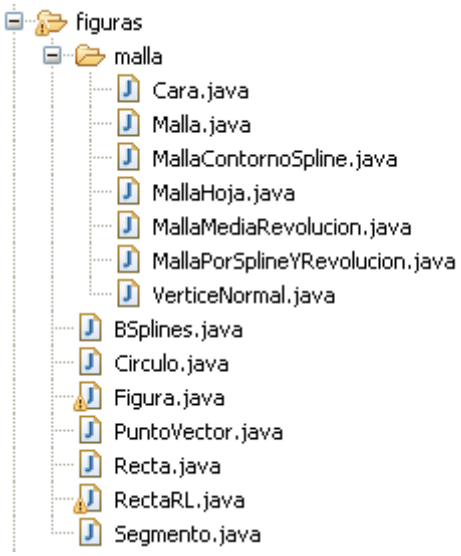


Ilustración 114 : Estructura del paquete figuras

9.8.- GUI

9.8.1.- GLJPANELS

Contienen clases que representan los distintos tipos de paneles (canvas) que aparecen en la aplicación.

- GLJPanelCurvas → Panel con características comunes a aquellos que permiten la introducción de trazos por parte del usuario.
- GLJPanelEstambres → Panel para la creación de estambres.
- GLJPanelHoja → Panel para la creación de pétalos y sépalos.
- GLJPanelNW → Panel padre de todos.
- GLJPanelPistilo → Panel para la creación de pistilos.
- GLJPanelReceptaculo → Panel para la creación de receptáculos.

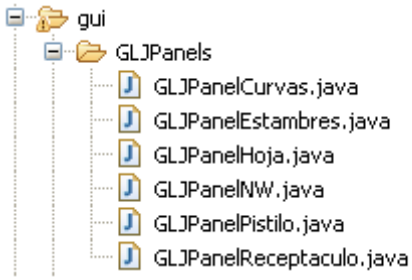


Ilustración 115 : Estructura del paquete GLJPanels

9.8.2.- GLLISTENERS

Contiene clases que gestionan los eventos de los canvas.

- GLCirculos → Gestores de eventos para las circunferencias que representan la distribución de componentes en 2D.
    - Círculos
    - CirculosLib
    - CirculosResize
- GLReceptáculos → Gestores de eventos para los distintos canvas donde aparece un receptáculo.
    - GLListenerReceptaculo → Clase con características comunes a todos los listeners del receptáculo.
    - GLListenerReceptaculoCircules → Gestiona los eventos en el canvas del receptáculo de la pestaña del editor estructural.
    - GLListenerReceptaculoDesign → Gestiona los eventos del canvas del editor geométrico del receptáculo.
    - GLListenerReceptaculoFlower → Gestiona los eventos del canvas de la visión global de la flor en tres dimensiones. Contiene métodos para colocar en el receptáculo todos los componentes que forman la flor de acuerdo con los parámetros elegidos en el editor estructural de la flor.

- GLListener → Gestión de eventos de los canvas en 3D. Puede considerarse como un esqueleto en tres dimensiones para dibujar.
- GLListener2D → Gestión de eventos de los canvas en 2D. Puede considerarse como un esqueleto en dos dimensiones para dibujar.
- GLListenerCreacionHoja → Gestión de eventos para la creación de la hoja en GLJPanelHoja.
- GLListenerCurvarHoja → Gestión de eventos para curvar la hoja en GLJPanelHoja.
- GLListenerCurvas → Gestión de eventos relacionados con la introdución de trazos en GLJPanelCurvas.
- GLListenerEstambres → Gestión de eventos para la creación del estambre en GLJPanelEstambre.
- GLListenerPistilo→ Gestión de eventos para la creación del pistilo en GLJPanelPistilo.
- GLListenerVisor → Gestiona los visores del editor geométrico.

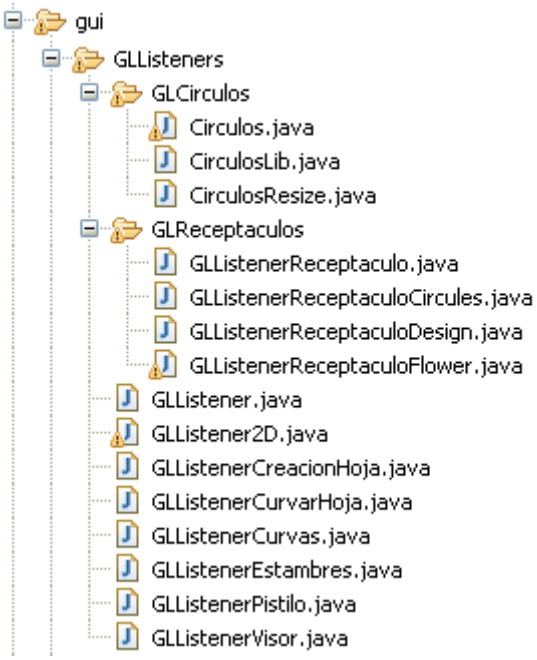


Ilustración 116 : Estructura del paquete GLListeners

9.8.3.- JMENUBAR

Barra de menús

- JMenuBarMain
- jmenu → Contiene la clase JMenuApp que representa cada menú y las clases que la extienden y conforman los menús de la aplicación.
    - JMenuApp
    - JMenuArchivo
    - JMenuAyuda
    - JMenuCamara
    - JMenuEstambre
    - JMenuFlor

- JMenuHerramientas
- JMenuModo
- JMenuPetalo
- JMenuPistilo
- JMenuReceptaculo
- JMenuSepalo

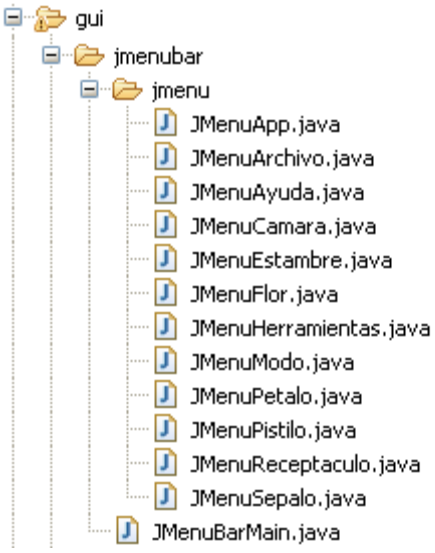


Ilustración 117 : Estructura del paquete jmenubar

9.8.4.- JPANELS

Clases que representan cada uno de los paneles que incluyen botones de interacción.

- JPanelCurvas
- JPanelGalería
- JPanelGaleríaFotos
- JPanelGaleríaFotosReceptaculo
- JPanelReceptaculo
- Settings

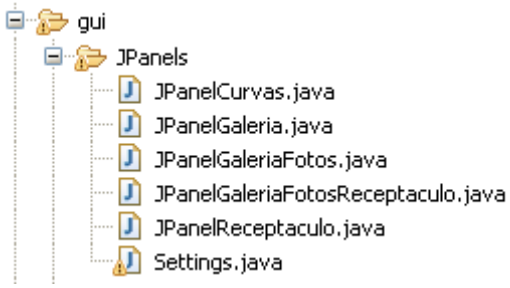


Ilustración 118 : Estructura del paquete JPanels

9.8.5.- jTab

Contiene las clases relativas a la creación y gestión de las distintas pestañas de la aplicación.

- Common
  - JTabTextureOptions → Pestaña de opciones de texturas.
- Estambre
  - JTabObjectEstambre → Pestaña del editor geométrico del estambre.
- Flower → Pestaña flor
  - JTabCirculos → Vista del editor estructural de la flor.
  - JTabFlor → Vista de la flor.
- Pistilo
  - JTabObjectPistilo→ Pestaña del editor geométrico del pistilo.
- Leaf
  - JTabDentationOption → Pestaña de opciones para dentar la hoja.
  - JTabObjectLeaf → Pestaña del editor geométrico del pétalo.
  - JTabObjectSepal → Pestaña del editor geométrico del sépalo.
- Receptaculo
  - JTabObjectReceptaculo → Pestaña del editor geométrico del receptáculo.

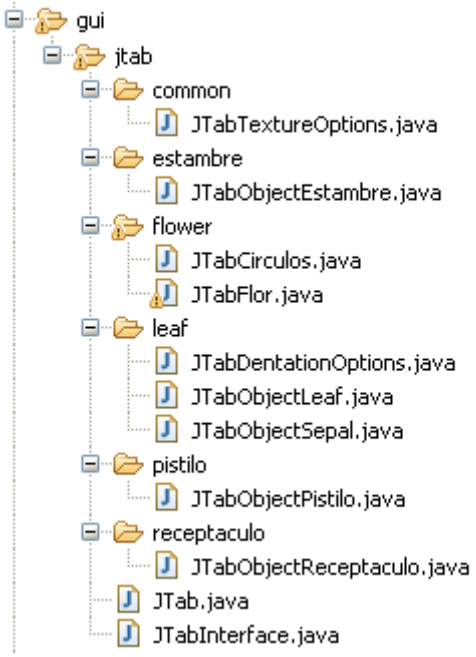


Ilustración 119 : Estructura del paquete jTab

9.9.- HANDLERS

- BSplinesHandler → manejador para el control de las B-splines.
- RLHandler → manejador para el control de la recta de regresión.

Ilustración 120 : Estructura del paquete handlers

9.10.- JAXB

Clases para el tratamiento de los archivos XML

- Generated → Clases generadas por la librería JAXB de java que representan cada uno de los registros que se escriben en el archivo XML. La clase ObjectFactory es la clase factoría que crea cada uno de los objetos de las diferentes clases generadas.
    - Coordenadas2DXml
    - DataCirculoXml
    - DataFlowersXml
    - ObjectFactory
    - PuntoVectorXml
- Handlers
    - DatosFlorHandler → manejador del archivo XML, el cual permite crear, añadir, modificar o borrar registros.
    - AbstractJAXBHandler → Clase abstracta para los manejadores XML
    - JAXBReaderWriterHandler → manejador genérico para cualquier tipo de archivo XML.
- DataStoreException →Clase que gestiona la excepción si falta información en el XML o no está bien formado con respecto al esquema.
- RecordNotFoundException →Clase que gestiona la excepción de solicitud de un registro que no se encuentra en el XML.
- ReaderWriter → interfaz a implementar para leer o escribir en cualquier tipo de archivos.
- FileReaderWriter → Clase que implementa la interfaz ReaderWriter
- datosFlor.xsd → no es una clase java sino el esquema del archivo XML.

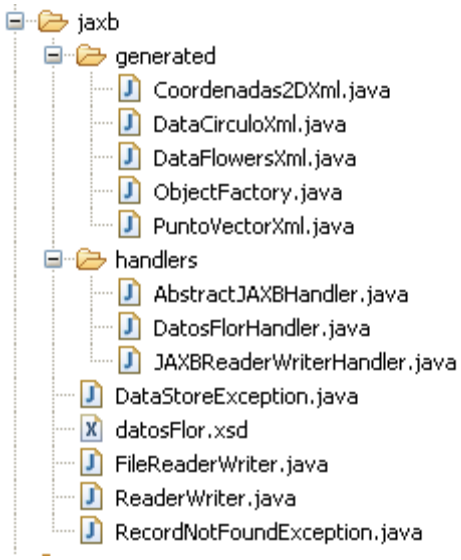


Ilustración 121 : Estructura del paquete jaxb

9.11.- LAPIZ

Clases que permiten dibujar objetos mediante movimiento relativo en un plano.

- Lapiz
- LapizXY
- LapizXZ
- LapizYZ

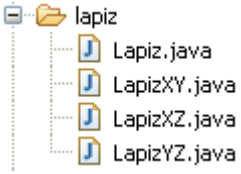


Ilustración 122 : Estructura del paquete lapiz

9.12.- PROPERTIES

- Config → Clase para la gestión del archivo de propiedades que configura la aplicación.

Ilustración 123 : estructura del paquete properties

9.13.- TEXTURES

Clases para el tratamiento de imágenes y texturas.

- Viewer
    - JPanelViewImage → Panel para mostrar imágenes.
- BitmapLoader
- ExceptionHandler
- HelpOverlay

- ResourceRetriever
- TextureReader

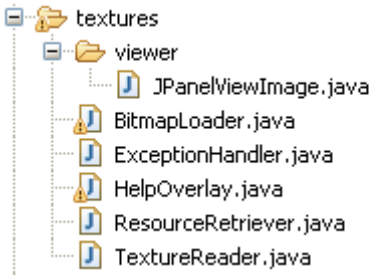


Ilustración 124 : Estructura del paquete textures

9.14.- TRANSFORMACIONES

Clase que gestiona las transformaciones afines que se aplican a los elementos de la escena. Contiene la clase TAfin que almacena la matriz de modelado de cada componente de la escena. El resto son clases para el movimiento de los objetos mediante el botón derecho del ratón (ver sección 5.18.4.1).

- ArcBall
- Matrix4f
- Point2f
- Quat4f
- TAfin
- Vector3f

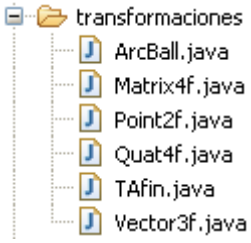


Ilustración 125 : Estructura del paquete transformaciones

9.15.- UTILS

Paquete que contiene clases con algoritmos y funciones útiles para la implementación.

- Files → contiene clases para la compresión y descompresión de los archivos .flw (ver sección 7.2).
  - FileManager
  - FileManagerFlowers
- Images → contiene clases para tratamiento y carga de archivos de imagen.
  - CodigosFormatosImagenes
  - ProcesadorImagenes

- Math
    - MathFlowers → contiene métodos para multiplicar matrices y otros que se usan en la creación de PLY.
- ConversorAngulos → funciones útiles para pasar de grados a radianes y viceversa.
- SistemaCoordenadas → Contiene métodos para cambiar el sistema de coordenadas de un objeto (ver Fundamentos matemáticos y teóricos →cambio sistema de coordenadas).
- UtilidadesGraficas →Contiene métodos para dibujar los ejes de coordenadas.
- UtilesMalleado → contiene funciones necesarias para la creación de la malla de la hoja (ver sección 5.8).
- UtilesGenera3D → Contiene el algoritmo de correspondencia entre 2D y 3D (ver Fundamentos matemáticos y teóricos →correspondencia 2D→3D).
- UtilPoligono →Contiene funciones para crear polígonos regulares.

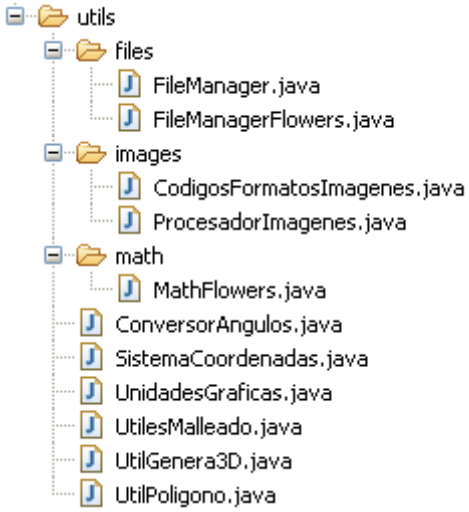


Ilustración 126 : Estructura del paquete utils

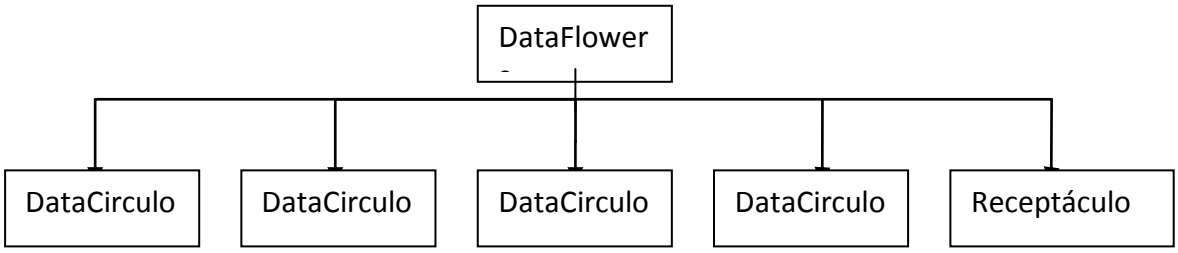
10.- ESTRUCTURAS DE DATOS

A lo largo del desarrollo de la aplicación han sido necesarios el diseño de estructuras de datos nuevas y la implementación de otras ya existentes, para almacenar en tiempo de ejecución los datos generados. Para mantener la estructura de la flor hemos diseñado una estructura de datos específica que se adapta a su representación, y que hemos llamado DataFlowers. Por otra parte, para generar las mallas, hemos reutilizado una estructura estándar, (ver sección 5.4.1).

10.1.- DATAFLOWERS

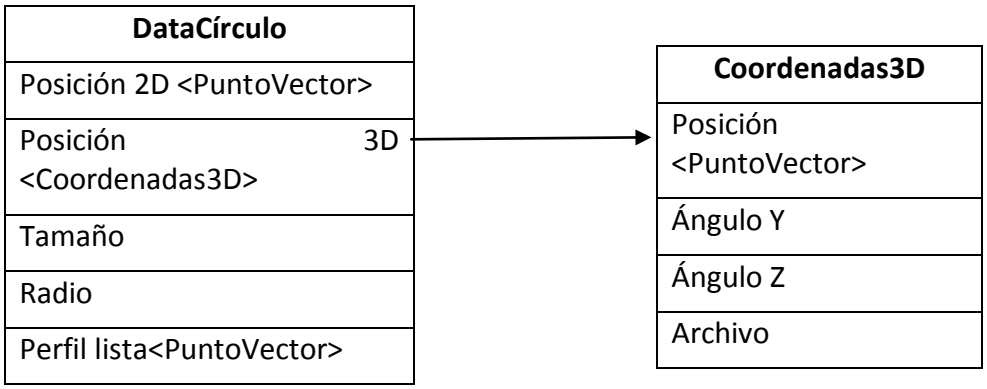
La estructura DataFlowers representa esquemáticamente la estructura de la flor que proponemos en la aplicación. Ésta mantiene actualizado el estado actual de la estructura permitiendo el desplazamiento entre pantallas, la importación/exportación de archivos, y las modificaciones que surjan en la edición de la flor, todo sin ninguna pérdida de información.

DataFlowers almacena los componentes que forman la flor junto con su tamaño, posición y otros elementos necesarios para su representación, así como el receptáculo que los contiene. Está formada por cuatro elementos llamados DataCírculo, que representan los cuatro tipos de componentes, y otro adicional que guarda el archivo con el receptáculo.



Cada uno de los DataCírculo contiene los puntos de la escena donde se colocan los componentes, su tamaño, el radio de la circunferencia correspondiente en 2D y el perfil del receptáculo. Es importante señalar que el tamaño de los elementos de un mismo tipo de componentes es el mismo para todos, motivo por el cual almacenamos un único tamaño por DataCírculo. Los datos guardados con la información de los puntos hacen referencia a su posición en 2D y en 3D. Para 2 dimensiones almacenamos una lista de elementos del tipo PuntoVector (véase el siguiente apartado) con la posición de cada componente en su circunferencia. Para tres dimensiones tenemos una lista de elementos del tipo Coordenadas3D que contiene los datos necesarios para situar cada componente sobre la superficie del receptáculo.

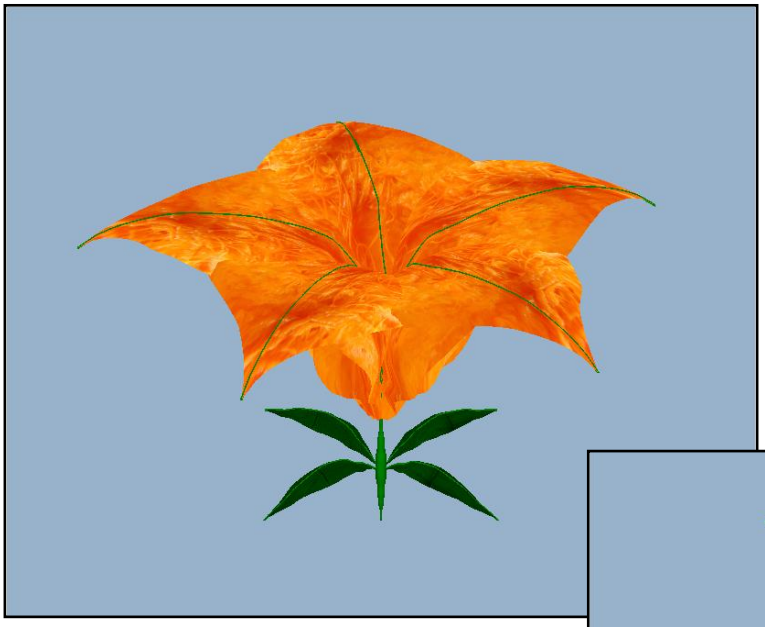
Cada elemento de la clase Coordenadas3D está formado por un elemento PuntoVector con la posición 3D del componente sobre el receptáculo, fijando así el lugar definitivo que ocupa en la flor. Además contiene el ángulo de inclinación de dicho componente, el ángulo de rotación de éste sobre el plano XZ y el nombre del archivo que lo contiene. Las Coordenadas3D se calculan cada vez que se ejecuta el algoritmo de correspondencia entre 2D y 3D.



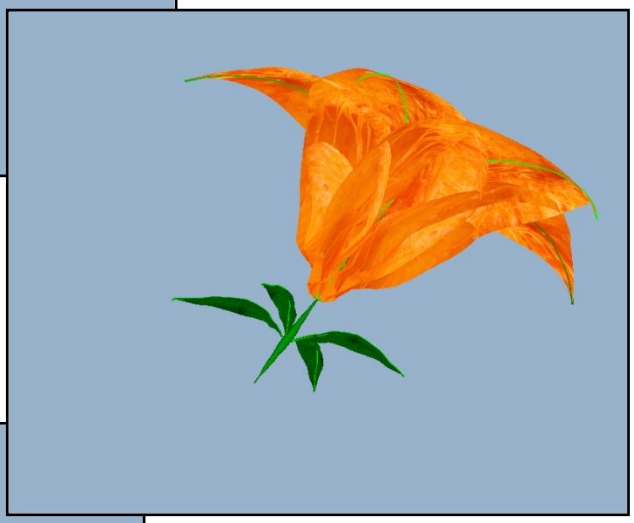
10.2.- PUNTOVECTOR

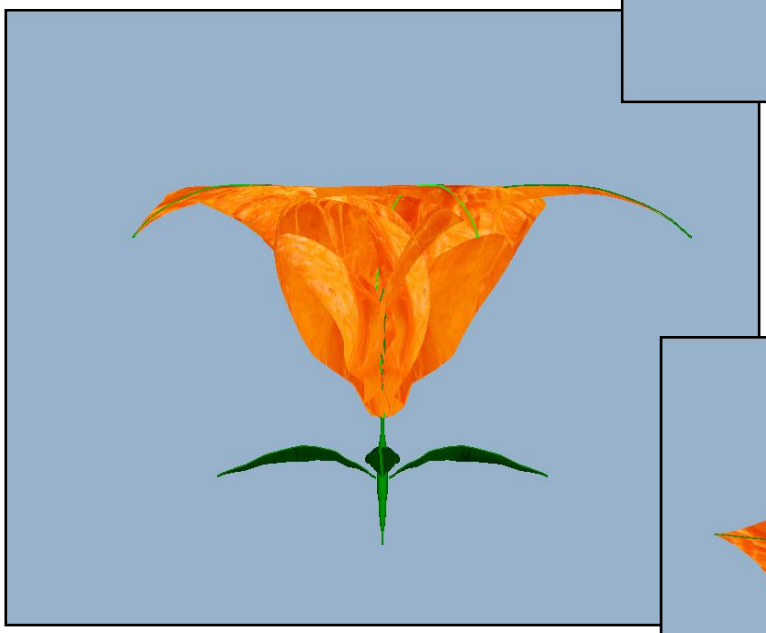
Es una estructura que almacena la posición de un punto en la escena, siendo la clase más básica de la aplicación. Para ello, guarda sus coordenadas en los ejes X, Y, Z (0 si es en 2D), un cuarto elemento nos indica si se trata de un punto o un vector.

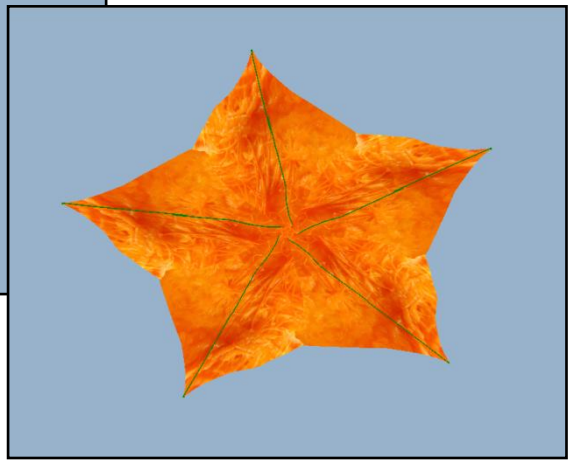
A1.- APENDICE [GALERÍA DE FOTOS]

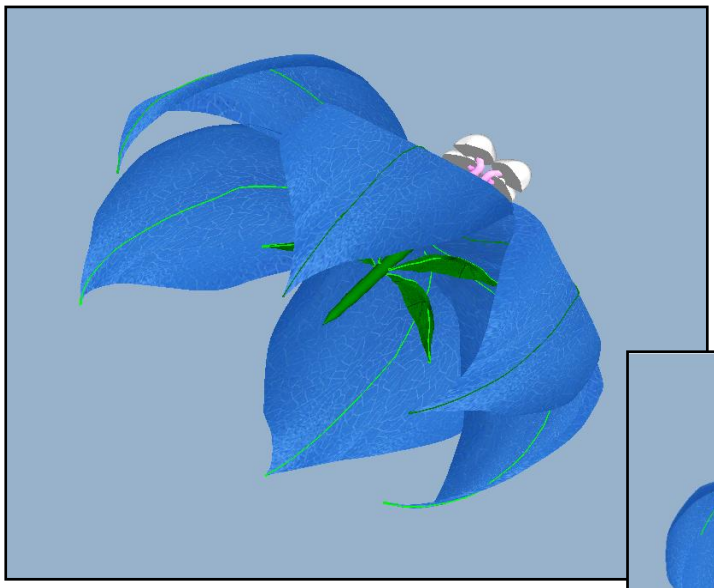


TULIPÁN

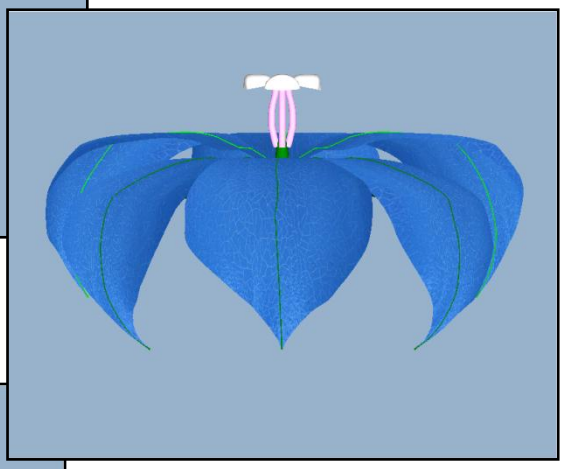


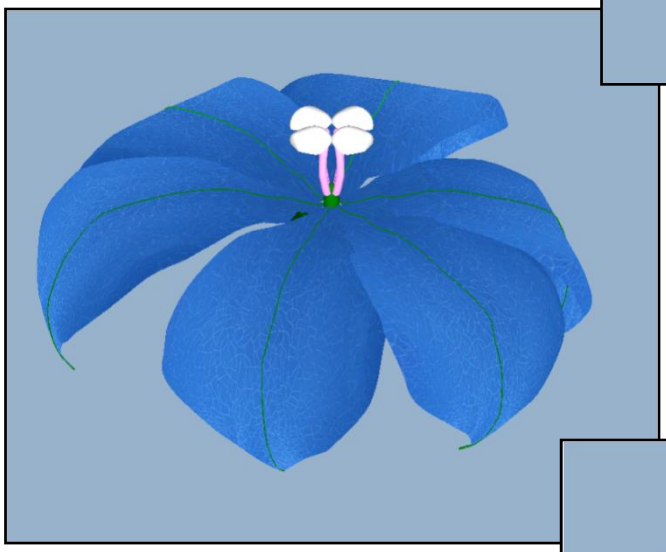


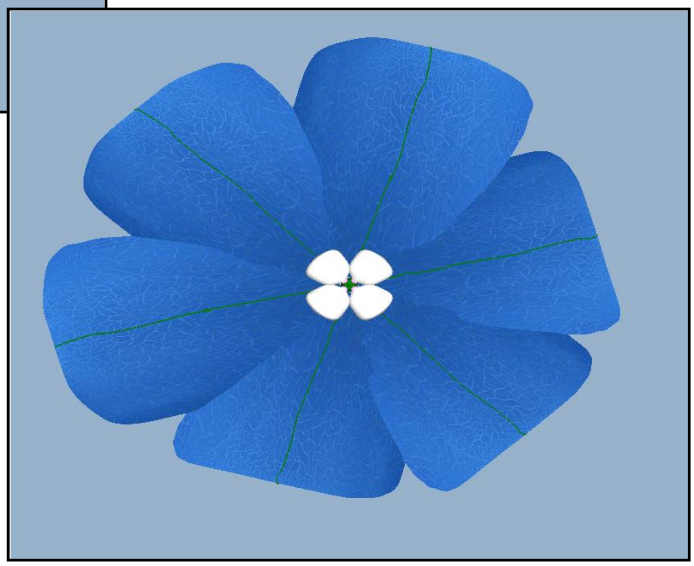


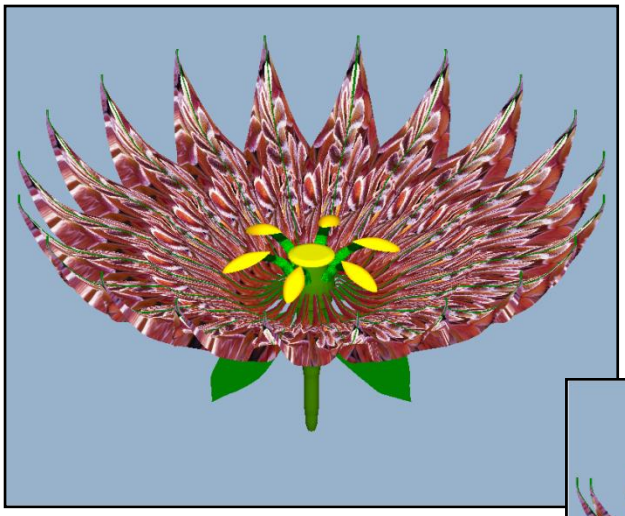


FANTASÍA

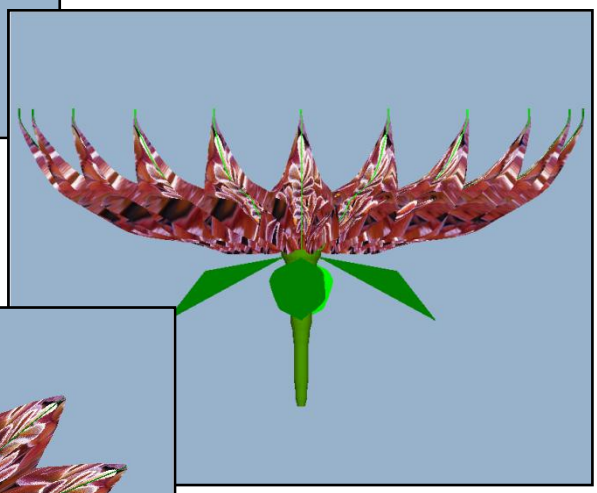


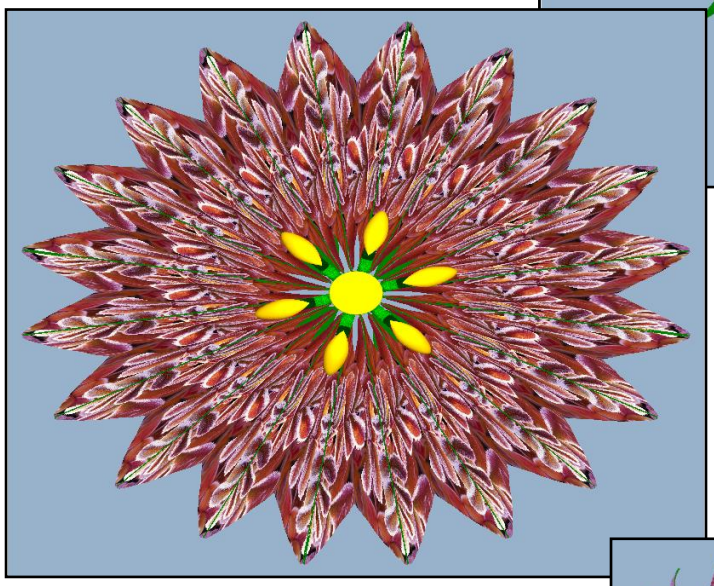


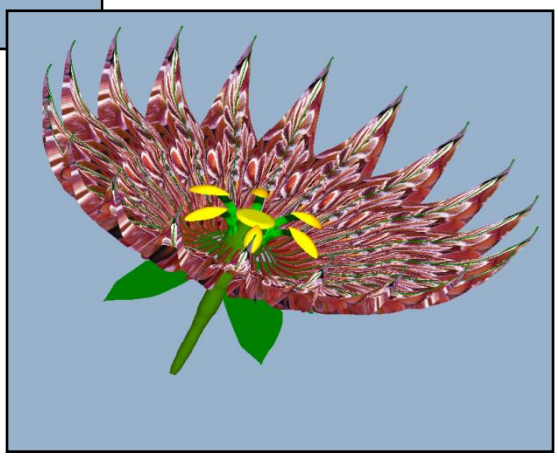


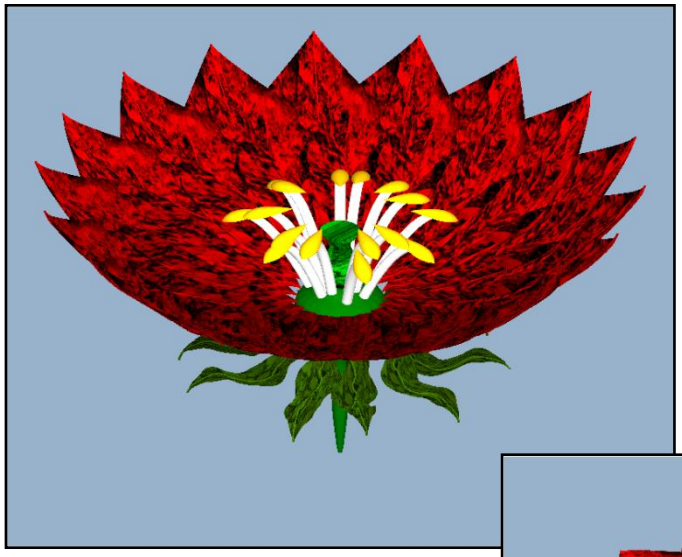


PRIMERA CREACIÓN

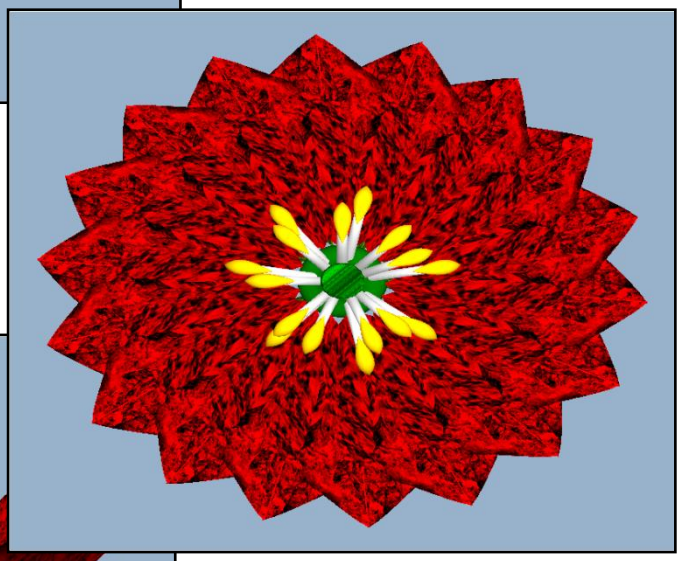


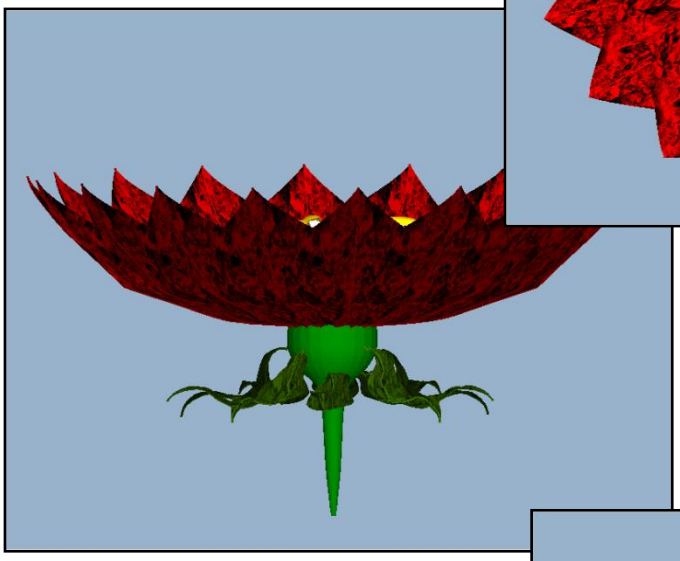


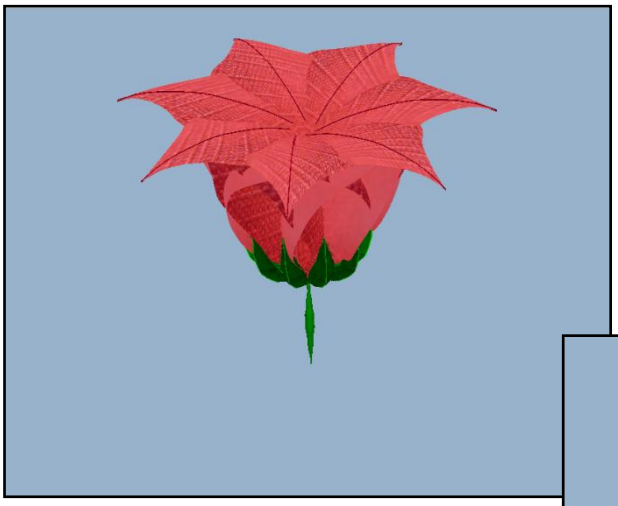




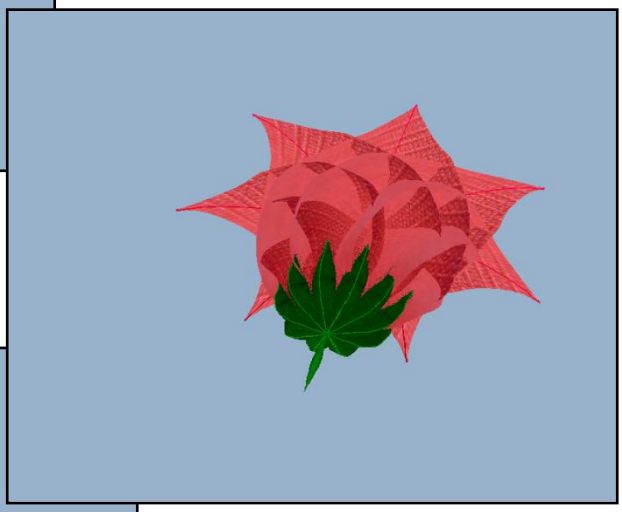
ESTAMBRES NO UNIFORMES

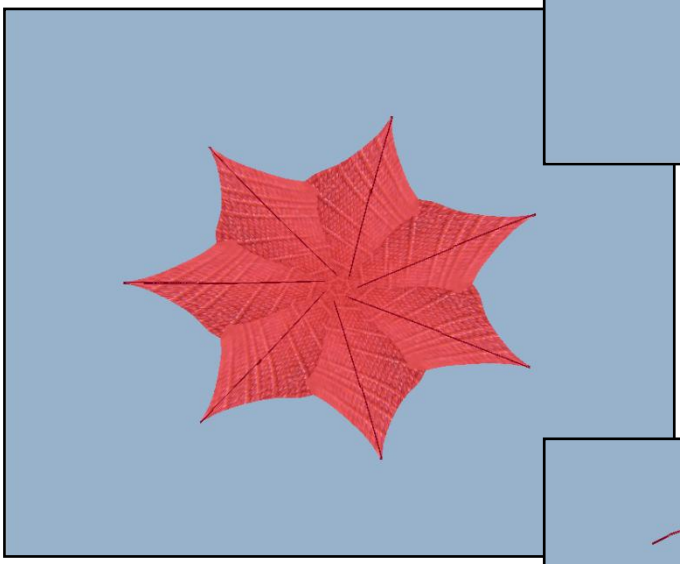


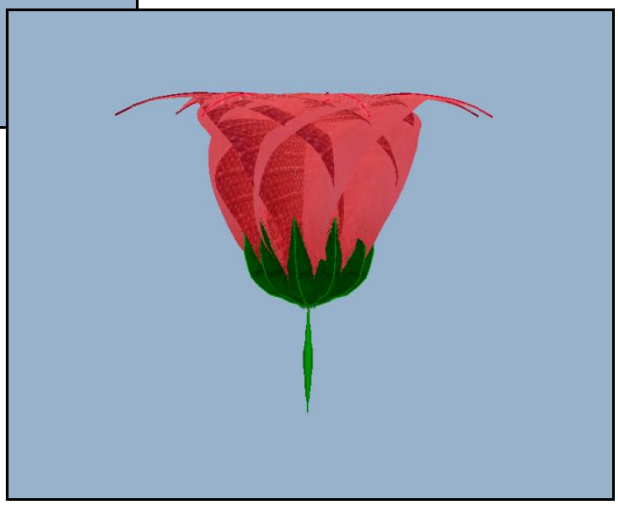




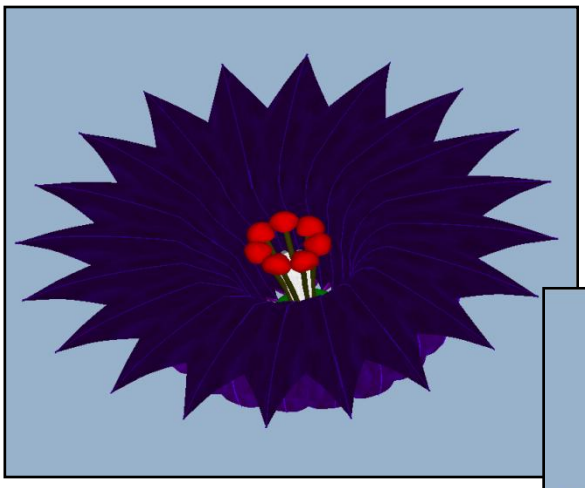
ROSA

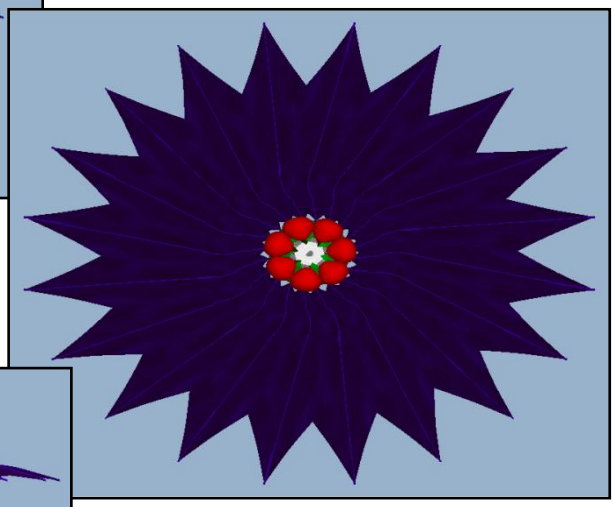


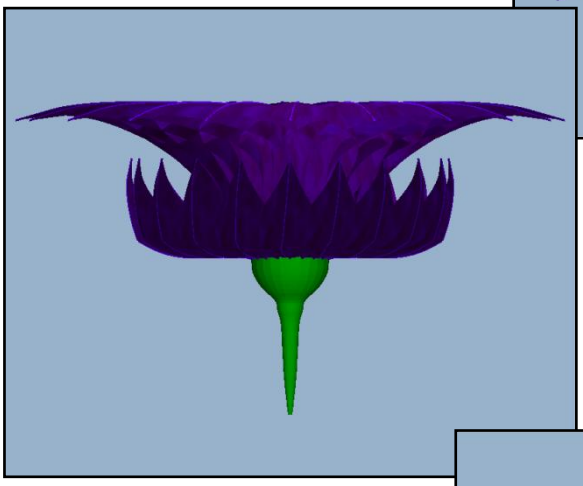


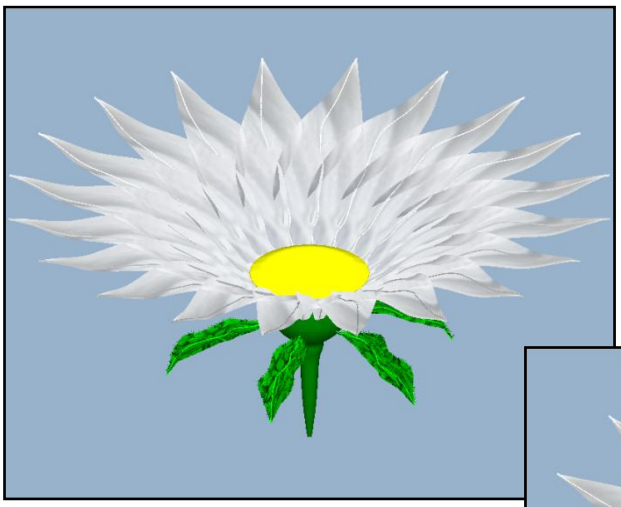


EXÓTICA 2

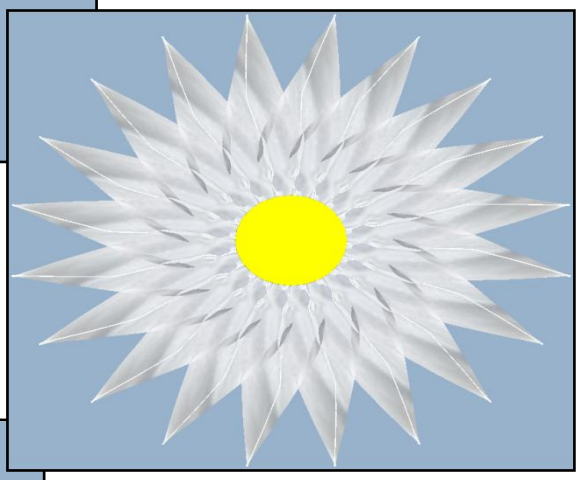


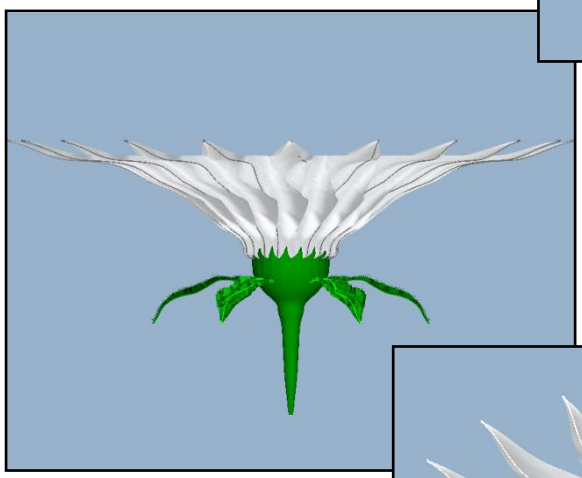


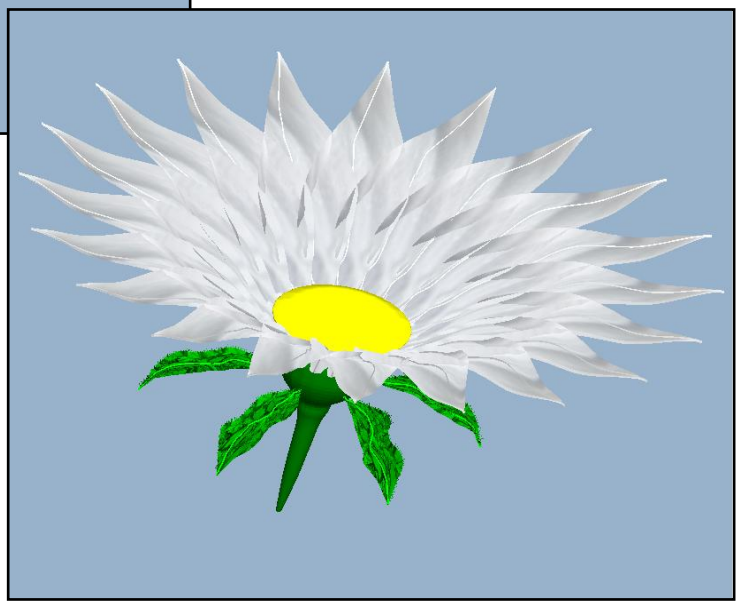


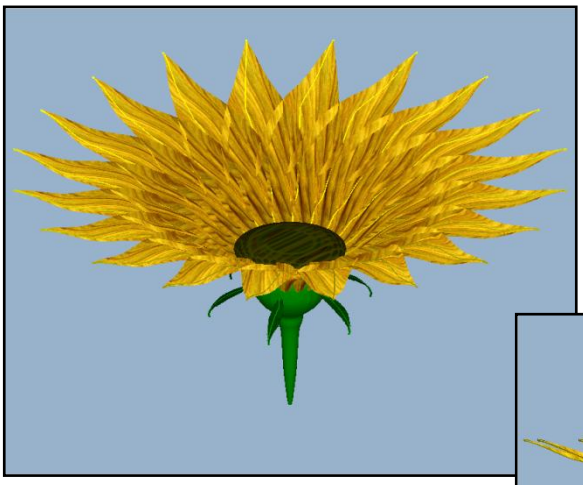


MARGARITA

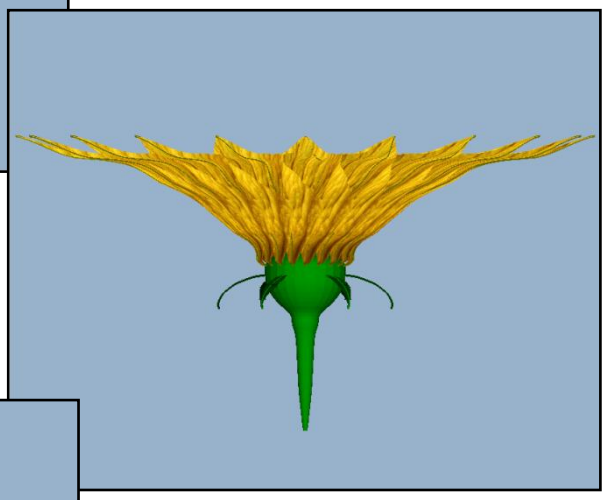


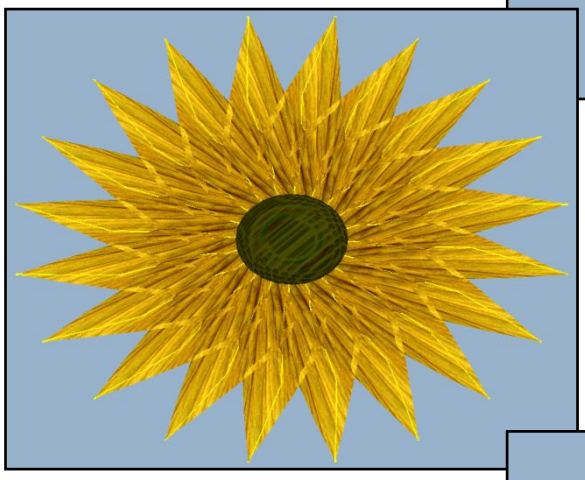






GIRASOL





A2.- APÉNDICE [PLY]

PLY es un formato de archivos conocido como **Polygon File Format** o **Stanford Triangle Format**. Consiste en una simple descripción de objetos 3D, que fue diseñado como un formato para los investigadores que trabajan con modelos poligonales.

El formato de archivos PLY no pretende ser un lenguaje de descripción de la escena en general, ni de sombreado. Esto significa que no incluye matrices de transformación, modelado jerárquico, instanciaciones de objetos, texturas…

Un archivo PLY consta de una cabecera seguida de una lista de vértices y, a continuación, una lista de polígonos denominados caras. El encabezado especifica el número de vértices y polígonos que están en el archivo, y también las propiedades asociadas a cada vértice, como pueden ser sus coordenadas, su normal y el color. Los polígonos o caras se representan como listas de los índices de los vértices que lo forman, y cada cara comienza con un recuento del número de elementos que forman su la lista.

La forma en que pueden almacenarse todos estos datos es mediante archivos en formato ASCII o binario.

La estructura típica de estos ficheros es la siguiente:

- Cabecera.
- Lista de vértices.
- Lista de caras.

Un ejemplo sencillo podría ser el siguiente:

```
ply
format ascii 1.0           { ascii/binary, format version number }
comment made by            { comments keyword specified, like all lines }
comment this file is a cube
element vertex 8           { define "vertex" element, 8 of them in file }
property float x           { vertex contains float "x" coordinate }
property float y           { y coordinate is also a vertex property }
property float z           { z coordinate, too }
element face 6             { there are 6 "face" elements in the file }
property list uchar int vertex_index { "vertex_indices" is a list of ints }
end_header                 { delimits the end of the header }
0 0 0                      { start of vertex list }
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3                  { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

Ilustración 127: Formato archivo PLY

En la zona sombreada de la imagen podemos ver la descripción típica de un fichero PLY. Se trata de la cabecera que describe el formato de la lista de vértices y caras, y otras propiedades de los vértices, como el color, coordenadas, puntos de textura…

A3.- APÉNDICE [MATEMÁTICAS Y ALGORITMIA]

A3.1.- B-SPLINES

Una curva B-spline es matemáticamente más compleja que la interpolación lineal de unos puntos. La principal diferencia consiste en que las B-splines sólo aproximan los puntos de control, y no tiene la obligación de pasar por ellos.

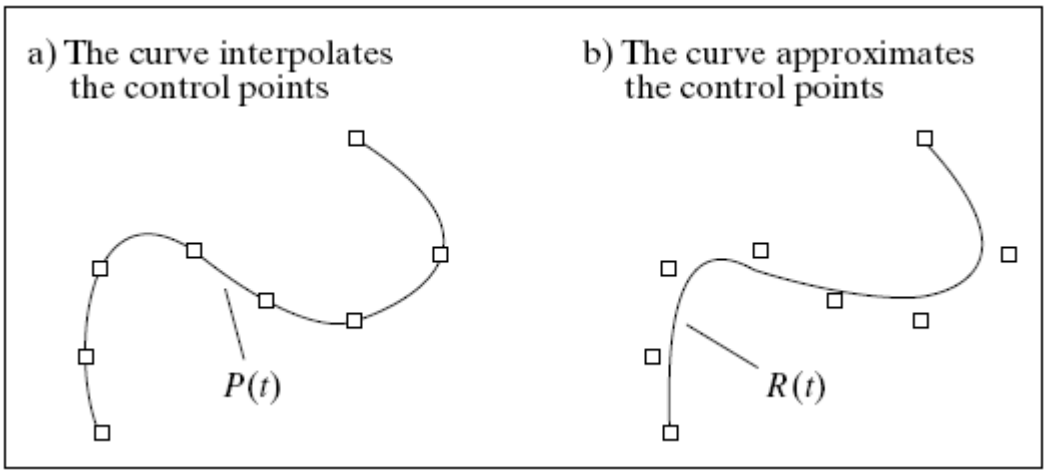


Ilustración 128: Interpolación versus métodos de aproximación de curvas.

Aunque existen muchas aproximaciones para formular B-splines, existe una fórmula que define todas las funciones B-spline de cualquier orden. Es una relación recursiva que es fácil de implementar en un programa y que se comporta bien numéricamente. Cada función B-spline se basa en polinomios de cierto grado m. Si m = 3 los polinomios serán de orden 3 y por tanto de grado 2, por lo que serán B-splines cuadráticas. Si m = 4 los polinomios serán de grado 3 o cúbicos. Estos son los casos más frecuentes aunque la formulación nos permite construir B-splines de cualquier orden.

Propiedades de las B-splines:

1. Son muy suaves.
2. Control local: cada punto de control afecta sólo a una porción local de la curva (debido al soporte local de las bases).
3. El grado de las bases no depende de la cantidad de puntos de control.

Para nuestra aplicación hemos usado B-splines **cuadráticas**. Para calcular la B-spline cuadrática asociada a los puntos de control P_0, P_1,…,P_n(n>1) usaremos la siguiente expresión:

$$P(t) = \sum_{k=0}^{n} P_k N_{0,3}(t\text{-}k), \quad t \in [2, n+1]$$

Donde $N_{0,3}(t)$ es la función:

$$N_{0,3}(t) = \begin{cases} \frac{1}{2}t^2 & for\ 0 \le t \le 1 \\ \frac{3}{4} - \left(t - \frac{3}{2}\right)^2 & for\ 1 \le t \le 2 \\ \frac{1}{2}(3-t)^2 & for\ 2 \le t \le 3 \\ 0 & otherwise \end{cases}$$

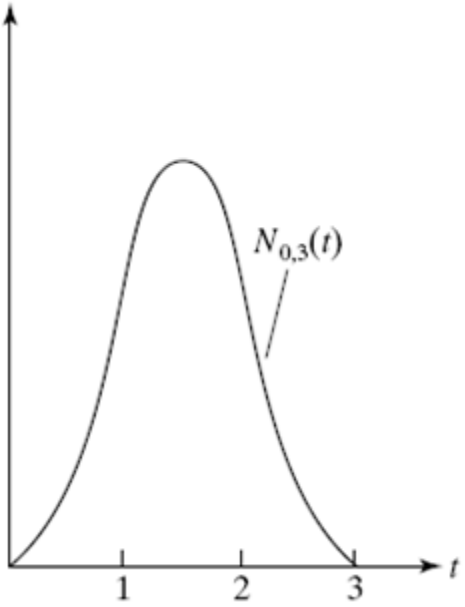


Ilustración 129: Forma de la primera B-spline cuadrática.

A3.2.- TRANSFORMACIONES AFINES.

Suponemos que tenemos un sistema de coordenadas con un origen de coordenadas y tres ejes mutuamente perpendiculares llamados **i**, **j** y **k**. Un punto P en este sistema se representa por $P = P_x\mathbf{i} + P_y\mathbf{j} + P_z\mathbf{k}$. En forma vectorial:

$$P = \begin{pmatrix} Px \\ Py \\ Pz \\ 1 \end{pmatrix}$$

Ahora suponemos que tenemos una transformación afín T que transforma el punto P en el punto Q. T está representada por la matriz:

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Entonces las coordenadas del punto Q en el mismo sistema de coordenadas se obtiene multiplicando P por M:

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

Cabe mencionar que en la última fila de la matriz los ceros representan vectores y el uno puntos (representación homogénea usada en todo el proyecto).

A3.2.1.- TRANSFORMACIONES AFINES ELEMENTALES.

A3.2.1.1.- Traslación

Trasladar un punto de un lugar a otro. La matriz de la representación es:

$$\begin{pmatrix} 1 & 0 & 0 & m_{14} \\ 0 & 1 & 0 & m_{24} \\ 0 & 0 & 0 & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

La siguiente imagen muestra un ejemplo de traslación:

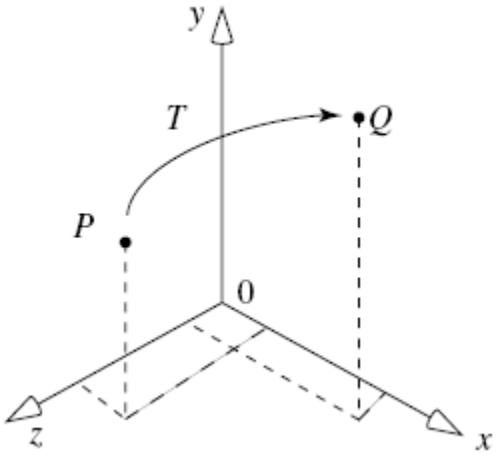


Ilustración 130: Ejemplo de traslación de P a Q.

A3.2.1.2.- Escalación

Esta transformación tiene asociada la matriz:

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde S_x, S_y y S_z son los factores de escalación de las coordenadas correspondientes.

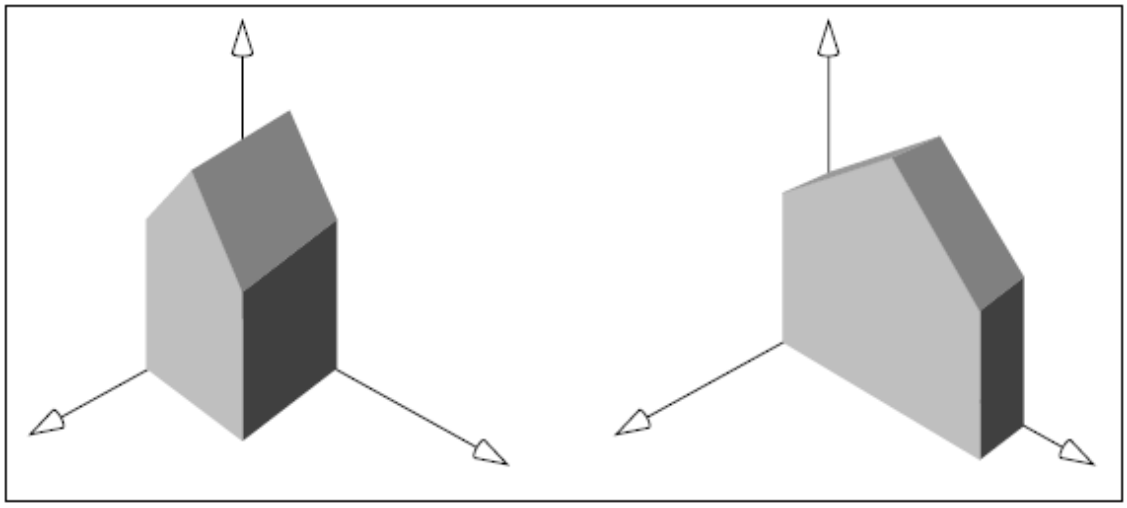


Ilustración 131: (a) figura original, (b) efecto de escalar 0,5 sobre el eje z y 2 sobre el eje x.

A3.2.1.3.- Shearing

Consiste en añadir a P_y un *offset* proporcional a P_x, quedando igual el resto de coordenadas. La matriz asociada es:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ f & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En efecto, tras el producto resulta Q = (P_x, f*P_x + P_y, P_z). La siguiente imagen muestra un ejemplo de shearing. El proyecto no se usa la transformación de shearing.

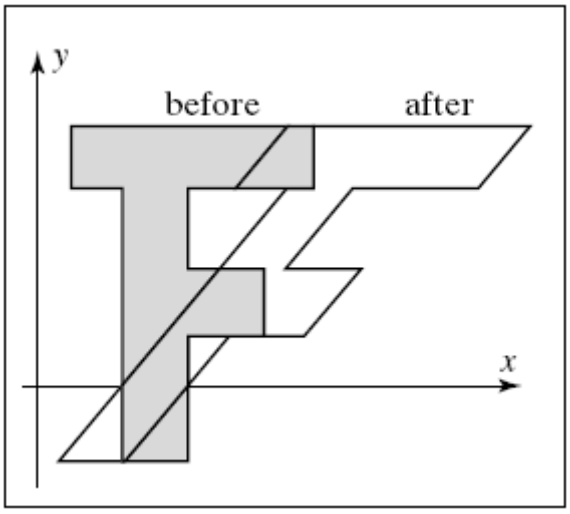


Ilustración 132: Ejemplo de shearing.

A3.2.1.4.- Rotación

Las rotaciones elementales son sobre los ejes de coordenadas. Un valor positivo del ángulo de rotación hará que se rote en sentido contrario a las agujas del reloj, y uno negativo lo hará en sentido horario.

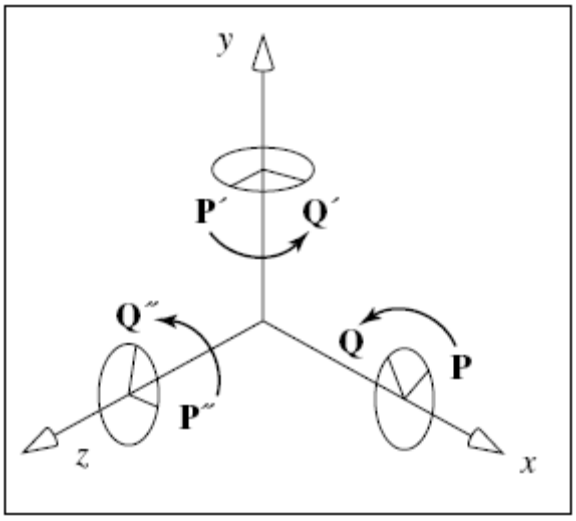


Ilustración 133: Rotaciones positivas sobre los tres ejes.

Las matrices de rotación de los ejes x, y, z respectivamente son:

$$R_x(\beta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_y(\beta) = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\beta) = \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde s = sen (β) y c = cos(β).

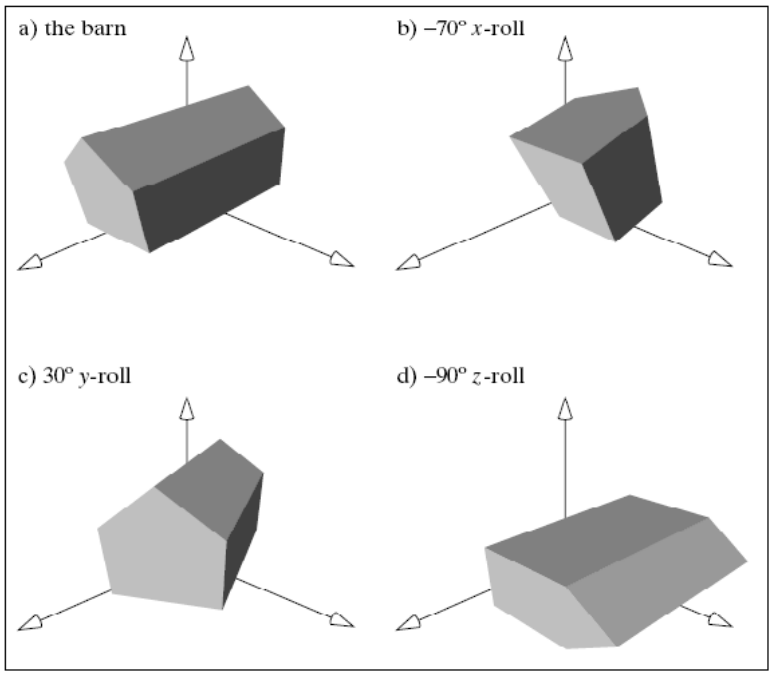


Ilustración 134: Rotaciones elementales.

Para rotar un elemento sobre un eje arbitrario u con coordenadas (u_x, u_y, u_z), la matriz de trasformación es la siguiente, siendo s = sen (β) y c = cos(β).

$$R_u(\beta) = \begin{pmatrix} c + (1-c)u_x^2 & (1-c)u_y u_x - su_z & (1-c)u_z u_x + su_y & 0 \\ (1-c)u_x u_y + su_z & c + (1-c)u_y^2 & (1-c)u_z u_y - su_x & 0 \\ (1-c)u_x u_z - su_y & (1-c)u_y u_z + su_x & c + (1-c)u_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

A3.2.1.5.- Composición de trasformaciones afines

Existen dos formas de entender la composición de trasformaciones afines:

1. Como transformaciones de puntos. En este caso la matriz de la transformación se obtiene pre-multiplicando las matrices implicadas. Por ejemplo para dos matrices la matriz M resultante sería M = M2*M1, siendo M1 y M2 las matrices de la primera y segunda transformación a componer respectivamente.
2. Como transformaciones de sistemas de coordenadas. En este caso la matriz de la transformación se obtiene post-multiplicando las matrices implicadas. Por ejemplo para dos matrices la matriz M resultante sería M = M1*M2, siendo M1 y M2 las

matrices de la primera y segunda transformación a componer respectivamente. Cabe destacar que Open GL usa este enfoque, ya que post-multiplica para acumular las transformaciones afines.

A3.2.1.6.- Propiedades de las transformaciones afines

1. La transformación afín de una combinación afín es la misma combinación afín pero de los puntos transformados. Por ejemplo si a + b = 1, entonces aP + bQ es un punto del espacio, y T(aP + bQ) = aT(P) + bT(Q).
2. Las transformaciones afines preservan líneas y planos.
3. El paralelismo se conserva.
4. Las proporciones se conservan.

A3.3.- ALGORITMO DE REGRESIÓN LINEAL

Las rectas de regresión son las rectas que mejor se ajustan a una nube de puntos (o diagrama de dispersión) generada por una distribución binomial. La forma de obtener estas rectas es por el procedimiento conocido como el método de los mínimos cuadrados. Buscamos una recta de ecuación $y=m_x+n$ que sea la mejor aproximación. Cada punto xi de la primera variable tendrá, por una parte, el valor correspondiente a la segunda variable yi, y por otra, su imagen por la recta de regresión $y=m_{xi}+n$. Entre estos dos valores existirá una diferencia $di=m_{xi}+n-y_i$. Vamos a calcular la recta con la condición de que la suma de los cuadrados de todas estas diferencias $\Sigma(m_{xi}+n-y_i)^2$ sea mínima. Derivando respecto de m y de n y realizando los cálculos matemáticos necesarios, llegamos a la recta de regresión de Y sobre X, que tiene por ecuación en la forma punto-pendiente:

$$y = \bar{y} + \frac{\sigma_{xy}}{\sigma_x^2}(x - \bar{x})$$

Si ahora cambiamos los papeles de las variables X e Y, y realizamos el mismo procedimiento, obtenemos la ecuación de la recta de regresión de X sobre Y:

$$x = \bar{x} + \frac{\sigma_{xy}}{\sigma_y^2}(y - \bar{y})$$

La siguiente imagen muestra la recta de regresión de los puntos representados en la gráfica.

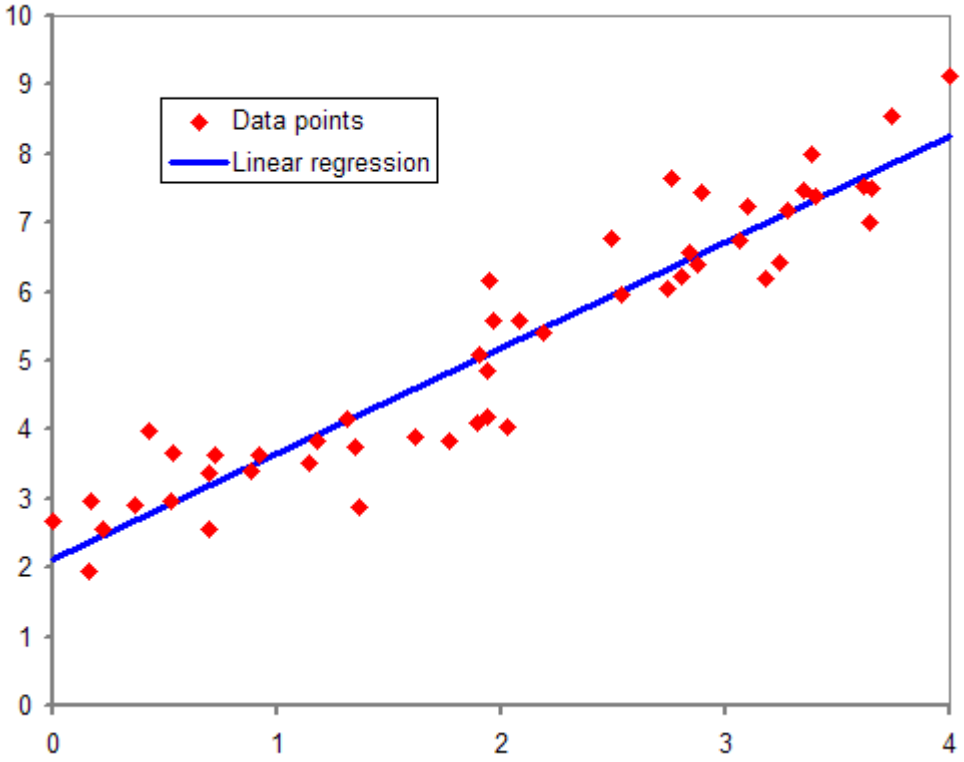


Ilustración 135: Nube de puntos y su recta de regresión.

Una recta se puede definir de la siguiente manera:

y = ax +b

para calcular las constantes a y b hemos usado las siguientes formulas.

$$a = \frac{n\sum_1^n x_i y_i - \left(\sum_1^n x_i\right)\left(\sum_1^n y_i\right)}{n\sum_1^n x_i^2 - \left(\sum_1^n x_i\right)^2} \qquad b = \frac{\sum_1^n y_i - a\sum_1^n x_i}{n}$$

A3.4.- MODELO DE COLOR RGB

Cada píxel en la pantalla tiene asociado un color representado mediante un valor numérico. La forma más común de representar colores es el modelo RGB.

La descripción RGB (del inglés Red, Green, Blue; "rojo, verde, azul") de un color hace referencia a la composición del color en términos de la intensidad de los colores primarios con que se forma: el rojo, el verde y el azul. Es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores primarios. Conviene indicar que el modelo de color RGB no define por sí solo lo que significa exactamente rojo, verde o azul, razón por la cual los mismos valores RGB pueden mostrar colores notablemente diferentes en distintos

dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, los espacios de color de estos dispositivos pueden variar considerablemente.

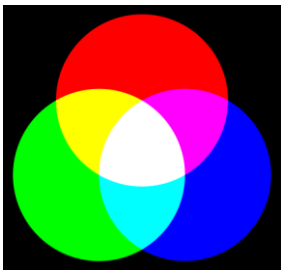


Ilustración 136: Síntesis aditiva.

Para indicar en qué proporción mezclamos cada color, se asigna un valor a cada uno de los colores primarios, de manera que el valor 0 significa que no interviene en la mezcla y, a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla. Aunque el intervalo de valores podría ser cualquiera (valores reales entre 0 y 1, valores enteros entre 0 y 37, etc.), es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255.

Por lo tanto, el rojo se obtiene con (255, 0, 0), el verde con (0, 255, 0) y el azul con (0, 0, 255), obteniendo en cada caso un color resultante monocromático. La ausencia de color —lo que nosotros conocemos como color negro— se obtiene cuando las tres componentes son 0, esto es con la mezcla (0, 0, 0).

La combinación de dos colores a nivel 255 con un tercero en nivel 0 da lugar a tres colores intermedios. De esta forma el amarillo es (255, 255, 0), el cian (0, 255, 255) y el magenta (255, 0, 255).

Obviamente, el color blanco se forma con los tres colores primarios a su máximo nivel (255, 255, 255).

El conjunto de todos los colores se puede representar en forma de cubo de lado 1 .Cada color es un punto de la superficie o del interior de éste. La escala de grises estaría situada en la diagonal que une el color blanco con el negro.

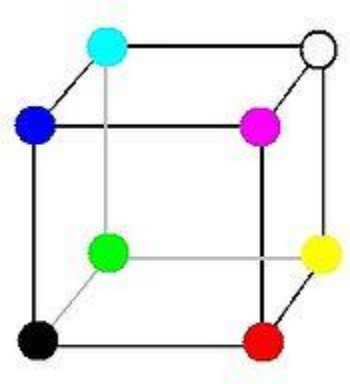


Ilustración 137: Cubo de representación de los colores.

La *profundidad de color* se define como la suma de los bits asociados a las componentes r, g y b. Por ejemplo, si usamos 2 bits para el rojo, 2 para el verde y 2 para el azul, tendremos una profundidad de color de 6 bits por píxel, y una paleta de colores de 2^6 = 64 posibles colores. La profundidad del color ideal es la de 24 bits por píxel, conocido como *color verdadero*. Por encima de esta profundidad el ojo humano es incapaz de notar diferencia.

A3.5.- POLÍGONOS REGULARES

Un polígono regular es un polígono en el que todos los lados tienen la misma longitud y todos los ángulos interiores son de la misma medida.

Una característica de los polígonos regulares, es que pueden dibujarse inscritos a una circunferencia que pasa por cada uno de sus vértices. A medida que crece el número de lados de un polígono regular, su apariencia se asemeja cada vez más a la de una circunferencia.

En un polígono regular podemos distinguir:

- Lado, L: es cada uno de los segmentos que forman el polígono.
- Vértice, V: el punto de unión de dos lados consecutivos.
- Centro, C: El punto central equidistante de todos los vértices.
- Radio, r: segmento que une el centro del polígono con uno de sus vértices.
- Apotema, a: segmento perpendicular a un lado, que termine en el centro del polígono.
- Diagonal, d: segmento que une dos vértices no contiguos.
- Perímetro, P: es la suma de la medida de sus lados.

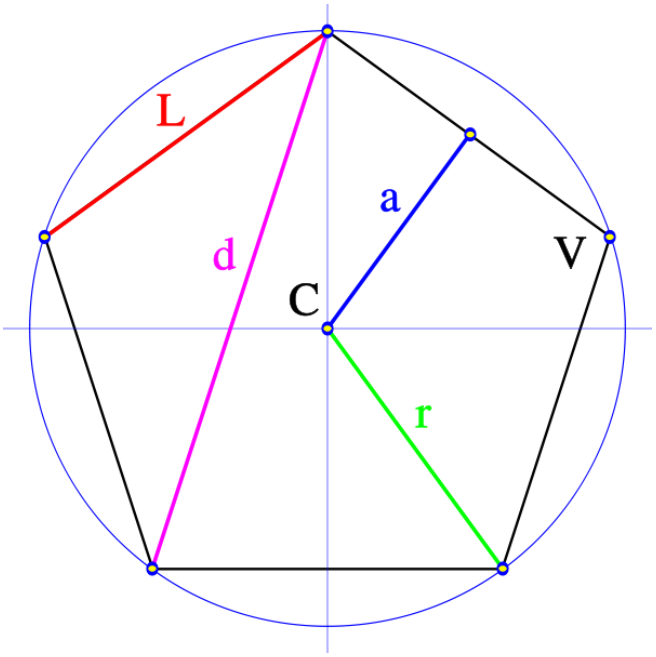


Ilustración 138: Creación de polígonos regulares.

Dadas las características de los polígonos regulares, podemos diferenciar algunas propiedades que se dan siempre, y que son de gran utilidad para determinar sus propiedades, y dimensiones geométricas.

- Los polígonos regulares son equiláteros; todos sus lados tienen la misma longitud.
- Todos los ángulos interiores de un polígono regular tienen la misma medida, es decir, son congruentes.
- El centro de un polígono regular es un punto equidistante de todos los vértices del polígono.
- Los polígonos se pueden dividir en triángulos cuyos lados son el lado del polígono y los dos radios.

A3.6.- OPEN GL

Open GL es una API que permite la comunicación entre el programador y el hardware de la máquina para el diseño de gráficos. Es una máquina de estados en la que existe una colección de variables (tamaños, grosor, colores…) cuyos valores se van modificando.

Open GL es una librería C portable que consta de unos 150 comandos muy básicos. No dispone de comandos de alto nivel para describir escenas en 3D, por lo que debemos construir el gráfico a partir de sus primitivas geométricas que son los puntos, líneas y polígonos.

Open GL está diseñado para trabajar en red, posibilitando la creación de aplicaciones cliente-servidor.

La tubería gráfica de Open GL trabaja internamente con matrices. Cada uno de los vértices del gráfico pasa por una serie de transformaciones de matrices hasta que se visualiza en pantalla.

Las matrices de transformación son las siguientes:

- Matriz de modelado: recoge las distintas transformaciones que ha sufrido el objeto.
- Matriz de vista: recoge el sistema de coordenadas fijado en la cámara.
- Matriz de proyección: proyecta la escena 3D sobre un plano 2D, de acuerdo con la proyección elegida.
- Matriz del puerto de vista: La imagen proyectada pasa a visualizarse en la parte de la ventana especificada por el puerto de vista.

Existen una serie de librerías que amplían la funcionalidad de Open GL, ofreciendo comandos de mayor nivel de abstracción. Estas librerías son GLU y GLUT.

- GLU (Open GL Utility Library) es una librería que se distribuye con Open GL, que está construida sobre Open GL y que suministra comandos de alto nivel para el dibujo de gráficos 3D.
- GLUT (Open GL Utility Toolkit) es una librería para la manipulación de ventanas de E/S que también es portable.

Java Open GL (JOGL) es una biblioteca que permite acceder a Open GL desde programas Java.

BIBLIOGRAFÍA

Francis S. Hill, Jr. Computer Graphics Using Open GL, 2/E

http://es.wikipedia.org/wiki/Flor

http://es.wikipedia.org/wiki/Estambre

http://es.wikipedia.org/wiki/P%C3%A9talo

http://es.wikipedia.org/wiki/S%C3%A9palo

http://es.wikipedia.org/wiki/Pol%C3%ADgono_regular

http://es.wikipedia.org/wiki/Modelo_de_color_RGB

materias.fi.uba.ar/6671/Curvas%202.pdf

http://www.tecgraf.puc-rio.br/~mgattass/fcg/material/shoemake92.pdf

http://www.okino.com/conv/exp_ply.htm

http://www.cc.gatech.edu/projects/large_models/ply.html

http://en.wikipedia.org/wiki/PLY_(file_format)

http://local.wasp.uwa.edu.au/~pbourke/dataformats/ply/http://www.tecnun.es/asignaturas/grafcomp/OpenGL/practica/capitulo4.pdf

https://polimedia.upv.es/visor/?id=6aabaec1-1478-5442-b420-6ffd8d3207e5

http://personal.redestb.es/javfuetub/geometria/coordenadas.htm

http://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal

http://www.sc.ehu.es/sbweb/fisica/cinematica/regresion/regresion.htm

http://pgrafica.webideas4all.com/Graficos3D.html

http://ak.kiet.le.googlepages.com/theredbookinjava.html#CH02

http://jerome.jouvie.free.fr/index.php

https://jogl.games.dev.java.net/

http://www.fdi.ucm.es/profesor/segundo/Documentacion/GL.htm

http://lsi.ugr.es/~curena/doce/vr/pracs/3/

http://descartes.cnice.mec.es/materiales_didacticos/bidimensional_lbarrios/regresion_est.htm

REFERENCIAS:

[1] Glenn McCord, Burkhard Wünsche, Beryl Plimmer, Greg Gilbert and Christian Hirsch (2008). *A pen and paper metaphor for orchid modeling.* In SIGGRAPH '08. ACM Press.

[2] McCord, G. (2007). *Surface manipulation using a paper sculpture metaphor*. BSc Honours Dissertation, Dept. of Computer Science, University of Auckland, Auckland,

New Zealand. URL: http://www.cs.auckland.ac.nz/~burkhard/Reports/2007 S1 GlenMcCord.pdf.

[3] Cherlin, J. J., Samavati, F., Sousa, M. C., , and Jorge, J. A.(2005). Sketch-based modeling with few strokes. In Proceedings of the 21st spring conference on Computer graphics non-photorealistic animation and rendering (SCCG 05), pages 137–145. ACM Press.

[4] Ijiri, T., Owad, S., Okabe, M., and Igarashi, T. (2005). *Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints*. In Proceedings of SIGGRAPH '05, pages 720–726. ACM Press.

[5] Ijiri, T., Owada, S., and Igarashi, T. (2006). *Seamless integration of initial sketching and subsequent detail editing in flower modeling*. In Proceedings of Eurographics 2006, pages 617–624. ACM Press.

[6] E. Galin, P. Poulin. *Real-time Rendering of Realistic-looking Grass.* Eurographics Workshop on Natural Phenomena (2005).

[7] Igarashia, T., Matsuoka, S., and Tanaka, H. (1999). *Teddy: a sketching interface for 3d freeform design.* In Proceedings of SIGGRAPH '99, pages 409–416. ACM Press.

[8] Takashi Ijiri, Shigeru Owada and Takeo Igarashi. *The Sketch L-System: Global Control of Tree Modeling Using Free-form Strokes*. SmartGraphics 2006.

[9] Lisa Streit, Paul Lapides, Mario Costa Sousa, Ehud Sharlin. *Modeling Plant Variations through 3D Interactive Sketches* (2005).

[10] Michael A. Kowalski, Lee Markosian, J.D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, John F. Hughe. *Art-Based Rendering of Fur, Grass, and Trees*. In SIGGRAPH 99.

[11] Ken Perlin. *Improving noise* (2002). ACM Press.