Grado en Ingeniería Informática

Escuela Politécnica Superior de Jaén

Universidad de Jaén
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Prototipo de Simulación 3D de Olivos

Alumno: Juan Valenzuela del Barco

Tutor: Prof. D. Francisco Ramón Feito Higueruela
Cotutor: Prof. D. Francisco de Asís Conde Rodríguez
Dpto: Informática

Junio, 2019

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don **Francisco Ramón Feito Higueruela**, tutor del Proyecto Fin de Grado titulado: **Prototipo de Simulación 3D de Olivos**, que presenta **Juan Valenzuela del Barco**, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2019

El alumno: Los tutores:

Juan Valenzuela del Barco

Francisco Ramón Feito Higueruela

Francisco de Asís Conde Rodríguez

Contenidos

Listado de figuras

Listado de tablas

1 INTRODUCCIÓN Y OBJETIVOS

Este proyecto consiste en la realización de un estudio teórico-experimental sobre técnicas avanzadas para la simulación de olivos, mediante métodos procedurales. Se realizará un breve estudio de las principales características del desarrollo del olivo para a partir de él modelizar su crecimiento y posteriormente replicarlo en un ordenador. Se desarrollará un prototipo de aplicación que permitirá generar olivares virtuales que faciliten estudios de diferentes tipologías de olivar y circunstancias de crecimiento, tarea que sería inviable con otro tipo de herramientas.

Objetivos concretos de este proyecto:

- Elaborar una revisión bibliográfica de los mecanismos más usados para el modelado de árboles.
- Realizar un estudio teórico básico sobre el crecimiento de los olivos.
- Estudiar posibles reglas procedurales que permitan reproducir el estudio realizado.
- Diseñar e implementar un prototipo de aplicación que desarrolle los aspectos anteriores.
- Experimentar con los resultados obtenidos generando diversos modelos y tipologías de olivar.
- Realizar una visualización realista que permita comprobar la calidad de los resultados, así como el rendimiento obtenido.
- Redactar las conclusiones del estudio destacando los aspectos más interesantes sobre el método estudiado, la implementación realizada y los resultados obtenidos.

2 DEFINICIONES Y ABREVIATURAS

Modelado: en informática gráfica el modelado es el proceso por el cual se crea la representación matemática de un objeto.

Modelado procedural: es un tipo de modelado que genera una serie de modelos de forma automática a partir de una serie de reglas llamadas reglas procedurales.

Superficie de revolución: es la superficie generada mediante la rotación de una curva denominada generatriz al rededor de un eje de rotación.

Displacement mapping: en informática gráfica es una técnica que permite modificar la posición de los vértices asociados a una textura a partir de un mapa de altura y un mapa de normales.

OpenGL (Open Graphics Library): es una especificación estándar que define una API multilenguaje y multiplataforma para la creación de aplicaciones gráficas. Para poder usar esta especificación, es necesario escoger una biblioteca específica que la implemente en el lenguaje deseado.

GLEW (*openGL Extension Wrangler*): es la biblioteca para C/C++ usada en el prototipo a desarrollar que proporciona funcionalidades OpenGL Core Profile (la especificación más avanzada de OpenGL).

GLFW (*openGL FrameWork*): es una biblioteca para C/C++ que proporciona una API para crear ventanas con un contexto gráfico OpenGL Core Profile asociado, además de ser capaz de recibir eventos de teclado, ratón, etc.

ImGui: es una biblioteca para C/C++ que permite crear elementos básicos de interfaz dentro de un contexto gráfico de OpenGL.

3 MATERIAL Y MÉTODOS

Uno de los objetivos principales de este proyecto es la generación procedural de modelos 3D de un tipo concreto de árbol, el olivo, para lograr esto, se partirá de un método de modelado de árboles genérico, que será adaptado para cumplir las necesidades del proyecto. En este apartado se exponen las características principales de algunos de los métodos más relevantes, se elige el método a usar y se detallan los aspectos más importantes del mismo.

3.1 Revisión bibliográfica de mecanismos de modelado de árboles

El primer modelo para simular árboles fue propuesto por el matemático Ulam Stanisław, su modelo se basa en el uso de autómatas celulares, concepto que él mismo inventó, y considera que un árbol es una estructura organizada en ramas que surge a partir de una **competición entre autómatas por el espacio**.

La idea básica del modelo de Ulam era ir coloreando celdas de una cuadrícula triangular para generar las estructuras organizadas en ramas, el colorear la siguiente celda o no, se decide con algún parámetro que tengan las últimas celadas coloreadas, ya que cada celda es un autómata celular.

A partir de esta idea básica surgieron diversas extensiones basadas en el mismo concepto de autómatas celulares compitiendo por espacio, estas técnicas tienen en cuenta las interacciones que pueden existir entre los diferentes elementos que forman el árbol además de con su entorno, y aunque estas interacciones influencian claramente el desarrollo de árboles reales, también incrementan bastante la complejidad del modelo, por este motivo, otros métodos más simples (que ignoran factores tan básicos como las colisiones entre ramas) son más usados que los basados en autómatas celulares [1].

El primer modelo no basado en autómatas celulares que surgió fue el propuesto por Hisao Honda, su modelo se basa en la generación recursiva de segmentos a partir de unos parámetros numéricos y unas reglas bien definidas [1]:

- Todas las ramas serán segmentos rectos.
- Una rama padre producirá dos ramas hijas en un proceso de ramificación.
- La longitud de las dos ramas hijas se reduce según dos parámetros constantes, r_1 y r_2 con respecto a la rama padre.
- Las ramas hijas estarán separadas de la rama padre con dos ángulos constantes a_1 y a_2, además las tres ramas estarán en el mimo plano, que será distinto en la siguiente ramificación.
- El plano en el que se encontrarán la rama padre y las ramas hijas se fijará en función de la gravedad, de forma que esté lo más cerca posible de un plano horizontal, a excepción de ramas completamente verticales como el tronco principal.

Combinando diferentes parámetros numéricos, el **método de Honda** era capaz de obtener una gran variedad de estructuras, que con algunas mejoras de otros autores consiguieron de producir efectos de viento, gravedad, aportar más realismo con reglas estocásticas, texturas, superficies, etc.

Uno de los métodos más relevantes en la tarea del modelado de árboles son l**os L-systems**, también llamados Lindenmayer systems en honor a su creador, el biólogo Aristid Lindenmayer, que creó este tipo de sistemas para poder dar una descripción formal del desarrollo de organismos multicelulares, pero que hoy en día es de los métodos más usados para generar modelos de todo tipo de plantas.

Este tipo de sistemas se basan en el uso de las gramáticas formales con reglas de reescritura, que tienen una naturaleza recursiva, igual que las plantas, lo que permite replicar la topología de las mismas de forma muy sencilla.

Los L-systems aportan una gran flexibilidad para modelar cualquier tipo de planta, incluyendo árboles, dado que se requiere del estudio y el diseño de una gramática formal que genere la estructura deseada.

El último método que se revisará, está basado en un algoritmo de **colonización del espacio**, este método, genera una nube de puntos, que son denominados puntos de atracción y que representan el espacio disponible por el que las ramas del árbol se pueden desarrollar, por lo que dependiendo del árbol a generar serán necesarias distintas nubes de puntos.

Una vez establecidos los puntos de atracción, el esqueleto del árbol se forma en un proceso iterativo, empezando en un único nodo en la base del árbol, lejos de la nube de puntos, en cada iteración, se seleccionan nuevos nodos que extenderán el esqueleto en la dirección de los puntos de atracción más cercanos, el proceso termina cuando ocurre alguna de las siguientes condiciones [3]:

- Todos los puntos de atracción son eliminados.
- No quedan nodos bajo la influencia de puntos de atracción.
- Se llega a un número máximo de iteraciones.

Después de este proceso, el esqueleto resultante será tratado para eliminar imperfecciones que pudieran haber surgido en el proceso, como nodos repetidos o muy similares.

3.2 Comparación y selección del método a utilizar

Como se ha comentado anteriormente, los modelos basados en autómatas celulares fueron los primeros que surgieron, y tienen el inconveniente de que producir estructuras complejas es costoso computacionalmente.

El modelo de la colonización del espacio, que es más reciente, usa la misma base que los métodos basados en autómatas, que es la de la competición por el espacio, sin embargo, el método de colonización reduce en gran medida la complejidad, ya que el espacio está limitado por la nube de puntos.

En cuanto al modelo de Honda y sus derivados, sus resultados no serán tan buenos como los de los métodos anteriores, pero, tienen un coste mucho más bajo y son más sencillos de implementar, y es que solamente hay que definir algunos parámetros y reglas que deberán ser aplicadas recursivamente.

Por último, están los L-systems, cuya implementación no es tan sencilla como la del modelo de Honda, pero que son más flexibles, ya que, en los primeros para cambiar aspectos importantes en la estructura bastará con modificar la definición de la gramática, mientras que en el modelo de Honda habría que modificar la implementación al completo.

Por otro lado, los L-systems son capaces de generar cualquier estructura generada por el modelo de Honda, para ello, simplemente habrá que definir las reglas y parámetros de manera formal mediante una gramática.

Por tanto, se pueden descartar los métodos basados el modelo de Honda y los basados en autómatas, quedando dos métodos predominantes:

- Colonización del espacio:
  - Ventajas: aportará unos resultados muy buenos, ya que está basado en la competición por el espacio.
  - Inconvenientes: la generación de la nube de puntos y su tratamiento son tareas bastante complejas de implementar.
- L-system:
  - Ventajas: la creación del sistema y su implementación es más sencilla.
  - Inconvenientes: los resultados no serán tan buenos, ya que no está basado en la competición por el espacio (por lo que se ignoran factores tan básicos como las colisiones entre ramas).

A la hora de elegir entre los dos métodos anteriores, además de tener en cuenta las ventajas e inconvenientes presentados, hay que recordar que estos métodos modelan árboles genéricos y en este proyecto se pretende adaptar el método elegido para que genere un tipo de árbol concreto.

El método elegido y el que más flexibilidad aporta para ser adaptado y producir un tipo de árbol concreto son los L-systems, ya que en el método de colonización, se tendría que generar una nube de puntos que siga la estructura del árbol deseado, mientras que con los L-systems, la tarea de especificación se basa en estudiar la forma concreta del árbol y definirla de manera formal con una gramática.

3.3 Modelado mediante L-systems

Los **L-systems**, son un marco matemático con el que se puede describir el proceso de crecimiento de organismos multicelulares como las plantas, este marco se basa en el uso de una gramática formal, un conjunto de símbolos con unas reglas de formación bien definidas.

Para el modelado de plantas, se asume que estas están compuestas por diferentes módulos, como simples células o complejos órganos (hojas, flores) en un L-system cada símbolo de la gramática representará uno de estos módulos.

El crecimiento de una planta es simulado a partir de un proceso de reescritura, en este proceso, a cada símbolo del sistema se le aplica una regla de reescritura, haciendo que el símbolo inicial se convierta en otro u otros.

A continuación se expone un ejemplo básico de L-system:

- Definición del sistema:
  - Se consideran los caracteres a y b como el alfabeto del sistema.
  - Este sistema tendrá dos reglas de reescritura:
    - $a \rightarrow ab$, que quiere decir, en cada iteración del sistema, el carácter a deberá sustituirse por la cadena de caracteres ab .
    - $b \rightarrow a$, que quiere decir, que en cada iteración del sistema, el carácter b deberá sustituirse por el carácter a .
  - Además, el sistema tendrá una cadena de caracteres básica con la que se inicia, esta cadena es denominada **axioma**, que para este ejemplo será el carácter b .
- Los L-system funcionan a través de iteraciones, una vez definido el sistema se puede empezar a iterar, de forma que se empieza con el axioma y a partir de este aplican todas las reglas de reescritura posibles. Para este ejemplo, que el axioma es b , en la iteración inicial, la única regla que se puede aplicar es $b \rightarrow a$, con lo que el resultado de la primera iteración es el carácter a .

- Para las siguientes iteraciones, se cogerá el resultado de iteración anterior y se aplicarán otra vez todas las reglas de reescritura posibles. Para el caso de la segunda iteración, solamente se tiene una regla, $a \rightarrow ab$, con lo que el resultado de esta será la cadena de caracteres ab .
- En la tercera iteración, se tienen dos caracteres al inicio, con lo que se aplicarán dos reglas de reescritura, $a \rightarrow ab$ y $b \rightarrow a$, con lo que al final de esta iteración se obtiene la cadena aba .
- En la cuarta iteración se aplicarán tres producciones, obteniendo la cadena $abaab$ y así se pueden seguir generando cadenas de caracteres de forma recursiva, como se puede apreciar en la siguiente figura.

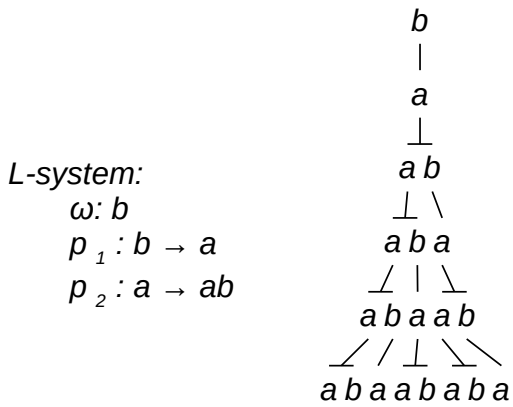


Figura 1: Ejemplo del desarrollo de un DOL-system

Cuando se habla de L-systems, se está haciendo referencia a la técnica en general, sin embargo, los L-system se clasifican en diferentes tipos dependiendo de las características de su gramática.

Con el ejemplo anterior se ha introducido la idea principal del funcionamiento que tiene todo tipo de L-system, a continuación se aporta una definición formal de los tipos más sencillos de L-systems (los OL-system y los DOL-system).

Un **OL-system** es una tripla $G=\{V,\omega,P\}$, donde:

- V es el alfabeto, la lista finita de posibles caracteres con los que el sistema puede operar. De este conjunto derivan los siguientes:
  - $V*$ el conjunto infinito de palabras que se pueden formar con V.
  - $V+$ el conjunto infinito de palabras no vacías que se pueden formar con V.
- ω es el axioma con el que empieza el sistema, ω es una cadena no vacía formada con algunos elementos del alfabeto, $\omega \in V+$.
- P es el conjunto de **reglas de producción** o **producciones**.
  - Una producción es una tupla $(a,\chi) \in P$ notada por $a \rightarrow \chi$, donde la cadena de caracteres a es denominada **predecesor** y la cadena χ es denominada **sucesor**.
  - Este conjunto de producciones serán las reglas de reescritura comentadas anteriormente, de forma que en cada iteración del sistema, el predecesor es sustituido por el sucesor.
  - Si un OL-system, define un carácter $a \in V$ que no aparece en ninguna producción como predecesor (del tipo $a \rightarrow \chi$) se asume la producción identidad: $a \rightarrow a$.

Un **OL-system** será **determinístico** (**DOL-system**), si y solo si, para cada $a \in V$ existe solamente una producción del tipo $a \rightarrow \chi$, donde $\chi \in V*$.

3.3.1 Turtle interpretation

Como se ha expuesto anteriormente, el resultado de un L-system es una cadena de caracteres, es decir, una descripción matemática de una planta, sin embargo, los L-system no determinan de forma explicita ningún mecanismo para su representación gráfica.

Existen diferentes enfoques para crear esta representación gráfica, pero el más extendido es el llamado **turtle interpretation**, esta técnica se basa en interpretar cada carácter de la cadena generada por un L-system como órdenes que se le dan a una *tortuga* que irá generando la geometría.

Para **dos dimensiones**, esta técnica define los siguiente elementos:

- El estado de la tortuga, notado por la tripla (x, y, α), donde (x, y) son las coordenadas cartesianas de la posición de la tortuga en un plano y α es un ángulo con respecto a alguno de los dos ejes que determina la dirección hacia donde mira la tortuga.
- El tamaño de paso d, que indica la cantidad de unidades que debe avanzar la tortuga en la dirección en la que mira.
- El incremento de ángulo δ, que indica el ángulo con el que se modificará la dirección en la que mira la tortuga.

Si se quiere utilizar la técnica de turtle interpretation para generar una representación gráfica de un L-system, este debe definir un alfabeto concreto, cuyos símbolos puedan ser interpretados como órdenes de dibujo. En dos dimensiones, el alfabeto más simple es el siguiente:

- F: indica que la tortuga debe moverse hacia delante un paso de longitud d, **dibujando una línea**, este carácter hace que el estado pase de (x, y, α) a $(x+d\cos(\alpha), y+d\,sen(\alpha), \alpha)$.
- $+$: indica que la tortuga debe girar a la izquierda en un ángulo de δ, este carácter hace que el estado pase de (x, y, α) a $(x, y, \alpha+\delta)$.
- $-$: indica que la tortuga debe girar a la derecha en un ángulo de δ, este carácter hace que el estado pase de (x, y, α) a $(x, y, \alpha-\delta)$.

A continuación se muestra un ejemplo de L-system cuya interpretación gráfica se ha generado mediante turtle interpretation:

L-system:

$\omega : F - F - F - F$

$p_1 : F \rightarrow F - F + F + F F - F - F + F$

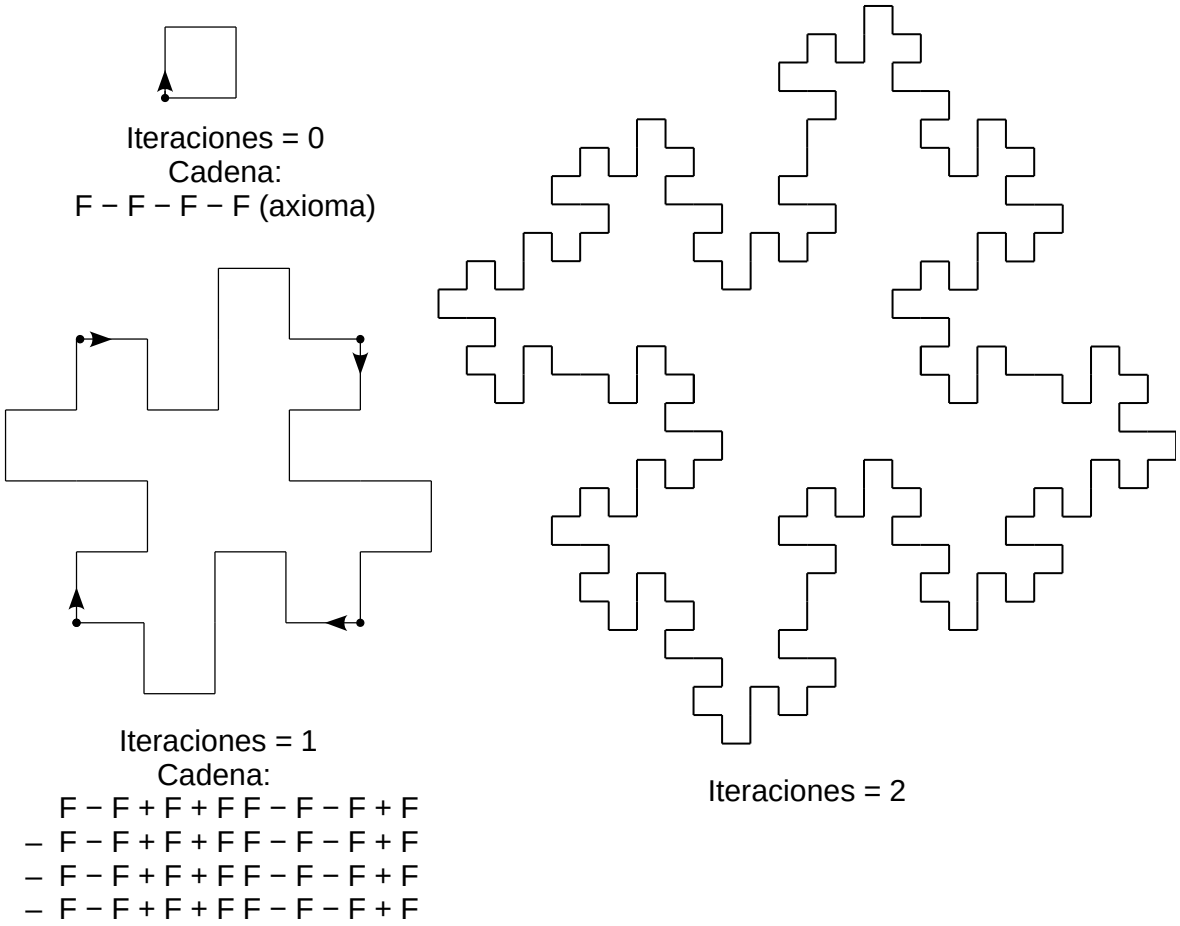


Figura 2: Las dos primeras iteraciones del fractal: quadratic Koch island generado a partir de un L-system con turtle interpretation

A la hora de diseñar la gramática asociada a un L-system, es muy importante conocer como este va a ser interpretado, ya que de lo contrario, la representación gráfica del mismo podría ser errónea. A parte de usar los mismos caracteres que se usan en la interpretación, se pueden añadir caracteres auxiliares, que se ignorarán a la hora de hacer la interpretación.

Hasta ahora, la interpretación se ha limitado a dos dimensiones, pero se puede expandir a tres para generar estructuras tridimensionales, en este caso, se definen los siguientes elementos:

- El estado de la tortuga lo determinan dos triplas:
  - (x, y, z): indica las coordenadas cartesianas de la posición de la tortuga en el espacio.
  - $(\vec{H}, \vec{L}, \vec{U})$: que establecen un sistema de ejes cartesianos que indican hacia donde mira la tortuga en el espacio, el vector $\vec{H}$ (*heading*) indica hacia donde mira la tortuga, el vector $\vec{L}$ (*left*) indica hacia donde está la parte izquierda de la tortuga y el vector $\vec{U}$ (*up*) indica cual es la dirección hacia arriba de la tortuga.
- El tamaño de paso d, que indica la cantidad de unidades que debe avanzar la tortuga en la dirección de $\vec{H}$.
- El incremento de ángulo δ, que indica el ángulo en el cual se debe rotar el sistema $(\vec{H}, \vec{L}, \vec{U})$ en caso de que sea necesario.

Además de lo anterior, el alfabeto debe cambiar para poder modificar los nuevos vectores $\vec{H}$, $\vec{L}$ y $\vec{U}$. En tres dimensiones, el alfabeto más simple es el siguiente:

- F: indica que la tortuga debe moverse hacia delante un paso de longitud d, dibujando una línea, este carácter hace que la posición de la tortuga pase de (x, y, z) a $(x, y, z)+(\vec{H} \cdot d)$.
- $-$: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{U}$ en un ángulo de $-\delta$ (esta rotación es denominada *turn left*).
- +: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{U}$ en un ángulo de $+\delta$ (esta rotación es denominada *turn right*).
- &: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{L}$ en un ángulo de $-\delta$ (esta rotación es denominada *pitch down*).

(continua)

- ^ : indica que el sistema de ejes $(\vec{H},\vec{L},\vec{U})$ debe girar al rededor de $\vec{L}$ en un ángulo de $+\delta$ (esta rotación es denominada *pitch up*).
- \ : indica que el sistema de ejes $(\vec{H},\vec{L},\vec{U})$ debe girar al rededor de $\vec{H}$ en un ángulo de $-\delta$ (esta rotación es denominada *roll left*).
- / : indica que el sistema de ejes $(\vec{H},\vec{L},\vec{U})$ debe girar al rededor de $\vec{H}$ en un ángulo de $+\delta$ (esta rotación es denominada *roll right*).

3.3.2 Bracketed L-systems

Con las características presentadas hasta ahora, la combinación L-system y turtle interpretation es capaz de generar estructuras más o menos complejas, pero al fin y al cabo, la geometría generada sigue una simple línea conectada.

Modelar plantas a partir de una línea es muy complejo, principalmente porque las plantas suelen seguir una estructura basada en **ramificaciones**, para solucionar esto y producir ramificaciones de forma sencilla se define un nuevo tipo de L-system, los **bracketed L-systems**.

Este tipo de L-system añaden dos nuevos caracteres al alfabeto:

- [: indica que el estado actual de la tortuga debe introducirse en una pila de estados, con este símbolo no se dibuja ni se cambia el estado de la tortuga.
-] : indica que se debe sacar de la pila de estados el último estado introducido y establecerlo como el estado actual, este símbolo tampoco se dibuja, pero sí cambia el estado de la tortuga a uno anterior, permitiendo así generar ramificaciones.

Gracias a la pila de estados que implementan los bracketed L-system se pueden generar estructuras ramificadas de forma muy sencilla, resumiendo, lo que esté entre corchetes pertenecerá a una rama y si dentro de una de estas ramas se tienen corchetes, lo que se tendrán serán ramas hijas.

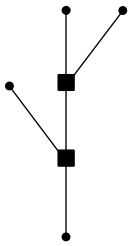
A continuación se muestra un ejemplo de bracketed L-system en dos dimensiones junto con su interpretación geométrica:

L-system:

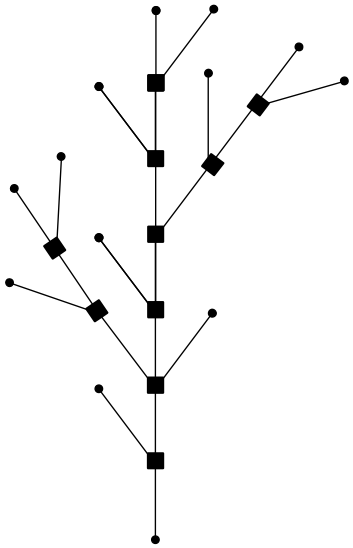
$\omega : F$

$p_1 : F \rightarrow F\,[\,+\,F\,]\,F\,[\,-\,F\,]\,[\,F\,]$

Iteraciones = 0
Cadena: F
(axioma)



Iteraciones = 1
Cadena:
F [+ F] F [– F] [F]



Iteraciones = 2
Cadena:
F [+ F] F [– F] [F]
[+ F [+ F] F [– F] [F]]
F [+ F] F [– F] [F]
[– F [+ F] F [– F] [F]]
[F [+ F] F [– F] [F]]

Figura 3: Ejemplo de bracketed L-system

3.3.3 L-systems estocásticos

Un L-system será **estocástico**, si y solo si, existen dos o más reglas de producción con el mismo predecesor, en cuyo caso se elegirá una frente a otras de manera aleatoria, de ahí su nombre. Es usual que las producciones que repiten predecesor tengan pesos asociados, de forma que una producción tenga más posibilidades de elegirse que otra.

Un L-system estocástico es todo lo contrario a uno determinístico, en donde no se pueden tener varias reglas de producción con el mismo predecesor.

Los L-systems determinísticos solamente permiten modelar plantas con una forma fija bien definida, mientra que con L-systems estocásticos, se pueden modelar especímenes de plantas, analizando estadísticamente las características del espécimen y reproduciéndolas a partir de reglas de producción.

A continuación se muestra un ejemplo de L-system estocástico en el que podemos observar como a partir de solamente tres reglas de producción se pueden obtener estructuras ramificadas diferentes pero cuya naturaleza es similar:

L-system:

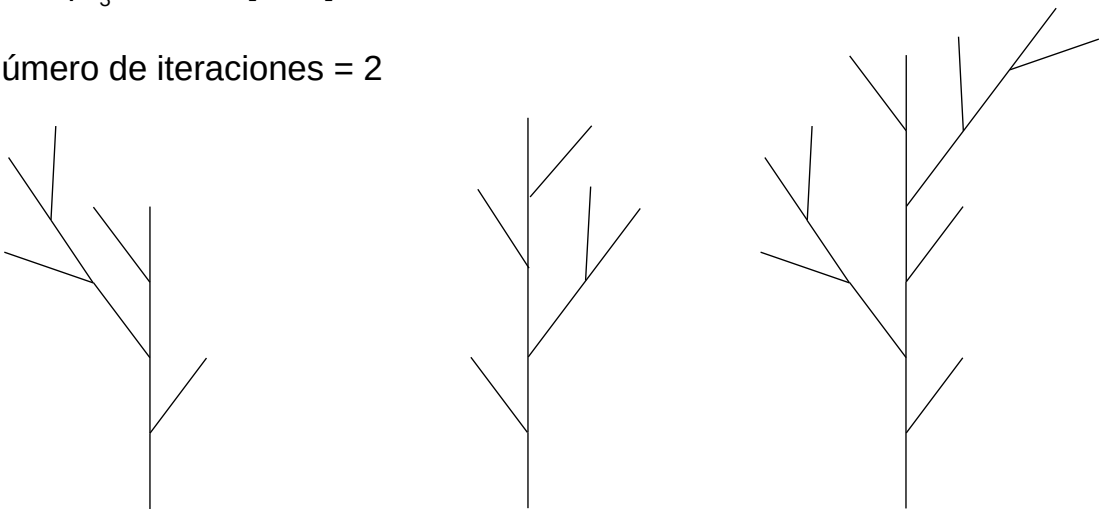
$\omega : F$

$p_1 : F \overset{33\%}{\rightarrow} F\,[\,+F\,]\,F\,[\,-F\,]\,F$

$p_2 : F \overset{33\%}{\rightarrow} F\,[\,+F\,]\,F$

$p_3 : F \overset{34\%}{\rightarrow} F\,[\,-F\,]\,F$

Número de iteraciones = 2



Cadena:
F [– F] F
[+ F [+ F] F [– F] F]
F [+ F] F

Cadena:
F [+ F] F
[– F [+ F] F]
F [+ F] F [– F] F

Cadena:
F [– F] F
[+ F [+ F] F [– F] F]
F [– F] F
[– F [+ F] F [– F] F]
F [+ F] F

Figura 4: Ejemplo de L-system estocástico

3.3.4 L-systems sensibles al contexto

Un L-system será **sensible al contexto** si al menos una de sus reglas de producción es **sensible al contexto**, una producción sensible al contexto solamente se aplicará si en el estado actual del sistema se da una determinada condición. Estas reglas de producción son todo lo contrario a las **libres de contexto** que son las vistas hasta el momento y que se aplican ignorando el estado del sistema.

A continuación se exponen los tipos de L-systems sensibles al contexto más comunes:

- Los **2L-systems**, que usan producciones de la forma $l<a>r \rightarrow \chi$, donde el carácter a (denominado predecesor estricto) se sustituirá por el sucesor χ, si y solo si, en el estado actual del sistema (la cadena de caracteres de la iteración anterior), el carácter a es precedido por el carácter l y seguido del carácter r (es decir, se encuentra la forma $...lar...$). Los caracteres l y r se denominan contexto izquierdo y derecho de la producción.
- Los **1L-systems**, que usan producciones que solamente tienen contexto hacia un lado, con la forma $l<a \rightarrow \chi$ ó $a>r \rightarrow \chi$. Para definir producciones sensibles al contexto se está usando una nomenclatura estándar en la que el contexto izquierdo aparece en la parte izquierda del sucesor estricto y el contexto derecho en la parte derecha acompañados de $<$ ó $>$.
- Los **(k,l)-systems**, cuyo contexto izquierdo está compuesto por una cadena de caracteres de longitud k y cuyo contexto derecho está compuesto por una cadena de caracteres de longitud l.

En general los L-systems sensibles al contexto se conocen como **IL-systems**, en contraposición a los OL-systems (libres de contexto). Los IL-system son una clase más amplia y permiten producciones con contextos de cualquier longitud, hacia izquierda, derecha o ambos e incluso permite combinar producciones sensibles al contexto con las libres de contexto.

Los L-systems sensibles al contexto tienen diferentes usos a la hora de modelar plantas, pueden simular el paso de una señal a lo largo del sistema, el nacimiento de tallos en diferentes momentos del desarrollo, etc. A continuación se muestra un ejemplo de IL-system:

L-system:
ω : baaaaaaaa
p_1 *: b < a → b*
p_2 *: b → a*

Iteración 0: baaaaaaaa
Iteración 1: abaaaaaaa
Iteración 2: aabaaaaaa
Iteración 3: aaabaaaaa
Iteración 4: aaaabaaaa

Figura 5: Ejemplo de IL-system que pasa una señal (carácter b) a lo largo del sistema

3.3.5 L-systems parametrizable

Los L-systems presentados hasta ahora tienen varias limitaciones a la hora de modelar cualquier elemento, una de ellas es que, la longitud de los segmentos dibujados por el L-system deberán ser múltiplos del tamaño de paso d.

Se podría pensar que una aproximación para solucionar este problema es definir d con un valor muy pequeño, así, para representar segmentos de gran longitud habría que combinar varios de estos segmentos pequeños, sin embargo, esta solución es limitada, ya que para la representación de estructuras complejas serían necesarios una gran cantidad de caracteres.

Este mismo problema se tiene con los ángulos, y es que para modificar el ángulo entre segmentos solamente se tiene la variable de incremento de ángulo δ que ocasiona los mimos problemas que d.

A la hora de modelar plantas, estos problemas se acentúan aún más, sobre todo a la hora de definir funciones de crecimiento. Para solucionar estos inconvenientes, surgen los **L-systems parametrizables**.

Estos L-systems operan con **cadenas de caracteres parametrizables**, cuyos caracteres pueden tener un parámetro numérico asociado, que irá después del carácter y entre paréntesis, este enfoque permite operaciones aritméticas entre parámetros, el paso de parámetros entre reglas de producción e incluso el uso de variables predefinidas.

Además, las producciones de los L-systems parametrizables suelen tener una condición asociada que dependiendo de los parámetros dados hace que la producción se pueda aplicar o no:

L-system:
ω: B(2)A(4,4)
p_1 : A(x,y) : y ≤ 3 → A(2x, x+y)
p_2 : A(x,y) : y > 3 → B(x)A(x/y, 0)
p_3 : B(x) : x ≥ 1 → B(x – 1)
p_4 : B(x) : x < 1 → C

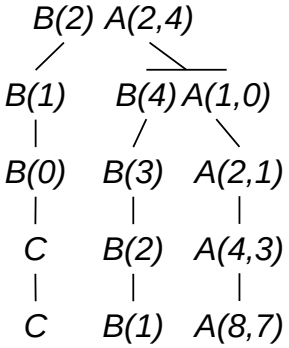


Figura 6: Ejemplo de L-system parametrizable con condiciones asociadas

Dado que ahora los caracteres pueden tener parámetros asociados, la interpretación geométrica de los mismos debe cambiar, el alfabeto parametrizable más simple sería el siguiente:

- $F(a)$: indica que la tortuga debe moverse hacia delante un paso de longitud a, dibujando una línea, este carácter hace que la posición de la tortuga pase de (x, y, z) a $(x, y, z)+(\vec{H} \cdot a)$.
- $+(a)$: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{U}$ en un ángulo de a (gracias a la parametrización, se puede prescindir del carácter $-$ ya que si el parámetro dado es negativo, el giro será negativo).
- $\&(a)$: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{L}$ en un ángulo de a (gracias a la parametrización, se puede prescindir del carácter $\wedge$ ya que si el parámetro dado es negativo, el giro será negativo).
- $/(a)$: indica que el sistema de ejes $(\vec{H}, \vec{L}, \vec{U})$ debe girar al rededor de $\vec{H}$ en un ángulo de a (gracias a la parametrización, se puede prescindir del carácter $\backslash$ ya que si el parámetro dado es negativo, el giro será negativo).

4 MODELADO PROCEDURAL DE OLIVOS

4.1 Estudio básico sobre el crecimiento de los olivos

Antes de iniciar el proceso de modelado de olivos, es necesario hacer un estudio del crecimiento de este árbol, para más tarde poder dar una definición formal del mismo a partir de la gramática de un L-system.

En este estudio, se han tenido en cuenta dos fuentes principales de información:

- Un **estudio de campo** realizado en zonas con olivos cercanas a la Universidad de Jaén, donde se han estudiado los elementos generales de la morfología de los olivos, a través la observación hecha por alguien que no es un experto en la materia.
- La **fuente bibliográfica** [4] que se trata de un libro escrito por expertos, en el que se mencionan, entre otras cosas, algunos aspectos técnicos de la morfología del olivo y con el que se han constatado o corregido los datos del estudio anterior.

Figura 7: Izquierda: foto satélite del olivar cercano a la Universidad de Jaén donde se realizó el estudio de campo. Derecha: portada de la fuente bibliográfica [4]

El elemento más importante a la hora de determinar la forma de un árbol, será la estructura de sus ramas, que se repetirá de manera recursiva.

Las ramas del olivo se estructuran a partir del nacimiento de sus hojas, en una rama de olivo siempre nacen las hojas de dos en dos, y enfrentadas, de forma que entre cada par de hojas existe una rotación de noventa grados, que ayuda a que las hojas no se quiten la luz entre ellas.

A simple vista puede parecer que las hojas no siguen esta estructura, debido a que las mismas tienden a doblarse para recibir la mayor cantidad de luz posible, sin embargo, su nacimiento es el descrito como se puede ver en la siguiente figura:

Figura 8: Rama de olivo donde se ha marcado el nacimiento enfrentado y la rotación de las hojas

Entre la hoja y la propia rama siempre nace una yema, que a lo largo del desarrollo del árbol tiene tres posibilidades:

- La yema puede desarrollarse en una rama común.
- La yema puede desarrollarse en una rama con fruto.
- La yema puede no desarrollarse.

Figura 9: Yemas en una rama de olivo (fuente [4])

El primer caso es que la yema se desarrolle en una rama común, es decir, a partir de la yema nacerá una rama con la misma estructura anteriormente mencionada, creando así una estructura recursiva:

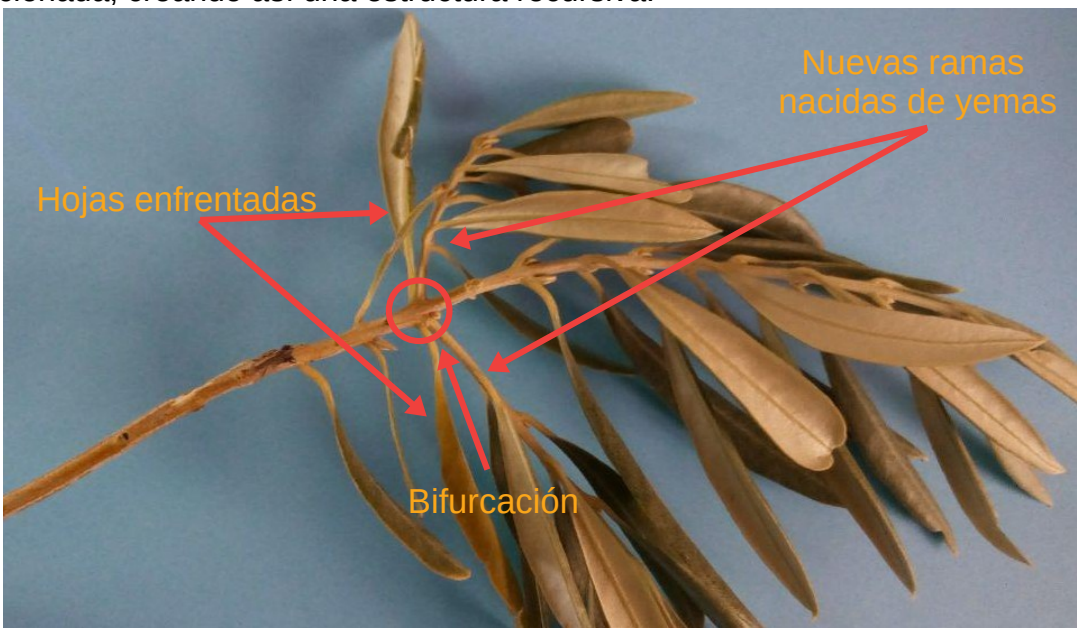


Figura 10: Ejemplo de bifurcación en una rama de olivo

En el segundo caso, que se dará sobre todo si el olivo está en época de floración, las yemas se desarrollan en un tipo de ramas que más tarde generarán el fruto, estas nuevas ramas se denominan **botón floral** y su estructura es diferente a la de las ramas comunes, en este estudio, no se han tenido en cuenta más allá de su definición, ya que no modifican en gran medida la morfología general del olivo.

Figura 11: Ejemplo de rama de olivo en floración (fuente [4])

Por último, el tercer caso, es que la yema no se desarrolle, cosa que sucede a menudo, ya que no todas las yemas generadas se convierten en ramas.

Con los aspectos mencionados anteriormente, se ha descrito el crecimiento natural del olivo, sin embargo, esto no es suficiente para determinar la forma de estos árboles, ya que los agricultores modifican la forma de los olivos *de manera artificial* a través de **podas** para así obtener el mejor rendimiento de la planta.

La poda de un olivo se divide en tres fases:

- La **poda de formación**: es la poda que determina la futura forma del olivo, se debe hacer sobre un material de partida procedente de vivero y durará hasta que el árbol empiece a producir fruto.
- La **poda de producción**: es la poda que se realiza a un olivo que está produciendo fruto, esta poda intentará que el nivel productivo sea el máximo posible y no debe ser muy agresiva.
- La **poda de renovación**: es la poda que se realiza cuando los olivos son más viejos y tienen un rendimiento bajo, estas podas suelen ser bastante drásticas, se cortan ramas viejas y grandes para que el olivo produzca nuevas que mejoren su rendimiento.

Existen diferentes tipologías de poda, las cuales definen de manera diferente que hacer en cada una de las fases mencionadas anteriormente, para este proyecto se ha estudiado la **poda en vaso libre**, que es la más común en la zona de Jaén.

En la poda de formación en vaso libre se pretende conseguir un esqueleto compuesto por un solo tronco y unas ramas principales, para ello, al tallo inicial se le cortarán todas las brotaciones que nazcan por debajo del metro, dejando a los brotes superiores crecer libremente, pasado un tiempo, se eliminarán todas las ramas de la parte superior, excepto un par de ramas principales, dejando al árbol con un esqueleto parecido al siguiente:

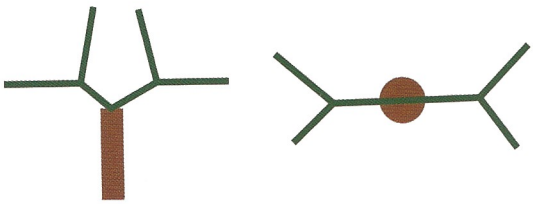


Figura 12: Esquema del esqueleto resultante tras aplicar la poda de formación en vaso libre (fuente [4])

En la poda de producción en vaso libre, se divide al olivo en diferentes zonas y en cada zona se opera de forma diferente [4]:

- Zona interior: se suprimirán las ramas que presenten gran vigor.
- Zona exterior: se cortarán las ramas sobrantes que impidan la entrada de luz al resto del olivo.
- Zona superior: se eliminarán las ramas que tengan gran vigor y que crezcan por encima de las ramas principales.
- Zona inferior: se suprimirán o acortarán ramas muy bajas, que puedan dificultar la realización de determinadas prácticas de cultivo.

En la poda de renovación, se pretende reducir la cantidad de madera del árbol, que es lo que hace que su rendimiento disminuya en mayor medida, para ello, en la poda en baso libre, se cortan de forma total o parcial las ramas principales definidas en la formación, dejando que ramas nuevas se conviertan en principales. Antes de realizar cualquier corte, se deben identificar brotes jóvenes por la zona a cortar o intentar provocarlos mediante una incisión en el tronco.

4.2 Implementación propia de L-system + turtle interpretation

Para poder usar la técnica de L-system con turtle interpretation en el prototipo a desarrollar, es necesario hacer una implementación propia de la misma, además, como se trata de una implementación propia, será más fácil de adaptar a las necesidades específicas que requiera el proyecto.

En el apartado 5 se detallará en profundidad la elección de la plataforma en la que se desarrolla el prototipo, pero, en este punto es importante mencionar que se sigue un enfoque **orientado a objetos**.

Con este enfoque, lo más lógico es encapsular la funcionalidad de los L-systems dentro de una clase, esta clase, en su versión más simple, tendrá dos atributos, el axioma del sistema y un mapa tipo clave-valor con las reglas de producción, donde la clave se corresponde con el predecesor y el valor con el sucesor.

Para implementar el funcionamiento básico de los L-systems, se ha creado un método **recursivo**, que es más fácil de implementar que un enfoque iterativo y es más eficiente. Un resumen de la implementación se puede ver a continuación:

```
Entrada:
    iterationsLeft: el número de iteraciones a realizar
    oldString: la cadena de caracteres actual del L-system
Salida: la cadena de caracteres con el estado del sistema
tras aplicar un número de iteraciones dadas

function newIteration(iterationsLeft, oldString) {
    if(iterationsLeft == 0) {
        return oldString
    } else {
        string newString = ""
        foreach(char c in oldString) {
            string successor = productions.find(c)
            if(successor == "") {
                newString += c
            } else {
                newString += successor
            }
        }
        return newIteration(iterationsLeft - 1, newString)
    }
}
```

Figura 13: Pseudocódigo del método recursivo que genera una nueva iteración en un L-system básico

En la figura anterior, se puede ver que cada carácter que forma parte de la cadena del L-system es susceptible de poder ser reemplazado por su sucesor, esto hace que en esta implementación los predecesores de las producciones solo puedan tener **un carácter**, cuestión a tener en cuenta a la hora de diseñar una gramática.

La clase `LSystem` será la encargada de generar la cadena de caracteres que defina el desarrollo del árbol a modelar, pero será la clase `TurtleInterpreter` la que a partir de esa cadena, genere la geometría.

De esta forma, la clase `TurtleInterpreter` genera y almacenará la información a dibujar, pero no será esta la que la dibuje, ya que esto lo hará la clase `DrawableTree`, cuya funcionalidad principal es dibujar la información que genera una instancia de tipo `TurtleInterpreter`.

Esta separación es necesaria, ya que la tarea de dibujado es una tarea compleja que debería realizar un módulo de la aplicación diferente al módulo de la generación procedural de la geometría, además, de esta forma, si se desea cambiar el sistema de visualización de la aplicación, solamente se tendrá que modificar ese módulo.

Con la separación de las clases `TurtleInterpreter` y `DrawableTree`, se consigue un **bajo acoplamiento** (entre el módulo de dibujado y el de generación procedural) y una **alta cohesión** (entre diferentes clases ya que cada una hace una función concreta bien definida), estos dos elementos son uno de los principios GRASP de buenas prácticas en el diseño de sofware.

Por otra parte, `TurtleInterpreter` sí realiza la tarea de exportar modelos, ya que esta es mucho menos compleja, solamente requiere de la escritura en un archivo de la información geométrica.

Las tres clases `LSystem`, `TurtleInterpreter` y `DrawableTree` se relacionan entre ellas a través de **agregaciones**, de forma que una instancia de `DrawableTree` tiene una referencia a una instancia de `TurtleInterpreter`, que a su vez tendrá una referencia a una instancia de `LSystem`, como se muestra en el siguiente diagrama de clases:

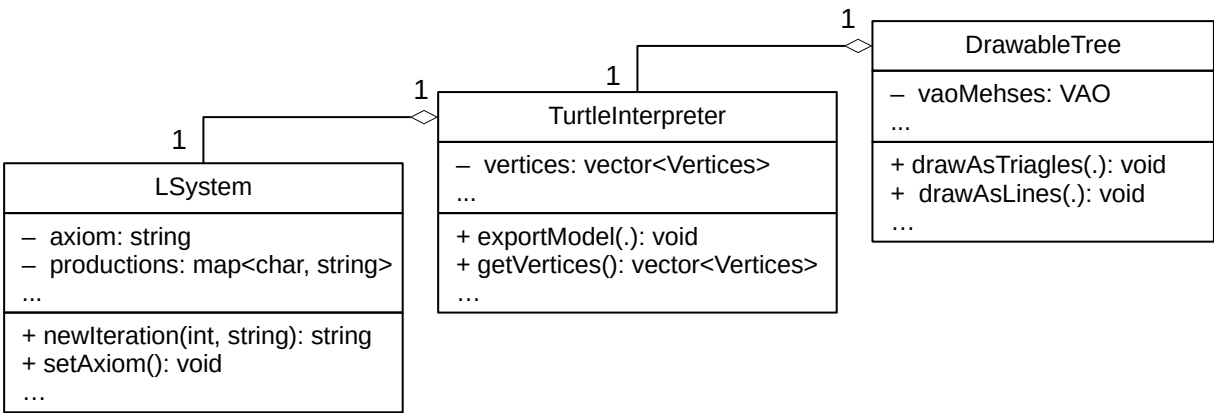


Figura 14: Diagrama de clases UML simplificado con las relaciones entre las clases `LSystem`, `TurlteInterpreter` y `DrawableTree`

De esta forma, en la aplicación se pueden definir L-systems e interpretadores con unos parámetros concretos y usarlos en diferentes combinaciones a un coste muy bajo ya que entre ellos hay referencias, de forma que, a la hora de dibujar, se usará una instancia de `DrawableTree` que deberá leer la información geométrica de la referencia que tiene a la clase `TurtleInterpreter`.

A la hora de hacer la implementación de un L-system, es importante tener en cuenta los tipos que existen, ya que algunos aportan funcionalidades muy relevantes. Los tipos estudiados solamente se diferencian en pequeños aspectos por lo que es posible implementar varios de estos tipos en una sola clase, que es el enfoque que se ha seguido.

Los L-systems estocásticos son un ejemplo, para implementarlos, habrá que cambiar el mapa de producciones por un multi-mapa que almacene para un mismo predecesor, diferentes sucesores a los que se le asocia un peso concreto, de forma que a la hora de elegir un predecesor, se seleccione uno aleatorio teniendo en cuenta los pesos, esto se ha implementado de la siguiente manera:

```
Entrada:
    predecessor: el carácter a sustituir en un L-system
Salida: la cadena de caracteres con un sucesor elegido de
forma aleatoria teniendo en cuenta los pesos de cada
las producciones que se pueden aplicar

function chooseSuccessor(predecessor) {
    int random = randomInt(0,100)
    int sumProb = 0
    foreach(Successor s in productions.find(predecessor)) {
        sumProb += s.probability
        if(random <= sumProb) {
            return s.successorStr
        }
    }
    return ""
}
```

Figura 15: Pseudocódigo con una función que escoge el sucesor de una producción en un L-system estocástico

Otro tipo muy importante de L-system que se debe implementar, son los bracketed L-systems, estos sistemas no modificarán ningún elemento en la parte del L-system, simplemente, a la hora de diseñar la gramática, habrá que tener en cuenta que los caracteres ［ y ］ se usarán para especificar ramificaciones.

Donde sí es importante tener en cuenta los bracketed L-system es a la hora de hacer la interpretación de la cadena de caracteres (clase `TurtleInterpreter`), en ese momento, habrá que definir una pila de estados, de forma que al encontrar el carácter [se introduzca el estado actual en la pila y al encontrar el carácter] se saque el último estado de la pila y se establezca como el estado actual.

El último tipo de L-system implementado, han sido los L-system parametrizables, ya que los L-system sensibles al contexto no aportaban funcionalidades importantes al proyecto. Para implementar L-systems parametrizables, será necesario definir operadores, tanto aritméticos, para operar parámetros numéricos, como lógicos, para implementar las condiciones asociadas que suelen tener este tipo de sistemas.

Para facilitar la implementación de estos operadores, los mismos se han definido como si fuesen funciones de un lenguaje de programación, se tiene primero el símbolo del operador, que en este caso debe ocupar solamente un carácter, y después, entre paréntesis, se tienen los parámetros separados por comas:

Operadores aritméticos		Operadores lógicos	
Operación	Acción	Operación	Acción
$+(a,b,c,...)$	$a+b+c+...$	$<(a,b)$	$a<b$
$-(a,b,c,...)$	$a-b-c...$	$>(a,b)$	$a>b$
$*(a,b,c,...)$	$a*b*c*...$	$=(a,b)$	$a==b$
$/(a,b)$	a/b	$\&(c1,c2)$	$c1 \wedge c2$
$\sim(a,b)$	Genera un número aleatorio en: $[a,b]$	$\|(c1,c2)$	$c1 \vee c2$

Tabla 1: Operadores aritmético-lógicos implementados

En los L-system parametrizables hay que tener en cuenta el paso de parámetros, un predecesor con parámetros asociados deberá poder pasar sus parámetros a su sucesor, para implementar esto, en los sucesores se han reservado seis caracteres: a,b,c,d,e,f, los cuales se usarán para referirse a parámetros dados por su predecesor, de forma que el carácter a se refiere al primer parámetro, el b al segundo, el c al tercero y así sucesivamente.

Al iterar sobre el L-system, se irán sustituyendo los parámetros que se pasan entre predecesor y sucesor, se realizarán las operaciones aritméticas para que al acabar de iterar no quede ninguna expresión y se resolverán las condiciones asociadas a las reglas de producción, haciendo que unas se elijan unas frente a otras. Este proceso se puede resumir con el siguiente pseudocódigo, que amplia el método presentado anteriormente en la figura 13:

```
Entrada:
    iterationsLeft: el número de iteraciones a realizar
    oldString: la cadena de caracteres actual del L-system
Salida: la cadena de caracteres con el estado del sistema
tras aplicar un número de iteraciones dadas
Funciones auxiliares:
    chooseSuccessor(.): que elige el sucesor de un carácter
teniendo en cuenta las producciones estocásticas y las
condiciones con parámetros
    rewriteParameters(.): que dadas dos cadenas de
caracteres, una de un predecesor y otra de un sucesor, ambos
con parámetros, se devuelve la cadena del sucesor al que se
le han pasado los parámetros del predecesor
    evaluateStr(.): que obtiene una cadena de caracteres con
parámetros a operar y devuelve la misma cadena con los
parámetros ya operados

function newIteration(iterationsLeft, oldString) {
    if(iterationsLeft == 0) {
        return oldString
    } else {
        string newString = ""
        for(int i=0; i<oldString.length(); ++i) {
            string successor = chooseSuccessor(oldString[i])
            if(successor == "") {
                newString += oldString[i]
            } else {
                if(oldString[i + 1] == "(") {
                    // Se tienen parámetros
                    int finalPos = oldString[i+1].find(")")
                    successor = rewriteParameters(
                                oldString.sub(i+1, finalPos),
                                successor)
                    newString += evaluateStr(successor)
                } else {
                    newString += successor
                }
            }
        }//fin foreach
        return newIteration(iterationsLeft - 1, newString)
    }
}
```

Figura 16: Pseudocódigo del método recursivo que genera una nueva iteración en los L-systems implementados

Igual que ocurría con los bracketed L-systems, al haber añadido los L-systems parametrizables será necesario modificar la manera de interpretar la cadena de caracteres (clase `TurtleInterpreter`), en este caso, será necesario definir un alfabeto que acepte que sus caracteres tengan parámetros asociados, el alfabeto implementado es el siguiente:

- $F(a)$: modifica el estado actual moviendo su posición a unidades hacia delante (***F**orward*), en dirección al vector *heading*.
- $L(a,b,c)$: añade una nueva hoja (***L**eaf*) a la geometría, que vendrá determinada por un rectángulo generado al rededor de la posición actual en el plano que forman por los vectores *heading* y *left*. El parámetro a indica el ancho de la hoja, la cantidad (positiva y negativa) a moverse sobre el vector *left*, el parámetro b indica el largo de la hoja, la cantidad (positiva) a moverse en el vector *heading*, el parámetro c será un valor aleatorio utilizado en la asignación de texturas, que se comentará más adelante.
- $O(a)$: modifica el estado actual estableciendo el radio de los segmentos, este radio viene determinado por el parámetro a.
- $T(a)$: indica que el estado actual debe hacer un giro ***T**urn*, es decir, al rededor del vector *up* en un ángulo de a grados.
- $P(a)$: indica que el estado actual debe hacer un giro ***P**itch*, es decir, al rededor del vector *left* en un ángulo de a grados.
- $R(a)$: indica que el estado actual debe hacer un giro ***R**oll*, es decir, al rededor del vector *heading* en un ángulo de a grados.

En el proceso de interpretación será necesaria la estructura de datos **estado interpretado**, la cual almacena la posición en el espacio, el sistema de ejes con los vectores *heading*, *left* y *up* y el radio de un estado concreto.

Con la estructura de datos anterior y el alfabeto a utilizar, ya se puede exponer como se hace la interpretación de la cadena de caracteres de un L-system, lo primero es definir un estado actual, después, habrá que recorrerse la cadena de caracteres modificando este estado, según el carácter encontrado, teniendo en cuenta que cada vez que se encuentre la operación $F(a)$, habrá que añadir el estado actual a un vector con los estados interpretados, como se muestra en la siguiente figura:

```
Entrada:
    lSystemStr: la cadena de caracteres generada en un LSystem
Salida: un vector final con los estados interpretados

function interpretate(lSystemStr) {
    // Pila de estados para ramificaciones
    Stack<State> stateStack
    // Vector final con los estados
    Vector<State> interpretatedStates
    State actualState
    foreach(char c in lSystemStr) {
        switch(c) {
        case 'F':
            actualState.position = actualState.position +
                        (actualState.heading * getParameters(c))
            // Añade el estado solo en este carácter
            interpretatedStates.add(actualState)
        case 'O':
            actualState.radius = getParameters(c)
        case 'L':
            // Añade la geometría de la hoja
            addLeaf(actualState, c)
        case 'T':
            actualState.rotateAround(actualState.up,
                                                getParameters(c))
        case 'P':
            actualState.rotateAround(actualState.left,
                                                getParameters(c))
        case 'R':
            actualState.rotateAround(actualState.heading,
                                                getParameters(c))
        case '[':
            stateStack.push(actualState)
        case ']':
            actualState = stateStack.pop()
        }
    }
    return interpretatedStates
}
```

Figura 17: Pseudocódigo con el proceso básico de interpretación de la cadena de caracteres generada por un L-system

Tener un vector de estados interpretados es algo muy importante, pero no es el objetivo de la interpretación, este vector de estados se usará como estructura auxiliar para generar la **superficie de las ramas**, de forma que, cada vez que se encuentre el carácter] (fin de la rama), habrá que generar la superficie asociada a los nuevos estados interpretados hasta el momento.

La superficie a generar, es una superficie de revolución, cuya curva generatriz está formada por las posiciones de los estados interpretados trasladadas en dirección al vector *left* y en una cantidad igual al radio del estado y cuyo eje de rotación estará formado por los vectores *heading* de cada estado, como se puede ver en la siguiente figura:

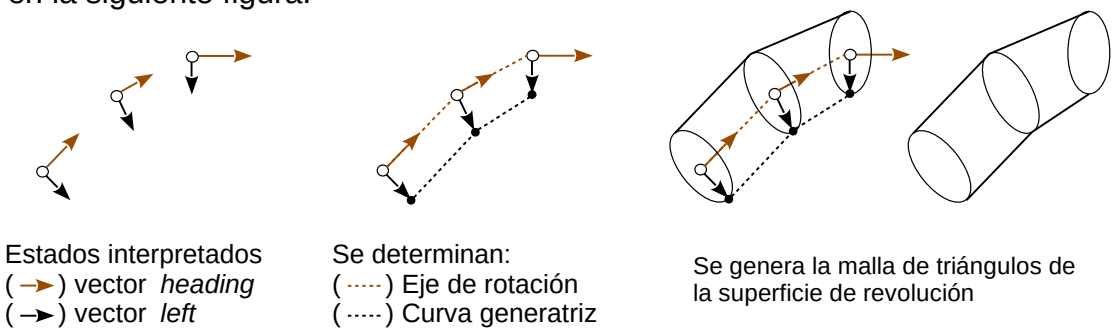


Figura 18: Proceso de generación de la superficie de revolución a partir de estados

La superficie de revolución, igual que cualquier elemento geométrico definido en la aplicación, se genera a partir de mallas de triángulos, definidas por vértices dispuestos en un orden concreto, según una primitiva de dibujo.

La malla de las ramas al fin y al cabo serán cilindros unidos, para generar estos cilindros es necesario un parámetro con el número de puntos que tendrá la base de cada cilindro, este número son los *slices* de la superficie de revolución, cuanto mayor sea, más redonda quedará la rama.

Cada vez que se añade una nueva rama, habrá que introducir un reinicio en la malla de triángulos, ya que, es posible que la siguiente rama no aparezca justo después. Donde también se añade un reinicio de malla de triángulos, es cada vez que se añade una nueva hoja, ya que la siguiente hoja es seguro que no estará a continuación de la anterior.

Aunque la geometría de las hojas y la de las ramas podrían ir en una misma malla de triángulos, ya que usan la misma primitiva geométrica: `TRIANGLE_STRIP`, se ha decido separarlas en dos mallas distintas para facilitar la labor de añadir texturas.

Por otro lado, se ha añadido una tercera malla de triángulos, esta generada con la primitiva `TRIANGLE_FAN`, que es necesaria ya que los cilindros generados con `TRIANGLE_STRIP`, no pueden tener tapas, debido a que en estos casos las tiras de triángulos deberían acabar en un mismo vértice final, lo que generará algunos triángulos degenerados, es decir, triángulos cuyos vértices están en la misma línea.

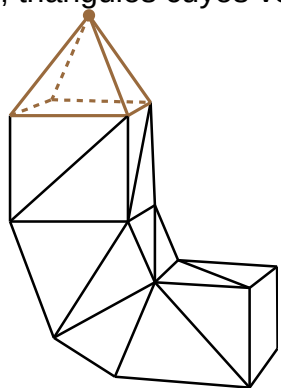


Figura 19: Estructura de la malla de triángulos de una rama con 4 slices. La parte marrón se crea con la primitiva `TRIANGLE_FAN` y la parte negra con `TRIANGLE_STRIP`

4.3 Diseño del L-system para la generación procedural de olivos

La gramática a diseñar, deberá generar la estructura de crecimiento descrita en el apartado 4.1, teniendo en cuenta los detalles de implementación del apartado 4.2.

Antes de exponer el diseño de la gramática, se definen las siguientes pautas:

- Se reservarán los caracteres en mayúsculas para definir elementos que harán algún tipo de función concreta, como los caracteres del alfabeto o los predecesores de las reglas de producción.
- Las letras en minúsculas se entenderán como variables que deberán tener algún valor numérico y en concreto los caracteres a, b, c, d, e, f se reservan para hacer referencia a los parámetros de los predecesores.
- Los símbolos de operaciones definidas o los caracteres numéricos no aparecerán como predecesores en ninguna producción.

En la siguiente tabla se exponen todos los elementos que forman parte de la gramática diseñada, que a continuación se detallarán y justificarán.

Axioma			
$O(0'01)F(1)S$			
Reglas de producción			
Predecesor	Sucesor	Probabilidad	Condición
S	$A(30)$	50%	–
S	$C(30)$	50%	–
$A(a)$	$[B(\sim(-(a,10),+(a,10)),\sim(-5,5))]$ $[B(\sim(-(-a,10),+(-a,10)),\sim(-5,5))]$ $G(\sim(0'4,0'7))C(a)$	100%	–
$B(a,b)$	$[P(b)T(a)R(\sim(0,360))L(0'07,0'4,\sim(0,1))]$ $P(b)T(/(a,2))O(/(0'01,10))F(0'4)Y$	100%	–
$C(a)$	$[D(\sim(-(a,10),+(a,10)),\sim(-5,5))]$ $[D(\sim(-(-a,10),+(-a,10)),\sim(-5,5))]$ $G(\sim(0'4,0'7))A(a)$	100%	–
$D(a)$	$[T(b)P(a)R(\sim(0,360))L(0'07,0'4,\sim(0,1))]$ $T(b)P(/(a,2))O(/(0'01,10))F(0'4)Y$	100%	–
Y	$A(30)$	30%	–
Y	$C(30)$	30%	–
Y	X	40%	–
$L(a,b,c)$	$L(+(a,0'005),+(b,0.05),c)$	–	$<(b,0'51)$
$L(a,b,c)$	$L(a,b,c)$	–	*else*
$O(a)$	$O(+(a,0'01))$	100%	–
$G(a)$	$T(\sim(-5,5))P(\sim(5,5))F(a)$	100%	–

Tabla 2: Resumen de la gramática diseñada para el modelado de olivos

El primer elemento es el axioma, que determina el estado inicial del sistema, en este, lo que se hace es establecer un radio inicial y añadir un segmento de longitud 1 hacia arriba, este segmento, se genera para tener un pequeño tronco antes de que el desarrollo del árbol comience.

Este desarrollo empezará con el reemplazo del último carácter del axioma, S, carácter que unicamente se usa para iniciar el sistema (***S**tart*) y tiene dos sucesores, cada uno con una probabilidad de ser usado del 50%.

El carácter S puede sustituirse por $A(30)$ o por $C(30)$, las dos posibilidades tienen una estructura similar y se justifican de la siguiente manera:

- Cada una de estas reglas generan **las dos ramificaciones enfrentadas** que se desarrollan en los olivos y el parámetro asociado que tienen los predecesores será el ángulo entre la rama principal y la rama hija, que en un olivo normal es de aproximadamente de 30 grados.
- Son necesarias las dos reglas, A y C, ya que cada una genera las ramas enfrentadas en un plano diferente, A lo hace en el plano *heading-up*, a través del carácter B y C en el plano *heading-left* a través de D.
- Para conseguir **la ramificación hacia un lado**, se establece, entre corchetes el carácter B ó D, dependiendo del predecesor seleccionado y a este carácter se le pasarán como parámetros dos números aleatorios:
  - El primero, en el rango $[30-10,30+10]$, que se corresponderá con el ángulo entre la rama principal y la rama hija, de forma que se tenga un ángulo aproximado a los 30 grados.
  - El segundo, en el rango $[-5,5]$, que determinará una pequeña perturbación en la inclinación de la rama.
- Para conseguir **la ramificación opuesta**, se establece la misma estructura anterior, el carácter B ó D, entre corchetes, sin embargo, aquí se cambia el ángulo, que ahora estará en el rango $[-30-10,-30+10]$, es decir, será una ángulo aproximadamente de -30, lo que generará la rama opuesta.
- Después de estas dos estructuras, está $G(\sim(0'4,0'7))$, que hará que la rama principal siga hacia delante con una longitud aleatoria en el rango $[0'4,0'7]$, este carácter se usa en lugar de F directamente, para que el segmento no vaya siempre linea recta, ya que G hace dos pequeñas rotaciones sobre los vectores *left* y *up* antes de añadir el carácter F.
- Por último, las dos reglas añaden a la regla contraria al final, en A se añade $C(a)$ y al final de C, se añade $A(a)$, para que la siguiente ramificación sea en el plano contrario a la que se acaba de hacer.

Se ha comentado que las reglas cuyos caracteres predecesores son B y D generan una ramificación a partir de dos valores numéricos, esto lo hacen gracias a su sucesor, que tiene las siguientes características:

- Lo primero, es generar la hoja que debe haber en toda bifurcación al tratarse de un olivo, para generar la hoja, primero se hacen dos giros, que dependiendo de la regla son diferentes:
  - Para B, primero se hace un giro sobre el vector *left* con el segundo parámetro, que tiene la pequeña perturbación y después, se hace el giro principal sobre el vector *up*, para así, obtener la hoja movida sobretodo en el plano *heading-up*, y muy ligeramente en el *heading-left*. Esto se hace gracias a la zona: $P(b)T(a)$.
  - Para D, primero se hace un giro sobre el vector *up* con el segundo parámetro, que tiene la pequeña perturbación y después, se hace el giro principal sobre el vector *left*, para así, obtener la hoja movida sobretodo en el plano *heading-left*, y muy ligeramente en el *heading-up*. Esto se hace gracias a la zona: $T(b)P(a)$.
- Después, en ambos casos, se hace un giro sobre el vector *heading*, con un ángulo aleatorio en el rango $[0,360]$, para que las hojas, que se añadirán después con el carácter L tengan una rotación aleatoria sobre si mismas.
- Todo lo anterior, se hace entre corchetes, ya que a continuación se añadirá una pequeña rama y el estado inicial de esta deberá ser igual al de la hoja.
- Para generar esta rama, se vuelven a hacer los giros iniciales, teniendo en cuenta que la rotación principal deberá de ser de la mitad, $/(a,2)$, para que la nueva rama quede entre la rama padre y la hoja, después se establecerá el radio de la nueva rama $O(/(0'01,10))$ y se añadirá esta a partir de un segmento recto mediante $F(0'4)$.
- Esta pequeña rama representará la yema que podrá o no convertirse en una rama de olivo, para determinar si la rama se podrá seguir desarrollando, se añade el carácter Y al final.

El carácter Y (**y**ema), que se introduce al final de las pequeñas ramas generadas con B y D tiene tres posibles sucesores, que se justifican de la siguiente manera:

- $A(30)$ ó $C(30)$, que harán que la yema siga su desarrollo como una rama común, con el mismo proceso descrito al iniciar el sistema.
- X, que hará que la yema pare su desarrollo, ya que el carácter X no define ningún sucesor en la gramática.

Las probabilidades de escoger cualquiera de los tres sucesores se han elegido de forma experimental, dando un 40% para que la rama no se reproduzca y un 60% para que la rama sí se desarrolle, con un 30% para cada opción.

El carácter L, aunque forme parte del alfabeto, se le pueden definir reglas de producción, en este diseño se le definen dos, la primera, $L(+(a,0'005),+(b,0.05),c)$, lo que hace es aumentar el tamaño de las hojas en cada iteración, hasta llegar a un límite, establecido en la condición asociada, si esta condición no se cumple, es decir, se ha llegado al tamaño máximo, se usará la segunda regla, que deja la hoja con los mismos parámetros que su predecesora, $L(a,b,c)$, así no se modifica el tamaño.

La última regla, que es parecida a la anterior, aumenta el radio de las ramas en cada iteración, $O(+(a,0'01))$, esta no tiene un límite como las hojas, ya que a lo largo del desarrollo de un árbol el radio de sus ramas siempre está creciendo. Con esta regla de producción se consigue que todas las ramas generadas tengan un radio acorde a su vejez, puesto que al generar cualquier rama, primero se establece un radio fijo y después todos se modifican de forma constante.

4.3.1 Mejoras aplicadas al diseño

Aunque este diseño inicial genera unas estructuras básicas algo parecidas a las de olivos pequeños (figura 20), se han implementado una serie de mejoras, en diferentes aspectos, parametrización, estructura y visualización que mejoran en gran medida los resultados y que a continuación se detallan.

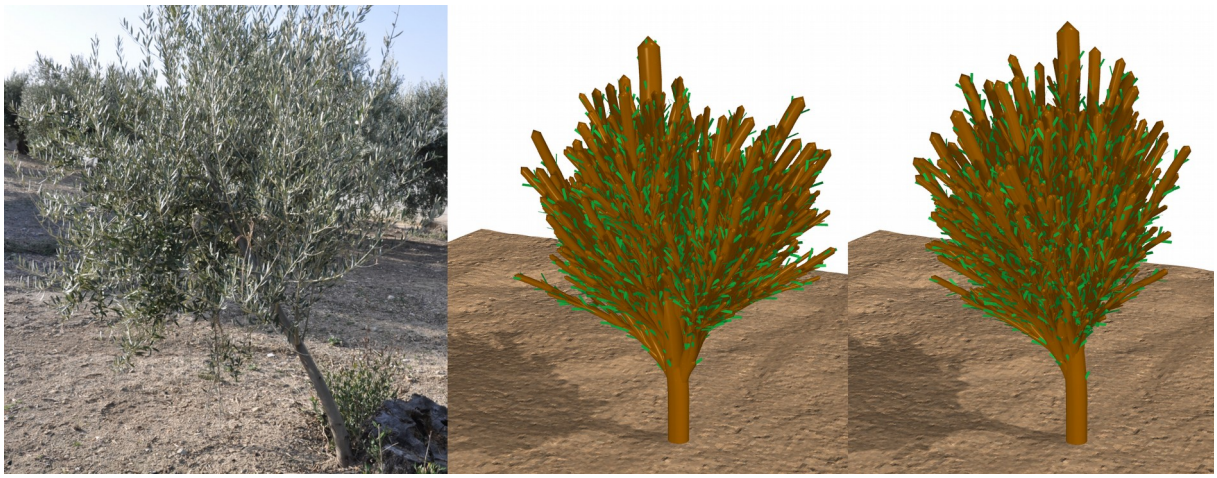


Figura 20: Izquierda: olivo joven real. Derecha: dos estructuras generadas con el diseño inicial, sin añadir mejoras

La primera mejora implementada, fue la creación de variables, que serán determinados caracteres en minúscula, que se introducirán en las reglas de producción, de forma que en el desarrollo del sistema, cada vez que aparezca una variable, se cambiará por su valor numérico, así, para modificar determinados parámetros, lo único que habrá que hacer, será modificar el valor de la variable y no la regla de producción del L-system.

Las variables definidas son las siguientes:

Variable	Valor por defecto	Uso
i	1	Determina la longitud del segmento inicial, el que aparece en el axioma
n	30	Determina el ángulo aproximado que debe haber entre las ramas hijas, se usa para: $A(n)$ y $C(n)$
s	10	Determina los valores mínimos y máximos posibles del ángulo n, que estarán en el rango: $[s-n, s+n]$
t	5	Determina los valores mínimos y máximos posibles para pequeñas perturbaciones aleatorias, estas estarán en el rango: $[-t, t]$
l	$0'7$	Será la longitud máxima de la distancia entre yemas
y	$0'4$	Será la longitud mínima de la distancia entre yemas
o	$0'1$	Determina el radio inicial
r	$0'01$	Determina el valor en el que aumentar el radio de las ramas en cada iteración

Tabla 3: Variables establecidas en el sistema

El diseño inicial del L-system deberá cambiar, reemplazando algunos valores numéricos en favor de las variables definidas, esto hará que el L-system quede de la siguiente manera:

Axioma			
$O(o)F(i)S$			
Reglas de producción			
Predecesor	Sucesor	Probabilidad	Condición
S	$A(n)$	50%	–
S	$C(n)$	50%	–
$A(a)$	$[B(\sim(-(a,s),+(a,s)),\sim(-t,t))]$ $[B(\sim(-(-a,s),+(-a,s)),\sim(-t,t))]$ $G(\sim(y,l))C(a)$	100%	–
$B(a,b)$	$[P(b)T(a)R(\sim(0{,}360))L(0'07{,}0'4,\sim(0{,}1))]$ $P(b)T(/(a,2))O(/(o,10))F(y)Y$	100%	–
$C(a)$	$[D(\sim(-(a,r),+(a,r)),\sim(-s,s))]$ $[D(\sim(-(-a,r),+(-a,r)),\sim(-s,s))]$ $G(\sim(y,l))A(a)$	100%	–
$D(a)$	$[T(b)P(a)R(\sim(0{,}360))L(0'07{,}0'4,\sim(0{,}1))]$ $T(b)P(/(a,2))O(/(o,10))F(y)Y$	100%	–
Y	$A(n)$	30%	–
Y	$C(n)$	30%	–
Y	X	40%	–
$L(a,b,c)$	$L(+(a,0'005),+(b,0.05),c)$	–	$<(a,0'1)$
$L(a,b,c)$	$L(a,b,c)$	–	*else*
$O(a)$	$O(+(a,r))$	100%	–
$G(a)$	$T(\sim(-t,t))P(\sim(t,t))F(a)$	100%	–

Tabla 4: Especificación final del L-system diseñado con variables añadidas

La siguiente mejora se ha aplicado en la parte de la interpretación del L-system y trata de **simular la gravedad** doblando hacia abajo las ramas generadas, según su peso, esto, suele ocurrir en cualquier árbol y en los olivos, ocurre especialmente, ya que estos se estructuran a partir de ramas principales, que, debido a su peso se acaban doblando.

Esta mejora se implementa en la interpretación y no en el L-system, ya que, en el L-system solamente se pueden doblar las ramas en determinadas direcciones, a partir de rotaciones, y aquí, se quiere conseguir que la rama se doble hacia abajo, tarea que sería más compleja de realizar a partir de rotaciones. Por este motivo, lo mejor es modificar geométricamente los estados interpretados, esta modificación sigue el siguiente esquema:

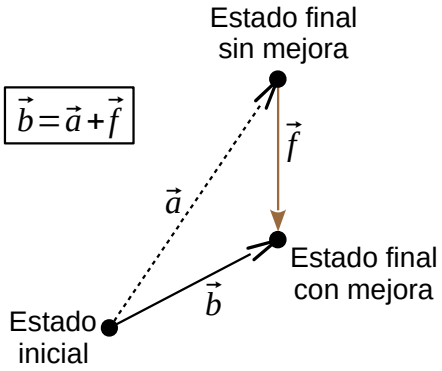


Figura 21: Funcionamiento básico de la mejora para doblar ramas según la gravedad

Cada vez que se produzca un nuevo segmento, en lugar de seguir el vector $\vec{a}$, que es conocido, se seguirá el vector $\vec{b}=\vec{a}+\vec{f}$, para ello, habrá que calcular el vector $\vec{f}$, cuya dirección es hacia abajo, $(0,-1,0)$, y cuyo módulo debería ser la fuerza a aplicar para que la rama se doble.

Para calcular el módulo del vector $\vec{f}$ se ha desarrollado la siguiente fórmula,

$$\frac{nº\_caracteres\_rama}{CORRECCIÓN \cdot usuario \cdot nº\_iteraciones^{EXP}}$$; Donde:

- $nº\_caracteres\_rama$: es el número de caracteres que tiene a la rama que se está dibujando, calculado contando los caracteres que hay entre corchetes. Este valor está en el numerador ya que cuantos más caracteres haya en una rama, más pesará y por tanto mayor deberá ser el módulo de $\vec{f}$.
- $CORRECCIÓN$ (valor = 1000): es un factor de corrección constante, que es necesario ya que $nº\_caracteres\_rama$ es un número entero y lo que se quiere obtener con esta fórmula, es una fuerza de desplazamiento que debe ser pequeña, menor que cero.

(continua)

- *usuario* : es un valor de corrección modificable, este valor se encuentra en la clase `TurtleInterpreter` y controla la susceptibilidad del árbol a doblarse, así, cuanto más cercano a cero sea este valor, las ramas del árbol se doblarán con más facilidad y al contrario si este valor es alto.
- *nº_iteraciones* : es el número de iteraciones que ha dado el L-system, este valor se comporta como otro factor de corrección, intenta corregir que entre una iteración y la siguiente, no haya una diferencia muy grande en la fuerza aplicada. Al contrario que el factor *CORRECCIÓN* , este no es constante, ya que en cada iteración la corrección a aplicar deberá ser distinta.
- *EXP* (valor = 2'2): es el exponente al que está elevado *nº_iteraciones* , este es necesario debido a que las ramas normalmente no crecen de forma lineal (una rama puede generar hasta dos nuevas), por lo que ese parámetro no puede tener una forma lineal. El valor de este exponente se ha establecido de forma experimental probando diferentes valores.

Este efecto de gravedad suele dar buenos resultados, haciendo que las ramas más pesadas disminuyan su altura con forme a su peso, aunque en algunas instancias, se producen efectos algo exagerados, pero estos se pueden solucionar modificando el factor de corrección *usuario* . Algunos ejemplos:

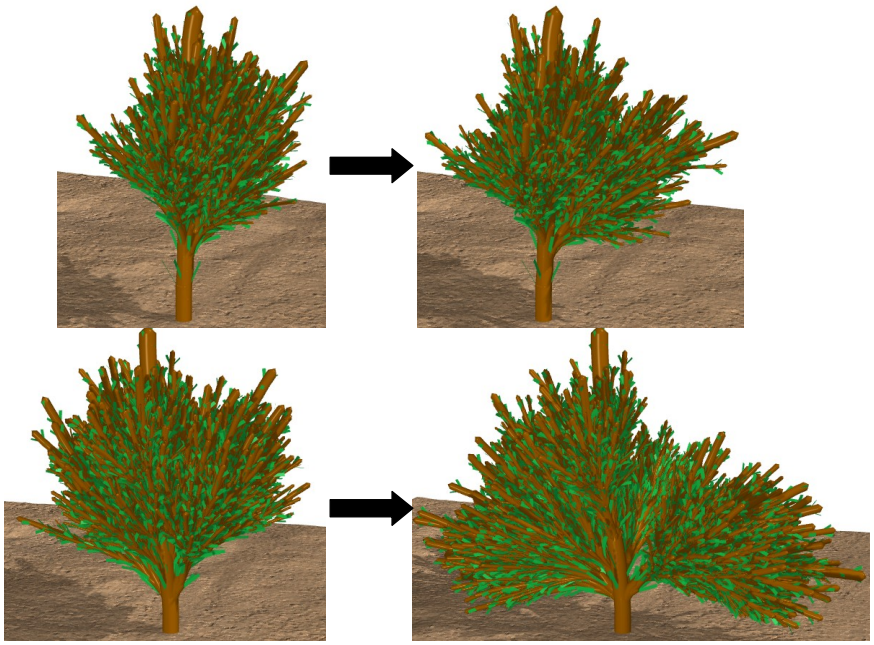


Figura 22: Dos casos diferentes en los que se ha aplicado esta mejora, el caso de abajo produce un efecto más exagerado.

La siguiente mejora, es la implementación de las **podas del olivo**, para implementar las podas en los L-system, existe un enfoque, propuesto por Hanan [2] en el que, se define un carácter especial, que, al encontrarlo durante el desarrollo del sistema, elimina todos los caracteres pertenecientes a la rama actual, es decir, quita todos los caracteres que estuviesen entre corchetes en el nivel actual o en sus niveles hijos.

Para seguir este enfoque, habría que tener reglas de producción que introdujesen este carácter en el momento y en el lugar adecuado para hacer la poda deseada. Sin embargo, en un sistema tan complejo como el que ya se tiene, definir una regla de producción de este tipo sería muy complejo, por lo que se ha decidido implementar las podas con un enfoque propio.

Este enfoque se basa en la modificación del diagrama de clases usado para implementar la funcionalidad básica de los L-system y expuesto anteriormente (figura 14), al cual, se le añadirá una clase nueva, la clase `OliveLSystem` que heredará de `LSystem` y cuyas características principales son las siguientes:

- En su constructor, mediante llamadas a su clase padre, añadirá las reglas básicas de la generación de olivos expuestas también con anterioridad. De esta forma, la clase no se corresponde con un L-system genérico, sino que este es un L-system que genera olivos, así, al usar la clase no será necesario establecer reglas de producción.
- Sobrescribirá el método `newIteration(.)` cuya función es generar una nueva iteración en el sistema:
  - En el método sobrescrito, se **realizará un pre-procesamiento** de la cadena de caracteres del sistema, para después llamar al mismo método en la clase padre y generar la nueva iteración en el L-system.
  - Este pre-procesamiento se ha usado para implementar las podas del olivo, de esta forma, cuando sea necesario, se aplicará una poda que elimine ramas de la cadena de caracteres del sistema antes de hacer la nueva iteración.

Se ha implementado unicamente la poda de formación en vaso libre, ya que es la que más aporta a la estructura general del olivo. Para implementar esta poda, se han definido tres niveles con tres acciones diferentes, que se aplicarán en orden.

La acción a realizar en el primer nivel será eliminar las ramas inferiores que nacen del tronco principal, el número de ramas a quitar lo determinará un atributo de la clase `OliveLSystem`.

En el segundo nivel, se eliminarán todas las ramas que nacen del tronco principal excepto dos, para hacer esto, se buscan dos ramas que salgan del tronco principal y que estén desarrolladas. En esta búsqueda, se cogen las primeras ramas que salgan del tronco principal y si alguna de las ramas no está desarrollada, es decir, tiene el carácter X en el primer nivel, se eliminará y se comprobará la siguiente pareja de ramas, esto se repetirá hasta encontrar una pareja con ambas ramas desarrolladas.

En el tercer nivel, para cada rama principal generada en el nivel dos, se eliminan todas las ramas hijas excepto dos, generando así una estructura básica, con un tronco del que salen dos ramas, de las que, a su vez, saldrán otras dos. El método para elegir las ramas será el mismo que el aplicado en el segundo nivel.

Para determinar cuando aplicar cada uno de estos niveles, se definen tres atributos, cada uno con un número de iteración, de forma que al alcanzar ese número de iteración, se aplique el nivel asociado. Ejemplo de poda aplicada:

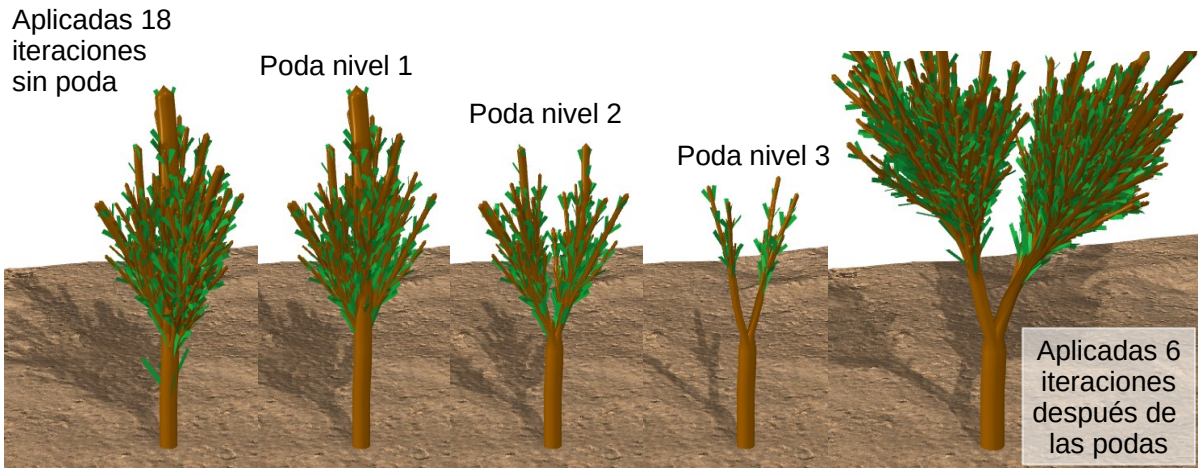


Figure 23: Ejemplo del desarrollo de las diferentes podas implementadas

Con todo lo anterior, el diagrama de clases finalmente implementado queda de la siguiente manera:

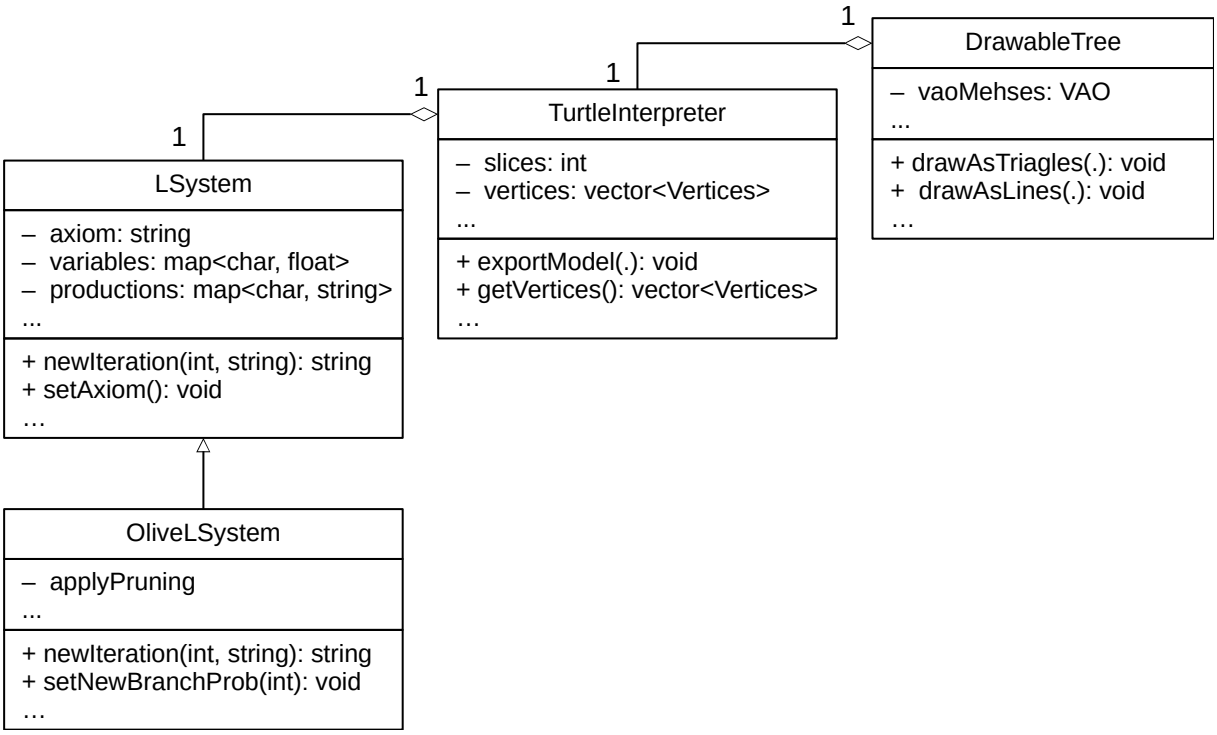


Figura 24: Diagrama de clases UML simplificado implementado

Al añadir la nueva clase, se cuenta con una funcionalidad extra, y es que, se puede cambiar la definición de las reglas de producción implementando métodos específicos, para así dar la posibilidad usuario de personalizar aún más las reglas, en concreto, se ha implementado un método que cambia la probabilidad de que una rama no se desarrolle, para así poder simular diferentes entornos.

La última mejora implementada, será la adición de **texturas** a los modelos para generar una visualización más realista, en este entorno procedural, el mejor enfoque para generar texturas lo más realistas posibles, sería a partir de una generación también procedural de las mismas, sin embargo, esta tarea es demasiado compleja como para añadirla al proyecto actual.

Por tanto, las texturas que se usarán, serán pequeñas imágenes tomadas de olivos reales, que se irán repitiendo de forma periódica a lo largo del modelo, se definirán tres texturas, una para cada malla de triángulos que tiene el modelo.

La primera textura será la de las hojas, que deberá ser semi-transparente, ya que estas se dibujan sobre planos, que no se ajustan a la forma de la hoja, además, para que no de la sensación de que las hojas son las mismas, se ha dividido la textura en cuatro zonas y cada una tendrá una hoja diferente, de esta manera:

Figura 25: Textura semi-transparente usada para las hojas

Para determinar cual de las cuatro hojas usar, se tiene el tercer parámetro del carácter L del L-system que es un valor aleatorio entre 0 y 1, de forma que cuando se va a generar la geometría de la hoja, se coge este valor aleatorio y:

- Si el valor está entre $[0, 0'25)$, se usará la primera hoja.
- Si el valor está entre $[0'25, 0'5)$, se usará la segunda hoja.
- Si el valor está entre $[0'5, 0'75)$, se usará la tercera hoja.
- Si el valor está entre $[0'5, 1]$, se usará la cuarta hoja.

Este parámetro es necesario que esté en el L-system, para que un carácter tenga siempre la misma textura, ya que si la elección de la textura se hace en la interpretación, es posible que una hoja tenga una textura en una iteración y otra diferente en la siguiente.

Las coordenadas de textura asociadas a la geometría, en la dimensión X no tendrán valores en el intervalo $[0,1]$ sino que dependerá de la hoja escogida, de forma que la coordenada X para la primera hoja estará en el rango $[0, 0'25]$, la de la segunda hoja en el $[0'25, 0'5]$ y así sucesivamente aumentando $0'25$.

Para la textura de las ramas y el tronco se ha usado un enfoque parecido, se han definido cuatro texturas con diferentes estados de la corteza del olivo y se han colocado de forma horizontal, ya que los cilindros a los que se le aplicará la textura tienen son más anchos que altos, la textura usada es la siguiente:

Figura 26: Textura usada para las ramas y el tronco

Igual que ocurría en las hojas, habrá que definir como asociar las coordenadas de texturas a la geometría, en este caso, la dimensión que no está entre 0 y 1, es la Y, que se establecerá según la antigüedad de la rama, igual que en las hojas, las coordenadas irán de $0'25$ en $0'25$ dependiendo de la textura a usar.

Además de dar color, en la zona de las ramas, se añadirá una textura para modificar la geometría a través de la técnica de displacement mapping, que aportará un mayor detalle las ramas. Las texturas usadas para el displacement mapping son las siguientes:

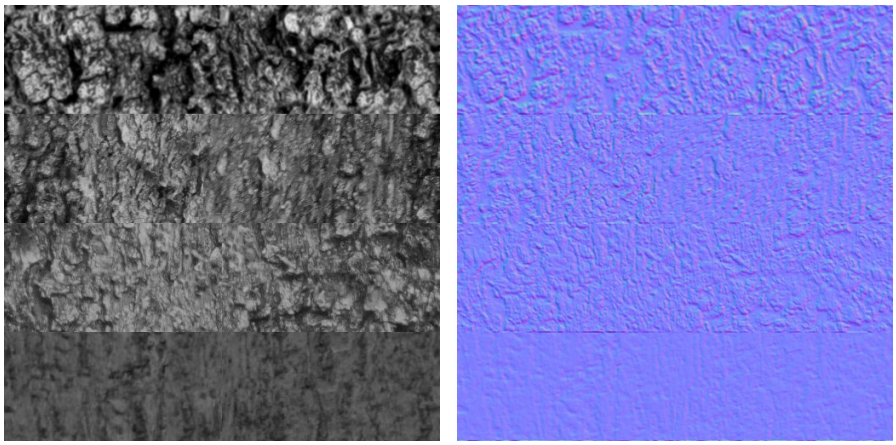


Figura 27: Texturas usadas para en el displacement mapping. Izquierda: mapa de desplazamiento. Derecha: mapa de normales

Por último, a la malla que hay al final de las ramas (las tapas), también será necesario añadirle una textura, esta vez, se ha usado una textura normal, sin dividirla en varias, ya que la zona en la que esta malla se produce es muy pequeña y no tiene un efecto muy importante sobre la visualización. La textura usada para esta zona es la siguiente:

Figura 28: Textura usada en las mallas del final de las ramas

Con las texturas, los olivos cambian de la siguiente manera:

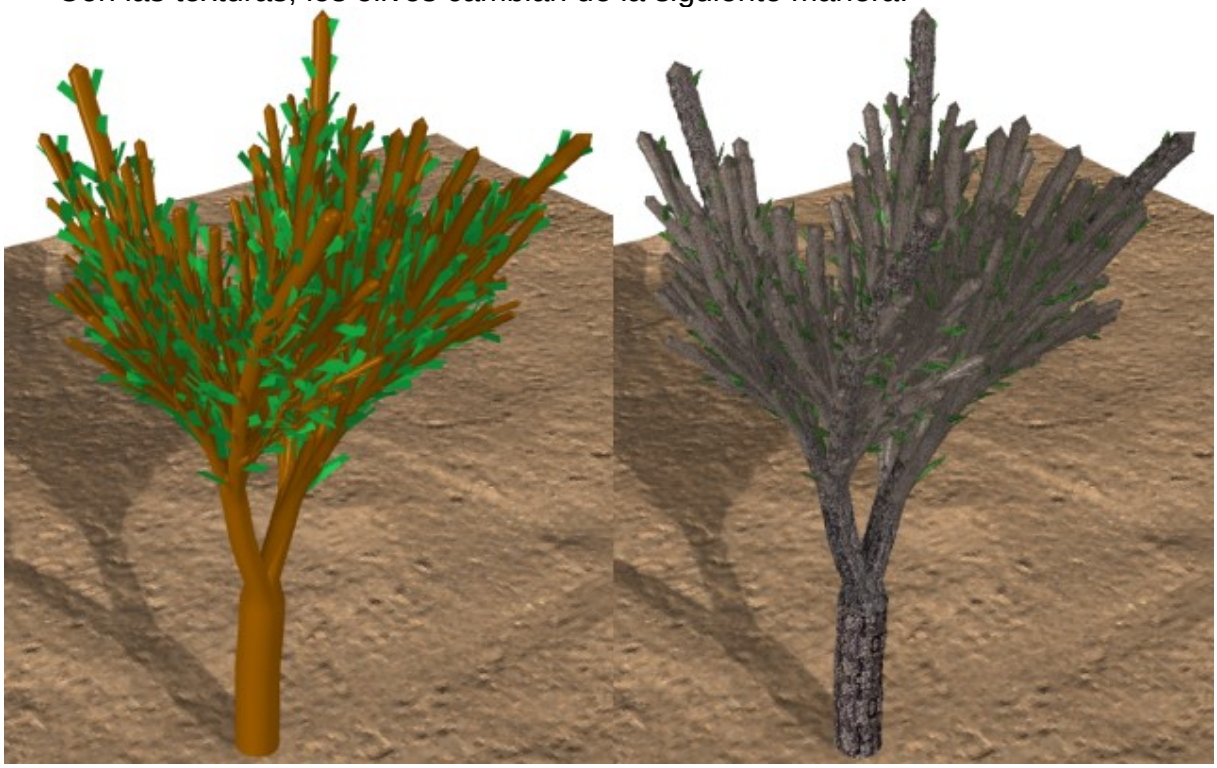


Figura 29: Comparación entre olivo sin texturas (izquierda) y olivo con texturas (derecha)

5 ESPECIFICACIÓN DEL PROTOTIPO

El primer elemento a destacar sobre el prototipo es la plataforma en la que se ha desarrollado, esta, deberá contar con funcionalidades gráficas básicas para poder visualizar los modelos 3D generados.

La plataforma elegida fue C++ con OpenGL, que aporta funcionalidades de bajo nivel, acorde al objetivo principal de la aplicación, que es generar modelos a partir de una serie de parámetros. Además, en esta plataforma ya se cuenta con bastante experiencia, de hecho, el desarrollo del prototipo no parte desde cero, lo hace desde de una aplicación propia que implementa todos los elementos básicos de un visor gráfico basado en OpenGL, esta aplicación de partida fue desarrollada para las prácticas de la asignatura de *Programación de Aplicaciones Gráficas*, impartida por el profesor Francisco de Asís Conde Rodríguez en el grado de *Ingeniería Informática de la Universidad de Jaén*.

La aplicación de partida, además de implementar la visualización de superficies triángulos, contaba con diferentes funcionalidades gráficas que se han mantenido:

- Visualización en diferentes modos (nube de puntos, modelo de alambre).
- Gestión de la cámara a partir de movimientos estándar.
- Definición y administración de luces con diferentes tipos.
- Aplicación de materiales, con o sin texturas para los modelos.
- Implementación de la técnica de displacement mapping.
- Generación de sombras proyectadas mediante shadowmaps.

La aplicación de partida sin embargo, no contaba con una interfaz gráfica de usuario que en el prototipo actual es necesaria para poder modificar los parámetros de los modelos generados de forma fácil. De esta forma se decidió añadir una interfaz a la aplicación y para ello, se recurrió a la biblioteca ImGui, que se une a las otras bibliotecas usadas en la aplicación de partida: GLEW, GLFW y GLM.

Para exponer de lo que es capaz el prototipo realizado, se detallarán todas las utilidades que hay definidas en su interfaz de usuario. La interfaz se divide en dos elementos principales, una **barra de menú** con tres sub-menús y un **panel de opciones**, que cuenta con otras tres pestañas.

El primer menú de la barra es el menú “Archivo”, a partir del cual se podrá abrir cualquiera de las pestañas del panel principal, se podrá exportar de forma rápida el modelo del olivo que se esté visualizando y se podrá cerrar la aplicación:

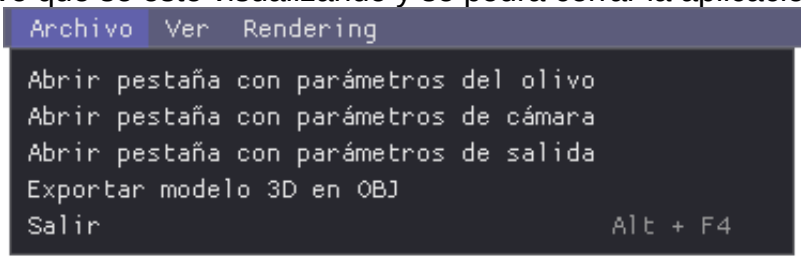


Figura 30: Captura del prototipo con el menú archivo

El menú “Ver”, permitirá mostrar u ocultar diferentes elementos, como el terreno o el panel principal. También permitirá cambiar de escena, se han definido dos escenas en la aplicación:

- Olivo parametrizable, que estará compuesta un solo olivo al cual se le pueden cambiar los diferentes parámetros que veremos a continuación.
- Pequeño olivar, que, a partir de los parámetros dados al olivo anterior, generará 25 olivos dispuestos en un orden parecido al de un olivar, para poder observar diferentes modelos juntos.

Figura 31: Captura del prototipo con el menú ver

Por último, está el menú “Rendering”, que modificará la forma en la que se dibujan los elementos en la aplicación, se definen varias opciones:

- El modo de visualización, que puede ser, en forma de nube de puntos, en forma de líneas, juntando líneas y puntos, en modo debug, para depurar la geometría o en modo luces y texturas.
- Utilizar o no la atenuación debida a la profundidad, esto solamente es compatible en el modo de líneas y en el luces y texturas.
- Visualizar o no la estructura generada por el L-system en el árbol, si se activa, al establecer el modo líneas, no se verá el modelo de alambres sino esta estructura.
- Tipo de debug, despliega un sub-menú para escoger el tipo de debug a usar en el modo debug, se puede elegir entre normales, coordenadas de texturas y tangentes, al elegir uno, el modelo se dibujará con diferentes colores para comprobar que ese parámetro es correcto.
- Luces definidas, despliega un sub-menú para encender o apagar alguna de las luces pre-definidas en el modo luces y texturas.

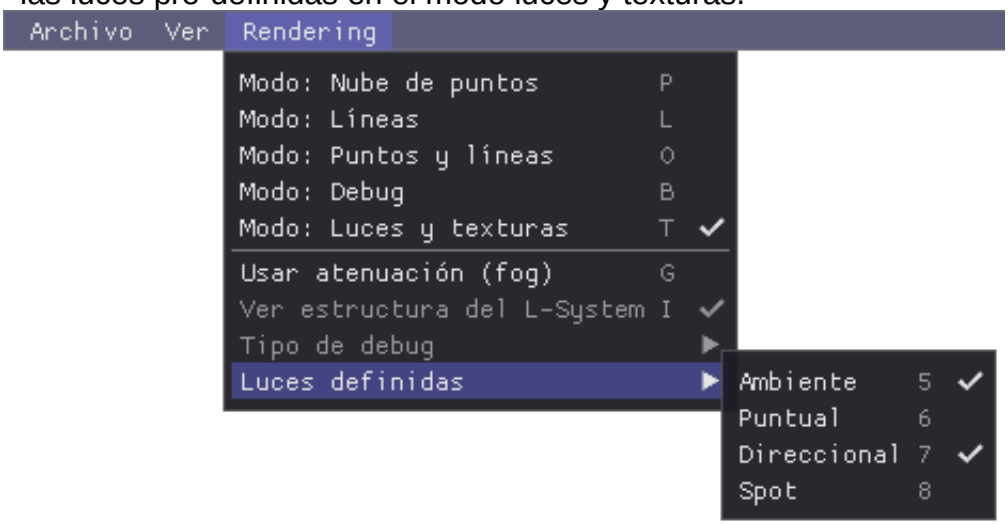


Figura 32: Captura del prototipo con el menú rendering

La pestaña más importante que hay definida en el panel, es la del olivo, que contiene todos los parámetros que se pueden modificar para generar un olivo con la aplicación. Estos parámetros son los siguientes:

- La semilla: si no se fija una semilla, cada vez que se pulse aplicar, se generarán diferentes números aleatorios, de forma que siempre se producirá una estructura diferente. Esto no ocurrirá si se fija la semilla, en cuyo caso, los números aleatorios se generarán siempre en el mismo orden para una semilla determinada.
- Visualizar texturas: se podrán usar o no las texturas definidas para los árboles, pulsando en el botón asociado.
- El número de iteraciones a aplicar: cuando se pulse aplicar, se generarán el número de iteraciones especificadas en este campo.
- Parámetros de la poda: aquí se podrán especificar cuando se deberá hacer la poda de cada tipo.
- Parámetros del L-system: aquí se podrá modificar cualquier componente del L-system, desde una variable como el ángulo entre ramas, hasta la probabilidad de que las ramas no se desarrollen.
- Parámetros del interpretador: aquí hay menos parámetros que en la zona anterior, y estos son, el número de subdivisiones a generar en el árbol, el número de *slices* y el factor de gravedad.

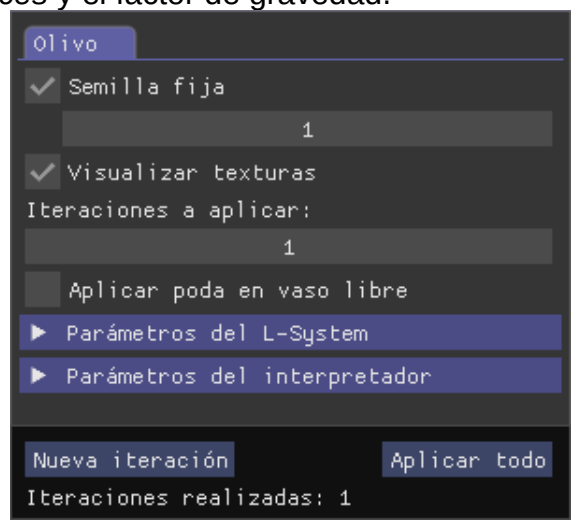


Figura 33: Captura del prototipo con la pestaña de olivo

Otra pestaña que puede abrirse en el panel es la que tiene los parámetros de la cámara virtual, en esta pestaña se pueden modificar:

- La posición de la cámara y el punto al que se mira, que aunque se puedan modificar a través de eventos de ratón, aquí se pueden cambiar coordenada a coordenada o establecer una posición fija de las cuatro definidas.
- El valor de los planos de recorte cercano y lejano.
- El campo de visión horizontal en grados.
- El color del fondo cuando.
- Los valores de profundidad para la atenuación debida a la profundidad siempre que esta esté activada.

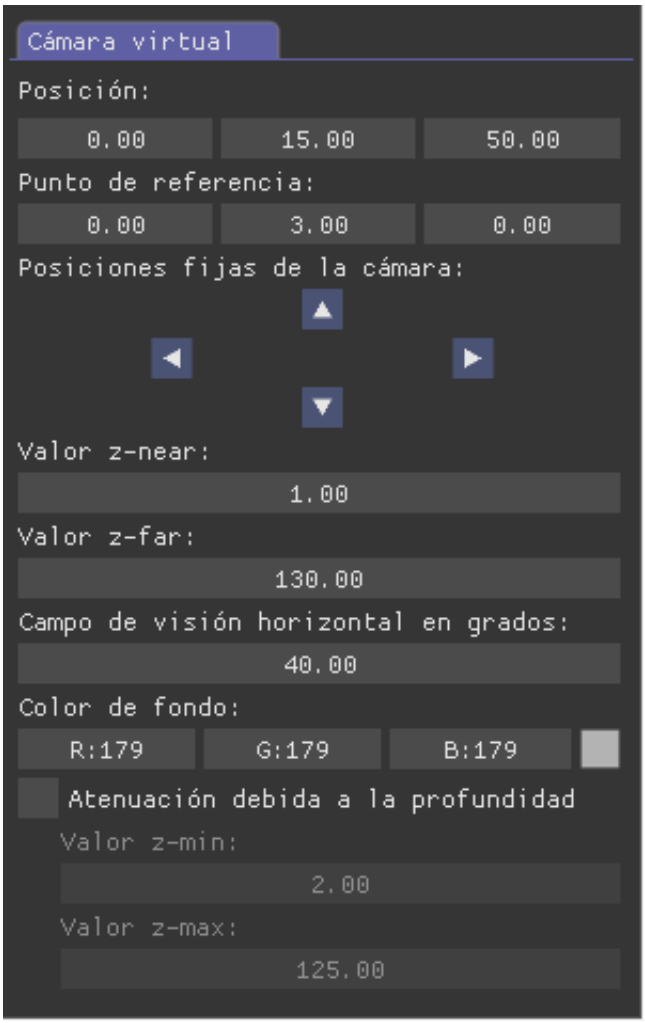


Figura 34: Captura del prototipo con la pestaña de cámara virtual

La última pestaña del panel principal, es la que genera elementos de salida, se podrán generar dos tipos de elementos:

- El modelo del olivo en formato OBJ, que actualmente se está visualizando, pudiendo elegir las mallas que exportar y el nombre del archivo.
- Una captura de la aplicación a través de un renderizado a textura, pudiendo elegir la resolución de la imagen.

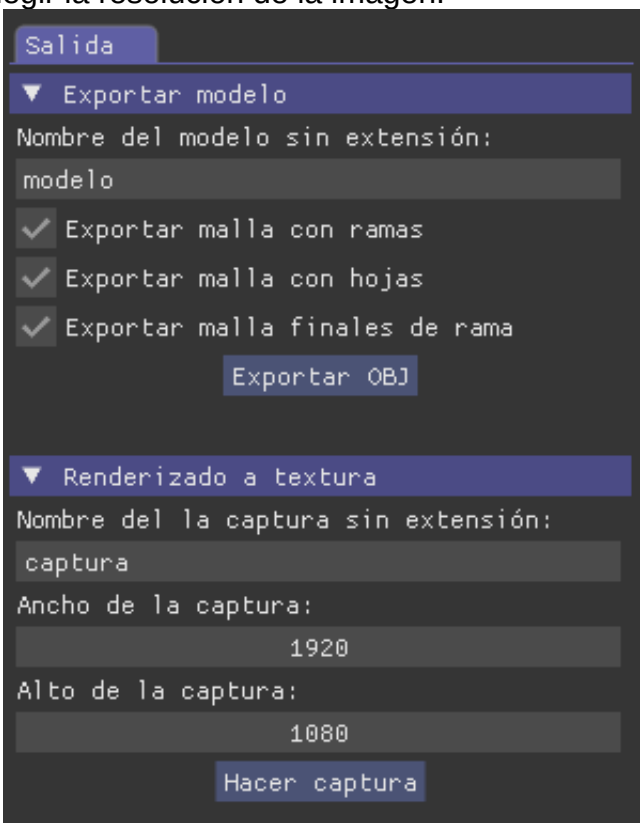


Figura 35: Captura del prototipo con la pestaña de salida

Además de la interacción a través de la interfaz gráfica, en el prototipo soporta las siguientes interacciones a partir de teclado y ratón:

Control de cámara	
Tecla	Función
Ratón–izq	Hace que la cámara haga el movimiento de orbit
Ratón–der	Hace que la cámara haga el movimiento de pan/crane
Flecha–arriba	Hace que la cámara haga el movimiento de orbitar hacia arriba
Flecha–abajo	Hace que la cámara haga el movimiento de orbitar hacia abajo
Flecha–izq	Hace que la cámara haga el movimiento de orbitar hacia la izquierda
Flecha–der	Hace que la cámara haga el movimiento de orbitar hacia la derecha
RePag	Hace que la cámara haga el movimiento de crane hacia abajo
AvPag	Hace que la cámara haga el movimiento de crane hacia arriba
Inicio	Hace que la cámara haga el movimiento de truck hacia la izquierda
Fin	Hace que la cámara haga el movimiento de truck hacia la derecha
Insert	Hace que la cámara haga el movimiento de dolly in
Supr	Hace que la cámara haga el movimiento de dolly out
W	Hace que la cámara se mueva en el mundo hacia delante
A	Hace que la cámara se mueva en el mundo hacia la izquierda
S	Hace que la cámara se mueva en el mundo hacia atrás
D	Hace que la cámara se mueva en el mundo hacia la derecha
Espacio	Hace que la cámara se mueva en el mundo hacia arriba
Shift + Espacio	Hace que la cámara se mueva en el mundo hacia abajo
F	Hace que la cámara haga el movimiento de tilt hacia abajo
R	Hace que la cámara haga el movimiento de tilt hacia arriba
E	Hace que la cámara haga el movimiento de pan hacia la derecha
Q	Hace que la cámara haga el movimiento de pan hacia la izquierda
+	Hace que la cámara haga zoom in
–	Hace que la cámara haga zoom out
1	Sitúa la cámara en el punto fijo 1 (en frente)
2	Sitúa la cámara en el punto fijo 2 (izquierda)
3	Sitúa la cámara en el punto fijo 3 (derecha)
4	Sitúa la cámara en el punto fijo 4 (detrás)

Atajos de teclado varios	
Telca	Función
P	Activa/desactiva el renderizado en modo nube de puntos
L	Activa/desactiva el renderizado en modo modelo de alambres
O	Activa/desactiva el renderizado en modo nube de puntos junto con el modo modelo de alambres
B	Activa/desactiva el renderizado en modo debug
I	Activa/desactiva la visualización de la estructura del L-system en modo de alambres
G	Activa/desactiva la atenuación debida a la profundidad
T	Activa/desactiva el renderizado en modo texturas + luces
5	Enciende/apaga la luz ambiente
6	Enciende/apaga la luz puntual
7	Enciende/apaga la luz direccional
8	Enciende/apaga la luz spot
K	Abre/cierra el panel principal con la interfaz
Alt + F4	Cierra la aplicación

Tabla 5: Resumen de las teclas que se pueden usar en la aplicación

6 PLANIFICACIÓN

6.1 Estimación de tiempos

El proyecto se ha desarrollado con una metodología incremental, en la que el trabajo se divide una serie de iteraciones que aportan nuevas funcionalidades al prototipo final. Estas iteraciones se han subdividido a su vez en diferentes tareas para una mejor planificación temporal, la división del trabajo es la siguiente:

- Iteración 1 – Implementación del método de generación de árboles 2D:
  - Adaptación de la aplicación de partida.
  - Implementación de L-systems.
  - Implementación de turtle interpretation 2D.
  - Experimentación generando algunas estructuras 2D.
- Iteración 2 – Extensión del método de generación de árboles a 3D:
  - Implementación de turtle interpretation 3D.
  - Extensión del esqueleto de árbol (nube de puntos) a superficies.
  - Experimentación generando algunas estructuras 3D.
  - Inclusión de la biblioteca de interfaz de usuario.
- Iteración 3 – Diseño del L-systems que genera olivos:
  - Estudio del crecimiento de olivos.
  - Diseño del L-system y experimentación.
  - Definición de la interfaz de usuario principal.
- Iteración 4 – Aplicación de mejoras:
  - Implementación de gravedad.
  - Implementación de diferentes tipos de podas.
  - Ampliar interfaz para ajustarse a nuevas funcionalidades.

La estimación temporal se resume en el siguiente diagrama de Gantt:

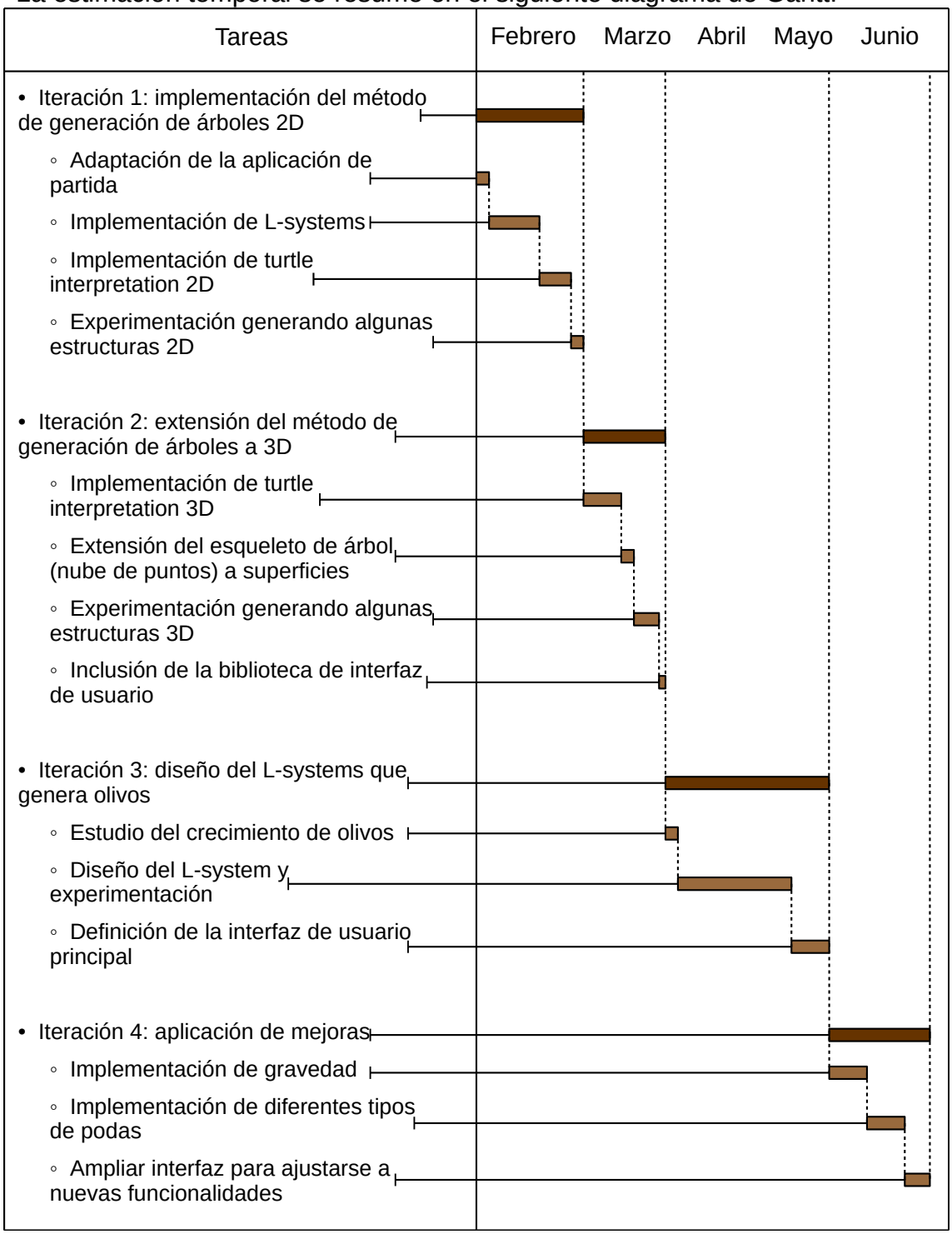


Figura 36: Diagrama de Gantt con la estimación de tiempos

6.2 Estimación de costes

Para la realización del proyecto han sido necesarios los siguientes recursos:

- Puesto de programador junior: cuyas tareas son las de análisis e implementación de los aspectos técnicos del proyecto. Este es el puesto: área 3, grupo E, nivel 1, que según el BOE [5], tiene un salario base establecido por convenio en 2019 de unos 14 817 € al año, dado que se estima que el proyecto tendrá una duración de cinco meses, este recurso tendrá un coste aproximado de 6 170 €.
- Puesto de jefe de proyecto: cuyas tareas son las de dirección y supervisión del proyecto. Este es el puesto: área 3, grupo A, nivel 1, que según el BOE [5], tiene un sueldo base en 2019 de 25 035 € al año, pero, dado un puesto de supervisión normalmente incluye varios proyectos, para la estimación de costes de este se tendrá en cuenta solamente en un mes de trabajo, lo que aporta un coste aproximado de 2 086 €.
- Recursos hardware: se trata principalmente del ordenador con el que se ha realizado el proyecto, este tuvo un costo de unos 900 € y se estima que el mismo tendrá una vida útil de 4 años, dado que el proyecto dura cinco meses, el coste de este recurso sera de unos 94 €.

Sumando los recursos anteriores se tiene un coste estimado del proyecto en unos 8 349 €, a continuación se muestra una tabla resumen con los recursos utilizados y su coste final:

Recurso	Coste final
Puesto de analista programador junior	6 170 €
Puesto de jefe de proyecto	2 086 €
Recursos hardaware	94 €
Total:	8 349 €

Tabla 6: Resumen de la estimación de costes

7 RESULTADOS Y CONCLUSIONES

Los resultados obtenidos se valorarán según el grado de cumplimiento de los objetivos definidos al inicio del proyecto.

Los estudios realizados han sido de vital importancia para llegar a los resultados finales, en ellos se ha intentado exponer en detalle todos los elementos que han sido de utilidad a la hora de hacer una implementación propia.

En cuanto a los modelos generados con el prototipo, se considera que, en general, son de buena calidad, ya que a simple vista, estos modelos se llegan a parecer a olivos reales, como en los casos que se muestran a continuación:

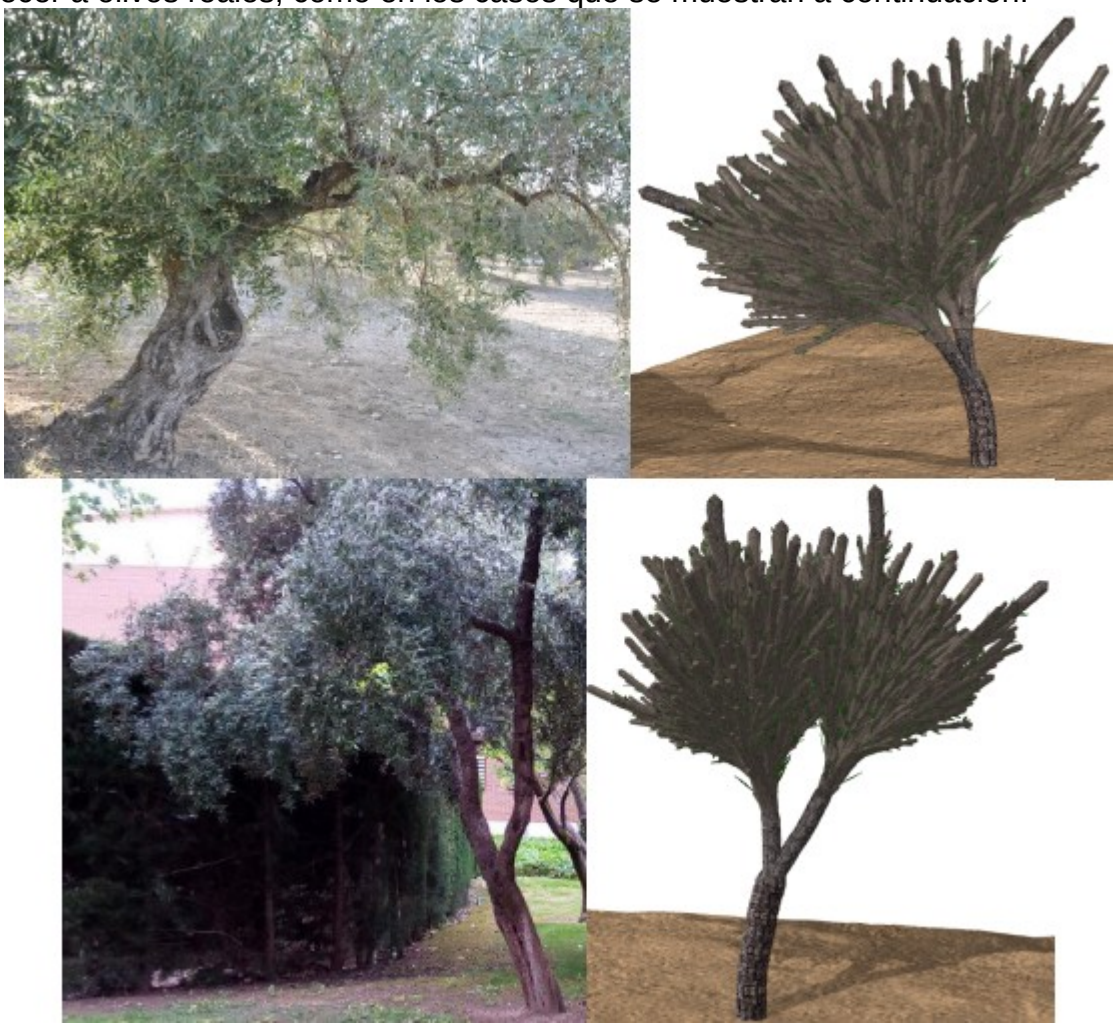


Figura 37: Comparación entre olivos reales y modelos generados

En la figura anterior, se pueden observar dos olivos reales, que son muy diferentes entre ellos y que de cierta manera se pueden simular mediante el prototipo desarrollado, y es que existe una gran variedad de estructuras a generar a partir de los parámetros definidos.

Con los parámetros definidos, además de diferentes estructuras básicas, se podrán simular determinados entornos, por ejemplo:

- Para simular un entorno en el que el suelo no aporta los suficientes nutrientes al olivo, bastará con aumentar la probabilidad de que las ramas no se desarrollen.
- Los olivos no siempre tienen el mismo peso en sus ramas, si un olivo esta produciendo fruto, sus ramas pesarán más, esto se puede simular gracias al factor de gravedad implementado.

Si bien, existirán ocasiones en las que el modelo falle, dando resultados no realistas o imposibles en la realidad, estos son los menos casos y cuando esto ocurre, normalmente es debido a alguno de los dos siguientes parámetros:

- Las iteraciones en las que aplicar las podas, cuando las podas se realizan con iteraciones bajas, al haber pocas posibles ramas a quedarse, es posible que se eliminen todas, generando una estructura que no se desarrolla.
- La probabilidad de que una yema no se reproduzca, este parámetro es muy importante, ya que determinará en gran medida el número de ramas que tendrá el olivo.

Por último, en cuanto al realismo de los modelos, como ya se comentó, se podría mejorar incluyendo algún tipo de generación procedural de texturas, ya que aunque las que se han incluido aportan bastante realismo, modificando tanto la apariencia como la geometría del modelo, es evidente que se repiten a lo largo del modelo:

Figura 38: Vista en detalle un tronco generado que tiene texturas

8 BIBLIOGRAFÍA Y REFERENCIAS

[1] Przemyslaw Prusinkiewicz y Aristid Lindenmayer (1996). *The Algorithmic Beauty of Plants*, capítulos 1 y 2. Disponible en: http://algorithmicbotany.org/papers/abop/abop.pdf

[2] James Hanan (1992). *Parametric L-systems and Their Application To the Modelling and Visualization of Plants*. Disponible en: http://algorithmicbotany.org/papers/hanan.dis1992.pdf

[3] Adam Runions; Brendan Lane y Przemyslaw Prusinkiewicz (2007). *Modeling trees with a space colonization algorithm*. Disponible en: http://algorithmicbotany.org/papers/colonization.egwnp2007.large.pdf

[4] Diego Barranco Navero; Ricardo Fernandez Escobar y Luis Rallo Romero (2017). *El cultivo del olivo 7ª edición*, capítulos 2 y 14.

[5] *Boletín Oficial del Estado, 6 de marzo de 2018, núm. 57, sec. III*, última página. Consultado en junio de 2019. Disponible en: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf