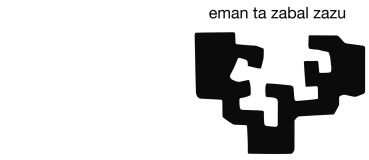


Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Generación Procedural de vegetación en entornos 3D

Autor

Gonzalo Piérola Azanza

2018

Resumen

Este trabajo estudia la generación de modelos realistas de vegetación para entornos virtuales, tales como la animación por computador o los videojuegos. En estos entornos surge a menudo la necesidad de crear paisajes con una alta variedad e vegetación, como podrían ser bosques o selvas. Para obtener un resultado mas realista es necesario introducir cierta variación entre los diferentes modelos generados para cada entorno. Realizar estas variaciones de forma manual es un proceso demasiado costoso, ya que requiere diseñar y modelar cada uno de los objeto de la escena.

Para evitar tener que modelar cada objeto a mano, se utiliza un algoritmo de generación procedural, mediante el cual es posible generar modelos de forma automática. Debido a que existe una gran variedad de vegetación, y no es posible abarcar toda, este proyecto se centra en la generación de modelos de árboles.

En este trabajo se ha implementado el algoritmo conocido como "Space Colonization" [Runions et al., 2007]. Se ha elegido porque ese algoritmo muestra similitudes con el proceso de crecimiento de los árboles en la naturaleza. Además, el algoritmo "Space Colonization" ofrece un gran control sobre la forma final del modelo.

Índice general

Índice de figuras

1. CAPÍTULO

Introducción

El primer paso para realizar este trabajo es analizar el problema a resolver, para ello hay que plantear el modo de llevar a cavo los objetivos planteados. Una vez definido el problema se analiza el modo de resolverlos, mediante una labor de investigación y se define el método que se va a utilizar para llevar a cabo el trabajo.

1.1. Descripción del Problema

El problema que se pretende resolver en este proyecto es el de generar modelos de vegetación realistas que muestren variaciones entre si.

Este problema surge de la necesidad de crear grandes entornos virtuales poblados de vegetación (bosques, selvas, ...). En muchos casos se opta por repetir un solo modelo o un pequeño conjunto de modelos, aplicándoles transformaciones como el escalado o la rotación del modelo original. Este tipo de soluciones causa a menudo un resultado menos realista, ya que esta poblado por modelos de plantas y árboles que no muestran ninguna variación.

Hay que descartar también la opción de crear los modelos de forma manual, debido a la inmensa cantidad de elementos que puede contener un escenario como un bosque: diseñar y modelar cada uno de los modelos a mano supondría demasiado trabajo. Además, este método supondría un gasto de memoria excesivo, ya que habría que guardar cada uno de los modelos diseñados.

En este proyecto se explora la opción de generar los modelos de los árboles de forma procedural. Para esto es necesario un sistema capaz de de crear la estructura de un árbol, y a partir de esta estructura generar un modelo 3D para dicho árbol.

La forma de generar los modelos debe de seguir ciertas normas, ya que el objetivo es que estos se asemejen lo máximo posible a los árboles que nos encontramos en la naturaleza:

- **Crecimiento hacia arriba**: El modelo debe de partir desde una superficie y crecer hacia arriba, simulando el modo en el que los árboles crecen en dirección a la luz del sol que se encuentra encima de ellos. El crecimiento, no pude ser siempre en la misma dirección, ya que esto le restaría realismo al modelo, por lo que es necesario introducir ligeras perturbaciones en la dirección de crecimiento.

- **Estructura de ramas**: Del mismo modo que los árboles se dividen cada vez en mas ramas conforme van creciendo, los modelos generados mediante este sistema deben imitar este comportamiento creando divisiones a lo largo del modelo. También es importante calcular el tamaño de cada una de las ramas de modo que se vayan estrechando conforme llegan a su final.

Figura 1.1: Visualización del crecimiento y ramificación de un árbol a lo largo de su crecimiento

- **Parámetros de modificación**: Existen una gran variedad de especies de árboles, cada una de ellas con sus características propias, es por eso que surge la necesidad de modificar ciertos parámetros a la hora de generar el modelo, como la altura máxima, la frecuencia de división de las ramas y tantos otros. Estos modificadores permiten definir la especie del árbol que se va a generar.

Figura 1.2: Diferencias entre distintas especies de árboles

En la siguiente figura, se muestran dos modelos generados mediante la aplicación desarrollada en este proyecto.

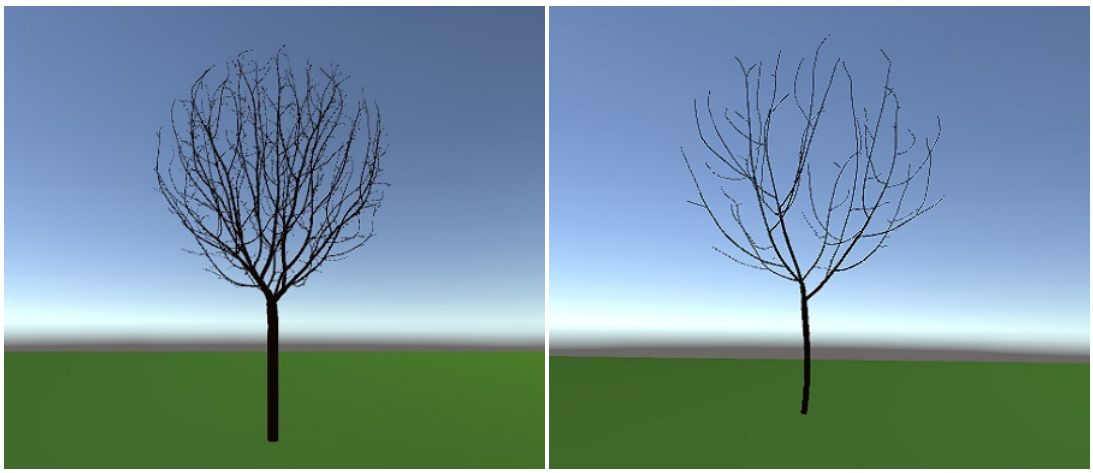


Figura 1.3: Modelos de diferentes árboles generados de manera procedural

1.2. Investigación previa

1.2.1. Proyectos similares en ADDI

Se ha realizado una búsqueda en la plataforma de ADDI, para comprobar si se ha realizado antes algún trabajo relacionado con este. Aquí se han encontrado dos trabajos relacionados con generación procedural, "Generación procedural de variaciones en modelos 3D" [Santesteban, 2017] y "Generación procedural de ciudades" [Vázquez, 2013], aunque ninguno de ellos trata acerca de generar vegetación por lo que no resultan de ayuda para realizar este trabajo. También se ha encontrado un trabajo titulado "Simulación de Materiales Naturales mediante Texturas Volumétricas Procedurales" [Crego, 2017], el cual podría resultar interesante para el desarrollo de los materiales del modelo, pero este trabajo se centra en materiales con apariencia de madera procesada, por lo que también se ha descartado.

1.2.2. Sistemas de generación procedural

En la fase de investigación previa al desarrollo de esta aplicación, se han tenido en cuenta diferentes sistemas de generación procedural. Estos sistemas no están directamente relacionados con la generación de plantas pero algunos sí que guardan cierta relación con ella como podría ser la generación procedural del terreno.

El primer sistema a tener en cuenta es el de la generación procedural del terreno, para el cual se ha tomado como referencia el articulo "Realtime Procedural Terrain Generation" [Olsen, 2004]. En este articulo se describe un método para generar terreno fractal y de apariencia natural. Parte de un trabajo anterior y desarrolla un nuevo método que genera en tiempo real el modelo del terreno

Por otro lado, en el articulo "Real-time Procedural Generation of 'Pseudo Infinite' Cities" [Greuter et al., 2003], se describe un método para generar ciudades con grandes edificios de manera procedural. Este sistema permite crear ciudades visualmente interesantes y con una amplia diversidad de edificios complejos, diseñados a partir de elementos mas simples.

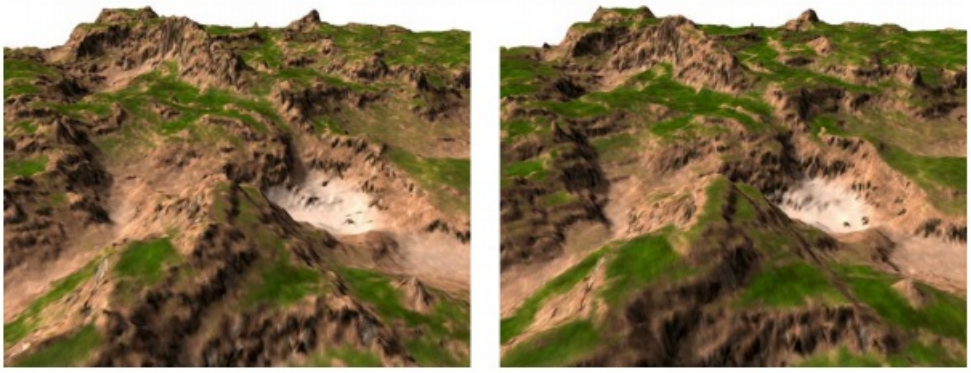


Figura 1.4: Imagen del articulo “Realtime Procedural Terrain Generation”: El terreno generado por el algoritmo original (izquierda) y el de tiempo real (derecha)

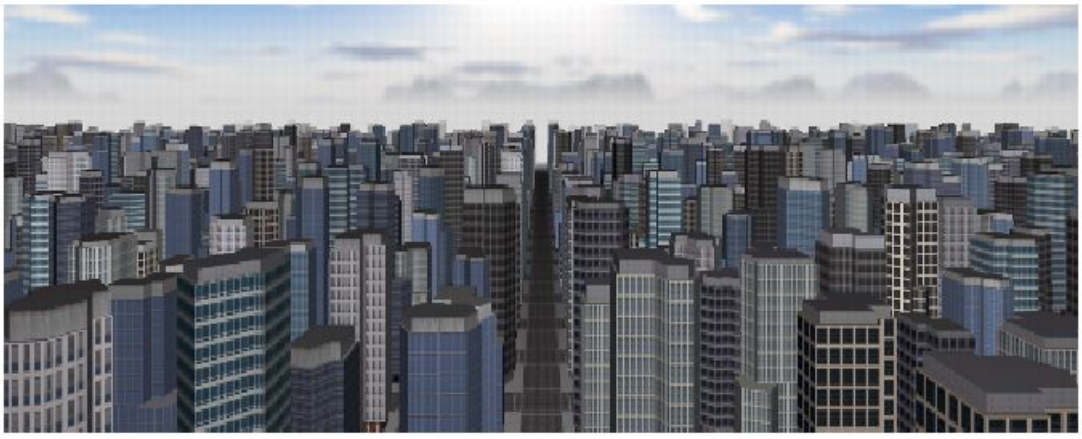


Figura 1.5: Imagen del artículo “Real-time Procedural Generation of ‘Pseudo Infinite’ Cities” en la cual se muestra una ciudad generada mediante el método descrito en el artículo

Finalmente, en el libro The Algoritmic Beauty of Plants se describen el concepto de L-System, una gramática formal creada con el fin de proveer una descripción formal del desarrollo de diferentes tipos de plantas. Un L-System esta definido por los siguientes elementos (se explica con un ejemplo):

- V (variables): Es el conjunto de símbolos que son sustituidos en cada iteración del sistema.
- S (constantes): Es el conjunto de símbolos que se mantienen constantes.
- ω (axioma): Se trata del conjunto de símbolos con los que comienza el sistema.
- P (reglas): Es un conjunto de reglas mediante las que se sustituyen las variables por un conjunto de símbolos.

Un ejempo de los L-System es el árbol binario, el cual esta definido a partir de los siguientes parámetros:

- V: 0, 1
- S: [,]
- ω: 0
- P: $(1 \rightarrow 11)$, $(0 \rightarrow 1[0]0)$

Proceso de generación a partir del axioma:

Axioma: 0

Primera iteración: 1[0]0

Segunda iteración: 11[1[0]0]1[0]0

Tercera iteración: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

Esta gramática define el modo de dibujar un árbol binario, para ello se le da un significado a cada uno de los símbolos empleados. Para realizar este dibujo se utiliza un método de dibujo vectorial conocido como “Turtle Graphics”, el cual consiste en trazar un segmento a partir de un punto de origen, una orientación. En el caso de el árbol binario, los símbolos que constituyen su gramática tienen el siguiente significado:

- 0: dibuja un hoja
- 1: dibuja un segmento de rama
- [: guarda la posición y ángulo actual y gira 45 grados a la izquierda.
-]: recupera la ultima posición y ángulo guardados y gira 45 grados a la derecha.

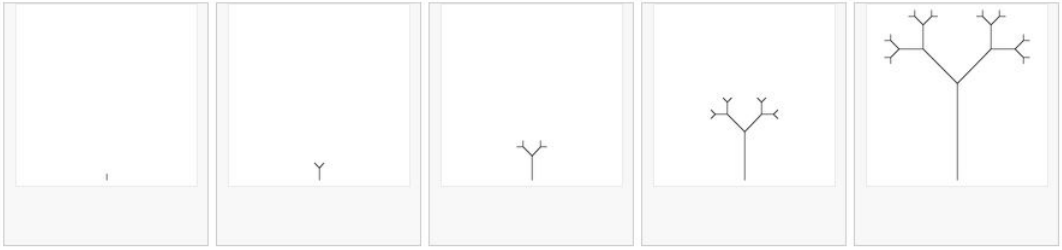


Figura 1.6: Imagen obtenida de Wikimedia Commons acerca de L-Systems, en la cual se observan las diferentes iteraciones del L-System, siendo la primera el axioma y la ultima el resultado de la cuarta iteración

1.3. Herramientas utilizadas

Para poder realizar este proyecto se ha hecho uso de diversas herramientas, tanto para realizar desarrollar la aplicación en si como para otras labores relacionadas con el proyecto, como es la elaboración de la memoria.

La principal herramienta utilizada para el desarrollo de la aplicación ha sido Unity, un motor de videojuegos gratuito destinado al desarrollo de videojuegos así como de aplicaciones gráficas como es el caso de este proyecto.

La elección de esta aplicación ha sido tomada por varios factores, siendo el primero que se trata de una aplicación de uso gratuito, lo cual a parte de facilitar su obtención hace que cuente con un gran numero de usuarios, esto resulta de gran ayuda a la hora de realizar proyectos de Unity, pues al contar con una gran comunidad resulta mas fácil encontrar información acerca de como utilizarla. Otro factor de gran importancia es el de que aparte de la ayuda de otros usuarios Unity cuenta con una API [Unity Technologies, 2018], en la cual se describe detalladamente el funcionamiento de todos los elementos que proporciona. Además de esto, Unity cuenta con una gran variedad de tutoriales dirigidos a realizar pequeños proyectos lo que ayuda a aprender como utilizar la aplicación de forma correcta.

La aplicación se ha desarrollado utilizando la ultima versión de Unity disponible en la fecha de inicio de su desarrollo, Unity 2018.2.

Para escribir el código de los scripts de Unity se ha utilizado el entorno de desarrollo Rider desarrollado por JetBrains. La elección de este entorno se basa en la experiencia con otros entornos desarrollados por esta misma compañía, lo cual ha hecho que resulte mas sencillo su uso. Este editor se puede configurar fácilmente para trabajar con el en Unity siguiendo la siguiente guía [Matt Ellis, 2017]

Para escribir la memoria se ha utilizado la plantilla de LaTex proporcionada por la Facultad de Informática de San Sebastián, además del editor online ShareLaTex. Este editor permite escribir documentos LaTex online además de compartirlos.

Finalmente cabe mencionar otras herramientas utilizadas aunque en menor medida que las anteriores. La primera de ellas es el programa de modelado 3D Blender para realizar algunos de los modelos utilizados para generar los árboles. La segunda es la librería de JavaScript para crear gráficos, la cual permite crear aplicaciones gráficas simples de forma sencilla. Esta libreria cuenta con una documentación muy detallada de cada una de sus funciones [Processing Foundation,]. Con ella se han generado algunas de las imágenes utilizadas para explicar ciertos conceptos en la memoria.

2. CAPÍTULO

Generación de la estructura del árbol

El primer paso para generar el modelo de un árbol, es definir la estructura de este. El objetivo de este proyecto es el de generar esta estructura de manera automática, y que además se asemeje a la estructura de los diferentes árboles que podemos encontrar en la naturaleza.

Para este propósito se ha utilizado el algoritmo descrito en el articulo “Modeling Trees with a Space Colonization Algorithm” [Runions et al., 2007]. Este algoritmo describe un sistema capaz de generar modelos de árboles a partir de una serie de puntos colocado de manera aleatoria a lo largo de un volumen. Este método se puede equiparar con la forma en la que se desarrollan los arboles en la naturaleza, ya que en la naturaleza es necesario competir por el espacio con el fin de conseguir la mayor cantidad de luz posible. Los puntos definidos en este método simulan la fuente de luz natural que un árbol intenta obtener, por lo tanto el árbol irá creciendo en dirección a esta, intentando alcanzar todos los puntos.

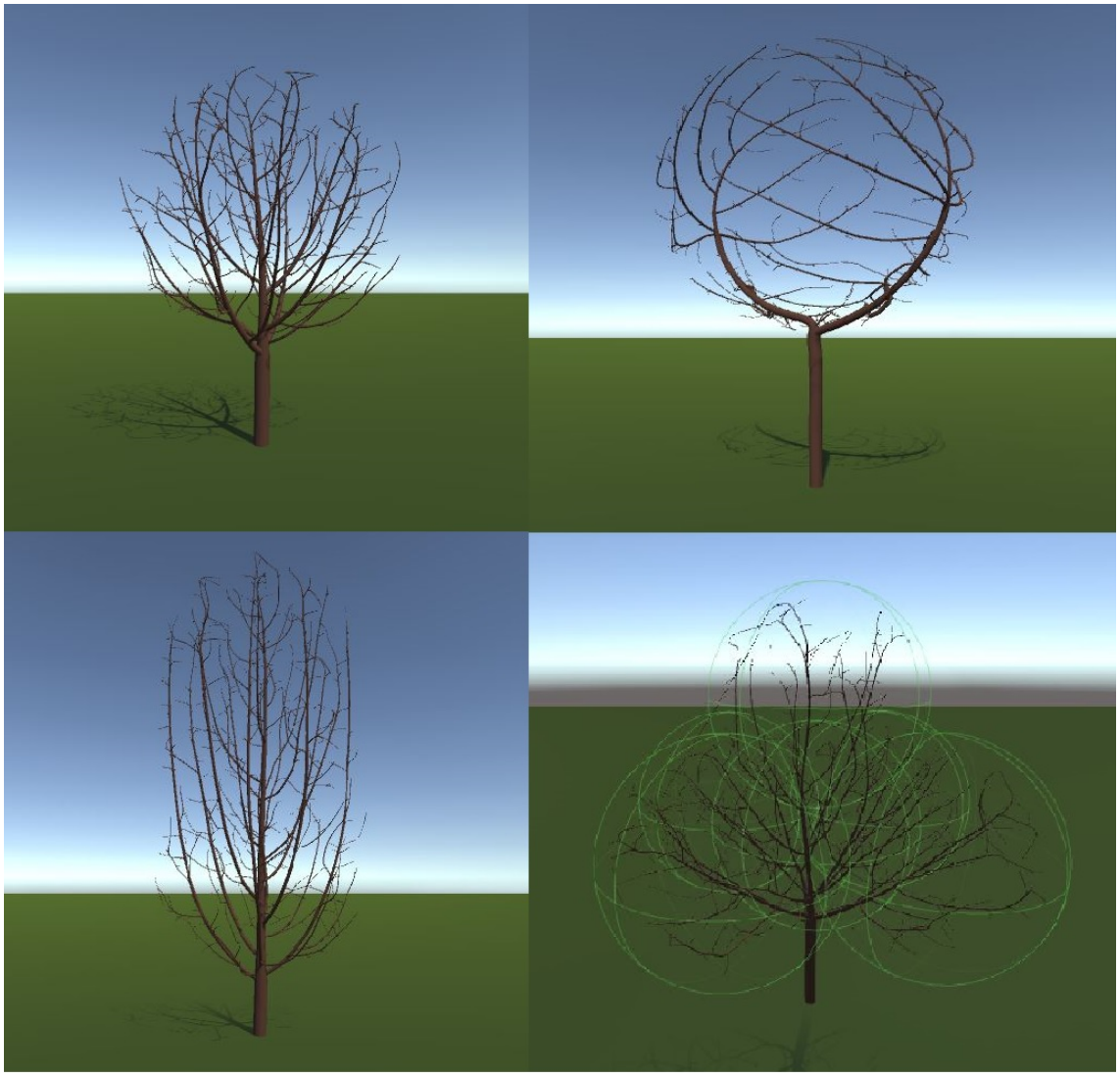


Figura 2.1: Diferente modelos generados mediante el algoritmo de “Space Colonization”

Para generar un árbol mediante el algoritmo de “Space Colonization” es necesario definir las dos estructuras principales de este. Por un lado tenemos los puntos distribuidos en el espacio los cuales llamaremos nube de puntos. Esta estructura consiste en un conjunto de puntos situados en posiciones aleatorias dentro de un volumen. Por otro lado tenemos los segmentos que conforman el árbol, estos segmentos se irán generando conforme el algoritmo se va ejecutando, haciendo crecer el árbol en cada iteración en la dirección de los puntos de atracción.

2.1. Nube de puntos

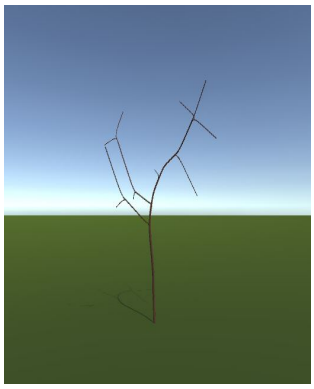
Se trata de una lista de puntos generados en posiciones aleatorias, los cuales definirán la dirección en la que va a crecer el árbol. Esta estructura será la encargada de definir la forma general del árbol, ya que las ramas de este intentaran acercarse a estos puntos, y dejan de crecer una vez alcanzados todos los puntos de la nube.

Esta nube esta definida por una figura tridimensional, y los puntos que la conforman se generan dentro o alrededor de esta figura. Esta figura es el parámetro de entrada a partir del cual se genera cada árbol. De todos modos los puntos se generan dentro de la figura de manera aleatoria, por lo que es posible obtener diferentes arboles a partir de una misma figura.

2.1.1. Parámetros

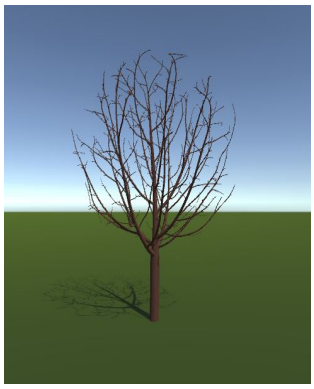
A la hora de definir cada nube de puntos es necesario definir una serie de parámetros, los cuales permiten influir en la posición de los puntos que la conforman, alterando de este modo la forma de el árbol generado a partir de esta.

- **Numero de puntos**: El numero de puntos que conforman la nube tiene influencia en las ramas del modelo resultante, ya que al aumentar el numero de puntos cada rama es atraída hacia un mayor numero de direcciones, obteniendo como resultado árboles mas ramificados.



(a) 10 puntos de atracción

(b) 100 puntos de atracción



(c) 1000 puntos de atracción

Figura 2.2: Visualización del efecto que tiene el numero de puntos de la nube en el modelo resultante

- **Figura**: Cada nube de puntos se genera a partir de una figura tridimensional, de modo que los puntos que la conforman se encuentran dentro o alrededor de ella. Modificar esta figura tiene un impacto directo en la forma del árbol resultante, ya que esta se asemejará a la figura.

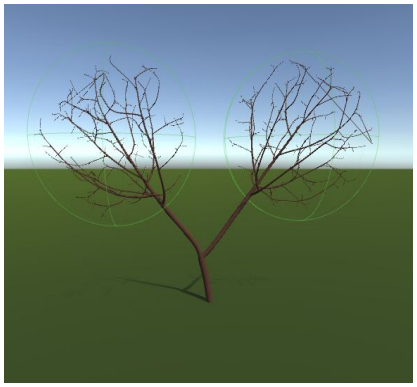
(a) Árbol generado a partir de una esfera

(b) Árbol generado a partir de un cubo

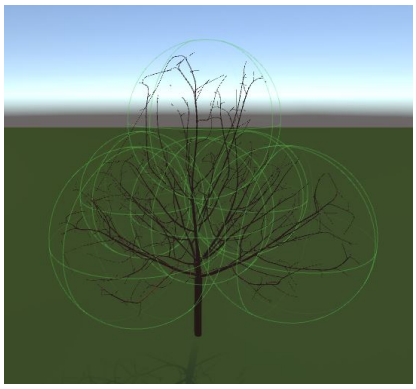
(c) Árbol generado a partir de una cápsula

Figura 2.3: Comparación de modelos de árboles generados a partir de diferente figuras

- **Numero de figuras**: Es posible emplear mas de una figura para conformar la nube de puntos, bien para obtener una figura mas compleja combinando figuras simples, o bien para hacer que el árbol se divida en diferentes zonas.



(a) Árbol generado a partir de dos esferas separadas en el espacio



(b) Geometría generada combinando varias esferas

Figura 2.4: Ejemplo de posibles nuevas geometrías generadas combinando formas simples a la hora de generar la nube de puntos

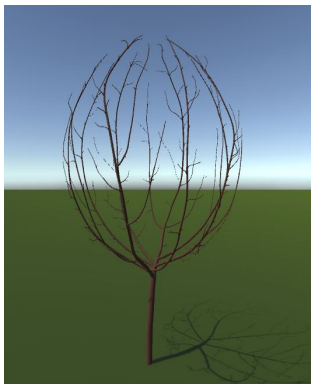
- **Modo de generación de los puntos**: Los puntos de la nube son generados en posiciones aleatorias a partir de la figura, pero es posible generarlos siguiendo diferentes métodos:

1. Llenar la figura: Consiste en generar los puntos dentro de la figura, consiguiendo de este modo que el árbol resultante rellene completamente la figura.

2. Rodear la figura: consiste en generar los puntos alrededor de la figura. Para ello se define un variable que indica la distancia máxima a la que pueden estar los puntos de la superficie de la figura.

3. Utilizar los vértices de la figura: Se trata a de una variación del segundo método que tiene como objetivo generar árboles que se ajusten mejor a las figuras complejas, para ello se generan todos los puntos de la nube a partir de los vértices que conforman la malla.

(a) Método de llenar la figura



(b) Método de rodear la figura

(c) Método de utilizar los vértices

Figura 2.5: Ejemplo de como afecta el modo en que afecta el método utilizado para generar los puntos mediante la figura, en el modelo resultante

2.1.2. Implementación

Para generar la nube de puntos, es necesario un método el cual permita generar puntos en posiciones aleatorias dentro de la figura inicial:

1. Se genera un punto en una posición aleatoria dentro del cuadro delimitador de la figura.
2. Se comprueba si ese punto se encuentra dentro de la figura, utilizando un método que consiste en trazar un segmento desde el punto generado hasta una posición situada fuera de la figura, si el segmento trazado atraviesa la figura un numero par de veces quiere decir que el punto esta fuera de la figura por lo que el punto queda descartado, en caso de que atraviese la figura un numero impar de veces significa que el punto esta dentro de la figura así que es añadido a la nube de puntos.
3. Se repite el paso anterior hasta que el numero de puntos dentro de la nube de puntos sea el requerido.

2.2. Estructura del árbol

Consiste en un estructura en forma de árbol que define el modelo del árbol. Se trata de una estructura de datos en forma de árbol, esta compuesta por varios nodos los cuales

representan segmentos del modelo. Los nodos están definidos por una posición en el espacio, un nodo padre y una lista de nodos hijos. Cada uno de estos nodos representara un segmento del tronco del árbol generado.

La posición de cada nodo, definida por un vector de tres dimensiones, representa la posición de inicio de el segmento. En el caso del nodo inicial, el cual se llamara raíz, esta posición será la del punto desde el que se quiera generar el árbol. En el resto de los nodos la posición se generara mediante el algoritmo de "Space Colonization".

El nodo padre es una referencia a otro nodo del árbol, siendo este nodo el anterior en el árbol, es decir el nodo desde el cual ha crecido el segmento actual. En al caso de la raíz, al ser generada al inicio del algoritmo y no creciendo desde otro segmento su valor será nulo.

Otro elemento utilizado a la hora de definir la estructura del árbol es la lista de hijos. Se trata de una lista que contiene referencias a todos los nodos generados a partir del nodo actual. Los nodos situados en las hojas de la estructura del árbol, es decir aquellos cuyo único nodo adyacente sea su nodo padre, tendrán una lista vacía.

Mediante estos parámetros se define la estructura del árbol, pero cada nodo consta también de otros parámetros utilizados a la hora de generar la estructura como la dirección de crecimiento del nodo. También se han definido una serie de variables cuyo objetivo es el de ayudar a generar el modelo 3D del árbol una vez la estructura se termina de generar.

La dirección de crecimiento del árbol, definida mediante un vector unitario de tres dimensiones, representa la dirección en la que ese nodo concreto va a generar un hijo en la iteración actual. En caso de que el nodo actual no vaya a crecer en la atracción actual el valor de la dirección de crecimiento sera un vector nulo.

En cuanto a los parámetros utilizados para generar el modelo 3D del árbol, se habla en mas profundidad a cerca de ellos en la sección dedicada a la generación del modelo. En la siguiente lista se nombran estos parámetros y se da una breve explicación acerca de su utilidad.

- Radio: Se trata del valor que toma el radio de la rama del árbol en el punto en el que se encuentra el nodo.
- Iteración de crecimiento: Indica la iteración en la que se ha generado esa rama en concreto.
- Hojas: se trata de una lista que guarda las hojas que brotan del nodo.

- Es parte del tronco: Variable que indica si el nodo representa un segmento del tronco del árbol, o de las ramas del árbol.
- Lista de mallas: Lista que guarda todas las mallas correspondientes a cada uno del los nodos del árbol.

2.2.1. Parámetros

- **Distancia de crecimiento**: La distancia que crece cada uno de los segmentos en la dirección definida por los puntos de atracción.
- **Distancia de influencia de los puntos**: Distancia máxima a la que un segmento puede ser atraído por alguno de los puntos.
- **Distancia de eliminación de puntos**: Distancia máxima a la que puede estar un segmento de un punto sin que este sea eliminado de la nube.
- **Vector de tropismo**: Los tropismos son factores físicos que afectan al crecimiento de las plantas, como la posición de la luz, la gravedad o la dirección del viento, estos factores influyen en la dirección de crecimiento de las plantas. Para emular este fenómeno se utiliza un vector de dirección, este vector se aplica a la dirección de crecimiento de las ramas, alterando de este modo la forma final del árbol.
- **Factor de crecimiento de las ramas**: Al calcular el valor del radio de cada rama, se utiliza el valor del radio de las ramas que crecen a partir de esta. El radio r de un segmento del que parten dos segmentos de radio r_1 y r_2, se calcula mediante la siguiente formula $r^n = {r_1}^n + {r_2}^n$, siendo n el factor de crecimiento de las ramas.
- **Radio de las terminaciones de las ramas**: Se trata del valor que tiene el radio en las ramas exteriores del árbol, es decir las ramas a partir de las cuales no crece ningún otro segmento.

2.2.2. Algoritmo

El algoritmo de "Space Colonization"parte de un punto en la superficie y genera la estructura del árbol haciendo crecer los segmentos de este en dirección a los puntos de la nube de puntos. A la hora de generar el árbol se diferencian dos fases, durante la primera se generara el tronco del árbol, el cual consiste en una serie de segmentos los cuales no

generan mas que un único hijo. Durante la segunda fase se generan las ramas del árbol, las cuales están formadas por segmentos que pueden tener una, varias o incluso ningún hijo, dependiendo de la posición en la que se encuentren.

El proceso de crecimiento definido por el algoritmo de "Space Colonization" es un proceso iterativo, en el cual se hace crecer el árbol teniendo en cuenta las condiciones de la iteración actual, siendo estas condiciones la posición de los nodos del árbol y la posición del los puntos de la nube de puntos.

En esta implementación se han definido dos tipos de iteración, la primera encargada de generar el tronco del árbol, y la segunda para generar las ramas del árbol.

El proceso de generar el tronco es el siguiente:

1. Generar la raíz del árbol, la cual esta definida como un nodo situado en la superficie, a partir de la cual se quiere generar el árbol, y cuyo padre es una referencia nula.

2. Calcular la dirección de crecimiento del ultimo nodo añadido al árbol situado en la posición v. Esta dirección se define como la suma de todos los vectores de dirección posibles entre el ultimo nodo añadido y cada uno de los puntos del conjunto definido en la nube de puntos el cual denotaremos como S. De este modo podemos expresar el vector de dirección del siguiente modo:

$$d = \sum_{s \in S} \frac{s-v}{||s-v||}$$

3. Normalizar el vector d y calcular la posición v' para el nuevo nodo la cual se calcula mediante la siguiente formula: $v' = v + D\hat{d}$, donde D es la distancia de crecimiento definida para cada iteración.

   En el caso de haber utilizado un vector de tropismo $\vec{g}$ para modificar la trayectoria del árbol, la formula para calcular la nueva posición se modifica:

$$v' = v + D\frac{\hat{d}+\vec{g}}{||\hat{d}+\vec{g}||}$$

4. Generar un nuevo nodo en la posición v' y se le añade una referencia a el nodo a partir del cual ha crecido, el cual sera su nodo padre.

5. Añadir el nuevo nodo a la lista de hijos del nodo situado en la posición v.

6. Calcular la distancia desde el nuevo nodo hasta el nodo mas cercano dentro de la nube de puntos. En caso de que esta sea inferior a la distancia de influencia de los

puntos, finalizar el proceso de generar el tronco. En caso de que la distancia sea mayor o igual a la distancia de influencia de los puntos repetir los pasos del 2 al 6.

Algoritmo 1: Algoritmo de generación de ramas

```
1  Función Generar_Nuevo_Nodo(nodo):
2      if nodo.dir ≠ vector_nulo then
3          nuevo_nodo = Crear_Nodo_En_Posicin(nodo.pos + D (nodô.dir + g⃗) / ||nodô.dir + g⃗||);
4          nodo.dir = vector_nulo;
5          nuevo_nodo.padre = nodo;
6          nodo.hijos.Insertar(nuevo_nodo);
7          return nuevo_nodo;
8      else
9          return null;
10     end
11 end

12 nodo_actual = raz;
13 distancia_mnima = ∞;
14 acabar = false;
15 while ¬acabar do
16     for s ∈ S do
17         dist = Distancia_Euclidiana(nodo_actual.pos, s) nodo_actual.dir =
             nodo_actual.dir + (s − nodo_actual.pos) / ||s − nodo_actual.pos||
18         if dist < distancia_mnima then
19             distancia_mnima = true;
20             if distancia_mnima < distancia_de_atraccin then
21                 acabar = true;
22             end
23         end
24     end
25     if ¬acabar then
26         nuevo_nodo = Generar_Nuevo_Nodo(nodo_actual);
27         nodo_actual = nuevo_nodo();
28     else
29 end
```

Mediante el proceso de generar el tronco se consigue que el ultimo segmento de este se encuentre a la distancia de influencia de por lo menos uno de los puntos. Esto permite iniciar el proceso de crecimiento de las ramas del árbol, ya que estas solo crecen hacia los puntos que se encuentran a una distancia menor a esta.

El proceso de crecimiento de las ramas del árbol se asemeja al del tronco, con la diferencia de que por cada iteración se harán crecer todas las ramas influenciadas por los puntos de atracción.

1. Crear una lista de nodos que denotaremos N, y añadir el ultimo nodo generado en el tronco.

2. Para cada punto dentro de S (conjunto de puntos que conforman la nube de puntos), buscar el nodo dentro de N mas cercano y modificar su dirección de crecimiento mediante la siguiente formula:

$$d = d + \frac{s-v}{s-v}$$

   siendo v la posición del nodo mas cercano, d su dirección de crecimiento, y s un punto dentro de S.

3. Hacer crecer los nodos dentro de N, para ello se utilizara el mismo método descrito en los pasos del 3 al 5, con la diferencia de que en caso de que el vector de dirección sea nulo, el nodo no crecerá. Todo nuevo nodo generado durante la iteración es añadido a N.

4. Eliminar todos los puntos que se encuentren a una distancia menor a la distancia de eliminación de puntos, de alguna de los nodos en N.

5. Si queda algún nodo dentro de N repetir los pasos del 2 al 4, en caso contrario terminar la ejecución.

Algoritmo 2: Algoritmo de generación de ramas

```
1  Función Generar_Nuevo_Nodo(nodo):
2      if nodo.dir ≠ vector_nulo then
3          nuevo_nodo = Crear_Nodo_En_Posicin(nodo.pos + D * (nodô.dir + g⃗) / ||nodô.dir + g⃗||);
4          nodo.dir = vector_nulo;
5          nuevo_nodo.padre = nodo;
6          nodo.hijos.Insertar(nuevo_nodo);
7          return nuevo_nodo;
8      else
9          return null;
10     end
11 end

12 acabar = false;
13 While(¬acabar)
14     for s ∈ S do
15         distancia_mnima = ∞;
16         nodo_a_distancia_mnima;
17         for n ∈ N do
18             dist = Distancia_Euclidiana(s.pos, n.pos);
19             if dist < distancia_mnima then
20                 distancia_mnima = dist;
21                 nodo_a_distancia_mnima = n;
22             end
23         end
24         if distancia_mnima < distancia_de_atraccin then
25             n.dir = n.dir + (s − n.pos) / ||s − n.pos||;
26         end
27     end
```

```
27
28      for n ∈ N do
29          nuevo_nodo = Generar_Nuevo_Nodo(n);
30          if nuevo_nodo = null then
31              N.Eliminar(n);
32          else
33              N.Insertar(nuevo_nodo);
34          end
35      end
36      if N = ∅ then
37          acabar = true;
38      end
39      for s ∈ S do
40          for n ∈ N do
41              dist = Distancia_Euclidiana(s.pos, n.pos);
42              if dist < distancia_de_eliminacin then
43                  S.Eliminar(s);
44                  break;
45              end
46          end
47      end
48  end
```

Mediante este proceso se obtiene una estructura en forma de árbol, la cual contendrá los segmentos del árbol, a partir de la cual se va a generar el modelo 3D del árbol. Antes de empezar a generar el modelo es necesario aplicarle algunas modificaciones a los nodos de esta estructura. El objetivo de estas modificaciones es generar un modelo más natural.

La primera modificación consta en alterar la posición de los nodos del árbol, desplazando cada uno de los nodos medio camino en dirección a su nodo padre, de modo que su nueva posición sea, $v' = v + \frac{v_p - v}{2}$, siendo v la posición del nodo y v_p la de su nodo padre. Mediante este proceso se consigue reducir el ángulo de ramificación.

La siguiente modificación consiste en calcular el radio de los segmentos del árbol al pasar por cada uno de los nodos. El valor de un segmento depende de los segmentos que surgen de él, de modo que el radio de un segmento del que brotan dos segmentos con radio r_1 y r_2 se calcula del siguiente modo, $r^n = {r_1}^n + {r_2}^n$, siendo n un parámetro cuyo valor suele definirse entre 2 y 3.

3. CAPÍTULO

Generación del Modelo

Para generar un modelo 3D es necesario definir todas las partes que conforman a este. Para ello de ha definido un nuevo objeto llamado BranchMesh, el cual contiene la información necesaria para crear el modelo.

3.1. Malla de triángulos

Se trata de un conjunto de vértices y triángulos que conforman una superficie 3D. Esto se debe a que muchos paquetes de software gráficos y dispositivos hardware están optimizados para trabajar con mallas de triángulos.

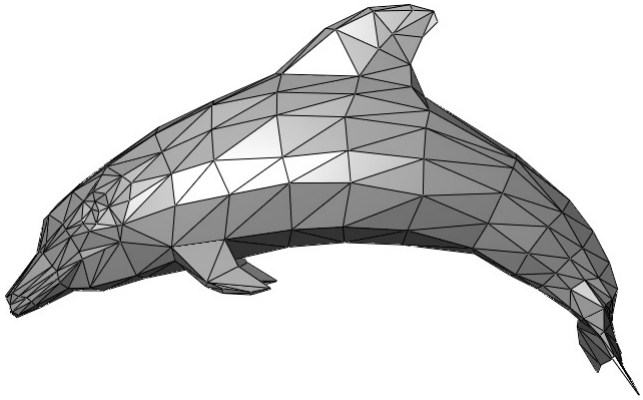


Figura 3.1: Modelo de un delfín representado mediante una malla de triángulos (Fuente: Wikimedia Commons)

Para generar la malla del modelo del árbol se ha utilizado el método descrito en el articulo "Modeling the Mighty Maple"[Bloomenthal, 1985], este método describe como generar el modelo 3D de un árbol, a partir de puntos tridimensionales y una serie de conexiones entre ellos. El tronco y las ramas son modeladas a partir de una serie de cilindros.

Para cada uno de los nodos se genera un numero de cilindros igual al numero de hijos, de modo que cada vez que un nuevo nodo es generado a partir de otro, también se le añade un malla. Estos cilindro son posicionados y rotados de modo que cubran la estructura del árbol.

Estos cilindros son generados a partir de dos anillos de puntos, situados cada uno en uno de los extremos del cilindro. Cada uno de estos anillos tiene $n+1$ puntos, ya que al tratarse de una malla unida por una de sus aristas es necesario duplicar los vértices de esta para el mapeado de textura.

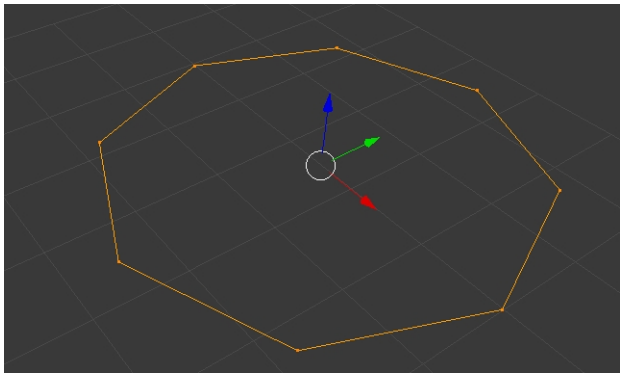


Figura 3.2: Anillo formado por 8 puntos

Considerando como v la posición de un nodo, v_p la de su nodo padre y v_h la del hijo para el que queremos generar el cilindro. El centro del primer anillo de puntos esta centrado en el punto v y su rotación es la del vector $v - v_p$, y el segundo anillo esta centrado en la posición v_h y su dirección es la del vector $v_h - v$.

El radio de cada uno de los anillos es el del nodo en el cual están situados, de modo que en el caso del primer anillo será el radio calculado para el nodo desde el que se genera, y en el caso del segundo anillo sera el del nodo hijo para el cual se está generando el cilindro.

Para generar las caras del cilindro basta con conectar las aristas del primer anillo con las del segundo mediante quads.

Finalmente se añadirá una cara que cubra el segundo anillo, ya que en las terminaciones de las ramas estos no tienen ningún cilindro adyacente, lo que deja el interior del modelo

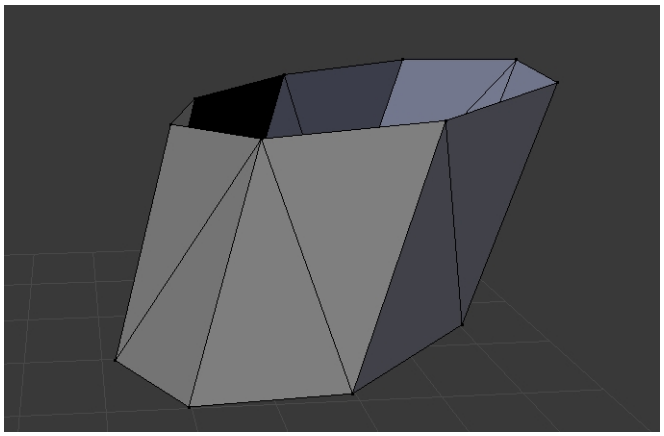


Figura 3.3: Cilindro generado a partir de dos anillos

al descubierto. Teniendo en cuenta que esta cara esta formada por un polígono regular de n lados, es posible cubrirlo añadiendo caras triangulares a la malla.

Repitiendo este proceso con cada una de las uniones entre nodos de la estructura generada, obtenemos una malla de triángulos con la apariencia de un árbol.

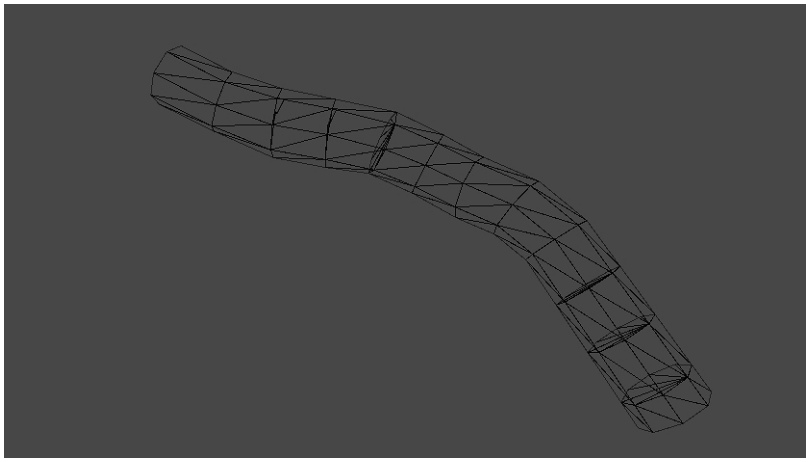


Figura 3.4: Trozo de un malla generada mediante el método descrito

3.2. Mapeado UV

En un modelo 3D el mapeado UV consiste en proyectar un imagen bidimensional en la superficie de una figura tridimensional. De este modo es posible pintar una superficie con los colores de un imagen. Para ello se asigna una coordenada de la imagen a cada uno de

los vértices del modelo, y se interpola el valor del color para cada uno de los puntos del modelo.

En el caso de un cilindro resulta sencillo definir el mapeado UV, ya que si desenvolvemos la figura separando una de sus aristas, obtenemos una superficie plana. De este modo basta con proyectar la textura sobre esta superficie para generar el mapeado UV.

La textura utilizada es una imagen de la corteza de un árbol, modificada de modo que no se note el salto de un segmento a otro, ya que al utilizar para la misma textura para todos los segmentos, la parte inferior de esta debe coincidir con la parte superior.

Figura 3.5: Imagen del tronco proyectada sobre el cilindro desenvuelto

3.3. Vectores normales

En computación gráfica, un normal es un vector asociado a cada uno de los vértices el cual sustituye al verdadero normal geométrico. Esto permite calcular la iluminación por medio de modelos como "Phong Shading".

Mediante esta técnica se consigue suavizar la superficie del cono dando la apariencia de que es una superficie continua en vez de tener n caras. Para ello es necesario asignar a cada vértice de los anillos el vector que va desde el centro del anillo al vértice.

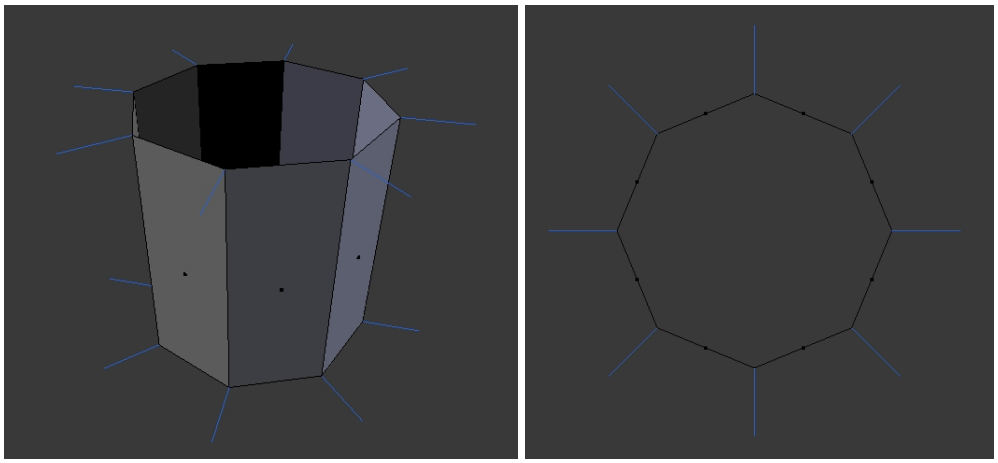


Figura 3.6: Visualización de los vectores normales

3.4. Mallas en Unity

Para generar la malla del modelo en Unity, se ha utilizado el objeto Mesh [Unity Technologies, 2017a] que proporciona la aplicación. Este objeto proporciona las herramientas necesarias para crear un malla procedural, y luego renderizarla mediante el motor gráfico de Unity.

Esta estructura cuenta con cuatro array-s, cada uno con el objetivo de guardar una una de las estructuras que componen la malla.

- vertices: Array de vectores de tres dimensiones que guarda los vértices del modelo.
- triangles: Array de integers, cada grupo de tres valores consecutivos, indican la posición en el vector anterior de los vértices que forman una cara.
- uv: Array de vectores de dos dimensiones, indica las coordenadas de mapeado uv para cada vértice.
- normals: Array de vectores de tres dimensiones, indica el vector normal que le corresponde a cada uno de los vértices.

4. CAPÍTULO

Introducción de Hojas en el Modelo

Si observamos los árboles en la naturaleza, es fácil advertir que una de las sus principales características visuales es el follaje. Este varia tanto en forma como en color de entre las diferentes especies de árboles.

Con el fin de obtener un aspecto mas realista en el modelo, es necesario añadir hojas al modelo. Utilizando hojas de diferentes formas y tamaños se puede conseguir imitar diferentes especies de árboles. Otro factor importante es el color del follaje, ya que alterando este también se obtienen modelos más variados, y también permite simular las diferentes fases de un árbol, como seria por ejemplo el tono marrón que adquieren las hojas en el otoño.

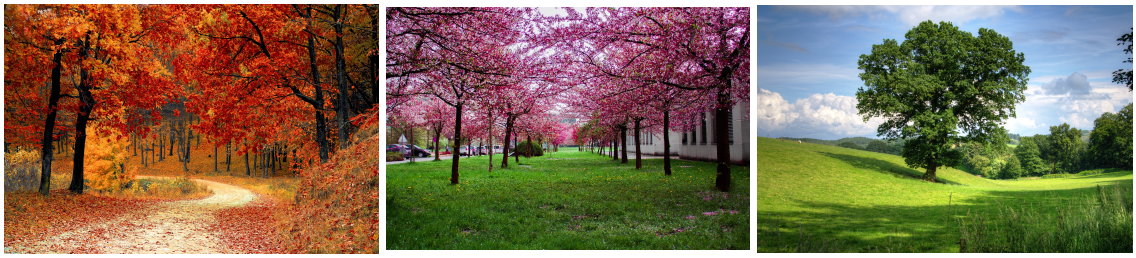


Figura 4.1: Diferentes follajes que se pueden encontrar en la naturaleza

4.1. Generación de hojas

A la hora de generar las hojas del modelo, el primer paso es definir donde y con que frecuencia se deben generar estas. Para llevar a cabo este proceso se parte de una estructura de árbol ya generada, de este modo se puede saber cuales son los segmentos de rama situados en los bordes, estos serán los que generen las hojas.

Para obtener resultados mas naturales la cantidad de hojas generada en cada uno de los segmentos se decidirá mediante un número aleatorio, el cual esta situado dentro de una rango definido por el usuario. De este modo, cada uno de los segmentos producirá un numero diferente de hojas.

Para crear el modelo de las hojas se utilizan quads de doble cara. Cada uno de los quads está unido al segmento que lo genera por el centro de una de sus aristas exteriores, y esta rotado con respecto a ese punto, de modo que cada una de las hojas que brote de uno de los segmentos tenga una orientación diferente, de este modo se consigue que no se amontonen todas las hojas de un segmento en la misma posición.

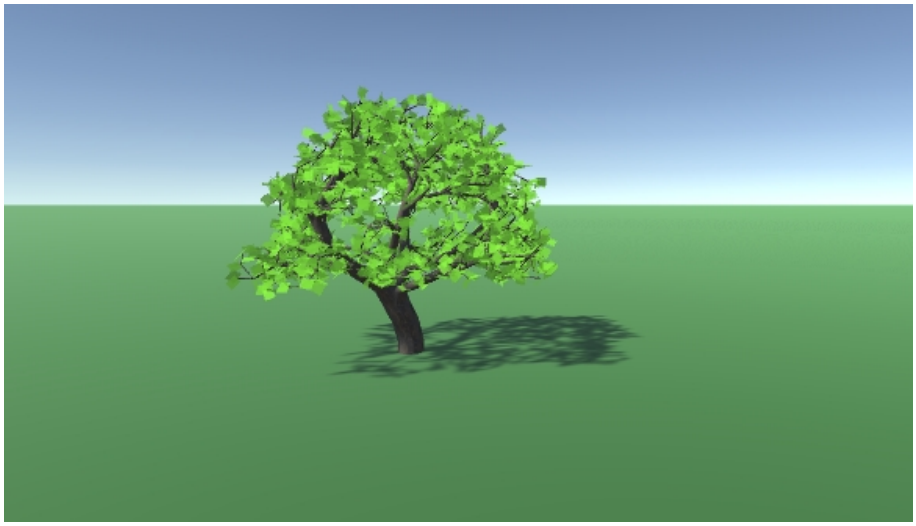


Figura 4.2: Modelo de un árbol con hojas formasdas por quads

4.2. Shader

Utilizar quads como hojas da buenos resultados a distancia, ya que al tener muchas hojas de pequeño tamaño en un árbol, no se distingue con facilidad que en realidad sean cuadrados, pero al acercarse al árbol se empieza a percibir que no son hojas de verdad.

Para solucionar esto se pueden sustituir los quads por otra figura con apariencia de hoja, pero al haber tantas hojas este proceso añadiría muchos vértices al modelo. Es por ello que se ha optado por utilizar un sahder para modificar la apariencia de los quads.

Un shader es un programa que se ejecuta en la GPU, para realizar los cálculos gráficos necesarios para renderizar el modelo, como transformar los vértices a la perspectiva de la cámara o calculara el color de cada uno de los píxeles.

Para modificar la apariencia de las hojas mediante un shader se ha utilizado una máscara: una textura con una imagen binarizada de una hoja. Una imagen binarizada se trata de un imagen en la que todos los píxeles son o bien completamente blancos o bien completamente negros. Por lo tanto esta máscara muestra la forma de una hoja.

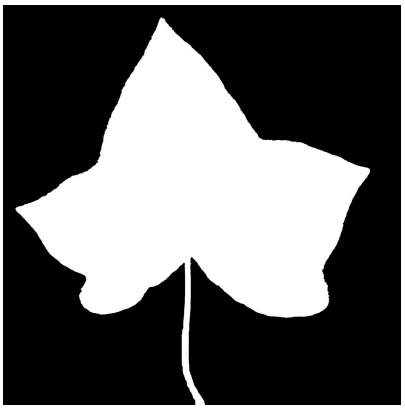


Figura 4.3: Máscara utilizada para dar forma a las hojas

Como hemos visto en la sección de generar el modelo, al asignar una textura a un modelo esta se mapea a lo largo de este según las coordenadas UV definidas, y se le asigna a cada uno de los píxeles su valor correspondiente en la textura. En este caso no se quiere mapear directamente el color del objeto a través del valor de la textura, el objetivo de este shader es ocultar las zonas en las que la textura tiene color blanco, y mostrar únicamente aquellas en las que es de color negro. Para ello Unity brinda la opción de escribir tus propios shaders.

Para obtener el efecto que se ha descrito basta con utilizar la textura para calcular el valor

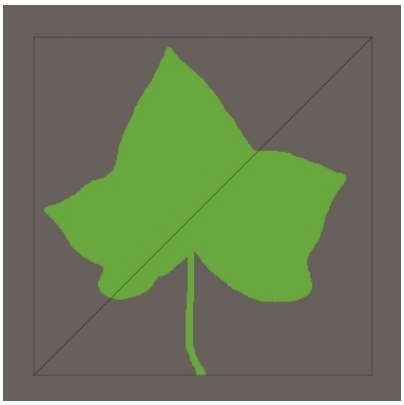


Figura 4.4: Apariencia de un quad tras aplicarle el shader de transparencia

de alpha en el color de cada píxel. En el espacio de colores RGBA se utilizan cuatro canales para describir cada uno de los colores R (red) para el rojo, G (green) para el verde, B (blue) para el azul y A (alpha) para la opacidad. De modo que un color con valor 0 en el canal alpha sera completamente transparente, y un color con valor de 1 sera completamente opaco. De este modo conseguimos que las partes en las que la textura es blanca sean completamente transparentes, dando la impresión de que el quad tiene forma de hoja.

Para calcular el color del resto de la hoja se utilizan dos colores, los cuales decide el usuario a través de la interfaz que proporciona Unity para modificar las variables de los shaders. El color final es calculado interpolando un valor aleatorio entre los dos colores proporcionados, obteniendo que cada una de las hojas tenga un tono ligeramente distinto al resto, mediante este sistema conseguimos un apariencia mas acorde a la realidad, ya que en la mayoría de los casos las hojas de un árbol muestran pequeñas variaciones en su tonalidad. Para evitar este efecto basta con seleccionar el mismo color en ambos campos de la interfaz del shader.

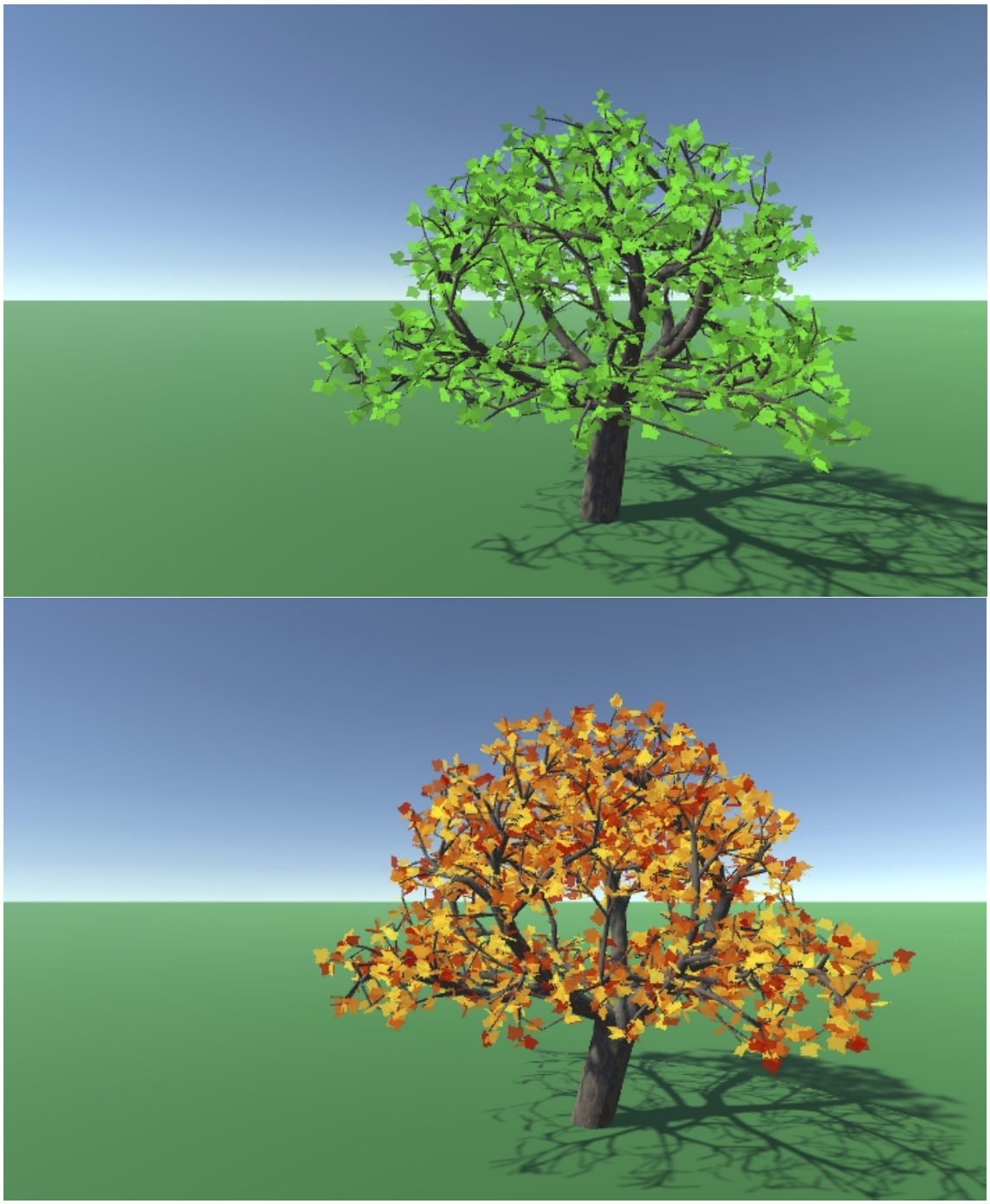


Figura 4.5: Modelos obtenidos al utilizar el shader de las hojas

5. CAPÍTULO

Creación de Bosques a partir de los árboles procedurales

Mediante el sistema descrito hasta ahora, es posible generar múltiples modelos de árboles de modo automático, ajustando una serie de parámetros. Los modelos generados a partir de los mismos parámetros de entrada, tienen una apariencia general similar, pero cada uno de ellos tiene rasgos que los convierte en únicos. Esto es similar a lo que ocurre con los árboles que conforman un bosque, los cuales, al ser de la misma especie, se parecen a los de su alrededor pero al observarlos con mas detalle, se puede ver claramente como varían.

Esta característica, hace que el algoritmo de generación procedural de árboles sea un buen punto de partida para un generador de bosques. Para ello bastará con aplicar el algoritmo repetidas veces, generando así múltiples modelos en lugar de uno.

El principal problema a resolver para poder generar un bosque es el de posicionar los arboles, ya que el sistema que se ha descrito genera un único árbol en el origen, pero a la hora de generar un bosque es necesario que los arboles estén distribuidos a lo largo de una superficie. Para resolver el problema definiremos esta superficie como un plano centrado en el origen.

Para generar estas posiciones, hay que definir un sistema que tomando como parámetro de entrada el plano en el que se quiere generar el bosque, calcule una serie de puntos dentro del plano, en los que se generaran los modelos de los árboles mediante el algoritmo de "Space Colonization". Esto se puede hacer tomando el plano y dividiéndolo en cuadriculas, de este modo se puede generar un árbol en el centro de cada cuadricula, consiguiendo para cada uno de los árboles una posición diferente. El problema con este sistema es que

genera bosques de apariencia muy artificial, ya que todos los árboles estarán alineados y la separación entre ellos sera siempre la misma. De todos modos esta apariencia puede resultar útil para generar bosques artificiales, que son aquellos establecidos por el hombre a partir de procedimientos de plantación.

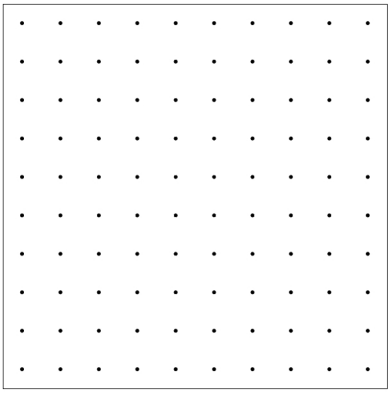


Figura 5.1: Ejemplo de posicionamiento de objetos mediante muestreo regular

Para alterar esta apariencia artificial y generar bosques como los que se encuentran en la naturaleza es necesario introducir algún tipo de ruido en el proceso de generar las posiciones. Este ruido se introduce utilizando números aleatorios para calcular la posición de cada uno de los árboles.

5.1. Blue noise

Un método sencillo para generar posiciones aleatorias dentro de una superficie es utilizar un generador de números aleatorios para calcular cada una de las coordenadas (x e y en este caso) de esta posición, este método es conocido como White noise. El principal problema de este método, a la hora de aplicarlo en este sistema, es que no hay ningún tipo de control sobre las posiciones generadas, pudiendo hacer que un modelos se genere encima de otro. Para evitar esta situacion, se utiliza otro tipo de ruido llamado Blue noise.

El método de Blue noise se encuentra a medio camino entre el sistema de decidir el plano en cuadriculas, y el de White noise. Para generar los puntos mediante este metodo hay que dar los siguientes pasos:

1. Dividir el plano en celdas de igual tamaño, de un modo similar al utilizado al dividir el plano en cuadriculas pero dejando cierto espacio entre las celdas contiguas. Este

espacio debe ser igual o mayor al espacio mínimo entre dos de los árboles, es decir la separación mínima que debe haber entre dos árboles para que los modelos no colisionen. Este espacio se calcula a partir de la figura o conjunto de figuras utilizadas para generar la nube de puntos.

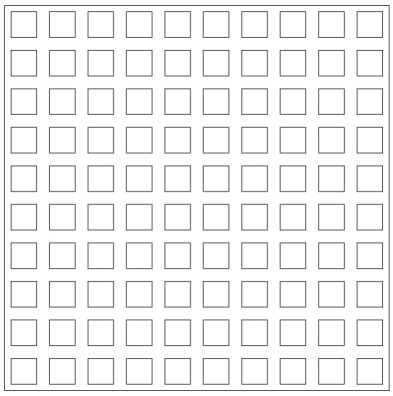


Figura 5.2: Celdas generadas para aplicar el método de blue noise

2. Para cada una de las celdas generadas se calcula un punto en su interior utilizando White noise, y se genera un árbol en esa posición. Debido a la separación entre las celdas no es posible que se generen dos arboles lo suficientemente cerca como para que los modelos colisionen.

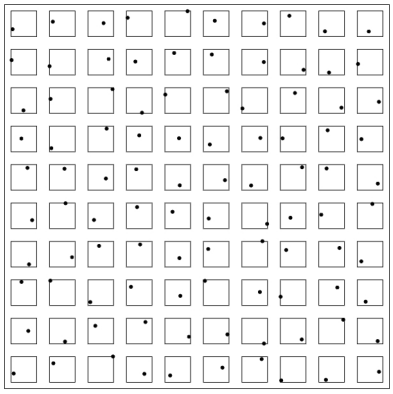


Figura 5.3: Cuadricula con puntos generados mediante white noise

Utilizando este sistema, conseguimos que la distribución de los árboles generados tenga una apariencia natural, y evitamos que se amontonen en una zona del plano.

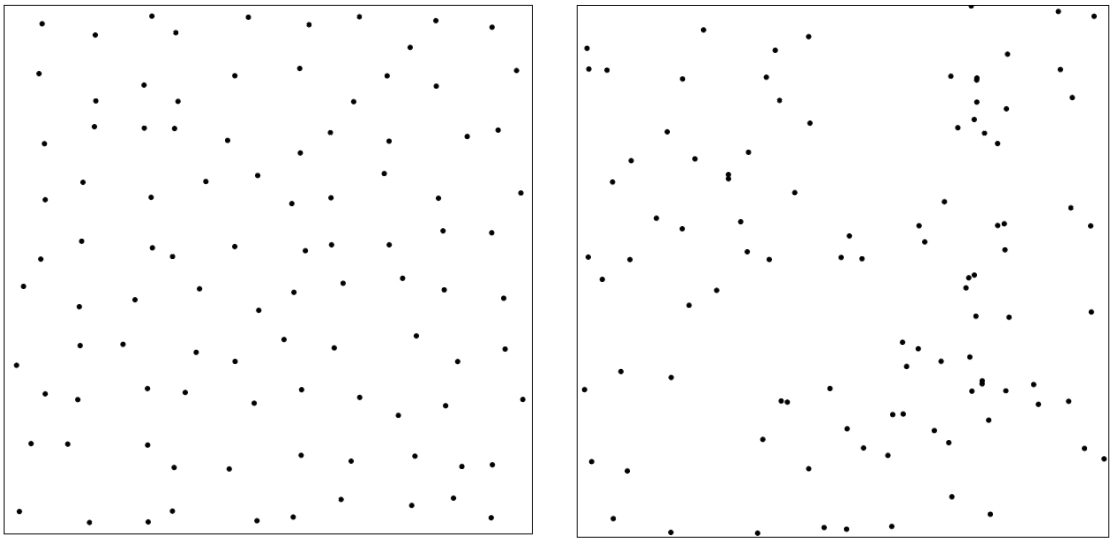


Figura 5.4: Comparación de una distribución generada mediante blue noise (izquierda) y una generada mediante white noise (derecha)

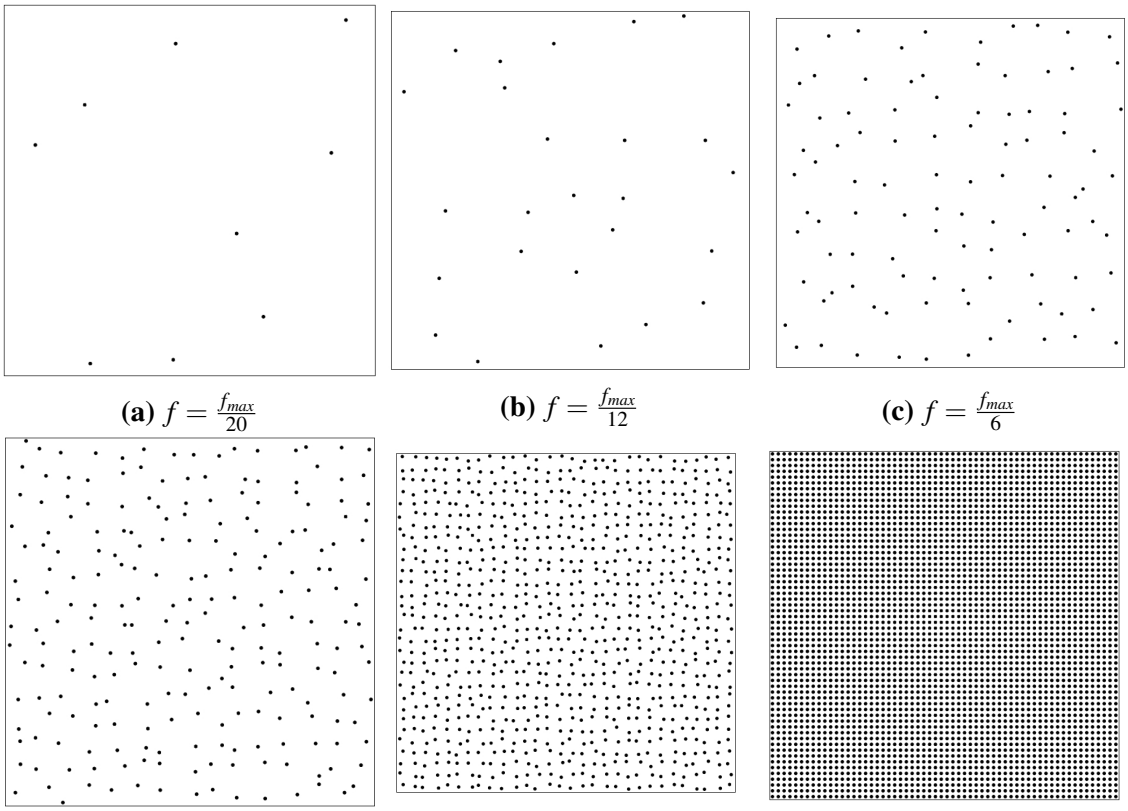
Ajustando la frecuencia del ruido, es posible obtener bosques con mayor o menor cantidad de árboles. La frecuencia define el numero de celdas que se crean dentro de la superficie, por lo que aumentando su valor obtenemos bosques con un mayor numero de arboles, y disminuyendo su valor también lo hace la cantidad de arboles en el bosque.

Debido a la distancia de separación mínima entre cada una de las celdas y las dimensiones de la superficie, el valor de la frecuencia esta limitado, ya que a partir de una frecuencia no es posible generar el numero de celdas necesario manteniendo entre ellas la distancia mínima de separación.

Tomando una superficie cuadrada de dimensiones *DxD*, y considerando d como la distancia mínima que debe mantenerse entre dos celdas, podemos calcular la frecuencia máxima f_{max} mediante esta expresión:

$$f_{max} = \frac{D}{d}$$

Cuando se utiliza el valor máximo de la frecuencia se obtiene el mismo resultado que mediante el método de muestreo regular, por lo tanto es recomendable utilizar siempre valores inferiores a este. Tras realizar una serie de pruebas, se ha podido observar como la mejor opción para conseguir una distribución de apariencia natural, es utilizar frecuencias inferiores a $\frac{f_{max}}{2}$, ya que el uso de frecuencias superiores genera resultados demasiado similares a los obtenidos mediante el muestreo regular.



(a) $f = \frac{f_{max}}{20}$ **(b)** $f = \frac{f_{max}}{12}$ **(c)** $f = \frac{f_{max}}{6}$

(d) $f = \frac{f_{max}}{4}$ **(e)** $f = \frac{f_{max}}{2}$ **(f)** $f = f_{max}$

Figura 5.5: Comparación de distribuciones generadas con diferentes frecuencias

5.2. Resultados

Combinado el método de generar posiciones en una superficie mediante blue noise, con el sistema para generar modelos de árboles descrito anteriormente, se pueden obtener escenas de bosques procedurales.

Para ello se utiliza un plano sobre el que situar el bosque, y se calcula un numero de puntos dentro de ese plano. Para calcular los puntos se define un valor para la frecuencia, y se aplica el método descrito para generar una distribución mediante blue noise.

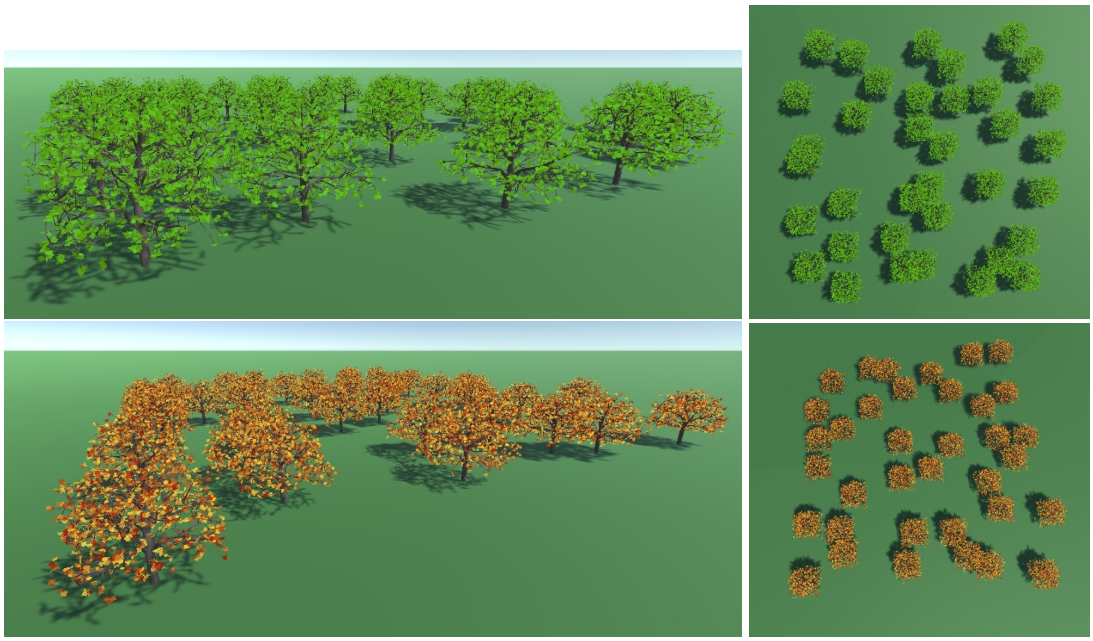
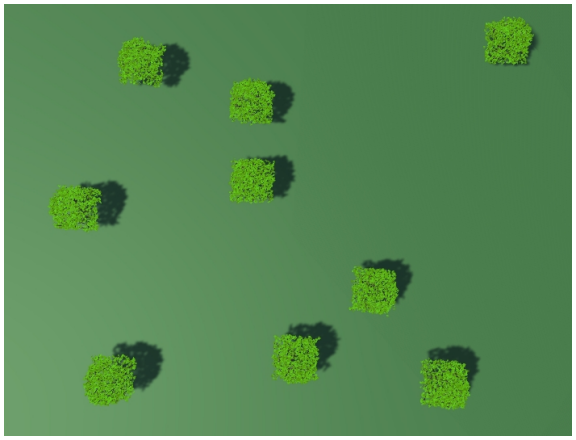


Figura 5.6: Bosque generado mediante el método descrito

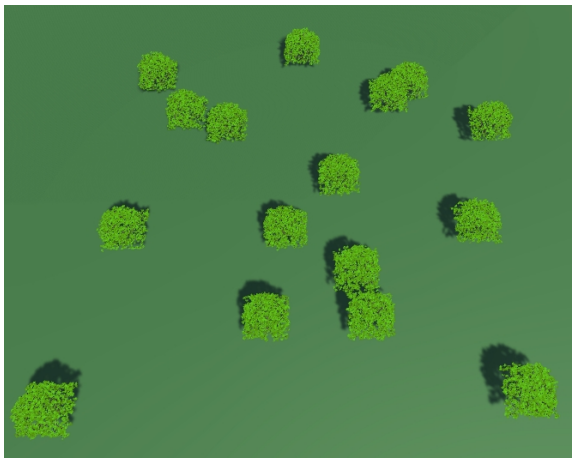
Para generar estos bosques se ha utilizado se ha utilizado una frecuencia de 6, y teniendo en cuenta que el numero de arboles que se genera es igual al cuadrado de la frecuencia, obtenemos que en cada escena se generan 36 árboles.

Para esta cantidad de árboles la aplicación tarda alrededor de 50 segundos en generar la escena. Este tiempo resulta excesivo para generar el bosque en tiempo de ejecución, además de esto el proceso de generar los arboles ralentiza demasiado el resto de los procesos. Por estos motivos el uso de la aplicación estará orientado a la creación de escenas, que luego puedan ser introducidas en un videojuego.

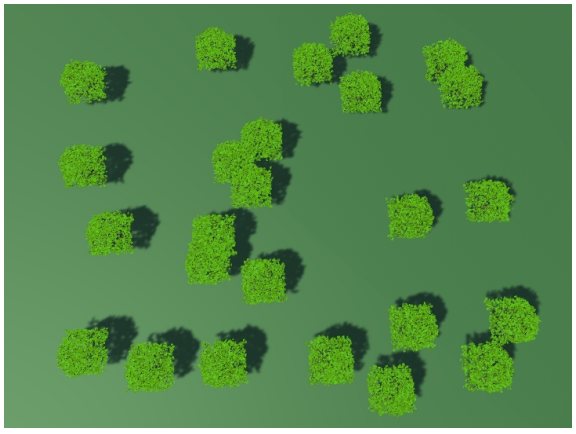
Generando bosques menos poblados se reduce el tiempo de ejecución, pero sigue estando el problema de ralentizar el resto de los procesos durante la generación de los árboles.



(a) $f = 3$



(b) $f = 4$



(c) $f = 5$

Figura 5.7: Distribución de un bosque generado con un valor menor de frecuencia, con el objetivo de agilizar el proceso

6. CAPÍTULO

Desarrollo de la aplicación en Unity

El desarrollo de la aplicación se ha llevado a cavo utilizando el motor de videojuegos Unity. Esto es debido a que una de las principales aplicaciones de la aplicación es el de utilizarla como herramienta a la hora de generar escenas en videojuegos, y Unity es uno de los motores de videojuegos mas populares, además de ser totalmente gratuito en su versión personal.

Otro factor ha sido el hecho de contar con cierta experiencia en el uso de esta herramienta, aunque a un nivel muy básico.

6.1. Aprendizaje

El lenguaje que se utiliza en Unity para escribir los scripts es C#, al tratarse de un lenguaje orientado a objetos con cierto parecido a otros como C++ o Java, no ha habido ningún problema a la hora de entender el funcionamiento de la mayoría de los elementos de este.

El proceso de aprendizaje se ha centrado en como utilizar el objeto Mesh de Unity, el cual permite generar una malla de vértices de manera procedural y modificarla utilizando scripts. Para ello hay un sección den el manual de Unity dedicada a la generación procedural de mallas “Procedural Mesh Geometry” [Unity Technologies, 2017b].

En el caso de los shaders Unity utiliza el lenguaje Cg, este lenguaje es muy similar a HLSL utilizado en otras aplicaciones. Este lenguaje muestra una sintaxis similar a la de C pero la manera en la que se estructura el código resulta muy diferente, y al no contar con

experiencia previa escribiendo shaders en Unity, para ello se han utilizado recursos como un guía introductoria a como se escriben los shaders en Unity "A gentle introduction to shaders in Unity3D" [Alan Zucconi, 2015], o la serie de videos "Writing Your First Shader In Unit" [Unity Technologies, 2017c] de la seccion de Unity

6.2. Desarrollo del código

Para desarrollar la aplicación se han desarrollado varios scripts en c#, encargados de la generación del modelo, así como un shader escrito en Cg el cual se ha utilizado para dar forma a las hojas.

6.2.1. C# Scripts

Para desarrollar la aplicación, se han creado los scripts **TreeGenerator** y **ForestGenerator**, el primero encargado de generar un único árbol y el segundo para generar un bosque. Ambos scripts están preparados para poder modificar las variables de la generación de los árboles a través de la interfaz de Unity. Para poder utilizarlos basta con añadir el script a un GameObject de Unity (preferiblemente uno vacío), y ajustar los parámetros. Al ejecutar la aplicación se ejecutara el script generando un árbol o un bosque.

Aparte de estos dos scripts, también se ha utilizado otros para poder definir los objetos utilizados por el sistema de generación de arboles, estos scripts son: **PointCloud**, **Tree**, **Branch**, **BranchMesh**, **TwoSideQuad**. Estos script son los que utilizan **TreeGenerator** y **ForestGenerator** para generar los modelos.

El ultimo script creado es el de **FirstPersonCamera**, el objetivo de este es el de poder controlar la cámara mediante el teclado y el ratón al ejecutar la aplicación.

También se utilizan algunos scripts obtenidos de Unity Comunity Wiki, una pagina donde se comparten scripts para unity, con el fin de implementar funcionalidades que el motor no posee. De esta pagina se han obtenido los scripts **ObjExporterScript**, el cual permite exportar objetos de Unity como archivos obj, y **MeshHelper** que permite modificar una malla subdividiendo los triángulos que la componen.

6.2.2. Cg Shaders

Para esta aplicación solo ha sido necesario desarrollar un shader, ya que la mayoría de los materiales utilizados en la aplicación utilizan el shader por defecto incorporado en Unity. Este shader permite, entre otras tantas cosas, aplicar un color, una textura y un "normal map.[a] los objetos, lo cual es suficiente en el caso del tronco y el suelo.

El shader creado para la aplicación es **leaveShader**, el cual se utiliza en en el material de las hojas para conseguir el efecto descrito en la seccion de “Shader” del capitulo 4.

6.3. Modo de empleo de la aplicación

Para poder utilizar la aplicación es hay que crear un nuevo proyecto en Unity y añadir todos los Assets, entre los que se encuentran los scripts, las texturas, los shaders y los materiales, también es recomendable añadir los objetos creados para generar la nube de puntos, aunque el usuario puede utilizar los objetos primitivos de Unity o cualquier otro modelo 3D cargado en Unity en lugar de estos.

El siguiente paso es crear un GameObject vació mediante la aplicación de Unity y añadirle uno de los scripts de generación de árboles (**TreeGenerator** o **ForestGenerator**). Al añadir el script a el objeto se creara en el inspector de este un interfaz donde el usuario puede ajustar los parámetros de crecimiento de el árbol. Además de los parámetros de crecimiento, la interfaz también dispone de dos campos que el usuario debe asignar, “Tree Mat” define el materia utilizado para el modelo del árbol, este campo puede ser asignado con el material DefaultTreeMat, o por cualquier otro material creado por el usuario. El campo de “Leave Material” hay que asignarlo con el material LeaveMaterial, en este caso no es posible crear un material propio ya que este esta preparado para utilizar el shader definido para las hojas.

Para añadir el objeto o los objetos encargados de definir la nube de puntos basta con añadir un objeto con el modelo elegido, y añadirle el componente Mesh Collider mediante el inspector de Unity. Tas añadir todos los objetos y posicionarlos en las posiciones deseadas mediante el editor de escenas, asignamos todos los objetos como hijos del objeto que contiene el script generador de árboles, para ello basta con seleccionar todos los objetos y arrastrarlos encima del objeto del generador.

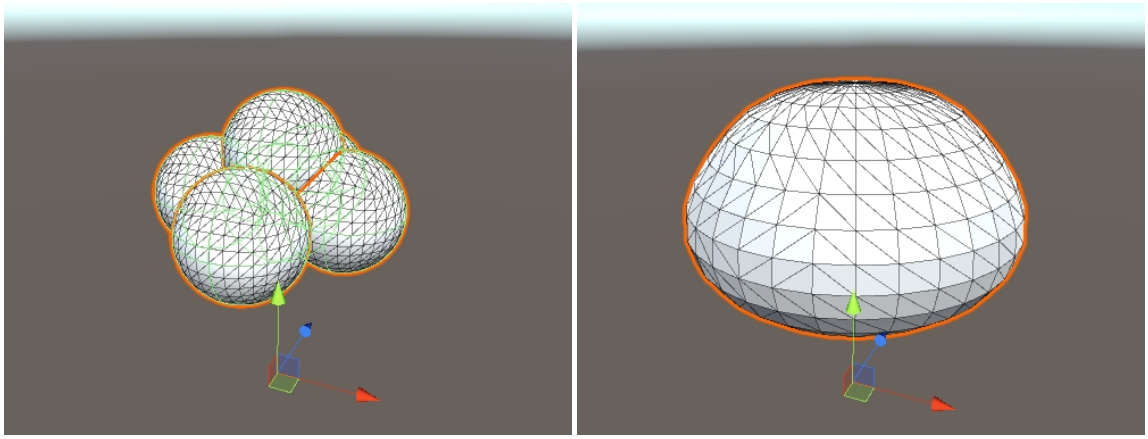


Figura 6.1: Ejemplo de dos estructuras utilizadas para generar un nube de puntos dentro del editor de escenas de Unity

En el caso del generador de bosques aparece un campo extra en la interfaz del generador "Plane Transform", para rellenar este campo es necesario crear un plano en Unity, este plano define la superficie que se quiere ocupar con el bosque. Tras crear el plano es recomendable desactivar o eliminar los componentes Mesh Renderer y Mash Collider de éste, ya que éstos no van a ser utilizados por la aplicación y en el caso de Mesh Renderer dibujara el plano al ejecutar la aplicación, otra opción es asignarle un material a este plano y utilizarlo como si fuera el terreno sobre el cual se genera el bosque.

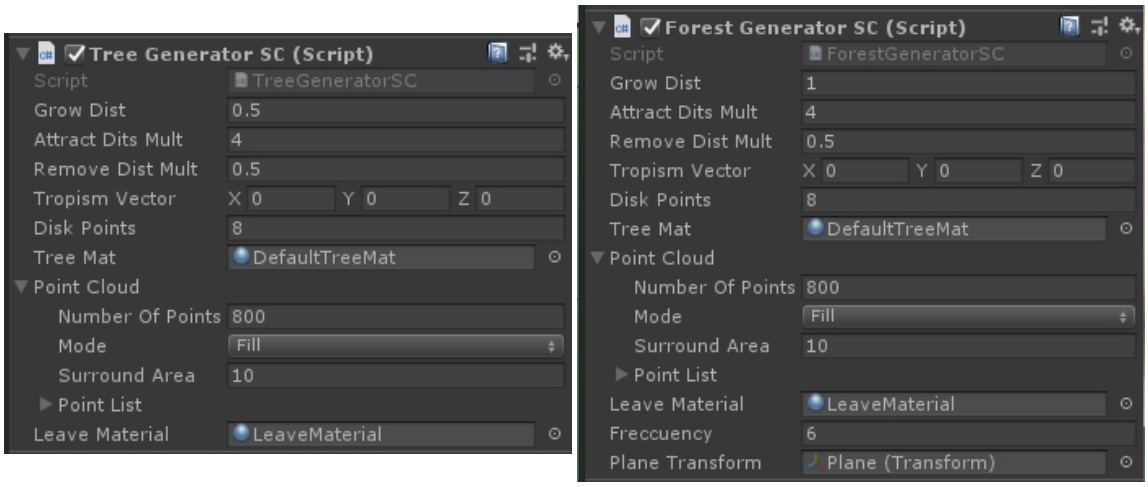


Figura 6.2: Interfaces del inspector de Unity generadas para el script generador de árboles (izquierda) y para el script genrador de bosques (derecha)

7. CAPÍTULO

Conclusiones

Tras haber desarrollado la aplicación para generar modelos de árboles, es necesario realizar un análisis de los resultados obtenidos con el fin de comprobar si se han cumplido los objetivos marcados. Tras analizar estos resultados, también se han incluido ciertas mejoras que podrían introducirse en la aplicación en un futuro, a pesar de no encontrarse dentro de los objetivos mercados inicialmente.

7.1. Resultados

7.1.1. Resultados Visuales

El objetivo de la aplicación es el de generar modelos de arboles, por lo que el mejor modo de analizar los resultados obtenidos con la aplicación es comprobar si los modelos que genera la aplicación tienen realmente una apariencia similar a la de los árboles que podemos encontrar en la naturaleza.

Además de esto es importante comprobar si los árboles generados muestran diferencias entre ellos, ya que el principal objetivo de la aplicación es la de generar una estructura diferente para cada modelo.

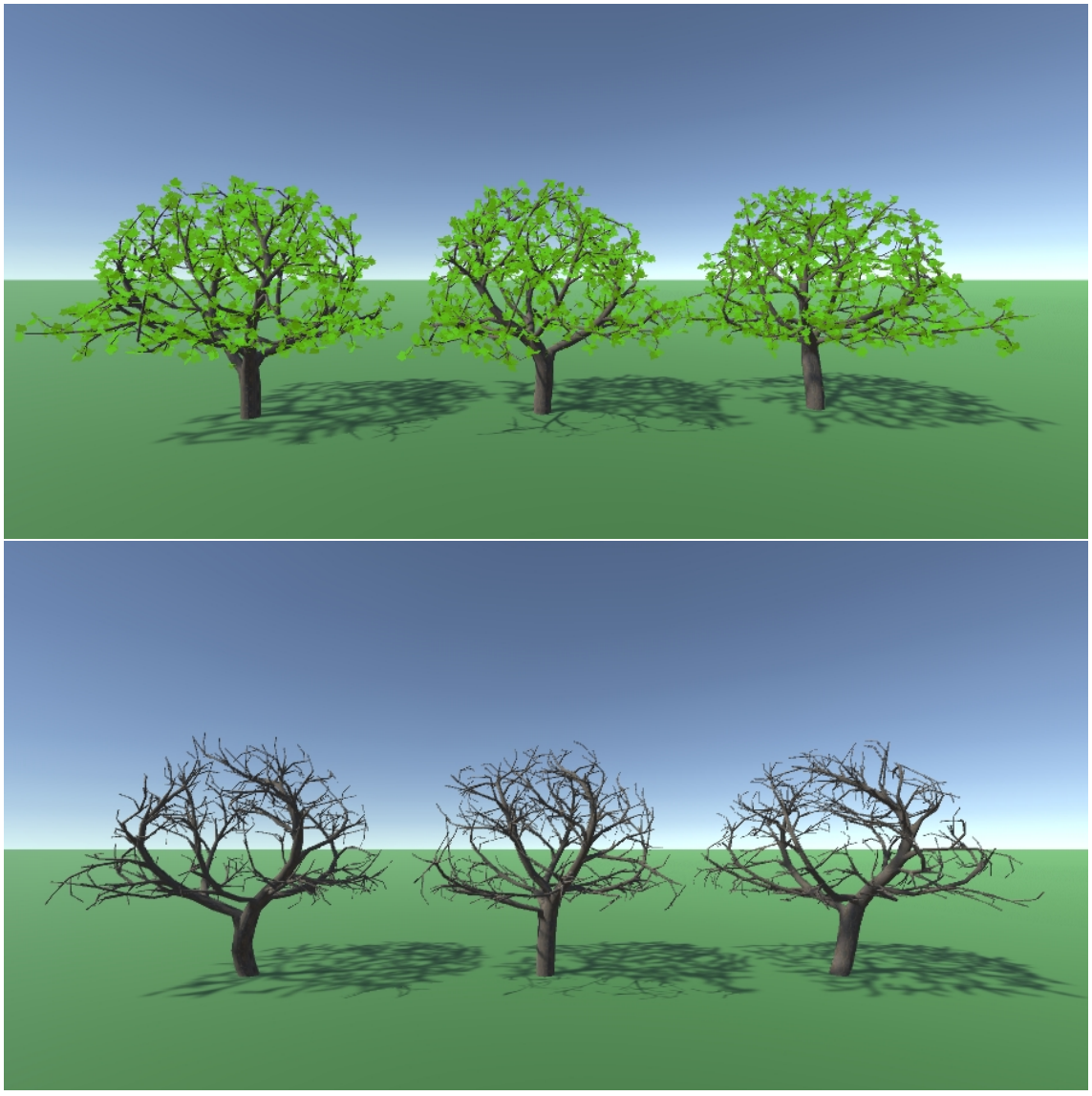


Figura 7.1: Ejemplo de varios modelos diferentes generados partiendo de los mismos parámetros, en la segunda imagen se muestran modelos sin las hojas para poder apreciar mejor la forma de éstos

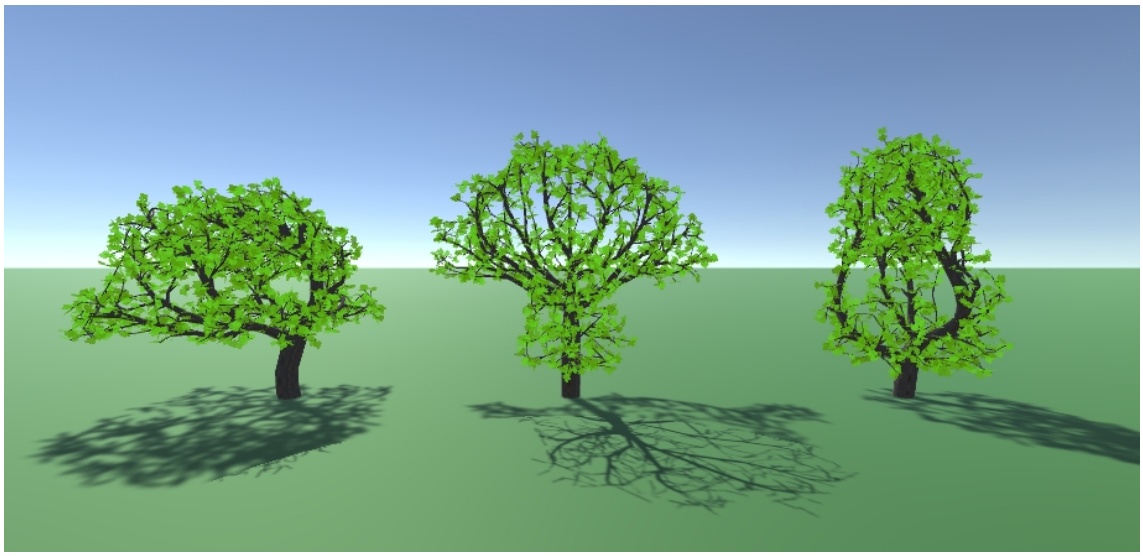


Figura 7.2: Modelos generados utilizando diferentes figuras para generar la nube de puntos

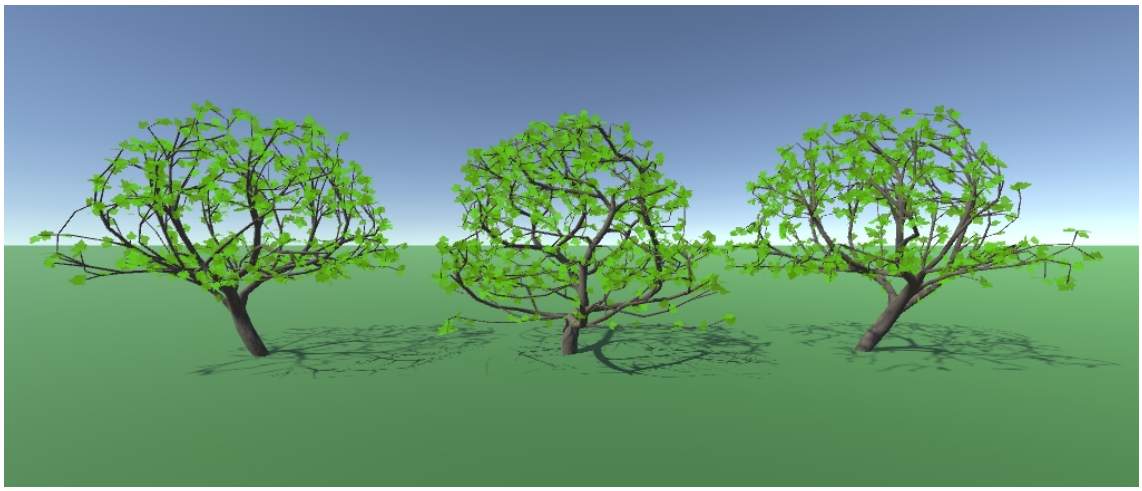
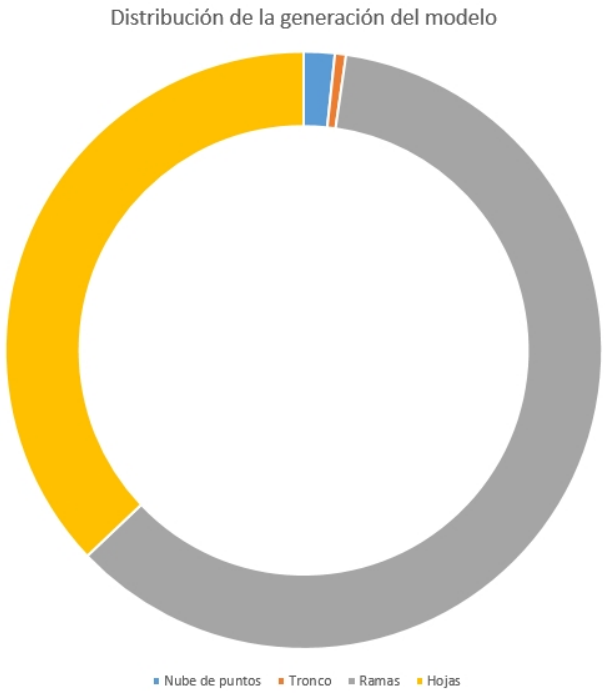


Figura 7.3: Variación de modelos mediante el vector de tropismo

7.1.2. Tiempos de creación

Generación de árboles

En esta sección se analiza el tiempo de creación de árboles individuales. Para realizar este paso se realizan varios pasos: generar la nube de puntos, generar el tronco, generar las ramas y generar las hojas. En la siguiente figura se puede observar la distribución del coste temporal de estos procesos.

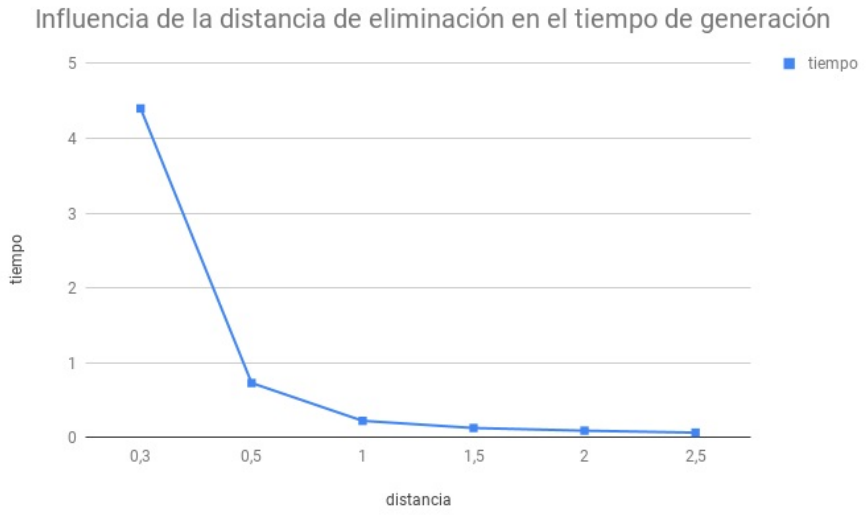


En el gráfico anterior se puede comprobar como el mayor coste reside en la generación de las ramas y las hojas, ocupando estas casi todo el tiempo del proceso. Por otro lado tanto la generación de la nube de puntos, como la generación del tronco apenas tienen influencia en el proceso.

Algunos de los parámetros de la generación de los árboles pueden influir de manera notable en el tiempo que se tarda en generar cada modelo. Los parámetros que mas afectan a esto son la el numero de puntos de la nube de puntos y la distancia de eliminación de puntos.

En el caso del numero de puntos en la nube de puntos, al aumentar la cantidad también aumenta el numero de comparaciones a realizar al generar las ramas, ya que cada uno de los segmentos es comparado con todos los puntos a la hora de calcular su dirección de crecimiento. En la siguiente imagen se puede observar la relación entre el numero de puntos y el tiempo de generación.

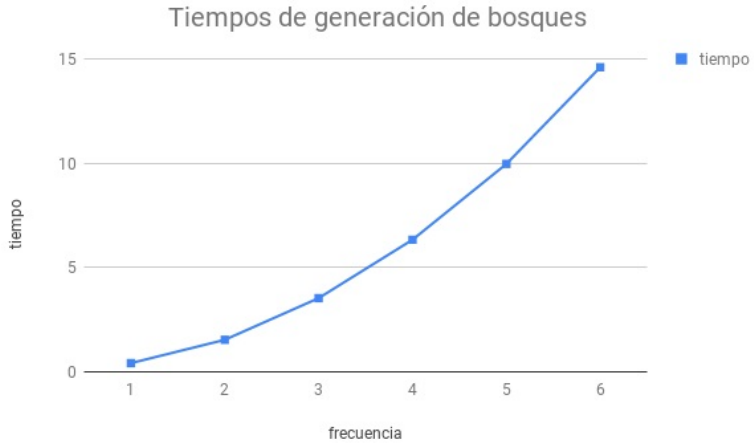
La distancia de eliminación de puntos, define la distancia máxima a la que puede estar un punto de alguno de los segmentos de rama, al reducir esta menos puntos son eliminados en cada una de las iteraciones, incrementando el numero de comparaciones necesarias.



Aumentar el numero de puntos de la nube de puntos, o reducir la distancia de eliminación de puntos, genera modelos de arboles con un mayor numero de ramificaciones, pero aumenta el tiempo necesario para generar el modelo. Debido a esto se recomienda utilizar un numero de puntos entre 500 y 1500, y distancias de eliminación entre 0,5 y 1. Con estos valores se pueden obtener modelos de árboles detallados sin tardar demasiado tiempo.

Generación de bosques

En esta sección se analiza el crecimiento del coste temporal al generar bosques mas poblados, es decir incrementando el valor de la frecuencia del blue noise.



Al aumentar el valor de la frecuencia el numero de árboles generados se incrementa el numero de arboles que se genera siendo este el cuadrado de la frecuencia, como se señala en la sección de generación de bosques. Debido a esto el tiempo de cineración de un bosque también crece con el cuadrado de la frecuencia.

7.2. Posibles mejoras para el futuro

En esta sección se recopilan una serie de mejoras que se le podrían añadir a la aplicación en un futuro. La idea de estas mejoras no están contempladas entre los objetivos descritos al plantear el proyecto, pero han ido surgiendo durante su desarrollo.

1. Optimizar el proceso de generación de segmentos utilizando octrees. Un octree es una estructura similar a un árbol de datos, que permite almacenar puntos dentro de un espacio tridimensional, de modo que resulte menos costoso buscar los puntos que se encuentra en una zona cercana. Por otro lado, durante el proceso de generación del árbol se realizan muchas operaciones relacionadas con buscar puntos a un determinada distancia, al buscar los puntos de atracción que afectan a cada segmento, o decidir que puntos deben ser eliminados por estar demasiado próximos a un segmento. Es por esto que utilizar este tipo de estructura para almacenar tanto

las posiciones de los segmentos como las de los puntos en la nube de puntos, incrementara la velocidad a la que se puede generar cada árbol mediante el algoritmo de "Space Colonization"

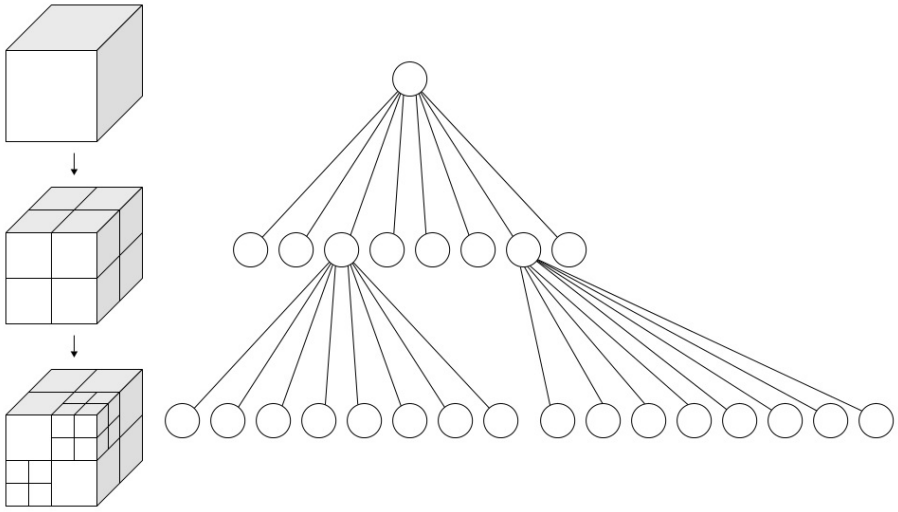


Figura 7.4: Ejemplo del modo en el que se organizan los puntos dentro de un octree (Fuente: Wikipedia)

2. Generar el material de los árboles manera procedural, para ello existen diversos shaders que combinando diferentes tipos de ruidos, generan materiales de apariencia similar a la de la corteza de los árboles. Utilizar este tipo de shaders en lugar de la textura que se utiliza en la aplicación ayudaría a obtener resultados mas realistas. Otra opción podría ser utilizar un mayor numero de texturas, o incluso alterar estas con algún tipo de ruido para generar variaciones en ellas.

3. Del mismo modo que ocurre con la textura de los troncos, la hojas también se generan a partir de una única textura, por lo que la forma de todas ellas es la misma. Del mismo modo que con las texturas de los árboles este problema podría solucionarse añadiendo mas texturas o aplicando variaciones mediante ruido a la textura utilizada. Otra solución podría ser la de utilizar el propio algoritmo de "Space Colonization" para generar la forma de éstas, ya que este método fue originalmente diseñado ese objetivo [Runions et al., 2005].

Bibliografía

[Alan Zucconi, 2015] Alan Zucconi (2015). A gentle introduction to shaders in unity3d. http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/.

[Bloomenthal, 1985] Bloomenthal, J. (1985). Modeling the Mighty Maple. *ACM SIGGRAPH Computer Graphics*, pages 305–311.

[Crego, 2017] Crego, D. (2017). Simulación de Materiales Naturales mediante Texturas Volumétricas Procedurales. https://addi.ehu.es/handle/10810/21790.

[Greuter et al., 2003] Greuter, S., Parker, J., Stewart, N., and Leach, G. (2003). Real-time Procedural Generation of 'Pseudo Infinite' Cities.

[Matt Ellis, 2017] Matt Ellis (2017). Getting started with rider and unity. https://blog.jetbrains.com/dotnet/2017/08/30/getting-started-rider-unity/.

[Olsen, 2004] Olsen, J. (2004). Realtime Procedural Terrain Generation.

[Processing Foundation,] Processing Foundation. P5 reference. https://p5js.org/reference/.

[Runions et al., 2005] Runions, A., Fuhrer, M., Lane, B., Federl, P., Rolland-Lagan, A.-G., and Prusinkiewicz, P. (2005). Modeling and visualization of leaf venation patterns. *ACM Transactions on Graphics*.

[Runions et al., 2007] Runions, A., Lane, B., and Prusinkiewicz, P. (2007). Modeling Trees with a Space Colonization Algorithm. *Eurographics Workshop on Natural Phenomena*.

[Santesteban, 2017] Santesteban, I. (2017). Generación procedural de variaciones en modelos 3D. https://addi.ehu.es/handle/10810/23787.

[Unity Technologies, 2017a] Unity Technologies (2017a). Scripting API: Mesh. https://docs.unity3d.com/ScriptReference/Mesh.html.

[Unity Technologies, 2017b] Unity Technologies (2017b). Unity Manual: Procedural Mesh Geometry. https://docs.unity3d.com/Manual/GeneratingMeshGeometryProcedurally.html.

[Unity Technologies, 2017c] Unity Technologies (2017c). Writing Your First Shader In Unity. https://unity3d.com/es/learn/live-training/session/writing-your-first-shader-unity.

[Unity Technologies, 2018] Unity Technologies (2018). Unity scripting api. https://docs.unity3d.com/ScriptReference/.

[Vázquez, 2013] Vázquez, I. (2013). Generación procedural de ciudades. https://addi.ehu.es/handle/10810/10703.