Generación procedural de árboles usando `Processing`

Diana Carolina Eslava, David Fernando Niño

20 de julio de 2013

Resumen

El presente proyecto muestra una forma mediante la cual cualquier persona puede crear y visualizar árboles sin la necesidad de tener conocimientos avanzados en programación; para esto, se presenta la librería myTree, una librería basada en los estudios realizados por el biólogo Aristid Lindenmayer quien demostró que a partir de una serie de gramáticas y reglas se puede simular el crecimiento de diferentes organismos. A estas gramáticas y reglas se les conoce hoy en día como L-Sistemas.

Básicamente, la librería myTree consta de varias funciones que le permitirán al usuario controlar el ciclo de vida de una planta, haciendo que sea personalizable la forma en que se comporta el crecimiento de cada uno de los elementos que compongan el árbol que se desee dibujar.

En el primer capitulo se habla de diferentes elementos que ayudaron al momento de la creación de esta librería; En el segundo capitulo se muestra básicamente el diseño y estructura el funcionamiento de la misma, en el ultimo capitulo se observan los resultados obtenidos durante el proceso de desarrollo, ejemplos sobre el manejo del archivo de gramáticas y los métodos que se tendrían que utilizar para manipular el crecimiento del árbol.

Índice general

Capítulo 1

Preliminares

1.1. Introducción

En 1968, Aristid Lindenmayer descubrió que a través de una serie de reglas y gramáticas se podría simular la forma de crear fractales y el crecimiento de algunos organismos vivos (como los árboles, las algas, conchas... etc.), Lindenmayer se dió cuenta de que el crecimiento de las plantas se encuentra envueto en una serie de patrones de que se repiten varias veces durante el ciclo de vida de cualquier arbol y que esta serie de patrones se podria representar facilmente usando gramáticas y reglas, y, a esta aplicación de gramáticas y reglas se les conoce hoy en día como L-Sistemas o Sistemas de Lindenmayer (en Ingles L-Systems)[7].

Los L–Sistemas son una aproximación bastante cercana al crecimiento de las plantas ya que nos facilitan el estudio del posible comportamiento de cada uno de los elementos que las componen, como por ejemplo la forma de crecer de las raíces o del tronco, el tamaño que pueden tomar sus hojas, o la manera en que el fruto madura con el tiempo. Simular el crecimiento de una planta, puede ser bastante productivo para el ser humano ya que usado con otras herramientas puede enseñar varias cosas útiles, como por ejemplo la manera de cuidar un cultivo, el comportamiento de un árbol en un ambiente extraño o incliso se podría estudiar la forma de crear nuevas especies mezclando caracteristicas de diferentes plantas.[7, 2]

La finalidad del presente proyecto es generar una librería que con base en las investigaciones de Aristid Lindenmayer y el profesor Przemyslaw Prusinkiewicz en sus libros «Visual models of plant development», «Visual models of plants interacting with their environment» y «LSystems: from the theory to visual models of plants», permita a personas con pocas o nulas nociones de programación simular diferentes modelos de plantas con solo especificar un archivo de gramáticas y reglas de crecimiento.

1.2. Gramáticas

Las gramáticas son un conjunto de reglas que a partir de los símbolos de un alfabeto permiten generar diferentes cadenas que componenen un lenguaje.

1.2.1. Definición de las gramáticas

Una gramática está compuesta por cuatro elementos principales:

$$G = (V, A, S, R) \tag{1.2.1}$$

- V = Variables, conjunto finito.
- A = Alfabeto, simbolos diferentes de V.
- S = Estado inicial $\in$ V.
- R = Conjunto de reglas $\subseteq$ V.

Lindenmayer introdujo una manera formal de representar el crecimiento de organismos multicelulares por medio de gramáticas formales.

1.3. Árboles N-arios

Estructura recursiva acíclica y no dirigida en la que cada nodo padre tiene N nodos hijos, se puede considerar que cada nodo hijo compone un nuevo árbol n-ario a excepción de las hojas, la forma de recorrer un arbol n-ario puede realizarse con las formas basicas de recorridos en arboles binarios (preorden, postorden), o recorriendo cada nodo de forma recursiva llamando el conjunto de hijos en cada nodo.

Conceptos generales de los elementos que se trabajaran en un árbol:

- Raiz: Elemento inicial del arbol, no tiene padre.
- Nodo: Elemento del arbol.
- Hoja: Ultimo nodo del arbol, no tiene hijos.
- Camino: Nodos que se encuentran entre un nodo inicial y uno final.
- Rama: Secuencia de nodos entre la raiz y un hijo.

1.4. Recursión

Funciones que se llaman a si mismas y se encargan de solucionar problemas dividiendolos en problemas similares pero más pequeños y menos complejos, esta division se compone de dos casos principalmente, el caso base que detiene la recursividad y el caso que se llama a si mismo modificando sus parámetros.

http://www.fdi.ucm.es/profesor/rgonzale/TrAlgoritmosRecursivos3339.pdf

```
void nombreProc ( τ1x1 , ... , τnxn , δ1 & y1 , ... , δm & ym )
{
// P: Precondición
// declaración de constantes

   τ1x'1 ; ... ; τnx'n ; // x', parámetros de la llamada recursiva
   δ1y'1 ; ... ; δmy'm ; // y', resultados de la llamada recursiva

  if ( d(x) )          // caso directo
    y = g(x);          // solución al caso directo
  else if ( ¬d(x) )    // caso no directo
  {
    x' = s(x);         // función sucesor: descomposición de datos
    nombreProc(x', y');// llamada recursiva
    y = c(x, y'); // función de combinación: composición de solución
  }

// Q: Postcondición
}
```

Figura 1.1: Metodo recursivo

1.5. Sistemas - L

El Biólogo Húngaro, Aristid Lindenmayer, se dedicó a estudiar los patrones de crecimiento de las plantas, su caso de estudio principal fue el de las algas, introduciendo un conjunto de reglas llamado Sistemas-L o sistemas Lindenmayer. Un L–Sistema es una gramática (o una serie de gramáticas formales) con un conjunto de reglas establecidas en un lenguaje yasea formal o lenguaje natural.

Lo anterior, no implica que L-sistemas sea lo mismo que un Lenguaje Formal, a continuación mostramos algunas de las diferencias:

- En los lenguajes formales se hace una clara distinción entre los símbolos terminales y no terminales. En los sistemas Lindenmayer no existen los símbolos terminales.
- Las producciones en los lenguajes formales, se aplican de forma secuencial, a diferencia de los sistemas Lindenmayer estas se aplican de forma paralela.

1.5.1. Trabajos sobre Modelamiento Biológico

Algoríthmic Botany, es uno de los grupos de investigación que hoy se dedican al modelamiento biológico, dirigido por el profesor Przemyslaw Prusinkiewicz `http://algorithmicbotany.org/`, por medio de la aplicación de conceptos y métodos informáticos se creó con el fin de tener una mejor comprensión de los fenómenos que se presentan en los objetos estudiados.

Para esto el equipo de investigación desarrolló una serie de paquetes (Laboratorio Virtual / L-studio) de software que modelan las plantas que están relacionadas entre sí.

Mitch Allen, Przemyslaw Prusinkiewicz y Theodore DeJong - hacen parte el grupo de investigación de Algoríthmic Botany, estudian Modelado L-PE-ACH basado en el desarrollo de árboles de durazno, cuyo enfoque es la asignación de carbono a las plantas y con base en ello, analiza la respuesta de la planta, a nivel de estructura, crecimiento y producción de fruto.

Realizaron la simulación de la asignación de carbono tomando cada uno de los órganos de la planta como un órgano con las mismas características de los otros. El modelo simula la absorción de luz, sensibilidad al agua disponible, la cual genera producción de hidratos de carbono, generando flujo de hidratos de carbono y con ello simulando el crecimiento de un árbol de durazno.

1.6. Notación de Backus-Naur (Backus Naur Form o BNF)

Es un metalenguaje utilizado para expresar lenguajes formales; es muy utilizado para definir la notación de los lenguajes de programación, pero también se puede usar para definir cualquier lenguaje natural.

Básicamente, para definir un lenguaje usando BNF es necesario especificar lo que se desea hacer de la siguiente forma:

$< Variable >::=< Expresinconsmbolos >$

Dónde $< Variable >$ es un carácter no terminal y:

$< Expresinconsmbolos >$ es una serie de caracteres que representan el lenguaje dado o un conjunto de variables concatenadas o un conjunto de variables separadas por el carácter $'|'$.

Ejemplo de un lenguaje usando BNF:

A continuación definimos el lenguaje de las operaciones básicas de las matemáticas (suma, resta, multiplicación y división) usando BNF:

$< Operacin >::=< Valor >< Operador >< Valor >$

$< Valor >::=< Signo >< Nmero >$

$< Nmero >::=< SecuenciaDgitos > | < SecuenciaDgitos >< Punto >< SecuenciaDgitos >$

$< SecuenciaDgitos >::=< Dgito >< SecuenciaDgitos > | < Dgito >$

$< Dgito >::='1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'|'0'$
$< Operador >::='+'|'-'|'/'|'*'$
$< Signo >::='-'$
$< Punto >::='.'$

Capítulo 2

Generación procedural de árboles

2.1. Objetivos del proyecto

2.1.1. Objetivo general

1. Construir un sistema para la creación y visualización de Lsistemas simbolizando plantas a partir de las reglas de gramática dadas.

2.1.2. Objetivos específicos

1. Construir un parcelador para interpretar la gramática de un Lsistema dados.
2. Construir un intérprete para gramáticas de Lsistemas dados.
3. Construir la visualización de un Lsistema dado.

2.2. Requerimientos de la librería myTree

1. Crear una aplicación fácil de usar para alguien que no tiene conocimientos de programación.
2. Cargar un archivo .txt que contenga una gramática con la estructura indicada en la sección 2.5, realizando las validaciones necesarias, generando un árbol a partir de su estado inicial y retornando mensajes que indican las razones por las que pueden ocurrir diferentes errores (en caso que existan).
3. Pintar el árbol generado a partir del estado inicial cargado desde la gramática dada.
4. Parcelar el árbol en una cadena de caracteres
5. Generar archivo de gramáticas que permita pintar un árbol desde un punto dado que da como utilidad pintar una cantidad número de árboles en la misma interfaz.

2.3. BNF aplicado a la librería myTree

La librería myTree usa cadenas que deben venir formadas de la siguiente forma:
<Cadena>::= <Elemento><contenido>| <Elemento>
<Contenido>::= <LlaveApertura><Cadena><LlaveCierre>
<Elemento>::= <Parámetros><Nodo>| <Nodo>
<Parámetros>::= <ParéntesisApertura><Param><ParéntesisCierre>
<Nodo>::=<SecuenciaCaracteres>
<Param>::= <Tiempo>,<HPadre>,<Largo>,<ÁnguloPhi>,<ÁnguloTheta>
<Tiempo>::= <Número>
<Hpadre>::= <Número>
<Largo>::= <Número>
<ÁnguloPhi>::= <Número>
<ÁnguloTheta>::= <Número>
<Número>::= <SecuenciaDígitos>| <SecuenciaDígitos><Punto><SecuenciaDígitos>
<SecuenciaDígitos>::= <SecuenciaDígitos><Dígito>| <Dígito>
<SecuenciaCaracteres>::= <Carácter><SecuenciaCaracteres>| <Carácter>
<Dígito>::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '0'
<LlaveApertura>::='['
<LlaveCierre>::=']'
<ParéntesisApertura>::='('
<ParéntesisCierre>::= ')'
<Carácter>::= cualquier carácter ASCCI, a excepción del punto y coma ';', los paréntesis '(', ')' y las llaves '[',']'.

2.4. Diseño de la interfaz

Al momento de dibujar cada una de las partes del árbol es necesario tener en cuenta varios aspectos:

- ¿Qué es lo que se desea dibujar?

  Esta es una pregunta importante porque, no se puede hacer un pastel sin conocer sus ingredientes; es necesario saber que se desea pintar y como se va a pintar; Por esta razón se ha planteado la posibilidad de que el usuario decida que desea pintar para cada uno de los elementos que compongan su árbol; por ejemplo, si se quiere pintar un tronco, podría pintarlo usando esferas. Básicamente la idea es brindar la mayor libertad de lo que se desee hacer.

  Dado esto, se ha propuesto que el usuario pueda elegir entre diferentes objetos para pintar cada uno de los nodos que compongan su árbol, los posibles elementos que se pueden escoger son un cubo, una caja, una esfera, una línea un punto y un cuadrado.

Cuando se pinte una de las anteriores figuras se debe observar que la figura no tome un tamaño exagerado, lo cual nos lleva al siguiente punto.

- El tamaño Máximo

  Si estamos pintando un árbol y queremos que sus partes no se vean desproporcionadas, debemos tener en cuenta este aspecto ya que si pintamos frutos más grandes que el tronco, ¡se vería extraño! ¿Quién ha visto frutos más grandes que un tronco? Esta inquietud nos ha llevado a definir el tamaño máximo que pueda tener cada uno de los elementos de la planta, lo único que se debe hacer es decir el tamaño máximo que se desea y listo, el aplicativo hace el resto.

  Aun así si se desea evitar al máximo que ocurran cosas de este estilo se ha adicionado la posibilidad de que el usuario "escale" los objetos que componen su árbol; pero una vez hecho esto, todos los elementos que se pinten después de realizar la escala también se verán afectados, por eso hay que tener mucho cuidado a la hora de usarlo.

- ¿Se desea un árbol con ramas negras y con hojas amarillas?

  Aunque esta pregunta suene extraña, hay plantas que tienen muchas variedades de colores, y, por esta razón se da la posibilidad de que se seleccione el color que debe tomar cada uno de los elementos del árbol, ¿Por qué no pintar un cerezo japonés, o una rara lila blanca? O porque no pintar un campo de violetas, todo depende de quién use la librería.

- ¿Cómo manejar la dirección en que crecen las ramas?

  Un punto importante al momento de pintar las ramas es saber hacia dónde deben crecer, comúnmente observamos que crecen hacia arriba, pero, ¿Cómo calcular hacia donde debemos pintarlas? Para responder a esta pregunta hablemos de coordenadas esféricas:

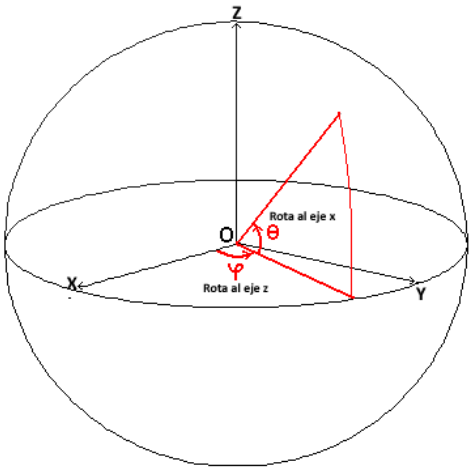


Figura 2.1: Coordenadas Esfericas

Las coordenadas esféricas se basan en la idea de las coordenadas polares, y básicamente consisten en determinar una posición en el espacio usando dos ángulos y una distancia, pero, ¿Para qué usar coordenadas esféricas? La respuesta es sencilla, si queremos calcular el ángulo de inclinación en el espacio de una rama lo único que debemos hacer el calcular dos ángulos, θ y ϕ, θ debe calcularse entre $-\frac{\Pi}{3}$ y $\frac{\Pi}{3}$, y ϕ entre $-\Pi$ y Π, haciendo esto, lo único que se debe hacer es rotar el elemento en los ángulos calculados y listo, ya se tiene la inclinación que debe tomar cada nodo.

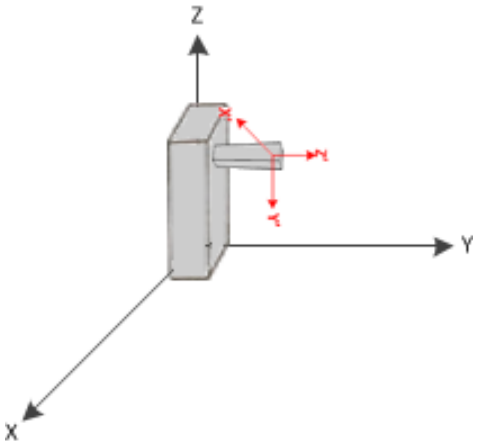


Figura 2.2: Coordenadas Nodos

- ¿Cómo crece cada elemento del árbol en el tiempo?

  Cada una de las partes del nodo crece poco a poco, la idea es que en cada instante

del crecimiento se aumente el tamaño de los elementos a pintar creciendo de a 0,1 pixeles por iteración.

- ¿Y cómo sé que se deben crear nuevos elementos?

  Bueno, básicamente la idea es que el usuario decida cada cuantas iteraciones desea que crezcan nuevos elementos en el árbol, por esta razón se recibirá un número que indique cada cuanto "tiempo" se desea crear nuevos elementos, esto se hace para evitar que en cada una de las iteraciones se generen nuevas ramas sin dar la posibilidad de crecer adecuadamente.

- ¿Y si quiero que los elementos crezcan de más de una forma?

  Es posible que las partes del árbol crezcan de diferentes formas dependiendo del tiempo que tienen de vida; los árboles, al igual que los seres humanos no crecen de la misma forma durante toda su vida, por esta razón se da la posibilidad de definir diferentes reglas para un mismo elemento, pero, con la única salvedad de que no puede haber dos reglas que se apliquen en el mismo instante.

  Por ejemplo, los árboles de café no dan frutos todos los días, solo hacen durante ciertas épocas, esto, sumado con que pueden pasar años antes de que un árbol de esta especie dé fruto por primera vez.

2.5. Diseño de persistencia

En base al comportamiento general sobre el crecimiento de las plantas se modela de la siguiente forma la aplicación.

Se parte de un archivo de configuración que tendrá la gramática de la planta (estructura del archivo). Cada ítem será considerado como un nodo. Se tendrá un tiempo de crecimiento para cada nodo con la siguiente estructura (dibujo del árbol indicando los ángulos de bifurcación X,Y,Z).

N
$ELEM_1; FIG; MAX; ESC; R; G; B$
$ELEM_2; FIG; MAX; ESC; R; G; B$
$ELEM_3; FIG; MAX; ESC; R; G; B$
.
.
.
$ELEM_N; FIG; MAX; ESC; R; G; B$
S
$REG_1 = ELEM_i; ElemCambio_j; TI; TF$
$REG_2 = ELEM_i; ElemCambio_j; TI; TF$
$REG_3 = ELEM_i; ElemCambio_j; TI; TF$
.

.
.
$REG_k = ELEM_i; ElemCambio_j; TI; TF$

- N : Cantidad de elementos que tendrá el árbol.
- k : Cantidad de reglas que define el usuario.
- i : $0 \leq$ i $\leq$ N.
- j : $0 \leq$ i $\leq$ N.
- ELEM : Tipo de elemento (P.E. Raíz, Rama, Hoja, Flor, Fruto, Fruto viejo... etc.).
- ElemCambio: Tipo de elemento por el cual cambiar(P.E. Raíz, Rama, Hoja, Flor, Fruto, Fruto viejo... etc.).
- FIG : Figura que representa el elemento (Caja, Cuadrado, esfera).
- MAX : Tamaño máximo que tomara el elemento.
- ESC : Define si el elemento tendrá la propiedad SCALE de processing.
- R : Color que representa en elemento ROJO.
- G : Color que representa en elemento VERDE.
- B : Color que representa en elemento AZUL.
- S : Elemento inicial del árbol.
- REG : Conjunto de reglas que definen el crecimiento.
- TI : Tiempo inicial de ejecución del elemento (No deben existir cruces de tiempo con otras reglas).
- TF : Tiempo final de ejecución del elemento.

Nota: El estado inicial S podría adicionalmente recibir 5 parámetros por cada uno de los subnodos que lo componen:

(T, HPadre, Largo, ϕ, θ)

- T : Tiempo actual del árbol.
- HPadre : Altura actual del nodo frente al padre $\frac{1}{2}$.
- Largo : Largo actual del arbol <MAX.
- ϕ : Angulo de giro de la rama $-\Pi$ y Π.
- θ : Angulo de giro de la rama $-\frac{\Pi}{3}$ y $\frac{\Pi}{3}$.

Capítulo 3

Resultados y Conclusiones

3.1. Resultados

3.1.1. Arquitectura de la aplicación

Respecto a la arquitectura desarrollada, el siguiente es el diagrama UML del cual solo explicaremos los métodos usados por el usuario final, la información completa está dada en reference/Javadoc.[1]

Diagrama encargado de simbolizar el árbol y las gramáticas:

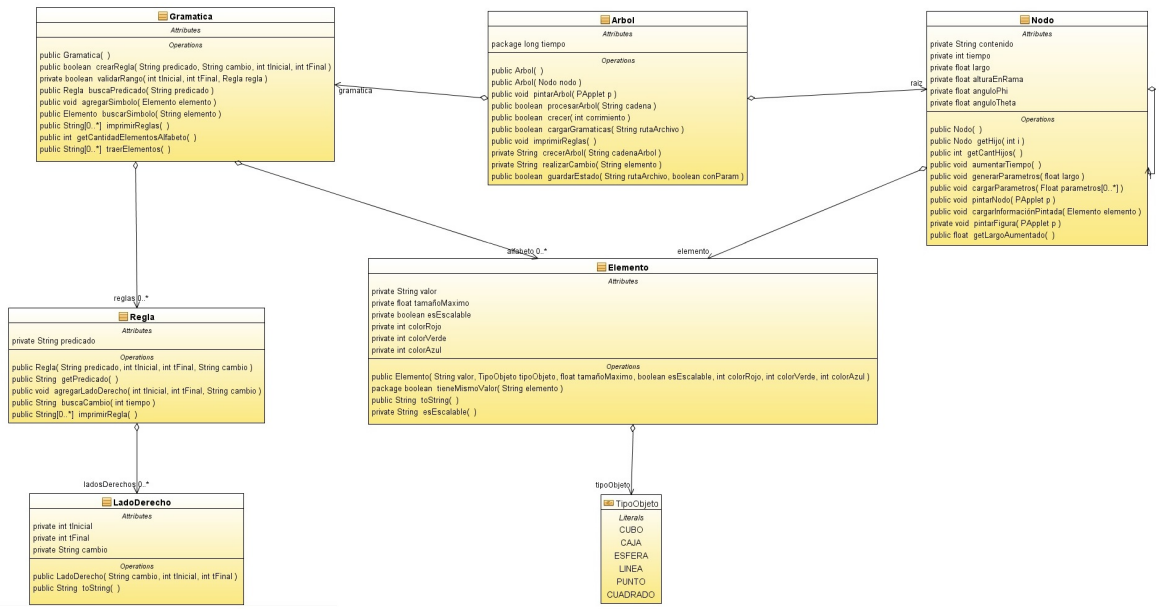


Figura 3.1: Diagrama UML de la librería myTree

Diagrama que contiene la clase ArbolNario encargada de realizar el crecimiento del

árbol:

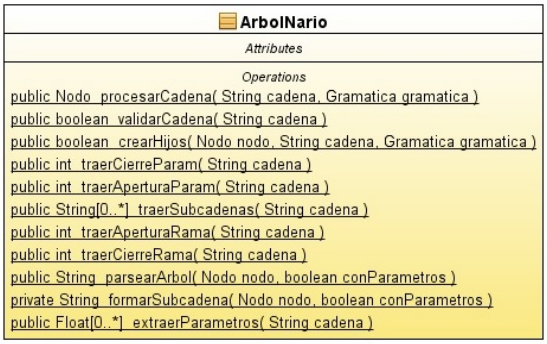


Figura 3.2: Diagrama UML de la librería myTree - vista

- **public boolean cargarGramaticas(String rutaArchivo) throws FileNotFoundException:** Método encargado de cargar al árbol las gramáticas y reglas con las cuales se va a realizar su crecimiento, recibe la ruta donde se encuentran los archivos a cargar y retorna true si se cargó correctamente el archivo, y false sí ocurrió algún error al momento de realizar la lectura. Cabe recordar que la gramática debe cumplir con la estructura especificada en el capítulo 2 2.5.

	Localización	Descripción
1	Error cargando gramática	No se reconoce el tipo de objeto XXXXXXX
2	Error cargando gramática	No se encuentra el color azul
3	Error cargando gramática	No se encuentra el color verde
4	Error cargando gramática	No se encuentra el color rojo
5	Error cargando gramática	Error leyendo la propiedad escalable
6	Error cargando gramática	Error leyendo el tamaño máximo
7	Error cargando gramática	Error leyendo tipo de objeto
8	Error cargando gramática	La cantidad de simbolos es menor a la dicha
9	Error cargando gramática	El simbolo XXXX no existe
10	Error cargando gramática	No se encuentra el tiempo final
11	Error cargando gramática	No se encuentra el tiempo inicial
12	Error cargando gramática	No se encuentra un cambio
13	Error cargando gramática	No se encuentran reglas
14	Error cargando gramática	El estado inicial tiene errores
15	Error cargando gramática	No se encuentra el estado inicial
16	Error cargando gramática	Archivo vacio o erroneo

- **public void crecer(int corrimiento):** Método encargado de hacer crecer el árbol tanto creando las nuevas ramas como haciendo que cada una de estas crezca de tamaño el crecimiento de nuevas ramas viene condicionado del parámetro corrimiento mientras mas pequeño sea mas rápido aumentara de ramas.

- **public boolean guardarEstado(String rutaArchivo, boolean conParam):** método encargado de generar un archivo de gramaticas usando como cadena inicial el estado actual del arbol.
- **public void pintarArbol(PApplet p):** Método encargado de llamar al método pintar del atributo raiz.
- **public void pintarNodo(PApplet p):** Método recursivo encargado de pintar el nodo actual, lee los campos del campo "elemento" para saber cómo debe comportarse la pintada del nodo y posteriormente se llama a sí mismo por cada uno de los hijos del nodo actual. Antes de pintar un elemento este método se rota en los ángulos X y Y, posteriormente traslada la posición del cursor a las coordenadas (0,-$\frac{largo}{2}$,0).

3.1.2. Algoritmos especiales utilizados

- parsearArbol: El parser es un algoritmo recursivo que permite a partir de un árbol dado generar nuevamente la cadena (con parámetros si el valor del atributo conParam es true).

```
public static String parsearArbol(Nodo nodo, boolean conParametros) {
String cadena = formarSubcadena(nodo,conParametros);

for(int i =0; i<nodo.getCantHijos(); ++i){
cadena= cadena + "[" +parsearArbol(nodo.getHijo(i), conParametros) + "]";
}
return cadena;
}
```

- crearHijos: Algoritmo recursivo que permite la creación de los nodos para cada uno de los elementos de la gramática.

```
crearHijos(Nodo nodo, String cadena, Gramatica gramatica){
boolean estadoCreación = true;

// Si la cadena no viene vacia
if(cadena.length()>=1) {
int inicioP = traerAperturaParam(cadena);
int cierreP = traerCierreParam(cadena);
int pos = traerAperturaRama(cadena);

if(cierreP != -1) {
```

```
//Si se encuentran parámetros se cargan al arbol
String param = cadena.substring(inicioP + 1, cierreP);
ArrayList<Float> parametrosArbol = extraerParametros(param);
nodo.cargarParametros(parametrosArbol);
cadena = cadena.substring(cierreP +1);
} else {

// si no se encuentran parametros se generan unos nuevos.
nodo.generarParametros(10);
}
if(pos != -1) {

// si la rama actual tiene hijos
pos = traerAperturaRama(cadena);
nodo.setContenido(cadena.substring(0, pos));
Elemento elemento = gramatica.buscarSimbolo(nodo.getContenido());
//si el valor de la rama existe en el alfabeto

if(elemento != null) {
nodo.cargarInformaciónPintada(elemento);
cadena = cadena.substring(pos);
ArrayList<String> subcadenas = traerSubcadenas(cadena);

// se realiza el llamado a la función crearHijos por cada uno de los nodos.

for(int i =0; i<subcadenas.size() && estadoCreación; ++i) {
if(subcadenas.get(i).length()>0){
Nodo nodoN = new Nodo();
nodo.addHijo(nodoN);
estadoCreación = crearHijos(nodoN, subcadenas.get(i), gramatica);
}
}
}else {
// si el valor de la rama no existe en el alfabeto
// se detiene la ejecución.
estadoCreación = false;
}
}else {

// si la rama no tiene hijos se continua con el proceso.
nodo.setContenido(cadena);
Elemento elemento = gramatica.buscarSimbolo(nodo.getContenido());
```

```
if(elemento!=null) {

// si el valor de la rama existe en el alfabeto se carga la nueva rama.
nodo.cargarInformaciónPintada(elemento);
} else {
// si el valor de la rama no existe en el alfabeto
// se detiene la ejecución.
estadoCreación = false;
}
}
}
return estadoCreación;
}
```

3.2. Evolución del desarrollo de la librería myTree

El propósito de esta sección es dar un recorrido en la evolución de la librería myTree, desde sus diseños básicos, lo que se tiene actualmente y un abrebocas de lo que se esperaría a futuro.

En la primera versión se contempló lo siguiente:

- Figuras y colores estándar para los elementos del árbol; para la raíz y ramas eran cajas, para hojas y frutos eran cuadrados de diferentes colores y para frutas eran esferas.

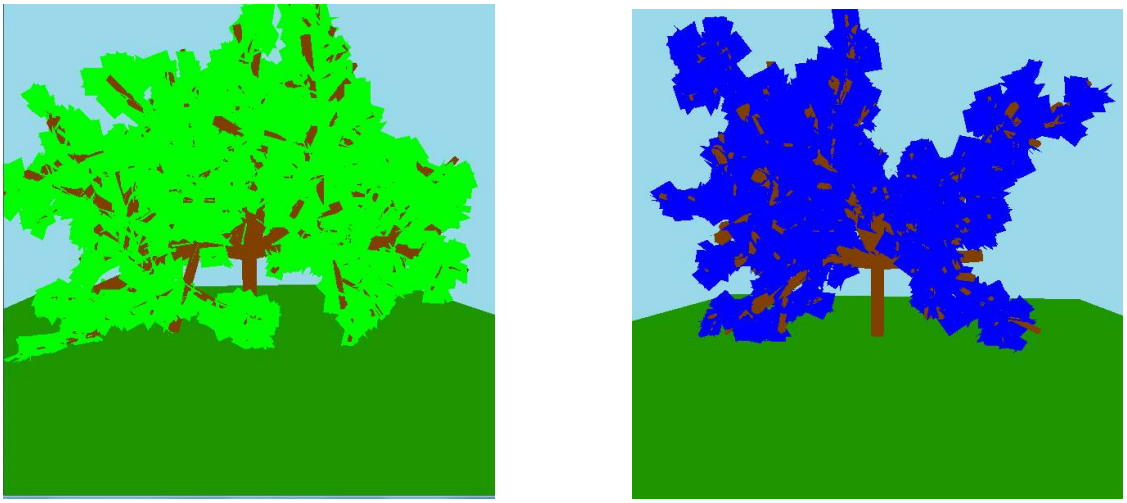
- Ángulos de rotación entre $-\frac{\Pi}{2}$ y $\frac{\Pi}{2}$.

De la ejecución 3.3 se deducen las siguientes conclusiones :

- Se observa una imagen plana y que a pesar de tener estructura de planta aun le faltaba forma.

- La forma de crecimiento de las hojas estaba muy dispersa por lo que en algunos casos las hojas llegaban al piso.

- No le daba al usuario la posibilidad de cambiar la forma y color de las figuras.



(a) (b)

Figura 3.3: Primera versión librería myTree (a) primer árbol generado con la librería, (b) segundo ejemplo generado con cambios de color

A partir de las imágenes obtenidas, el primer punto que se busca mejorar es darle fondo e iluminación a los elementos, por lo que se utiliza el método lights(); de processing, con esto la planta adquiere una forma algo más realista y sombreada.

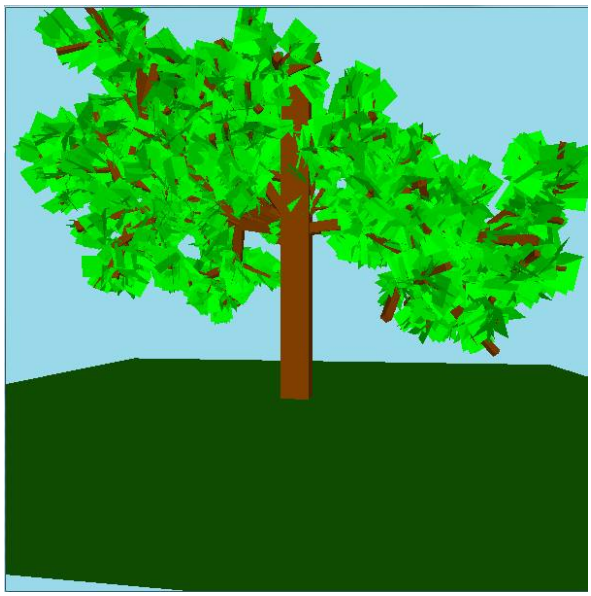


Figura 3.4: Segunda versión librería myTree aplicando el método lights() de processing

Lo que se observa en el árbol de la figura 3.4 es que sus hojas están muy dispersas

aun, esto se debe a que su ángulo de abertura θ es muy grande $-\frac{\Pi}{2}$ y $\frac{\Pi}{2}$. Para mejorar el ángulo de abertura en que se bifurca cada rama se reduce el ángulo a $-\frac{\Pi}{3}$ y $\frac{\Pi}{3}$.

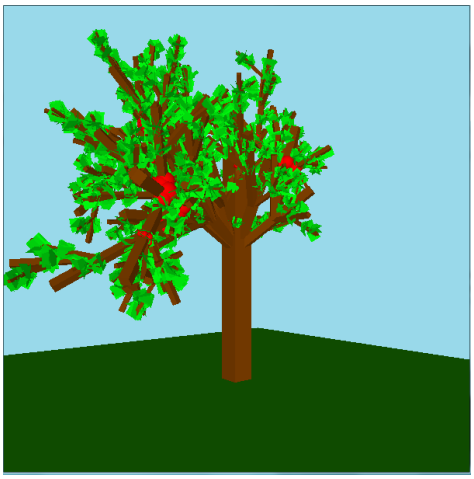
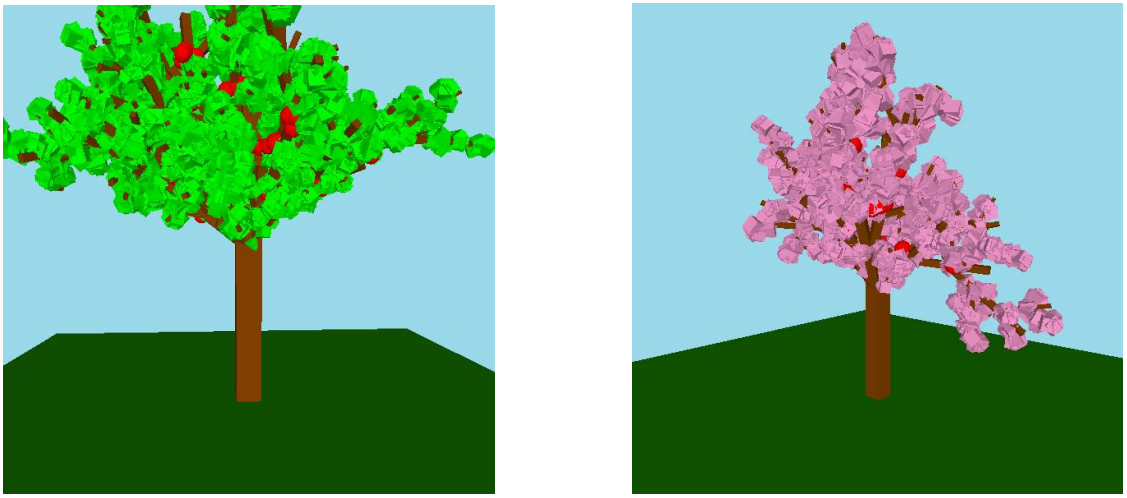


Figura 3.5: Tercera versión de la librería myTree que permite acortar el ángulo de apertura del árbol

En este punto ya se cuenta con una adaptación más real de las plantas, pero si el usuario quisiera cambiar el estilo del árbol ya sea el color o forma de las hojas, flores, frutos o ramas no podría debido a que la interfaz se dejó estándar. A partir de este problema nace la idea planteada en el capítulo 2.5, que permite ingresar de cada elemento que se quiere pintar, figura, tamaño máximo, si es escalable o no y color. En la siguiente imagen se muestra la nueva implementación utilizando para las hojas cubos y en la segunda imagen se ve el cambio de color en las hojas.



(a) (b)

Figura 3.6: Cuarta versión de la librería myTree que permite parametrizar los elementos del árbol. (a) Las hojas se reemplazan por cubos, (b) Cambia el color de las hojas por medio del archivo de gramáticas

Se realiza la implementación al método crecer un parámetro de corrimiento, que básicamente le indica al árbol que tan rápido (iteraciones) le nacerán nuevas ramas, mientras más pequeño sea el número más rápido le crecerán ramas y en consecuencia el árbol será más pequeño y frondoso.

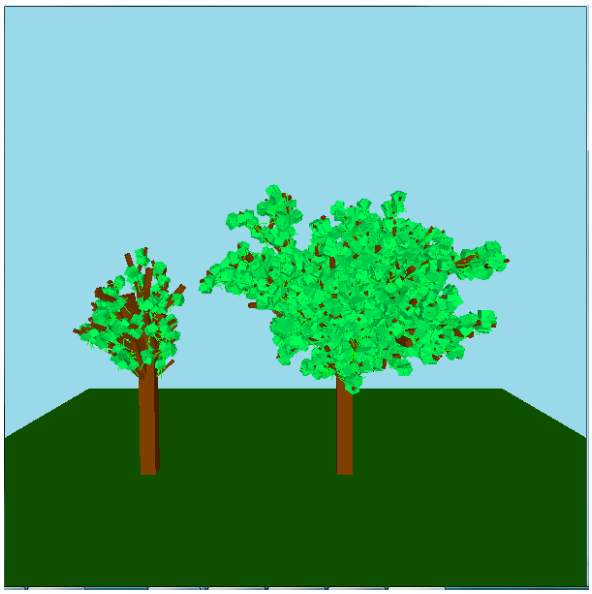
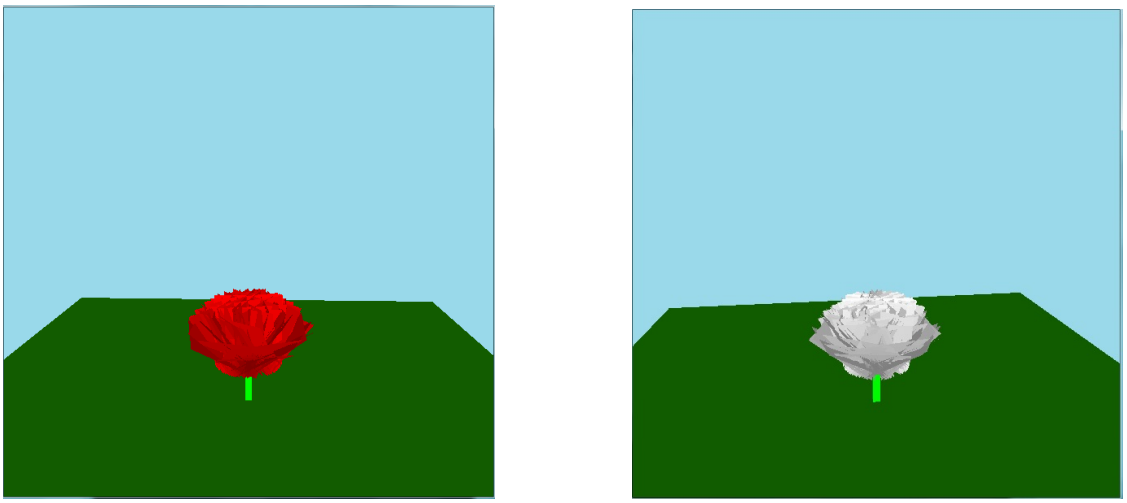


Figura 3.7: Comparativo en tiempo del parámetro corrimiento en el método crecer. (Izq) valor 400, (Der) valor 200

Como mejora de la librería se pensó el caso en que el usuario quiera pintar más de un árbol al mismo tiempo con el fin de generar un bosque o cultivo. En este caso, para evitar procesar el crecimiento de muchos árboles simultáneamente, se da como solucion que a partir de un árbol en crecimiento se pueda generar la gramática que se tiene hasta ese momento en un archivo *.txt. A partir de este archivo se puede cargar la gramática nuevamente, teniendo en cuenta que el crecimiento del árbol se encuentra a libre escogencia del usuario. En el caso que los arboles crezcan, a partir del punto de la gramática lo harán de forma diferente cada uno.

Figura 3.8: Bosque cargado a través de la gramática generada por el método guardarEstado()

A partir del archivo *.txt estructurado al inicio del documento 2.5, se pueden generar diferentes tipos de gramáticas, desde las más sencillas, hasta algunas más complejas de plantas o flores, sin tener la necesidad de conocer sobre desarrollo. A continuación y como ejemplo final se muestra cómo se puede generar una rosa a través de la librería myTree.



(a) (b)

Figura 3.9: Rosas generadas por la librería myTree con cuadrados y cajas, (a) roja roja, (b) rosa blanca

3.3. Interfaz y visualización

En base a la librería myTree.jar, se muestra a continuación dos ejemplos completos iniciando con el archivo de gramáticas con la estructura descrita en el 2.5:

- Archivo de gramáticas gram2.txt

```
6
raiz;CAJA;500;ESCALABLE;128;64;0
rama;CAJA;250;NOESCALABLE;128;64;0
hoja;ESFERA;25;NOESCALABLE;75;84;200
flor;CUADRADO;50;NOESCALABLE;255;64;0
fruto;ESFERA;35;NOESCALABLE;255;0;0
fruto viejo;ESFERA;35;NOESCALABLE;255;60;0
raiz
raiz;raiz[rama][rama];0;1000
rama;rama[rama[hoja][hoja][hoja][hoja][hoja][hoja]];0;100
rama;rama[rama[hoja][rama]];101;200
hoja;hoja[fruto];5;6
flor;fruto;2;3
fruto;fruto viejo;50;51
```

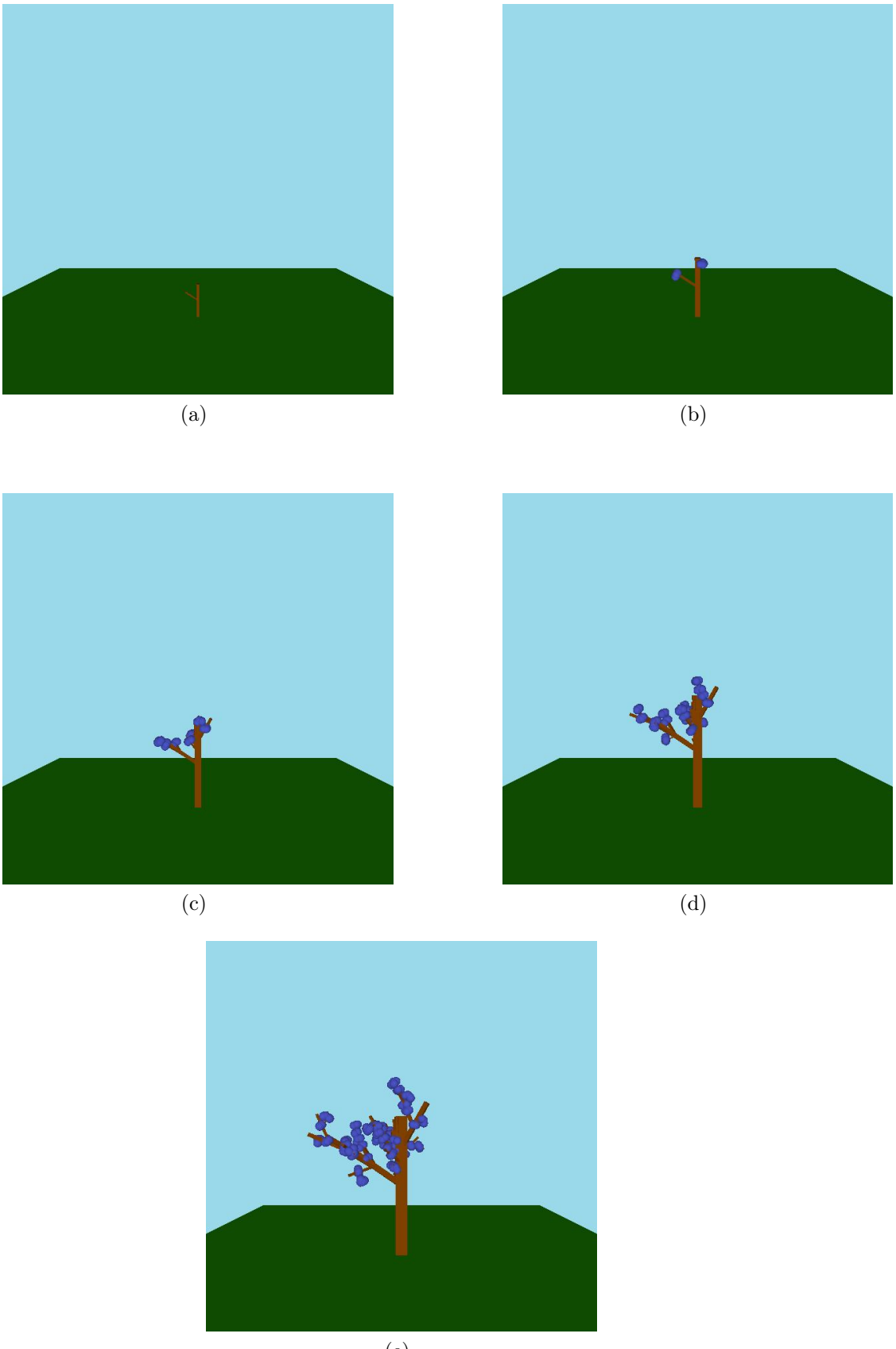
Con este archivo se pueden llamar los siguientes métodos principales de la librería para la manipulación de la misma, es importante recordar que la carpeta myTree debe estar ubicada en "/Users/XXX/Documents/Processing".

- cargarGramaticas(string rutaArchivo)
- pintarArbol(PApplet papplet)
- crecer(int rangoCrecimiento)

```
//Importar el paquete myTree de la librería myTree
import myTree.*;

//Instanciar el objeto Arbol
Arbol miArbol = new Arbol();

int rangoCrecimiento = 400;

//En el método setup indicar la ruta de la gramática
public void setup(){

rutaArchivo = "gram2.txt" ;

try {

//cargar gramáticas del objeto árbol
miArbol.cargarGramaticas(rutaArchivo);

} catch (FileNotFoundException ex) {}
}

public void draw() {

lights();

//Pintar el árbol en base a la gramática cargada
miArbol.pintarArbol(this);

//Hacer crecer el árbol
miArbol.crecer(rangoCrecimiento);

}
```

El método lights() de processing permite dar iluminación al árbol y debe llamarse antes de pintar el árbol.

El proceso de pintar y hacer crecer el árbol se encuentran separados como se muestra en el código lo que permite pintar varias arboles grandes sin que el procesamiento detenga o vuelva lenta la aplicación. Con la gramática y el código indicado la aplicación genera el siguiente árbol:



(a) (b) (c) (d) (e)

Figura 3.10: Secuencia de crecimiento de la gramática gram2.txt, (a) crecimiento de la raíz, (b) nacimiento de las primeras hojas, (c) crecimiento de la planta y sus nuevos hijos, (d) crecimiento de hojas parametrizadas como cubos y color violeta, (e) Estructura empieza a tomar forma de árbol

La librería cuenta con método guardarEstado(String rutaArchivo, boolean conParam); que permite generar a través de un árbol que se encuentre en crecimiento, la gramática y parámetros obtenidos hasta el momento, en el siguiente método se usa el evento keyPressed(). Para usarla utilizamos los métodos vistos anteriormente cargarGramaticas(String rutaArchivo), pintarArbol(PApplet papplet), crecer(int rangoCrecimiento) si se quiere que el árbol siga creciendo. Se puede ver el archivo de gramáticas generado en Gramática generada

```
public void keyPressed(){
if (key == 'd' || key == 'D') {
miArbol.guardarEstado("D:\\gram2ConParam.txt", true);
    }
}
```

En la primera imagen se ve cuando se guarda la gramática con el nombre gram2ConParam.txt y en el segundo se carga en un nuevo proyecto esta gramática para que el árbol inicie con esos parámetros.

```
  //setup()
rutaArchivo = "D:\\gram2ConParam.txt" ;
funciona = miArbol.cargarGramaticas(rutaArchivo);

//draw()
miArbol.pintarArbol(this);
```

(a) (b)

Figura 3.11: Arboles cargados por medio de una gramática guardada previamente, (a) Generación de la gramática, (b) Carga de la gramática guardada anteriormente

3.4. Cosas a mejorar

1. Migrar a processing 2.0 debido a que esta versión facilita el manejo de archivos XML y JSON.

2. Implementación que permita manipular la escala de las ramas teniendo en cuenta largo, ancho y profundidad.

3. Implementar mayas en 3D con el fin de mejorar la interfaz gráfica.

4. Aplicación de texturas al momento de visualizar las mayas.

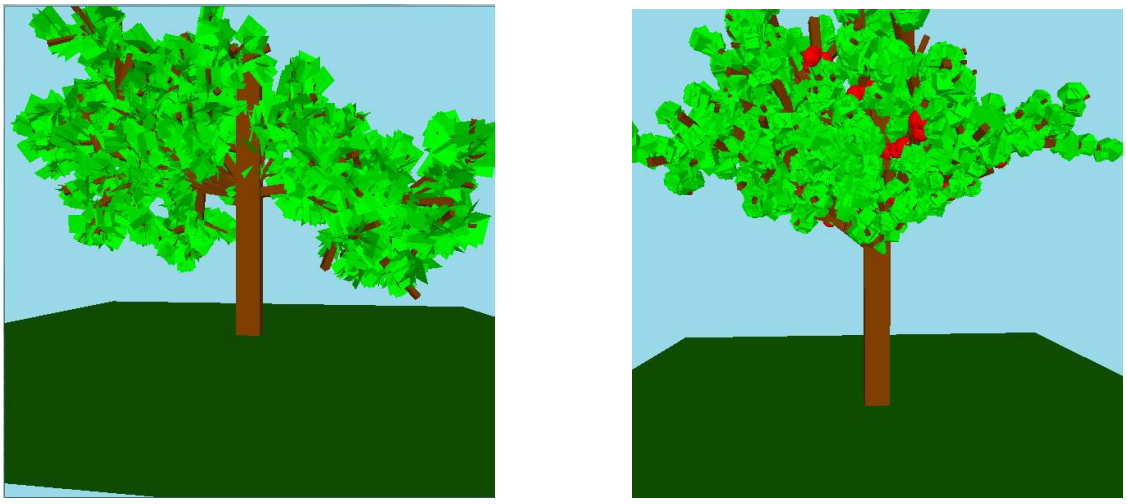
5. A futuro se quiere pintar cualquier tipo de elemento en cualquier altura con respecto a su padre es por esto que se usa el parámetro HPadre (sección 2.5) que en el momento no se usa; la altura a la cual se pintan los nodos nuevos es 0.5 (la mitad del nodo anterior).

6. Actualmente los ángulos de cada uno de los nodos (ϕ, θ) se encuentran en los mismos rangos de variación sin importar el tipo de nodo que se está pintando, se ve la necesidad de independizar los ángulos por tipo de nodo permitiéndole al usuario la manipulación de la dirección de los objetos dibujados.

7. El método guardarEstado(String rutaArchivo, boolean conParam) recibe un atributo booleano que se usa para decir si el árbol se va a guardar con sus cinco parámetros actuales (los parámetros de cada nodo) o si solo guarda el parámetro LARGO (sección 2.5), esto permite que a partir de una misma gramática y alturas se puedan generar diferentes formas de árboles. Actualmente este método esta actualizado en true.

3.5. Conclusiones

1. A partir de la gramática se define la forma y el orden en que crece el árbol.
2. El ángulo de inclinación θ debe estar entre $-\frac{\Pi}{3}$ y $\frac{\Pi}{3}$ de lo contrario el árbol tiende a tener un espacio de abertura mayor y pierde su frondosidad (3.12).



(a) (b)

Figura 3.12: Angulo de bifurcación entre raíces, (a)Angulo entre $-\frac{\Pi}{2}$ y $\frac{\Pi}{2}$,(b) Modificación del ángulo $-\frac{\Pi}{3}$ y $\frac{\Pi}{3}$

3. Es necesario hacer los cambios de las reglas en cada una de las partes de la cadena, no solo en los nodos finales ya que si se hiciera esto el crecimiento sería muy disparejo.
4. Dado que la generación de los ángulos y la altura en la rama son aleatorios se obtienen diferentes representaciones para una misma gramática, lo cual se puede observar en el crecimiento real de los árboles.
5. En el proceso de pintado y crecimiento del árbol entre más grande y complejo sea objeto más se demorara en pintarlo y más recursos de máquina consumirá.

6. Los métodos pushMatrix() y popMatrix() de processing facilitan el posicionamiento y rotación de cada uno de los elementos que componen un árbol durante su crecimiento.

7. Processing posee varios métodos optimizados para que pintar las figuras tome menos tiempo que si se pintaran en otros lenguajes de programación.

Bibliografía

[1] Ronan Amorim, Emilio Vital Brazil, Daniel Patel, and Mario Costa Sousa. Sketch Modeling of Seismic Horizons from Uncertainty. pages 1–10.

[2] Frederic Boudon, Christophe Pradal, Przemyslaw Prusinkiewicz, and Christophe Godin. L-py: an l-system simulation framework for modeling plant architecture development based on a dynamic language. *Frontiers in Technical Advances in Plant Science*, 3:76, 2012.

[3] David Gries. *A logical approach to discrete math.* Springer, New York [u.a., 1993.

[4] Gabor T Herman and Grzegorz Rozenberg. *Developmental systems and languages.* North-Holland Pub. Co. ; American Elsevier Pub. Co., Amsterdam; New York, 1975.

[5] Przemyslaw Prusinkiewicz. *Lindenmayer systems, fractals, and plants.* Springer-Verlag, Berlin; New York, 1989.

[6] Przemyslaw Prusinkiewicz and Brendan Lane. Modeling morphogenesis in multicellular structures with cell complexes and l-systems. In Vincenzo Capasso, Misha Gromov, Annick Harel-Bellan, Nadya Morozova, and Linda Louise Pritchard, editors, *Pattern Formation in Morphogenesis*, number 15 in Springer Proceedings in Mathematics, pages 137–151. Springer Berlin Heidelberg, January 2013.

[7] Przemyslaw Prusinkiewicz and Adam Runions. Computational models of plant development and form. *New Phytologist*, 193(3):549â569, 2012.

[8] Lindenmayer Przemyslaw. *The Algorithmic Beauty of Plants.* Springer, January 1991.

[9] (primero) Salmon. *Backus-Naur Forms.* Irwin Professional Publishing, July 1992.

[10] Daniel Shiffman and Shannon Fry. *The nature of code.* 2012.

[11] Karan Singh and Levent Burak Kara, editors. *Sketch Based Interfaces and Modeling*, Annecy, France, 2012. Eurographics Association.