INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS CIUDAD DE MÉXICO

NATURALEZA VIRTUAL INTERACTIVA

T E S I S

QUE PARA OPTAR AL GRADO DE

MAESTRO EN CIENCIAS COMPUTACIONALES
ESPECIALIDAD EN GRÁFICAS

P R E S E N T A

EDUARDO HERNÁNDEZ PALMA

ASESOR: DR. BEDRICH BENES

MEXICO, D.F. 2005

Índice de Contenido

Capítulo 1

La naturaleza virtual

La representación de elementos de la naturaleza por medio de gráficas computacionales siempre ha sido un reto importante debido a su alto grado de complejidad y características fractales. El problema se complica cuando necesitamos incluir estos elementos en aplicaciones de tiempo real, donde es necesario mantener una tasa mínima de cuadros por segundo (FPS — *Frames Per Second*) para que el usuario experimente una respuesta interactiva.

Se han logrado avances importantes en la representación de terrenos, nubes y algunos otros elementos. Sin embargo la complejidad creciente de las aplicaciones en tiempo real exigen algoritmos más eficientes, que mejoren la representación visual y consuman menos recursos.

El problema de representar plantas en Gráficas Computacionales (CG — *Computer Graphics*) se puede clasificar en 2 grandes áreas:

Modelado: Se enfoca en la creación de la geometría de la planta. La complejidad de esta geometría radica en que a diferencia de los objetos creados por el hombre, las plantas están compuestas por muchos objetos aislados y pequeños.

Actualmente existen varias técnicas que permiten modelar plantas con alta calidad visual. Algunos ejemplos son *L-systems* [PL90], modelado basado en componentes como *xfrog* [xfr04], etc.

Rendering: Se encarga de interpretar la geometría de las plantas y mostrarla en el dispositivo de visualización, considera aspectos como iluminación, sombra, nivel de detalle (LOD — *Level Of Detail*), dinámica, entre otros. Debido a la compleja geometría de las plantas, se requieren

algoritmos eficientes, especialmente en aplicaciones de tiempo real.

El objetivo inicial de esta investigación fue extender la técnica de nivel de detalle desarrollada por Deussen *et al.* [DCSD02] con un enfoque específico en árboles y mejorar varios aspectos entre los cuales destacan:

- ☆ El análisis automático de árboles para generar una versión basada en puntos y líneas.
- ☆ Segmentación automática de árboles.
- ☆ Mejorar la transición entre los distintos niveles de detalle.

Durante esta investigación se detectaron áreas de oportunidad para ofrecer soluciones a otros problemas asociados con el *rendering* de árboles. Finalmente la investigación completa abarca las siguientes áreas:

- ☆ Nivel de detalle (LOD)
- ☆ Dinámica
- ☆ Nivel de detalle en tiempo crítico (TCLOD — *Time Critical Level Of Detail*)

1.1. Organización del contenido

El contenido de esta tesis esta organizado en base a las tareas que el usuario necesita efectuar para generar un escenario virtual. Estas tareas han sido agrupadas en tres grandes etapas que permiten al usuario modular y controlar el tipo de naturaleza virtual que desea. Estas etapas son:

1. Geometría del modelo y preproceso
2. Generación de archivos
   a) Archivos de árboles
   b) Archivos de ecosistemas
3. Rendering
   a) Nivel de detalle (LOD)
   b) Dinámica

c) Nivel de detalle en tiempo crítico (TCLOD)

Las dos primeras etapas establecen las bases para efectuar los algoritmos de *rendering* en la última etapa. A continuación se presenta una breve descripción de cada una de estas etapas y se indica como ha sido organizada la información en este texto:

Geometría del modelo y preproceso: Establece las características del modelo del árbol y las restricciones necesarias para que este modelo pueda ser útil en los procesos posteriores. Un preproceso analiza el modelo del árbol con el fin de obtener información de cómo esta construido, en base a esta información se organizan y reescriben los elementos del modelo. Los detalles se encuentran en el capítulo titulado "Análisis de datos".

Generación de archivos: Se pueden generar dos tipos de archivos: de árboles y de ecosistemas. Los archivos de árboles utilizan la información preprocesada y la almacenan en una forma conveniente para el proceso de *rendering*. Dependiendo del objetivo que busque el usuario puede elegir entre dos variantes: archivos de árboles estáticos con extensión ".tree" o archivos de árboles sensitivos con extensión ".stree". Los primeros están descritos en el capítulo titulado: "Nivel de Detalle" mientras que los segundos están descritos en el capítulo titulado "Dinámica".

Los archivos de ecosistemas describen las características del ecosistema virtual que contendrá una o más instancias de árboles. Su contenido esta descrito en el capítulo titulado "Ecosistemas".

Rendering: Como se mencionó anteriormente, esta tesis se enfoca en los siguientes problemas de *rendering*:

Nivel de detalle: Consiste en sustituir una geometría compleja por otra más simple que tenga una apariencia similar con el fin de economizar los recursos de cómputo que se requieren para el *rendering* de la escena. La descripción de esta investigación se encuentra en el capítulo titulado "Nivel de Detalle".

Dinámica: Permite que la naturaleza virtual sea sensible a la simulación de fuerzas presentes en la escena. La descripción de esta investigación se encuentra en el capítulo titulado "Dinámica".

Nivel de detalle en tiempo crítico: Cuando esta técnica es aplicada a la generación de imágenes, su propósito es ajustar la calidad de la imagen dinámicamente para mantener la tasa de cuadros requerida por el usuario o la aplicación. La descripción de esta investigación se encuentra en el capítulo titulado "Nivel de Detalle en Tiempo Crítico".

Debido a que esta tesis se enfoca en diversos problemas del *rendering* de árboles, la descripción del estado del arte, los resultados, las conclusiones parciales de cada tema y el trabajo a futuro han sido distribuidos en las distintas secciones del texto, de tal forma que el lector pueda encontrar el contenido relacionado en un mismo capítulo.

1.2. Línea de investigación

La línea de investigación de esta tesis conforme a la especificación de líneas de investigación publicadas por el ACM (*Association for Computing Machinery*) en 1998 [acm98] es:

- ☆ I.3 Gráficas computacionales (CG)
  - ☆ I.3.3 Generación de imágenes
  - ☆ 1.3.5 Modelado de objetos y Geometría computacional
  - ☆ 1.3.7 Gráficas en tercera dimensión y realismo

Capítulo 2

Análisis de datos

En este capítulo se describe que características debe tener la geometría del árbol que se desea procesar. Posteriormente se identifican los pasos necesarios para el análisis de la geometría y la forma en que se reorganizan los datos para un uso posterior.

2.1. Geometría del modelo

Los modelos que se utilizan en esta investigación deben cumplir con ciertas restricciones que se describen en esta sección. Cualquier software o técnica especializada para la construcción de plantas y árboles virtuales que cumpla con estas restricciones puede exportar sus modelos a la aplicación de preproceso y posteriormente obtener los beneficios de los algoritmos de *rendering* propuestos en esta tesis. Las restricciones son las siguientes:

- El modelo esta compuesto en su totalidad por triángulos, es decir, no se admite ningún otra primitiva o polígono que no tenga tres vértices.
- El modelo se descompone en dos grandes partes:
  - Las hojas
  - El tronco
- Hojas:
  - Se representan como triángulos aislados que no tienen ninguna conectividad entre ellos ni tampoco con el tronco.

- ☆ Tienen asociadas un material que describe su interacción con la iluminación de la escena.
- ☆ Tienen asociada una textura.

☆ Tronco:

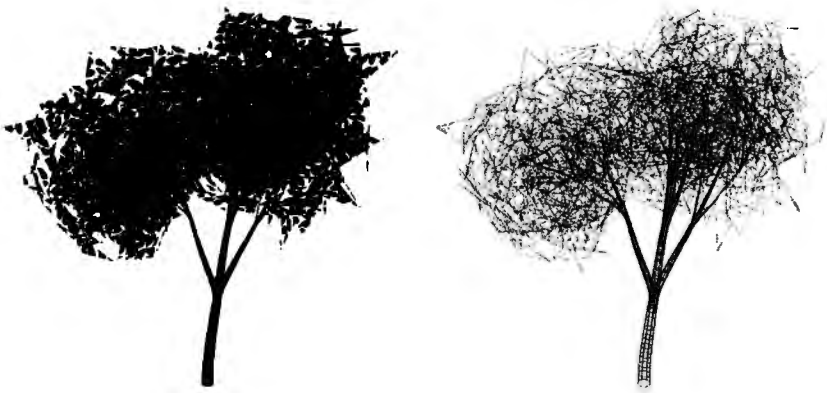
- ☆ El tronco se interpreta como un conjunto de cilindros generalizados conectados entre si que forman una o varias jerarquías.
- ☆ A los extremos de cada cilindro se les conoce como "radios". Dos cilindros generalizados conectados comparten un mismo radio.
- ☆ Cada radio de la jerarquía puede tener el número de radios hijos que desee.
- ☆ Un radio hijo debe tener un solo radio padre.
- ☆ Cada radio esta formado por k número de vértices, donde k debe ser constante para todos los radios que forman el tronco.
- ☆ Debe existir un "radio raíz" para cada jerarquía de cilindros generalizados contenida en el modelo. El radio raíz es aquél cuyos vértices tienen un valor negativo en el eje Y. Debido a que el eje Y intercepta al plano formado por los ejes XZ en su valor cero, el radio raíz es análogo a la raíz de un árbol real que se encuentra por debajo de la tierra.
- ☆ Tiene asociado un material que describe su interacción con la iluminación de la escena.
- ☆ Tiene asociado una textura.

Note que en el contexto de esta tesis un "radio" no es un simple medida sino un nodo de la jerarquía del tronco del árbol que almacena un conjunto de datos. A pesar de que existen varias herramientas comerciales y técnicas de modelado que cumplen con estas restricciones [eki04], durante la investigación fue evidente la necesidad de una herramienta propia que permitiera crear árboles con mayor calidad visual y características particulares. La figura 2.1 muestra un ejemplo de una geometría que cumple con estas restricciones.

2.2. Preprocesamiento del modelo

La aplicación que efectúa el preprocesamiento lee el modelo con las características descritas en la sección anterior.

Figura 2.1: Ejemplo de geometría de un árbol



Smooth shading y textura *Wireframe*

El objetivo de los pasos que se presentan a continuación es recolectar información de como esta construido el árbol, esta información será útil para almacenar los archivos de forma conveniente. Debido a que el preproceso se realiza como una operación independiente a las aplicaciones que hacen uso de los algoritmos de *rendering*, el costo del preprocesamiento en términos de tiempo de cómputo no es un factor importante.

Cada elemento del árbol, el tronco y las hojas, se analiza con distintos algoritmos que se describen a continuación.

2.3. Análisis del tronco

El siguiente pseudo código describe el algoritmo necesario para el análisis del tronco:

Tronco

```
1 Identifico los radios raíces del modelo.
2 Obtengo el número de vértices k que tiene el primer radio raíz.
3 Por cada radio raíz que detecté:
4     Verifico que el número de vértices k sea el mismo.
5     Promedio los vértices y obtengo el centro del radio raíz.
6     Obtengo el diámetro promedio del radio raíz.
7     Inserto los datos del radio en una estructura jerárquica.
8     Llamo a un procedimiento recursivo que identifique los radios hijos.
```

Como se puede apreciar en el pseudo código anterior, un solo modelo puede contener varias jerarquías, cada una con un radio raíz. Esto permite agrupar muchas plantas pequeñas en un solo modelo disminuyendo de esta forma los cálculos al momento de desplegar el objeto. El proceso recursivo que se menciona en la linea 8 es el responsable de construir la jerarquía y realiza las siguientes actividades:

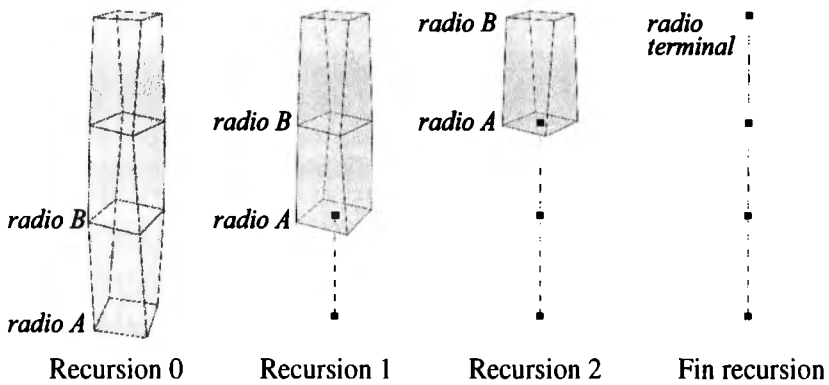
Tronco recursivo

```
1  Dado un radio padre que recibo como parámetro.
2  Busco los vértices vecinos que están conectados a los vértices del radio
3  padre.
4  Si no encuentro ningún vértice vecino:
5      El radio no tiene ninguno hijo y por lo tanto regreso.
6  De otro modo:
7      Extraigo de los vértices vecinos los radios hijos.
8      Por cada radio hijo:
9          Verifico que el número de vértices k sea el mismo.
10         Promedio los vértices para obtener el centro del radio.
11         Obtengo el diámetro promedio del radio.
12         Inserto los datos del radio en una estructura jerárquica.
13         Llamo este proceso recursivamente pasando como padre el radio
14         actual.
```

Durante la primera recursión del pseudo código anterior el radio padre que se menciona en la línea 1 siempre será un radio raíz. Al buscar los vértices vecinos en la línea 2 sólo se toman en cuenta los vértices que forman parte de los radios que no han sido insertados a la estructura jerárquica que se ha construido hasta el momento. La recursión culmina cuando se cumple la condición en la línea 4 puesto que el radio padre ya no tiene ningún vértice vecino, en este caso al radio padre se le denomina "radio terminal".

Una vez que ha terminado el preprocesamiento del tronco obtenemos una estructura jerárquica con los datos de los radios. A los datos de dos radios conectados que forman un cilindro generalizado se han denominado "segmento".

Figura 2.2: Procesamiento de un tronco simple



Dado que el segmento conoce la información de los vértices de los radios que forman el cilindro generalizado puede desplegarse usando triángulos, pero gracias a que también conoce los centros de los radios que forman el cilindro, puede visualizarse como una línea que une estos centros. Esta información será útil para efectuar el nivel de detalle.

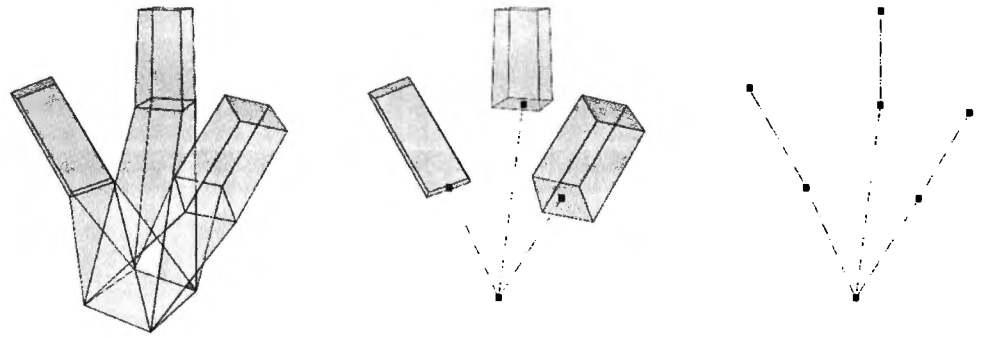
La figura 2.2 muestra una simulación de cómo se efectúa este proceso para una sola jerarquía compuesta por tres cilindros generalizados y cuatro radios. Inicialmente, el algoritmo detecta la presencia de un radio raíz A, formado por cuatro vértices: A0, A1, A2 y A3. Se procesa la información asociada al radio raíz y el proceso recursivo comienza a analizar el resto de la jerarquía. Se identifica un radio hijo B, y se efectúa el análisis necesario; al unir los centros de estos dos radios se puede apreciar una representación en base a líneas. Gracias a la recursión, B ahora será el radio padre y tomará el lugar de A, el proceso recursivo continuará hasta que se detecte un radio terminal, es decir, un radio que no posee ningún hijo.

La figura 2.3 presenta el preprocesamiento de una sola jerarquía formada por seis cilindros generalizados y siete radios. Como se puede observar, a partir del radio raíz se desprenden tres ramificaciones, cada ramificación representa una rama del tronco.

2.4. Análisis de las hojas

El siguiente pseudo código describe el algoritmo necesario para el análisis de las hojas:

Figura 2.3: Procesamiento de un tronco con ramificaciones



Recursion 0 Recursion 1 Fin recursion

Hojas

```
1 Para cada una de las hojas:
2     Promedio sus tres vértices y obtengo su centro.
3     Identifico la distancia del centro de la hoja con respecto a cada uno
4     de los radios terminales del tronco.
5     Asocio la hoja y su centro con el radio terminal más cercano.
```

Al conjunto de hojas asociadas a un radio terminal se ha denominado "grupo de hojas". Los grupos son un mecanismo para agrupar las hojas de forma automática e integrarlas a la jerarquía. Con esta operación termina el preproceso que se efectúa al árbol.

2.5. Estructura jerárquica

En las secciones 2.3 y 2.4 se describe la forma en que los elementos del tronco y de las hojas se integran a una estructura jerárquica, esta jerarquía esta compuesta por los radios que forman el tronco y los grupos de hojas. Cada uno de los radios almacenados en la jerarquía contiene la siguiente información:

1. Centro
2. Diámetro
3. Distancia con respecto al radio padre

4. Apuntador a cada uno de sus hijos en la jerarquía
5. Apuntador a cada uno de sus vértices
6. Vector que une su centro a cada uno de sus vértices

El centro del radio (elemento #1) se puede definir de dos formas: a) como el promedio de sus vértices en caso de ser un radio raíz; o b) por medio de un vector que conecta el centro de su radio padre a su propio centro en caso de ser cualquier otro tipo de radio. Los hijos en la jerarquía (elemento #4) pueden ser algún otro radio de la jerarquía, o en caso de ser un radio terminal puede ser un grupo de hojas o nulo. Esta información se utiliza para efectuar el nivel de detalle y simular el movimiento del árbol, estos algoritmos se describen en los capítulos 4 y 5.

Los grupos de hojas contienen la siguiente información:

1. Centro del grupo
2. Radio del grupo
3. Índice de cada hoja dentro del grupo
4. Centro de cada hoja dentro del grupo

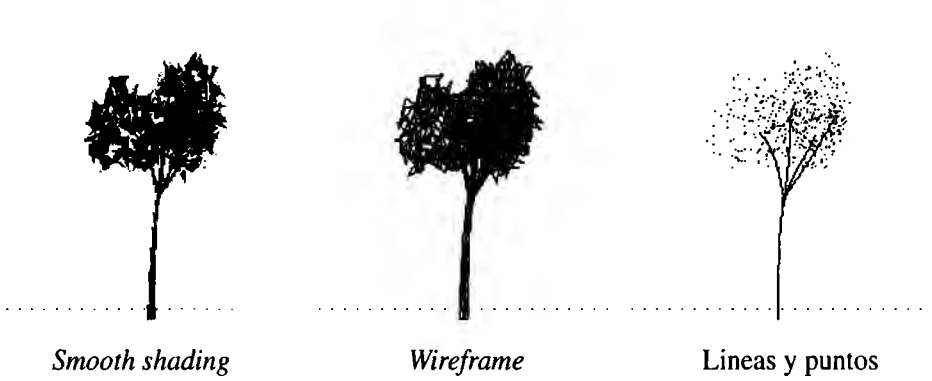
El centro y el radio (elementos #1 y #2) describen un volumen contenedor en forma de esfera (BS — *Bounding Sphere*) que encierra todo el grupo de hojas, es útil para determinar si el grupo de hojas es visible desde el punto de vista de la cámara (ver sección 4.2). Cada hoja es representada por un triángulo, su centro se determina promediando sus tres vértices.

Una vez que el análisis del tronco y hojas ha concluido y se ha generado una estructura jerárquica, se puede extraer una representación basada en puntos y líneas a partir de la representación original basada en polígonos como se muestra en la figura 2.4.

2.6. Conclusiones

Un problema del preproceso actual es que aquellos árboles con muchos radios terminales generan muchos grupos de hojas donde cada uno contiene muy pocas hojas. Esto es ineficiente puesto que el cálculo de nivel de detalle y dinámica de cada grupo de hojas consume cierto tiempo de cómputo.

Figura 2.4: Preprocesamiento de un árbol



Un área de mejora es que el usuario pueda seleccionar el número máximo de grupo de hojas que se desean generar y que el preproceso integre los grupos más pequeños al grupo de hojas más grande y más cercano hasta satisfacer la restricción. Una implementación más elaborada podría organizar los grupos de hojas dentro de una jerarquía de volúmenes contenedores cuya raíz encierra todas las hojas del árbol, de esta forma se podría ajustar el número de validaciones eligiendo un nivel distinto de la jerarquía y agilizar los cálculos de oclusión, LOD y movimiento. Una idea similar se puede aplicar en el tronco.

Otra área de trabajo a futuro consiste en elaborar una herramienta que permita construir nuestros propios árboles con las características descritas en este capítulo. Haciendo uso de esta herramienta los datos del árbol se organizarían en la forma requerida al momento de su construcción y evitaría el análisis anteriormente descrito.

Capítulo 3

Ecosistemas

Debido a que un árbol aislado tiene poco uso en aplicaciones prácticas, fue necesario crear una herramienta que permitiera especificar territorios poblados con árboles y plantas. Esta herramienta permite al usuario modular y ajustar varias de las características del ecosistema virtual como son:

- ☆ Definir cuantos tipos de árboles están presentes en la escena.
- ☆ El número de instancias por cada tipo de árbol.
- ☆ Características de cada instancia como posición, material, textura, tamaño y rotación.
- ☆ Especificar un terreno para colocar los árboles.
- ☆ Especificar el índice de movimiento para aquellos árboles sensitivos a las fuerzas del viento.

Una vez que el usuario ha diseñado su ecosistema, la herramienta genera un archivo que almacena esta descripción, este archivo será utilizado posteriormente por la aplicación de tiempo real que simula un paseo por este ecosistema virtual. A continuación se presenta una descripción de cada una de las partes de esta herramienta, su forma de uso y los algoritmos utilizados.

3.1. Materiales

Los materiales describen la interacción de un objeto con la iluminación de la escena. La definición de los materiales se basa en el modelo de iluminación de Blinn-Phong [SSC01]. Este modelo de iluminación utiliza cuatro valores que son la luz especular, de ambiente, difusa y de emisión.

Además se puede establecer un valor de brillantez (*shiness*) que afecta la forma en que se refleja la luz especular, un factor de transparencia (*alpha blending*) y un índice de textura.

El índice de textura hace referencia a una de las texturas que carga la aplicación de tiempo real que lee el archivo que contiene la definición del ecosistema. Una vez que el usuario ha establecido las características del material, puede asociar este material con el terreno, el tronco o las hojas de alguna instancia de un árbol. De esta forma un solo material puede utilizarse en varios elementos de la escena.

3.2. Instancias de árboles

Con el fin de economizar recursos de cómputo y poder desplegar grandes ecosistemas virtuales, la descripción del ecosistema hace uso de instancias de árboles, es decir, la geometría de un árbol se carga una sola vez en memoria pero se despliegan multiples instancias de este árbol sobre el terreno. El usuario puede determinar cuantos tipos de árboles distintos pretende utilizar y cuantas instancias de cada uno de estos árboles estarán presentes en el terreno.

Para dar la impresión de que las instancias de un mismo árbol son distintas entre si existen varios parámetros que el usuario puede configurar. La configuración se puede efectuar de dos formas: a) usando los parámetros globales que afectan a todas las instancias del árbol en cuestión; o b) especificando los valores de los parámetros de forma manual para cada una de las instancias. La mayor parte de los parámetros globales están basados en procedimientos que generan valores aleatorios y son muy útiles cuando el ecosistema contiene miles de instancias y es difícil especificar un valor de forma manual para cada una de ellas. A continuación se describe cada uno de estos parámetros.

3.2.1. Rotación

Existen 3 parámetros para configurar la rotación de una instancia de forma manual, estos parámetros son *head, pitch* y *roll*. El parámetro *head* determina la rotación de una instancia con respecto al eje Y, el parámetro *pitch* determina la rotación con respecto al eje X y por último el parámetro *roll* determina la rotación con respecto al eje Z. Los tres parámetros reciben sus valores en términos de

grados de rotación.

La versión global del parámetro *head* asocia un valor aleatorio de rotación en el eje Y para cada una de las instancias, el valor puede ir desde cero hasta el número máximo de grados especificados por el usuario. La version global de los parámetros *pitch* y *roll* se han agrupado en una sola categoría llamada *tilt*. El *tilt* o inclinación de cada instancia se puede determinar en base al terreno o ingresando el grado máximo de libertad para *pitch* y *roll* de la misma forma que *head*.

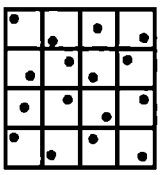
Cuando el usuario selecciona una inclinación en base al terreno, las instancias afectadas tendrán una inclinación perpendicular a la superficie del terreno donde la instancia ha sido plantada (ver sección 3.2.2). Los valores de *pitch* y *roll* se calculan una vez que la superficie del terreno (ver sección 3.3) ha sido transformada a una malla de triángulos, este cálculo se efectúa por medio de un procedimiento que se explica en el siguiente pseudo código.

Inclinación de la instancia

```
1  Identifica el triángulo del terreno que esta por debajo de la instancia.
2  Recupera su vector normal.
3  Copia los valores X y Y del vector normal a  los valores X y Y de un vector
4  bidimensional.
5  Normaliza el vector bidimensional.
6  Si el valor normalizado de X es mayor a cero:
7          Roll será el valor negativo del coseno inverso del valor de Y.
8  En caso contrario:
9          Roll será el valor positivo del coseno inverso del valor de Y.
10 Copia los valores Z y Y del vector normal a  los valores X y Y de un vector
11 bidimensional.
12 Normaliza el vector bidimensional.
13 Si el valor normalizado de X es menor a cero:
14         Pitch será el valor negativo del coseno inverso del valor de Y.
15 En caso contrario:
16         Pitch será el valor negativo del coseno inverso del valor de Y.
```

Para calcular la inclinación con respecto a un eje se utilizan los valores del vector normal del

Figura 3.1: Muestreo basado en *jittering*



triángulo del terreno que son distintos a ese eje (líneas 3 y 10), por ejemplo, dado que *roll* representa la inclinación en el eje Z utilizamos los valores de los ejes X y Y del vector normal (línea 3). Estos valores se copian a un vector bidimensional y se normalizan (líneas 5 y 12). El ángulo de rotación esta dado siempre por la función $\arccos(Y)$ que representa el ángulo que existe entre el vector bidimensional y el vector (0,1). Finalmente la dirección del ángulo depende del signo del valor X que corresponde al valor X y Z del vector normal para *roll* y *pitch* respectivamente.

3.2.2. Escala y posición

La escala en su versión global consiste en un rango donde el usuario establece los valores máximo y mínimo, la escala de cada instancia de este tipo de árbol será algún valor aleatorio contenido en el rango.

Con respecto a la posición de las instancias, su versión global utiliza valores aleatorios siguiendo una técnica llamada *jittering* [Ben03b] que frecuentemente se usa para efectuar un supermuestreo para el *antialiasing* de imágenes. La figura 3.1 ejemplifica un muestreo de 4 por 4 dentro de un *super-pixel*. Cada región del *super-pixel* se muestrea en un lugar aleatorio, posteriormente los valores de las muestras se promedian y producen el valor del *pixel* resultante para una version reducida de la imagen.

El posicionamiento de los árboles sobre el terreno se determina de forma similar haciendo uso del siguiente algoritmo:

Posición de las instancias

```
1 Busca el valor entero más chico que al multiplicarse por si mismo sea igual
2 o mayor que el número total de instancias en el ecosistema.
3 Subdivide el terreno en casillas, el número de casillas será el valor entero
4 encontrado al cuadrado.
```

```
5  Posiciona la casilla actual en la primera casilla del terreno.
6  Para cada tipo de árbol:
7          Encuentra la distancia ideal entre instancias dividiendo el número
8          de casillas entre el número de instancias de este tipo.
9          Para cada instancia de este tipo:
10                 Mueve la casilla actual una distancia ideal entre casillas
11                 mas/menos un margen aleatorio.
12                 Si la casilla actual esta libre:
13                         Posiciona la instancia actual en esta casilla.
14                 En caso contrario:
15                         Mueve la casilla actual hasta encontrar la primera
16                         casilla libre.
17                         Posiciona la instancia actual en esta casilla.
```

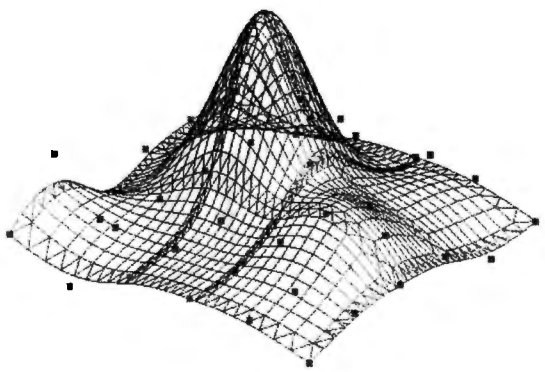
En la línea 10 puede suceder que la casilla actual supere el número de casillas disponibles, en este caso las instancias restantes se posicionan en alguna casilla libre de forma aleatoria. Una vez que se ha posicionado la instancia en la casilla (líneas 13 y 17) ésta queda ocupada impidiendo que otra instancia se pueda posicionar en la misma casilla. Finalmente las instancias establecen sus coordenadas X y Z dentro del rango del terreno que ocupa la casilla, dejando un margen para evitar que dos instancias ubicadas en dos casillas contiguas choquen. La altura de la instancia se determina a partir del triángulo del terreno sobre el cual esta posicionada.

3.2.3. Índice de materiales y movimiento

En el caso de los materiales existe un índice para el material de los troncos y otro para el de las hojas. Estos índices hacen referencia a los materiales que se definieron en el ecosistema para que pudieran ser utilizados por cualquier elemento, ver sección 3.1.

El índice de movimiento hace referencia a algún tipo de movimiento definido en la aplicación que leerá este ecosistema, cada tipo de movimiento tiene su propia configuración de resortes y

Figura 3.2: Superficie NURBS



reglas necesarias para modificar la geometría de la instancia (ver capítulo 5). El movimiento es otra forma de cambiar la apariencia de dos instancias pertenecientes al mismo tipo de árbol.

3.3. Terreno

La generación de los terrenos se efectúa con el siguiente proceso:

1. El usuario define la forma inicial del terreno por medio de puntos de control de una superficie basada en NURBS (*Non Uniform Rational B-Spline*) [Hil00].
2. La superficie se muestrea para obtener los vértices del terreno.
3. Los vértices se procesan utilizando una técnica llamada *random faults* [Ben03a] que le da al terreno una apariencia más natural.

Este módulo de la aplicación fue desarrollado con la finalidad de proveer al usuario control sobre la forma y condiciones del terreno. NURBS es una técnica para modelar curvas y superficies. La forma de la curva o superficie se establece por medio de puntos de control que son regiones que el usuario puede desplazar. Dependiendo del tipo de implementación, la porción de la curva o superficie se aproxima al punto de control o se mantiene pegada a él. La figura 3.2 muestra una superficie que cambia su forma al mover los puntos de control.

Random faults es una técnica basada en la generación de números aleatorios y consiste en generar una diferencia de alturas entre dos porciones de la superficie de manera que al repetir varias

Figura 3.3: Ejemplo de *random faults*

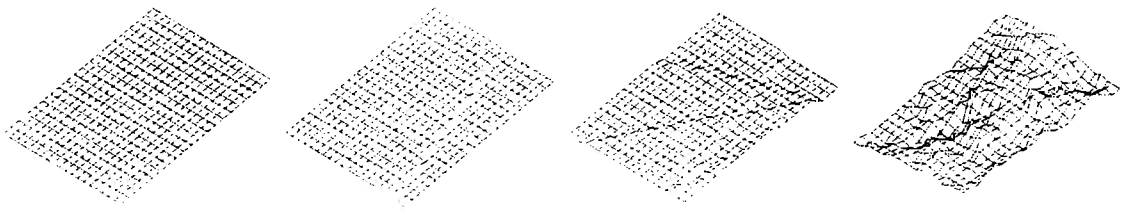
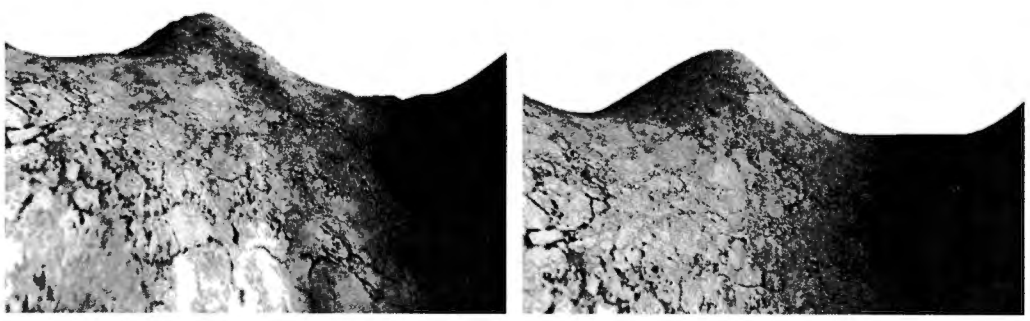


Figura 3.4: Ejemplo de terreno resultante



NURBS y *random faults* Sólo NURBS

veces este proceso se genere un terreno. La figura 3.3 muestra distintas iteraciones del algoritmo aplicado a una pequeña superficie.

Una vez que el usuario ha definido la forma del terreno por medio del módulo de NURBS (ver sección 3.3) puede darle una apariencia más natural a la superficie por medio de esta técnica. En la figura 3.4 se puede observar un terreno a partir de una superficie basada en NURBS, la imagen izquierda ha sido modificada por la técnica de *random faults*, podemos observar que tiene una apariencia más natural que la imagen derecha.

3.4. Conclusión

Se pueden emplear muchas técnicas para mejorar la definición de ecosistemas descrita en este capítulo y generar paisajes más interesantes. Los ecosistemas que se generan con las técnicas descritas proveen sólo las características necesarias para evaluar los algoritmos de *rendering* que se presentan en los capítulos posteriores.

En particular una técnica simple que podría mejorar ampliamente el control que tiene el diseñador sobre el ecosistema resultante, es el uso de imágenes que permitan definir características como escala y color de cada una de las instancias. Una imagen también podría utilizarse para indicar la probabilidad de que una instancia se pueda posicionar en alguna región del terreno y de esta forma definir áreas pobladas y áreas libres de vegetación.

Capítulo 4

Nivel de Detalle

El nivel de detalle en su forma más básica se puede describir como la substitución de una geometría compleja por otra más simple con una apariencia similar, esta técnica permite economizar los recursos de cómputo necesarios para el *rendering* de una escena.

El nivel de detalle puede ser de dos tipos:

LOD discreto:

- ☆ Conjunto de n representaciones distintas del mismo objeto donde cada una de ellas tiene menos detalle que la representación anterior.
- ☆ Todas las representaciones se almacenan como una sola entidad.
- ☆ La representación que se despliega será seleccionada en base al impacto visual del objeto en pantalla.
- ☆ Su principal desventaja es el factor de *popping*. Este factor es el efecto visual que se produce al cambiar de una representación a otra, debido a que produce molestias en el usuario es un efecto indeseable.
- ☆ La cantidad de memoria que se requiere para almacenar todas las representaciones puede ser grande.

LOD continuo:

- ☆ Se considera un solo objeto con una sola representación inicial y algún tipo de información extra.

- ☆ El objeto se transforma en tiempo real para que su complejidad sea proporcional al impacto visual que tiene en pantalla.
- ☆ Su desventaja es que el cálculo para generar la transformación puede ser muy costoso.

El resultado de esta investigación es un nivel de detalle semi continuo, dependiente de la vista y no recursivo. La implementación de este nivel de detalle hace uso de archivos que contienen información de cada árbol presente en el ecosistema, estos archivos se generan al finalizar el preproceso (ver capítulo 2).

A continuación se presenta una descripción del estado del arte, posteriormente se analizan detalladamente los archivos de árboles, se muestran las técnicas para la oclusión en base a la visibilidad de la cámara, los algoritmos necesarios para efectuar el LOD y por último se presentan los resultados y las conclusiones de este capítulo.

4.1. Estado del arte

Marshall *et al.* [MFC97] usaron impostores jerárquicos en forma de tetraedro para simplificar los ecosistemas de plantas virtuales. Basados en la suposición de que una planta puede ser aproximada por un tetraedro con la imagen de la planta proyectada en sus caras. Cuando el objeto se ve desde una distancia lejana, texturas transparentes de la planta son proyectadas en las caras del tetraedro creando impostores. Se recorre la representación jerárquica de la escena y si un grupo esta lo suficientemente lejos, los impostores se despliegan. El método es interesante pero tiene dos desventajas importantes: a) la cantidad de impostores requeridos es enorme y difícilmente puede mantenerse en la memoria de textura; y b) desplegar los impostores y las plantas desde lados distintos puede consumir un tiempo considerable.

Deussen *et al.* [DHL*98] se enfocaron en el problema de desplegar grandes ecosistemas de plantas virtuales por medio de instancias y agrupación de plantas. Cada grupo de plantas tiene un solo representante que es desplegado, a pesar de que esta técnica no incrementa la velocidad en el *rendering*, permite mantener todo el ecosistema en memoria.

Stamminger y Drettakis describieron una técnica de modelado y *rendering* basada en puntos

para objetos procedurales en [SD01]. En el caso de geometría compleja, árboles y plantas, inicialmente se efectúa un muestreo de la superficie haciendo uso de valores aleatorios, las muestras no tienen ningún orden espacial y están distribuidas en forma relativamente uniforme. En cada cuadro se determina la densidad de puntos necesaria para desplegar el objeto, al variar la densidad de puntos esta técnica es capaz de producir niveles de detalle dependientes de la vista y reducir de esta forma la complejidad del árbol. Los puntos se desplazan ligeramente en la dirección del viento para simular los efectos de brisa.

Deussen *et al.* [DCSD02] extendieron la técnica de rendering basada en puntos a un rendering basado en puntos y líneas. Inicialmente se tiene la representación exacta y poco a poco la geometría va cambiando, la hojas se substituyen por puntos y las ramas representadas por cilindros generalizados se substituyen por líneas. Un uso sofisticado de las *display lists* de OpenGL permite un nivel de detalle casi continuo. Adicionalmente, esta técnica también divide la escena en secciones organizadas en una jerarquía, esto reduce el tiempo de cómputo requerido cuando la geometría esta muy alejada de la cámara. A pesar de que hace uso de nivel de detalle es necesario procesar todos los objetos en cada cuadro, por este motivo esta técnica no resuelve el problema de visualización de grandes ecosistemas, simplemente permite mostrar escenas mas largas.

Un proceso de generación de nivel de detalle automático para plantas individuales fue presentado por Lluch *et al.* [LCV03]. Una representación formal de la planta por medio de sistemas de Lindenmayer es enriquecido con información de distintas resoluciones que permite la extracción de distintos niveles de detalle. La idea consiste en que las ramas de un nivel mas alto, es decir las mas alejadas de la raíz, son mas pequeñas y por lo tanto visualmente menos importantes.

4.2. Análisis de datos y generación de archivos

Estos archivos se generan al finalizar el preprocesamiento. Durante el proceso de generación del archivo se realizan ciertas actividades que serán de utilidad al momento de efectuar el nivel de detalle. Estas actividades son:

- ☆ Los segmentos (ver sección 2.3) se organizan en orden ascendente en base a la longitud mayor de los dos radios que lo componen.

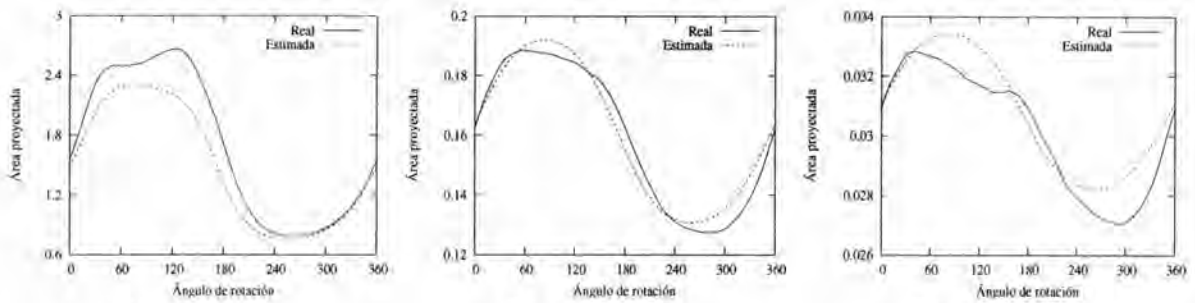
- ☆ Se generan los volúmenes contenedores (BV — *Bounding Volumes*). La estructura jerárquica del árbol que se formó durante el preproceso se encierra dentro de distintos volúmenes contenedores, que son útiles para efectuar la oclusión de aquellos elementos no visibles al momento de desplegar el ecosistema. Los volúmenes contenedores que se generan son:
  - ☆ Un contenedor en forma de caja ajustada a los ejes (AABB — *Axis Aligned Bounding Box*) que encierra todo el árbol incluyendo tronco y hojas.
  - ☆ Un contenedor en forma de caja ajustada a los ejes que encierra únicamente el tronco.
  - ☆ Un conjunto de contenedores en forma de esferas (BS — *Bounding Sphere*) que encierran cada uno de los grupos de hojas (ver sección 2.4).
- ☆ Se calcula el área promedio de los triángulos que forman las hojas.
- ☆ Se calcula el área promedio de los triángulos que forman el tronco.
- ☆ Se calcula el centro de densidad de área del tronco.

A continuación se presentan varias pruebas para verificar que el área promedio de los triángulos que forman las hojas y el tronco estiman de forma adecuada el área proyectada. Estos datos fueron capturados rotando la cámara alrededor de los ejes X y Y y variando la distancia de la cámara al centro del árbol.

En el caso de los triángulos que forman las hojas, la figura 4.1 muestra una comparación del valor estimado contra el valor real del área que proyectan las hojas. El área estimada es la suma del área promedio que proyecta cada grupo de hojas (ver sección 4.4.2). La gráfica izquierda presenta los resultados cuando la distancia es corta y el árbol cubre el 100 % del área de dibujo, en la gráfica central la distancia es media y el árbol cubre aproximadamente el 50 % del área de dibujo, finalmente en la gráfica del extremo derecho la distancia es larga y el árbol cubre sólo el 10 % del área de dibujo. Las gráficas muestran que el área estimada aproxima de forma adecuada el valor del área real, distintas pruebas con distintos movimientos de cámara mostraron siempre un error menor al 15 %.

En el caso del tronco se pensó inicialmente que una buena forma de estimar su área proyectada era detectar qué tan visible era cada una de sus caras desde el punto de vista de la cámara. Dependiendo de su visibilidad, cada cara aportaba un porcentaje de una medición de área en una

Figura 4.1: Área proyectada por las hojas



Distancia corta Distancia media Distancia larga

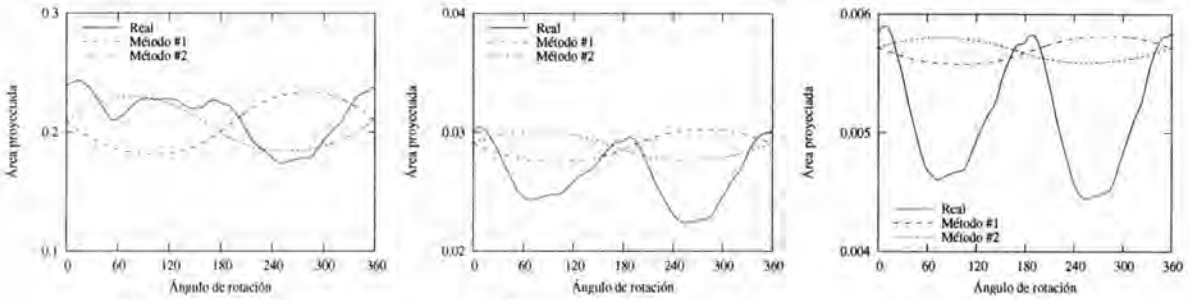
proyección ortogonal que se efectuaba durante un preproceso. Finalmente el área que acumulan las tres caras se proyectaba considerando la profundidad del tronco a partir de su centro. A pesar de que este método suena lógico, mediciones en distintas pruebas mostraron resultados muy poco satisfactorios. Por este motivo se exploraron los siguientes métodos:

1. Se considera el área promedio de cada uno de los triángulos que forman el tronco y el área proyectada se estima considerando la profundidad del tronco a partir de su centro.
2. Se calcula el centro de densidad de área del tronco. Para encontrar este centro se mide que porcentaje de área aporta cada triángulo al área total del tronco, este porcentaje multiplica el centro de cada triángulo y se van acumulando los resultados, la acumulación de resultados es el centro de densidad de área del tronco. Este cálculo simplemente localiza un punto cercano al lugar donde hay mayor cantidad de área en el tronco.

En la figura 4.2 se muestra una comparación del valor estimado por cada uno de estos métodos contra el valor real del área que proyecta el tronco. La posición y distancia de la cámara con respecto al árbol es idéntica a la que se presentó en la figura 4.1. Ambos métodos presentan valores similares pero en términos generales la gráfica del método #2 tiende a ajustarse mejor a la forma de la gráfica del área real. Un comportamiento muy similar se presentó en distintas pruebas con distintos movimientos de cámara.

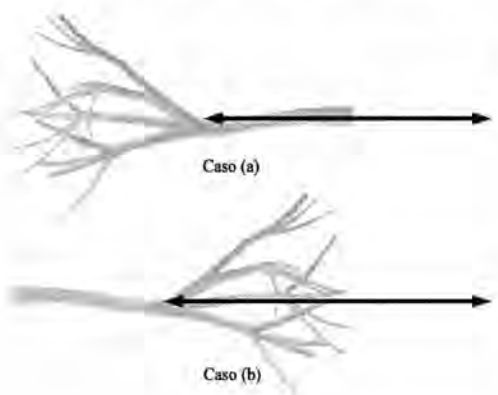
En el método #1 el error de estimación se debe a que se considera el centro del tronco como el punto para determinar la profundidad del tronco, sin embargo el centro del tronco no siempre es el punto donde se concentra la mayor cantidad de área del tronco. La figura 4.3 muestra dos casos

Figura 4.2: Área proyectada por el tronco



Distancia corta Distancia media Distancia larga

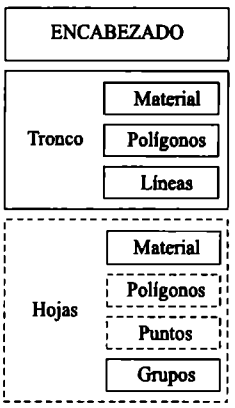
Figura 4.3: Estimación del área del tronco a partir de su centro



típicos donde el método #1 produce un error de estimación, en el caso (a) el valor estimado es mayor al valor real porque a pesar de que la mayor cantidad de área esta mas alejada de la cámara la estimación la considera como si estuviera a la misma profundidad que el centro, en contraste el valor estimado para el caso (b) es menor que el valor real. Al determinar el centro de densidad de área en el método #2 se busca disminuir este error de estimación. En base a las pruebas se seleccionó el método #2 que se utiliza en el nivel de detalle en tiempo crítico (ver capítulo 6).

Las secciones siguientes describen como esta organizada la información dentro del archivo, la figura 4.4 muestra la estructura general del archivo. Esta estructura es importante debido a que la aplicación recupera los datos del árbol en el mismo orden en el fueron almacenados, la eficiencia del algoritmo para LOD depende en gran medida de esta organización.

Figura 4.4: Estructura del archivo



4.2.1. Encabezado

El encabezado contiene la información de dos volúmenes contenedores en forma de cajas alineadas a los ejes que encierran el árbol completo y las ramas en su posición de reposo. Con respecto a la geometría del árbol el encabezado almacena el número de triángulos, vértices, vectores normales, coordenadas de textura y líneas que componen el tronco. Como se mencionó anteriormente en esta misma sección también se incluye el centro de densidad de área y el área promedio de los triángulos que forman el tronco.

En el caso de las hojas el encabezado indica cual es el área promedio de las hojas, número de triángulos, vértices y vectores normales así como el número de grupos de hojas.

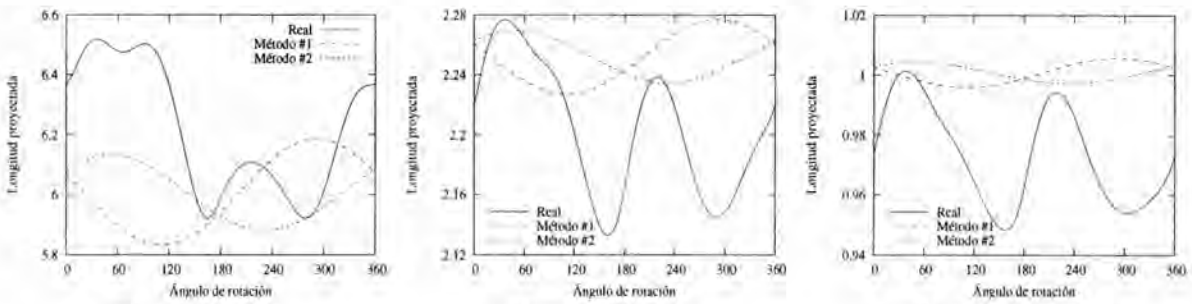
4.2.2. Tronco

El primer elemento del tronco es su material (mas información de materiales en la sección 3.1). Posteriormente se incluye la información de los segmentos, esta información esta organizada en varios arreglos paralelos:

Índices de triángulos: contiene los índices de los vértices, vectores normales y coordenadas de textura que forman cada triángulo de la geometría del tronco.

Índices de líneas: cada línea contiene dos índices que hacen referencia al arreglo de vértices de líneas que se almacena en esta misma sección.

Figura 4.5: Longitud proyectada por el tronco



Distancia corta Distancia media Distancia larga

Diámetro del segmento: el diámetro mayor de los dos radios asociados a cada segmento.

Longitud acumulada por los segmentos: este arreglo es útil para estimar la longitud que proyecta el grupo de segmentos que son dibujados utilizando líneas. Esta estimación consiste en multiplicar la longitud acumulada por los segmentos que se dibujan con líneas por un factor de ajuste que representa el coseno del ángulo que existe entre la línea y la dirección a la que apunta la cámara. En la figura 4.5 podemos apreciar un prueba similar a la que se efectuó en las figuras 4.1 y 4.2, en este caso se compara la longitud real que proyecta el tronco al ser rotado contra la longitud estimada a partir del centro del tronco (método #1) y del centro de densidad de área del tronco (método #2). Nuevamente la mejor estimación se produce al utilizar el centro de densidad del área del tronco. Este arreglo será de utilizada para efectuar el TCLOD en el capítulo 6.

Cada segmento esta definido por un diámetro, dos vértices para la representación en base a líneas y n triángulos para su representación sin nivel de detalle, para encontrar el valor de n basta con dividir el número de triángulos del tronco entre el número de líneas. Estos arreglos paralelos han sido ordenados basándose en el valor del diámetro del segmento, comenzando desde el más delgado y terminando con el mas grueso.

Esta sección también incluye los arreglos de vértices para triángulos y líneas que forman el tronco. Por último se almacena el arreglo de vectores normales y coordenadas de textura a los cuales hace referencia el arreglo de índices de triángulos.

4.2.3. Hojas

El primer elemento de las hojas es su material (mas información de materiales en la sección 3.1). Posteriormente se encuentran dos arreglos paralelos:

Índices de triángulos: contiene los índices de los vértices y vectores normales que forman cada triángulo que representa una hoja.

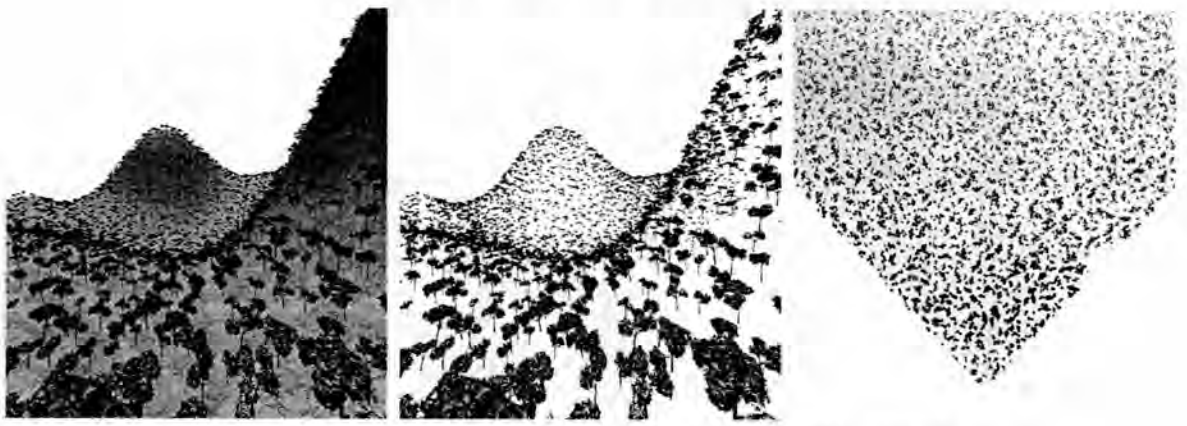
Puntos: centro de cada hoja.

Esta sección también contiene los arreglos de vértices y vectores normales a los cuales hace referencia el arreglo de índices de triángulos. Los grupos de hojas almacenados en esta sección contienen el centro y radio de su volumen contenedor en forma de esfera (ver sección 2.5) y el número de hojas que integran el grupo. Finalmente se encuentran los índices de cada hoja dentro del grupo que hacen referencia al arreglo de índices de triángulos descrito anteriormente, estos índices han sido ordenados basándose en la distancia del centro del grupo al centro de cada hoja, comenzando con las hojas mas cercanas y terminando con las mas alejadas.

4.3. Visibilidad

Con el fin de optimizar los recursos, la aplicación implementa oclusión de objetos en base a la visibilidad de la cámara, en otras palabras, aquellos objetos que estén fuera del alcance visual de la cámara no son desplegados. En la figura 4.6 podemos observar en la imagen un terreno poblado por árboles desde el punto de vista de la cámara, la imagen central muestra la misma vista sin terreno y por último la imagen derecha muestra la vista superior de la misma escena. En la vista superior podemos observar que sólo aquellos árboles visibles desde el punto de vista de la cámara son desplegados. Esta característica se obtiene haciendo uso de una jerarquía de contenedores en forma de cajas que encierran toda escena y de los volúmenes contenedores almacenados en los archivos de árboles.

Figura 4.6: Ejemplo de oclusión



Triángulos: 42814 Líneas: 6573 Puntos: 231122

4.3.1. Jerarquía de cajas

Al cargar la escena, se construye una jerarquía de contenedores en forma de cajas que encierran las porciones de la escena pobladas por árboles. La escena completa se encierra en una gran caja, dentro de esta caja habitan cajas mas pequeñas que a su vez están formadas por otras cajas y así consecutivamente hasta alcanzar la profundidad máxima de la jerarquía.

El usuario puede seleccionar la profundidad máxima de la jerarquía, si la profundidad es muy grande para el tipo de escena, las hojas de la jerarquía en términos de estructura de datos pueden llegar a incluir un solo árbol o estar vacías. Si la profundidad es muy pequeña para el tipo de escena, las hojas de la jerarquía en términos de estructura de datos seguramente contendrán muchos árboles.

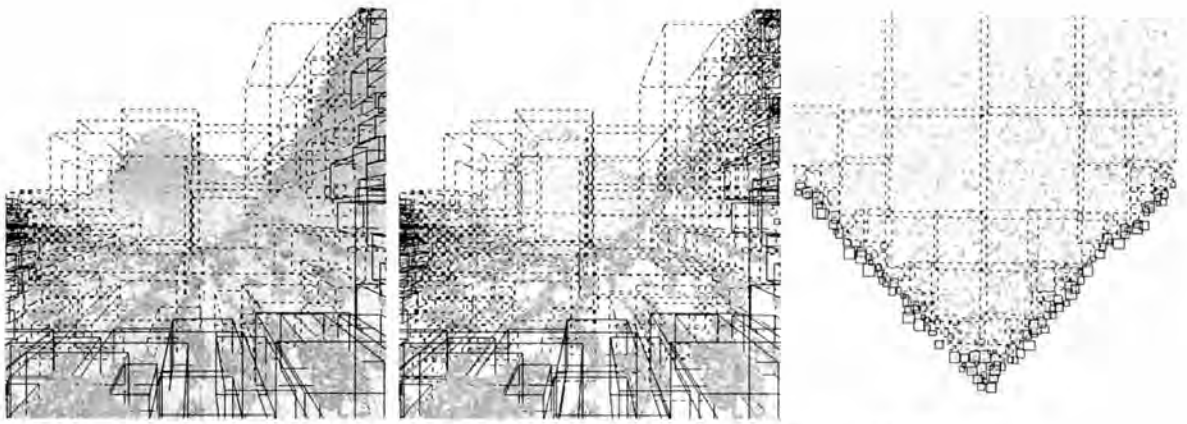
Durante la navegación interactiva, antes de desplegar cada cuadro de la animación, se recorre esta jerarquía a partir de la raíz, cuando la aplicación determina que una caja esta completamente dentro de la visibilidad de la cámara [Ebe00] detiene la recursión y todas las instancias de árboles que habitan dentro de esta caja se despliegan sin ninguna otra validación de visibilidad.

Para aquellas cajas que están parcialmente dentro de la vista de la cámara la recursión viaja hasta las hojas de la jerarquía, a las instancias de árboles que habitan dentro de estas cajas se les efectúa una revisión de visibilidad para detectar que porciones del árbol son visibles y que porciones son invisibles desde el punto de vista de la cámara. El siguiente pseudo código resume este proceso:

Rendering de las instancias

```
Para cada hoja de la jerarquía de cajas:
```

Figura 4.7: Jerarquía de cajas



Triángulos: 42814 Líneas: 6573 Puntos: 231122

```
    Si la hoja es total o parcialmente visible:
        Para cada instancia que habita en esta hoja:
            Si la hoja es totalmente visible:
                Despliega la instancia sin validación de visibilidad.
            De otro modo, si la hoja es parcialmente visible:
                Despliega la instancia con validación de visibilidad.
```

La figura 4.7 muestra la misma escena que la figura 4.6 pero en este caso se puede visualizar la jerarquía de cajas. Las cajas dibujadas con líneas punteadas son aquellas que esta completamente dentro de la vista de la cámara, por otro lado las cajas que esta dibujadas con una línea solida están parcialmente visibles desde el punto de vista de la cámara.

El siguiente pseudo código describe como se construye la jerarquía de cajas al momento de inicializar la escena:

Construcción de la jerarquía

```
Se construye una jerarquía de cajas con la profundidad requerida.
La escena completa se divide entre el número de hojas en la jerarquía.
A cada hoja de la jerarquía le pertenece una porción de la escena.
Para cada instancia en la escena:
    Se busca la hoja dueña de la porción de la escena donde la instancia
```

```
6       fue plantada.
7       Si la hoja no ha sido inicializada:
8           Se inicializa la hoja con una caja alineada a los ejes que encierre
9           a la instancia actual.
10      De otro modo, si la hoja ya esta inicializada:
11          Se expande el volumen actual de la hoja  para que encierre también
12          a la instancia actual.
13      Se le indica a la instancia actual la hoja de la jerarquía que la
14      contendrá.
15  Se recorre la jerarquía de forma recursiva desde las hojas hasta la raíz.
16      Si el nodo no es una hoja:
17          Se inicializa con una caja que encierre a sus descendientes.
```

Al construirse la jerarquía en la línea 1 del pseudo código anterior, ningún nodo ha sido inicializado, la inicialización se presenta en la línea 8 al momento de revisar las instancias de la escena. En la línea 13 se le indica a la instancia cual es la hoja que la contendrá, esto es útil en el proceso de *rendering* para saber cuales instancias habitan en que hoja. Al terminar la revisión de las instancias en la línea 15 se identifica si hay alguna hoja que no se inicializó y se marca como inválida, lo mismo le sucede al nodo en la línea 17 si se determina que todos sus descendientes son inválidos, esta marca es útil para evitar revisiones al momento de determinar las regiones visibles de la escena.

Una vez inicializada la escena, la visibilidad se determina haciendo uso del siguiente algoritmo:

Visibilidad

```
1   Las hojas de la jerarquía se marcan como no visibles.
2   Se recorre la jerarquía desde la raíz.
3       Se verifica la visibilidad del nodo actual.
4           Si no es visible:
5               Detén la recursión de esta rama y regresa.
6           Si es totalmente visible:
7               Si el nodo es una hoja:
```

```
8                   Se marca como totalmente visible.
9               De otro modo:
10                  Todas las hojas descendientes de este nodo se marcan como
11                  totalmente visibles.
12              Detén la recursión de esta rama y regresa.
13          De otro modo, si es parcialmente visible:
14              Si el nodo es una hoja:
15                  Se marca como parcialmente visible.
16                  Se detiene la recursión de esta rama y regresa.
17              De otro modo:
18                  Continúa la recursión con los descendientes de este nodo.
```

Inicialmente el la línea 1, suponemos que ninguna porción de la escena es visible, durante la recursión se determinarán las porciones visibles. En la línea 3 sólo se hace la verificación para aquellos nodos que son válidos, es decir, que encierran al menos una instancia. La línea 10 muestra un proceso eficiente puesto que al determinar un nodo totalmente visible, todas sus hojas descendientes se marcan como totalmente visibles y no tenemos que continuar en la recursión, este caso se muestra en la figura 4.7 como grandes cajas ubicadas en la parte mas ancha de la pirámide de visibilidad de la cámara (*frustum*). En el caso de nodos parcialmente visibles (línea 13) no se puede efectuar la operación anterior puesto que algunos de sus descendientes pueden ser totales, parciales o no visibles.

4.3.2. Visibilidad de las instancias

Aquellas instancias que están contenidas dentro de una caja de la jerarquía que es parcialmente visible desde el punto de vista de la cámara requieren una revisión extra de visibilidad para detectar cuales son las porciones visibles del árbol. A continuación se presenta el pseudo código que efectúa esta revisión:

Visibilidad del árbol

```
1 Si la caja que encierra el árbol completo no es visible:
2         Regresa falso.
3 Si la caja que encierra al tronco es visible:
4         Dibuja el tronco usando LOD.
5 Para cada uno de los grupos de hojas:
6         Si la esfera que encierra al grupo es visible:
7                 Dibuja este grupo de hojas utilizando LOD.
```

Las cajas y esferas que se mencionan en las líneas 1, 3 y 6 son los volúmenes contenedores que se almacenaron en el archivo del árbol (ver sección 4.2). Note que el tronco se considera como un solo objeto, si una porción de la caja que lo encierra es visible se dibuja el tronco completo. En el caso de las hojas cada uno de sus grupos valida su visibilidad (línea 6).

4.4. Algoritmo de nivel de detalle

Como se mencionó al inicio de este capítulo, el algoritmo de nivel de detalle propuesto en esta tesis es semi continuo y a grandes rasgos trabaja de la siguiente forma:

- ☆ Dado que conocemos el centro de los dos radios que forman cada segmento del tronco y el diámetro del segmento que es equivalente a la longitud mayor de sus dos radios. Cuando el segmento es muy delgado en términos de la longitud proyectada en pantalla, lo podemos representar con una sola línea que es una primitiva mucho más económica en términos de recursos de cómputo.
- ☆ Dado que tenemos un punto representativo por cada hoja y el área promedio de las hojas dentro del grupo. Podemos sustituir las hojas por puntos cuando el área promedio sea muy pequeña desde el punto de vista de la cámara. Nuevamente los puntos son una primitiva más económica que los triángulos en términos de recursos de cómputo.

4.4.1. Tronco

El algoritmo de nivel de detalle del tronco hace uso de varios valores, la mayoría de los cuales pueden precalcularse. El primer valor es la longitud que proyecta un píxel (p) sobre el plano mas cercano del *frustum* de la cámara. Si el área de dibujo (*viewport*) es cuadrada este valor se puede calcular dividiendo en ancho del plano cercano entre el ancho de la resolución del área de dibujo.

El algoritmo esta diseñado para que el programador pueda establecer el ancho máximo de línea que puede utilizarse para el nivel de detalle. Entre mayor sea el ancho máximo de la línea el proceso de *rendering* será mas rápido pero la calidad de la imagen será menor. Para determinar el ancho de la línea (w) con la que se dibujará cierto segmento se han definido rangos de validez para cada ancho, estos rangos se expresan en términos de longitud proyectada del ancho sobre el plano cercano del *frustum* de la cámara. El limite inferior (w^-) y el limite superior (w^+) del rango para un cierto ancho se pueden calcular de la siguiente forma:

$$w^- = (w - \frac{1}{2}) * p \tag{4.1}$$

$$w^+ = (w + \frac{1}{2}) * p \tag{4.2}$$

A continuación se presenta el pseudo código del algoritmo utilizado para efectuar el nivel de detalle del tronco:

LOD del tronco

```
Calcula el factor de reducción.
Si existe al menos un segmento que se pueda dibujar con líneas:
    Descarta los segmentos que son muy delgados.
    Para cada ancho de línea disponible:
        Dibuja con líneas todos los segmentos que se encuentren dentro del
        rango válido para el ancho actual.
    Si aún quedan segmentos sin dibujar:
        Dibuja el resto de los segmentos como triángulos.
De otro modo:
```

```
10    Llama al display list del tronco.
```

El factor de reducción que se menciona en la línea 1 se calcula dividiendo la escala de la instancia entre la profundidad de la misma. La profundidad de la instancia con respecto a la cámara se calcula multiplicando el centro de la caja que encierra al tronco por la matriz de modelo-vista de la instancia. El valor negativo del componente Z del vector resultante será la profundidad de la instancia.

Para verificar si existe al menos un segmento que se pueda dibujar con líneas (línea 2) es necesario calcular la longitud que proyecta el diámetro del segmento (d) mas delgado del tronco y comparar este valor con el límite superior (w^+) (ver ecuación 4.2) del rango de validez para el ancho máximo de línea disponible, en caso de que la longitud sea menor la condición de la línea 2 se satisface, en el caso contrario el algoritmo continúa en la línea 9. En otras palabras, si el diámetro del segmento mas delgado es menor que el ancho máximo de línea entonces sabemos que al menos un segmento se puede dibujar con líneas.

La longitud que proyecta el diámetro de cualquier segmento (d) se puede calcular multiplicando el diámetro del segmento por el factor de reducción en los casos donde la distancia entre el plano cercano del *frustum* y la cámara es 1.

Los segmentos que son muy delgados (línea 3) son aquellos que proyectan un diámetro (d) con una longitud menor al límite inferior (w^-) (ver ecuación 4.1) del rango de validez para el ancho mínimo de línea disponible. Por ejemplo, si el ancho mínimo es de 1 píxel, todos aquellos segmentos que proyecten menos de medio píxel serán ignorados.

En la línea 5 se recorren los segmentos y se dibujan con el ancho actual siempre y cuando el segmento proyecte un diámetro con longitud menor al limite superior del rango de validez del ancho de línea actual. Cuando la longitud deja de ser menor se pasa al siguiente ancho de línea, el ciclo que comenzó en la línea 4 termina cuando se rebasa el ancho de línea máximo. Es importante notar que la eficiencia de este algoritmo radica en que los segmentos fueron ordenados basándose en su diámetro como se explica en la sección 4.2.2.

Una vez que ha concluido el ciclo de la línea 4, se verifica si aún quedan segmentos que no se pudieron dibujar con nivel de detalle (línea 7), estos segmentos se dibujan con triángulos como se

indica en la línea 8. Para saber cual es el primer triángulo a dibujar, utilizamos un valor llamado "factor de segmento" que indica cuantos triángulos le corresponden a una sola línea, este valor se calcula dividiendo el número de triángulos del tronco entre el número de líneas. El primer triángulo a dibujar se encuentra en la localidad dada por la multiplicación del número de segmentos dibujados con líneas por el factor de segmento. Es importante notar que esta operación es válida debido a que el arreglo de triángulos y de líneas son paralelos como se explicó en la sección 4.2.2.

4.4.2. Hojas

De la misma forma que en la sección 4.4.1 el nivel de detalle en las hojas requiere varios valores, algunos de los cuales pueden precalcularse. El primer valor es el área que proyecta un punto (a_p) en el plano cercano del *frustum* de la cámara. En los casos donde el área de dibujo es cuadrada este valor se puede calcular basándose en el tamaño del punto (p_s) y el ancho de la resolución del área de dibujo (v) haciendo uso de la siguiente ecuación:

$$a_p = (p_s * v)^2 \tag{4.3}$$

Al cambiar el tamaño del punto el usuario puede ajustar la calidad de la imagen, por ejemplo, si el usuario selecciona un tamaño de punto muy grande, mayor número de triángulos serán representados por puntos y la velocidad de *rendering* aumentará pero la calidad de la imagen disminuirá.

A continuación se presenta el pseudo código del algoritmo utilizado para efectuar el nivel de detalle en las hojas:

LOD en las hojas

```
1 Para cada grupo de hojas visible:
2     Encuentra el área que proyecta el grupo.
3     Determina el número de puntos requeridos.
4     Si el número de puntos requeridos es menor que el número de hojas en el
5     grupo:
6         Dibuja el número de puntos requeridos desde el final del arreglo de
7         puntos.
```

```
8     Si el número de puntos requeridos es menor que dos veces el número de
9     hojas en el grupo:
10        Ajusta el número de puntos requeridos.
11        Dibuja el número de puntos requeridos desde el inicio del arreglo
12        de puntos.
13        Dibuja las hojas restantes como triángulos.
14    En cualquier otro caso:
15        Llama al display list del grupo de hojas.
```

Para encontrar el área que proyecta cada grupo de hojas (a_g) (línea 2) en el plano cercano del *frustum* de la cámara se requiere calcular la profundidad del grupo (z). Al igual que la sección 4.4.1, la profundidad del grupo se calcula transformando su centro con la matriz de modelo-vista y recuperando el valor negativo del componente Z del vector resultante. Para el cálculo del área también se requiere conocer la escala de la instancia (s), el número de hojas en el grupo (n) y el área promedio de cada hoja (a_h). El área proyectada de cada grupo de hojas se puede calcular de la siguiente forma:

$$a_g = \frac{n * a_h * s^2}{1{,}8 * z^2} \tag{4.4}$$

El 1,8 de la ecuación anterior es una constante que se detecto el efectuar varias pruebas, este valor asume que sólo la mitad de las hojas del grupo están viendo directamente a la cámara. En la línea 3 se calcula el número de puntos requeridos (n_p) de la siguiente forma:

$$n_p = \frac{a_g}{a_p} \tag{4.5}$$

En la línea 6 desplegamos el número de puntos requeridos (n_p) comenzando desde el extremo final en dirección al inicio del arreglo de puntos, es decir en sentido inverso. Los puntos ubicados al final del arreglo representan las hojas que se encuentran mas alejadas del radio terminal del tronco asociado al grupo (ver sección 2.4), estas hojas tienen mayor importancia visual debido a que definen el contorno del grupo.

En la línea 10 es necesario ajustar el número de puntos requeridos (n_p) puesto que es mayor que el número de hojas disponibles (n). Este ajuste nos va permitir hacer una transición suave entre la representación basada en triángulos y la representación basada en puntos. El ajuste se efectúa con la siguiente formula:

$$n_p = 2n - n_p \tag{4.6}$$

Una vez que se ha ajustado el número de puntos requeridos (n_p) se despliegan desde el inicio del arreglo de puntos como se indica en la línea 11. La razón es que los puntos ubicados al inicio del arreglo son los que están mas cercanos al radio terminal del tronco asociado al grupo (ver sección 2.4) y por lo tanto los que tienen menor importancia visual. Cuando comenzamos a transformar el grupo de hojas deseamos que la transición sea poco visible y por eso escondemos las hojas dibujadas con puntos detrás de las hojas dibujadas con triángulos (línea 13) que se encuentran en el exterior del grupo.

4.5. Resultados

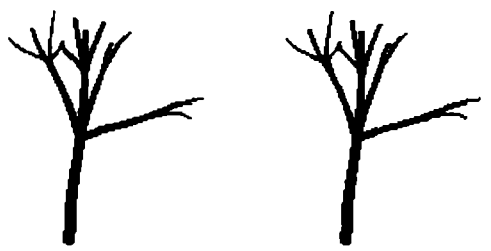
En contraste con el algoritmo de Deussen *et al.* [DCSD02] donde los cilindros que representan el tronco se reorganizan de forma aleatoria, nosotros optamos por ordenar los segmentos en base a su diámetro lo cual nos proporciona dos ventajas que son: a) con una simple comparación podemos descartar aquellos segmentos que por ser tan delgados no vale la pena mostrarlos en pantalla; y b) la ventaja más importante es que podemos utilizar líneas de distintos anchos que se ajusten lo mejor posible al diámetro del segmento.

Estas dos innovaciones permiten un uso más extenso del nivel de detalle basado en líneas y le ofrece al usuario la capacidad de controlar la calidad de la imagen seleccionando el ancho máximo de línea que la aplicación puede utilizar.

Como podemos observar en la figura 4.8 la versión con nivel de detalle logra disminuir el número de triángulos un 82 % haciendo uso de unas cuantas líneas con distintos anchos que preservan con gran similitud la forma original del árbol.

El hecho de que los segmentos estén ordenados en base a su diámetro proporciona otra ventaja

Figura 4.8: Comparación de dos troncos



Triángulos: 560
Líneas: 0

Triángulos: 96
Líneas: 58

que es: en el momento que encontremos un segmento cuyo diámetro sea tan ancho que no se pueda representar con líneas, podemos estar seguros que los segmentos restantes serán igual o más anchos. Es decir, existe una relación monotónica entre el diámetro de los segmentos.

Las ventajas descritas anteriormente permiten que el algoritmo sea muy rápido y eficiente agrupando el despliegue de todas las líneas en una etapa y todos los triángulos en otra.

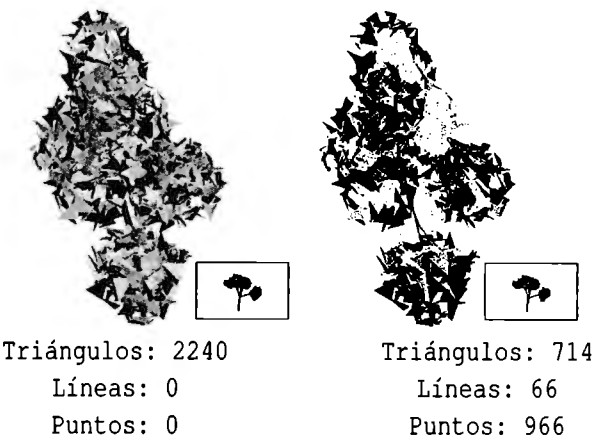
En el caso de las hojas Deussen *et al.* [DCSD02] consideran todas las hojas del árbol como una sola entidad, en contraste nuestra técnica utiliza la jerarquía implícita en la forma del árbol para organizar las hojas en grupos (ver sección 2.4), permitiendo que la transición de cada grupo sea independiente a la de los otros grupos de hojas, por lo tanto el algoritmo puede empezar a cambiar triángulos por puntos tan pronto como sea posible.

La transición es muy suave, porque se realiza en dos etapas: a) la transición que sufre cada uno de los grupos; y b) la transición que se observa al considerar todos los grupos en conjunto.

A diferencia del algoritmo de Deussen *et al.* [DCSD02] donde las hojas se ordenan en forma aleatoria, nosotros optamos por ordenar las hojas dentro de los grupos en base a la distancia que tienen con respecto al radio terminal (ver sección 2.3). Esto permite que las hojas más cercanas al radio terminal se transformen antes en puntos que las hojas más lejanas minimizando de esta forma el impacto visual al momento de utilizar el nivel de detalle.

Cuando se requieren muy pocos puntos para representar el grupo, tienen preferencia aquellos puntos correspondientes a las hojas mas alejadas del radio terminal, lo cual permite preservar el contorno y la forma de las hojas del árbol inclusive cuando proyecta un área muy pequeña.

Figura 4.9: Comportamiento de los grupos de hojas

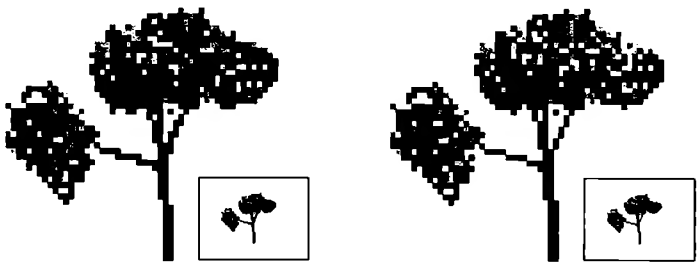


En la figura 4.9 podemos observar la transformación que sufre el árbol desde una vista superior, la apariencia del árbol desde el punto de vista de la cámara se puede observar en el recuadro de cada imagen. Podemos ver que las hojas que sufren primero la transformación a puntos son aquellas que están más cercanas a los radios terminales y que cada grupo tiene una transformación independiente de los otros grupos. Esto permite que la forma del árbol y su follaje basado en puntos y líneas sean muy similares a la versión sin nivel de detalle.

A continuación se presentan ejemplos del uso de esta técnica de nivel de detalle. La figura 4.10 muestra la comparación de un mismo árbol desplegado sin y con nivel de detalle, las imágenes han sido escaladas para apreciar el detalle, el tamaño original desde el punto de vista de la cámara se puede apreciar en el recuadro. Podemos observar que ambas representaciones son muy similares y el contador de triángulos disminuye un 50 % aproximadamente en la versión que utiliza nivel de detalle al hacer uso de unas cuantas líneas y un número considerable de puntos.

Para optimizar aún más los recursos en la aplicación, la porción del árbol que se despliega sin nivel de detalle utiliza *display lists* de OpenGL. En la figura 4.11 se presenta una comparación de la taza de cuadros por segundo y contador de primitivas para una misma escena sin y con nivel de detalle. Para efectuar una comparación justa, ambas versiones optimizan la porción del árbol que se despliega sin nivel de detalle y utilizan oclusión en base a la visión. Como podemos observar ambas imágenes son muy similares pero la taza de cuadros por segundo de la versión que utiliza nivel de detalle es mucho mayor que aquella que no lo hace.

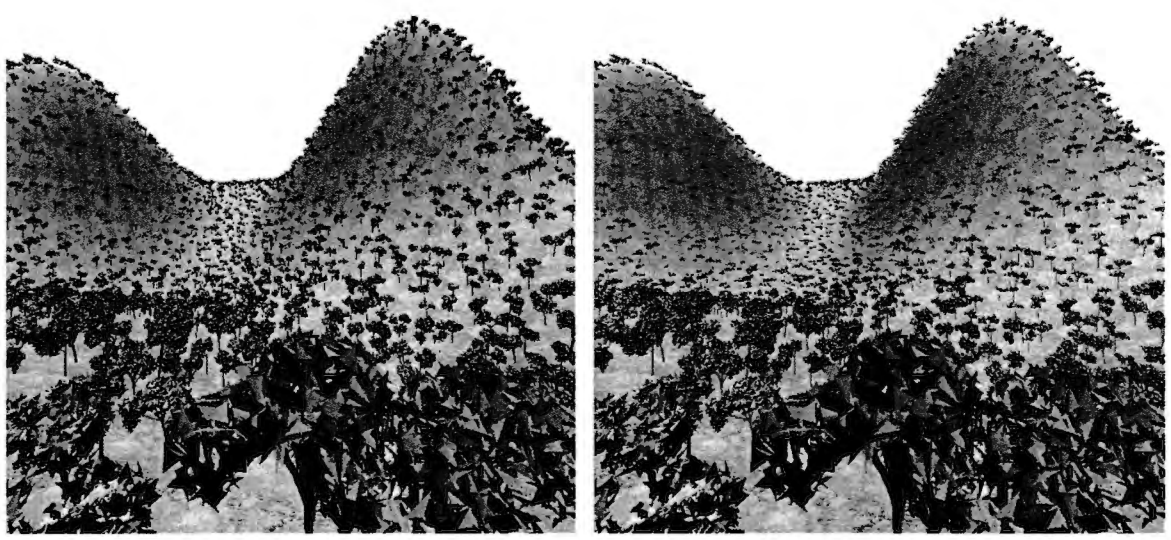
Figura 4.10: Comparación de dos árboles con follaje



Triángulos: 2240
Líneas: 0
Puntos: 0

Triángulos: 1145
Líneas: 66
Puntos: 535

Figura 4.11: Nivel de detalle y cuadros por segundo



Triángulos: 2634236
Líneas: 0
Puntos: 0
FPS: 6

Triángulos: 47044
Líneas: 6609
Puntos: 223554
FPS: 23

4.6. Conclusión

Al efectuar varias pruebas encontramos que aplicar textura en las líneas no proporcionaba ninguna información relevante y en algunos casos lucía extraño. Con respecto a la iluminación, la representación poligonal del tronco tiene muchos vectores normales apuntando en direcciones distintas, es imposible que la línea encuentre un solo vector normal que represente de forma adecuada todos los vectores de la representación poligonal. Por este motivo las líneas se despliegan sin iluminación ni textura haciendo uso de un color simple que se calcula al cargar la escena en memoria basándose en el color promedio de la textura del tronco, el material del polígono y las condiciones de iluminación de la escena.

En el caso de las hojas, cada punto utiliza el vector normal promedio del triángulo que representa. Los puntos se despliegan sin textura pero con propiedades de iluminación lo cual les da una apariencia muy similar a los triángulos. Nuevamente, el color de su material se calcula basándose en el color promedio de la textura de las hojas, el material de los polígonos y las condiciones de iluminación de la escena.

Capítulo 5

Dinámica

Simular la dinámica de árboles en comparación con otro tipo de dinámicas es un problema difícil y poco intuitivo para un diseñador debido a los siguientes motivos:

- ☆ La geometría de un árbol puede ser extremadamente compleja debido a sus múltiples ramificaciones.
- ☆ Se debe cuidar que las extremidades del árbol no sufran deformaciones debido a la dinámica, es decir que las ramas se estiren o se tuerzan de forma no deseable.
- ☆ Cuando aplicamos una fuerza como puede ser el viento, cada parte del árbol reacciona de forma distinta. Esto se debe al espesor de cada rama y el peso que ejercen las hojas sobre las ramas entre muchos otros factores.
- ☆ Puede ser que el árbol o la planta se vea afectado por fuerzas distintas y que cada una de esas fuerzas afecte porciones distintas del árbol, por ejemplo: distintas corrientes marinas que mueven un alga en el agua.

Nuestro objetivo primordial con respecto a la dinámica se enfocó en desarrollar un algoritmo rápido y eficiente en términos de tiempo de cómputo, que permitiera mostrar los efectos de fuerzas físicas como el viento en ecosistemas poblados por varios árboles.

Pensamos que esta técnica puede ser útil para aplicaciones que requieran simular movimientos de árboles en forma interactiva, pero donde los árboles no jueguen el rol principal y el usuario preste más atención a otros eventos que ocurren en la pantalla.

La implementación de este algoritmo nuevamente hace uso de los archivos resultantes del preproceso, estos archivos contienen la información necesaria para efectuar la simulación de movimiento los árboles y por este motivo son distintos a los que se mostraron en el capítulo de nivel de detalle en la sección 4.2.

El algoritmo propuesto mueve cada parte del árbol haciendo uso de resortes que reaccionan a las fuerzas presentes en la escena. Los resortes tienen la ventaja de tener pocos parámetros, ser fáciles y rápidos de calcular y ajustar sus valores es un proceso relativamente intuitivo para el diseñador.

La idea consiste en generar uno o más resortes organizados en su propia estructura de datos y posteriormente asociarlos con la jerarquía de radios (ver sección 2.5) de cada instancia. Cuando los resortes reaccionan a las fuerzas presentes en la escena, los radios intentan moverse en la misma dirección que los resortes y modifican la geometría del árbol, la geometría modificada se puede desplegar haciendo uso de un algoritmo de nivel de detalle similar al que se mostró en la sección 4.4.

A continuación se presenta el estado del arte, posteriormente se describe a detalle el contenido de los archivos, los algoritmos necesarios para efectuar el movimiento, posibles aplicaciones de esta técnica y por ultimo se muestran las conclusiones de este capítulo.

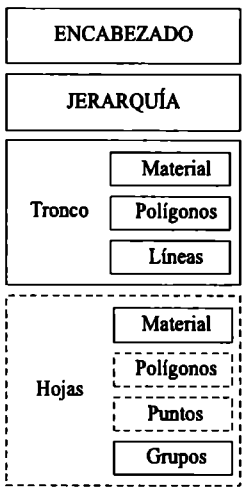
5.1. Estado del Arte

La simulación de dinámica de la naturaleza virtual se puede clasificar en dos tipos: física y procedural. El primer tipo realiza cálculos físicos y considera factores como viento, gravedad, densidad de las ramas, etc. El segundo tipo no es físicamente consistente y generalmente enfatiza más la velocidad del algoritmo, utiliza técnicas como animación basada en *key frames*.

En la categoría de animación basada en física encontramos a Sakaguchi *et al.* [SO99] que propone una innovadora técnica para modelar árboles en base a fotos y permite animar estos modelos considerando varios factores físicos y tratando cada parte del tronco como una rama rígida.

Recientemente Kondo *et al.* [KMOK03] presentaron una técnica donde la animación del árbol puede tomar ventaja de las nuevas capacidades que permiten programar el hardware de las tarjetas de video (GPU — *Graphics Processing Unit*). Este paper también introduce el nivel de detalle en animación evitando el desplazamiento de las hojas cuando el movimiento es insignificante desde el

Figura 5.1: Archivo de árboles con movimiento



punto de vista de la cámara.

En la categoría de animación procedural Fang y Hart [FH02] proponen un método que genera *key frames* de animación muestreando una simulación en una etapa de preprocesamiento, posteriormente utilizan ruido para generar una animación más realista.

Como podemos observar, hay pocos intentos de mezclar técnicas de nivel de detalle con la simulación de dinámica en árboles y plantas. Nuestra propuesta consiste en utilizar el algoritmo de nivel de detalle (ver capítulo 4) para disminuir la complejidad de las instancias de árboles y agilizar los cálculos necesarios para la simulación de movimiento.

5.2. Archivos de árboles con movimiento

La estructura del archivo es importante debido a que los datos se cargan en memoria en el mismo orden en el que se encuentran en el archivo y la eficiencia del algoritmo que simula el movimiento de los árboles depende en gran medida de esta organización. El archivo es similar al que se presentó en la sección 4.2 para efectuar el nivel de detalle, pero tiene varias diferencias de las cuales la más relevante es que en esta ocasión se almacena la estructura jerárquica que se generó durante el preproceso (ver sección 2.5). La figura 5.1 muestra la organización del archivo.

5.2.1. Encabezado

La diferencia básica con respecto al encabezado que se presentó en la sección 4.2.1 es que en este caso se incluyen datos de la jerarquía que son el número de raíces, número de radios y una bandera que indica si existe una relación monotónica entre los diámetros de los radios de la jerarquía, es decir que un radio hijo siempre tiene un diámetro menor o igual que el de su radio padre, esto permite agilizar cálculos y validaciones en el algoritmo que se presenta en la sección 5.3.2. Para determinar el número de vértices que forman un radio basta con dividir el número de vértices del tronco entre el número de radios en la jerarquía.

5.2.2. Jerarquía

La información de la jerarquía es idéntica a la que se generó al final del preproceso (ver sección 2.5) pero ha sido organizada en forma distinta. La estructura de datos que contenía la jerarquía durante el preproceso ha sido transformada en dos arreglos planos:

Jerarquía de radios: cada radio de este arreglo contiene su centro, diámetro y distancia con respecto a su radio padre. En contraste con la jerarquía del preproceso, esta jerarquía de radios ya no almacena apuntadores a sus hijos, en su lugar se han incluido los siguientes elementos:

- ☆ Índice del primer hijo
- ☆ Número de hijos
- ☆ Bandera que indica si es un radio terminal

Para cualquier radio no terminal el número de hijos es forzosamente mayor o igual a uno, en este caso el índice del primer hijo hace referencia a otro radio dentro de este mismo arreglo. Si el radio es terminal el número de hijos puede ser uno o cero, si tiene un hijo significa que el índice de su hijo hace referencia a una localidad en el arreglo de grupos de hojas (ver sección 4.2.3).

Vector a cada vértice: a diferencia del preproceso donde el radio apunta a cada uno de los vértices que lo conforman, el archivo almacena esta relación haciendo uso de este arreglo que es paralelo a la jerarquía de radios. Este arreglo contiene vectores que unen el centro del radio

a cada uno de sus vértices. De esta forma, cuando se carga el archivo en memoria se recorre la jerarquía de radios por primera vez y sumando los centros de los radios al conjunto de vectores contenidos en este arreglo se puede recuperar la posición en reposo de los vértices del tronco. Este arreglo también es de utilidad al momento de efectuar la simulación de movimiento y reescribir la posición de los vértices (ver sección 5.3.2).

5.2.3. Tronco y hojas

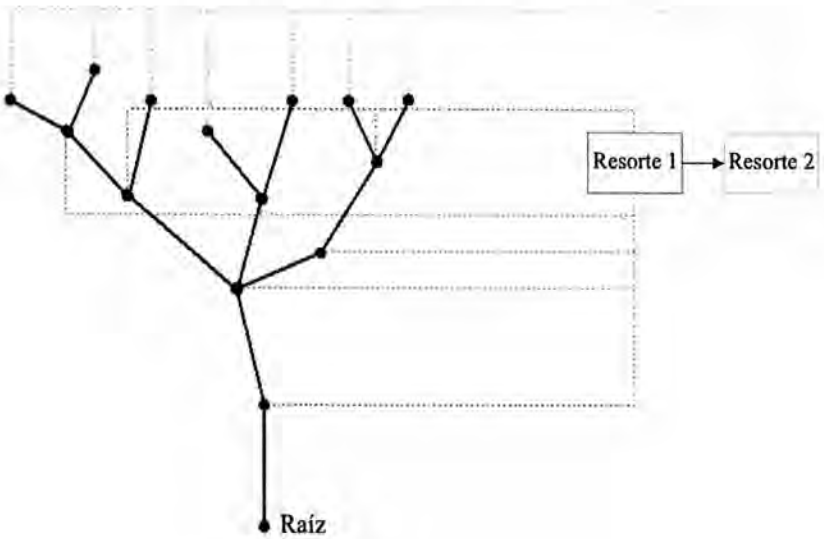
En el caso de los troncos, la diferencia básica con respecto a la información que se mostró en la sección 4.2.2 es que se han eliminado los arreglos de vértices de los triángulos y líneas que forman el tronco. El arreglo de vértices para los triángulos se construye al momento de leer el archivo como se explica al final de la sección 5.2.2. En el caso de las líneas sus índices hacen referencia a la jerarquía de radios, de esta forma los segmentos se pueden representar como una línea que une el centro de dos radios. En el caso de las hojas la información es idéntica a la que se mostró en la sección 4.2.3.

5.3. Algoritmo de movimiento

El movimiento se genera haciendo uso de resortes que han sido organizados en su propia estructura de datos. Al inicializar la escena, la jerarquía de radios de cada instancia se asocia con la estructura de resortes. Un radio se asocia a uno o más resortes cuando cumple ciertas condiciones establecidas por el programador, las condiciones están basadas en varios aspectos de la jerarquía de radios como son:

- ☆ Diámetro del radio
- ☆ Radios terminales (ver sección 2.3)
- ☆ Altura del radio
- ☆ Número de radios consecutivos
- ☆ Nivel que ocupa el radio en la jerarquía

Figura 5.2: Jerarquía de radios y resortes



El movimiento de la instancia se produce cuando los resortes reaccionan a las fuerzas presentes en la escena y los radios intentan moverse en la misma dirección que el resorte asociado. Este algoritmo se puede utilizar para el caso donde existe una fuerza que mueve todas las instancias de la escena en la misma dirección o en el caso donde hay múltiples fuerzas con distintas direcciones actuando simultáneamente. El número y dirección de las fuerzas presentes en la escena determina si muchas instancias comparten una sola estructura de resortes, o existe una estructura de resortes para cada instancia o grupo de instancias.

En la figura 5.2 podemos ver como una jerarquía de radios representada con líneas sólidas que unen pequeños círculos se asocia a una lista de resortes, la asociación se representa con líneas punteadas. En este caso las condiciones establecen que los radios terminales se asocian al resorte dos mientras que cualquier otro radio se asocia al resorte uno. Es importante notar que el radio raíz (ver sección 2.1) nunca se asocia a un resorte porque debe ser fijo para servir como punto de referencia para el movimiento del los radios restantes.

El algoritmo para efectuar esta asociación se describe en el siguiente pseudo código:

Asociación de movimiento

```
1 Para cada instancia:
2     Construye la matriz de transformación que invierte la orientación de la
3     instancia.
```

Figura 5.3: Distinta orientación con movimiento

```
4     Para cada raíz en la jerarquía de radios:
5         Llama el proceso recursivo de asociación pasando la raíz como
6         parámetro.
```

Además de la asociación entre radios y resortes, la relación entre instancia y estructura de resortes también incluye una matriz de transformación que invierte la orientación de la instancia (línea 2), esta matriz permite orientar el movimiento de los resortes en una forma consistente. La figura 5.3 muestra tres instancias del mismo árbol que a pesar de tener una orientación distinta se mueven en la misma dirección haciendo uso de la matriz de transformación.

El proceso recursivo que se menciona en la línea 5 del algoritmo anterior es el siguiente:

Asociación recursiva

```
1  Para cada radio hijo del radio padre que se recibe como parámetro:
2      Para cada resorte en la estructura:
3          Si el radio hijo cumple con la condición del resorte:
4              Asocia el radio hijo con el resorte actual.
5              Caracteriza la asociación.
6              Si ningún ancestro del radio hijo se ha asociado con un
7              resorte:
8                  Marca este radio como el inicio del movimiento.
9      Si el radio hijo tiene descendientes:
10         Llama este proceso recursivo pasando el radio hijo como parámetro.
```

En el algoritmo anterior podemos ver que muchos radios pueden estar asociados al mismo resorte, sin embargo estos radios pueden moverse de forma distinta en base a características que

Figura 5.4: Distinta escala con movimiento

establece el programador, por ejemplo: un radio ancho se ve afectado solamente en un 50 % del movimiento total del resorte mientras que uno más delgado se ve afectado en un 90 %. Estas características se establecen en la línea 5 del proceso recursivo.

Muchas instancias pueden asociarse a la misma estructura de resortes, sin embargo características de cada instancia como su orientación o escala afectan la forma en la que su jerarquía de radios se asocia a la estructura de resortes, por este motivo debe existir una relación entre jerarquía de radios y estructura de resortes por cada instancia. La figura 5.4 muestra a tres instancias del mismo árbol que reaccionan de forma distinta a las fuerzas en la escena debido a que su escala es diferente.

En la línea 8 del proceso recursivo se marcan los primeros radios en la jerarquía que pueden llegar a moverse, cuando se calcula el movimiento de la instancia, en lugar de comenzar la recursión en las raíces originales de la jerarquía se comienza a partir de estos radios que fueron marcados como inicio de movimiento y de esta forma se evitan los cálculos innecesarios de radios que no cumplieron con ninguna condición y por lo tanto no se asociaron a ningún resorte.

5.3.1. Física de resortes

La fuerza que afecta a los resortes se calcula con la siguiente ecuación:

$$\vec{f} = -k * \Delta\vec{x} \tag{5.1}$$

$$\Delta\vec{x} = \vec{x}_c - \vec{x}_i \tag{5.2}$$

La ecuación 5.1 representa la fuerza creada por el resorte cuando su longitud cambia. El cambio de longitud se representa con $\Delta\vec{x}$. Esta fuerza siempre trata de restaurar la posición de inercia del resorte que se representa con $\vec{x}_i$. La posición actual del resorte es representada con $\vec{x}_c$. El valor k es el coeficiente de elasticidad. Si $\vec{x}_i$ es el origen entonces la ecuación 5.1 se puede simplificar a la ecuación 5.3 que se muestra a continuación.

$$\vec{f} = -k * \vec{x}_c \tag{5.3}$$

A pesar de que los resortes han sido organizados en una estructura de datos, el movimiento de un resorte es independiente al del resto. Después de acumular las fuerzas en la escena se obtiene la aceleración $\vec{a}$ integrando numéricamente por el método de Euler [Bou01] la fórmula $\vec{a} = \frac{\vec{F}}{M}$ donde $\vec{F}$ es la fuerza acumulada y M es la masa. Haciendo uso de la aceleración se calcula la nueva velocidad $\vec{v}'$ con la ecuación $\vec{v}' = \vec{v} + \vec{a}t$ donde $\vec{v}$ es la velocidad de la iteración anterior y t es el tiempo. La nueva posición $\vec{x}_c'$ se obtiene con la ecuación $\vec{x}_c' = \vec{x}_c + \vec{v}'t$ donde $\vec{x}_c$ es la posición del resorte en la iteración anterior. Finalmente $\vec{v}'$ se multiplica por un coeficiente de pérdida de energía ℓ en cada iteración. Dado que $\ell < 1$ el resorte siempre regresará a la posición de inercia.

5.3.2. Movimiento de la jerarquía de radios

El algoritmo para efectuar el movimiento de la jerarquía de una instancia una vez que se ha actualizado el estado de los resortes como se vio en la sección 5.3.1 se puede resumir con el siguiente pseudo código:

Movimiento de los radios

```
1 Si existe movimiento significativo:
2     Restaura la forma original de la geometría del árbol.
3     Orienta los resortes acorde a la instancia.
4     Para cada radio marcado como origen del movimiento:
5         Calcula el diámetro que proyecta su radio padre.
6         Llama un proceso recursivo pasando como parámetros la posición,
```

```
7       diámetro proyectado de su radio padre y la referencia al radio que
8       es el origen del movimiento como único hijo.
```

El la línea 1 se determina si existe un movimiento significativo proyectando el cambio de longitud más grande (ΔX) que hayan sufrido los resortes asociados a la instancia, si esta longitud es menor que un umbral se considera que el movimiento es despreciable. Al modificar los valores de este umbral el usuario puede controlar la calidad del movimiento. La proyección se efectúa de la misma forma en que se proyectan los diámetros de los segmentos en el algoritmo de nivel de detalle haciendo uso del factor de reducción (ver sección 4.4.1).

En la línea 2 se elimina cualquier modificación a la geometría que se haya presentado en iteraciones anteriores y se restauran los valores en reposo de los vértices que describen el centro de cada radio y la representación en base a triángulos. En la línea 3 se orientan los resortes haciendo uso de la matriz de transformación que invierte la orientación de la instancia (ver sección 5.3). Como se mencionó en la sección 5.3 el cálculo de movimiento comienza a partir de la porción del árbol que puede moverse, es decir los radios que se marcaron como origen de movimiento (línea 4).

Debido a que cada segmento del árbol se considera como una rama rígida que une dos radios, el proceso recursivo que se menciona en la línea 6 requiere conocer la posición del padre del radio que se marcó como origen de movimiento. El diámetro proyectado del radio padre en la línea 5 será de utilidad para determinar si es necesario actualizar la posición de los vértices de la representación basada en triángulos. El proceso recursivo que se menciona en la línea 6 se describe con el siguiente pseudo código:

Movimiento recursivo

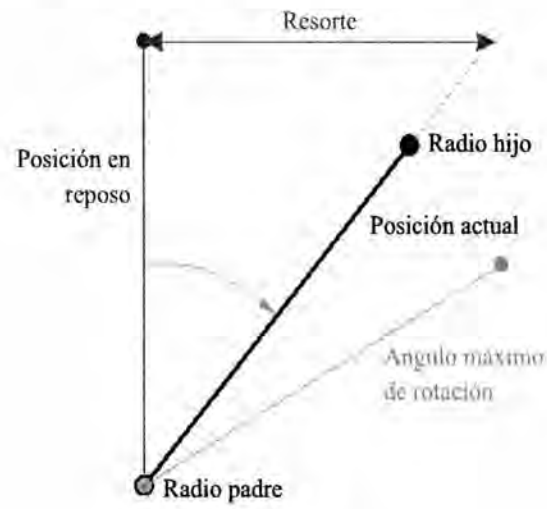
```
1  Para cada radio hijo que se recibió como parámetro:
2      Si el radio esta asociado a un resorte:
3          Calcula el movimiento del resorte que le corresponde a este radio.
4          Si el movimiento es despreciable y algún ancestro se ha movido:
5              Calcula la nueva posición del radio sumando la posición del
6              padre al vector que une a su padre con él.
7          De otro modo:
```

```
8           Mueve el radio conforme al movimiento del resorte asociado.
9       De otro modo, si algún ancestro se ha movido:
10          Calcula la nueva posición del radio sumando la posición del padre
11          al vector que une a su padre con él.
12      Calcula el diámetro que proyecta.
13      Si el diámetro proyectado de este radio o de su padre es mayor que
14      aquél que se puede representar con líneas y el radio de movió:
15          Actualiza los vértices de los triángulos asociados al radio.
16      Si es un radio terminal y el radio se movió:
17          Indica el desplazamiento que sufrió el radio al grupo de hojas.
18          Regresa.
19      De otro modo:
20          Llama este proceso recursivo pasando como parámetros la posición,
21          diámetro proyectado del radio y referencias a cada uno de sus hijos.
```

Este procedimiento recursivo mueve todos los radios que recibe como parámetros, se llama al menos una vez para cada radio marcado como origen del movimiento y la recursion se encarga de mover a todos los descendientes restantes. En la línea 2 verificamos si el radio esta asociado a un resorte, en caso de estar asociado se calcula el movimiento del resorte que le corresponde al radio en cuestión, este movimiento toma en cuenta las características de la asociación como es el porcentaje de influencia que se estableció al momento de asociar la jerarquía de radios con los resortes (ver sección 5.3).

En la línea 4 verificamos si el movimiento del resorte que le corresponde a este radio es despreciable, para efectuar este cálculo proyectamos la longitud de este movimiento y comparamos el resultado con un umbral que establece el usuario, si la longitud es menor al umbral la consideramos despreciable, de esta forma efectuamos el nivel de detalle en movimiento. Si el movimiento es despreciable y ningún radio ancestro se ha movido aún, entonces no hay necesidad de actualizar la posición de ningún vértice porque esta porción del árbol permanece en reposo, por otro lado si algún ancestro se movió la nueva posición del radio actual esta dada simplemente por la

Figura 5.5: Movimiento de un radio



posición de su padre mas un vector constante que une a su padre con el radio actual, este vector se calculó durante el preproceso en base a la geometría en reposo del árbol, ver sección 2.5.

En caso de que el movimiento no sea despreciable (línea 7) movemos el radio. Como se puede apreciar en la figura 5.5 el radio actual y su padre representan los extremos de una rama rígida, el padre se considera como el extremo con posición fija mientras que el radio actual trata de seguir el movimiento del resorte bajo ciertas restricciones como es el ángulo máximo de rotación que se estableció al momento de asociar la jerarquía de radios con los resortes (ver sección 5.3).

En la línea 12 calculamos el diámetro que proyecta el radio actual. El segmento esta formado por la unión del radio padre con el radio hijo y el diámetro del segmento es el diámetro mayor de los dos radios que lo forman. Debido a que también conocemos el diámetro que proyectó el radio padre podemos determinar el diámetro que proyecta el segmento y saber de antemano si este segmento se puede representar con nivel de detalle o no. Para saber si el segmento se puede representar con líneas es necesario comparar el diámetro que proyecta con el rango mayor (w^+) del ancho máximo de línea permitido (ver sección 4.4.1).

Si el diámetro proyectado es muy grande para ser representado con líneas y el radio actual tiene una nueva posición (línea 15) es necesario actualizar la posición de los vértices de la representación en base a triángulos asociados a este radio. En este caso los vértices sufren el mismo desplazamiento que sufrió el centro del radio. Es importante aclarar que los vértices no se rotan simplemente

se desplazan, la razón es que los árboles tienden a regresar a su posición en reposo, su geometría modificada sólo es visible en pequeños lapsos de tiempo y es difícil detectar la deformación especialmente si el movimiento es ligero, además esta suposición permite aumentar la velocidad del algoritmo.

Finalmente si el radio es terminal (ver sección 2.3) y tiene un grupo de hojas asociado se le notifica al grupo el desplazamiento que sufrió el radio (línea 17), en caso de no ser terminal la recursión continúa en cada uno de los hijos de este radio.

5.4. Visibilidad en movimiento

Para efectuar la oclusión de los objetos que están fuera del alcance visual de la cámara, se utiliza la misma jerarquía de cajas descrita en la sección 4.3.1. Sin embargo el movimiento de las instancias introducen nuevos aspectos que se deben considerar para determinar su visibilidad.

Con el fin de reducir el tiempo requerido para efectuar el cálculo de visibilidad el algoritmo establece la siguiente suposición: aquellas instancias que habiten en una caja de la jerarquía que esta completamente dentro de la visibilidad de la cámara es imposible que desaparezcan de la vista aun cuando estén en movimiento y difícilmente una porción completa desaparecerá de la vista a menos que el movimiento sea muy drástico y se encuentre muy cerca de los planos de oclusión de la vista de la cámara. Esta suposición permite omitir la validación de visibilidad para la mayor parte de instancias presentes en la escena y da buenos resultados en la práctica.

Cuando las instancias se encuentran en cajas de la jerarquía que son parcialmente visibles desde el punto de vista de la cámara, la probabilidad de que algunas porciones del árbol o inclusive el árbol completo desaparezcan de la vista debido al movimiento es muy alta. Por este motivo estas instancias requieren una validación de visibilidad. La validación para las instancias en reposo como para aquellas que están en movimiento es idéntica a la que se presentó en la sección 4.3.2. La diferencia radica en que los volúmenes contenedores para las instancias en movimiento que se encuentran en esta situación han sido recalculados.

Para recalcular los volúmenes contenedores y mover las instancias se utiliza un algoritmo muy similar al que se presentó en la sección 5.3.2, sin embargo, debido a la validación de visibilidad, la

modificación de la geometría se efectúa en dos pasos:

1. Se calculan las nuevas posiciones de los centros de los radios que están en movimiento y una aproximación de los volúmenes contenedores. En el caso del tronco suponemos que cada radio esta encerrado en una esfera cuyo diámetro es equivalente al diámetro del radio que encierra, se calcula una caja que contenga todas estas esferas y de esta forma evitamos actualizar los vertices de la representación en base a triángulos de cada radio. En el caso de las hojas simplemente se modifica el centro de cada esfera que encierra un grupo de hojas.
2. Si los volúmenes contenedores de la instancia en movimiento detectan que el tronco es visible y existen porciones del tronco que se representan sin nivel de detalle, se recorre nuevamente la jerarquía del árbol con el fin de actualizar los vertices necesarios de la representación en base a triángulos.

Estos dos pasos evitan que se calcule toda la geometría de un árbol que posiblemente no sea visible.

En la figura 5.6 podemos apreciar el caso de una instancia que se encuentra en una caja de la jerarquía (caja con contorno punteado) que es parcialmente visible desde el punto de vista de la cámara que se muestra en la imagen izquierda. Haciendo uso de una vista superior podemos ver lo que sucede al momento de mover la instancia, aquellas porciones de la instancia que salen de la vista de la cámara (representada como la región sombreada) desaparecen, note que los volúmenes contenedores (cajas con contorno sólido) cambian de tamaño y posición acorde al movimiento de la instancia.

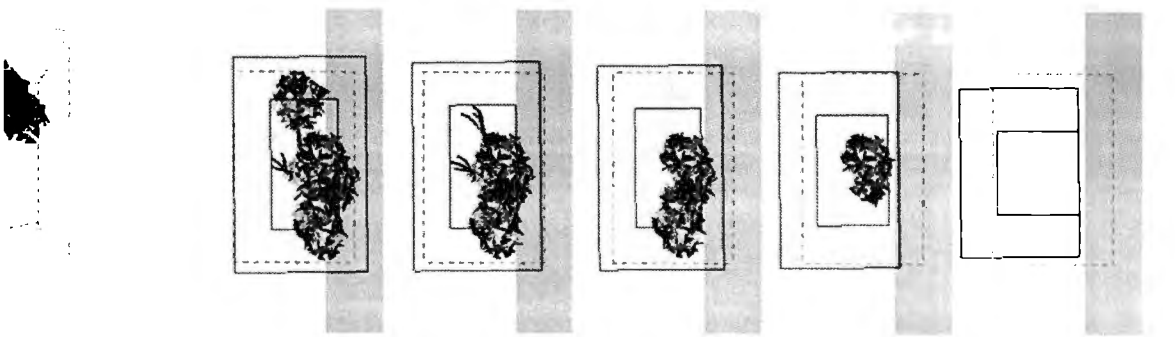
5.5. Modificaciones al nivel de detalle

Los algoritmos para efectuar el nivel de detalle basado en primitivas cuando existe movimiento son prácticamente idénticos a los que se describieron en la sección 4.4. A continuación se presenta el pseudo código para efectuar el nivel de detalle en el tronco:

LOD del tronco

```
1 Calcula el factor de reducción.
2 Si existe al menos un segmento que se pueda dibujar con líneas:
```

Figura 5.6: Ejemplo de oclusión en movimiento




```
3       Descarta los segmentos que son muy delgados.
4       Para cada ancho de línea disponible:
5           Dibuja con líneas todos los segmentos que se encuentren dentro del
6           rango válido para el ancho actual.
7       Si aún quedan segmentos sin dibujar:
8           Dibuja el resto de los segmentos como triángulos.
9   De otro modo:
10      Si el tronco esta en reposo:
11          Llama al display list del tronco.
12      De otro modo:
13          Dibuja los segmentos como triángulos.
```

Como podemos observar el algoritmo es muy parecido al que se analizó en la sección 4.4.1, las diferencias comienzan a partir de la línea 10 donde se verifica si el tronco esta o no en reposo. Si el tronco esta en reposo significa que su forma no ha sido alterada por las fuerzas presentes en la escena (ver el nivel de detalle en movimiento en la sección 5.3.2), por este motivo podemos utilizar el *display list* que se generó al momento de cargar la escena en memoria.

Por otro lado si el tronco esta en movimiento significa que su geometría ha sido alterada, por este motivo no podemos usar el *display list* que contiene el tronco en reposo, es necesario dibujarlo en base a la geometría modificada. En este caso se dibujará como un conjunto de líneas y triángulos

(líneas 5 y 8) cuando es posible utilizar el nivel de detalle o únicamente con triángulos (línea 13) cuando no se puede hacer uso del nivel de detalle.

En el caso de las hojas el algoritmo de nivel de detalle que considera el movimiento es casi idéntico al que se presentó en la sección 4.4.2, la única diferencia es que para mover los grupos de hojas de forma consistente con el movimiento del tronco es necesario mover el grupo de la misma forma en que se movió su radio terminal asociado (ver sección 2.4), esta traslación modifica la posición del grupo y por lo tanto también puede modificar la profundidad de su centro con respecto a la cámara que se utiliza para calcular la visibilidad (ver sección 5.4) y el área que proyecta el grupo.

5.6. Resultados

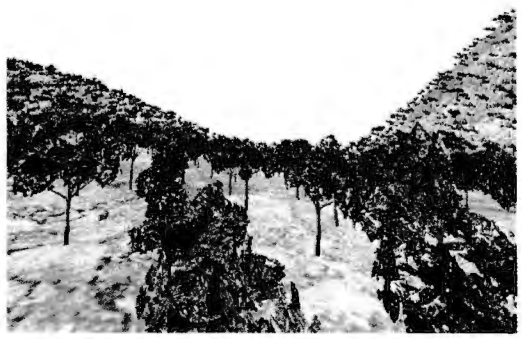
En la figura 5.7 podemos observar una escena en reposo en las imágenes del renglón superior y en movimiento en las imágenes del renglón inferior. Las imágenes de la columna izquierda muestran la escena, el contador de primitivas y cuadros por segundo sin hacer uso de ningún tipo de nivel de detalle mientras que la columna derecha presenta la misma información cuando se aplica nivel de detalle al movimiento y a la geometría de los árboles haciendo uso de puntos y líneas.

El comportamiento del tiempo requerido para desplegar cada escena se muestra en la figura 5.8, la línea sólida representa el tiempo que consumió cada cuadro al hacer uso de nivel de detalle mientras que la línea punteada muestra el tiempo que consumió cada cuadro cuando no se aplica nivel de detalle. Podemos observar que el tiempo requerido es mayor cuando comienza el movimiento en ambos casos (del cuadro 50 al 60), pero al hacer uso de nivel de detalle en el movimiento (línea sólida) el tiempo disminuye en forma constante conforme el movimiento de las instancias va disminuyendo y regresan a su posición en reposo.

En la figura 5.9 podemos apreciar una comparación del tiempo requerido para desplegar cada cuadro cuando se utiliza LOD tanto en la geometría como en el movimiento de las instancias (línea sólida) y cuando sólo se usa nivel de detalle en la geometría (línea punteada). A pesar de que en ambos casos se esta utilizando nivel de detalle basado en puntos y líneas, podemos observar que al inicio del movimiento (del cuadro 50 al 60) los cuadros consumen mayor tiempo en ambos

Figura 5.7: Ejemplo de escena en movimiento

Triángulos: 1212741
Líneas: 0
Puntos: 0
FPS: 11

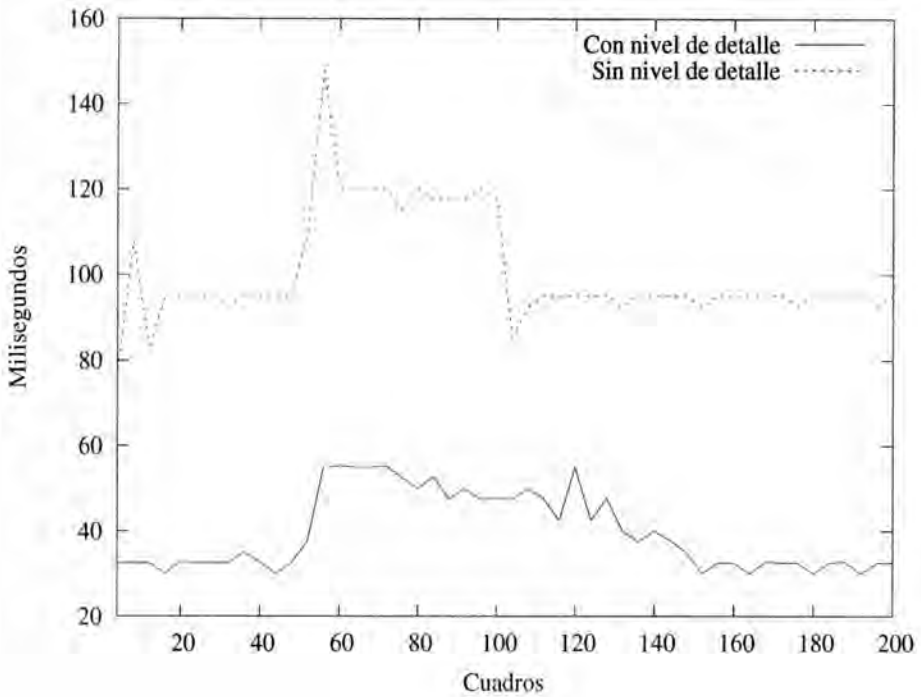


Triángulos: 41631
Líneas: 5006
Puntos: 154464
FPS: 32

Triángulos: 1222048
Líneas: 0
Puntos: 0
FPS: 9

Triángulos: 42199
Líneas: 5009
Puntos: 154874
FPS: 18

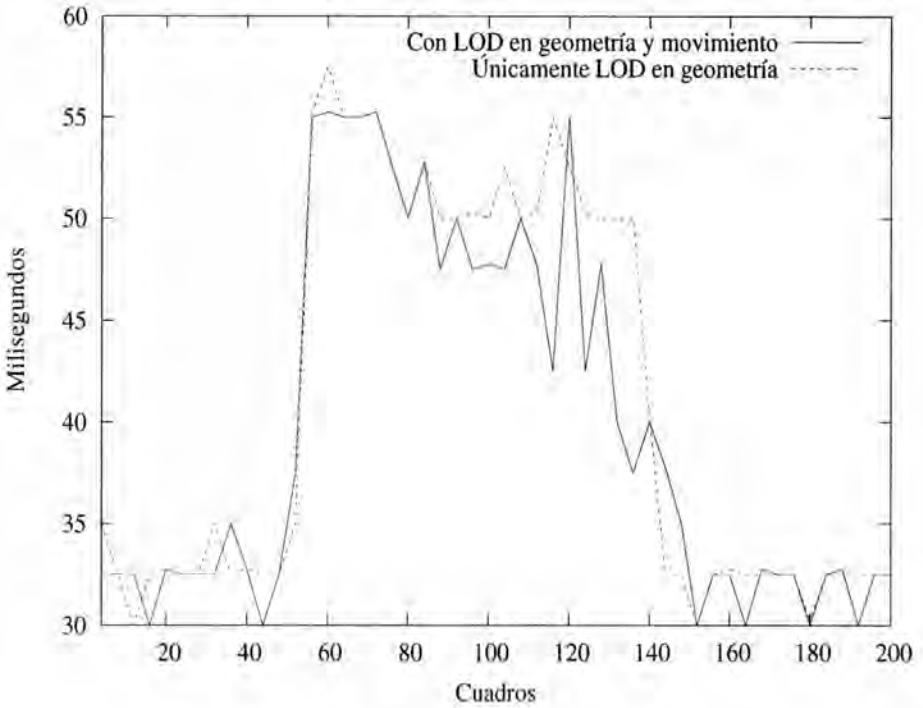
Figura 5.8: Gráfica de escena en movimiento (caso #1)



casos pero cuando se usa nivel de detalle en el movimiento (línea sólida) el tiempo requerido para desplegar cada cuadro disminuye en forma constante debido a que evita el cálculo de los movimientos poco relevantes. Por otro lado cuando se omite el nivel de detalle en movimiento (línea punteada), el tiempo que consumen los cuadros no disminuye hasta que todos los resortes regresan al estado de reposo (cuadro 140).

En ambas gráficas se puede apreciar que al utilizar el nivel de detalle durante el movimiento de las instancias (línea sólida) se presentan repentinas depresiones en el tiempo requerido para desplegar cada cuadro, la depresión más notable se puede apreciar en la figura 5.9 aproximadamente en el cuadro 115. Estas depresiones se deben a que inicialmente el resorte se estira por la fuerza del viento, sin embargo la fuerza de restitución trata de restaurar el estado en reposo del resorte ($\Delta X = 0$), cuando la longitud del resorte es cercana a cero las ramas parecen estar en reposo, por este motivo el nivel de detalle en movimiento omite muchos cálculos y el cuadro se despliega en menor tiempo. Sin embargo, la inercia que lleva el resorte lo obliga a oscilar hasta que poco a poco pierde velocidad y regresa a su estado en reposo, estas oscilaciones incrementan nuevamente la longitud del resorte y modifican la forma de las ramas asociadas por lo cual se reduce el uso del

Figura 5.9: Gráfica de escena en movimiento (caso #2)



nivel de detalle en movimiento y el tiempo necesario para desplegar el cuadro aumenta nuevamente (cuadro 120).

Es importante aclarar que el mecanismo que se presentó para efectuar el nivel de detalle en movimiento (ver sección 5.3.2) introduce un pequeño error cada vez que se considera que un movimiento es despreciable, a pesar de es este error no es visible para el radio en cuestión, puede afectar el movimiento de sus hijos si muchos de estos pequeños errores se van acumulando conforme avanza la recursion. Por ejemplo, si se omite tres veces consecutivas un movimiento de 0.4 píxeles, se esta ignorando un movimiento de 1.2 píxeles que debería de afectar al cuarto radio consecutivo. Este error se manifiesta como un movimiento ligeramente brusco en las ramas formadas por los radios más cercanos a las hojas de la jerarquía. Para evitar este error es necesario ir acumulando la longitud proyectada del movimiento de los radios que se han omitido de forma consecutiva, el contador regresa a cero cuando se encuentra un radio que tiene un movimiento acumulado mayor al umbral y se mueve de forma normal.

En la figura 5.10 podemos apreciar imágenes de un terreno con pasto, tanto el pasto como los árboles se mueven haciendo uso del algoritmo de movimiento descrito en este capítulo. A pesar

Figura 5.10: Pasto en movimiento

Escena en reposo

Escena en movimiento

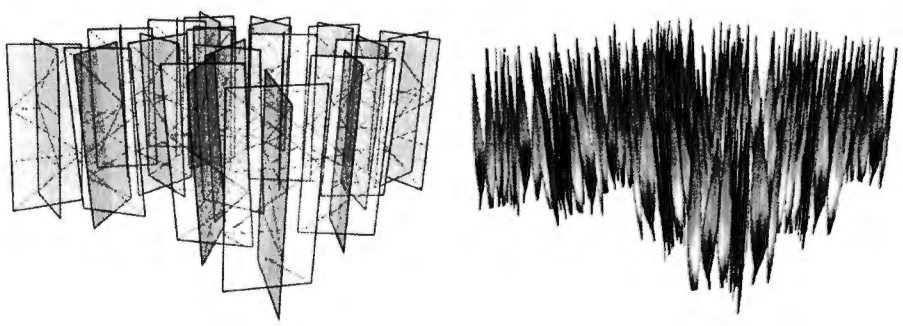
Vista superior

de que la fuerza del viento sólo actúa en una dirección, esta aplicación incluye varios resortes que reaccionan de forma ligeramente distinta, algunos ejercen más oposición a la fuerza del viento mientras que otros son más flexibles, de esta forma se produce ruido en el movimiento de las instancias y se obtiene un resultado más natural. Otra ventaja de que existan varios resortes en la escena es que el movimiento no es el mismo para todas las instancias, mientras unas instancias se encuentran en reposo otras están en movimiento, esto reduce el impacto en el tiempo requerido para el cálculo de movimiento.

Para optimizar recursos una sola instancia de pasto representa varios pastos haciendo uso de *billboards* donde cada uno forma una pequeña jerarquía con su propio radio raíz (ver sección 2.1) como se muestra en la figura 5.11, debido a que estos *billboards* son prácticamente paralelos al eje Y, los vectores normales han sido modificados para que puedan ser iluminados. Estos bloques de pasto se distribuyen sobre el terreno con una inclinación equivalente al vector normal de la superficie que cubren (ver sección 3.2.1). Para optimizar el cálculo de movimiento el algoritmo para calcular el desplazamiento de los radios sólo se efectúa para la jerarquía del primer *billboard* en cada bloque de pasto, y gracias a que todas las jerarquías del bloque tienen el mismo número de radios, los desplazamientos de los centros de los radios de la primera jerarquía se pueden copiar al resto de las jerarquías.

Con lo que respecta al nivel de detalle, en el caso del pasto no es posible utilizar el algoritmo basado en líneas puesto que los *billboards* utilizan textura para definir la forma del pasto y sus

Figura 5.11: Bloque de pasto



Smooth shading y textura *Wireframe*

triángulos no forman cilindros. Por este motivo en el caso del pasto se definió un rango de visión, aquellos bloques que están fuera de este rango simplemente se ignoran, esta situación se puede apreciar en la vista superior de la escena en la figura 5.10. El control de nivel de detalle para el pasto es un área importante de trabajo a futuro y podría efectuarse haciendo uso de impostores [Sch95] o simplemente substituyendo cada bloque de *billboards* por un solo *billboard* rectangular orientado hacia la cámara en los casos donde la distancia de la cámara al *billboard* sea muy grande, estos *billboards* podrían desvanecerse [AMH02] con un valor de transparencia para efectuar una transición suave.

5.7. Conclusión

En la figura 5.1 se puede apreciar que la sección de las hojas y algunos de sus elementos tienen un contorno punteado, este contorno significa que estos elementos son opcionales, por ejemplo un árbol puede carecer de hojas o incluir únicamente la información de los grupos pero sin ninguna representación geométrica. Estas regiones opcionales se diseñaron con la finalidad de que el programador pudiera sustituir la geometría original de las hojas por alguna otra que puede ser generada por métodos procedurales o haciendo uso de primitivas con textura.

Un área importante de trabajo a futuro es desarrollar un algoritmo para simular el movimiento de cada hoja del árbol. El algoritmo que se presentó en este capítulo considera a las hojas como parte de un grupo y simplemente traslada los grupos para que sigan el movimiento de la rama que los

sostiene pero las hojas individuales no tienen ningún otro tipo de movimiento. Pienso que una simulación de movimiento de tipo procedural como la que presentan Fang y Hart [FH02] o Stamminger y Drettakis [SD01] podría ser una buena opción. El algoritmo que se presentó en este capítulo también omite la detección de colisiones entre los componentes del árbol así como entre instancias en movimiento. Esta detención puede ser de utilidad para simulaciones de crecimiento [MP96] o interacción directa con alguna porción del árbol. Para agregar más realismo al movimiento también es importante considerar otro tipo de fuerzas como pueden ser torceduras que se ejercen sobre el segmento padre cuando únicamente el segmento hijo se ve afectado por la fuerza.

Otra área importante de investigación a futuro es desarrollar un algoritmo que permita simular el movimiento de árboles pero en lugar de hacer uso de un nivel de detalle basado en primitivas, utilice uno basado en geometría para desplegar el tronco. En otras palabras, en lugar de substituir triángulos por líneas, podríamos reducir dinámicamente el número de triángulos que se requieren para representar el tronco. La idea esta basada en utilizar la jerarquía y haciendo uso de mediciones dependientes de la vista de la cámara, identificar los radios que podrían omitirse y conectar únicamente los radios que son visualmente importantes. Algunas ideas para descartar los radios son:

1. En base al ángulo:
   a) Ángulo real entre dos segmentos contiguos.
   b) Ángulo proyectado entre dos segmentos contiguos.
2. En base al diámetro proyectado del cilindro.

La opción 1 toma en cuenta el ángulo entre dos segmentos contiguos para eliminar el radio que comparten, en este caso se puede usar el ángulo sin tomar en cuenta su orientación (caso (a)) o el ángulo proyectado (caso (b)). El caso (b) a pesar de requerir más cálculos permitiría hacer un uso más extenso del LOD, puesto que existen puntos de vista donde el ángulo entre dos segmentos queda oculto y se puede eliminar el radio compartido para unir los dos radios restantes y formar un solo segmento que represente los segmentos originales de forma adecuada. Sin embargo estas dos opciones son costosas puesto que al eliminar radios de la jerarquía, se crea un nuevo grupo de segmentos con nuevos ángulos entre ellos que se deben calcular.

Una opción más económica en términos de cómputo consiste en considerar únicamente el diámetro del cilindro, en este caso el LOD eliminar los radios más pequeños y afectara la forma del árbol en las partes visualmente menos importantes.

La tabla 5.1 presenta una comparación de algunas ventajas y desventajas que se pueden anticipar al comparar la idea de un nivel de detalle basado en geometría para el tronco con el algoritmo presentado en este capitulo.

Tabla 5.1: Comparación de algoritmos de dinámica y LOD para el tronco

LOD basado en primitivas	**LOD basado en geometría**
Haciendo uso de distintos anchos de línea se puede preservar el contorno del tronco	El contorno del tronco se ve afectado debido a que se modifica la geometría con el uso del nivel de detalle
Es recursivo únicamente cuando el árbol esta en movimiento	Es recursivo cuando el árbol esta en movimiento y cuando se aplica LOD lo cual puede reducir su velocidad
Tiene dependencia del hardware debido a que cada GPU soporta un número distinto de anchos para las líneas y la velocidad de *rendering* cambia drásticamente entre distintos modelos de GPU o al utilizar distintos anchos	Debido a que sólo utiliza triángulos, no presenta tanta dependencia del hardware gráfico
Es forzosamente dependiente de la vista	Dependiendo de la implementación puede o no ser dependiente de la vista
Utiliza índices para construir la representación basada en polígonos y líneas	No requiere de índices, la jerarquía y los arreglos paralelos pueden establecer la forma en que debe construirse la geometría
La textura e iluminación se pierden al hacer uso del nivel de detalle	La información de textura e iluminación no se pierde al utilizar nivel de detalle, simplemente disminuye en igual medida que la geometría
La actualización de la jerarquía es independiente del proceso de *rendering*	La actualización de la jerarquía y el proceso de *rendering* son simultáneos lo cual podría ser útil para simular efectos como ruptura de ramas

Capítulo 6

Nivel de Detalle en Tiempo Crítico

El concepto de tiempo crítico puede utilizarse en muchos cálculos como puede ser movimiento, iluminación, detección de colisiones, etc. Cuando esta técnica es aplicada a la generación de imágenes, su propósito es ajustar la calidad de la imagen dinámicamente para mantener la tasa de cuadros por segundos requerida por el usuario o la aplicación. Es decir, el tiempo que consumen todas las tareas debe ser menor o igual al tiempo especificado.

Este problema tiene muchas aplicaciones prácticas en áreas como realidad virtual, realidad mixta, simuladores y vídeo juegos. Las técnicas clásicas de nivel de detalle son una primera aproximación para solucionar este problema, sin embargo su utilidad consiste en ahorrar recursos de cómputo que podemos utilizar para desplegar más objetos pero no proveen una solución general, esto se debe a que la forma en que se asignan los recursos a cada objeto no considera los demás objetos presentes en la escena.

Una forma natural de resolver este problema es aplicar nivel de detalle a la escena que se pretende desplegar y tratar de seleccionar distintos niveles de detalle para cada objeto de tal forma que el tiempo total necesario para desplegar la escena no exceda el tiempo establecido. Es aquí donde surge el concepto de nivel de detalle en tiempo crítico (TCLOD — *Time Critical Level Of Detail*).

A continuación se presenta una descripción del estado del arte, posteriormente se describen algoritmos para medir y estimar el tiempo que consume la aplicación así como el algoritmo propuesto para efectuar el TCLOD, finalmente se presentan dos implementaciones que ponen en práctica este algoritmo, con sus respectivos resultados y conclusiones.

6.1. Estado del arte

Funkhouser y Sequin [FS93] mencionan diferentes técnicas para efectuar el nivel de detalle en tiempo crítico, una de las más populares se conoce como "*Adaptive Detail Elision*" [Sch83]. Esta es una técnica reactiva que ajusta los parámetros de nivel de detalle en base al tiempo requerido para el despliegue de los cuadros anteriores. El problema de esta técnica es que solamente es adecuada para escenas con un gran factor de coherencia, es decir, la complejidad de la escena que se pretende desplegar no debe variar drásticamente entre cuadro y cuadro.

Cuando tenemos escenarios con poco nivel de coherencia, que son muy comunes en aplicaciones reales, los algoritmos reactivos no son de utilidad debido a que dependiendo de la navegación o algún otro factor, la complejidad de la escena a desplegar puede variar drásticamente entre cuadro y cuadro. En este caso la solución más deseable es un algoritmo pro-activo que pueda anticipar el problema.

Funkhouser y Sequin [FS93] tratan de encontrar una solución al nivel de detalle en tiempo crítico que garantice al menos una taza de cuadros establecida por el usuario durante la visualización de ambientes virtuales discontinuos. Para alcanzar este objetivo desarrollan un algoritmo capaz de predecir la complejidad del próximo cuadro a desplegarse. Este algoritmo basa sus predicciones en dos métricas: a) el beneficio; y b) el costo de cada uno de los objetos a desplegarse por cada uno de sus niveles de detalle y configuraciones de *rendering*. El costo es el tiempo que tarda en procesarse un objeto dado. El beneficio depende de varios factores como son: tamaño, precisión, semántica, etc., e indica la importancia de un objeto dado en la imagen que esta por desplegarse.

El problema detectado por Funkhouser y Sequin [FS93] consiste en maximizar el beneficio con la restricción de que el costo sea menor que el tiempo asignado para ese cuadro. Sin embargo este problema cae en la categoría de NP-duro [GJ90] y se conoce como "*Multiple Choice Knapsack Problem*" (MCKP) [IHTI78]. Este problema tiene las siguientes características:

- ☆ Cada objeto dentro de la escena y sus posibles niveles de detalle forman un conjunto candidato.
- ☆ La restricción de tiempo es el tamaño de la mochila.
- ☆ El costo del objeto en términos del tiempo es el tamaño del objeto a transportar.

☆ El beneficio del objeto en términos de la importancia visual es la ganancia por transportar este objeto.

La solución propuesta por Funkhouser y Sequin [FS93] consiste en desplegar únicamente los objetos con mayor valor, donde el valor es el resultado de dividir el beneficio entre el costo. La solución propuesta genera buenos resultados en términos de mediciones numéricas y es en el peor de los casos la mitad de buena que la solución óptima.

El problema con esta solución es que el tiempo destinado para el cuadro se puede agotar y algunos objetos no se despliegan. Esto se ve reflejado en pantalla como pequeños objetos que aparecen y desaparecen durante la navegación en el escenario virtual.

Maciel y Shirley [MS95] desarrollaron una extensión del trabajo de Funkhouser y Sequin organizando la escena completa en una jerarquía de primitivas texturizadas que representan los objetos de la escena. A diferencia del método propuesto por Funkhouser y Sequin [FS93], esta técnica asegura que todos los objetos en la escena son desplegados en cada cuadro. El problema de esta solución es que sólo es útil en ambientes al aire libre, no soporta objetos en movimiento y puede ser costoso en términos de recursos computacionales si la jerarquía crece demasiado y se requiere muchas texturas.

Mason y Blake [MB97] propusieron un método híbrido entre la solución de Funkhouser y Sequin [FS93] y la solución de Maciel y Shirley [MS95]. En este caso los LODs se organizan en una jerarquía donde cada nodo representa impostores de uno o más objetos en la escena, en contraste con [MS95] los impostores no son primitivas texturizadas. En cada cuadro un algoritmo recorre la jerarquía desde la raíz hasta encontrar el conjunto de nodos que represente la porción visible de la escena. Sin embargo, así como [FS93] esta solución sólo puede garantizar resultados que son al menos la mitad de buenos que la solución óptima y de la misma forma que [MS95] carece de utilidad en escenas que incluyen objetos en movimiento.

En el caso de LOD continuo, las variables discretas del MCKP se pueden remplazar por variables continuas. Gobbetti y Bouvier [GB99] proponen una solución que transforma el problema original en un problema sin restricciones que puede ser resuelto con optimizaciones iterativas hasta: a) encontrar un resultado con un error aceptable; ó b) consumir todo el tiempo destinado para la

optimización y utilizar el resultado no óptimo de la última iteración. Este método ofrece muchas ventajas con respecto a las soluciones anteriores, pero la optimización puede ser costosa cuando la escena tiene múltiples objetos y hay muy poco tiempo disponible para generar el cuadro. Wimmer y Schmalstieg utilizaron multiplicadores de Lagrange en [WS98] y describieron una solución no iterativa para aquellos casos donde el número máximo de triángulos que componen a cada objeto en la escena es al menos igual que el número máximo de triángulos que se pueden desplegar dentro de la restricción de tiempo para el cuadro, otros casos requieren iteraciones. Una desventaja de esta solución es asumir que todos los triángulos tienen el mismo costo e ignorar la configuración de *rendering* o el área proyectada.

Finalmente Zach *et al.* [ZMK02] combinaron la jerarquía de LOD discreto de [MB97] con métodos de gradiente ascendente que muestran una dirección de búsqueda para seleccionar el LOD continuo. Este método también utiliza optimizaciones iterativas hasta que el tiempo del cuadro se ha distribuido entre todos los objetos visibles ó el tiempo de la optimización se ha agotado, en cuyo caso se utiliza una solución no óptima.

6.2. Muestreo

El algoritmo que se presenta en este capítulo debe conocer de antemano el tiempo que consume el procesamiento y despliegue de cada cuadro, este tiempo puede cambiar dependiendo de muchos factores como son: número de aplicaciones activas al momento de ejecutar la aplicación, características del hardware de la máquina huésped, etc. Nuestra propuesta para estimar el tiempo de cada cuadro considera únicamente dos aspectos que son: a) el tiempo de *rendering* que consume cada elemento a desplegarse; y b) el tiempo que consumen los procesos de la aplicación ajenos al *rendering*. La estimación de estos tiempos se deriva de dos muestreos que se describen a continuación.

6.2.1. *Rendering*

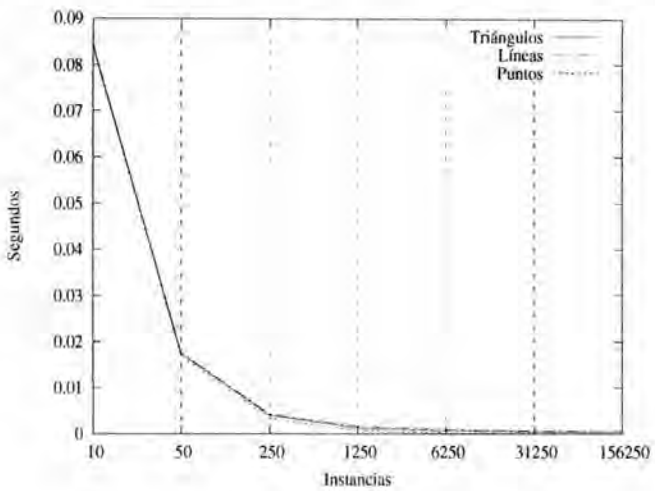
El muestreo del *rendering* se efectuó en una aplicación distinta a aquella que permite navegar de forma interactiva en el ecosistema. El objetivo de esta aplicación es medir el tiempo que consume desplegar cada una de las primitivas en distintas configuraciones de *renderig* y distintos tamaños o áreas proyectadas. En una etapa de preproceso la aplicación simplemente despliega un número de instancias de cada primitiva, mide el tiempo que consumió el *rendering* del cuadro y finalmente calcula el tiempo promedio de cada instancia. Al efectuar estas sencillas mediciones se identificaron varios retos.

Para efectuar estas mediciones fue necesario implementar un cronómetro de alta precisión que devuelve valores con 64 bits de exactitud. Otro tipo de cronómetros introducían errores que se acumulaban al incrementar el número de veces que se media el mismo cuadro o presentaban únicamente valores que saltaban de 5 en 5 milisegundos aun desactivando la sincronización vertical del monitor.

Otro aspecto que dificulta la medición es que el tiempo que consume cada primitiva es dependiente del número total de instancias que se procesan en el cuadro, por ejemplo, el tiempo que consume cada triángulo cuando sólo se están dibujando 10 triángulos por cuadro es mayor que el tiempo que consume cuando se dibujan 1000 triángulos. En la figura 6.1 podemos observar que el tiempo promedio de las instancia de cada primitiva cambia al modificar el número total de instancias que se despliegan en cada cuadro, los valores describen una trayectoria similar a la curva que forma la función exponencial $f(x) = e^{\frac{1}{x}}$ para los valores positivos de x.

Afortunadamente las primitivas tienen un comportamiento relativamente similar, en el caso del hardware donde se efectuó esta medición se puede apreciar que al rebasar las 1500 instancias por cuadro, el tiempo de la primitiva se estabiliza. A pesar de que esta gráfica sólo muestra puntos de tamaño 1, líneas de espesor 1 y triángulos que cubren un 50 % del área de dibujo, donde cada primitiva tiene una configuración de *rendering* particular, distintos tamaños, espesores, áreas proyectadas y configuraciones de *rendering* presentan el mismo comportamiento. De la misma forma, un comportamiento similar se observó en distintas configuraciones de hardware donde se efectuó esta prueba.

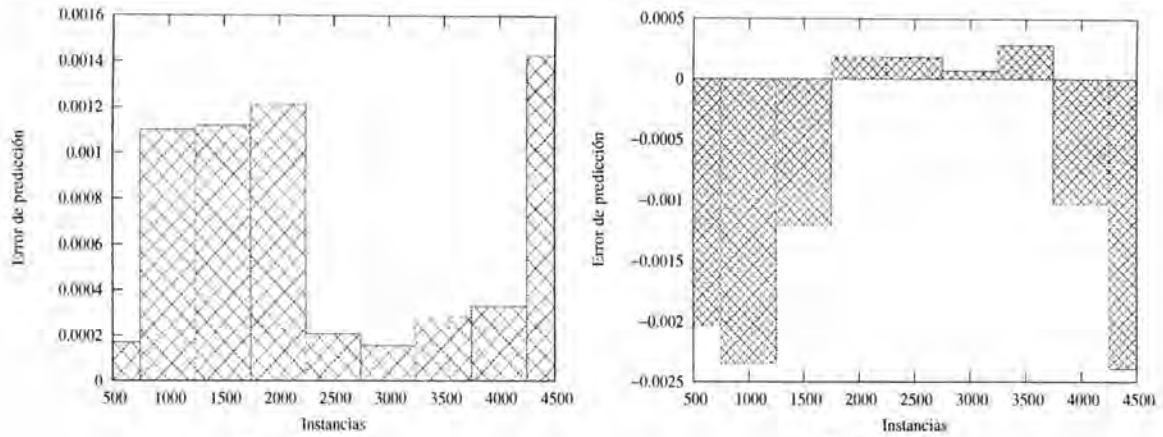
Figura 6.1: Tiempo al medir distinto número de instancias



La figura 6.2 muestra los errores de predicción obtenidos mediante muestreos efectuados con distinto número de instancias por cuadro, todos los muestreos tratan de predecir el costo de dos cuadros que despliegan 1000 puntos, 1000 líneas y 1000 triángulos, cada primitiva con una distinta configuración de *rendering*. Podemos observar que en ambos casos el muestreo con el menor error de predicción es aquél que muestreó el mismo número de instancias totales (3000 instancias) presentes en el cuadro a predecir, y no aquél que muestreó el mismo número de instancias por primitiva (1000 instancias). Un efecto similar se apreció en distintas pruebas. Aun cuando en estas gráficas algunas predicciones pueden parecer muy erróneas, como es el caso de la medición con 4500 instancias, ninguno de estos errores supera el 10 % del tiempo real de cada cuadro. A partir de estas pruebas se puede determinar que el costo de cada primitiva también se ve afectado por el número de instancias de primitivas distintas presentes en el mismo cuadro y en términos generales la mejor predicción se presenta cuando utilizamos un muestreo que midió el mismo número de instancias totales presentes en el cuadro a predecir.

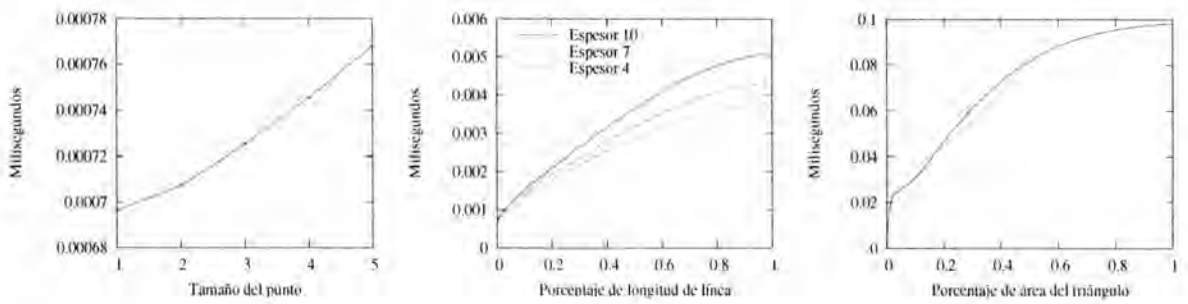
Finalmente, la porción del área de dibujo que cubre la primitiva dependiendo de su tamaño, espesor o área proyectada afecta el tiempo que consume. La figura 6.3 muestra el tiempo que consume cada primitiva con una cierta configuración de *rendering* al modificar su tamaño, espesor, longitud o área proyectada. En el caso de las líneas no basta con modificar su espesor puesto que la longitud también afecta su costo, la gráfica central muestra el tiempo que consumen tres

Figura 6.2: Predicción del tiempo de dos cuadros



Error de predicción medido en segundos

Figura 6.3: Tamaños, longitudes y áreas de primitivas



espesores distintos de línea al incrementar su longitud. Los porcentajes de longitud y áreas para líneas y triángulos se miden con respecto al tamaño del plano de proyección, un triángulo con un porcentaje igual a 1.0 tiene un área del mismo tamaño que el plano de proyección. En el caso de los puntos la gráfica muestra una recta mientras que los triángulos y las líneas presentan una trayectoria similar a la curva que describe la función $f(x) = \sqrt{x}$, una de las razones por las cuales se produce este comportamiento es que conforme van creciendo las primitivas, varias porciones tienden a salir del área de dibujo, es decir son eliminadas por los planos de oclusión y por lo tanto no incrementan el costo del *rendering* en forma lineal.

Cabe aclarar que este muestreo procesa todas las instancias de primitivas con un solo `glBegin`—`glEnd`, el tiempo extra que introduce cada una de estas aperturas y cierres no mostró tener un

impacto muy drástico en la medición y se considera como parte del costo de la aplicación (ver sección 6.2.2).

La información que se recupera durante el muestreo se almacena y posteriormente es leída por la aplicación que permite la navegación interactiva en al escenario virtual, con esta información se puede determinar el tiempo de *rendering* que consume cada elemento de la escena a desplegarse. En el caso de los puntos, basta con recuperar el tiempo promedio de un punto con un determinado tamaño y multiplicarlo por el número de puntos que se utilizan en el nivel de detalle de las hojas.

En el caso de las porciones del tronco y las hojas que se dibujan sin LOD, es necesario identificar la configuración de *rendering* de los triángulos (*flat shading*, *smooth shading*, textura, uso de *alpha test*, etc.), posteriormente determinar el tamaño promedio que proyectan los triángulos en el plano frontal del *frustum* (ver sección 4.2) y obtener el porcentaje de área que cubren del plano frontal. Estos tres datos son necesarios para recuperar el tiempo promedio que consume un triángulo con estas características y al multiplicar por el número de triángulos se obtiene el tiempo que consume la porción del árbol que se dibuja sin LOD.

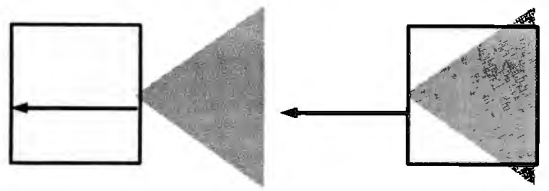
Como se mencionó en la sección 4.2 el archivo que contiene la información del árbol incluye un arreglo con la longitud acumulada de los segmentos ordenados que permite estimar la longitud que proyecta un grupo de segmentos del tronco que han sido dibujados utilizando líneas. Debido a que conocemos la longitud y el ancho de las líneas que se utilizaron para dibujar los segmentos podemos estimar el tiempo de *rendering* que consume el LOD para el tronco.

6.2.2. Aplicación

El muestreo de la aplicación consiste en evaluar el tiempo que consumen los procesos ajenos al *rendering* como puede ser el cálculo de visibilidad de los objetos en la escena (ver sección 4.3.1), determinar el número de puntos, líneas y triángulos para cada instancia, movimientos de cámara, etc. Algunas tareas relacionadas con el *rendering* se han incluido también en esta etapa como son: las transformaciones de matrices, habilitar y deshabilitar opciones que modifican las configuraciones de *rendering*, apertura y cierre de las instrucciones `glBegin`—`glEnd`, etc.

En contraste con el muestreo del *rendering* (ver sección 6.2.1), este muestreo efectúa sus me-

Figura 6.4: Trayectorias para el muestreo de la aplicación



Primera trayectoria Segunda trayectoria

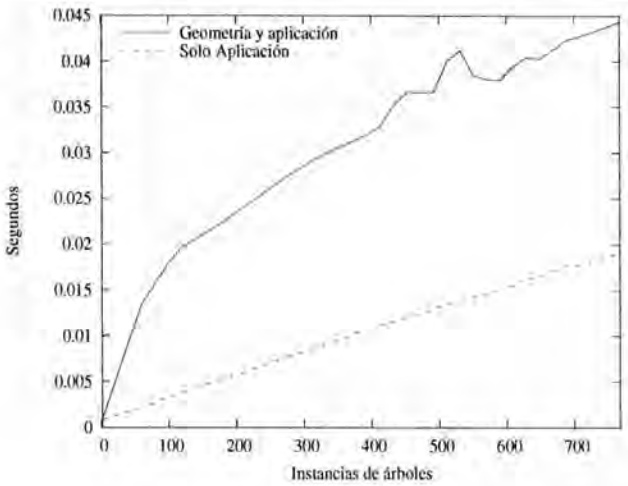
diciones sobre la escena real que se utilizará posteriormente en la aplicación que permite efectuar la navegación interactiva, de tal forma que el número de instancias, la profundidad del *frustum*, la resolución y el tamaño del terreno así como otras características sean idénticas en ambas aplicaciones.

La implementación actual del muestreo de la aplicación simplemente considera dos trayectorias de movimiento para la cámara durante las cuales se mide el tiempo que consume la aplicación sin procesar los vértices para su despliegue, estas trayectorias están representadas en la figura 6.4. La primera trayectoria ubica la cámara en un extremo del terreno y la desplaza de tal forma que el área del *frustum* que cae dentro del terreno se incremente en una proporción constante, conforme la cámara retrocede el número de instancias que encierra el *frustum* se incrementa hasta alcanzar el número máximo de instancias que el *frustum* puede visualizar en un momento dado. La segunda trayectoria es muy similar pero en este caso la cámara se ubica en un extremo del terreno y conforme retrocede el número de instancias que encierra el *frustum* va disminuyendo. En la figura 6.5 podemos apreciar una gráfica de la primera trayectoria que compara el tiempo que consumen únicamente los procesos de la aplicación contra el tiempo que consumen tanto los procesos de la aplicación como el *rendering* de las instancias utilizando el algoritmo de LOD que se presentó en el capítulo 4.

6.2.3. Estimación real

Para verificar que las mediciones descritas en las secciones anteriores pueden estimar de forma adecuada el tiempo que consume una aplicación real, se efectuó una prueba en la aplicación descrita en el capítulo 4 utilizando varios ecosistemas con distintas características. Para estimar el tiempo que

Figura 6.5: Tiempo de geometría y aplicación

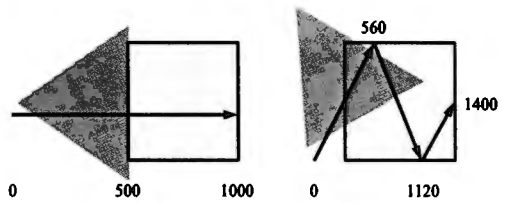


consume el *rendering*, se calcula el número de puntos, líneas y triángulos que utiliza cada instancia de árbol que se pretende desplegar, este cálculo considera las distintas variantes de tamaño, espesor, longitud y área proyectada así como configuraciones de *rendering*, en base al muestreo de *rendering* se calcula el tiempo de cada una de estas primitivas en la forma descrita en la sección 6.2.1. La suma del tiempo que consumen todas las instancias a desplegar constituye la estimación del tiempo de *rendering* (t_r) para un cuadro en particular.

La estimación del tiempo de la aplicación (t_a) consiste únicamente en identificar el número de instancias que se desplegarán en el cuadro e interpolar las dos muestras más cercanas obtenidas durante el muestreo de la aplicación (ver sección 6.2.2).

Dado que el CPU y el GPU efectuan sus tareas en forma paralela, una simple suma de las estimaciones de tiempos (t_a, t_r) no aproximó de forma correcta el tiempo del cuadro. Al efectuar varias pruebas se observó el siguiente comportamiento: a) cuando uno de los dos valores (t_a, t_r) es mucho mayor que el otro, el valor más pequeño es prácticamente despreciable; y b) cuando ambos valores son iguales el valor real generalmente es un poco menor que la suma de dos los valores estimados (t_a, t_r). Para tratar de simular este comportamiento inicialmente se pensó en las siguiente ecuación que produce el tiempo total estimado (t_e).

Figura 6.6: Trayectorias para las pruebas



Trayectoria lineal Trayectoria zigzag

$$t_e = \begin{cases} \left(\frac{t_r}{t_a}\right)^2 * t_r + t_a & \text{si } t_r < t_a \\ \left(\frac{t_a}{t_r}\right)^2 * t_a + t_r & \text{si } t_r \geq t_a \end{cases} \tag{6.1}$$

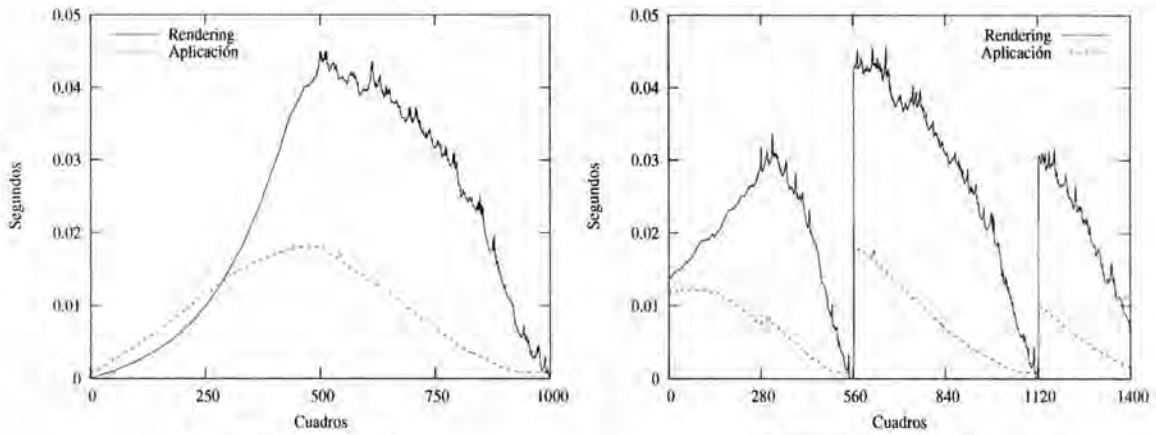
El tiempo estimado con el menor valor se multiplica por el término entre paréntesis, debido a la condición de desigualdad este término siempre tiene un valor menor o igual a 1 y al elevarlo al cuadrado su valor se reduce aún más. De esta forma se minimiza la importancia del tiempo estimado con el menor valor.

La figura 6.6 muestra dos trayectorias de cámara distintas que se utilizaron para evaluar los tiempos de estimación, se ha incluido el número de cuadro para facilitar el entendimiento de las gráficas posteriores. La trayectoria lineal es muy similar a las trayectorias que se utilizaron para el muestreo de los tiempos de aplicación (ver sección 6.2.2) mientras que la trayectoria zigzag es completamente distinta y produce saltos drásticos en la complejidad de la escena a desplegar, esta última característica es útil para mostrar que las estimaciones no dependen en ningún sentido del tiempo que consumió el *rendering* de los cuadros anteriores.

La figura 6.7 muestra dos gráficas con los tiempos estimados para el *rendering* y la aplicación en las dos trayectorias que se presentaron en la figura 6.6, cuando el número de instancias contenidas en el *frustum* crece demasiado, el tiempo estimado para el *rendering* es muy superior al tiempo estimado para la aplicación, por otro lado cuando hay pocas instancias visibles el tiempo estimado para la aplicación llega a ser igual o inclusive superior al tiempo estimado para el *rendering*.

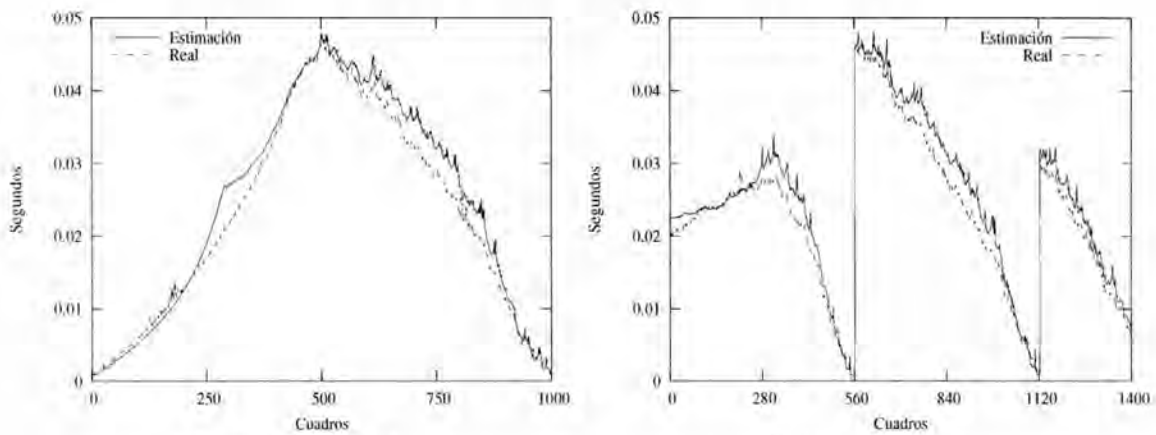
Finalmente la figura 6.8 muestra la comparación entre los tiempos estimados utilizando los valores mostrados en la figura 6.7 y la ecuación 6.1. En ambas trayectorias la curva de la estimación se ajusta relativamente bien a la curva del tiempo real que consumió el cuadro, en la trayectoria

Figura 6.7: Estimación del tiempo de *rendering* y aplicación



Trayectoria lineal Trayectoria zigzag

Figura 6.8: Comparación del tiempo estimado y el tiempo real



Trayectoria lineal Trayectoria zigzag

lineal el error promedio es de 10.89 % mientras que en la trayectoria zigzag es de 10.55 %. Distintos ecosistemas con distintas características presentaron resultados muy similares. Al concluir estas pruebas se determinó que la ecuación 6.1 podía aproximar de forma adecuada el tiempo real que consumió el cuadro, sin embargo esta ecuación presenta ciertos inconvenientes especialmente en el punto de inflexión cuando un tiempo estimado supera a otro y se cambia la condición de desigualdad en uso, estos inconvenientes y una posible solución se describen en la sección 6.4.

6.3. El algoritmo de Robin Hood para tiempo crítico

Este algoritmo fue diseñado para distribuir los recursos entre los objetos que se pretenden desplegar en pantalla, toma el nombre de este personaje histórico porque sigue la misma filosofía "tomar el exceso de recursos de los objetos que generan sobrantes y distribuirlo entre los objetos más necesitados". El algoritmo trata de eliminar varias de las restricciones de las soluciones propuestas por otros autores y fue diseñado bajo los siguientes objetivos:

- ☆ El algoritmo se enfoca exclusivamente en resolver el problema de distribución de recursos, en este caso el recurso limitado es el tiempo que existe para desplegar el cuadro.
- ☆ A diferencia de otras soluciones, debe evitar que existan objetos potencialmente visibles a los cuales no se les asigne tiempo para su despliegue.
- ☆ Debe ser útil para cualquier tipo de escena, sin importar si son al aire libre o en espacios cerrados.
- ☆ Debe ser pro-activo, es decir, debe ser capaz de predecir la complejidad del cuadro y actuar en consecuencia.
- ☆ Debe ser rápido y no representar una carga excesiva en el tiempo de cómputo de la escena.
- ☆ Debe soportar escenas con objetos estáticos así como escenas con objeto en movimiento.
- ☆ Debe poder combinarse con otras técnicas de aceleración ya existentes como son: LOD, oclusión de objetos no visibles, grafo de escena, etc.

Las restricciones que se establecieron fueron:

- ☆ El algoritmo no tratará de ofrecer una solución global al problema de LOD. El motivo es que dentro de una escena generalmente existen objetos con características variadas, una aplicación puede utilizar distintos tipos de algoritmos de LOD con el fin de generar la solución más eficiente, por ejemplo: el terreno puede utilizar ROAM (*real-time optimally adapting meshes*) [DWS*97], los personajes pueden utilizar *progressive meshes* [Hop96], las nubes posiblemente utilicen primitivas texturizadas, etc. Otras soluciones están fuertemente atadas a un tipo de algoritmo de LOD.

- ☆ Este algoritmo tratará de no confrontar problemas NP-duros [GJ90]. Encontrar la solución óptima a este tipo de problemas consumiría mucho tiempo y las soluciones estocásticas generalmente no ofrecen un resultado con la calidad o el tiempo requerido.
- ☆ El algoritmo omitirá muchos factores que pueden afectar el rendimiento de una aplicación en operación como son: procesos en ejecución, actividades del sistema operativo, etc. Sin embargo en ambientes controlados como son las consolas de juegos se espera que produzca mejores estimaciones.

El algoritmo resultante tiene dos etapas, la primera es una etapa de preproceso que realiza las siguientes actividades:

1. Determina la complejidad de cada uno de los objetos en la escena.
2. Organiza los objetos en base a su complejidad de forma ascendente.

La complejidad de un objeto se determina a partir de su representación óptima, aquella sin LOD. La complejidad es equivalente al tiempo que consume desplegar el objeto y toma en cuenta el número de primitivas y la configuración de *rendering*. En esta etapa no se considera el área proyectada del objeto puesto que ésta puede variar dependiendo de la escala o posición que ocupe cada una de las instancias de este objeto en la escena, por lo tanto en esta etapa se supone que todos los objetos proyectan la misma área. El tiempo se puede definir con la siguiente función:

Tiempo (objeto, LOD*, área proyectada)*

El segundo parámetro de la función anterior especifica el LOD considerando que cada LOD tiene su propia configuración de *rendering* y que el LOD con índice cero es la representación óptima de cada objeto. Suponiendo que existen n objetos en la escena, podemos expresar el primer paso de esta etapa como:

$$Costo\ (x) = Tiempo\ (x, 0, A) \qquad \text{para } x = 0, 1, 2, ..., n-1$$

En este caso A es el valor de área proyectada constante que se utiliza para evaluar el costo de todos los objetos. La segunda etapa se efectúa al momento de desplegar cada cuadro, los procesos se describen en el siguiente pseudo código.

Algoritmo de Robin Hood para tiempo crítico

```
1  Calcula el valor de cada instancia visible.
2  Acumula el valor de todas las instancias visibles y obtén el valor total del
3  cuadro.
4  Estima el tiempo total disponible para el rendering del cuadro.
5  Recorre las instancias de cada objeto en el orden que dicta la complejidad
6  comenzando por los objetos menos complejos.
7  Para cada una de las instancias visibles:
8          Calcula el tiempo destinado para esta instancia.
9          Resta el valor que aportó esta instancia del valor total del cuadro.
10         Resta el tiempo destinado para esta instancia del tiempo total
11         disponible para el rendering del cuadro.
12         Busca el mejor LOD que pueda desplegarse dentro del tiempo
13         destinado para esta instancia.
14         En caso de haber tiempo sobrante, súmalo al tiempo total disponible
15         para el rendering del cuadro.
```

Note que el algoritmo recorre las instancias de los objetos dos veces, el primer ciclo comprende de la línea 1 a la línea 3 y determina las instancias visibles, el valor de cada una de ellas y el valor total del cuadro. El segundo ciclo comienza en la línea 5 y efectúa el *rendering* de las instancias visibles, a diferencia del primer ciclo en este caso es necesario recorrer las instancias en el orden que se determinó en el segundo paso de la etapa de preproceso de este algoritmo.

El valor de cada instancia se determina por medio de la función de distribución de valor. Esta función indica la importancia que el algoritmo da a cada uno de las instancias, como en este caso el algoritmo se utiliza para ajustar la calidad visual del cuadro desplegado, la función generalmente dependerá del área que proyecta cada instancia en pantalla o algún factor similar. Esta función también puede incluir factores relacionados con el contexto de la aplicación como puede ser la importancia semántica de algún objeto, imagine por ejemplo una habitación virtual donde una llave tiene una gran importancia para el contexto de la aplicación, el programador desea que esta llave

se despliegue con mucha calidad aun cuando su área proyectada sea mucho menor que la del resto de los objetos presentes en la habitación, en este caso puede definir un factor de importancia para la llave que eleve el valor que se le asigna y de esta forma reducir el uso de LOD para este objeto. Esta función es muy flexible y se pueden modificar sus valores para obtener distintos resultados, la sección 6.4.2 presenta varios ejemplos de estas funciones.

Si existen n instancias visibles y cada instancia visible tiene un valor v_i, el valor total del cuadro (v_c) se puede definir como:

$$v_c = \sum_{i=0}^{n-1} v_i \tag{6.2}$$

En la línea 4 se estima el tiempo total disponible para el *rendering* del cuadro (t_r) con la ecuación 6.1 o alguna otra similar que tome en cuenta el tiempo estimado para la aplicación (t_a) y el tiempo máximo que debe consumir el cuadro (t). Es importante aclarar que el valor mínimo de t es t_a en cuyo caso $t_r = 0$.

Dada una instancia visible, podemos calcular el tiempo destinado para esta instancia (t_i) con la siguiente ecuación:

$$t_i = \frac{v_i * v_c}{t_r} \tag{6.3}$$

En la ecuación anterior v_i representa el valor de la instancia, v_c representa el valor total del cuadro y t_r es el tiempo total disponible para el *rendering* del cuadro. Una vez que se ha calculado el tiempo destinado para la instancia, procedemos a restar de los valores totales los recursos que ya se han asignado:

$$t_r' = t_r - t_i \tag{6.4}$$

$$v_c' = v_c - v_i \tag{6.5}$$

Note que t_r' y v_c' nunca tendrán un valor negativo y sólo se reducirían a cero después de haber asignando tiempo a la última instancia visible. En la línea 12 se busca aquél LOD cuyo tiempo de *rendering* no exceda el valor de t_i y genere el menor residuo posible, en caso de haber residuo se

agrega a t_r para que sea aprovechado por los objetos restantes. La búsqueda es mucho más rápida si los niveles de detalle para cada objeto en la escena han sido ordenados durante el preproceso, iniciando con aquella representación sin LOD y terminando con la representación más económica donde se aplica el LOD más severo, este orden esta implícito en la mayoría de los algoritmos de LOD continuo.

La razón por la cual el segundo ciclo (línea 5) comienza por los objetos menos complejos es que existe mayor probabilidad de que un objeto simple tenga algún sobrante de tiempo aun cuando se este desplegando sin LOD a que un objeto complejo tenga alguno. El algoritmo busca utilizar este sobrante en el *rendering* de objetos más complejos que por lo general requieren de mayor cantidad de recursos. Este mecanismo de reutilización del tiempo sobrante funciona bastante bien en la mayoría de los casos, sin embargo para el funcionamiento óptimo del algoritmo es deseable que este residuo sea mínimo, esto depende de la granularidad de niveles de detalle que ofrece cada objeto y de la función de distribución de valor. Efectos de este residuo se describen en la sección 6.5.

El funcionamiento del algoritmo se puede entender al plantear dos escenarios extremos:

1. Los objetos simples proyectan un área mucho mayor que los objetos complejos. En este caso los objetos simples reciben un valor muy grande y por ser simples es probable que se desplieguen sin ningún LOD y generen un gran residuo, este residuo se reutilizará en los objetos complejos que se despliegan al final.
2. Los objetos complejos proyectan un área mucho mayor que los objetos simples. Nuevamente el *rendering* comienza con los objetos simples los cuales reciben un valor muy pequeño puesto que proyectan poca área, en este caso es probable que tengan que utilizar algún LOD para su despliegue y generen un residuo mínimo o nulo, finalmente se despliegan los objetos complejos que a pesar de recibir un valor muy grande, debido a su complejidad, consumirán la totalidad o mayor parte de su tiempo asignado y reducirán el sobrante que se queda sin uso.

Como se puede observar, en ambos casos el algoritmo tiene un comportamiento correcto. Note que aun cuando una instancia proyecte un área diminuta el algoritmo le asignará un tiempo, la calidad del *rendering* de esta instancia dependerá de la granularidad en sus niveles de detalle. Las

secciones siguientes muestran resultados y conclusiones de distintas implementaciones.

6.4. Únicamente tiempo crítico

Esta sección describe una implementación de los muestreos y el algoritmo de Robin Hood para TC que se presentaron en las secciones 6.2 y 6.3. La implementación utiliza las características de la aplicación que se mostró en el capítulo 4 pero no implementa ningún tipo de LOD, en su lugar despliega los elementos del árbol haciendo uso exclusivamente de triángulos. La implementación restringe el número de triángulos que cada instancia de árbol puede utilizar por medio del tiempo que le fue asignado, si este tiempo es escaso la instancia se despliega de forma incompleta.

El propósito de esta implementación es probar el desempeño del algoritmo en un ambiente simple que no utilice mas que un tipo de primitivas para el *rendering* de las instancias y no incluya ningún tipo de LOD. Esta implementación pretende clarificar conceptos mencionados en la sección 6.2.3 como es la ecuación para estimar el tiempo así como aquellos mencionados en la sección 6.3 como son la función de distribución de valor, los factores de importancia y los efectos del residuo.

Inicialmente el usuario establece el tiempo máximo que puede consumir el cuadro (t) y el programa debe ser capaz de cumplir con esta restricción siempre y cuando este tiempo sea mayor o igual que el tiempo que consumen los procesos de la aplicación (t_a), así mismo la implementación tratará de utilizar este tiempo de la mejor forma para ofrecer una imagen con la mejor calidad posible. El funcionamiento de esta implementación puede resumirse en los siguientes pasos:

Inicialización: Cada elemento del árbol, las hojas y el tronco, han sido tratados como objetos independientes, por lo tanto cada uno tiene su propio valor (ver sección 6.3) y proyecta su propia área en la forma descrita en la sección 4.2. Las hojas y los troncos de todos los tipos de árboles presentes en la escena se ordenan en forma ascendente de acuerdo a su complejidad.

Primer ciclo: Cada vez que se genera un cuadro, se identifican los objetos visibles utilizando los planos de oclusión y la jerarquía de cajas presentada en el sección 4.3.1. Cada objeto visible proyecta un área en el plano de dibujo y utilizando una función de distribución de

valor se le asigna un valor. Al finalizar este primer ciclo se han determinado los objetos visibles y se ha calculado el valor total del cuadro a generar. El muestreo de la aplicación (ver sección 6.2.2) estima el tiempo que consumen los procesos de la aplicación (t_a) tomando en cuenta el número de objetos visibles. Por medio de t_a y del tiempo máximo que puede consumir el cuadro (t) la ecuación 6.1 estima el tiempo total que puede consumir el *rendering* del cuadro (t_r).

Segundo ciclo: Se recorren las hojas y los troncos en el orden que dicta su complejidad, a cada elemento visible se le asigna un tiempo para su *rendering* basándose en el valor que aportó, el elemento despliega tantos triángulos como le sea posible hasta agotar el tiempo que le fue asignado, en caso de existir algún residuo se suma al tiempo total del *rendering* (t_r) para que sea aprovechado por los elementos restantes.

6.4.1. Tiempo de *rendering*

Para poder calcular el tiempo total que puede consumir el *rendering* del cuadro (t_r) inicialmente se utilizo la ecuación 6.1 presentada en la sección 6.2.3. Debido a la condición de desigualdad, el valor de t_r se puede determinar de dos formas: a) cuando $t_r < t_a$; y b) cuando $t_r \geq t_a$. El primer problema que se presentó al utilizar esta ecuación es que el valor de t_r es desconocido, por lo tanto no se sabe si es mayor o menor que t_a.

Dado que conocemos el tiempo máximo que puede consumir el cuadro (t) se estableció la siguiente suposición: si $t_a \leq \frac{t}{2}$ entonces $t_r \geq t_a$, en el caso contrario si $t_a > \frac{t}{2}$ entonces $t_r < t_a$. A primera vista esta suposición puede parecer adecuada sin embargo no es del todo correcta, supongamos que $t = 10$ y $t_a = 4$, en este caso podría pensarse que $t_r > t_a$ sin embargo existen tres valores para t_r que satisfacen la ecuación (9.25, -2.28, 3). Esto se debe a que la parte de la ecuación cuya condición de desigualdad es $t_r \geq t_a$ se reduce a una ecuación de grado 3 que produce 3 raíces, los efectos de esta ecuación se explican más adelante en esta misma sección.

Cuando la condición de desigualdad es $t_r < t_a$ la ecuación 6.1 se reduce a la siguiente forma:

$$t = \left(\frac{t_r}{t_a}\right)^2 * t_r + t_a$$

$$t = \frac{t_r^3}{t_a^2} + t_a$$

$$t_r^3 = t * t_a^2 - t_a^3$$

$$t_r = \sqrt[3]{t * t_a^2 - t_a^3} \qquad \text{si } t_r < t_a \tag{6.6}$$

Cuando la condición de desigualdad es $t_r \geq t_a$ despejar el valor t_r de la ecuación 6.1 no es tan simple como en el caso anterior y se obtiene la siguiente ecuación de grado 3:

$$t = \left(\frac{t_a}{t_r}\right)^2 * t_a + t_r$$

$$t = \frac{t_a^3}{t_r^2} + t_r$$

$$t * t_r^2 - t_r^3 = t_a^3$$

$$-t_r^3 + t * t_r^2 - t_a^3 = 0 \tag{6.7}$$

Resolver esta ecuación de grado 3 requiere aplicar el método de Tartaglia [Wei04] que permite eliminar el término cuadrado de la ecuación cúbica. La ecuación 6.7 se puede reescribir en la forma $x^3 + a_2x^2 + a_1x + a_0 = 0$, donde $a_2 = -t$, $a_1 = 0$ y $a_0 = t_a^3$. Al aplicar la solución para la ecuación cúbica se obtienen los valores Q y R:

$$Q = \frac{3a_1 - a_2^2}{9} \tag{6.8}$$

$$R = \frac{9a_2a_1 - 27a_0 - 2a_2^3}{54} \tag{6.9}$$

El método de Tartaglia tiene el siguiente discriminante polinomial: $Q^2 + R^2$. Dado que estas raíces sólo se utilizarán cuando $t_a \leq \frac{t}{2}$ según la suposición descrita anteriormente, el discriminante polinomial siempre es negativo y su raíz genera tres valores imaginarios que finalmente producen tres raíces reales, estas raíces se pueden extraer definiendo:

$$\theta = \arccos\left(\frac{R}{\sqrt{-Q^3}}\right) \tag{6.10}$$

Debido a que se estableció la condición $t_a \leq \frac{t}{2}$, el resultado de la división que encierran los paréntesis de la ecuación 6.10 siempre estará dentro del rango [-1,1]. Las soluciones reales son:

$$x_1 = 2\sqrt{-Q}\cos\left(\frac{\theta}{3}\right) - \frac{1}{3}a_2 \tag{6.11}$$

$$x_2 = 2\sqrt{-Q}\cos\left(\frac{\theta+2\pi}{3}\right) - \frac{1}{3}a_2 \tag{6.12}$$

$$x_3 = 2\sqrt{-Q}\cos\left(\frac{\theta+4\pi}{3}\right) - \frac{1}{3}a_2 \tag{6.13}$$

En este caso x_1 representa el valor positivo de t_r mayor que t_a que satisface la ecuación 6.7, x_2 es un valor negativo que satisface la ecuación pero que carece de significado en este contexto y finalmente x_3 representa el valor positivo de t_r menor que t_a que también satisface la ecuación. Debido a que la ecuacíón 6.7 se utiliza en los casos donde $t_a \leq \frac{t}{2}$, el valor que nos interesa es x_1.

En la figura 6.9 podemos apreciar los resultados de utilizar las ecuaciones 6.6 y 6.11 para calcular el tiempo destinado para el *rendering* (t_r). La gráfica presenta el tiempo que consume cada cuadro cuando la cámara sigue la trayectoria lineal presentada en la figura 6.6 en un escenario con 600 instancias de tres tipos distintos de árboles. Se puede observar que el tiempo mayor se produce alrededor del cuadro 500 justo en el punto donde el *frustum* de la cámara encierra el mayor número de instancias visibles. La línea gruesa sólida es el tiempo que consume cada cuadro cuando no se utiliza TC, es decir, cuando los árboles despliegan todos los triángulos de sus troncos y hojas. En esta misma gráfica se muestran varias pruebas variando el tiempo máximo que puede consumir el cuadro (t) (0.0125, 0.025, 0.05, 0.01 y 0.02 segundos), estos tiempos se representan con las líneas punteadas horizontales que atraviesan la gráfica, finalmente el tiempo real que consumieron los cuadros durante estas pruebas se representa con la línea solida delgada.

Se puede observar que el tiempo real en cada una de las pruebas se aproxima mucho al tiempo máximo que puede consumir el cuadro (t). En la tabla 6.1 se puede apreciar que los errores promedio de cada una de estas pruebas son todos menores al 20 %.

A pesar de que los resultados de varias pruebas similares a las que se presentan en la figura 6.9 fueron bastante aceptables, todos ellos presentaban un incremento en el error promedio cuando el valor de t se acercaba mucho al valor de t_a. El motivo radica en que en estos casos es necesario

Figura 6.9: Tiempo crítico con las ecuaciones 6.6 y 6.11

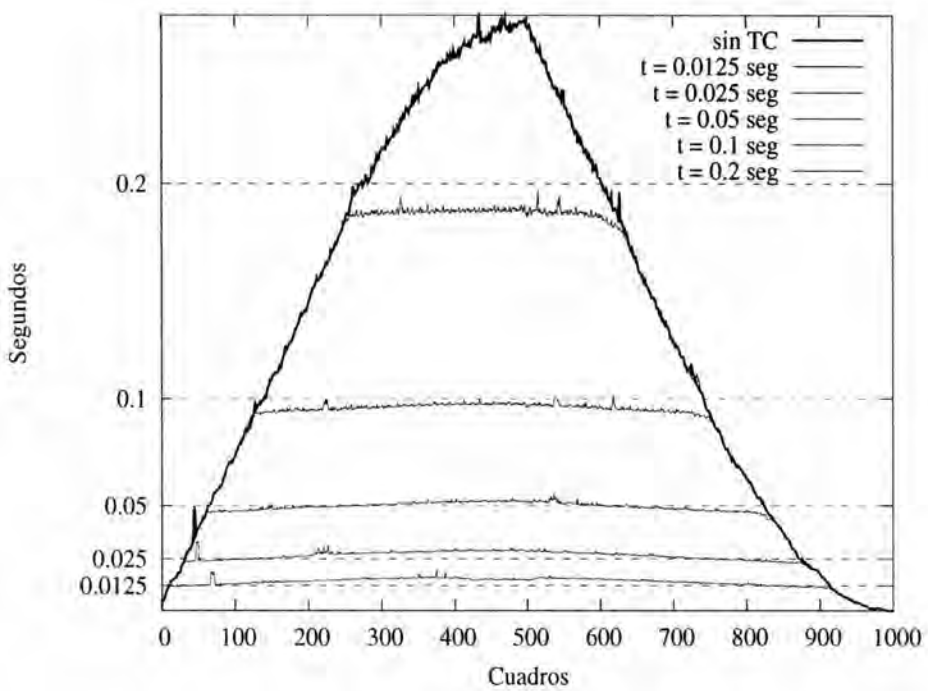


Tabla 6.1: Trayectoria lineal de TC con las ecuaciones 6.6 y 6.11

t	Media	Desv. estándar	Error promedio
0.0125	0.014512	0.001397	0.168387
0.025	0.026683	0.001831	0.082632
0.05	0.049760	0.001761	0.030140
0.1	0.095946	0.001570	0.040623
0.2	0.187236	0.001458	0.063822

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

utilizar tanto la ecuación 6.6 como la ecuación 6.11 puesto que existen momentos en que t_r es mayor que t_a pero al incrementarse el número de instancias visibles y mantener el valor de t constante, el valor de t_a se incrementa e inclusive llega a ser mayor que t_r por lo cual se utiliza la ecuación 6.6. Es precisamente en el momento del cambio de la ecuación que se utiliza para estimar el valor de t_r cuando se producen saltos drásticos en el valor real.

Supongamos que $t = 10$ y $t_a = 5$, dado que $t_a \leq \frac{t}{2}$ se utiliza la ecuación 6.11 para calcular el tiempo total que puede consumir el *rendering* del cuadro (t_r), al resolver la ecuación se obtiene que $t_r = 8$. Sin embargo supongamos que el número de instancias visibles aumenta y el valor de t_a cambia, ahora $t_a = 5.1$, debido que $t_a > \frac{t}{2}$ la ecuación para determinar el valor de t_r ahora es la 6.6, al resolver para estos valores se obtiene que $t_r = 5$. Este cambio repentino del valor 8 a 5 que sufre t_r produce saltos inesperados en el tiempo del cuadro.

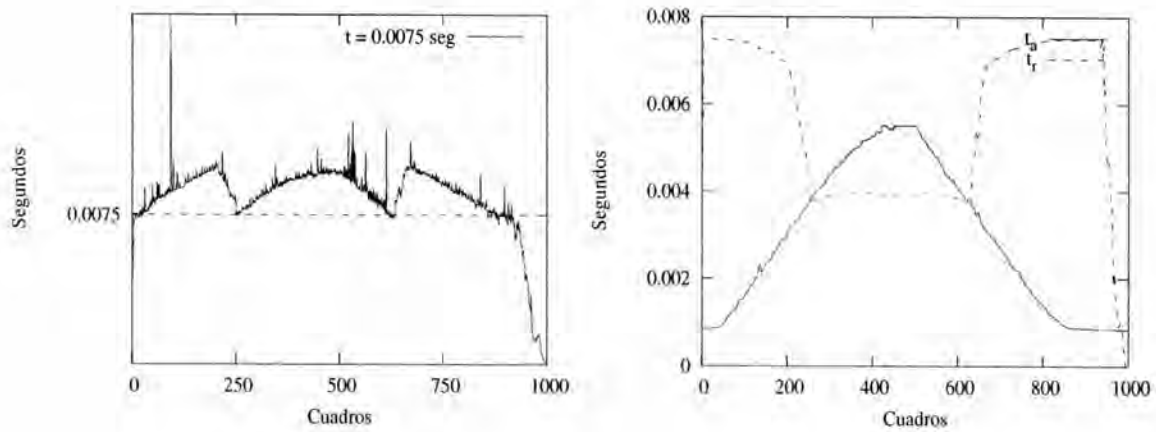
Para tratar de solucionar este problema se probaron varias alternativas como son: a) utilizar la ecuación 6.11 siempre que fuera posible definir un valor correcto para θ en la ecuación 6.10 con el fin de que la raíz x_1 tuviera la oportunidad de reducir su valor suavemente; ó b) interpolar los valores de las raíces x_1 y x_3 conforme el valor de t_a se acerca más a $\frac{t}{2}$. La gráfica izquierda de la figura 6.10 muestra el tiempo real que consumieron los cuadros para la misma escena y trayectoria mostrada en la figura 6.9, en este caso el tiempo máximo que puede consumir el cuadro (t) fue 0.0075 segundos, en la gráfica derecha de esta misma figura se muestra el valor de t_a y t_r utilizando la alternativa de solución (b) planteada en este párrafo. Note que en los momentos en que el valor de t_a se aproxima mucho al valor de t se presenta el cambio repentino en el valor estimado para t_r.

A pesar de probar distintas alternativas las ecuaciones 6.6 y 6.11 seguían presentando el problema descrito anteriormente, por este motivo fue necesario idear otra ecuación. Esta ecuación es:

$$u^4 t_a^4 - 2u^2 t_a^2 - Cut_r + 1 = 0 \qquad \text{donde } u = \frac{1}{t} \tag{6.14}$$

En la ecuación anterior C es una constante que ayuda a ajustar algunos errores como son el error de la aproximación del área que proyecta cada elemento del árbol (ver sección 4.2) o el cambio del tiempo promedio de cada primitiva al cambiar el número de instancias (ver sección 6.2.1). La ecuación 6.14 surgió del siguiente razonamiento:

Figura 6.10: Comportamiento de las ecuaciones 6.6 y 6.11



$$A = \left(\frac{t_a}{t}\right)^2 \tag{6.15}$$

El valor A en la ecuación anterior es similar al porcentaje que representa t_a de t, sin embargo a sido elevado al cuadrado para tratar de disminuir su importancia cuando t_a represente un porcentaje muy pequeño. Note que el valor máximo de A es 1 puesto que las restricciones del algoritmo establecen que t_a nunca será mayor que t. Posteriormente se calcula un valor equivalente para t_r:

$$R = 1 - A \tag{6.16}$$

En este caso R representa el porcentaje necesario para completar el tiempo del cuadro. Finalmente para calcular el valor de t_r:

$$t_r = \frac{R^2 t}{C} \tag{6.17}$$

En la ecuación anterior R se eleva al cuadrado por la misma razón que se elevó al cuadrado el porcentaje de t_a en la ecuación 6.15. Si R tiene un valor cercano a 1 al elevarlo al cuadrado su valor no se verá muy afectado, pero si su valor es cercano a cero al elevarlo al cuadrado se hará aún más diminuto y el valor de t_r será insignificante. Eliminando las variables intermedias R y A las ecuaciones 6.15, 6.16 y 6.17 se simplifican en la ecuación 6.14.

Figura 6.11: Comportamiento de las ecuación 6.14

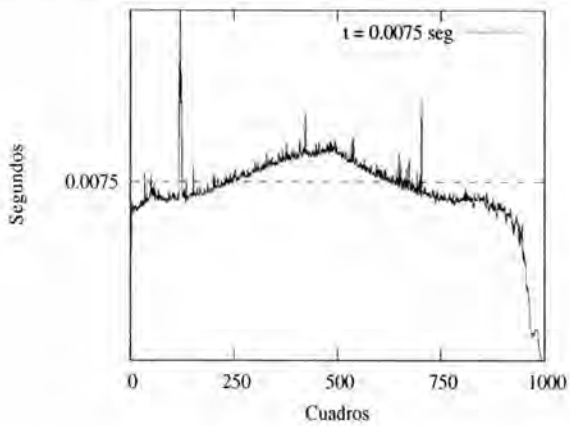


Tabla 6.2: Trayectoria lineal de TC con la ecuación 6.14

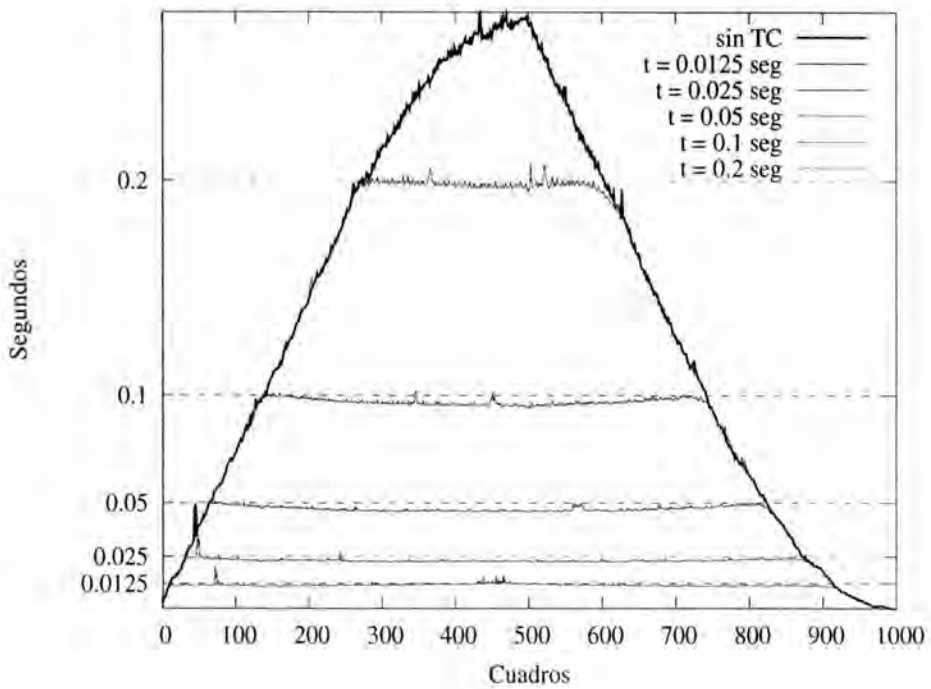
t	Media	Desv. estándar	Error promedio
0.0125	0.011791	0.000606	0.065083
0.025	0.023228	0.000830	0.073824
0.05	0.047253	0.001121	0.055051
0.1	0.097081	0.001478	0.029591
0.2	0.198541	0.001742	0.009370

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

Al repetir la prueba que se presentó en la figura 6.10 utilizando la ecuación 6.14 en lugar de las ecuaciones 6.6 y 6.11, se produjo el resultado que se muestra en la figura 6.11. Como se puede observar ya no se presentan los saltos drásticos en el tiempo que consume el cuadro y los errores también son menores.

De la misma forma se repitieron las pruebas que se mostraron en la figura 6.9 pero en esta ocasión utilizando la ecuación 6.14, los resultados de esta prueba se muestran en la figura 6.12. Se puede observar que el tiempo real se aproxima mejor al tiempo deseado al utilizar esta ecuación, el valor de C en la ecuación 6.14 permaneció constante en todas las pruebas que se efectuaron y aun así el tiempo real del cuadro se aproximo al tiempo deseado de igual forma. La tabla 6.2 muestra los errores que se produjeron en estas pruebas al utilizar la ecuación 6.14, al comparar estos valores con los que se presentaron en la tabla 6.1 se puede apreciar que esta ecuación logra reducir los errores promedio.

Figura 6.12: Trayectoria lineal en TC con la ecuación 6.14



Debido a los buenos resultados que se obtuvieron con la ecuación 6.14 se utilizó en las implementaciones y pruebas restantes que se presentan en este capítulo. La figura 6.13 muestra los resultados de estas mismas pruebas pero en esta ocasión siguiendo la trayectoria zigzag que se mostró en la figura 6.6, los errores se describen en la tabla 6.3. Note que cada vez que la cámara alcanza un extremo del escenario y gira (cuadros 562 y 1123), el número de instancias visibles y el tiempo que consume el cuadro se incrementa repentinamente, cualquier algoritmo reactivo hubiera tardado mucho tiempo en identificar el cambio pero debido a que el algoritmo es completamente pro-activo estos incrementos no afectan su precisión.

La figura 6.14 presenta varias imágenes de la escena que se utilizó para generar las gráficas mostradas en esta sección, estas imágenes muestran la misma escena variando el tiempo máximo que puede consumir el cuadro (t). Conforme se va reduciendo el tiempo los árboles lucen cada vez más incompletos, los primeros en verse afectados son aquellos que proyectan menos área, es decir, los más distantes. Cuando el tiempo es muy limitado también los árboles en primer plano se ven afectados, este comportamiento lo dicta la función de distribución de valor descrita en la sección 6.4.2.

Figura 6.13: Trayectoria zigzag en TC con la ecuación 6.14

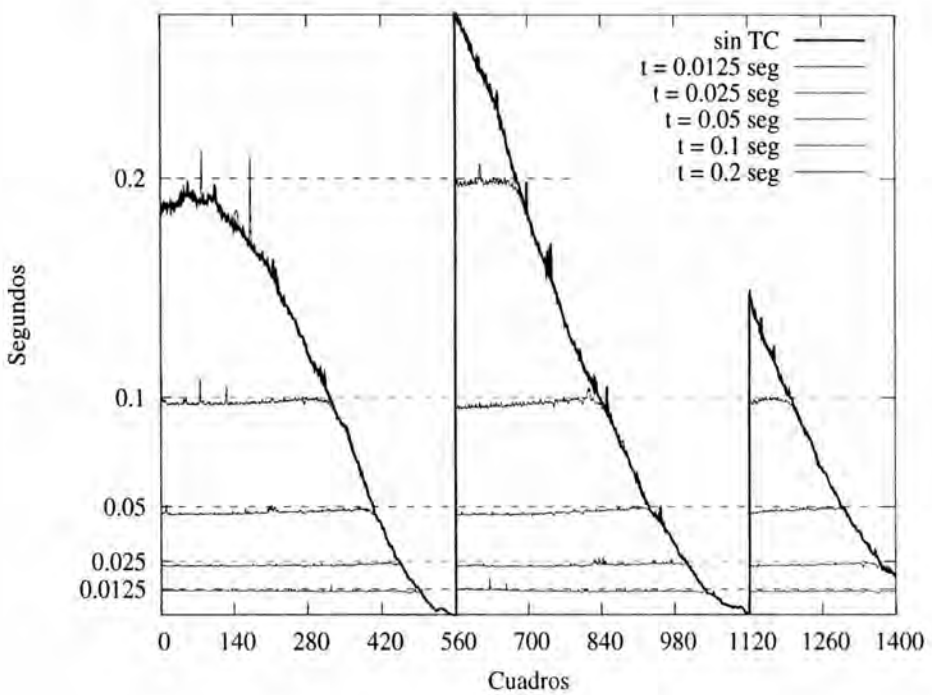


Tabla 6.3: Trayectoria zigzag de TC con la ecuación 6.14

t	Media	Desv. estándar	Error promedio
0.0125	0.011606	0.000391	0.073498
0.025	0.023329	0.000545	0.067375
0.05	0.047577	0.001012	0.048548
0.1	0.097725	0.001561	0.024280
0.2	0.198201	0.001725	0.010892

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

Figura 6.14: Efecto del TC con distintas restricciones de tiempo

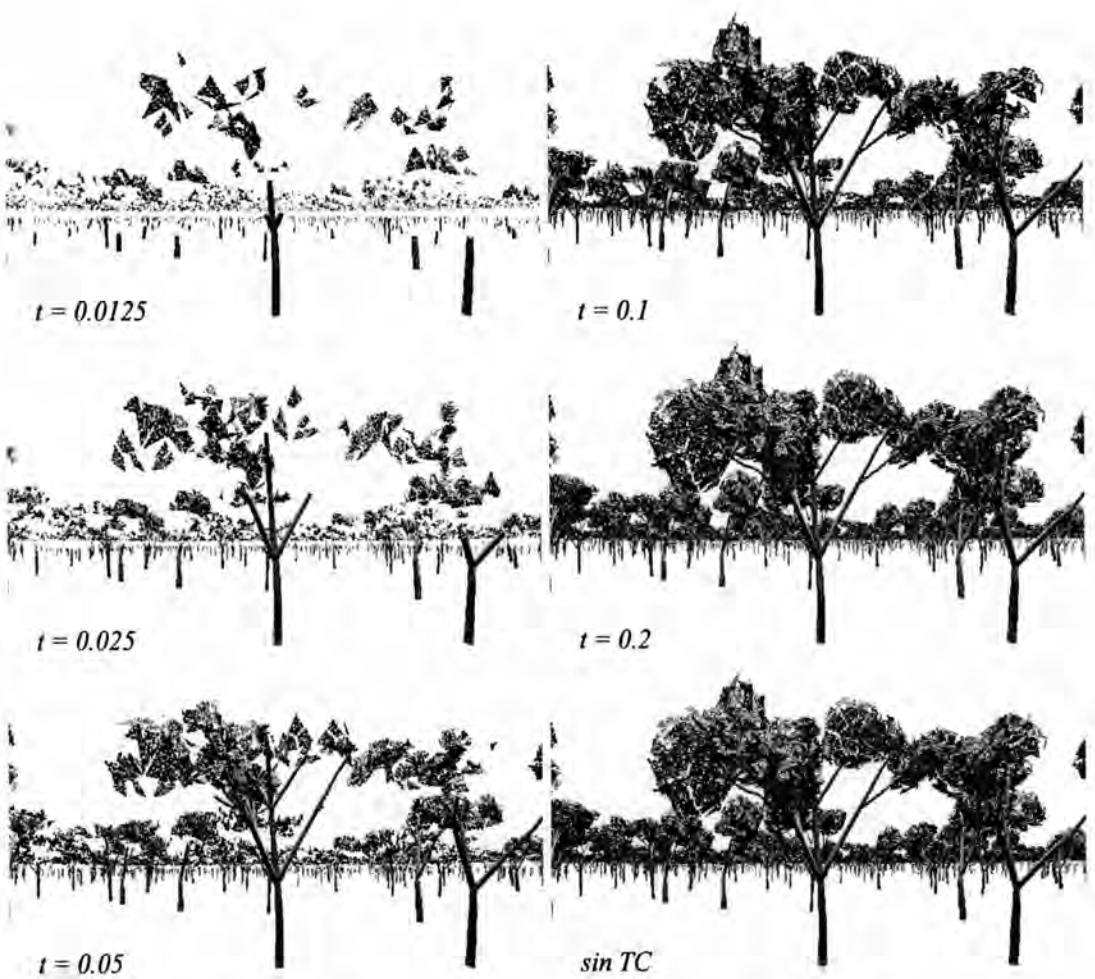


Tabla 6.4: Trayectoria lineal para varias escenas

Instancias	t	Media	Desv. estándar	Error promedio
	0.0125	0.011791	0.000606	0.065083
	0.025	0.023228	0.000830	0.073824
600	0.05	0.047253	0.001121	0.055051
	0.1	0.097081	0.001478	0.029591
	0.2	0.198541	0.001742	0.009370
	0.01875	0.017745	0.000638	0.055957
	0.0375	0.034653	0.000798	0.075942
1000	0.075	0.070469	0.001707	0.060858
	0.15	0.144267	0.002380	0.038403
	0.3	0.295999	0.002948	0.014860
	0.03125	0.030129	0.001657	0.057159
	0.0625	0.057915	0.001225	0.073595
2000	0.125	0.117363	0.002915	0.061286
	0.25	0.240084	0.005598	0.041146
	0.5	0.491082	0.005479	0.018939

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

Finalmente para verificar que el desempeño de la ecuación 6.14 es invariante también al tipo de escena, se efectuaron las pruebas de las figuras 6.12 y 6.13 en distintas escenas con distintas características, en todas estas pruebas no se alteró el valor de la constante C y aun así las gráficas resultantes fueron muy similares a las de las figuras 6.12 y 6.13. Las tablas 6.4 y 6.5 presentan una comparación de los errores de la escena mostrada en las gráficas anteriores que contiene 600 instancias de árboles, contra dos escenas con 1000 y 2000 instancias. Debido a que el tamaño del territorio es el mismo, la densidad de árboles en las escenas con 1000 y 2000 instancias es mucho mayor y sin embargo la tabla 6.4 muestra resultados muy similares para los tres casos.

6.4.2. Distribución de valor, importancia y residuo

Como se mencionó anteriormente la función de distribución de valor es bastante flexible y puede cambiar dependiendo del propósito de la aplicación. Dado que el propósito del algoritmo es controlar la calidad de la imagen resultante, la función generalmente dependerá del área que proyectan los objetos. Las gráficas que se mostraron en la sección 6.4.1 utilizan la siguiente función de distribución de valor:

Tabla 6.5: Trayectoria zigzag para varias escenas

Instancias	t	Media	Desv. estándar	Error promedio
	0.0125	0.011606	0.000391	0.073498
	0.025	0.023329	0.000545	0.067375
600	0.05	0.047577	0.001012	0.048548
	0.1	0.097725	0.001561	0.024280
	0.2	0.198201	0.001725	0.010892
	0.01875	0.017513	0.000680	0.069464
	0.0375	0.034765	0.000826	0.073179
1000	0.075	0.070968	0.001586	0.054123
	0.15	0.145539	0.002078	0.030134
	0.3	0.296794	0.003143	0.013521
	0.03125	0.029327	0.001198	0.066839
	0.0625	0.058149	0.001737	0.071883
2000	0.125	0.117955	0.002644	0.056803
	0.25	0.242108	0.004880	0.033060
	0.5	0.494490	0.006414	0.014025

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

$$Valor = \frac{Area}{\sqrt{Profundidad}} \tag{6.18}$$

El área que se menciona en la función anterior es el área del objeto sin proyectar. A pesar de que la función anterior asigna un mayor valor a aquellas instancias con una menor profundidad con respecto a la cámara, la diferencia del valor entre una instancia cercana y una lejana no es tan drástico, esto se debe a que la profundidad se ve afectada por la raíz cuadrada. Además de que esta función distribuye muy bien los recursos y minimiza el residuo, puede ser útil para aquellas aplicaciones en las cuales los objetos distantes son también importantes como por ejemplo un simulador de *golf* o un juego de tiro al blanco.

Por otro lado si la aplicación quiere asegurarse de que los objetos en primer plano mantengan una buena calidad visual aun cuando los objetos distantes se vean drásticamente afectados se puede utilizar alguna de las siguientes funciones:

$$Valor = \frac{Area}{Profundidad} \tag{6.19}$$

$$Valor = \frac{Area}{Profundidad^2} \tag{6.20}$$

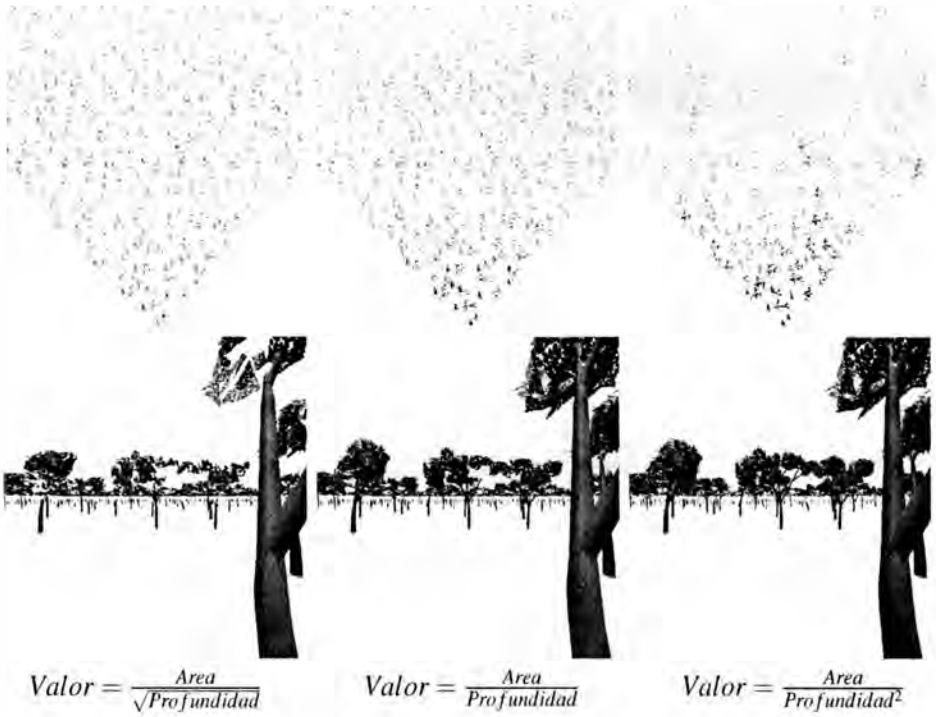
Las dos ecuaciones anteriores generan una diferencia drástica entre el valor que se le asigna a una instancia cercana y el valor que se le asigna a una instancia lejana, con esto se asegura que las instancias cercanas se desplieguen siempre con una buena calidad. En especial la función 6.20 corre el riesgo de asignar un valor muy grande a una instancia si ésta se encuentra muy cerca de la cámara y por lo tanto la instancia recibirá un tiempo muy grande para su *rendering*; si esta instancia no utiliza todo el tiempo que se le ha asignado se generará un residuo muy grande y si por casualidad esta es una instancia que pertenece a un objeto complejo y es de las últimas en desplegarse, el residuo quedará sin uso y se producirá una diferencia amplia entre el tiempo real del cuadro y el tiempo deseado. Este tipo de errores se producen cuando hay pocas instancias visibles y alguna de ellas esta muy cerca de la cámara. Si se opta por utilizar la función 6.20 o alguna similar, es recomendable que la aplicación siempre ofrezca una opción para mejorar el *rendering* de las instancias, de esta forma si alguna de ellas recibe un tiempo muy grande, la aplicación podrá aumentar el número de vértices, agregar efectos de textura como *bump mapping*, calcular sombras, utilizar *antialiasing*, etc. con el fin de reducir el tiempo sobrante y utilizar de mejor forma los recursos.

La figura 6.15 muestra el impacto visual que tienen las funciones 6.18, 6.19 y 6.20 en la misma escena, en la vista superior se aprecia la forma en como las instancias lejanas se ven afectadas por la distribución de valor mientras que la vista desde la perspectiva de la cámara muestra el impacto de las funciones en las instancias cercanas.

El factor de importancia puede utilizarse para aumentar o disminuir el valor de un objeto según la importancia que tenga para el contexto de la aplicación o se puede utilizar también para hacer ajustes en el valor debido a la forma del objeto, las hojas de los árboles en esta implementación son un ejemplo de ambos casos.

El valor de todas las hojas de un árbol es la suma del valor de cada uno de sus grupos de hojas (ver sección 2.4), donde cada grupo tiene su propia área y profundidad, sin embargo es necesario restar importancia al valor de los grupos puesto que los grupos de hojas frontales suelen cubrir a

Figura 6.15: Función de distribución de valor



los grupos que están detrás y por lo tanto no todos son visibles desde un punto de vista y algunos de ellos aportan muy poco valor a la imagen final. En segundo lugar es necesario reducir la importancia de las hojas puesto que tienen una importancia visual menor al tronco, si un árbol se muestra con el 50 % de hojas que tiene en su representación sin LOD posiblemente se vea poco frondoso pero sigue pareciendo un árbol, en cambio si se despliega con sólo el 50 % de los triángulos que forman su tronco, las hojas estarán desconectadas del pedazo de tronco que se alcanzó a desplegar y la forma distará mucho de un árbol. Por estos motivos el factor de las hojas en esta implementación en particular es de $\frac{1}{16}$, sin embargo pruebas con factores de $\frac{1}{8}$ y $\frac{1}{4}$ presentaron comportamientos numéricos muy similares pero resultados visuales distintos, esto indica que también el factor de importancia es flexible y se puede ajustar a las necesidades de la aplicación.

En resumen, el valor de cada instancia es igual al valor que determina la función de distribución de valor por el factor de importancia.

6.5. Nivel de detalle en tiempo crítico

Esta sección describe la implementación del LOD presentado en el capítulo 4 en tiempo crítico. El propósito de la aplicación es utilizar el LOD en aquellos casos donde el tiempo máximo para la generación del cuadro (t) sea limitado y de esta forma ofrecer un resultado similar al que se produce cuando no hay restricciones en el tiempo del cuadro.

El uso de nivel de detalle se controla modificando el ancho máximo y mínimo de línea y el tamaño máximo y mínimo de puntos que la aplicación puede utilizar para desplegar la geometría. Si después de probar todas las opciones posibles la aplicación es incapaz de encontrar un LOD que se ajuste al tiempo que se le ha asignado a la instancia, utilizará el LOD que se aproxime mejor a la restricción de tiempo y lo desplegará en TC, es decir, omitiendo la geometría que exceda el tiempo asignado.

La figura 6.16 muestra tres variantes de una misma escena, la imagen superior presenta la escena sin TCLOD, la imagen central muestra la escena cuando el rango de tamaños de puntos disponibles es [1, 20] y el rango de anchos de líneas disponibles es [1, 20], por último la imagen inferior muestra la escena restringiendo los tamaños de puntos al rango [1, 2] y mantiene el mismo rango de anchos para las líneas que la imagen superior. A pesar de que el valor de t es el mismo para la imagen superior y la imagen inferior, en la imagen inferior es necesario aplicar TC en el *rendering* de las hojas cercanas a la cámara debido a la escasez de tamaños de puntos disponibles.

El proceso para identificar el LOD de las hojas en TC se resume en el siguiente pseudo código:

TCLOD para las hojas

```
1 Identifica la configuración de rendering.
2 Para cada grupo de hojas:
3         Calcula el tiempo que consume cada triángulo en esta configuración.
4         En base al valor que aportó el grupo calcula el tiempo que se le
5         asigna.
6         Resta el tiempo asignado del tiempo total del rendering del cuadro.
7         Resta el valor del grupo del valor total del cuadro.
8         Incrementa el tiempo asignado con el residuo que han dejado otros
9         grupos de hojas de la misma instancia.
```

Figura 6.16: Uso de puntos y TC

```
10          Calcula el número máximo de triángulos que se pueden desplegar con
11          el tiempo asignado.
12          Si el número máximo de triángulos es mayor que el número de hojas
13          en el grupo:
14                  Despliega este grupo de hojas sin LOD.
15                  Guarda el tiempo sobrante para que sea utilizado por los
16                  grupos de hojas restantes de esta misma instancia.
17          En caso contrario:
18                  Busca algún LOD adecuado para este grupo
19                  Si no encuentras ninguno:
20                          Despliega este grupo en TC.
21                  Guarda el tiempo sobrante para que sea utilizado por los
22                  grupos de hojas restantes de esta misma instancia.
23  Añade el sobrante de tiempo que dejaron las hojas al tiempo total del
24  rendering del scuadro.
```

Note que el sobrante que dejan los grupos de hojas (líneas 15 y 21) se utiliza para el *rendering* de los demás grupos de hojas que pertenecen a la misma instancia de árbol (línea 8) puesto que estamos considerando todas las hojas como un solo elemento, el sobrante total que dejan las hojas se acumula a t_r únicamente cuando se han desplegado todas las hojas (línea 23). Cuando se dibuja el grupo de hojas en TC (línea 20) los triángulos corresponden a las hojas más lejanas del centro del grupo, de esta forma disminuye el impacto visual que provoca la ausencia de hojas.

La búsqueda del LOD que se menciona en la línea 18 efectúa las siguientes tareas:

Buesqueda de LOD para las hojas

```
1   Para cada tamaño de punto disponible:
2           Calcula el número de puntos requeridos.
3           Si es menor que el número de hojas:
4                   Si el tiempo que consume este número de puntos es menor o
5                   igual que el tiempo asignado a la instancia:
```

```
                Despliega este número de puntos.
                Regresa verdadero.
        En caso contrario, si este es el último tamaño de punto
        disponible:
                Despliega únicamente el número de puntos que quepan
                en el tiempo asignado al grupo de hojas.
                Regresa verdadero.
En caso contrario, si es menor que dos veces el número de hojas
en el grupo:
        El número de puntos requeridos será dos veces el número de
        hojas en el grupo menos el número de puntos que se había
        calculado.
        El número de triángulos requeridos será el número de hojas
        en el grupo menos el nuevo número de puntos requeridos.
        Si el número de triángulos requeridos es menor que el
        número máximo de triángulos que se pueden dibujar con el
        tiempo asignado:
                Si el tiempo que consume la combinación de puntos y
                triángulos requeridos es menor o igual al tiempo
                asignado al grupo de hojas:
                        Despliega este número de puntos y este
                        número de triángulos.
                        Regresa verdadero.
                En caso contrario, si este es el último tamaño de
                punto disponible:
                        Disminuye el número de puntos requeridos
                        que sea necesario hasta que se satisfaga
                        la restricción de tiempo para este grupo
```

```
34                                  de hojas.
35                                  Despliega este número de puntos y este
36                                  número de triángulos.
37                                  Regresa verdadero.
38  Regresa falso.
```

Como se puede observar el procedimiento para encontrar un LOD es casi idéntico al que se mostró en la sección 4.4.2 con las siguientes excepciones: a) se utilizan varios tamaños de puntos simultáneos (línea 1); y b) aun cuando no se encuentre un LOD completamente adecuado, cuando se alcanza el tamaño de punto máximo se trata de ajustar el LOD para que cumpla con el tiempo asignado al grupo de hojas (líneas 8 y 29). De la misma forma como se describió en la sección 4.4.2 cuando se cumple la condición de la línea 4 o 8 se dibujan únicamente los puntos correspondientes a las hojas más alejadas del centro del grupo de hojas. Por otro lado cuando se cumple la condición de la línea 23 o 29 los puntos corresponden a las hojas más cercanas al centro del grupo, de tal forma que queden cubiertos por las hojas lejanas que se dibujan como triángulos. Note que la condición en la línea 20 asegura que el LOD encontrado tenga posibilidades de cumplir con el tiempo asignado.

En el caso del tronco el procedimiento es el siguiente:

TCLOD para el tronco

```
1   Identifica la configuración de rendering.
2   Calcula el tiempo que consume cada triángulo en esta configuración.
3   En base al valor que aportó el tronco calcula el tiempo que se le asigna.
4   Resta el tiempo asignado del tiempo total del rendering del cuadro.
5   Resta el valor del tronco del valor total del cuadro.
6   Calcula el número máximo de triángulos que se pueden desplegar con el
7   tiempo asignado.
8   Si el número máximo de triángulos es mayor que el número de triángulos en
9   el tronco:
10          Despliega el tronco sin LOD.
11          Añade el sobrante de tiempo al tiempo total del rendering.
```

```
12 En caso contrario:
13         Busca algún LOD adecuado.
14         Si no encuentras ninguno:
15                 Despliega el tronco en TC.
16         Añade el sobrante de tiempo al tiempo total del rendering.
```

El proceso es casi idéntico al que se mostró para los grupos de hojas con excepción de que existe un solo objeto que es el tronco y no hay necesidad de iterar entre varios objetos como son los grupos de hojas. La búsqueda del LOD para el tronco que se menciona en la línea 13 realiza las siguientes tareas:

Buesqueda de LOD para el tronco

```
1  Si el segmento más delgado no se puede representar con la línea más gruesa:
2          Regresa falso.
3  Para cada ancho de línea disponible:
4          Identifica el primer y último segmento que se pueda dibujar con el
5          ancho de línea actual.
6          Si al menos un segmento se puede dibujar con este ancho:
7                  Calcula la longitud proyectada por los segmentos asociados
8                  a este ancho.
9                  Estima el tiempo que consume dibujar estos segmentos con
10                 esta longitud y este ancho.
11                 Añade este tiempo al tiempo que consume el LOD del tronco.
12         Si el número de segmentos sin LOD es menor al número máximo de
13         segmentos que se pueden dibujar usando triángulos con el tiempo
14         asignado al tronco:
15                 Si el tiempo que consumen los segmentos con LOD y aquellos
16                 sin LOD es menor que el tiempo asignado al tronco:
17                         Maximiza el número de segmentos que se dibujan sin
18                         LOD.
```

```
19                          Dibuja los segmentos con LOD usando líneas y
20                          aquellos sin LOD usando triángulos.
21                          Regresa verdadero.
22  Si el número de segmentos sin LOD es mayor al número máximo de segmentos
23  que se pueden dibujar usando triángulos con el tiempo asignado al tronco:
24          Regresa falso.
25  Para cada ancho de línea disponible:
26          Elimina los segmentos asociados a este ancho de línea que sean
27          necesarios para cumplir con la restricción de tiempo.
28          Disminuye del tiempo que consumen los segmentos con LOD el tiempo
29          de los segmentos eliminados.
30          Si el tiempo que consumen los segmentos con LOD y aquellos sin LOD
31          es menor que el tiempo asignado al tronco:
32                  Dibuja los segmentos con LOD usando líneas y aquellos sin
33                  LOD usando triángulos.
34                  Regresa verdadero.
```

El procedimiento para buscar un LOD para el tronco es muy similar al que se mostró en la sección 4.4.1, la diferencia es que el número de anchos de líneas que se va a utilizar lo restringe el tiempo que se asignó al tronco. La validación en la línea 1 nos asegura que al menos un segmento pueda ser representado con LOD. El ciclo que comienza en la línea 3 busca un LOD probando con todos los anchos de líneas disponibles. Cuando algún LOD requiere de un número de segmentos sin LOD que es menor o igual al número máximo de segmentos que se pueden dibujar usando únicamente triángulos con el tiempo asignado (línea 12), es probable que este LOD satisfaga la restricción de tiempo (línea 15).

Si un LOD cumple con la restricción de tiempo es necesario maximizar el número de segmentos que se dibujan sin LOD para reducir el residuo. Debido a que cada ancho de línea detecta todos los segmentos que puede dibujar (línea 4), es probable que incluya un mayor número de segmentos del que realmente se requieren para cumplir con la restricción del tiempo y dado que una vez que se

cumple la condición en la línea 15 el proceso termina (línea 21) podemos saber que ninguno de los anchos de líneas anteriores satisfacen la restricción de tiempo, por lo tanto el número de segmentos a maximizar esta dentro del rango de segmentos asociados al ancho de línea actual. Supongamos que $lod(x)$ es una función que devuelve el tiempo que consumen los segmentos dibujados con líneas hasta el ancho de línea x, si el ancho de línea actual es n y el tiempo total asignado al tronco es f, se puede definir la siguiente ecuación:

$$f - lod(n-1) = Tt_n + Ll_n \tag{6.21}$$

En la ecuación anterior t_n es el número de segmentos que se dibujan con triángulos y l_n es el número de segmentos que se dibujan con el ancho de línea actual (n). El valor T es el tiempo que consume dibujar un segmento con triángulos y el valor L es el tiempo que consume dibujar un segmento con el ancho de línea actual. La ecuación 6.21 indica que el tiempo disponible, una vez que se han dibujado los segmentos asociados a los anchos de líneas anteriores, debe ser suficiente para contener t_n segmentos dibujados como triángulos y l_n segmentos dibujados como líneas con el ancho actual. Además de la ecuación 6.21 existe otra restricción:

$$t_n + l_n = t_{n-1} \tag{6.22}$$

En la ecuación anterior t_{n-1} es el número de segmentos que se dibujaban como triángulos al utilizar todos los anchos de línea anteriores al actual. Dada estas dos ecuaciones podemos despejar para t_n de la siguiente forma:

$$Tt_n + Ll_n - f + lod(n-1) = 0 \tag{6.23}$$

Despejando l_n de la ecuación 6.22:

$$l_n = t_{n-1} - t_n$$

Substituyendo l_n en la ecuación 6.23:

$$Tt_n + L(t_{n-1} - t_n) - f + lod(n-1) = 0$$

$$Tt_n + Lt_{n-1} - Lt_n - f + lod(n-1) = 0$$

$$(T-L)t_n + Lt_{n-1} - f + lod(n-1) = 0$$

$$t_n = \frac{-Lt_{n-1} + f - lod(n-1)}{T-L} \tag{6.24}$$

Debido a que no hay medios segmentos, la parte entera del valor t_n en la ecuación 6.24 representa el número máximo de segmentos que podemos dibujar usando triángulos.

Por otra parte esta implementación utiliza la siguiente función de distribución de valor:

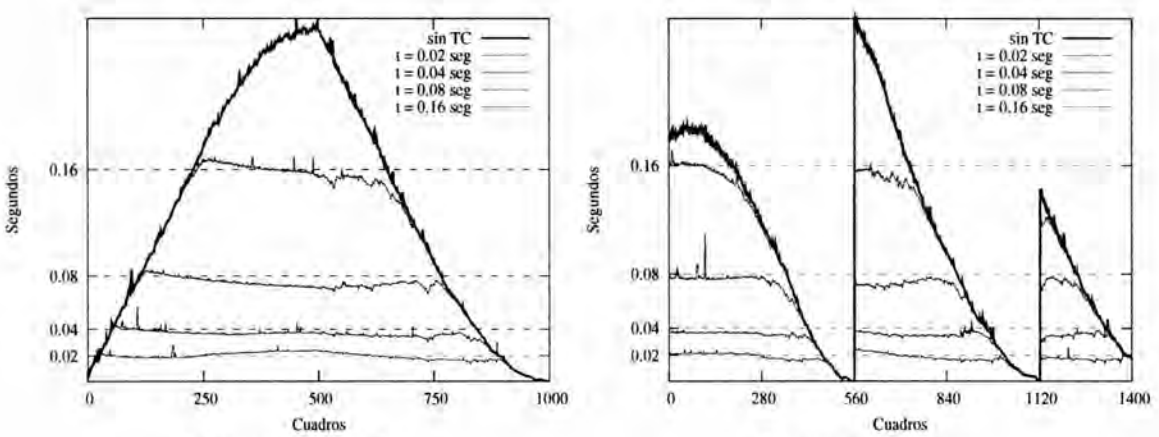
$$Valor = \sqrt{\frac{Area}{Distancia^2}} \tag{6.25}$$

Note que en la función anterior se esta utilizando la distancia en lugar de la profundidad, esto permite asignar más valor a los objetos que están en el centro del área de dibujo y por ende mejorar su calidad visual. La raíz cuadrada ayuda a reducir el error que se comentó en la sección 6.4.2 donde se le asignaba una gran cantidad de recursos a una instancia muy cercana a la cámara. Los factores de importancia permanecieron sin cambios con respecto a la implementación que se mostró en la sección 6.4.

En la figura 6.17 se muestra el tiempo que consumieron los cuadros de una escena con 600 instancias durante las dos trayectorias que se mostraron en la figura 6.6. La figura 6.18 muestra el uso de las primitivas para la prueba con un tiempo de 0.02 segundos de la figura 6.17, note que el uso de LOD se incrementa cuando el tiempo que consume el cuadro sin TC dista mucho del tiempo máximo permitido en la prueba que es 0.02 segundos. Aun cuando el uso de las líneas parece ser menor, cada una de ellas por lo general sustituye más triángulos que los que puede sustituir un solo punto por lo cual ofrecen un gran ahorro.

Las mismas pruebas para la primera trayectoria se efectuaron en la implementación que utiliza únicamente TC usando la misma escena, misma función de distribución de valor y la misma restric-

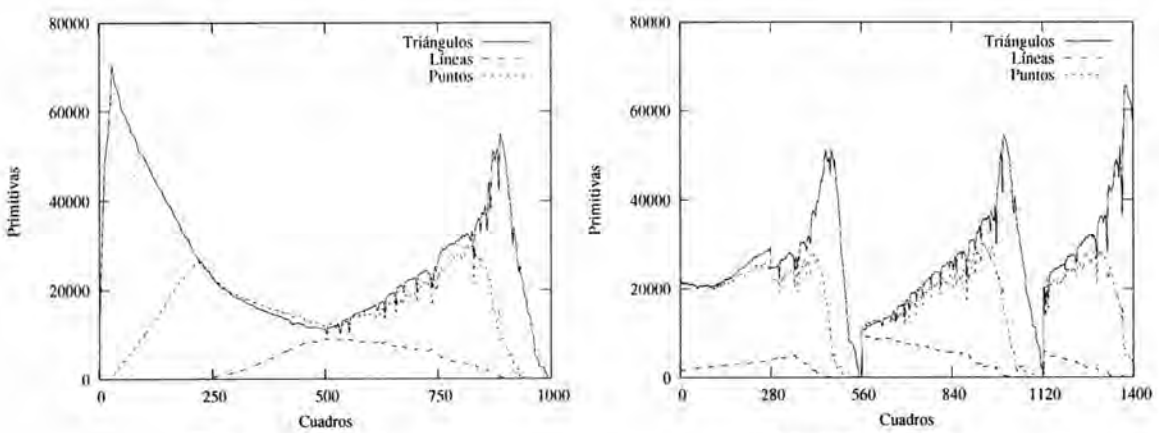
Figura 6.17: Tiempo de la escena con TCLOD



Trayectoria lineal

Trayectoria zigzag

Figura 6.18: Uso de primitivas en la pureaba con 0.02 segundos



Trayectoria lineal

Trayectoria zigzag

Tabla 6.6: Estadísticas durante la trayectoria lineal para TCLOD y TC

Aplicación	t	Media	Desv. estándar	Error promedio
TCLOD	0.02	0.020043	0.002487	0.106990
	0.04	0.036754	0.002260	0.088692
	0.08	0.074857	0.003483	0.069879
	0.16	0.158275	0.005503	0.028941
TC	0.02	0.018532	0.000934	0.077621
	0.04	0.037664	0.001360	0.059771
	0.08	0.077028	0.002216	0.039342
	0.16	0.157432	0.005302	0.026343

Todos los tiempos están en segundos, un error de 1.0 equivale un tiempo 100 % distinto de t

ción de tiempo máximo para la generación del cuadro (t). La tabla 6.6 presenta la comparación de los errores resultantes en la versión que utiliza únicamente TC y la versión con TCLOD. Note que los errores son ligeramente superiores al momento de utilizar TCLOD, algunos de los motivos son:

1. La función de distribución de valor genera residuo. Este es el problema que se comentó en la sección 6.4.2 que no sólo afecta a la versión con TCLOD sino también a la versión que utiliza únicamente TC. En este caso ninguna de las dos implementaciones ofrece algún mecanismo para utilizar el tiempo sobrante por lo cual se desperdicia y produce inexactitudes, especialmente cuando el tiempo que consume el cuadro sin usar TC se acerca al valor del tiempo máximo que puede consumir el cuadro (t) en cada prueba. A pesar de la existencia de este error, los cuadros afectados no muestran un cambio visual drástico, por ejemplo la figura 6.19 muestra la escena sin usar TC (imagen izquierda) y la escena que utiliza TCLOD (imagen derecha) cuando $t = 0.04$ segundos en el cuadro 830 de la trayectoria lineal, como se puede observar en la figura 6.17 en este momento se produce un gran sobrante de tiempo pero al haber pocas instancias la apariencia de las dos imágenes en la figura 6.19 es muy similar.
2. Se utilizan más tipos de primitivas para el despliegue de la imagen lo cual dificulta estimar de forma precisa el tiempo que consume el *rendering*.
3. Existen nuevas aproximaciones como es el cálculo de la longitud proyectada (ver sección 4.2) que introducen nuevos errores en los valores estimados.

Figura 6.19: Impacto del tiempo sobrante

Escena sin TC

Escena con TCLOD

4. No se puede medir el tiempo de la aplicación de forma precisa (este punto se trata a detalle más adelante en esta misma sección).

La figura 6.20 muestra una comparación de la misma escena, con la misma función de distribución de valor y el mismo tiempo máximo para la generación del cuadro (t) usando TC (imagen inferior) y TCLOD (imagen central). La imagen superior de esta misma figura presenta la versión de la escena sin TC. Note que la implementación de TCLOD, a pesar de tener la misma restricción de tiempo que la implementación de TC y de que el tiempo que consumen sus procesos de aplicación son mucho más tardados, presenta troncos mucho más completos y aun cunando el LOD en las hojas es evidente los árboles lucen más frondosos y el área que proyectan las hojas es más similar a la de la versión sin TC.

Aun cuando los resultados lucen aceptables esta implementación tiene una gran desventaja: debido a que la búsqueda de los niveles de detalle es un proceso ajeno al *rendering* pero dependiente de la vista de la cámara, el muestreo de la aplicación es dependiente del tiempo máximo que puede consumir la generación del cuadro (t) y presenta un comportamiento poco estable. La figura 6.21 ejemplifica esta dependencia, la línea sólida representa el tiempo que consume la aplicación cuando no se usa TCLOD, las líneas punteadas muestran el tiempo que consume la aplicación cuando se usa TCLOD variando el valor de t. Note que conforme el valor de t aumenta la aplicación reduce el uso del LOD y por lo tanto sus procesos son más rápidos.

Figura 6.20: Escena sin TC, con TCLOD y con TC

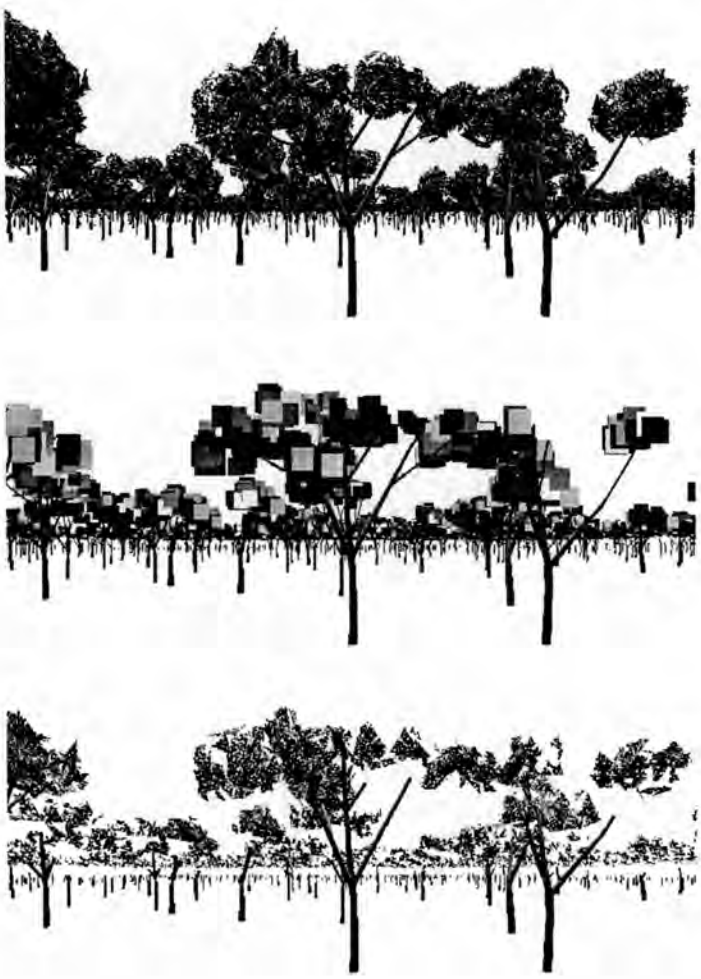
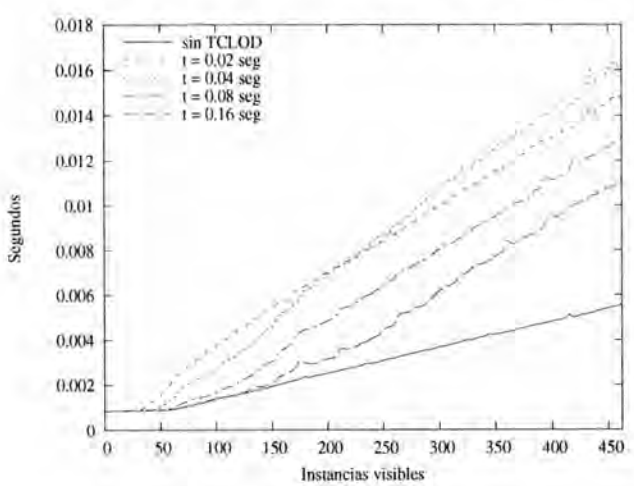


Figura 6.21: Tiempo de los procesos ajenos al *rendering* para TCLOD



En contraste con implementación de TC que se mostró en la sección 6.4 donde sólo era necesario efectuar un muestreo de los procesos de la aplicación ajenos al *rendering* el cual era válido para cualquier valor de t, esta implementación de TCLOD requiere un muestreo para cada uno de los valores de t. Por este motivo la implementación de TCLOD no puede cambiar el valor de t en el momento de la ejecución puesto que tendría que efectuar un nuevo muestreo que se ajustara al nuevo valor de t, además si existe movimiento en la escena este tipo de muestreos dejarán de ser válidos. Esta reducción de interactividad se debe a que la implementación de TCLOD utiliza un algoritmo de LOD dependiente de la vista.

Los algoritmos de LOD dependientes de la vista restringen el número de niveles de detalle disponibles en base a la relación que exista entre la cámara y el objeto, esto significa que se agrega una restricción mas a la restricción del tiempo que se le asigna a cada instancia. Conforme el número de restricciones se incrementa la búsqueda de un LOD correcto es más tardada.

Las causas mencionadas anteriormente arrojan la siguiente conclusión: es mejor evitar la combinación de algoritmos de LOD dependientes de la vista al hacer uso de esta técnica para TC puesto que la calidad visual de cada objeto ya esta implícita en la función de distribución de valor. Un algoritmo de LOD como puede ser *progressive mehes* [Hop96], en el cual todos los niveles de detalle están disponibles sin importar la relación del objeto con la cámara dará mejores resultados.

6.6. Conclusiones

La técnica que se presentó en este capítulo para efectuar el TCLOD demostró que puede ofrecer buenos resultados y que resuelve varias de las limitantes que tenían otras soluciones propuestas como son:

- ☆ A diferencia de la solución propuesta por Funkhouser y Sequin [FS93] el algoritmo de Robin Hood para TC (sección 6.3) asegura que cada objeto visible, sin importar su tamaño, recibirá un tiempo para su *rendering*. El *rendering* de los objetos que reciben poco tiempo ya no depende del algoritmo de TC sino de las capacidades del algoritmo de LOD para generar representaciones sencillas.

☆ El algoritmo de Robin Hood para TC (sección 6.3) funciona de la misma forma en escenas estáticas o con movimiento, en espacios cerrados o al aire libre. Además no esta atado a ningún tipo de LOD como son las primitivas texturizadas en la solución propuesta por Maciel y Shirley [MS95].

Además la técnica también presenta mecanismos para controlar el resultado por medio de la función de distribución de valor y los factores de importancia. Los errores que se llegan a producir se pueden anticipar por medio del residuo y utilizar el tiempo sobrante a favor de la aplicación. En conclusión, los factores críticos para el uso correcto de esta técnica son:

1. La exactitud con que se estima el área proyectada por los objetos.
2. La exactitud de los muestreos de *rendering* y los procesos de la aplicación.
3. La granularidad que ofrece el algoritmo de LOD.
4. Evitar cálculos dependientes de la vista que puedan afectar de forma drástica el muestreo de los procesos de la aplicación ajenos al *rendering*.

El trabajo a futuro se centra en mejorar los muestreos (sección 6.2) y la forma como se estima el área que proyectan los objetos en pantalla. Teniendo datos de entrada más precisos el desempeño del algoritmo de Robin Hood para TC (sección 6.3) puede ser mejor.

Lista de Figuras

Lista de Tablas

Referencias

[acm98] Líneas de investigación del ACM, 1998. http://www.acm.org/class/1998/. Ultima visita 6 de mayo del 2004.

[AMH02] AKENINE-MOLLER T., HAINES E.: *Real-Time Rendering*, second ed. AK Peters, Ltd., July 2002.

[Ben03a] BENEŠ B.: Artificial terrains. In *3D Computer Games and Real-Time Computer Graphics Applications Programming II course notes* (2003), DCSE CCM ITESM.

[Ben03b] BENEŠ B.: From computer graphics to virtual reality. In *Introduction to Computer graphics course notes* (2003), DCSE CCM ITESM.

[Bou01] BOURG D. M.: *Physics for Game Developers*, first ed. O'Reilly, November 2001.

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *Proceedings of the conference on Visualization '02* (2002), IEEE Computer Society, pp. 219–226.

[DHL*98] DEUSSEN O., HANRAHAN P., LINTERMANN B., MĚCH R., PHARR M., PRUSINKIEWICZ P.: Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), ACM Press, pp. 275–286.

[DWS*97] DUCHAINEAU M. A., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: ROAMing terrain: real-time optimally adapting meshes. In *IEEE Visualization* (1997), pp. 81–88.

[Ebe00] EBERLY D. H.: *3D Game Engine Design*, first ed. Morgan Kaufmann, September 2000.

[eki04] Eki's modpak, 2004. http://www.kolumbus.fi/erkki.halkka/. Ultima visita 12 de agosto del 2004.

[FH02] FANG H., HART J. C.: Randomly accessible procedural animation of physically approximate turbulent motion. In *Proceedings of Computer Animation '02* (June 2002), IEEE Computer Society, pp. 43–48.

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (1993), ACM Press, pp. 247–254.

[GB99] GOBBETTI E., BOUVIER E.: Time-critical multiresolution scene rendering. In *Proceedings of the conference on Visualization '99* (1999), IEEE Computer Society Press, pp. 123–130.

[GJ90] GAREY M. R., JOHNSON D. S.: *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1990.

[Hil00] HILL F. J.: *Computer Graphics Using OpenGL.* Prentice Hall PTR, 2000.

[Hop96] HOPPE H.: Progressive meshes. *Computer Graphics 30*, Annual Conference Series (1996), 99–108.

[IHTI78] IBARAKI T., HASEGAWA T., TERANAKA K., IWASE J.: The multiple choice knapsack problem. *J. Oper. Res. Soc. Japan 21* (1978), 59–95.

[KMOK03] KONDO R., MIYAJIMA S., OHIDE T., KANAI T.: Comparison of two accelerations for interactive animation of trees. In *Proceedings of the 3rd IASTED international conference on Visualization, Imaging, and Image Processing* (September 2003), ACTA Press, pp. 19–24.

[LCV03] LLUCH J., CAMAHORT E., VIVÓ R.: Procedural multiresolution for plant and tree rendering. In *Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa* (2003), ACM Press, pp. 31–38.

[MB97] MASON A. E. W., BLAKE E. H.: Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum 16*, 3 (September 1997), 191–199.

[MFC97] MARSHALL D., FUSSELL D., CAMPBELL III A. T.: Multiresolution rendering of complex botanical scenes. In *Graphics Interface '97* (1997), Davis W. A., Mantei M.,, Klassen R. V., (Eds.), Canadian Human-Computer Communications Society, pp. 97–104.

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 397–410.

[MS95] MACIEL P. W. C., SHIRLEY P.: Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (1995), ACM Press, pp. 95–ff.

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants.* Springer-Verlag New York, Inc., 1990.

[Sch83] SCHACHTER B. J.: *Computer Image Generation.* Krieger Publishing Co., Inc., 1983.

[Sch95] SCHAUFLER G.: Dynamically generated impostors. *GI Workshop on Modeling, Virtual Worlds and Distributed Graphics* (November 1995), 129–139.

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 151–162.

[SO99] SAKAGUCHI T., OHYA J.: Modeling and animation of botanical trees for interactive virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology* (1999), ACM Press, pp. 139–146.

[SSC01] SLATER M., STEED A., CHRYSANTHOU Y.: *Computer Graphics and Virtual Environments: From Realism to Real - Time.* Addison-Wesley Longman Publishing Co., Inc., 2001.

[Wei04] WEISSTEIN E. W.: Formula cúbica, 2004. http://mathworld.wolfram.com/CubicFormula.html. Ultima visita 27 de octubre del 2004.

[WS98] WIMMER M., SCHMALSTIEG D.: Load balancing for smooth lods, 1998. Technical report, Vienna University of Technology.

[xfr04] Greenworks organic software, 2004. http://www.xfrogdownloads.com/. Ultima visita 10 de agosto del 2004.

[ZMK02] ZACH C., MANTLER S., KARNER K.: Time-critical rendering of discrete and continuous levels of detail. In *Proceedings of the ACM symposium on Virtual reality software and technology* (2002), ACM Press, pp. 1–8.

Índice alfabético