Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 – Estructuras de datos y algoritmos

Informe 2

Estudiante: Felipe Baitelman

1. Estructura

a) Construcción a grandes rasgos

Durante la tarea nos enfrentamos al problema de optimizar la solución de un procedimiento de Ray Tracing. El programa que nos fue entregado al comienzo revisaba por cada rayo existente si este tocaba o no con alguno de los triángulos de la escena. Para verificar que triángulo tocaba y si esta era o no el más cercano el algoritmo tenía que iterar sobre todos los triángulo, para cada rayo.

Los objetivos de la tarea eran utilizar árboles para optimizar el algoritmo recién mencionado y aplicar un algoritmo de dividir y conquistar para enfrentar el problema geométrico. Es así como la solución al problema constaba de dos partes: por un lado generar un árbol que separase los triángulos en conjuntos de búsqueda más pequeños y por el otro crear un método para recorrer este árbol.

Decidir que tipo de árbol utilizar para optimizar la solución era un punto clave del problema. La opción más lógica era utilizar un K-d Tree. Un K-d Tree es una estructura que se utiliza para crear particiones del espacio y organizar puntos en el espacio de K dimensiones. Tienen varias aplicaciones prácticas pero la mayoría guardan relación con búsquedas en espacios multidimensionales.

Ya que nuestro problema se podía reducir, básicamente, a encontrar de forma más eficiente con que triángulo interceptaba cada rayo, un K-d Tree (específicamente un 3-d Tree) era una solución elegante y precisa para resolver el problema.

La idea de utilizar el K-d Tree era que este dividiera el espacio en cajas, y que cada caja tuviese triángulos en su interior. De esta forma, el problema ahora se trataba de ver con que cajas intersectaba el rayo, y una

vez detectada la caja más cercana al origen de este, se pasaba a iterar sobre los triángulo contenidos en dicha caja. Si el rayo intersectaba a por lo menos uno de los triángulos en la caja más cercana, el problema estaba solucionado. Si no, había que buscar la siguiente caja más cercana al origen del rayo, y así sucesivamente.

Los atributos generales del árbol son las siguientes:

- El árbol esta compuesto de nodos.
- Cada nodo tiene un hijo izquierdo, un hijo derecho y un padre.
- Cada nodo tiene un Bounding Box, caja espacial definida por dos puntos en el espacio.
- Cada nodo tiene una lista de triángulos.

La idea es generar el árbol de forma recursiva. Por lo tanto, cada nodo, dependiendo de la cantidad de triángulos en su lista, genera su hijo izquierdo y su hijo derecho. Según la profundidad en la se encuentra el nodo se calcula la mediana de los vértices de la lista de triángulos (depende de la profundidad ya que la idea es que se corte según la mediana alternando los ejes: x, y, z), y así se puede partir la caja del nodo actual en dos sub-cajas, que serán las cajas de los hijos del nodo (las dos cajas nuevas son una partición del espacio).

Cuando la cantidad de triángulos en la lista de de un nodo es menor a x el nodo deja a sus hijos como nulos y se detiene la recursión. Por otro lado, si existen muchos triángulos compartidos por los futuros hijos del nodo (triángulos que deben ir tanto al hijo izquierdo como al derecho por que son cortados por la línea de la mediana) , digamos una proporción y del total, también se detiene la recursión.

b) Propiedades del árbol

Las propiedades más importantes del árbol son:

- Es un árbol binario: Cada nodo puede generar máximo dos hijos.
- Es un árbol completo: Si un nodo tiene hijos, entonces tendrá dos (uno derecho y uno izquierdo).
- Es un árbol que puede no estar balanceado: Una rama puede tener un altura mayor a otra sin ninguna restricción.

c) Complejidad en la construcción

Siempre se utiliza *Quicksort* para ordenar los vértices de los triángulos y calcular la mediana, algoritmo que trabaja con una complejidad promedio de $\mathcal{O}(\mathrm{n}\log n)$ y de $\mathcal{O}(\mathrm{n}^2)$ en el peor caso. Además, podemos considerar que el árbol esta balanceado (aproximación) y que por lo tanto hay que ordenar todos los triángulos en cada nivel del árbol. Se N la cantidad total de triángulos en la escena. La altura del árbol será, aproximadamente $\log_2 N$. Luego, la complejidad promedio de la construcción será:

$$\mathrm{t}_{AVG} = \mathcal{O}(\mathrm{N}\log N) * \mathcal{O}(\log_2 N) = \mathcal{O}(\mathrm{N}\,(\log N)^2)$$

$$\mathrm{t}_{WORST} = \mathcal{O}(N^2) * \mathcal{O}(\log_2 N) = \mathcal{O}(\mathrm{N}^2 \log N)$$

2. Recorrer el árbol

a) Complejidad en la búsqueda

Como se menciona en la sección 1. a), la idea de utilizar el K-d Tree era que este dividiera el espacio en cajas, y que cada caja tuviese triángulos en su interior. De esta forma, el problema ahora se trataba de ver

con que cajas intersectaba el rayo, y una vez detectada la caja más cercana al origen de este, se pasaba a iterar sobre los triángulo contenidos en dicha caja. Si el rayo intersectaba a por lo menos uno de los triángulos en la caja más cercana, el problema estaba solucionado. Si no, había que buscar la siguiente caja más cercana al origen del rayo, y así sucesivamente.

Por ende, la búsqueda del triángulo indicado, en el pero caso, será incluso peor que la solución no optimizada, ya que tendremos que recorrer todos los nodos y todos los triángulos por nodo (a veces incluso habrán triángulos repetidos) y podría ocurrir que el rayo no intercepte con nada.

Sea *N* la cantidad total de triángulos. Ya que la búsqueda en un árbol binario toma $\mathcal{O}(h)$, y en el peor de los casos tendremos que buscar en todas las hojas (tenemos aproximadamente *N/x* hojas) y para cada hoja tendremos que revisar sus *x* triángulos, entonces la búsqueda nos costará en el pero de los casos:

$$\mathcal{O}\left(x * \frac{N}{x} * \log N\right) = \mathcal{O}(N \log N)$$

b) Complejidad en la búsqueda de un árbol mal formado

Digamos que en vez de utilizar un *x* razonable para detener la recursión, utilizamos $x = 1$. La complejidad de recorrer este árbol será exactamente la misma que en el caso anterior, $\mathcal{O}(N \log N)$ en el peor caso. No obstante, este árbol *en promedio* también lo recorreremos en más tiempo que $\mathcal{O}(N)$, ya que será necesario revisar la misma cantidad de triángulos que en el caso sin optimización, pero además incurrimos en el costo de recorrer el árbol.
Básicamente, siempre que formemos un árbol binario la complejidad de recorrerlo será $\mathcal{O}(N \log N)$ en el peor caso, pero lo que nos interesa, para efectos, prácticos, es mejorar el tiempo promedio.

3. Testing

La fórmula que analizaremos al hacer los test será: $t' = t/(q * n)$, donde *t* es el tiempo que tarda generar la imagen, *q* la cantidad de rayos generados y *n* la cantidad de rayos en la escena. El valor analizado, *t'*, nos representa el tiempo promedio que se demora en calcular la intersección para un rayo dividido por el número de triángulos de la escena y nos permite relacionar directamente a *n* con la demora en la resolución de intercepciones.

a) Análisis imágenes Easy y Normal

Figura 3.0[1]

Escenas vs. Tiempos				Solución original		Solución optimizada	
Tipo	Escena	n	q	factor = 1 (s)	t'	factor = 1 (s)	t'
Easy	Cube	12	282792	0.221096	6.51527E-08	0.175066	5.15886E-08
	Sphere	1280	287756	17.209414	4.67231E-08	0.412635	1.12029E-09
	Teapot	4032	314815	73.823952	5.81596E-08	2.601355	2.04939E-09
	Torus	2304	309288	39.939696	5.60479E-08	1.608741	2.25757E-09
Normal	Billiards	4164	1395207	322.433441	5.54997E-08	43.091984	7.41732E-09
	Infinicandy	140	4261851	31.945906	5.35413E-08	39.288831	6.5848E-08
	Starters	4480	776648	242.85791	6.97991E-08	19.688185	5.65853E-09
	Cornellbox	5130	1021491	325.472595	6.21101E-08	14.501113	2.76726E-09
	Metallic	7682	817456	414.524506	6.60103E-08	17.323032	2.75858E-09

[1] Sólo se realizaron pruebas con un factor igual a uno debido al alto costo de medir tiempo con factores mayores. Los tiempos desplegados son el promedio de tres pruebas distintas. Las pruebas además se computaron con *x* = 50 e *y* = 0.7.

i. Complejidad de la solución original

La complejidad de la solución original será de $\mathcal{O}(rn)$, donde r representa la cantidad de rayos y n la cantidad de triángulos. Esta complejidad es en el peor caso y en el promedio, ya que no hace falta interceptar un triángulo para saber si es el correcto, sino que siempre hay que interceptar todos los demás para saber si hay uno más cercano.

ii. Complejidad de la solución optimizada

La complejidad de la solución optimizada será de $\mathcal{O}(rn \log n)$, donde r representa la cantidad de rayos y n la cantidad de triángulos. No obstante, nuestro caso promedio es considerablemente superior al caso promedio original, donde podemos apreciar que t' para el caso optimizado es en general un orden de magnitud menor que en el caso original. Esto se debe a que en la solución optimizada podemos encontrar mucho más rápido los triángulos posibles, y además a que, una vez encontrado el triángulo más cercano en una caja, estamos totalmente que ese es "el" triángulo correcto y no puede existir uno más cercano.

La imagen que muestra la mejora más llamativa es el de la esfera, donde el tiempo original es más de 41 veces que el de la solución optimizada (lo sigue el Teapot con 28). El árbol se beneficia mucho con esta escena por que es muy simétrica (la esfera es simétrica en todos los puntos de su superficie). Por lo mismo, las cajas quedan muy bien distribuidas y el rayo encuentra al triángulo correcto casi siempre en la primera caja que revisa.

b) Análisis de Parámetros

Para efectos de esta pregunta utilizaremos las escenas Normal. No utilizamos escenas Hard debido a que nuestra solución optimizada no logra resolverlas en un tiempo considerablemente menor al algoritmo no optimizado. Esto se debe a que existen muchos triángulos y figuras complejas y el método de cortar las cajas recursivamente en la mediana no es el método óptimo para este tipo de escenas, sino que se debería haber utilizado alguna heurística, como por ejemplo SAH.

Como se menciona anteriormente en el informe, existen dos parámetros que influyen en el tiempo que se demora el Ray Tracing en computar las soluciones:

x: Mínima cantidad de triángulos que debe tener un nodo para generar hijos (proporción del total de triángulos de la imagen).

y: Máxima cantidad de triángulos compartidos entre la lista de triángulos que se van al hijo de la izquierda y los que se van al de la derecha que debe tener un nodo para generar hijos (proporción del total de triángulos en el nodo).

De la *Figura1.1* a la *Figura 1.5* se ve como varía el tiempo en función de x e y para las 5 escenas Normal. Aparece en verde el mejor tiempo de la tabla y en rojo el peor. Por último es importante notar que "no value" significa que no se pudo computar el tiempo con esa selección de parámetros.

Figura 3.1

Billiards (Triángulos totales: 4164)			
y/x	1%	5%	10%
10%	62.118736	64.548332	85.329865
50%	43.644493	61.697361	82.759232
90%	no value	60.817719	82.434143

Figura 3.2

Infnicandy (Triángulos totales: 140)			
m/n	1%	5%	10%
10%	32.716885	32.626659	32.727112
50%	32.710033	32.653099	32.484524
90%	no value	no value	61.414482

Figura 3.3

Starters (Triángulos totales: 4480)			
y/x	1%	5%	10%
10%	57.23941	57.071491	57.591507
50%	19.231804	33.228451	33.870136
90%	18.941767	33.362366	33.874027

Figura 3.4

Cornellbox (Triángulos totales: 5130)			
y/x	1%	5%	10%
10%	104.197632	125.371674	143.95343
50%	14.957147	26.241549	41.526199
90%	14.322734	26.628338	40.320377

Figura 3.5

Metallic (Triángulos totales: 7862)			
y/x	1%	5%	10%
10%	49.552898	46.225395	50.228558
50%	21.526592	46.199852	51.246326
90%	21.291544	46.276031	50.279114

De los resultados anteriores se pueden desprender las siguientes observaciones:

- Como tendencia general, los tiempos tendían a disminuir para valores pequeños de x y valores grande de y. Por otro lado, tendían a aumentar para valores grandes de x y valores pequeños de y.
- Siempre que se pudo computar el valor para x = 1% e y = 90% este fue el mejor tiempo. Si no se podía computar el valor, siempre el tiempo más corto se daba cuando x = 1% e y = 50% (con excepción de la *Figura 1.2*).
- Sólo existieron resultados no computables cuando y = 90%.

Luego, es posible analizar las observaciones obtenidas:

i. Variación de los tiempos

Los tiempos varían al modificar estos parámetros debido los valores de estos afectan en como se construye el árbol de K-d Nodos. De esta forma, propiedades del árbol mismo como su altura y cantidad de hijos variarán. Más aún, existirán variaciones con respecto a la ubicación de cada triángulo en el árbol modificado.

Por su parte, las observaciones hechas en las tablas anteriores también tienen una explicación lógica, y en particular, la tendencia general. Entre más pequeño es el valor de x, más nodos se forman en el árbol, y por ende, más cajas. Luego, al recorrer el árbol, es necesario revisar menos triángulos por cada caja. Si la escenas tiene muchos triángulos y distribuidos por el espacio, es probable encontrar el primer triángulo con el que

toca cada rayo en una de las cajas cercanas y por lo tanto no tendremos un costo alto de revisar muchas cajas.

Aumentar el valor de y cumple la misma función que disminuir el valor de x. No obstante, pudimos observar que en ciertos casos en que la valor de y era alto, no se podía computar la solución. Esto se debe a que el árbol no se puede generar correctamente y entra en un ciclo infinito, debido a que no es posible separar los triángulos de un nodo. Por lo tanto, es importante considerar este riesgo.

ii. El parámetro y las escenas

Es importante notar que en la *Figura 1.4* en el tiempo mínimo era casi 10 veces más pequeño que el tiempo máximo mientras que en la *Figura 1.2* la diferencia era marginal. Este fenómeno es atribuible a la construcción geométrica de la imagen. Si existen en escena donde los triángulos tienen una distribución geométrica extraña, o se forman cúmulos repartidos por el espacio, es posible que se den situaciones extrañas como la observada. La solución diseñada debiese ser óptima cuando los rayos llegan de forma uniforme la imagen y además los triángulos se distribuyen uniformemente. Si una escena fuerza estas condiciones mucho es posible que los resultados sean extraños.

c) Construcción del árbol vs. Recorrido del árbol

Figura 3.6[2]

Tiempos de Construcción y Generación				Tiempo de Construcción		**Tiempo de Generación**	
Tipo	Escena	n	q	factor = 1 (s)	t'	factor = 1 (s)	t'
Normal	Billiards	4164	1395207	0.030713	5.28656E-12	43.091984	7.41732E-09
	Infinicandy	140	4261851	0.000627	1.05085E-12	39.288831	6.5848E-08
	Starters	4480	776648	0.035713	1.02642E-11	19.688185	5.65853E-09
	Cornellbox	5130	1021491	0.035313	6.7388E-12	14.501113	2.76726E-09
	Metallic	7682	817456	0.055475	8.83402E-12	17.323032	2.75858E-09

i. Complejidad de la construcción esperada

Como se menciona en 1 c), la complejidad esperada en la construcción será:

$$t_{AVG} = \mathcal{O}(N \log N) * \mathcal{O}(\log_2 N) = \mathcal{O}(N (\log N)^2)$$

$$t_{WORST} = \mathcal{O}(N^2) * \mathcal{O}(\log_2 N) = \mathcal{O}(N^2 \log N)$$

Estos valores son coherentes también con los valores presentados en la tabla. En este caso, el tiempo es mucho más predecible que en el recorrido y los valores calzan con la teoría.

ii. Complejidad esperada recorriendo el árbol

Como se menciona en 3 a), la complejidad esperada recorriendo para cada rayo será:

$$t_{WORST} = \mathcal{O}(rN \log N)$$

Donde r es la cantidad de rayos. Dado que la cantidad de rayos es muy significativa, recorrer toma más tiempo que armar el árbol. Como se observa, por ende, la complejidad esperada se cumple, y los tiempos

[2] Sólo se realizaron pruebas con un factor igual a uno debido al alto costo de medir tiempo con factores mayores. Los tiempos desplegados son el promedio de tres pruebas distintas. Las pruebas además se computaron con $x = 50$ e $y = 0.7$.

están lejos de aproximarse al pero tiempo (si bien asintóticamente no se acerca, como ya se discutió, esto tiene sentido).