NOMBRE: Jose Maria de la Torre

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

IIC2133 —Estructuras de Datos y Algoritmos

Informe 2

1 Estructura

1. Para resolver el problema decidí construir un KD Tree. Cada nodo del KD Tree tiene: Un Bounding Box, un arreglo de triangulos, un boolean si es hoja, una cantidad de triangulos, y una referencia a sus hijos derecha e izquierda. Para construir este árbol, lo que hago es calcular la mediana del eje que voy a cortar. La mediana la calculo tomando todos los vértices de los triángulos que están en el nodo que voy a particionar. Luego una vez que tengo la mediana separo los triángulos del nodo en dos arreglos, mandado al nodo de la derecha todos los triángulos tal que todos sus vértices son mayores a la mediana, y al nodo izquierdo todos los triángulos cuales vértices son menores a la mediana. Los triángulos que no cumplen ninguna de estas dos condiciones se agregan a los dos nodos. Una vez terminada la separación, reviso si la cantidad de triángulos que están en los dos nodos cumplen cierta condición de repetición, ya que no es muy util si se repiten los mismos triángulos en los dos nodos. Si cumple esa condición esos nodos se volverán a separar (llamando en forma recursiva al mismo metodo), pero por un nuevo eje. Otra condición de termino es revisar cuantos triángulos tiene cierto nodo para ver si vale la pena seguir separándolo. Cuando se cumple una condicion de termino el boolean "soy hoja" se cambia a true. Los Bounding Box se calculan con la mediana, una vez terminada la separación de los triángulos que van a la derecha y a la izquierda (salvo el nodo padre que encierra todos los triángulos de la escena).

2. Los KD Node son arboles binarios, ya que solo los divido en el caso de que se creara un hijo derecho y uno izquierdo, nunca solo un hijo ya que no tiene sentido, ya que no se dividira la cantidad de triangulos. El arbol puede no estar balanceado ya que una rama puede seguir separándose mientras la otra no. Los KD Node tienen la propiedad de que van encerrando los items por los distintos ejes, haciendo su búsqueda mucho mas rápida. Esta propiedad ayuda en nuestro caso, ya que nos permite saber mucho mas rápido si un rayo va a intersectar con un triangulo o no. Y debido a que la cantidad de triángulos que tiene cada nodo va disminuyendo a medida que se va bajando por el arbol también podemos recorrer los arboles de forma mas rápido, ya que hay que ver menos triángulos al momento de intersectar el rayo con el Bounding Box y ver que triangulo hay que dibujar. Es importante tener en cuenta que el KD Node es una particion del espacio (del Bounding Box mas grande que encierra todas las figuras)

3. Para construir el arbol, la complejidad depende directamente de la cantidad de triangulos que hay en la escena. para la construccion es necesario calcular la mediana, y para calcular la mediana es necesario primero recorrer todos los triangulos de la escena y despues ordenarlos (se ordenan en O(log n) ya que se ocupa quicksort). Luego como en el peor de los casos esto hay que calcularlo n veces tenemos entonces que la complejidad de armar el arbol es $O(n^2 logn)$.

2 Recorrer el arbol

1. Para ver si hay una intersección entre el rayo y un triangulo , lo que hace el algoritmo es ir revisando los bounding boxes de cada uo de los nodos, acercándose a el triangulo al que debería chocar. En el caso que no choque o que este muy lejos automáticamente el algortimo devuelve un false. Si se recorre el arbol (avanzando solo a la derecha si no hay triangulos a la izquierda o viceversa, avanzando a la caja donde se intersecta primero en el caso que intersecte con los dos, o si intersectan en el mismo lugar a la caja cual se sale antes). Este algoritmo corre hasta llegar una hoja el algoritmo donde revisara los triangulos con ray intercept, y devolverá true si es que hay un triangulo que hay que dibujar o false si no hay. En el caso que no haya se revisa con la otra caja que hay o simplemente retorna un false si no hay triángulos que dibujar.
   Tarda O(log n) en revisar si un rayo intersecto o no ya que ese es el tiempo que se demora en llegar a la hoja del arbol.

2. Hay varias opciones de arboles mal armados. Una opcion es tener un arbol donde no se separa con la mediana, sino por ejemplo con el promedio de los triángulos, y eso puede terminar en que todos los triangulos terminen en una misma hoja, o al menos una gran cantidad. En ese caso resolver la intersección va a tardar O(n) que es básicamente revisar todos los triángulos para ver si intersecta o no. Lo mismo puede suceder si las condiciones de termino son malas, como por ejemplo no tener una condición de triángulos repetidos o la cantidad de triangulos por nodos. Estos son las dos condiciones mas importante y que se analizaran mas tarde, ya que dependiendo del numero que se utiliza el algoritmo tarda tiempo distinto en terminar de construir la escena.

3 Testing

Calculamos el T' para las distintas escenas. Para esto, mis heuristicas estara en matches <0.7*cantidad de triangulos, y menos de 100 triangulos por nodo.

Escena	Triangulos	Rayos	Tiempo (opt) seg	Tiempo (no opti) seg	t'	t' no opt	Mejora
Cube	12	282793	0.522	0.467	1.537E-7	1.376E-7	0.894
Sphere	1280	287756	1.171	33.078	3.17E-9	8.980E-8	28.247
Torus	2304	309289	2.57	74.745	3.606E-9	1.048E-7	29.083
Teapot	4032	314816	6.294	149.648	4.959E-9	1.178E-7	23.772
Billards	4164	1395208	111.893	679.6955	1.925E-8	1.16E-7	6.074
Cornellbox	5130	1021492	43.63	649.585	8.325E-9	1.239E-7	14.888
Infinitycandy	140	4261851	55.734	57.15	9.341E-8	9.578E-8	1.025
Metallic	7682	817456	45.084	956.553	7.179E-9	1.523E-7	21.217
Starters	4480	776648	47.746	440.416	1.372E-8	1.265E-7	9.224

1. (a) La complejidad de la solucion orginal es O(q*n), ya que para cada rayo, hay que revisar todos los triangulos y ver si alguno intersecta con el rayo o no, para luego determinar el triangulo mas cercano para imprimirlo.

   (b) Para mi solucion espero poder resolver el problema en O(log(n)*q) ya que ahora, no hay que revisar todos los triangulos por cada rayo y ademas tenemos la ventaja de saber con el bounding box si es que va a existir un triangulo en el sector donde esa apuntando el rayo. Pero en el caso de haberlo, tambien hay que recorrer menos triangulos que antes en el peor caso. La escena donde mi algortimo hace el mayor impacto es en el torus. Esto se debe a varios factores, partiendo con que existe un fondo negro bien grande, lo que hace todo mucho mas rapido. Ademas, hay muchos triangulos en la parte de atras del torus, por lo que tambien revisa bastante menos triangulos que antes. Estas dos razones son gracias a la construiccion del arbol.

Trian	Match	Cube	Sphere	Teapot	Torus	Billards	Cornell	Infinitycandy	Metallic	Starters
25	0.4	0.527	1.405	6.897	2.734	138.3193	54.305	65.218	57.408	50.229
50	0.5	0.543	1.319	7.730	3.157	132.452	45.076	61.802	54.773	46.709
50	0.7	0.517	1.167	7.402	4.448	136.235	46.009	103.568	54.015	46.308
100	0.5	0.534	1.353	6.52	2.859	108.797	49.976	61.662	73.19	45.787
100	0.7	0.484	1.2403	6.602	2.8860	107.870	49.213	82.294	58.463	52.697
150	0.9	0.522	1.3248	7.637	2.697	101.9409	52.054	61.10	61.996	61.777

2. En la tabla de arriba podemos observar distintos tiempos para distintas condiciones de termino para nuestro arbol.

   (a) Los tiempos se demoran distintos ya que los arboles quedan armados de formas distintas ya que cambian las condiciones de termino. Entonces a veces los arboles quedan mas grandes o mas chicos. Tambien cambia la cantidad de triangulos que quedan en las hojas de los arboles. Entonces al formar la imagen dependiendo de la esena, estos pueden formarse más rapido o más lento ante distintas condiciones de termino

   (b) No afecta de igual forma a todas las escenas ya que estas varian en cantidad de triangulos, rayos etc entonces las condiciones de termino afectan distinta manera la construccion de la imagen. Ademas en la tabla podemos observar que el tiempo mas eficiente cambia ante distitnas escenas. Por ejemplo el Cornell se renderea mas rapido con 50 triangulos por nodo y matches menores a el 70 porciento, pero exactamente las mismas condiciones son el peor caso para el infinitycandy.

3. A continuación tenemos la tabla de t' considerando las escenas hard

Escena	Triángulos	Rayos	Tiempo	Tiempo Construcción	t'
Dragon	130862	262144	221.1635	2.863	6.447E-9
Lucy	275694	262144	372.482	4.488	5.15E-9
Sponza	174840	262144	239.7165	3.628	5.230E-9

   (a) Si se espera la complejidad del tiempo de constricción del árbol, tarda notoriamente mas que en las escenas pasadas, pero esto se debe a que tienen muchos mas triángulos, entonces se demora mas en construir el árbol (lo que esta bien según la complejidad de construcción de mi árbol).

   (b) Al igual que en la pregunta anterior, el tiempo de generación de la imagen es de lo esperado de mi programa. Tarda mas tiempo ya que hay muchos mas triángulos que recorrer. De hecho si se mira bien la tabla se puede observar que la escena con mas triángulos es la que mas tarda en renderear. Por lo que no hay ninguna anomalia. Para haber logrado mas rapidez en estas escenas habría que haber implementado métodos como el SAH que permiten hacer hacer las escenas de una forma mucho mas rápida.