CS3102
ESTRUCTURAS DE DATOS AVANZADAS

Octree

Proyecto Final

Prof. Cristian Lopez del Alamo

Renato Bacigalupo
Jean Miraval
Mateo Noel

December 21, 2020

1 Actividades

1.1 Implementación de Octree

Se puede encontrar todo el código de este trabajo en el siguiente github:
https://github.com/jpmiraval/ProyectoEDA

Esta sección resultó ser un poco trivial, ya que la función principal que necesitábamos era el Insert y una función para detectar si toda la zona del cubo es de un mismo color (en nuestro caso se llama SameColor). Una vez teníamos esas dos funciones el Octree ya estaba implementado. Para el insert lo que hacemos es iniciar pasándole los límites del cubo en x, y, y z. Tomamos esos y corremos la función SameColor. La cual, recorre con un triple for cada pixel para determinar si toda ese sub-cubo es de un mismo color. Como decidimos no binarizar las imágenes, la función de SameColor se demora mucho más, ya que es necesario recorrer dos veces el sector cubo para primero determinar el color promedio, y luego determinar si es que este color es igual para todos. Una vez corremos SameColor, si es que ese sector del cubo no es de un mismo color, llamamos recursivamente a la función Insert 8 veces (por los 8 hijos del insert), y en cada uno ponemos uno de los 8 nuevos cuadrantes que se han creado. Si es que sí es de un mismo color, entonces ponemos como verdadero un flag que dicta que ese nodo es terminal y ponemos su color.

1.2 Implementación de Get Cut

La función de Get-Cut tiene dos variantes para ambos casos de aplicación de la misma. La función consiste en, dado un set de imágenes que componen una imagen tridimencional cuando se superponen entre sí en un orden determinado, y 2 puntos expresados en coordenadas, nos permite realizar un "corte" en esta imagen y proyectar el slide cortado como una imagen 2d. En este proyecto realizamos dos implementaciones de esta función, una aplicada al corte en un cubo (en este caso un paralelepípedo) de "pixeles" y el otro utilizando un Octree.

En ambos casos, se aplicaron dos tipos de cortes, sgún los cuales el input será diferente. Un corte nos permitirá realizar cortes de costado y de frente según la perspectiva que quisiéramos obtener.

Figure 1: Corte seleccionando 2 puntos en XZ

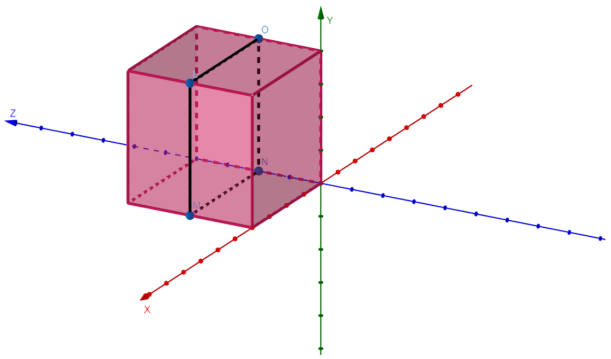
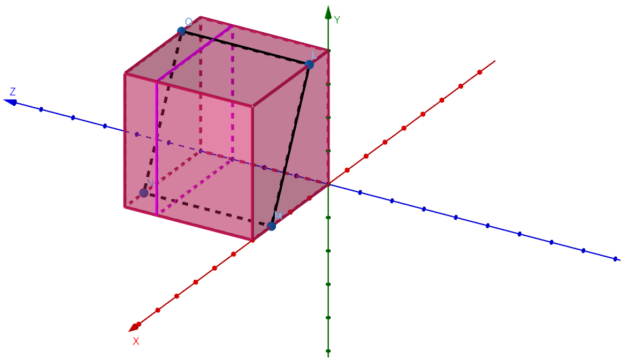


Figure 2: Corte seleccionando 2 puntos en XY



1.2.1 Get-Cut en cubo

Para implementarlo en el cubo, donde directamente tenemos cada pixel de cada coordenada x, y, z de la imagen 3D. Esta función nos dará dos coordenadas en uno de los planos según la perspectiva que deseemos utilizar, ya sea en el plano inferior o lateral. Con estas coordenadas podemos realizar unos cálculos con ambas coordenadas bidimiencionales y obtener la ecuación de la recta, que nos permitirá saber qué puntos son los que deberán ser proyectados.

$$m = \frac{Y2 - Y1}{X2 - X1} \tag{1}$$

$$y = m * x + c \tag{2}$$

Una vez tenemos esta relación entre Y y X, podemos simplemente iterar el cubo en búsqueda de coordenadas que cumplan con la relación, con cualquier Z dentro del cubo.

1.2.2 Get-Cut en octree

Para implementarlo en el octree, una vez insertado en el árbol, tendremos bloques terminales que de alguna u otra manera, contienen todos los pixeles de la imagen 3D, con la diferencia que en más de una ocasión, estos pixeles estarán agrupados en bloques con su mismo color.

El primer paso de la función ene l octree es bastante similar a la anterior, con el cálculo de las fórmulas:

$$m = \frac{Y2 - Y1}{X2 - X1} \tag{3}$$

$$y = m * x + c \tag{4}$$

Una vez tenemos esta relación entre Y y X, podemos recorrer el árbol de tal manera que consultemos a los hijos solo si dentro de sus límites tenemos valores en coordenadas X e Y. De esta manera, cuando lleguemos a un nodo terminal, podemos deducir que todos los puntos que cumplan con la función, y se encuentren dentro de los límites de ese nodo, son nodos que pertenecen a nuestro

plano, e independientemente de nuestro Z, podemos ya obtener el color de cada uno de ellos y generarlo en la imagen.

De esta manera, podemos al final encontrar todos los puntos que pertenecen al plano que hemos buscado, y obtener los colores con los cuales deberíamos completar el output.

En nuestra implementación, cuando encontramos los nodos los incluimos en un vector y luego punto por punto, fuimos rellenando la imagen de output que teníamos. Este proceso de almacenamietno en este vector adicional, y luego iterarlo para

2 Experimentación

2.1 Generador de 20 cortes random

En esta parte vamos a mostrar 4 imágenes que generamos con los cortes random. Y explicaremos un caso en que puede ocurrir al poner puntos dentro de la función:

Figure 3: Corte a base de los puntos (0, 511) y (0, 20)

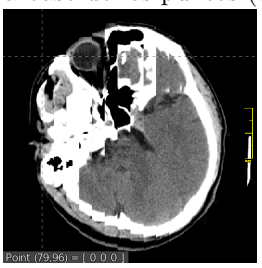


Figure 4: Corte a base de los puntos (0, 511) y (0, 39)

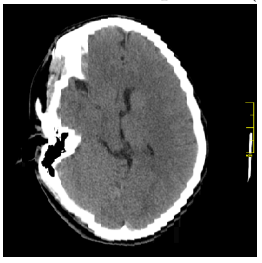


Figure 5: Corte a base de los puntos (3, 150) y (8, 20)

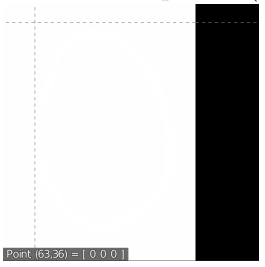
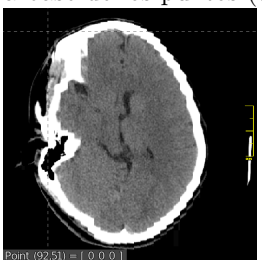
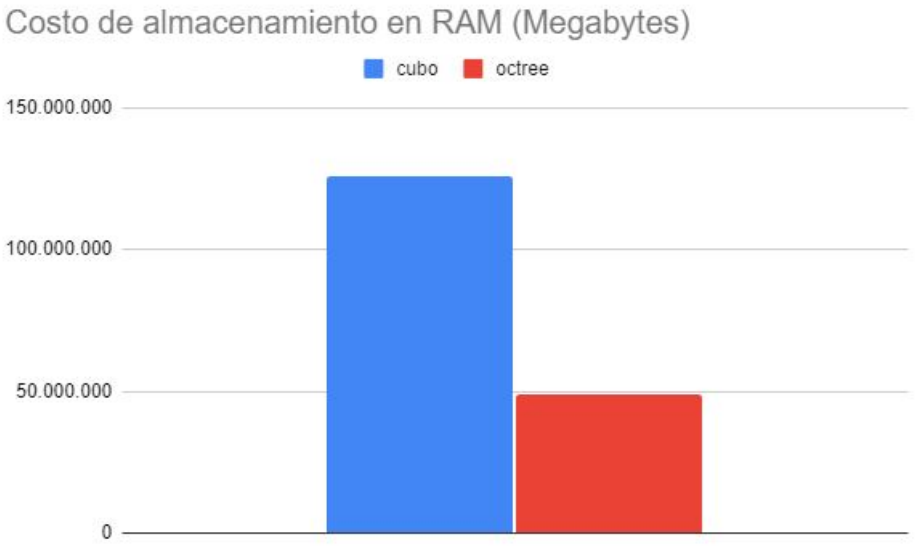


Figure 6: Corte a base de los puntos (511, 39) y (0, 0)



Como podemos ver en la figura 5, tenemos un corte malo. Esto pasa ya que, dados los puntos que recibe el algoritmo, la recta que ejecuta el corte tendrá una pendiende muy empinada, y al momento de tratar de realizar el corte el resultado es el corte correcto, pero el ángulo es tan bajo que la imagen resultante no llega a rellenar todo el output y los pixeles que recorre suelen ser redundantes. De hecho, es un mal corte desde los puntos que se ingresa, pues el ángulo no es nada útil.

2.2 Comparación del costo en RAM



2.3 Comparación del costo computacional

Tiempos en segundos: Split_Costado es la función para cortar donde se proyecta en la coordenada Y. Get_Corte es esa misma función, pero no usando el octree. Y por ultimo, los números al costado de los segundos son los parámetros de las funciones, donde dado a, b, c, d. a = x0, b = z0, b = x1 y d = z2.

Split_Costado	s	**Get_Corte**	s
0, 0, 511, 20	0.473	0, 0, 511, 20	0.626
10, 10, 500, 20	0.465	10, 10, 500, 20	0.631
511, 39, 0, 0	0.44	511, 39, 0, 0	0.628
0, 0, 511, 39	0.421	0, 0, 511, 39	0.629
300, 26, 15, 35	0.405	300, 26, 15, 35	0.631
132, 15, 33, 2	0.269	132, 15, 33, 2	0.627
14, 10, 342, 11	0.476	14, 10, 342, 11	0.601
3, 8, 150, 20	0.332	3, 8, 150, 20	0.625
33, 13, 53, 23	0.05	33, 13, 53, 23	0.615
69, 39, 1, 0	0.038	69, 39, 1, 0	0.574
420, 17, 120, 0	0.357	420, 17, 120, 0	0.571
19, 5, 20, 3	0.014	19, 5, 20, 3	0.614
300, 25, 820, 2	0.396	300, 25, 820, 2	0.569
500, 10, 100, 6	0.403	500, 10, 100, 6	0.631
311, 7, 51, 4	0.416	311, 7, 51, 4	0.627
100, 20, 200, 30	0.269	100, 20, 200, 30	0.632
156, 6, 62, 8	0.384	156, 6, 62, 8	0.614
467, 13, 312, 15	0.453	467, 13, 312, 15	0.63
1, 20, 400, 19	0.4	1, 20, 400, 19	0.638
30, 4,21, 39	0.006	30, 4,21, 39	0.627

3 Conclusión

En conclusión, en la gran mayoría de casos es mejor usar el octree, ya que como vemos se demore menos y consume menos espacio. El único inconveniente es que se pierde calidad de la imagen. Si es necesario tener o poder recrear las imágenes en la calidad original o no tan mala como el octree, es mejor no usar estructuras como esas. Y menos si binarizamos las imágenes lo cual crea más diferencia de calidad. Pero, en general el octree es más rápido y consume menos memoria, por ende, es mejor.