Voronoi en el GPU

Emmanuel Rojas Fredini

November 24, 2009

Contents

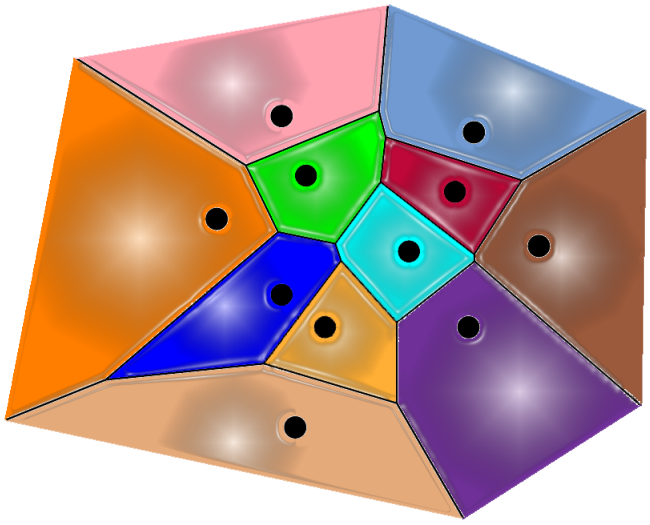
Chapter 1

Diagrama de Voronoi? se come?

1.1 Introduccion

dasgdksadhgsa

1.2 Definicion



References Definamos la distancia Euclidea entre 2 puntos $\boldsymbol{p}$ y $\boldsymbol{q}$ como $\boldsymbol{dist}(\boldsymbol{p}, \boldsymbol{q})$. Luego en el plano tenemos

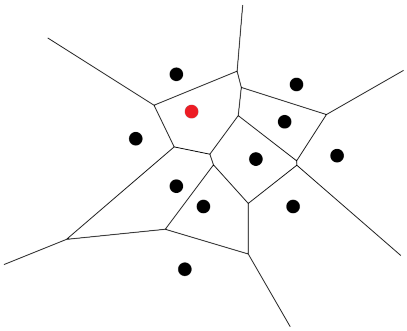
$$\boldsymbol{dist}(\boldsymbol{p}, \boldsymbol{q}) = (\boldsymbol{p_x} - \boldsymbol{q_x})^{\boldsymbol{2}} + (\boldsymbol{p_y} - \boldsymbol{q_y})^{\boldsymbol{2}} \tag{1.1}$$

Y análogamente para mas dimensiones. Sea $\boldsymbol{P} = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots, \boldsymbol{p_n}\}$ n puntos distintos en el espacio. Llamaremos a estos puntos sitios. Entonces definimos

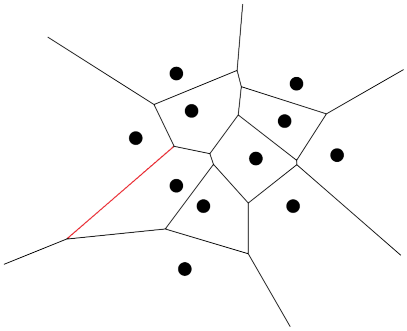
el diagrama de Voronoi de $\boldsymbol{P}$ como la subdivisión del plano(o del espacio) en $\boldsymbol{n}$ celdas, una para cada sitio en $\boldsymbol{P}$, con la propiedad de que el punto $\boldsymbol{q}$ esta en la celda correspondiente al sitio $\boldsymbol{p_i}$ $\boldsymbol{siidist(q, pi) < dist(q, pj)}$ para $\boldsymbol{p_i \wedge p_j \in Pj \neq i}$. Asimismo denotaremos al diagrama de Voronoi de $\boldsymbol{P}$ como $\boldsymbol{Vor(P)}$, y la celda $\boldsymbol{p_i}$ del diagrama anterior la denotamos $\boldsymbol{Vor(p_i)}$.

De esta forma el diagrama de forma de los siguientes elementos:

- Sitios, puntos en R2, R3, ..., Rn. [**?**]



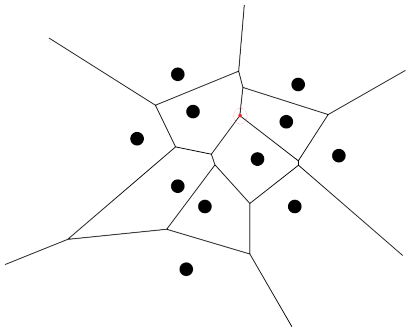
- Aristas o planos de bisección, son los planos(o segmento, hiper-planos) que separan a 2 celdas, delimitando los puntos más cercanos a un sitio $\boldsymbol{p_i}$ y otro $\boldsymbol{p_j}$ con $\boldsymbol{i \neq j}$. Tienen la propiedad de ser equidistantes a 2 sitios. [**?**]



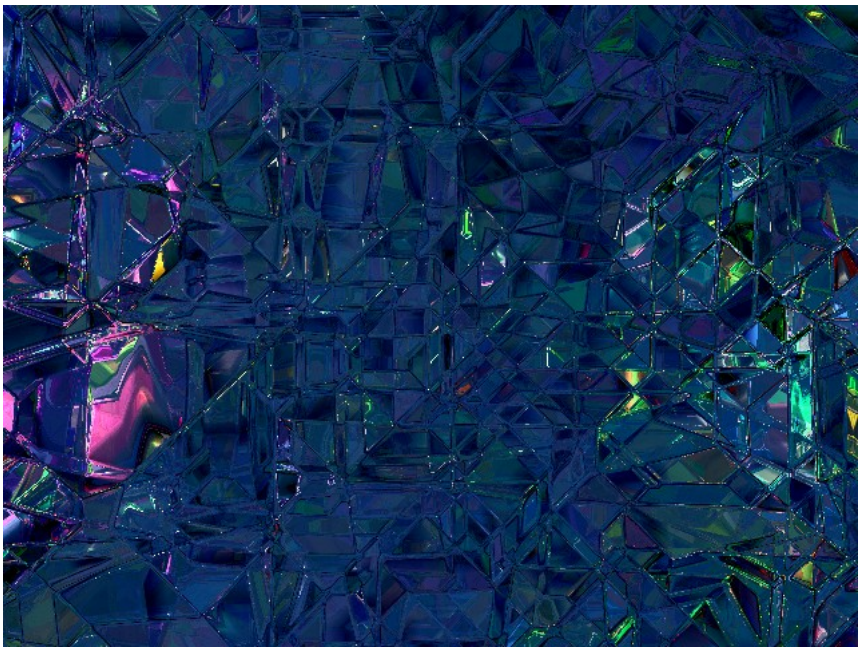
- Semillas o vértices, son las uniones de las aristas o planos de bisección, se caracterizan por que si una circunferencia(o esfera, hiper-esfera) crece desde este chocaría con varios sitios al mismo tiempo, ie son equidistantes a 3 o mas sitios. [**?**]

1.3 Aplicaciones

Se puede construir una esctructura en base al diagrama de Voronoi para encontrar el vecino mas cercano, relacionando a que celda pertenece el punto, y que celdas vecinas tiene, es decir las aristas de la tringulacion de Delaunay. Un uno muy extendido es queantizacion vectorial, muy usado en la compresion de datos.

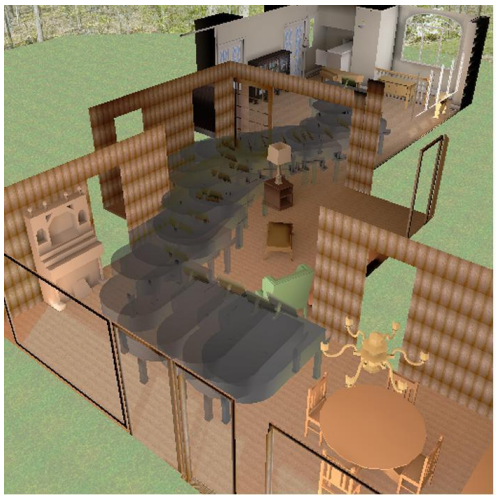


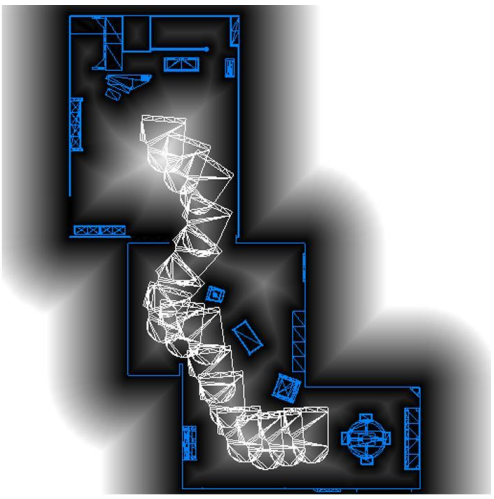
Se puede encontrar el mayor circulo vacioe ntre todos los sitios, y tambien poligonos, esto se lo reconoce verificando los vertices o semillas del diagrama. Esta propiedad puede usarse para identificar la posicion mas conveniente para el establecimiento de un negocio, ect. Es usado en la fisica de polimeros[1]. Con este se puede representar el volumen libre de un polimero. Se usa en el analisis de la capacidad de una red inalambrica. Se lo utiliza para analizar los patrones de creciemiento de bosques, y en predicciones de la expansion de incendios forestales. También se los usa para generar proceduralmente texturas con apariencia organica[**?**].



En robotica, se usa el diagrama de Voronoi para encontrar rutas de movimiento. Si los puntos son obstaculos, entonces las aristas del diagrama seran las rutas mas alejadas de los obstaculos[**?**].

[1]Un polimero es una molecula grande (macromolecula) compuesta por una estructura de unidades repetidas, generalmente conectadas por una union covalente





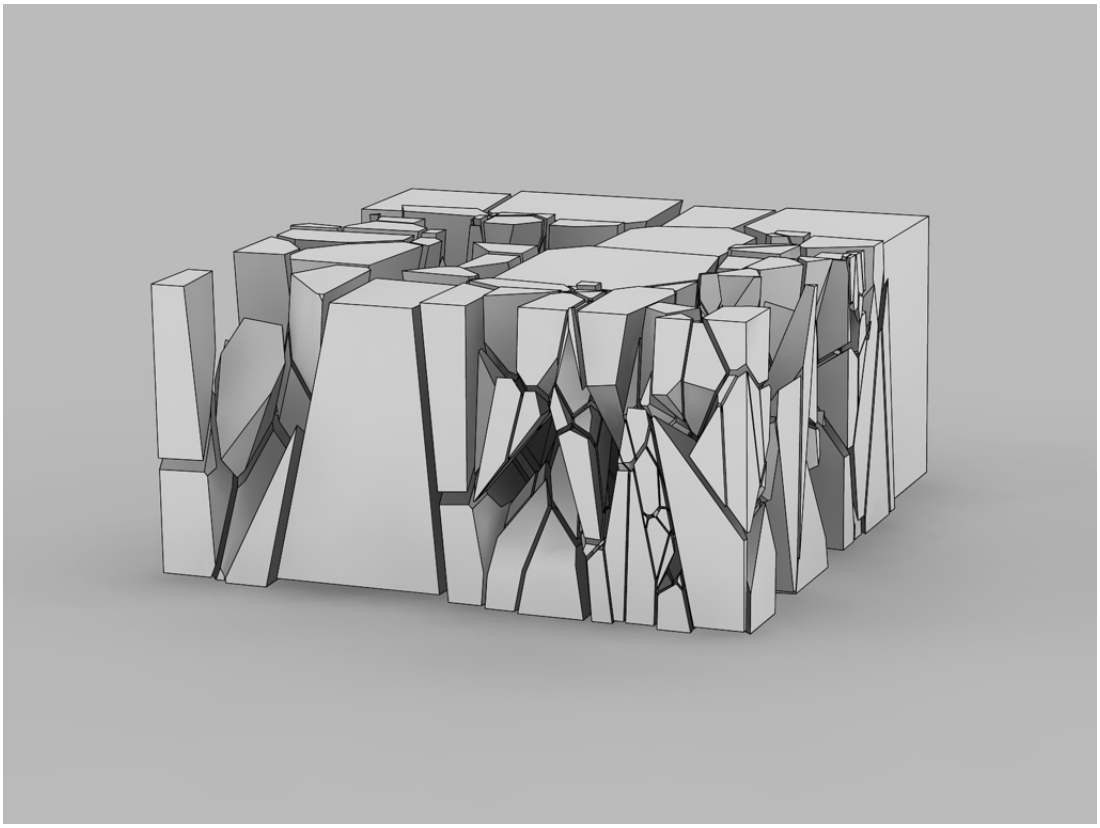
Se usa para teselar modelos 3D en fractales, para luego simular ruptura de elementos. Un ejemplo de esto es el Voronoi Shatter de Blender, que permite teselar un modelo, pudiendo luego hacer una simulacion fisica con el motor fisico de Blender[?][?].

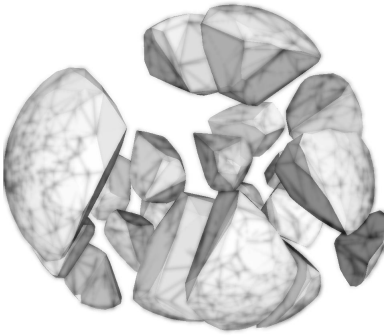
1.4 Calculo

Algunos métodos para calcular este diagrama de forma eficiente son:

1. Linea de Fortune que presenta un orden $\boldsymbol{O(n * log(n))}$ en el plano y $\boldsymbol{O(n)}$ en el espacio, donde n son los sitios en el espacio.
2. Algoritmo basado en Dividir y Conquistar.
3. Calculo de triangulación de Delaunay, y pasar por la dualidad *Voronoi-Delaunay* al Diagrama de Voronoi
4. Jump Flooding, paper de Guodong Rong y Tiow-Seng Tan, que presenta un orden $\boldsymbol{O(k)}$ donde $\boldsymbol{k}$ es una constante. Este método requiere de un procesador multi nucleo.
5. Renderizado de Voronoi usando funciones de distancia, paper de Hoff, requiere renderizador muli nucleo SIMD.

Explicaremos otro método que utiliza procesadores SIMD (más específicamente una GPU CUDA enabled) para calcular el diagrama de Voronoi de forma discreta, es decir que se calculara en una grilla de axb, o axbxc, o generalizado para mas dimensiones, en lugar de en el espacio continuo $\boldsymbol{R^2}$, $\boldsymbol{R^3}$, o $\boldsymbol{R^n}$.





shatte2.jpg

Chapter 2

Introduccion a CUDA

CUDA(Compute Unified Device Arquitecture) es un modelo de programacion paralela con operaciones SIMD(Single Instruction Multiple Data), que permite dar instrucciones a grupos de hilos(threads) de ejecucion mediante un lenguaje compuesto de C con extensiones. Este se basa en una arquitectura de implementacion de procesadores multi-nucleo, que actualmente solo es implementada por Nvidia(su creador). Asimismo existen otros modelos muy similares entre los que destacamos Stream Computing de Ati, OpenCL y ComputeShader sinendo todos estos distintos modelos de programacion SIMT(Single Instuction Multiple Thread), es decir que permiten ejecutar un grupo de instrucciones por gran cantidad de threads en paralelo.
El modelo es inerentemente jerarquico, tanto en su modelo de memoria, como en su modelo de threads. Aumentando la capacidad de granilaridad de un algoritmo.

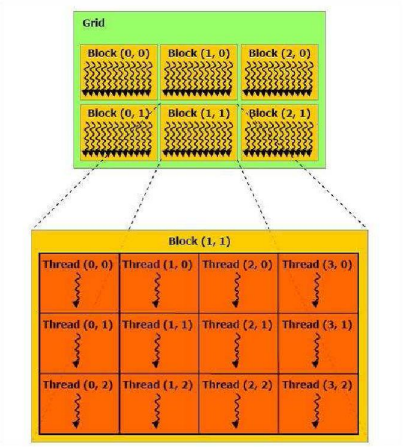
2.1 Jerarquia Hilos

Los hilos se identifican un identificador de 3 variables $\boldsymbol{x}$, $\boldsymbol{y}$ y $\boldsymbol{z}$. Esto permite que la paralelisacion en base a arreglos de hasta 3 dimensiones sea facil de implementar, y no se deba recurrir a mapeos obscuros que ofusquen el codigo.
A su vez los threads se agrupan en bloques, identificandose estos por 2 variables $\boldsymbol{x}$ e $\boldsymbol{y}$. También permitiendo esto simplificar los algoritmos.
En fin un thread es identificado universalmente por su id de hilo o *threadIdx*, y su id de bloque o *blockIdx*.
Cuando se habla de la grilla(Grid), se refiere al arreglo bi-dimensional de bloques con los que se ejecuta el kernel.

2.2 Jerarquia de Memoria

Esta se compone de:

1. Un conjunto de registros pertenecientes a cada thread exclusivamente. Tiene un periodo de vide ligado al thread.



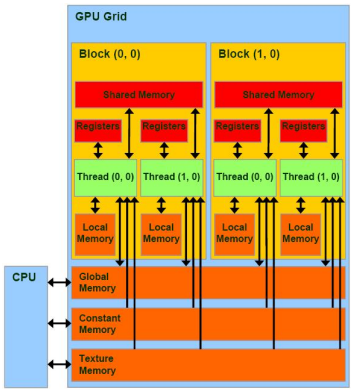
2. Una memoria compartida(Shared Memory) entre todos los threads de un bloque. Tiene un periodo de vide ligado al bloque/thread, ya que estos "nacen y mueren juntos".

3. Una memoria de Textura, es solo de lectura y esta optimizada para mapeos d texturas y sus formatos, compartida por el kernel. Tiene un periodo de vide ligado al kernel.

4. Una memoria Contante, es solo de lectura, compartida por el kernel. Tiene un periodo de vide ligado al kernel.

5. Una memoria global compartida por el kernel. Tiene un periodo de vide ligado al kernel.

La jerarquia fue nombrada en un orden de velocidad de memoria decresiente o identica, y de latencia creciente o identica. Empero en tamaños decendentes o identicos. Luego la memoria compartida, de textura y constante puede considerarse como un cache administrado por software y no por hardware, ya que permite una gran velocidad de acceso y modificacion. Por eso es conveniente copiar la memoria sobre la que se va a trabajar a alguno de estas memorias.



Chapter 3

Voronoi + CUDA = Mi Trabajo Final?

Este metodo de calculo de Voronoi discreto, o aproximacion a Voronoi, utilizando la GPU tiene la ventaja sobre los otros 2 algoritmos para calcular Voronoi en la GPU, el metodo de Hoff(renderizado de funciones de distancia) y Jump Flooding, en que este metodo es facil de modificar para obtener la solucion a otros problemas, como son obtener los puntos mas lejanos y k-esimo Diagrama de Voronoi. Ademas el metodo de Hoff puede tener un cuello de botella cuando se sobrecarga a la GPU con demasiados sitios, es decir que no se escala como el algoritmo que expondremos.
El algoritmo consiste en ejecutar por cada pixel de la grilla un thread, y en cada thread calcular la distancia al punto mas cercano, luego este pixel pertenece a la celda de ese sitio. Expresando esto en un pseudocodigo:

```
Para cada pixel
{
Calcular la distancia de este pixel a todo los sitios
Elegir el sitio mas cercano
Colorear este pixel con el color del sitio mas cercano
}
```

De esta forma si desea lograr la solucion a los puntos mas lejanos, en vez de buscar la menor distancia, se busca la mayor distancia. Para lograr una buena implementacion con este algoritmo se debe utilizar la memoria compartida para buscar la menor distancia, ademas llamar al kernel con los parametros ams convenientes para la GPU que la ejecuta. Esto se debe a que por ejemplo la placa Nvidia 8800 GTS posee 12 multiprocesadores, de esta forma lo mas conveniente es tener 12 blockes, y para aumentar el uso de cada multiprocesador, que posee 8 procesadores stream y puede ejecutar como maximo 512 threads al mismo timepo, asi que lo ams conveniente es hacer arreglos de 512 threads.
Otra forma de mejorar el algoritmo es utilizar una estructura de ordenamiento espacial como un kd-tree, un quadtree, o simplemente agrupar por grillas equidimensionadas.

List of Figures

List of Tables