DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

TP Final: Generación de terrenos montañosos por elevación

Organización del computador II

Integrante	LU	Correo electrónico
Florencia Zanollo	934/11	florenciazanollo@gmail.com
Luis Toffoletti	827/11	luis.toffoletti@gmail.com

Índice

1. Introduccion

Para este trabajo práctico nos propusimos implementar el modelo para generación de terreno explicado en el paper "The Uplift Model Terrain Generator".[1] Generación de terreno es la creación virtual de formaciones terrestres en computación gráfica. El objetivo es crear formaciones que se vean reales sin consumir demasiado tiempo de procesamiento, para que puedan ser utilizadas en simulaciones, videojuegos, etc.

En el paper se propone la generación de terreno montañoso a partir de elevaciones o picos, en contraste con métodos anteriores como el algoritmo fractal. El modelo se puede aplicar tanto en 2D como 3D y promete ser menos intensivo computacionalmente que sus antecesores. La idea general del algoritmo es generar picos de manera aleatoria y luego obtener la altura final de cada porción del terreno promediando las influencias provenientes de las elevaciones.

En nuestro trabajo realizaremos dos implementaciones del modelo, una de ellas en C++ y la otra en ASM utilizando la tecnología SIMD. Nuestros objetivos son:

- Obtener métricas para determinar ventajas y desventajas de realizar parte del procesamiento en assembler y decidir en qué casos podría recomendarse su utilización.
- Construir una solución que utilice de forma más eficiente el procesamiento de datos en paralelo.
- Obtener representaciones gráficas de los terrenos generados, que puedan tener utilidad práctica y, a partir del tamaño de su entrada, estimar el tiempo de procesamiento.

En este informe pretendemos mostrar los gráficos y resultados obtenidos así también como explicar de manera simplificada la solución de las distintas implementaciones.

[1] `https://www.dropbox.com/s/q6brk3jqwppxrhx/upliftTerrainGenerator.pdf?dl=0`

2. Implementación

El método requiere de los siguientes datos de entrada:

divisions cantidad de divisiones sobre la cuál se colocarán los picos.

nroPeaks cantidad total de picos.

yMin altura mínima permitida de un pico.

yMin altura máxima permitida de un pico.

ruggedness escabrosidad del terreno, es decir, qué tanto afecta un pico a sus posiciones aledañas.

Lo que denominamos '**pico**', es un valor que será generado y ubicado aleatoriamente ocupando alguna de las posiciones posibles (divisions). Representa un punto de elevación en el terreno que luego el algoritmo se encargará de interpretar (en conjunto con la información del resto de los picos) para definir el valor definitivo que tendrá la elevación en cada punto del terreno.

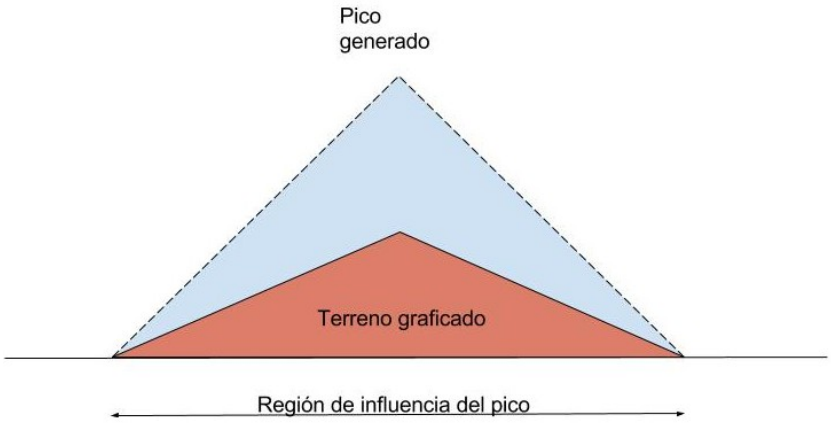


Figura 1

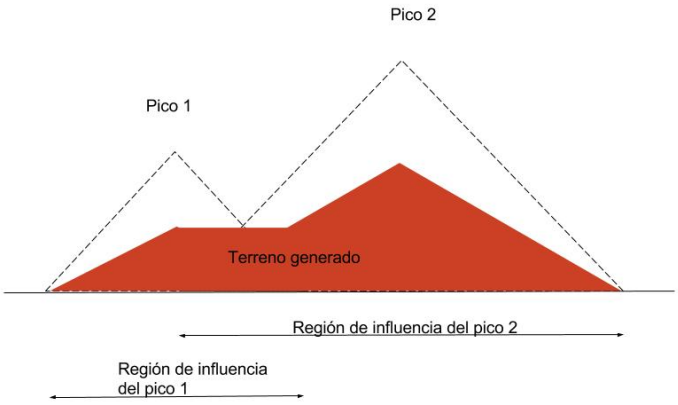


Figura 2

En la figura 1 podemos ver un terreno generado a partir de un único pico, como se puede apreciar el valor del pico se promedia con el suelo para obtener el terreno definitivo. Se observa también una zona de influencia del pico, es aquella en la cual el punto de elevación afecta en el cálculo final del terreno.

El concepto de '**influencia**' de un pico podemos visualizarlo mejor en la figura 2. El terreno fue generado a partir de dos picos, existe una región donde tenemos influencia por parte de ambos, es donde se forma una meseta ya que el algoritmo promedia la altura que tienen los picos en esa región. En el resto del terreno hay influencia de un sólo pico o ninguno.

En nuestro caso también contamos con las siguientes entradas, las cuáles agregamos por razones de utilidad a la hora de experimentar y comparar las implementaciones de C y ASM.

seed permite setear una semilla particular para el random, esto es para lograr el mismo gráfico y poder experimentar con datos más certeros.

debugging si una semilla es proporcionada entonces se le puede decir al programa que entre en modo verbouse; el cuál nos va a dar, además del gráfico, el valor numérico de cada posición del terrreno final y la posición y tamaño original de cada pico.

Para poder llevar a cabo el método vamos a estar utilizando las siguientes estructuras:

peaksPos arreglo con nroPeaks posiciones, cada una contiene la posición dentro del terreno de dicho pico. Las posiciones van de 0 a divisions-1.

peaksSize arreglo con nroPeaks posiciones, cada una contiene la altura de dicho pico. Las alturas posible están entre yMin e yMax (inclusive en ambos casos).

terrain vector con los valores numéricos finales de cada posición del terreno.

Los picos son generados de manera aleatoria, tanto su posición como su altura (dentro de los límites explicados más arriba). Luego se hace lo siguiente:

```
for all divisions do
    for all peaks do                    ▷ se calcula la influencia de cada pico para cada posición
        influencia ← altura del pico - distancia del pico a la posición actual * ruggedness.
    end for
    if la posición no tiene influencia de nadie then
        valor final de la posición ← 0.
    else if la posición solo es influenciada por un pico then
        valor final de la posición ← influencia / 2.
    else if la posición es influenciada por dos o más picos then
        valor final de la posición ← influencia / cantidad de picos con influencia sobre ella.
    end if
end for
```

Quedando así en cada posición del terreno final el valor de 'promediar' todos los picos que tienen influencia sobre ella.

La complejidad teórica del algoritmo es $\theta(divisions * peaks)$ ya que indefectiblemente se recorren ambos y el resto de las operaciones son $O(1)$.

2.1. Versión de C

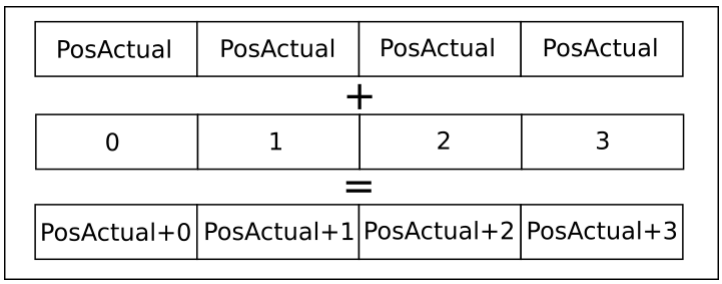
Está versión es muy sencilla ya que se puede mapear directamente el pseudocódigo a código, con pocas modificaciones. Luego basta con compilar utilizando las mejoras de sse para contar con las herramientas de SIMD.

2.2. Versión de ASM

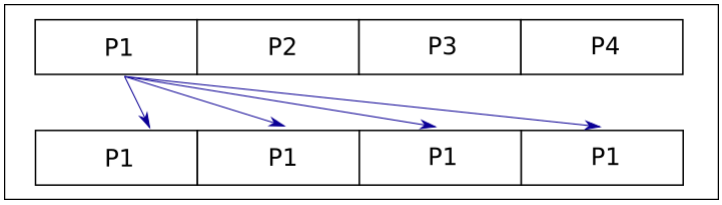
La idea es simple, recorrer las posiciones del terreno de a cuatro, ya que como cada una es un float esa es la cantidad que entran en un registro xmm. Luego, en cada iteración, recorrer todos los picos, también de a cuatro (son int pero de 32 bits, es decir, cuatro por registro xmm).

Cuando obtenemos los datos de los picos, expandimos y repetimos la información de cada uno en distintos registros. Para que la explicación sea más clara mostraremos el caso del pico uno (P1) siendo los demás prácticamente iguales salvo por la información a usar. Entonces:

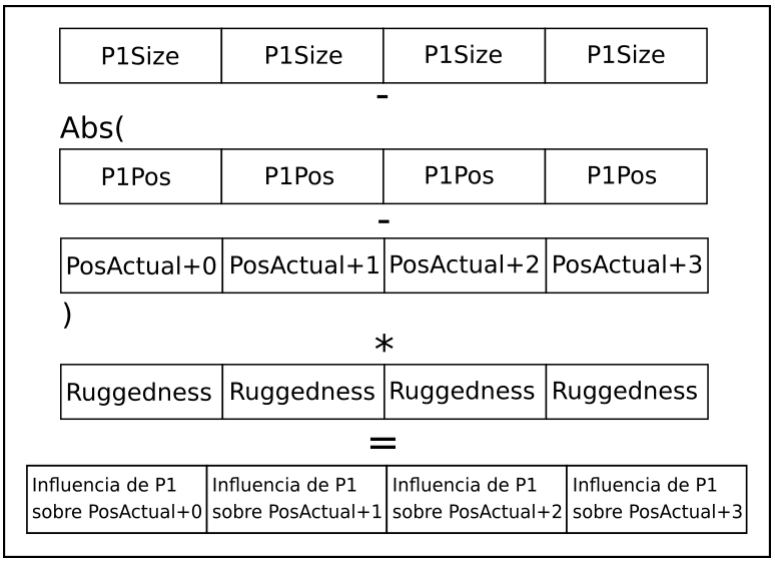
Primero se replica la posición actual en un registro xmm, al cuál le sumamos un registro constante con los enteros 0 a 3, es decir, nos queda el valor de cada una de las cuatro posiciones que estamos calculando en cada entero.



Luego, tanto para la posición (Pos) como para la altura (Size) de P1, se crean registros con la información replicada en cada int.



Contamos con dos acumuladores, uno para la 'influencia' de los picos en esa posición y otro para la cantidad de picos que influyen. Para calcular el primero realizamos lo siguiente, donde Abs hace referencia a la función de valor absoluto:



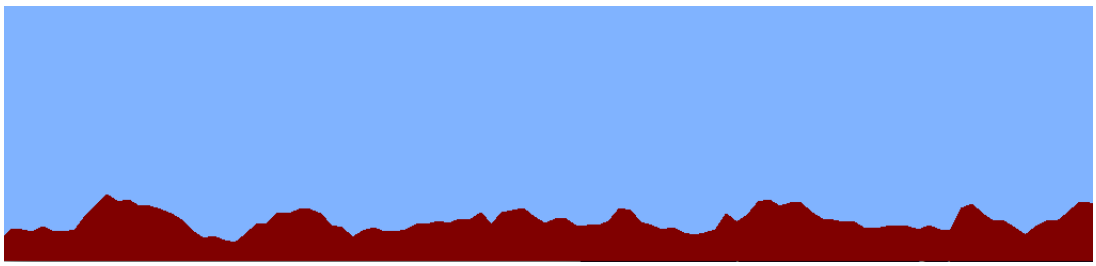
Utilizando el registro que contiene la influencia obtenemos si el pico influye (influencia > 0) o no (caso contrario) sobre cada una de las cuatro posiciones.

Sumando esta información para cada pico obtenemos cuánta influencia, y de cuántos picos, tiene una posición determinada. Así al final del ciclo que recorre los picos solo nos queda dividir influencia por cantidad de picos y así obtenemos el valor real del terreno en la posición.

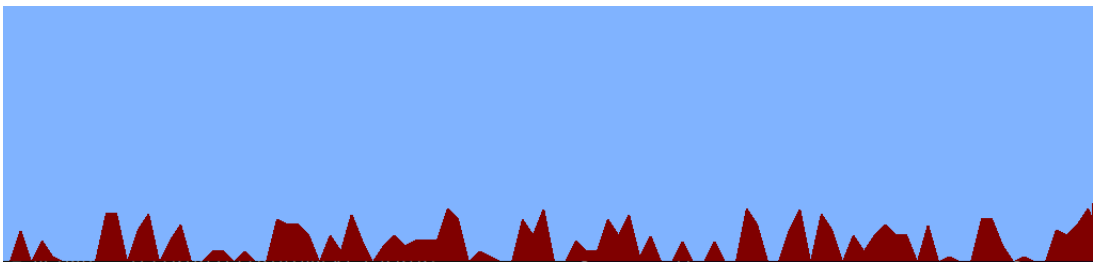
2.3. Algunos ejemplos

El objetivo de esta sección es brindar ejemplos concretos, con distintos datos de entrada, para mejorar el entendimiento de la funcionalidad de los mismos. El ruggedness o escarpamiento del terreno, por ejemplo, es algo que en principio cuesta entender pero que al comparar imágenes específicas se comprende mejor el funcionamiento.

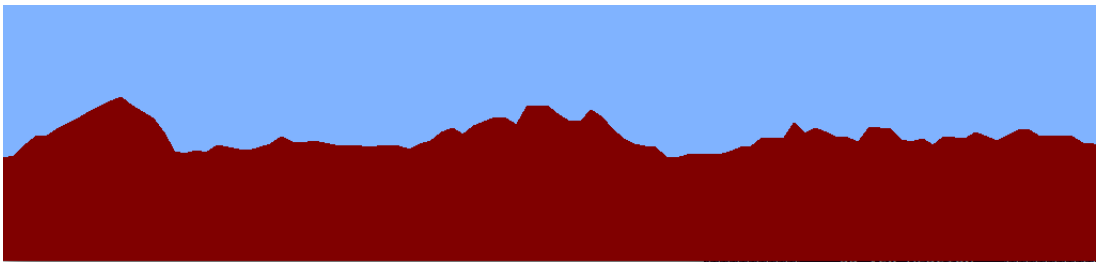
El siguiente terreno fue generado con 300 divisiones (se ven menos en la imágen). 200 picos. de alturas entre 1 y 10, con un escarpado de 1.[2]



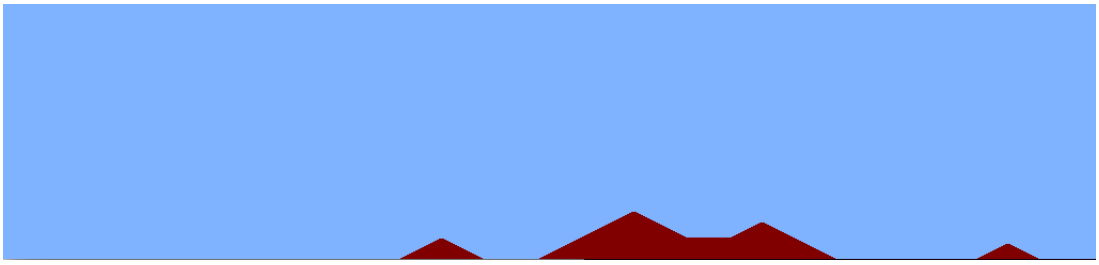
A continuación utilizamos la misma entrada salvo por el escarpado, el cuál es de 10.



Se puede ver cómo, al aumentar el escarpado, el terreno queda más abrupto. Esto se debe a que la influencia de los picos desciende con mayor velocidad, es decir, el rango de posiciones sobre las cuáles influye (y también el tamaño de su influencia) va decreciendo a medida que se aumenta el escarpado. Así, hay posiciones sobre las que antes influía que ahora no, o en menor medida, generando picos más aislados o separados.



En este caso variamos las alturas de los picos, ahora entre 10 y 30, utilizando la misma semilla y escarpado de 1. Como se puede observar, aumentar el tamaño de los picos no es mover la imágen más arriba, esto se debe a que las alturas se eligen al azar dentro de las posibles.



Por último queríamos mostrar qué sucede cuando hay muy poca proporción de picos. Éstos quedan aislados casi siempre y se genera un terreno que poco tiene que ver con la realidad, de todas maneras puede tener su utilidad, quizás en algún videojuego por ejemplo.

[2] Si se quiere generar, la semilla usada fue: 123456.

3. Experimentación y Resultados

Al experimentar siempre es de suma importancia elegir los casos de prueba. Como es un algoritmo que depende de una semilla random lo primero que decidimos fue dejarla fija, para que las sucesivas ejecuciones fuesen de la misma naturaleza. El algoritmo es de complejidad lineal respecto de divisiones por cantidad de picos, por ende al experimentar probamos diferentes combinaciones de los mismos.

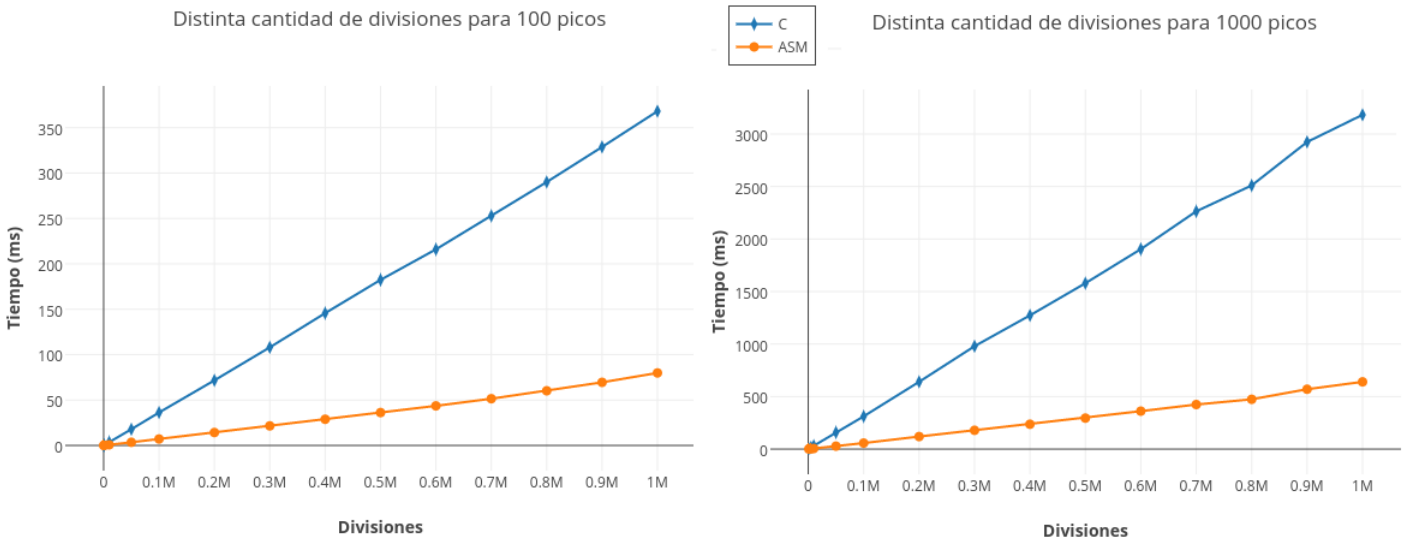
Basandonos solo en la teoría, en un comienzo suponíamos una mejora en la performance de ASM cercana a 16 veces la de C. Llegamos a ese número partiendo de que, si C no tenía ninguna mejora iba a poseer la complejidad teórica de divisiones por cantidad de picos; mientras que nuestro ASM recorría tanto las divisiones como los picos de a cuatro, si consideramos esto nos queda $divisiones/4 * picos/4$. Esta complejidad es del mismo orden de grado (lineal) modificada solo por la constante $1/16$, presentando así una mejora del 93.75 %.

Para la recolección de datos utilizamos la librería Nonius[3] que toma varias muestras, corriendolas muchas veces y procesa los resultados eliminando outliers. Las pruebas fueron corridas sobre un procesador Intel® Core™ i5-6200U[4].

Los datos con los que fueron generados los gráficos se encuentran la siguiente url `http://cor.to/OrgaIIDatos` y los gráficos se encuentran de manera interactiva en las url al pie de la página de cada uno.

3.1. Misma cantidad de picos, distintas divisiones

Como primer experimento fijamos la cantidad de picos y fuimos aumentando las divisiones.



[5]

En el gráfico de la izquierda probamos con 100 picos, como se ve al aumentar las divisiones el tiempo consumido crece de forma lineal. Si bien se presenta una mejora promedio del 78 %, no alcanza a los valores teóricos esperados.

En el de la derecha con 1000 picos. A pesar de que el gráfico muestra la misma tendencia analizando los datos encontramos que ahora la mejora es un poco más notoria y se eleva al 81 %.

[3] `https://nonius.io/`

[4] `https://ark.intel.com/es-es/products/88193/Intel-Core-i5-6200U-Processor-3M-Cache-up-to-2_80-GHz`

[5] 100 picos: `https://plot.ly/~fzanollo/55/` 1000 picos: `https://plot.ly/~fzanollo/53/`

3.2. Misma cantidad de divisiones, distinto porcentaje de picos

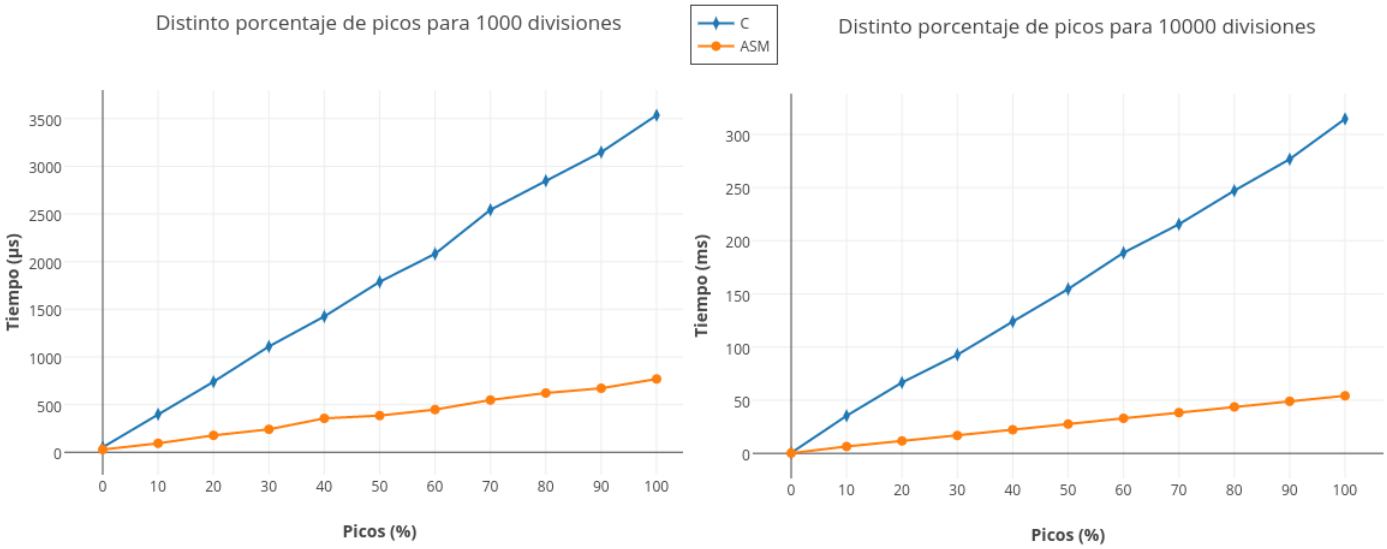
El siguiente conjunto de casos de prueba fueron con una cantidad fija de divisiones, variando el porcentaje de picos respecto a las mismas.

Estos gráficos resultaron muy parecidos entre sí y respecto al anterior conjunto de pruebas.

[6]

A medida que se aumentan las divisiones el gráfico se asemeja más al de una función lineal ya que se obtienen resultados más fehacientes.

Esto es porque, normalmente, a la hora de medir cantidad de ciclos de CPU pueden suceder diferentes eventos que alteran las mediciones, el ruido generado por los eventos se notan más si son pocos ciclos.



[7]

[6] 100000 divisiones: `https://plot.ly/~fzanollo/51/`
[7] 1000 divisiones: `https://plot.ly/~fzanollo/61/` 10000 divisiones: `https://plot.ly/~fzanollo/59/`

[8]

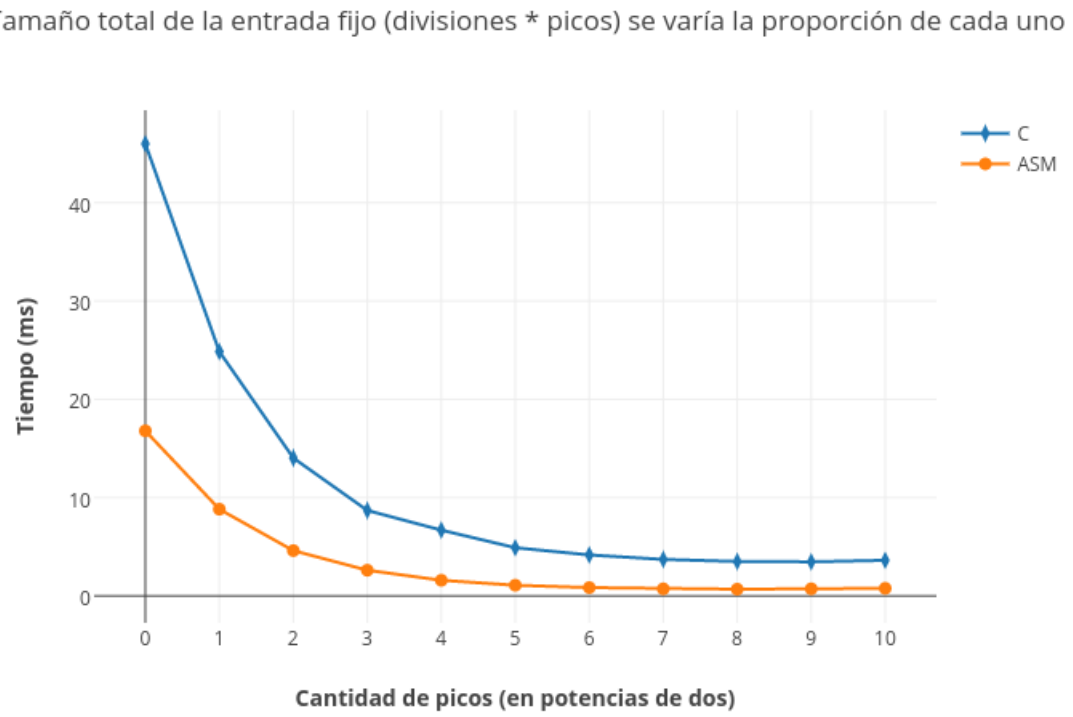
Dado que eran muy parecidos decidimos presentar los casos más chicos (1000 y 10000 divisiones) conjuntamente. La mejora promedio aproximada entre todos estos casos de prueba es del 80 %.

[8] 50000 divisiones: `https://plot.ly/~fzanollo/57/`

3.3. Misma cantidad de tamaño de entrada

En los casos de prueba que listamos previamente mantuvimos fija una variable (cantidad de picos o cantidad de divisiones) e incrementamos la restante, como complemento realizamos un análisis manteniendo el tamaño total fijo, lógicamente la cantidad de picos no puede superar a la de divisiones. Cantidad de picos p x Cantidad de divisiones d = Tamaño de entrada total T

En principio pensamos que el tiempo de ejecución debería mantenerse en todos los casos, debido a que tienen la misma complejidad teórica. En todo caso aumentando ligeramente para los casos con mayor cantidad de picos, debido a los cálculos previos para obtener los datos aleatorios de las ubicaciones y tamaños.



[9]

Como vemos en el gráfico, a medida que crece el número de picos y disminuye la cantidad de divisiones, el tiempo de procesamiento decrece.

Obtuvimos un grupo de casos, desde un pico con 1024x1024 divisiones hasta 8 picos con 1024x128 divisiones (en el gráfico desde 0 hasta 3), que presenta un comportamiento diferente al resto en cuanto a tiempo de procesamiento.

Si bien estos casos podrían analizarse en profundidad, vamos a descartarlos ya que poseen un porcentaje de picos muy bajos para una generación de terreno concreta (mucho menos del 1 %).

Para el resto de los casos donde los porcentajes de picos sobre divisiones son más considerables, las gráficas se observan más estables.

[9] Tamaño total fijo: `https://plot.ly/~fzanollo/63/`

4. Conclusión

Pudimos comprobar que nuestra implementación presentó una mejora considerable en cuanto al tiempo de procesamiento, en promedio del 80 %. Difiere de la estimación teórica que habíamos calculado por diversos factores, entre ellos:

- La optimización del código de C hecha por el compilador.
- Las variaciones y errores que puede haber a la hora de medir los tiempos de procesamiento.
- La omición de algún detalle en cuanto a la implementación que resulte en una desventaja.

Hablando de los gráficos utilizables en la práctica, estos representan un subconjunto de los casos que utilizamos para las pruebas. Por tanto los resultados mostrados tienen precedencia en la realidad.

En particular serían aquellos que tienen una relación de por lo menos 1 pico cada 10 divisiones, considerando en la cantidad a colocar la variación de tamaños de los picos. Ya que si se utilizan alturas muy grandes en generaciones chicas puede quedar un terreno no deseado.

4.1. Mejoras

Si se tiene en cuenta los valores máximos de cada división se podría llegar a utilizar enteros de menor tamaño para poder procesar de a más cantidad de datos. Por ejemplo si se usan enteros de 2Bytes se puede procesar hasta de a ocho por registro xmm.

Se podría hacer la versión 3D. La complejidad en cuanto al algoritmo e implementación no es muy elevada, sí nos pareció complicado dibujar el terreno generado, ya que deberíamos usar herramientas de SDL que no tenemos muy claras. Por esa cuestión y la falta de tiempo para aprenderlas decidimos incluir en este trabajo solo la versión 2D.