DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Final

Introducción a Computación Gráfica

Integrante	LU	Correo electrónico
Kevin Frachtenberg	247/14	kevinfra94@gmail.com
María Laura Muiño	399/11	lauramuino2@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introduccion

En este trabajo, decidimos realizar un Heightmap. En computación gráfica, un mapa de altura o heightmap, es uma imagen rasterizada utilizada principalmente como una grilla discreta para modelar un sector de terreno.
Logramos generar alturas y disposición de picos random, coloración de terreno por altura, controlado por parámetros de interfaz gráfica, iluminación con color y movimiento también contorlado por parámetros de interfaz gráfica.

2. Desarrollo

Para proveer un marco prolijo de trabajo, decidimos crear un mesh que nos permitiera ubicarnos mejor en el plano y brindara mejor interpretación de los resultados que obtuvieramos.

Intentamos dibujar punto a punto las lineas, usando como base el código de los tps de la materia. Sin embargo por varios errores que obtuvimos en el proceso, decidimos comenzar de cero utilizando Three Js. Uno de los errores que obtuvimos fue un corrimiento de las uniones a partir de tamaños iguales o superiores a 17x17, dejamos una imagen de muestra.

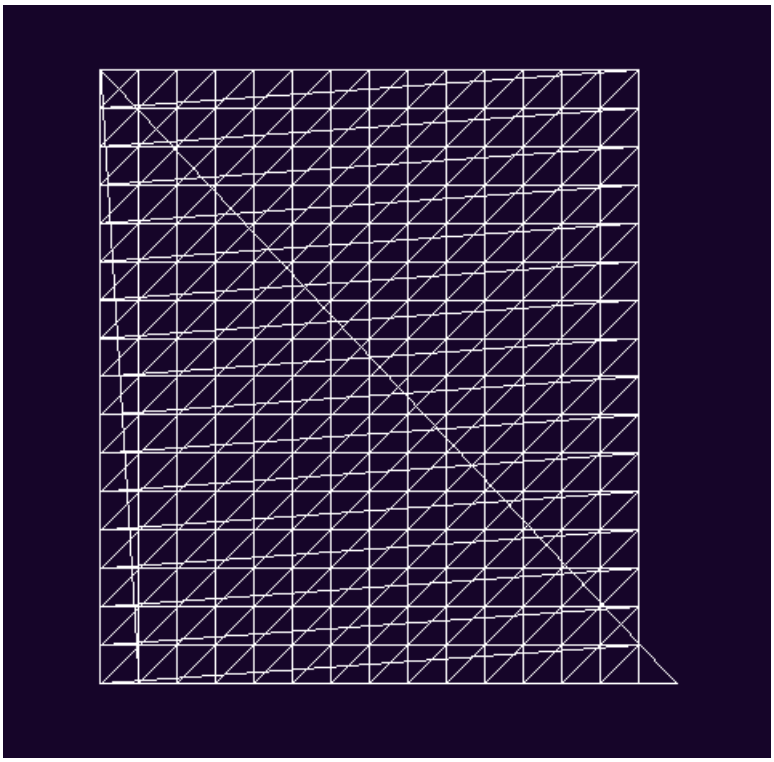

Figura 1: A partir de este tamaño, los mesh empiezan a tener un desplazamiento incorrecto.

Utilizamos algunas referencias[1] para poder completar la creación del mesh con Three Js. Adicio-

[1] https://blog.mastermaps.com/2013/10/terrain-building-with-threejs.html

nalmente, agregamos el dibujo de los ejes de coordenadas, el movimiento de la cámara respecto al eje y una luz de tipo *PointLight*.

Con este sitio[2] logramos arreglar un problema visual de luces que surgió por una rotación incorrecta de ejes en el plano (queriamos intercambair el plano z por el y para representar la altura). También descubrimos el tipo de luz que queriamos usar, cómo funcionaba y agregamos un objeto que representaba el origen de la luz, para ganar visibilidad. Luego optamos por un comportamiento automático para la rotación, y el plano giraba correctamente y las luces se comportaban bien.

2.1. Alturas

Para obtener resultados visibles rápidamente, decidimos comenzar utilizando generación de alturas random, definiendo un límite para la altura máxima. Las alturas para cada vértice las determinamos en la geometría, obteniendo la siguiente figura como resultado.

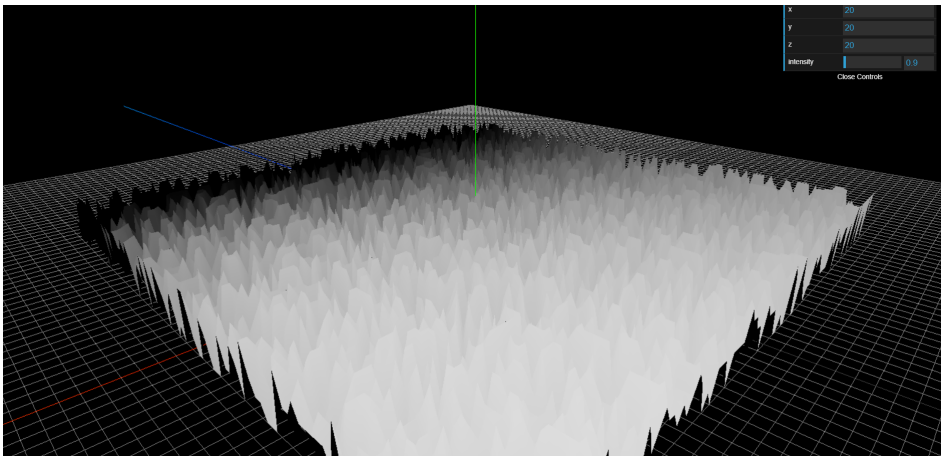

Figura 2: Terreno generado con alturas al azar.

Luego, procedimos a probar la generación de terrenos montañosos eligiendo sus picos en puntos random, y en los cuales a partir de un cierto radio, generar las montañas (o hills)[3]. Este fue un acercamiento interesante ya que logramos producir terrenos montañosos de manera pseudo-random, repitiendo el algoritmo varias veces y así pudiendo obtener resultados como el de la siguiente figura.

[2]https://threejsfundamentals.org/threejs/lessons/threejs-lights.html
[3]http://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html

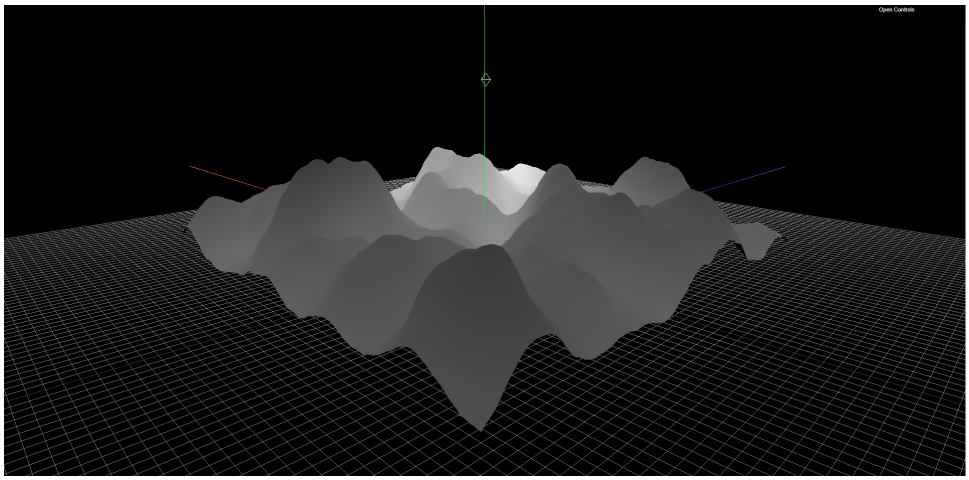

Figura 3: Terreno generado con alturas al azar.

Teniendo ya un mapa definido, procedimos a agregarle color.

2.2. El color

Al comenzar con el coloreo, intentamos hacerlo calculando y determinando los colores directamente con un Fragment Shader escrito por nosotros. Haciendo eso, logramos pintar el mapa tanto con un solo color como con degradé de colores. Sin embargo, se nos dificultó la interacción con las luces, ya que no encontramos buena documentación sobre cómo funciona Three.js cuando se usan shaders propios. Si bien encontramos algunos ejemplos en internet, muchas veces tenían versiones viejas de la librería o incluso hacían cosas específicamente distintas, lo cual no nos servía para poder avanzar con nuestro proyecto.

Como anteriormente ya habíamos realizado cambios en la geometría calculando las alturas de cada punto del terreno fuera del shader, decidimos intentar utilizar el mismo mecanismo para asignarle colores a los vértices en el mismo momento que modificamos la altura. Inicialmente definimos 3 secciones (blanco, marrón y verde) para poder observar algo relativamente sencillo. Calculamos en qué sección cae cada vértice según la diferencia de altura que hay entre él y la altura máxima del terreno.

A partir de ahí tuvimos dos problemas: el primero fue que Three.js no tomaba los colores definidos por la geometría. Encontramos un ejemplo donde se hacía esto, pero, al igual que en otros casos, empleaba una versión vieja de la librería por lo cual no podíamos implementarlo de la misma forma. Al encontrar la forma de configurarlo correctamente y poder ver el mapa con coloreo basado en altura como se ve en la Figura 4, quisimos poder cambiar el coloreo de forma dinámica a través de la interfaz. Aquí surgió el segundo problema, que fue que no lográbamos hacer que el mapa se actualice dinámicamente. Esto se solucionó cuando descubrimos que Three.js tiene un flag llamado $needsUpdate$, el cual al poner en $true$, la escena actualiza los cambios marcados con ese flag en cada tick.

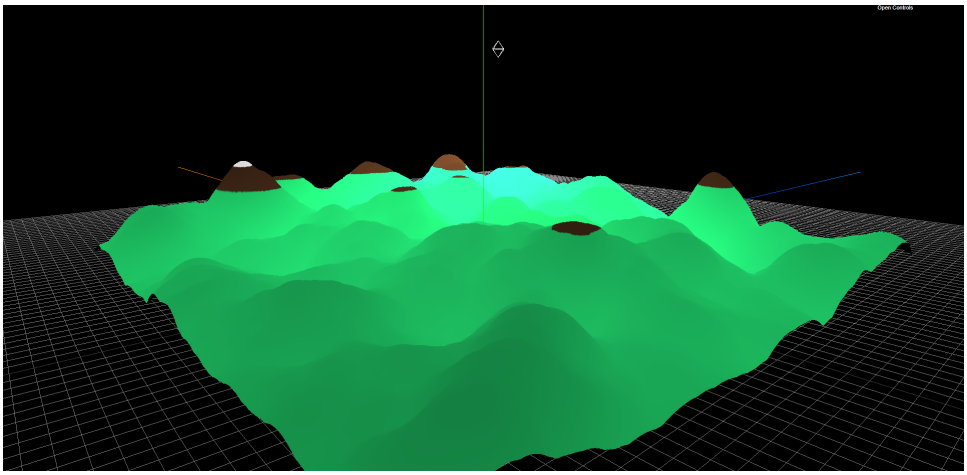

Figura 4: Terreno coloreado con 3 capas.

Habiendo logrado esto último, pudimos acercarnos al objetivo del proyecto logrando tener un terreno generado de manera automática con 3 secciones de colores ubicados según la diferencia de altura de cada punto. A partir de ahí, agregamos 3 secciones más y un nuevo selector para poder cambiar la cantidad de secciones dinámicamente. De este modo, obtuvimos resultados mucho más interesantes como se puede contemplar en la Figura 5.

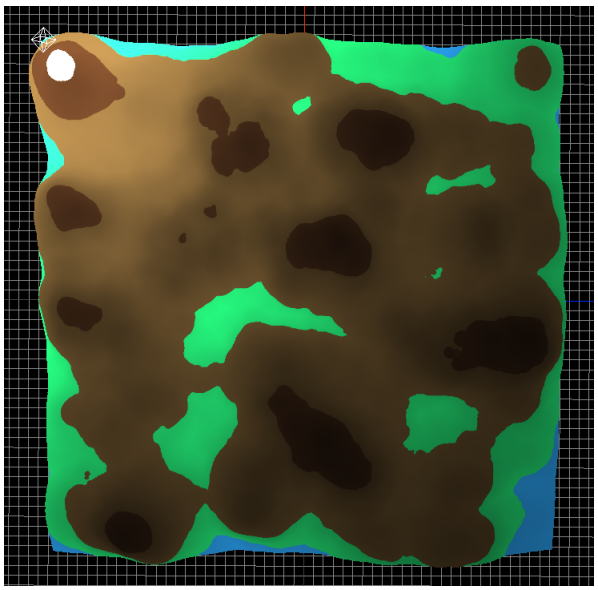

Figura 5: Terreno coloreado en base a altura visto desde arriba.

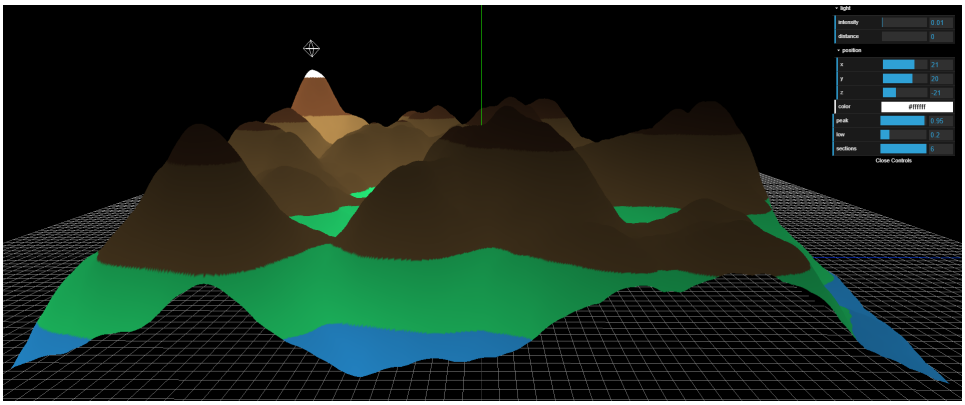

Figura 6: Terreno coloreado en base a altura visto de frente.

Esto también nos ayudó a notar que las generaciones de mapas donde las alturas no eran muy pronunciadas, podían verse como islas separadas, al mejor estilo Age Of Empires.

Otro punto a destacar en esta sección es el juego de luces. Para este proyecto solamente utilizamos el tipo de luz *PointLight* y agregamos 4 configuraciones a la interfaz: Posición, Intensidad, Distancia y Color. Con estas, nos fue posible realizar algunas pruebas para ver la reacción del terreno. A modo de ejemplo, en la siguiente figura se puede observar la luz cuando la distancia es limitada a 20.

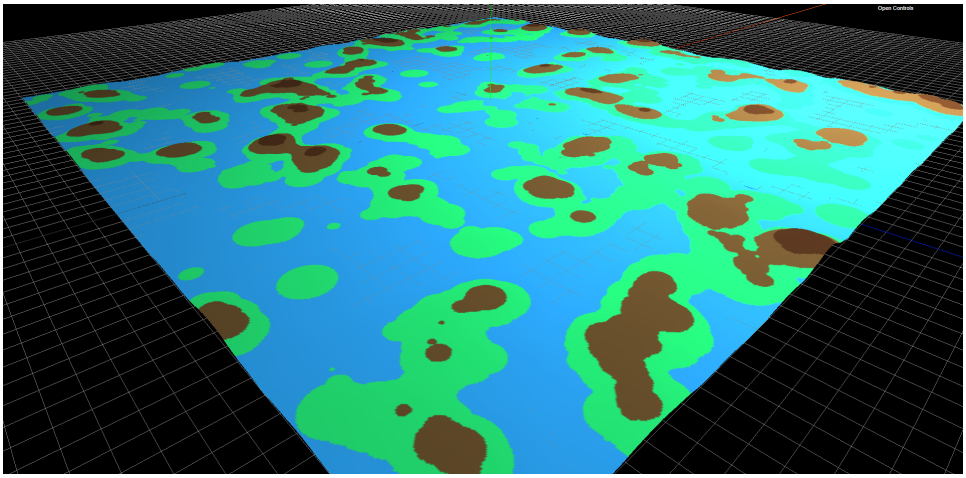

Figura 7: Mapa de islas.

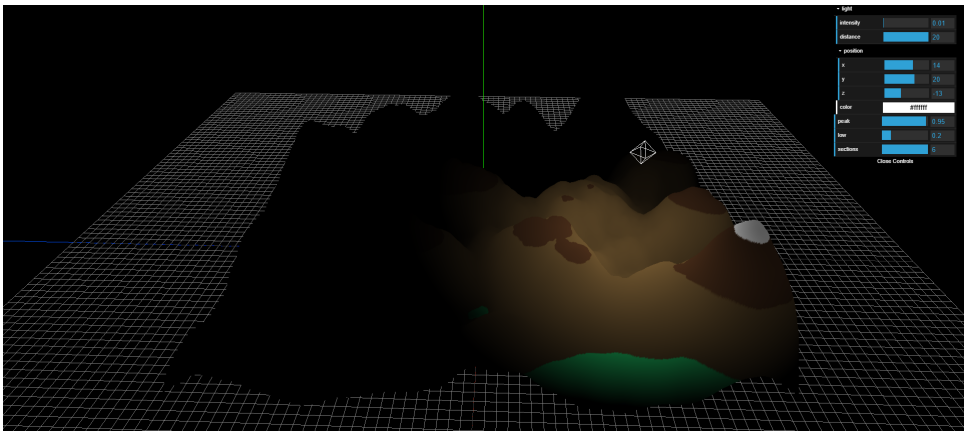

Figura 8: Distancia máxima 20 de la luz

3. Trabajo a futuro

Dado que los picos del terreno se generan de manera random, eventualmente surgen terrenos casi planos, que cuando no habia color, nos resultaba dificil evaluar resultados. Nos gustaria que la generacion del terreno (cantidad de picos, radio de esparcimiento, altura de picos, etc) sea controlada por la interfaz gráfica.

También sería bueno experimentar con mapas generados a partir de ruido, con algoritmos como el de Perlin, o con diferentes texturas para no depender en la totalidad de los colores elegidos a partir de la diferencia de alturas. De esta forma, podríamos generar mapas de distintos tipos (como bosques o desiertos).

Otro tema que queda pendiente es generar más terreno a partir del movimiento de la cámara. Así, se produciría terreno mientras la camara se mueve y daría una sensación de terreno infinito, a diferencia de lo actual, donde tenemos un área fija para el mapa y solo es modificable en el código.