UNIVERSIDAD POLITÉCNICA DE VALENCIA

Trabajo Final de Máster

Creación automática de contenido gráfico

MUIARFID

Presentado por:
Héctor Mayor Giménez
Dirigido por:
Roberto Agustín Vivó Hernando

Valencia, 2015

Resumen

Éste proyecto describe un Generador Automático de Terreno (GAT), una aplicación que genera de forma automática y aleatoria mapas de terreno. Se ha predefinido una serie de tipos de mapas para diferentes entornos, por ejemplo, se ha definido el tipo *isla* que como el nombre indica genera un isla rodeado de agua.
Los mapas de terreno cuentan con vegetación añadida para darle mayor realismo, así como agua para los entornos en los que se precisa.
Todos los entornos tienen infinitas composiciones de terreno, ya que se generan automáticamente y aleatoriamente, por lo tanto es casi imposible encontrarse dos mapas generados iguales, lo mismo ocurre con la vegetación.
En éste documento se describen todas las características usadas en la aplicación, así como los métodos empleados, como se generan los terrenos y sus diferentes tipos implementados.

Abstract

This project describes an automatic generator of land (GAT), an application that automatically generates terrain maps. There are a number of predefined map types for different environments, for example, it has defined the island type, as the name suggests, generates an island surrounded by water. The terrain maps feature vegetation, which is added to give more realism as well as water for the environments in which it is necessary.
All environments have infinite compositions of lands, since they are automatically and randomly generated, so it is almost impossible to find two identical generated maps, the vegetation is also generated automatically and randomly.
This document describes all the features used in the application as well as the methods used, how the land are generated and how the different types are implemented.

Índice

Capítulo 1

1. Introducción, motivación y objetivos

1.1 Introducción

La industria de videojuegos ha experimentado en los últimos años altas tasas de crecimiento, debido al desarrollo de la computación, capacidad de procesamiento, imágenes más reales y la estrecha relación entre películas de cine y los videojuegos, con lo cual los consumidores reconocen los títulos cada vez más pronto. En la década de 2000, los videojuegos han pasado a generar más dinero que la del cine (solo entradas vendidas) y la música juntas, como en el caso de España.

La industria de los videojuegos está aumentando considerablemente últimamente y cada vez es mayor el número de personas que juega habitualmente a videojuegos, y se espera que en los próximos años se incremente aún más. La figura 1.0 muestra cómo ha ido aumentando el beneficio obtenido a través de videojuegos y las previsiones para los próximos años.

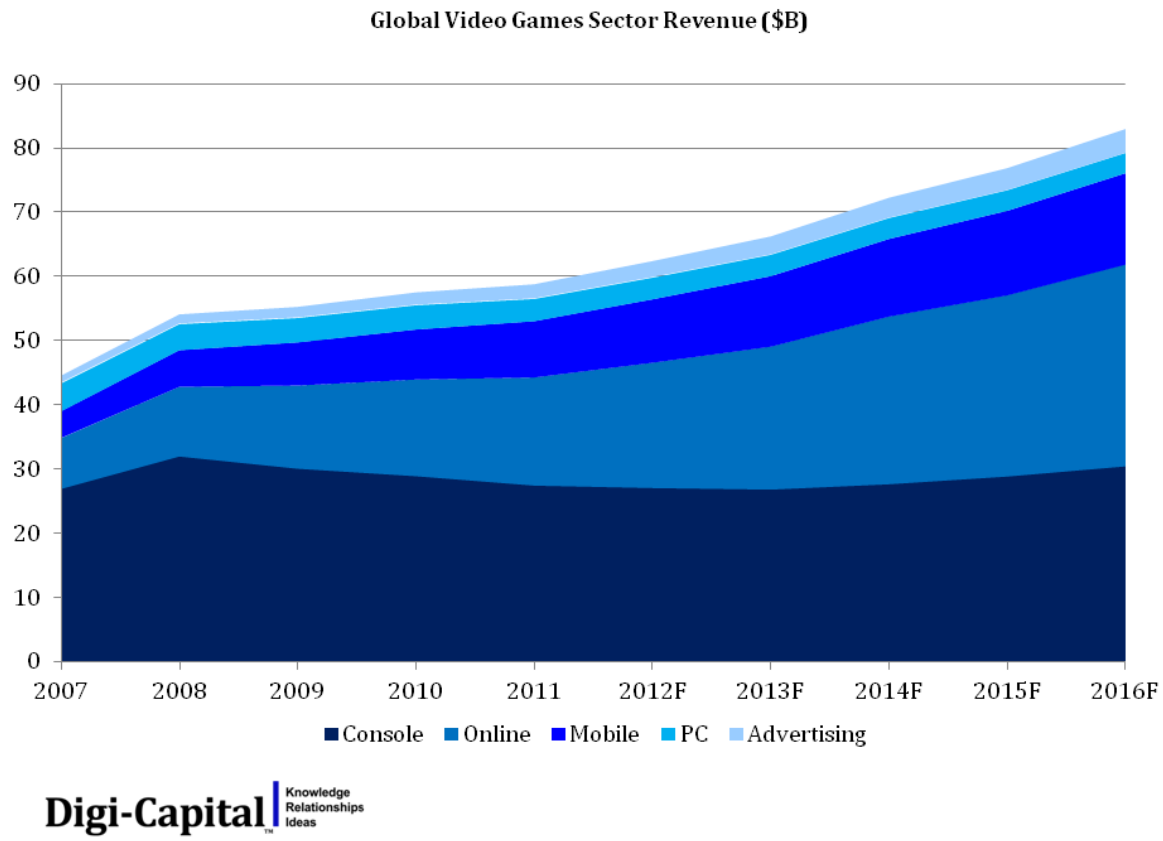


Figura 1: Ventas videojuegos a lo largo de los años

1.2 Motivación

Dada la gran cantidad de videojuegos que se crean anualmente y la necesidad de éstos por crear terrenos en los que se desarrolla la actividad, cuyo coste de tiempo es muy elevado dado que la generación del mismo no es nada trivial y se necesita mucha dedicación para desarrollarlo, ha motivado a la creación de ésta aplicación para generar el terreno de forma automática, con el consiguiente ahorro de tiempo que se podrá dedicar a otras tareas.

Mediante el uso del proyecto desarrollado se podría eliminar el tiempo empleado a la generación de terrenos y dedicarlo a otros menesteres.

1.3 Objetivos

El objetivo de éste proyecto es la creación de un programa que genere un terreno de forma automática, esquematizada y aleatoria, que satisfaga las necesidades de los proyectos que necesitan éste tipo de mapas de entorno.

Se han planteado una serie de objetivos para éste proyecto, así como posibles ampliaciones que pudieran mejorar lo desarrollado en éste proyecto.

A continuación se exponen los objetivos que se pretenden cumplir con este proyecto:

- Generar un mapa de alturas aleatorio que represente el terreno de forma automática.
- Aplicar diferentes texturas al terreno dependiendo del tipo que se trate.
- Incluir vegetación en el mapa generado de forma aleatoria y automática
- Añadir agua para aquellas regiones que se encuentren por debajo del nivel del mar.
- Añadir iluminación para dar forma a la escena.

Todos éstos objetivos se han realizado y se van a explicar detalladamente en otras secciones del documento.

Capítulo 2

2. Estado del arte

Cada vez más los estudios de videojuegos se plantean la creación de herramientas para la creación de terrenos para mejorar su calidad, hacer más fácil su creación, disminuir el tiempo empleado, y por lo tanto disminuir el coste total invertido.

Muchos de los motores de videojuegos más importantes del momento ya nos proporcionan un creador de terrenos, cómo por ejemplo el *Unreal Engine*, el *unity3d* y el *cryengine*. Cada uno de ellos tiene su propio editor de mapas para facilitar al usuario su creación.

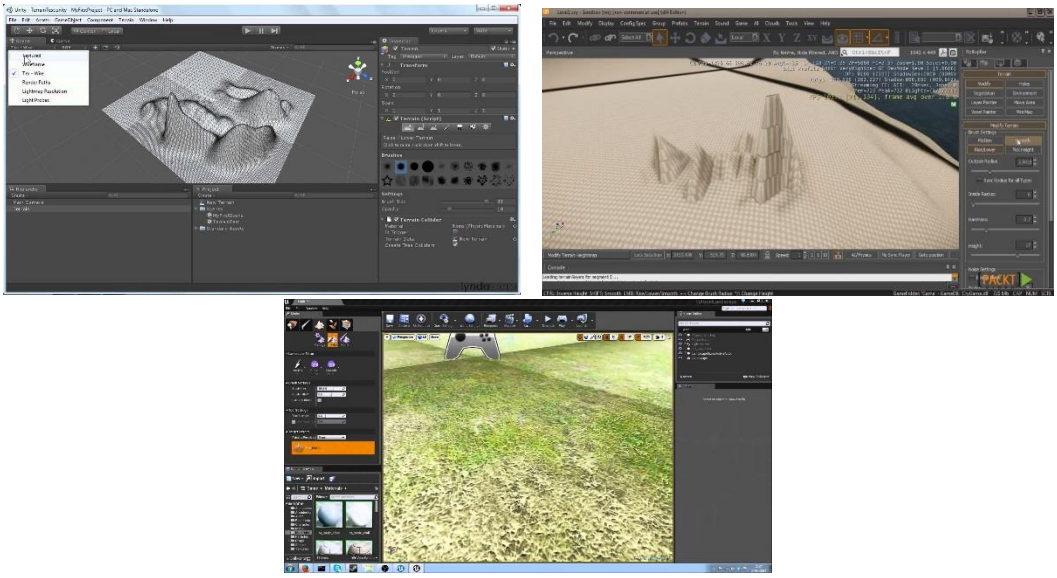


Figura 2: Editores de mapas

El problema viene en que la mayoría de todos estos editores de terreno son manuales y no generan ningún tipo de terreno aleatorio, por lo tanto siempre tendrá que aparecer la figura de un ser humano para crear estos entornos.

La idea de éste proyecto no es crear un editor que facilite la labor del usuario para crear éste tipo de terrenos, lo que se pretende es eliminar al usuario por completo y que los terrenos se generen de forma totalmente automática sin la intervención del usuario.

Capítulo 3

3. Software utilizado

3.1 Introducción

Para llevar a cabo éste proyecto se ha elegido como lenguaje de programación C++ puesto que es uno de los más eficientes y más utilizados en la industria del videojuego. Y como plataforma se ha elegido *visual studio 2013.*

Para la parte gráfica se ha utilizado la librería de openScenegrahp, que funciona bajo opengl pero a un nivel superior a éste, lo que nos ha permitido facilitar un poco las cosas a la hora de representar nuestro modelo en 3D.

3.2 OpenScenegrahp

3.3 Introducción

El OpenSceneGraph es un conjunto de herramientas de alto rendimiento de gráficos 3D de código abierto, utilizado por los desarrolladores de aplicaciones en campos como la simulación visual, juegos, realidad virtual, visualización científica y modelado. Escrito enteramente en C ++ y OpenGL funciona en todas las plataformas Windows, OSX, GNU / Linux, IRIX, Solaris, HP-UX, AIX y sistemas operativos FreeBSD. OpenSceneGraph se ha consolidado como el líder mundial en tecnología de escenario gráfico.

3.3.1 Características

Uso y Mercados

OpenSceneGraph es de código abierto, con gráficos en tiempo real, utilizado por los desarrolladores de aplicaciones en campos que van desde la simulación visual (vuelo, marina, vehículo, simulador de espacio) a la realidad virtual y aumentada, de visualización médica y científica, a la educación y juegos.

Multiplataforma

OpenSceneGraph es multiplataforma, funciona en dispositivos pequeños como plataformas integradas, gráficos para teléfonos, tabletas que utilizan OpenGL ES, ordenadores portátiles y de sobremesa

Licencias

OpenSceneGraph se publica bajo la Licencia Pública OpenSceneGraph, que es una relajación de la licencia Less GNU Public License (LGPL) que permite su uso en aplicaciones comerciales que requieren enlazado estático y en sistemas embebidos.

Tecnología

OpenSceneGraph está escrito en C ++, aprovechando la biblioteca de plantillas estándar (STL) para los contenedores. El software utiliza el enfoque de escenario gráfico para representar mundos 3D como un gráfico de nodos.

OpenGL 1.0 a través de OpenGL 4.2 y OpenGL ES 1.1 y 2.0 son compatibles, con lo que es posible apoyar al hardware, a sistemas operativos antiguos, a los últimos dispositivos móviles y a todas las características gráficas de escritorio más modernas.

Se utilizan patrones de diseño en todo el software por lo que es más fácil de mantener y entender cómo funciona el software, así como proporcionar un buen ejemplo de uso. El software se mantiene modular y extensible, de tal forma que el usuario sólo utiliza los componentes que necesita.

Los puntos fuertes de OpenSceneGraph son su rendimiento, escalabilidad, portabilidad y la productividad.

Rendimiento

Soporta view-frustum culling, occlusion culling, small feature culling, Level Of Detail (LOD) nodes, OpenGL state sorting, vertex arrays, vertex buffer objects, OpenGL Shader Language and display lists como parte del núcleo del grafo de escena. Todo esto junto hace de OpenSceneGraph uno de las mayores herramientas graficas disponibles en cuanto a rendimiento se refiere.

Productividad

El escenario gráfico básico encapsula la mayoría de la funcionalidad de OpenGL incluyendo las últimas extensiones, proporciona optimizaciones tales como el descarte y la clasificación, y todo un conjunto de add-on y bibliotecas que hacen posible el desarrollo de aplicaciones gráficas de alto rendimiento con gran rapidez de procesamiento. El desarrollador de la aplicación se libera para concentrarse en el contenido y la forma en que el contenido es controlado en lugar de la codificación de bajo nivel.

Combinando las lecciones aprendidas de los gráficos de escena establecidos como Intérprete y Open Inventor, con métodos de ingeniería de software moderno como patrones de diseño, junto con una gran cantidad de retroalimentación desde el principio en el ciclo de desarrollo, ha sido posible diseñar una biblioteca que es limpia y extensible. Esto ha hecho que sea fácil para los usuarios a adoptarse a OpenSceneGraph e integrarlo con sus propias aplicaciones.

Portabilidad

El escenario gráfico básico ha sido diseñado para tener una mínima dependencia en cualquier plataforma específica, que requiere poco más que de C ++ y OpenGL. Esto ha permitido que el escenario gráfico pueda ser rápidamente portado a una amplia gama de plataformas - desarrollada originalmente en IRIX, luego portado a Linux, y luego a Windows, a continuación a: FreeBSD, Mac OSX, Solaris, HP-UX, AIX, PlayStation2 y hasta iOS y Android!

El núcleo de la biblioteca gráfica es un sistema de ventanas completamente independiente, lo que hace que sea fácil para los usuarios añadir sus propias bibliotecas y aplicaciones de ventana. En la distribución OpenSceneGraph la biblioteca osgViewer proporciona soporte nativo de ventanas en Windows (Win32), Unix (X11) y OSX (carbono). La biblioteca osgViewer también se puede integrar fácilmente con otros conjuntos de herramientas de ventanas, como Qt, GLUT, FLTK, SDL, WxWidget, Cacao y MFC.

Escalabilidad

El escenario gráfico se ejecuta tanto en sistemas mono núcleo, como en sistemas multinúcleo, sistemas multi-GPU y clúster. Esto es posible debido a que el escenario gráfico núcleo soporta múltiples contextos gráficos para ambas listas de visualización de OpenGL y objetos de textura, y el recortado y dibujado han sido diseñados para almacenar en caché los datos de representación a nivel local y utilizar el escenario gráfico casi en su totalidad como una operación de sólo lectura. Esto permite que varios pares de recortado-dibujado se ejecuten en múltiples CPU’s.

Capítulo 4

4. Diseño

4.1 Introducción

Para la creación del terreno se ha creado una clase genérica que inicializa el mapa de alturas, para ello se utiliza el método *simplex noise*. Éste método nos permite una inicialización de ruido con el que trabajaremos posteriormente y nos servirá para crear una variabilidad en el terreno.

4.2 Simplex Noise

Simplex noise es un método para construir una función de ruido n-dimensional comparable al ruido *perlin noise* (ruido "clásico"), pero con un menor número de artefactos de dirección, mayores dimensiones y una sobrecarga computacional inferior.

El *perlin noise* es una función matemática que utiliza interpolación entre un gran número de gradientes pres calculados de vectores que construyen un valor que varía pseudo aleatoriamente en el espacio o tiempo. Se parece al ruido blanco, y es frecuentemente utilizado en imágenes generadas por computadora para simular variabilidad en todo tipo de fenómenos, acercándose estas así a un aspecto más natural.

Las ventajas de *simplex noise* sobre el *perlin noise son*:

- *Simplex noise* tiene una complejidad computacional menor y requiere menos multiplicaciones.
- *Simplex noise* escala a dimensiones superiores (4D, 5D) con mucho menos coste computacional, la complejidad es O (n ^ 2) para n dimensiones en lugar de la O (2 ^ n) de ruido clásico.
- *Simplex noise* no tiene artefactos direccionales sensibles (es visualmente isotrópico), aunque el ruido generado por las diferentes dimensiones son visualmente distintas (por ejemplo, el ruido 2D tiene un aspecto diferente al de las rebanadas de ruido 3D, y se ve cada vez peor para dimensiones superiores.
- *Simplex noise* tiene un gradiente bien definida y continua, lo que hace que se pueda calcular fácilmente.
- *Simplex noise* es fácil de implementar en hardware.

4.3 Terreno

4.3.1 Introducción

Como ya se ha mencionado, se ha creado una clase genérica que nos proporciona una matriz con ruido. Para generar el mapa de alturas se han creado diferentes tipos de terrenos. Cada uno de ellos tiene una inicialización diferente y se comportan según lo establecido en cada tipo, siendo su generación aleatoria y automática, de forma que es casi imposible encontrarse dos terrenos iguales, incluso dentro del mismo tipo.

El número de tipos de terreno que podemos crear es ilimitado, para éste trabajo se han creado cinco tipos: Isla, rio, desierto, archipiélago y cordillera.

Gracias a la clase genérica podemos implementar tantos tipos de terreno como quisiéramos, nos ofrece una serie de métodos comunes que nos proporcionan las bases para generar el mapa de alturas. Para crear otro subtipo lo único que tenemos que hacer es implementar un método en el que tenemos que indicarle las reglas específicas de éste subtipo.

Se han creado varios métodos para generar el terreno, que serán utilizados por los distintos tipos de terreno según convenga:

El primero de ellos ha sido para generar un relieve a baja altura, que será la base para casi todos los tipos de terreno implementados. Para ello se ha creado un algoritmo que contempla como atributos los límites del terreno en donde se va a aplicar.

Lo que hace dicho algoritmo es ir creando mini elevaciones del terreno aleatoriamente dentro de los límites indicados, las elevaciones tienen un tamaño predefinido, de tal forma que en el centro la elevación es mayor que en las esquinas. Al crear muchas elevaciones y ser muy poco elevadas se consigue el efecto deseado, creando una base con un cierto relieve.

Por otra parte se ha definido otro algoritmo para la generación de las montañas, que es muy similar al mencionado anteriormente, salvo con la salvedad de que el número de elevaciones es mucho menor y las elevaciones son más altas para dar el efecto de montaña.

Estos dos algoritmos configurados de manera diferente resultan en los diferentes tipos de terreno que se han implementado.

Por otra parte, se ha creado un nivel en el cual se encuentra el agua, y cualquier punto en el mapa de alturas que no sobrepase dicho nivel estará por debajo del agua y se visualizará como tal.

4.3.2 Tipos de terreno

4.3.2.1 Isla

Se trata de un tipo de terreno que consta de una isla en el centro que está rodeada de agua.

Para generar éste tipo de terreno se ha utilizado sólo el algoritmo base, se han establecido los límites del algoritmo de forma que la parte central de la isla quedara por encima del nivel del mar y los extremos quedaran sumergidos en el agua. Aparte se ha añadido la vegetación de forma aleatoria y se han aplicado las texturas correspondientes a éste tipo de mapa, así como los colores, dependiendo de la altura a la que se encuentren cada punto dentro del mapa de entorno.

Figura 3: Isla

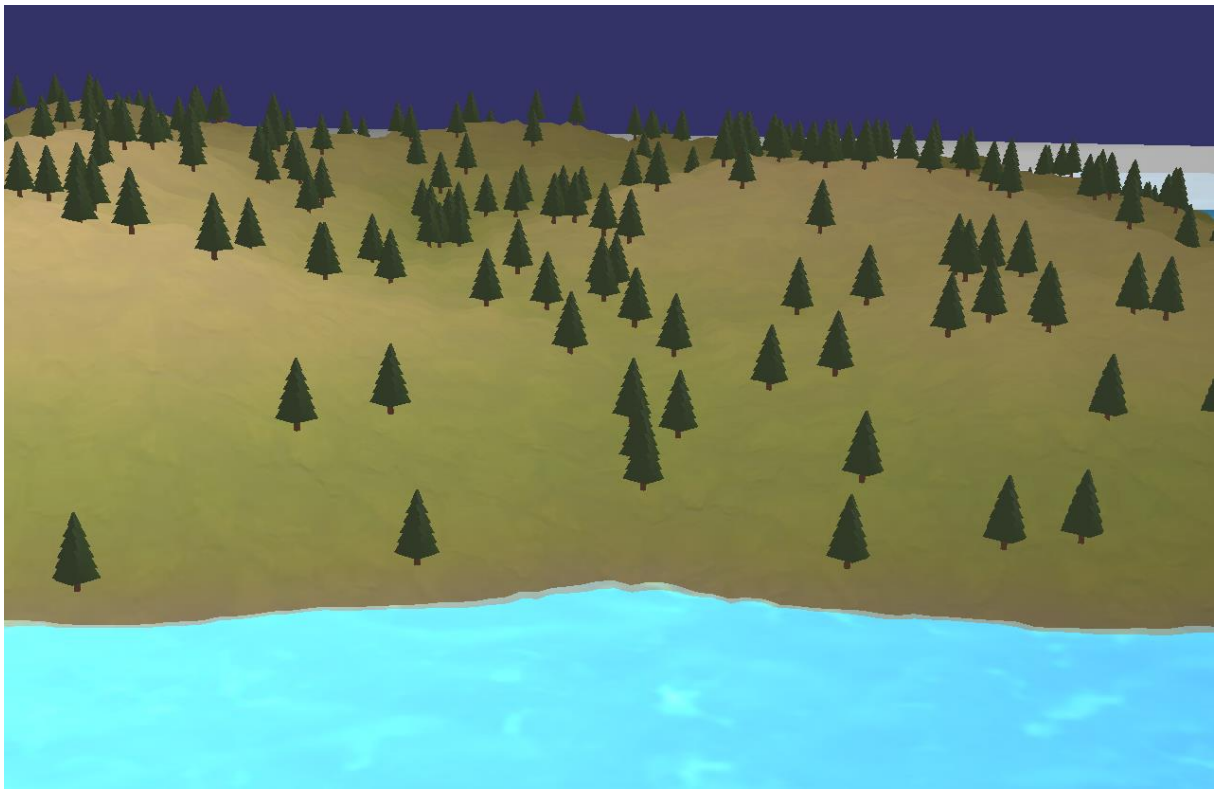


Figura 4: Isla con vista horizontal

4.3.2.2 Río

Para generar éste tipo de terreno se ha utilizado sólo el algoritmo base, se han establecido los límites del algoritmo de forma que la parte central del mapa está sumergida simulando un río.

Se ha reforzado el algoritmo en los extremos del mapa, puesto que al generarse aleatoriamente las elevaciones, en los extremos se produce el efecto de que tiende a estar a una altitud menor, puesto que en las zonas fuera del mapa no se aplica el algoritmo.

En las siguientes figuras podemos observar cómo queda éste tipo de terreno.

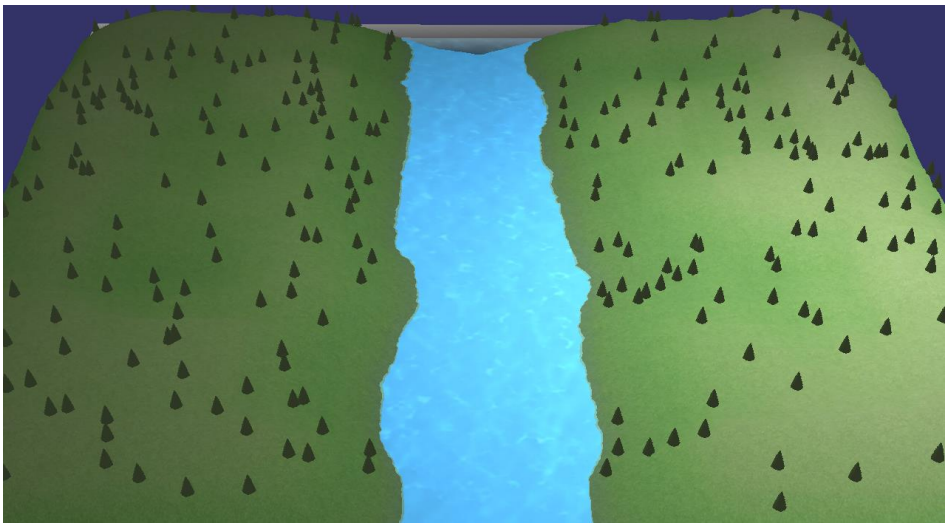


Figura 5: Rio desde arriba

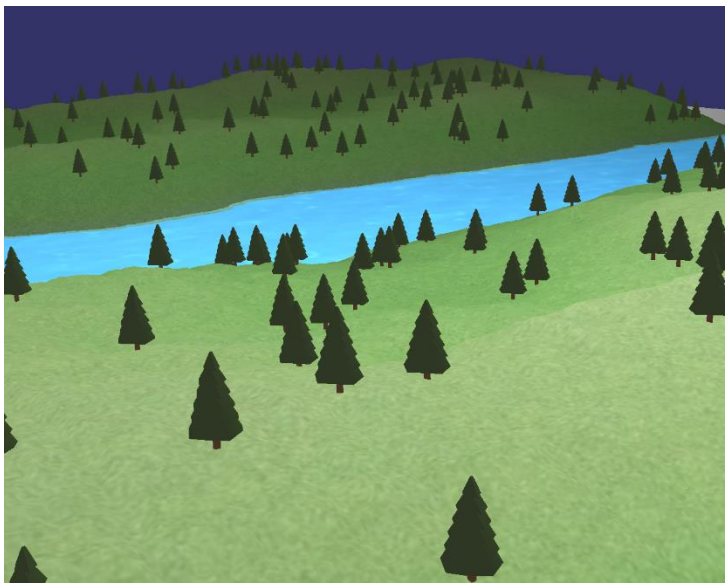


Figura 6: Rio con vista horizontal

4.3.2.3 Desierto

Éste tipo de terreno simula un posible desierto, en el que se incluye una vegetación distinta, los cactus, que se generan automáticamente de la misma forma que se generan los árboles.

Para generar éste tipo de terreno se ha utilizado sólo el algoritmo base, no se han puesto límites puesto que este tipo de terreno es todo tierra y no contiene agua.

Las texturas utilizadas también son diferentes para que dé efecto de arena, y no se diferencia el color en base a la altura del terreno.

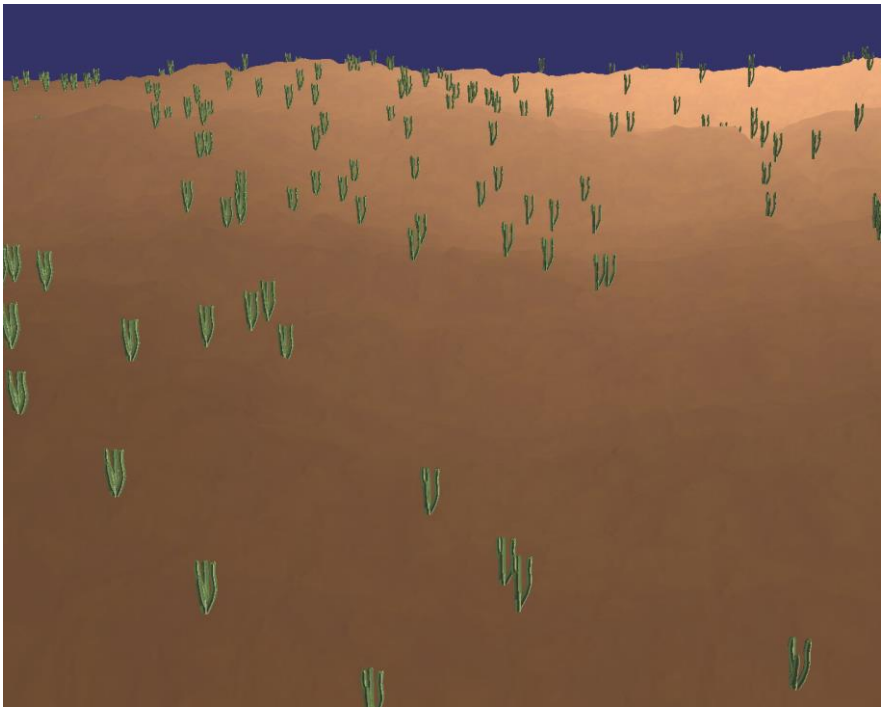


Figura 7: Desierto desde arriba

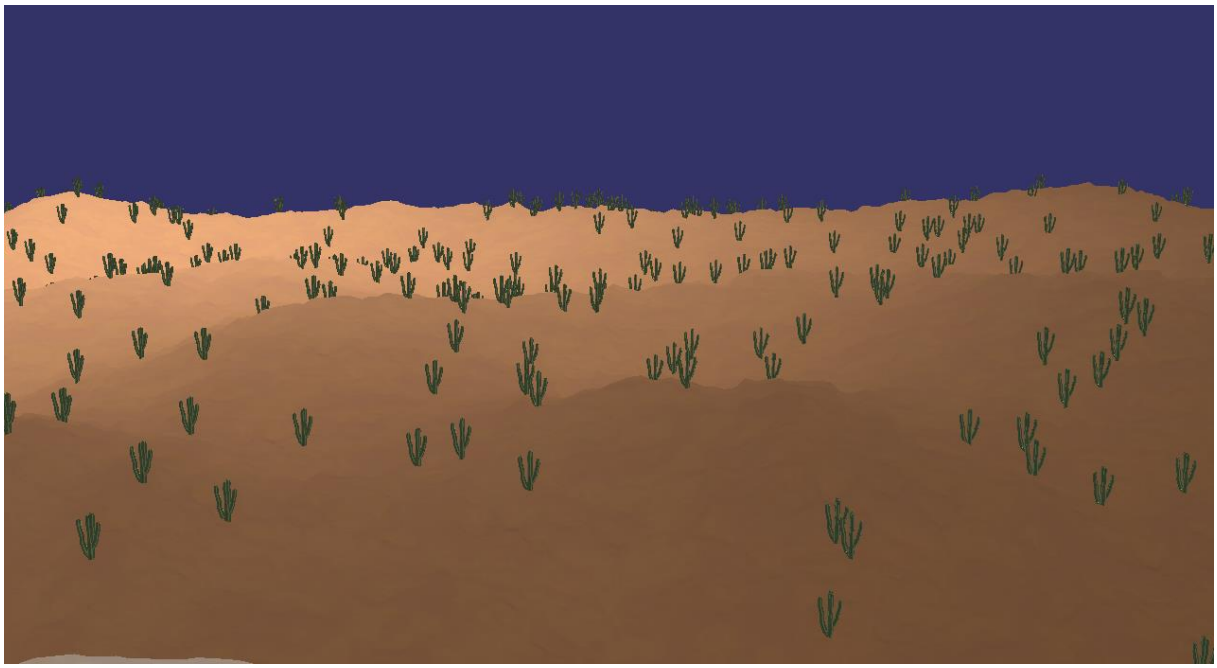


Figura 8: Desierto con vista horizontal

4.3.2.4 Archipiélago

Éste tipo de mapa ha sido creado para mostrar los reflejos que se producen en el agua.

Para éste tipo no se ha utilizado el terreno base, puesto que la sensación que queremos transmitir es la de un archipiélago con distintas montañas rodeadas de agua. Para ello se ha utilizado el método para crear montañas y se ha aplicado una textura acorde para ello.

Éste mapa es el mejor ejemplo para observar cómo se producen los reflejos en el agua tal y como se puede ver en las figuras.

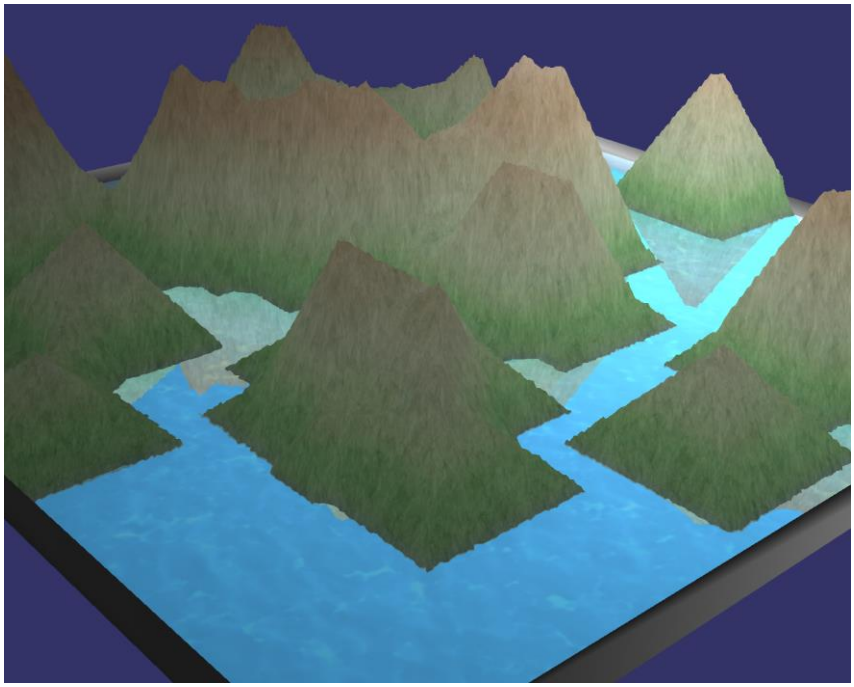


Figura 9: Archipiélago

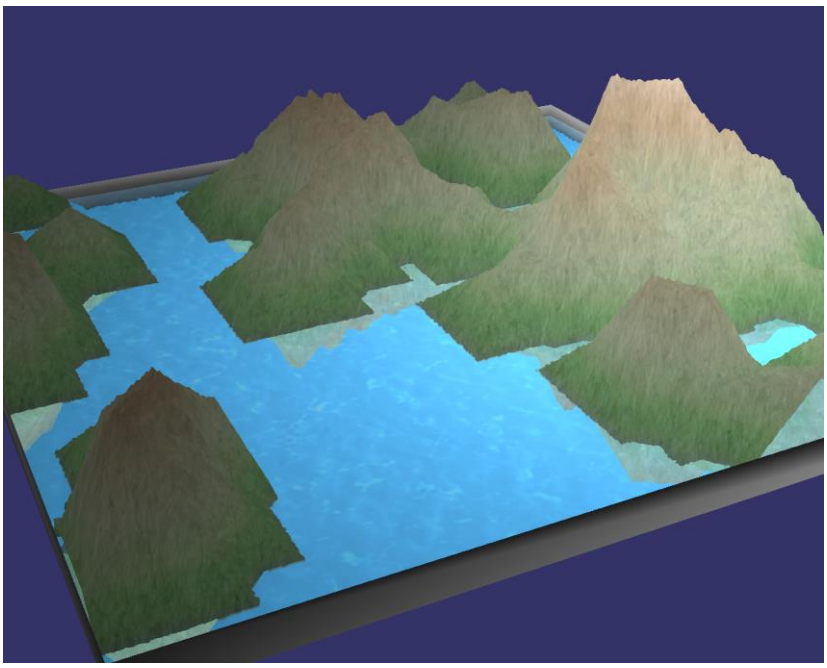


Figura 10: Archipiélago con reflejos

4.3.2.5 Cordillera

Éste tipo de terreno también utiliza el algoritmo principal para generar la base. Aparte se utiliza el método para generar las montañas de forma similar al del archipiélago pero con una altura un poco menor.

Como en los demás mapas se añade la vegetación y se le aplica la textura más adecuada.

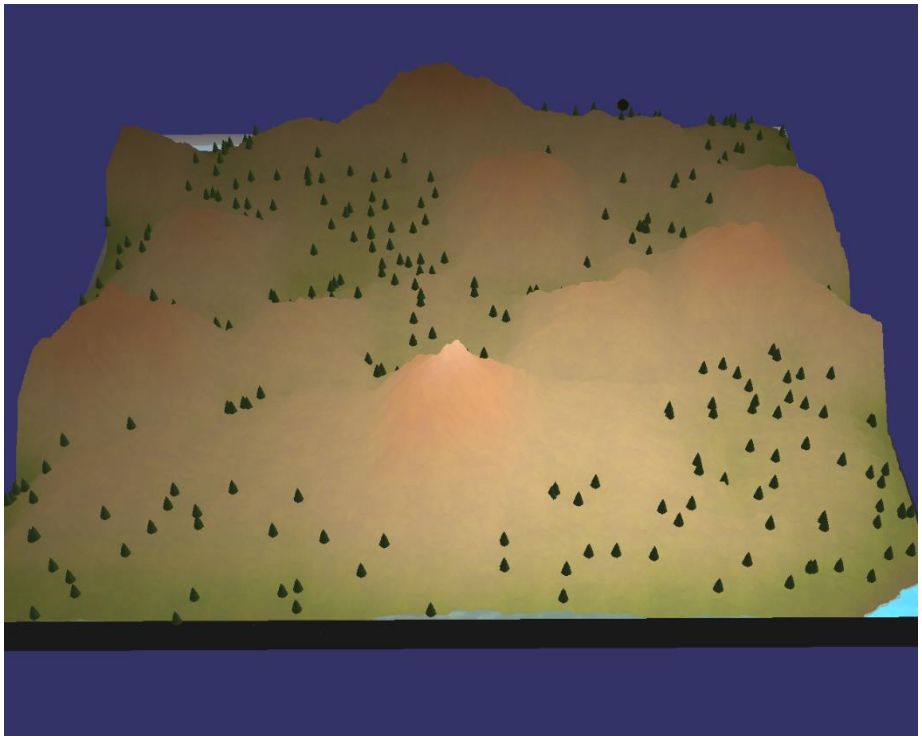


Figura 11: Cordillera desde arriba

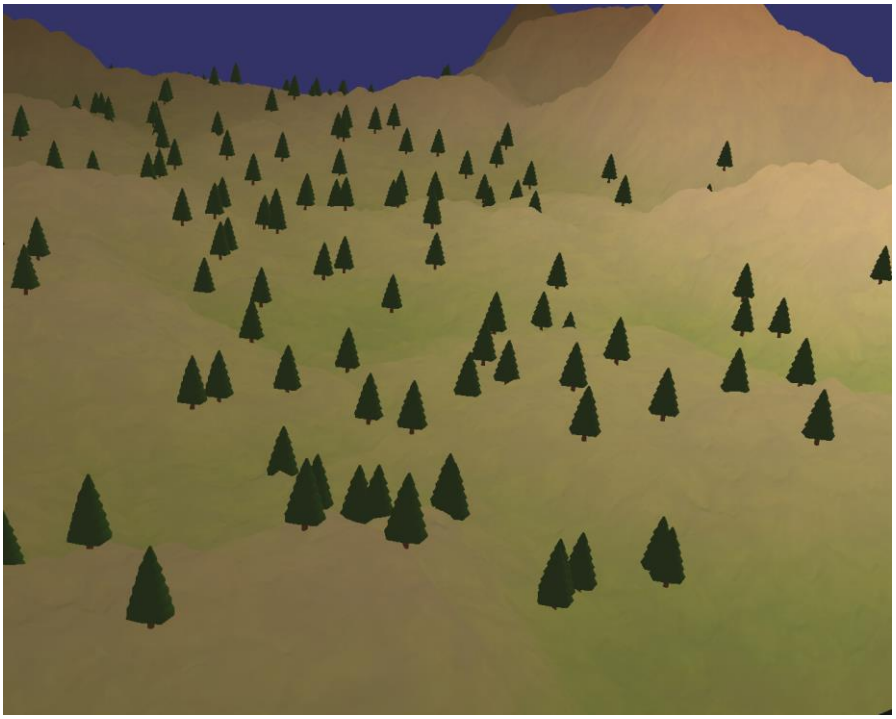


Figura 12: Cordillera con vista horizontal

4.4 Vegetación

4.4.1 Introducción

Los modelos de vegetación utilizados para éste proyecto han sido recopilados de internet y tienen licencia gratuita. Openscenegraph es capaz de leer modelos en 3D en diferentes formatos, pero para un uso óptimo de la herramienta se recomienda pasarlos a un formato propio *.osg*.

Para ello openscenegraph nos ofrece una herramienta para convertir otros tipos de modelos 3D en su formato. La herramienta de conversión no es perfecta y tiene dificultades y errores en pasar algunos modelos, pero funciona medianamente bien para la mayoría de los modelos.

La herramienta llamada *osgconv* es un ejecutable fácil de usar que basta con pasarle el modelo a convertir, por su parte tiene más opciones para modificar la posición, rotación, escalado etc…

4.4.2 Método aplicado

La vegetación predeterminada ha sido de un árbol sencillo y de pocos polígonos, ya que se ha probado con modelos más complejos pero bajaba mucho el rendimiento y se ha decidido trabajar con modelos con un número bajo de polígonos.

Para introducir la vegetación en el mapa se define el número de árboles que se van a generar y se ejecuta un algoritmo con las siguientes características:

- Se van colocando uno a uno los arboles hasta que estén todos localizados.
- Los arboles siempre tienen que estar sobre el nivel del mar.
- Cada tipo de terreno además impone sus propias reglas, como por ejemplo el de la cordillera que no se colocan en cierta altitud.
- El tipo desierto tiene un tipo de vegetación especial, se han cambiado los arboles por cactus.

4.5 Iluminación

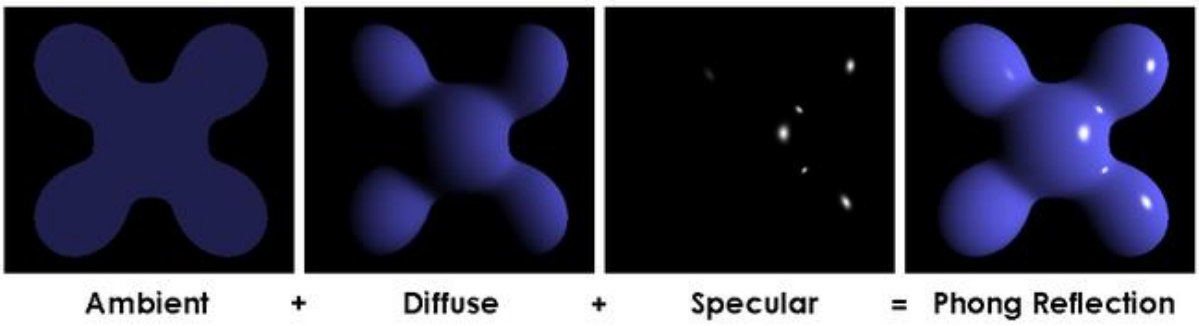
4.5.1 Introducción

Se ha creado un sistema de iluminación basándose en el algoritmo de *phong*, para ello se ha creado una fuente de luz puntual que se mueve de manera circular por el terreno y que es posible detenerla para poder observar con mayor detalle la escena.

4.5.2 Algoritmo de phong

Phong shading se refiere a una técnica de interpolación para superficies en gráficos por ordenador en 3D. También se le llama interpolación Phong. En concreto, se interpola normales a la superficie a través de polígonos rasterizados y calcula colores de los píxeles en base a las normales interpoladas y un modelo reflexión.

Phong es un modelo empírico de la iluminación local. Se describe la forma en que una superficie refleja la luz como una combinación de la reflexión difusa de las superficies ásperas con la reflexión especular de superficies brillantes. El modelo también incluye un término ambiente para dar cuenta de la pequeña cantidad de luz que se dispersa sobre toda la escena.



El algoritmo de *phong* se puede implementar tanto en el shader de vértices como en el de fragmentos, para ésta práctica se ha optado por implementarlo en el shader de fragmentos ya que genera una iluminación de mayor calidad al aplicarse el algoritmo a cada pixel de la pantalla, en lugar de a cada vértice de los polígonos.

4.6 Agua

4.6.1 Introducción

Para la generación de terreno es necesario la simulación de agua, ya que aporta realismo al terreno y es necesaria para generar algunos tipos de terreno que se han implementado.

Para la simulación del agua se han utilizado varias técnicas que se detallan a continuación.

4.6.2 Método aplicado

Para definir el agua se ha creado una malla que simula el oleaje, para ello se utiliza una imagen en escala de grises que representa la altura de cada uno de los puntos de dicha malla. Para realizar el movimiento de las olas con respecto al tiempo, lo que se hace es "mover" la imagen conforme va pasando el tiempo para definir las nuevas alturas a los puntos de la malla, de tal forma que se crea una especie de oleaje.

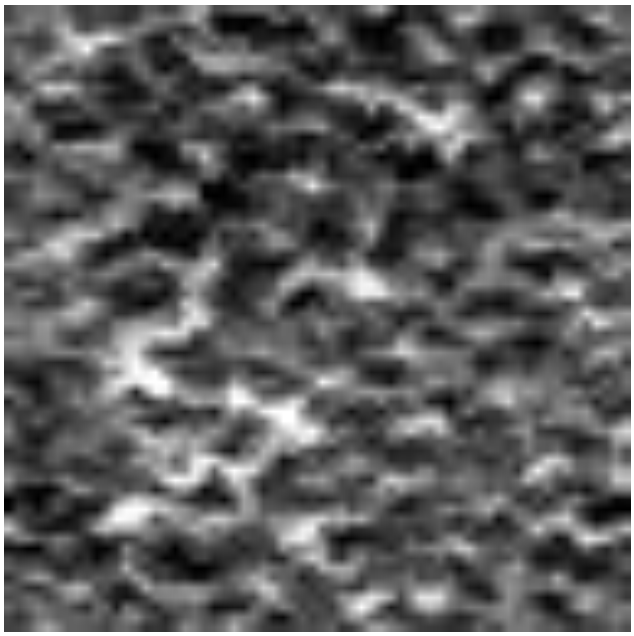


Ilustración 1: Imagen que representa el oleaje

Para texturizar el agua se utiliza una imagen semitransparente para que realce más el efecto del agua.

Por otra parte se ha simulado el reflejo del terreno en la malla del agua, para ello se ha utilizado la siguiente técnica:

- Se define un esténcil sobre la malla del agua, para que el terreno que se pintará más adelante solo se vea a través de la malla del agua.
- Se configura el buffer de profundidad para que sea la mayor posible. Esto se hace para que cuando se valla a dibujar el terreno en la siguiente fase no interfiera con el resto de la escena.
- Se dibuja el terreno volteado sobre el esténcil.
- Se dibuja la malla que representa al agua.

Por otra parte en el shader que se trata y se modifica la posición de la imagen respecto al tiempo, se crea una modificación del terreno invertido para que concuerde con el movimiento de las olas, para ello se modifica la posición z de los vértices del terreno invertido variándolo en proporción con la altura de la ola.

También se crea un ruido sobre el terreno invertido, para ello se utiliza una imagen para definir la cantidad de dicho ruido en cada punto del terreno, éste ruido se aplica a la componente x e y de los puntos del terreno.

Todo ésto, resulta en un reflejo sobre el agua que se asemeja bastante a cómo sería en la realidad.

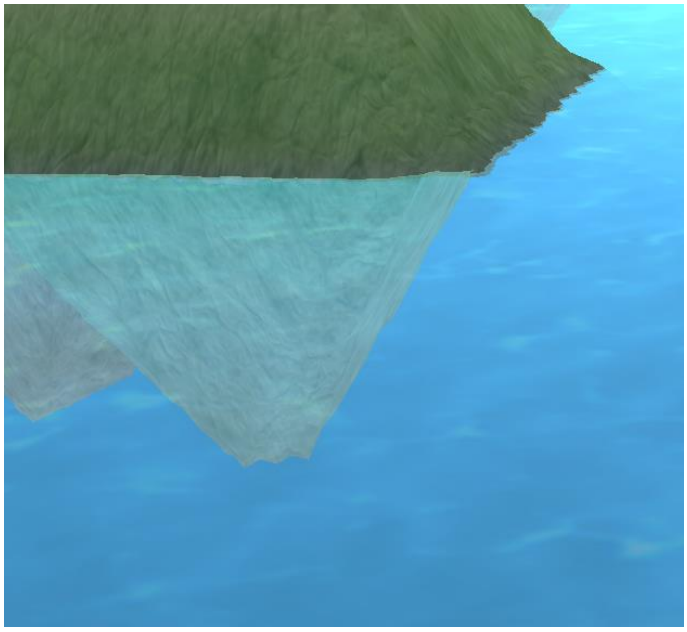


Figura 13: Reflejos

Capítulo 5

5. Conclusiones

5.1 Introducción

En éste proyecto se ha conseguido crear una base para un generador de terreno automático que podría dar paso a una herramienta que fuese utilizada por las empresas de videojuegos que necesiten crear entornos aleatorios dentro de unos patrones para permitir la rejugabilidad de sus juegos.

Esto sería un gran avance, ya que permitiría a muchas empresas no tener que preocuparse más de lo que deberían en crear los diferentes mapas del juego para que sean diferentes, ya que bastaría con crear el tipo en cuestión, y a partir de ahí se irían creando automáticamente los diferentes subtipos.

Éste proyecto me ha hecho ver que es posible la creación automática de terreno, y que en un futuro no muy lejano, cuando se estudie más acerca del tema y las desarrolladoras se metan de lleno en ello , todas las compañías utilizarán éste tipo de software, ya que les facilitaría mucho el trabajo y ahorrarían mucho tiempo y dinero.

5.2 Trabajo futuro y mejoras

Generador Automático de Terreno (GAT) es una aplicación en proceso de desarrollo sobre la que se va a seguir trabajando y añadiendo las siguientes mejoras:

- Extender la vegetación para añadir nuevos tipos de árboles para crear otros posibles entornos.
- Añadir vegetación a ras del suelo, es decir, hierba y otros tipos de vegetación similares.
- Añadir vegetación acuática dentro del agua o sobre ella.
- Añadir sombras al entorno, para realzar más el terreno y darle mayor realismo.
- Crear un editor de tipos de terreno, no de terreno, sino un creador de tipos de terreno, es decir, que el usuario sea capaz de definir un tipo de terreno igual que los que se han implementado en éste proyecto, pero mediante una herramienta con interfaz gráfica.
- Añadir fauna que no interfieran sobre el mapa, como por ejemplo pájaros, peces, etc...

6. Bibliografía

[1] Carlos Requena Farinós. Análisis de la industria del videojuego en España.

[2] Rui Wang, Xuelei Qian .Openscenegraph 3.0: Beginner's Guide.

[3] DEV. Libro blanco del desarrollo español de videojuegos .

[4] Dave Shreiner, Bill Licea-Kane, Graham Sellers, John M. Kessenich. OpenGL Programming Guide: The Official Guide to Learning OpenGL.

[5] Graham Sellers, Richard S.Wright, Nicholas Haemel. OpenGL SuperBible.

[6] Bill M. Licea-Kane, Randi J. Rost, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, Mike Weiblen. OpenGL Shading Language.

[7] Kevin Hawkins, Dave Astle. OpenGL Game Programming.