Universitat Politècnica de València
Departamento de Sistemas Informáticos y Computación

Generación automática de entornos naturales

Trabajo Final de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor: Eduardo Villa Valdés

Director: Roberto Agustín Vivó Hernando

Fecha 13/07/2015

Abstract/Resumen

The environment in which the action of a graphical application development, such as a video game, a simulator or an animated film, is an important part of the work at the moment of its creation, it requires great efforts that could be focused on the key elements of the main action. Therefore, environment procedural generation is especially useful when you don't have enough time or personnel to create it or when an unlimited or different map is desired in each session. To address this problem it has been built a generator simulating a natural environment through configurable functions, through which you can create different types of terrain only by changing the textures used and varying the parameters. To this terrain, which can be explored through an avatar, it has been added vegetation, which arrangement depends on the part of the terrain, its height and inclination, a mass water and a horizon. It has been achieved in this way a complete and immersive environment with natural look to start the creation numerous applications.

El entorno en el que se desarrolla la acción de una aplicación gráfica, como puede ser un videojuego, un simulador o una película de animación, supone una parte importante del trabajo a la hora de su creación, lo que requiere grandes esfuerzos que se podrían centrar en los elementos clave que componen la acción principal. Por ello, la generación automática del entorno es especialmente útil cuando no se dispone del tiempo o el personal necesario para crearlo o cuando se desea un mapa ilimitado o diferente en cada sesión. Para abordar este problema se ha construido un generador de entorno simulando un terreno natural mediante funciones parametrizables, gracias a las cuales se pueden crear distintos tipos de terreno únicamente con el cambio de las texturas utilizadas y la variación de los parámetros. A este terreno, que puede ser explorado mediante un avatar, se le ha añadido vegetación, cuya disposición depende de la zona del terreno, su altura y su inclinación, una masa de agua y un horizonte. Se ha conseguido de esta manera un entorno completo e inmersivo con aspecto natural del que partir para crear numerosas aplicaciones.

Keywords/Palabras Clave: procedural generation, terrain, vegetation, virtual reallity, videogames, graphics / generación automática, terreno, vegetación, realidad virtual, videojuegos, gráficos

Índice

Introducción 7
Motivación 7
Objetivos 7
Estructura de la memoria 8
Antecedentes 9
Descripción del problema 9
Estado del arte 9
Mapa de alturas 10
Agua: Ríos, Océanos y lagos 11
Plantas: modelos y distribución 12
Carreteras y caminos 13
Ciudades y edificios 13
Conclusiones 14
Diseño e implementación de la solución 15
Análisis 15
Tecnologías y conceptos 15
WebGL 15
Three.js 16
Diseño 16
Módulo de entrada de datos y elementos básicos de una escena 17
Avatar 18
Terreno 19
Vegetación 26
Sombras 31
Horizonte 32
Agua 34
Escena final 37
Estructura del software 37
Conclusión 38
Pruebas y resultados 39
Conclusiones 42
Objetivos alcanzados 42

Líneas Abiertas 43
Posibles usos de la herramienta 44
Bibliografía 45
Apéndices. 46
Apéndice 1. Imágenes 46
Apéndice 2. Vídeos y pruebas 47
Apéndice 3. Licencia MIT 48

Figuras

Figura 1. Elevación de mapa de alturas generado mediante Perlin Noise..... 11
Figura 2. Costa generada mediante un algoritmo de inundación..... 11
Figura 3. Distribución de vegetación generada con el ecosistema de Hammes..... 12
Figura 4. Carreteras generadas por el algoritmo de Galin variando costes..... 13
Figura 5. Edificios extruidos por Yong Kyun Roh..... 14
Figura 6. Instrucciones y controles del generador de entorno..... 17
Figura 7. Avatar representando al usuario..... 18
Figura 8. Terreno inicial en forma de parrilla..... 19
Figura 9. Rejilla con nivel de detalle en función de la distancia..... 20
Figura 10. Costura entre zonas con distinto nivel de detalle..... 21
Figura 11. Mismo terreno con 600.000 vértices sin aplicar nivel de detalle (arriba) y aplicándolo (abajo)..... 22
Figura 12. Variación de octavas: una octava (arriba izquierda), cinco octavas (arriba derecha), ocho octavas (abajo izquierda), doce octavas (abajo derecha)..... 23
Figura 13. Fragmento de las texturas utilizadas en el shader de fragmento, ordenadas según se usan en el algoritmo especificado anteriormente..... 25
Figura 14. Escena nevada conseguida únicamente mediante el cambio de todas las texturas por una textura nevada..... 25
Figura 15. Dos de las texturas usadas para representar la hierba (izquierda) y las flores (derecha)..... 26
Figura 16. Paneles de hierba antes (izquierda) y después (derecha) de pasar por el shader de vértices..... 27
Figura 17. Resultado del terreno tras añadir la hierba (arriba izquierda), las flores (arriba derecha) y vista panorámica de éstas (abajo)..... 28
Figura 18. Árbol antes (izquierda) y después (derecha) de aplicar las texturas (centro)..... 29
Figura 19. Escena resultante tras añadir los árboles..... 30
Figura 20. Comparación entre las sombras creadas con el algoritmo por defecto (izquierda) y las creadas tras cambiar el material de las hojas (derecha)..... 32
Figura 21. Efecto de la niebla en la escena..... 33
Figura 22. Comparación entre el fondo en blanco (izquierda) y el fondo con textura de cielo (derecha)..... 34
Figura 23. Efecto horizonte fundiendo el terreno lejano con el cielo (izquierda), efecto niebla fundiendo todos los objetos con la textura del cielo (derecha)..... 34
Figura 24. Diagrama de dibujado del agua. Se pueden ver la cámara y el rayo trazado al punto correspondiente del terreno (en gris), la cámara reflejada con el rayo reflejado (en rojo) y el agua (en azul). Y los puntos del terreno de los cuales se mezcla el color (A y B)..... 35
Figura 25. Masa de agua con reflejos imperfectos y ondas en movimiento (izquierda) y ampliación de la zona donde se encuentra el avatar (derecha)..... 36
Figura 26. Superficie lunar..... 46
Figura 27. Regiones definidas en una textura..... 46

Figura 28. Reflejo de un árbol. 47

Algoritmos

Algoritmo 1. Función de acumulación de ruido. 23
Algoritmo 2. Combinación de colores y texturas en el terreno. 24
Algoritmo 3. Cálculo de la posición y descarte de las briznas de hierba. 28
Algoritmo 4. Cálculo de la posición y descarte de los árboles. 30
Algoritmo 5. Cálculo de la iluminación mediante shadow mapping..... 31
Algoritmo 6. Fases del proceso de renderizado del agua con transparencia, reflejos y olas..... 35
Algoritmo 7. Proceso interno de coloreado del agua. 36

Introducción

Motivación

Actualmente el crecimiento del mercado en el campo de las aplicaciones gráficas interactivas, como pueden ser los videojuegos, ha despertado un especial interés en la generación de gráficos 3D. Estas aplicaciones generan una gran cantidad de puestos de trabajo en grandes estudios dedicados exclusivamente a este campo.

Recientemente el mercado de los videojuegos se ha abierto a pequeños estudios e incluso a desarrolladores en solitario, lo que resulta en una necesidad de automatización del trabajo menos relevante para evitar una pérdida significativa de calidad. Con el objetivo de poder crear aplicaciones de gran tamaño y de calidad se ha potenciado en gran medida la generación automática de contenido gráfico, cuyo primer uso se atribuye al videojuego Akalabeth (Richard Garriott, 1980).

La rama concreta de la generación automática que se plantea en este proyecto es la de generación de entornos naturales, ya que una herramienta bien desarrollada puede permitir la creación de mundos abiertos de tamaño ilimitado, donde se pueda explorar incontables horas y descubrir nuevos paisajes en cada sesión.

Este tipo de tecnología es muy utilizada en simuladores en los que se desean nuevos escenarios y distintas experiencias para cada sesión y puede ser utilizada tanto para servicios de entretenimiento como para fines profesionales o estéticos. Sus usos más directos van desde la amenización de entrenamientos en gimnasios, instalándolos en pantallas para cintas de correr o bicicletas estáticas, donde podría recorrerse el entorno sin llegar a cansarse de él, gracias a su variedad y a la facilidad de crear distintas sesiones, hasta la simulación de ventanas en habitaciones interiores, pudiendo simular distintos tipos de paisaje según las preferencias del usuario. Un uso más complejo y quizá más interesante sería el de, añadiendo ciertas reglas físicas, su uso como simulador para entrenamientos militares como el desarrollado por Smelik (Smelik, Tutenel, Jan de Kraker, & Bidarra, 2010) o en labores de control y extinción de incendios.

Objetivos

El objetivo que se propone en este proyecto es la creación de un generador automático de entornos naturales que sea completo en cuanto a elementos (agua, plantas, montañas…), variable en tiempo de ejecución y sobre todo de aspecto natural. Es imprescindible que la navegación por este entorno se pueda hacer de manera fluida y con un alto nivel de fotogramas por segundo.

Para lograr el objetivo principal se ha dividido el problema en varios subobjetivos:

1. Un módulo de entrada de datos que controle la navegación del usuario por el entorno.
2. Un avatar que represente al usuario y esté integrado en la escena.
3. Un terreno adaptable a las necesidades de cada sesión y de aspecto natural que se pueda adecuar a distintos tipos de entorno.

4. Una capa de vegetación con aspecto natural y frondoso.
5. Una capa de agua que sea capaz de reflejar el mundo que la rodea.
6. Un conjunto de texturas que formen un cielo y un horizonte que se integre correctamente en la aplicación.

Tras la resolución de estos objetivos se pretende unir todas las soluciones para crear escenas distintas y modificables en ejecución que tengan un aspecto realista y variado, para que el usuario pueda explorarlas y modificar sus características indefinidamente.

El producto resultante de este trabajo debe ser multiplataforma, es decir, debe poder ejecutarse igualmente en un ordenador, un móvil o una tablet, por eso se ha decidido implementarlo para que funcione en un navegador, dando así la opción de usarlo en cualquier plataforma que pueda utilizar un navegador moderno.

Estructura de la memoria

Este documento consta de las siguientes 6 partes:

1. Introducción:

   En el primer capítulo se explica de manera concisa la motivación que ha llevado a elaborar este trabajo, el objetivo que se desea cumplir con él y la estructura del documento.

2. Antecedentes:

   En el segundo capítulo se resume el problema que se va a tratar, se trata el estado actual de la generación automática de contenido gráfico y la utilidad de estas soluciones.

3. Diseño e implementación de la solución:

   El tercer capítulo es la parte principal de la memoria, en él se explica cómo se ha decidido proceder y por qué motivos, además se detallan los algoritmos y estructuras de datos que se han utilizado para construir el generador y el producto final obtenido.

4. Pruebas y resultados:

   El cuarto capítulo trata sobre las pruebas a las que se ha sometido el generador y los resultados obtenidos en ellas, evaluándolos de manera objetiva.

5. Conclusiones:

   En este último capítulo se resume brevemente el trabajo y se valoraran los objetivos alcanzados, las líneas de investigación que quedan abiertas para ampliar el trabajo y sus posibles aplicaciones.

6. Apéndices:

   Finalmente se incluyen imágenes de los resultados y enlaces a vídeos y páginas web donde probar el producto, además de la licencia MIT correspondiente a Three.js.

Antecedentes

Descripción del problema

El problema principal al que se hace frente en este trabajo es el de la automatización de la generación de entornos, este problema se debe abordar de manera que las soluciones aportadas sean fiables pero no requieran demasiado trabajo para cada sesión. Es decir, deben dar resultados realistas o de buen aspecto de manera automática, pero se deben poder controlar en cierta medida, ya que se desea que el entorno se adapte a la aplicación concreta que se integrará en él.

Otro problema importante es que se deben poder cambiar fácilmente las soluciones, para poder generar distintos entornos en un periodo corto de tiempo. También debe tenerse en cuenta que el contenido generado debe tener un aspecto variado, que evite la sensación de repetición, ya que esto daría un aspecto poco trabajado al entorno.

Si se resuelven estos problemas se pueden conseguir una amplia variedad de entornos de tamaño casi ilimitado y de aspecto fácilmente controlable, por lo que se podrían ahorra horas de preparación y diseño y se podrían ampliar, incluso eliminar, los límites de exploración de las aplicaciones interactivas.

Estado del arte

Para abordar este problema se ha revisado el estado actual de la generación automática de entornos, centrándose en los elementos utilizados en la creación del generador, aunque no de manera exclusiva.

Los estudios actuales están enfocados mayormente en la creación de terrenos, utilizando para esto diferentes técnicas, la más utilizada es la de mapa de alturas, sin duda la más eficiente, aunque para lograr ciertos resultados, como el poder destruir o construir terreno durante la ejecución de la aplicación, se utiliza la tecnología de vóxels y, para una mayor calidad de reflejos y sombras se usa la de trazado de rayos mediante los conocidos como pixel shaders.

Los elementos naturales más importantes a la hora de crear entorno virtual son la vegetación y el agua, los no naturales son las carreteras y caminos y las ciudades y edificios, que también serán comentados a continuación. Con todo esto se puede crear un entorno virtual muy completo en el que desarrollar aplicaciones gráficas de calidad.

A continuación se comentan los elementos más importantes de la generación de entorno gráfico.

Mapa de alturas

Los mapas de alturas son rejillas bidimensionales con valores correspondientes a la altura en cada posición, esta es la base más habitual de la que se parte al generar terrenos. Los algoritmos más usados para generarlos de manera automática son los de subdivisión y los de ruido fractal.

Los algoritmos de subdivisión son más antiguos y se basan en la división iterativa de un mapa de alturas añadiendo detalle de una manera aleatoria y controlada. El algoritmo de subdivisión más conocido es el de desplazamiento del punto medio, este se basa en la división del mapa de alturas en triángulos y elevación del punto medio de cada uno, desplazando después el resto de puntos en función de los de su alrededor, repitiéndolo iterativamente. En cada iteración se reduce la elevación del punto medio, controlando así la rugosidad del mapa de alturas.

Actualmente los mapas de alturas se generan mediante algoritmos de ruido fractal como el algoritmo Perlin noise (Perlin, 1985), que genera ruido mediante el muestreo e interpolación de puntos en una malla de vectores generados aleatoriamente. Reescalando y añadiendo distintos niveles de ruido, se obtienen mapas de alturas muy realistas (figura 1).

Tras generar un mapa de alturas, este puede ser modificado con algoritmos de filtrado como un suavizado o la simulación de fenómenos físicos, como la erosión. La erosión térmica, por ejemplo, suaviza los cambios de altura mayores que el ángulo de estabilidad, otros algoritmos, como la erosión fluvial, se tratan con métodos mucho más complejos. Para tratar de una manera más fiable con estos algoritmos, se han propuesto otros modelos como el de Benes y Forsbach (Benes & Forsbach, 2001), que en vez de mapas de alturas utiliza láminas horizontales con información sobre el material y la altura, es un modelo mucho más completo y complejo, aunque da la posibilidad de incluir estructuras como cuevas.

Aunque los algoritmos de erosión dan un aspecto más natural a los terrenos montañosos son muy lentos, por lo que actualmente se está tratando de pasarlos a la GPU.

Como los mapas de alturas básicos basados en ruido, presentan un aspecto muy aleatorio, Stachniak y Stuerzlinger (Stachniak & Stuerzlinger, 2005) propusieron un método basado en la aplicación de ciertas deformaciones en función de unos parámetros definidos por el usuario. Otra opción muy utilizada y con grandes resultados es la consistente en la generación del mapa de alturas y posterior editado por el usuario, la cual no resulta adecuada a este caso, ya que requiere de un usuario que diseñe los mapas.

Los mapas de alturas presentan una limitación, no pueden representar cuevas ni rocas salientes, por ello algunos autores como Gamito y Musgrave (Gamito & Musgrave, 2001) utilizan modelos de deformación de terreno, aunque con resultados algo regulares y artificiales.

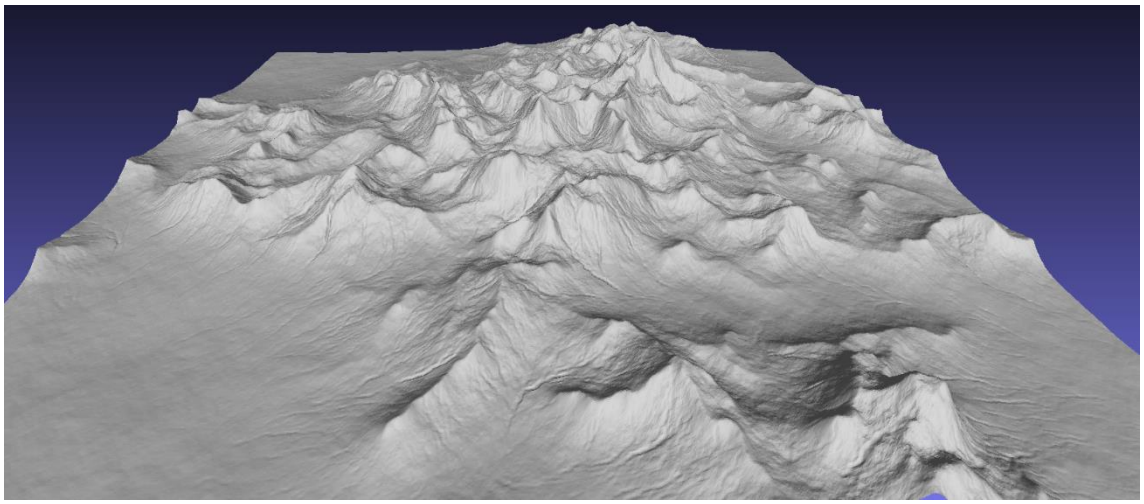

Figura 1. Elevación de mapa de alturas generado mediante Perlin Noise.

Agua: Ríos, Océanos y lagos

La generación de ríos se puede realizar durante la fase del mapa de alturas o tras ella, incluso se ha usado como base del mapa de alturas, generando una red de meandros que forman el esqueleto del mapa de alturas y después rellenándolo mediante interpolación y con datos aleatorios.

Una aproximación muy recurrida es la de rellenar con agua el área bajo un nivel de altura, consiguiendo un resultado no demasiado natural, pero muy sencillo. Una manera más acertada de abordarlo es la descrita por Belhadj y Audibert (Belhadj & Audibert, 2005), que crea un mapa de alturas montañoso y, a partir de él, los ríos. Para ello coloca partículas en lo alto de algunas montañas y las deja caer basándose en física básica, de manera parecida a la erosión hidráulica. Para ciertos paisajes el resultado es rápido y muy efectivo.

Figura 2. Costa generada mediante un algoritmo de inundación.

A otras masas de agua, como los océanos y lagos se les prestan menos atención, los lagos apenas están considerados y los océanos se forman habitualmente rellenando los mapas con agua hasta cierta altura o mediante algoritmos de inundación a partir de puntos de altura muy baja (figura 2).

Plantas: modelos y distribución

El modelado automático de plantas es importante, ya que permite obtener distintos modelos de la misma especie de manera rápida para después poder poblar, de manera también automática, bosques.

Estos algoritmos comienzan por la raíz y el tronco, añadiendo iterativamente ramas cada vez más pequeñas y acabando por las hojas. Se suelen basar en la reescritura de gramáticas o de grafos. Creando estructuras que representan su geometría en 3D.

Los algoritmos de distribución se basan en modelos de ecosistemas. Deussen et al. (Deussen, y otros, 1998) utilizan un mapa de alturas y otro de agua junto a diversas propiedades de las especies. Con ello, aplicando reglas para la competición por sol y agua se obtiene el paisaje tras varios minutos.

Otra manera de crear los ecosistemas, utilizada por Hammes (Hammes, 2001), utiliza únicamente datos sobre altura, altura relativa, pendiente, dirección de la pendiente y ruido fractal para seleccionar ecosistemas predefinidos. La vegetación del nivel del suelo se genera en ejecución en función del nivel de detalle y del ecosistema. Este ecosistema

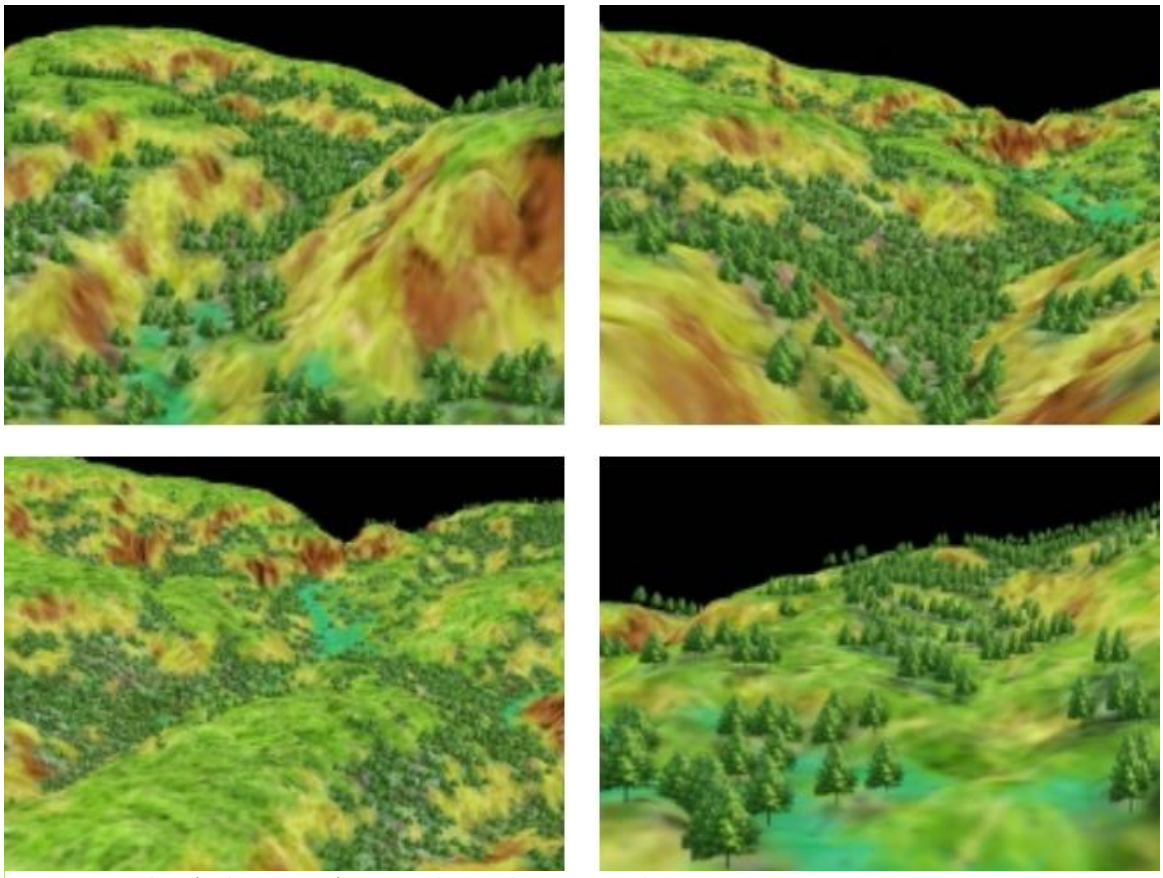

Figura 3. Distribución de vegetación generada con el ecosistema de Hammes.

también especifica el número de plantas de cada especie y las coloca aleatoriamente (figura 3).

Estos métodos se utilizan en juegos modernos, utilizando herramientas como SpeedTree (Interactive Data Visualization Inc, 2000).

Carreteras y caminos

Los algoritmos de generación de caminos se basan en la unión de dos puntos, por ejemplo dos ciudades, tratando de hallar el camino más lógico, es decir, de menor coste.

Para ello se calcula el camino más corto y después, en función de ciertas reglas, se modifica para que no atraviese masas de agua o bosques ni tenga inclinaciones no permitidas.

Peytavie et al. (Peytavie, Galin, Merillou, & Grosjean, 2009) generan el camino de menor coste en función de varios parámetros, como son la longitud del camino, el coste de crear puentes y túneles, la remodelación de bosques y las inclinaciones máximas permitidas (figura 4). Variando estos parámetros se obtienen resultados muy realistas y bien controlados, aunque el coste temporal puede llegar a ser alto.

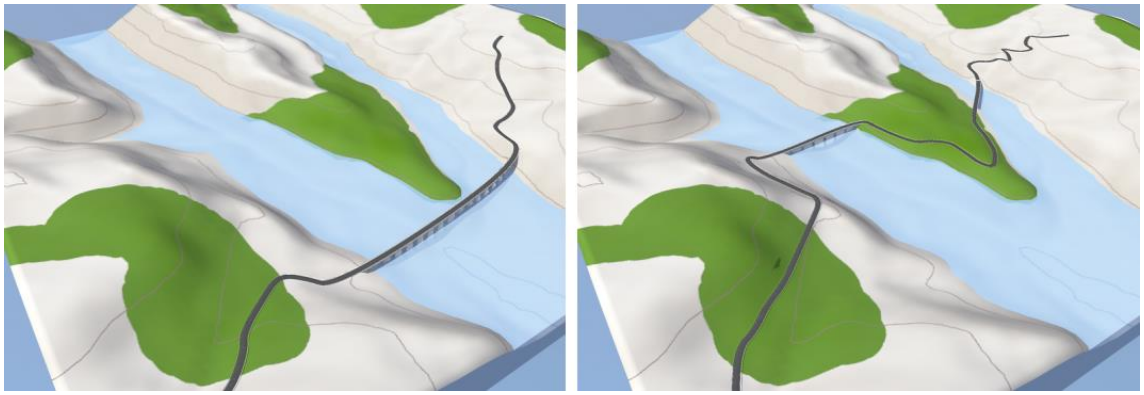

Figura 4. Carreteras generadas por el algoritmo de Galin variando costes.

Ciudades y edificios

La manera más común de generar una ciudad comienza por la creación de un mapa de carreteras, identificando las regiones poligonales entre las calles y subdividiéndolas en manzanas, que luego se rellenarán con edificios, ya sea tomando directamente la forma de la manzana o encajando en esta plantas de edificios prediseñadas o generadas. Para obtener ciudades de rascacielos y oficinas basta con extruir estas formas hacia arriba a distintas alturas (figura 5).

Se han presentado muchas maneras de crear casas con formatos distintos. Müller et al. (Müller, Zeng, Wonka, & Gool, 2007) consiguen modelos de fachadas a partir de imágenes de edificios reales, mediante métodos de reconstrucción. Finkenzeller y Bender (Finkenzeller & Bender, 2008) utilizan un grafo para capturar la información semántica referente a la geometría de un edificio y después reconstruirlo.

Aunque muchos métodos proveen resultados rápidos y visualmente buenos, las ciudades generadas carecen de estructuras realistas, por lo que actualmente es habitual basarse en modelos de ciudades reales, teniendo en cuenta la situación del casco histórico y la atracción del entorno, como ríos, océanos, colinas o depresiones.

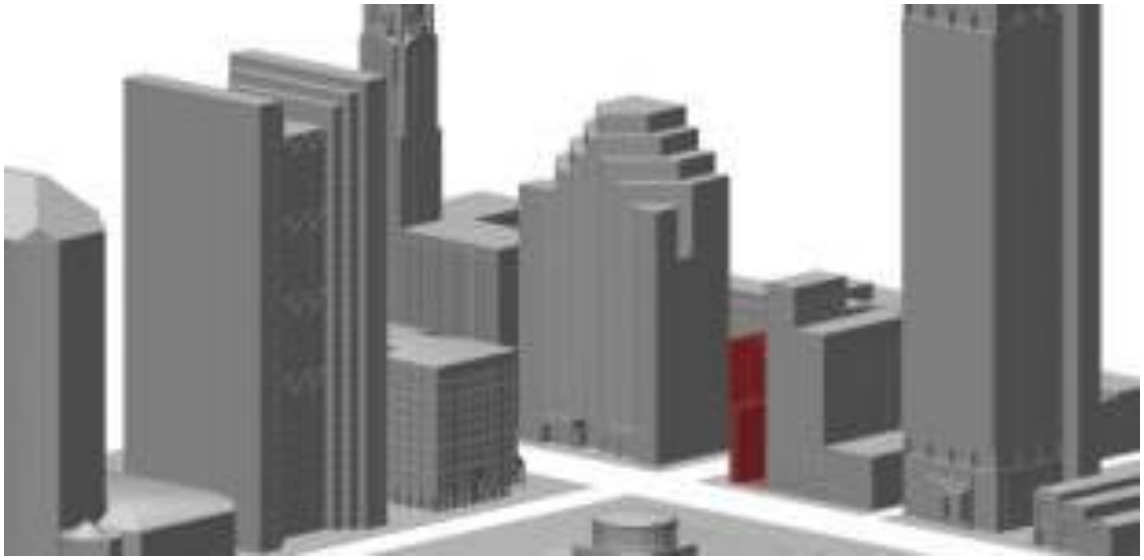

Figura 5. Edificios extruidos por Yong Kyun Roh.

Conclusiones

Las soluciones aportadas hasta ahora carecen de la calidad que puede aportar el trabajo de un equipo de diseñadores, por lo que actualmente se opta por dos opciones, generar el entorno y después modificarlo o crear un esbozo del entorno para que después se complete de manera automática. Estas soluciones tienen ciertos problemas en común:

- Los terrenos generados tienen una amplitud limitada, por lo que solo podrán ser explorados durante un tiempo determinado.
- Si se desea generar un entorno realmente amplio, el peso de los archivos necesarios será demasiado grande, por lo que no se podrá adaptar a dispositivos móviles o navegadores web.
- El tiempo necesario para crear cada parte del terreno es excesivo, por lo que si no se dispone de un gran equipo y mucho tiempo, el terreno será muy limitado.

La solución es usar opciones completamente automáticas, pero estas suelen dar resultados caóticos o tardan mucho en procesarse, como es el caso de la creación de vegetación, ríos, ciudades…

Por ello se ha optado por resolver el problema de manera que el tiempo de carga y preparación del programa sea pequeño, evitando los elementos que lo ralentizan o que requieren de ayuda humana en su creación.

Diseño e implementación de la solución

Análisis

La aplicación que se quiere construir debe tener un aspecto realista y un control sencillo, dando más importancia al tipo de terreno que se construye que a sus características concretas, es decir, se podrá controlar por ejemplo la rugosidad del terreno o la altura de sus montañas, pero no donde se situarán estas. Otro aspecto importante es que debe tener un acceso sencillo a los parámetros que definan el tipo de terreno y, una vez fijados estos parámetros, que se puedan construir distintos terrenos con características similares.

También se debe tener en cuenta que el objetivo de la aplicación será el uso en tiempo real, es decir, que sea interactiva, debe responder al control del usuario y mantener un alto nivel de fotogramas por segundo. Es importante que la carga de un nuevo mapa no sea lenta y la variación de ciertos parámetros se pueda hacer en tiempo real.

La aplicación será preparada para utilizarse en un navegador web, cargándola de internet cada vez que se vaya a usar, por lo que debe ser ligera, evitando un excesivo uso de modelos y texturas de gran tamaño.

Tecnologías y conceptos

Para hacer posible el uso de la aplicación en el navegador, descargándola de internet en cada sesión, se ha creado una página web mediante HTML. Desde esta página, se accede a los distintos ejemplos, que están construidos mediante HTML, JavaScript y las librerías WebGL y Three.js.

WebGL

WebGL es una especificación estándar desarrollada para mostrar gráficos 3D en navegador web, su objetivo es la adaptación de OpenGL ES 2.0 a los navegadores mediante JavaScript, así permite usar gráficos 3D acelerados por software (GPU).

WebGL es soportado por los principales navegadores web con muy buenos resultados, aunque, dado que la última versión estable de OpenGL es la 4.5 y la actual versión de WebGL fue publicada en enero de 2012, está un poco desfasado, aunque siguen desarrollándose avances.

Dado que WebGL está diseñado para el trabajo directo con la GPU, la programación resulta muy compleja. Por ello es habitual utilizar librerías como BabylonJS o Three.js (Cabello, 2010), la que se ha usado en este trabajo por ser sencilla de utilizar, muy completa y porque recibe actualizaciones habitualmente. Esta librería se complementa con el uso de shaders en el lenguaje GLSL, que interpreta la GPU directamente.

El uso de WebGL está permitido sin restricciones de licencia.

Three.js

Three.js es una librería muy ligera, la versión utilizada ocupa 410KB, escrita en JavaScript. Gracias a lo completa que resulta y a su simplicidad de uso, se ha convertido en una de las librerías más utilizadas para trabajar con WebGL.

Three.js resulta una combinación del código escrito por alrededor de 90 colaboradores, aunque siempre revisado y unido por su creador.

Algunas de las grandes virtudes de Three.js son su documentación intensiva y cuidada, su dedicación en los foros, su capacidad de aumento mediante colaboraciones y la facilidad de acceder y modificar su código, que resulta limpio y muy legible.

Las principales características utilizadas en el proyecto de Three.js son:

- Renderizador de WebGL.
- Escena con jerarquía de objetos 3D.
- Cámara perspectiva y ortográfica.
- Luces.
- Materiales prediseñados y personalizados.
- Shaders prediseñados y personalizados.
- Geometría prediseñada y personalizada.
- Cargador de objetos en formato JSON.
- Utilidades matemáticas como matrices y vectores.
- Soporte en forma de API y de ejemplos.
- Herramientas de depuración con manejo de objetos y escenas y contador de FPS.

Por estas características, su fiabilidad y su facilidad de uso, se ha utilizado esta herramienta como base de todo el proyecto, aprovechando que se encuentra bajo licencia MIT (apéndice 4).

Diseño

Para crear el generador automático de entornos se han utilizado algunos modelos de árboles y varias texturas, el resto es generado de manera automática en cada sesión. Este proyecto se ha desarrollado en distintas fases, tras las cuales se ha obtenido un resultado que se ha añadido al generador a modo de capas o subproyectos. Las distintas capas que forman el proyecto son:

1. Módulo de entrada de datos y elementos básicos de una escena.
2. Avatar.
3. Terreno.
4. Vegetación
    a. Hierba y plantas pequeñas.
    b. Árboles.
5. Sombras.
6. Agua.
7. Horizonte.

Módulo de entrada de datos y elementos básicos de una escena

Lo primero que se ha realizado ha sido un módulo de entrada de datos mediante teclado y ratón para permitir al usuario tanto la navegación a través del mundo generado como la modificación de las funciones base del terreno, estos controles aparecen especificados al cargar la página web donde se encuentra el generador y pueden ser consultados en cualquier momento presionando la tecla Esc (figura 6). Este paso ha sido esencial en el desarrollo, pues sin él sería imposible explorar y depurar el resto de elementos.

Tras ello se ha continuado con la creación de los elementos básicos del mundo, como son la cámara y las luces necesarias para iluminar la escena. El control de la cámara se ha cedido al objeto PointerLockControls, que se encarga de modificar la dirección en la que mira la cámara y su posición en función de las señales recibidas por el ratón y el teclado y, más adelante, en función de la posición del avatar y la altura del terreno, para no dejar nunca la cámara bajo el suelo.

Figura 6. Instrucciones y controles del generador de entorno.

En este módulo se ha creado el elemento escena, del que dependen todos los demás elementos que influyen en el dibujado del entorno. También se ha utilizado este módulo para sensibilizar la aplicación respecto al tiempo y así controlar la actualización del resto de módulos y el renderizado de la escena.

Avatar

En el mundo de los videojuegos un avatar es la representación gráfica de un usuario, su objetivo es que este se sienta identificado y representado por él, por ello habitualmente adoptan forma humanoide.

El avatar utilizado en el generador es muy sencillo, ya que solo tiene un papel como intermediario entre la aplicación y el usuario. Consiste en 4 esferas, formando el cuerpo, la nariz y las manos y dos esferas deformadas formando los pies, se ha utilizado el modelo de iluminación de Phong para darle un aspecto integrado en el entorno aplicando las mismas luces que al resto de los objetos de la escena (figura 7).

El modelo ha sido sensibilizado para que cuando el usuario pulse las teclas correspondientes al movimiento, modifique su posición global y la posición relativa de sus pies y manos, dando la sensación de un humanoide caminando. Obteniendo así cierta identificación con el personaje.

Con esto se ha conseguido mejorar notablemente el nivel de inmersión de la aplicación, ya que da una referencia del tamaño del mundo y, cuando el usuario se desplaza, una sensación de movimiento mucho mayor que al utilizar únicamente movimientos de cámara.

Figura 7. Avatar representando al usuario.

Terreno

Al tratar con un terreno ilimitado, no se puede generar todo el mundo a la vez, por lo que la opción escogida para representar el terreno ha sido la de crear una parrilla plana bidimensional sobre la que camina el personaje, esta se mueve con él y se deforma en la tarjeta gráfica de acuerdo a las funciones definidas, tomando la forma que debe tener el terreno. A esta parrilla a partir de ahora se le llamará mapa (figura 8).

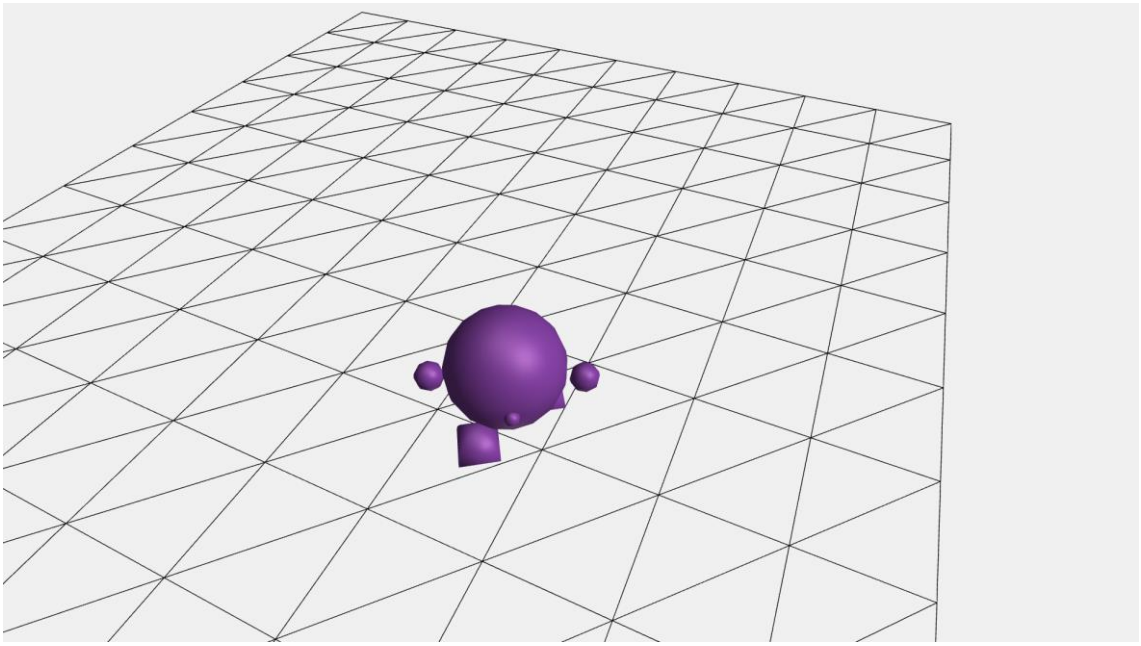

Figura 8. Terreno inicial en forma de parrilla.

La situación de los vértices del mapa, se reajusta cuando la distancia entre el centro del mapa y el usuario es mayor que cierto límite, en este ajuste únicamente se realiza una traslación en los ejes X y Z, de manera que a efectos prácticos únicamente se cambia la matriz que traslada el modelo a su situación en el mundo.

La posición de cada vértice en el eje Y se determina y modifica en la tarjeta gráfica, ya que el coste de modificar la geometría en cada cambio de situación del mapa supone la detención de la aplicación el tiempo suficiente como para dar la sensación de bloqueo de la aplicación, reduciendo en gran medida el número de fotogramas por segundo reproducidos. Esto se debe a la gran capacidad de paralelización que ofrece una GPU, mientras que para modificar la geometría en la CPU, únicamente disponemos de un procesador.

Como se desea tratar con un terreno muy amplio, el número de vértices contenidos en el mapa resulta muy elevado y esto nos obliga a aumentar la distancia entre vértices para poder cubrir suficiente espacio. Esto provoca un nivel de detalle excesivamente bajo en las zonas cercanas a la cámara y excesivamente alto para las zonas lejanas, que apenas se ven, para solucionarlo se ha creado un mapa con un nivel de detalle que se adapta en función de la distancia del vértice al centro del mapa, donde se encuentra la cámara.

Para modificar la altura de los vértices del mapa se han utilizado funciones dependientes del algoritmo de ruido Simplex, que es un avance del algoritmo de Perlin. Una vez combinadas varias octavas de este ruido se consigue una función que simula adecuadamente un paisaje montañoso por el que es prácticamente imposible desplazarse, por lo que se le han aplicado ciertos suavizados dando un aspecto más habitable al paisaje, ya que de otra forma sería prácticamente inútil para este ejemplo.

El color de cada fragmento correspondiente al mapa de alturas se ha calculado mediante una combinación de texturas seleccionada en función de la altura e inclinación de los vértices adyacentes, mezclándolas con cierto ruido, para dar un aspecto más natural.

Nivel de detalle

Dado que la cantidad de vértices que una tarjeta gráfica puede procesar por segundo es limitado y en una aplicación gráfica en tiempo real, como un videojuego, el número de fotogramas por segundo debe mantenerse por encima de 30, se ha tenido que limitar la cantidad de vértices que componen el mapa.

Ya que se limita la cantidad de vértices, se debe adaptar la distancia entre ellos para no perder demasiado detalle en las zonas que más atención recibirán, es decir las más cercanas la cámara. A cambio de esto, se pierde detalle en las zonas más alejadas, aunque se ha conseguido hacer sin perjudicar mucho el resultado.

Esta nueva rejilla consiste en varias zonas distribuidas en función de la distancia al usuario que aumentan la distancia entre sus vértices gradualmente, cada zona está constituida por varias rejillas, del mismo tamaño, con la misma separación entre sus vértices, de manera que se puede cambiar el tamaño de la zona variando el número de rejillas que contiene (figura 9). El nivel de detalle de una zona es inversamente proporcional a la distancia entre sus vértices.

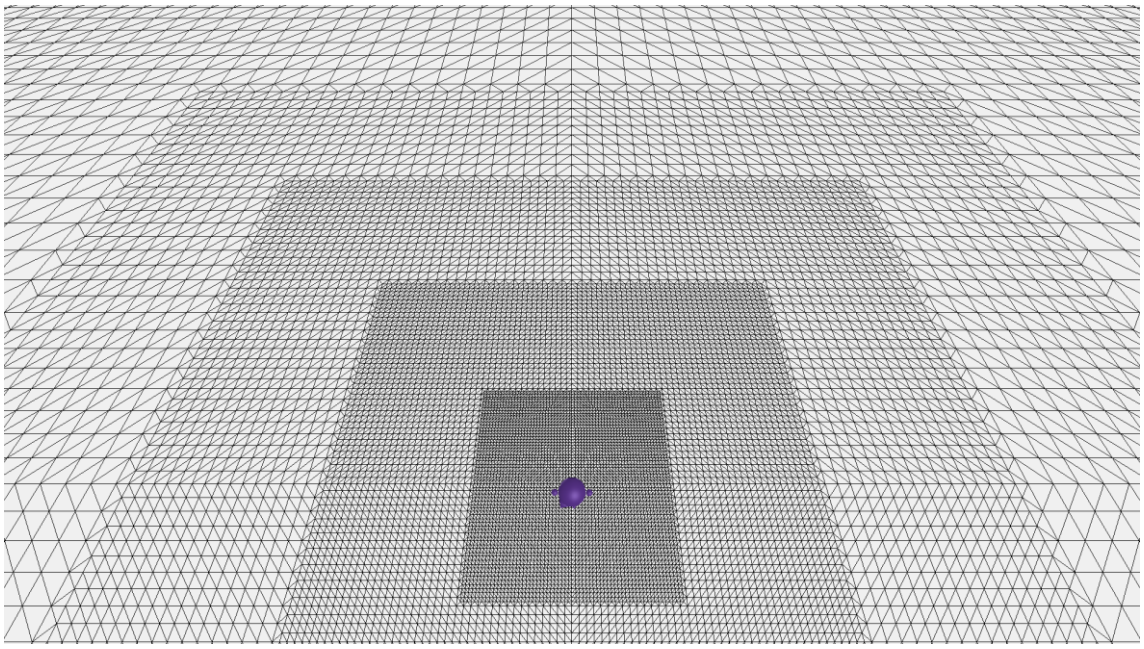

Figura 9. Rejilla con nivel de detalle en función de la distancia.

La unión entre zonas con distinto nivel de detalle crea agujeros en el mallado del terreno, ya que hay una gran cantidad de vértices en el extremo de la zona de más nivel que no coinciden con los de la zona de nivel inferior. Por ello se han tenido que crear zonas de nivel intermedio que disimulen estas diferencias de nivel llamadas costuras, mediante éstas no se aprecia ningún hueco entre zonas y se consigue que la transición entre ellas sea más suave (figura 10).

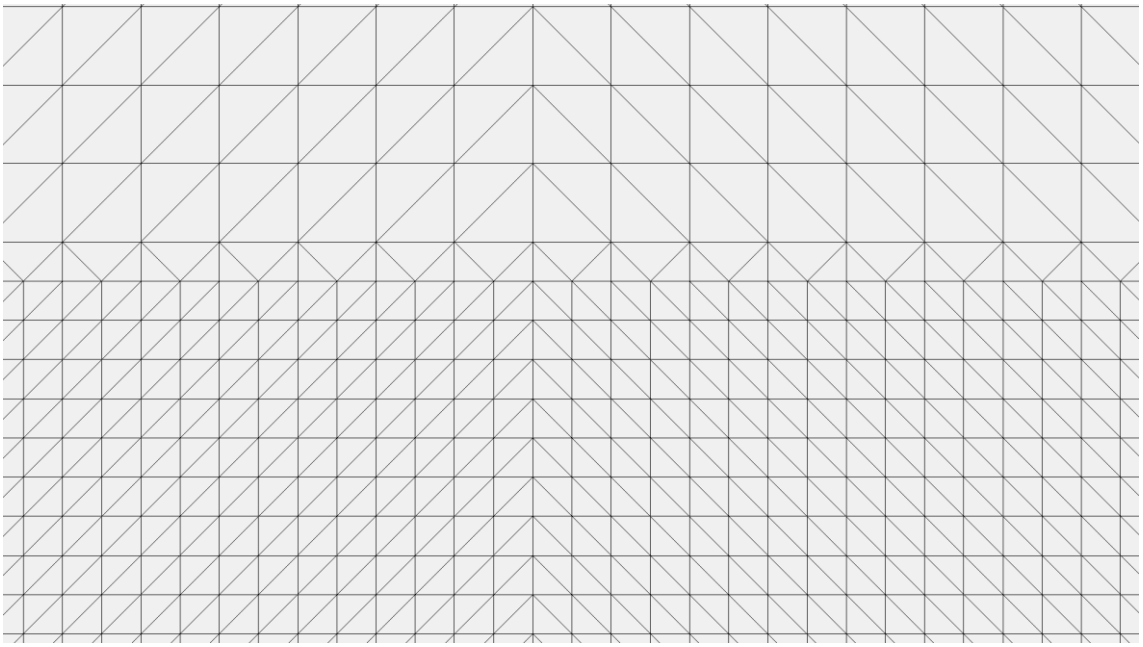

Figura 10. Costura entre zonas con distinto nivel de detalle.

La función que genera la nueva rejilla está parametrizada de manera que se puede elegir el nivel de detalle inicial, el tamaño total de la rejilla, el número de rejillas de la zona inicial y en qué medida aumenta el número de rejillas de cada zona. Este último parámetro es especialmente útil, ya que la diferencia entre una zona y otra es notable, especialmente cuando el nivel de detalle es muy pequeño, por lo que las zonas con bajo nivel de detalle deben ser mucho más amplias que las que tienen un nivel de detalle alto.

Gracias a la adaptación de la rejilla se ha conseguido ampliar el terreno visible en gran medida, manteniendo un número de fotogramas por segundo adecuado y consiguiendo un aumento del nivel de detalle gradual al acercarse a distintas zonas.

Los resultados de la aplicación de un nivel de detalle adaptado a la distancia se pueden ver claramente en la siguiente imagen, donde se compara el mismo terreno, con el mismo número de vértices, con mallado regular y con mallado adaptado a la distancia. La pérdida de detalle en las zonas lejanas apenas se percibe, mientras que la mejora de calidad en las zonas cernas y medias es más que notable (figura 11).

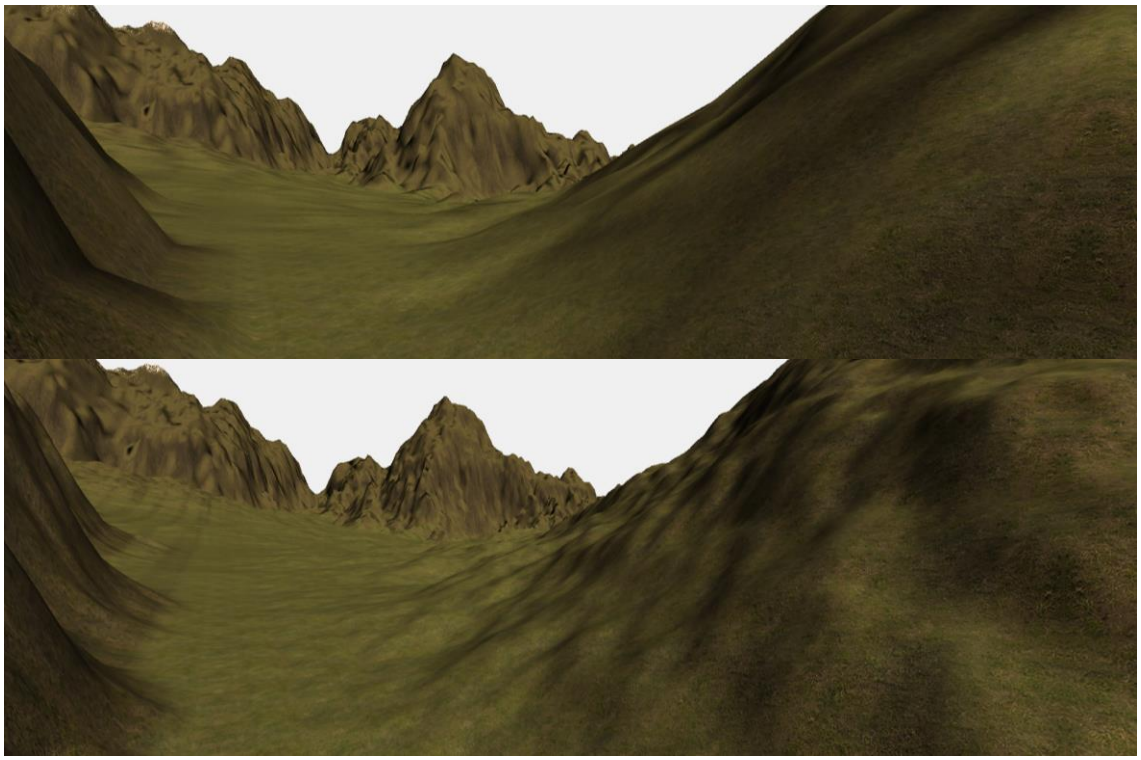

Figura 11. Mismo terreno con 600.000 vértices sin aplicar nivel de detalle (arriba) y aplicándolo (abajo).

Ruido fractal

Para la generación de las elevaciones y depresiones que conforman el mapa se ha utilizado el ruido Simplex, una adaptación del ruido de Perlin, que genera ruido mediante el muestreo e interpolación de puntos en una malla de vectores generados aleatoriamente. El ruido Simplex tiene menor complejidad computacional y su variación de gradiente es más suave. En adelante al ruido Simplex se le llamará únicamente ruido.

Dado que la aplicación directa de ruido genera un espacio demasiado sencillo e irreal se han combinado distintas generaciones de ruido variando sus amplitudes y frecuencias en función de ciertos parámetros ajustables durante la ejecución.

- Octavas: el número de octavas determina la cantidad de generaciones de ruido que se combinan, a mayor número de octavas, mayor rugosidad del terreno (figura 12).
- Amplitud: este parámetro determina el coeficiente por el que se multiplica la primera octava de ruido, variando este coeficiente se consiguen distintas alturas máximas y mínimas para las montañas o depresiones.
- Frecuencia: la frecuencia determina la expansión o contracción de las rugosidades generadas por la primera octava.
- Lacunaridad: la lacunaridad es el factor por el que se multiplica la frecuencia tras cada octava, cuando mayor sea, más rápido disminuirá la distancia entre rugosidades.
- Persistencia: la persistencia es el factor por el que se multiplica la amplitud tras cada octava, cuanto menor sea, más rápido disminuirán la altura máxima y mínima de las rugosidades.

Quedando, tras la selección de parámetros la siguiente función, que devuelve la altura correspondiente a la posición del vértice tras la que se aplican los filtros deseados, todo ello en el shader de vértices:

```
function simplexHeight (position, period, amplitude, persistence, lacunarity, octaves) {
	float noise = 0.;
	for ( int i = 0 ; i < octaves ; i++ ) {
		noise += simplexNoise( new vec2( position.x * period , position.y * period ) ) * amplitude;
		period *= lacunarity;
		amplitude *= persistence;
	}
	return noise;
}
```

Algoritmo 1. Función de acumulación de ruido.

Simplemente, con la modificación de los parámetros mencionados se puede variar el terreno de manera que el mundo generado sea completamente distinto, si a estos parámetros se une la combinación de distintas texturas se puede modificar el tipo de terreno desde una zona de montañas rocosas hasta una pradera o un desierto con un trabajo ínfimo.

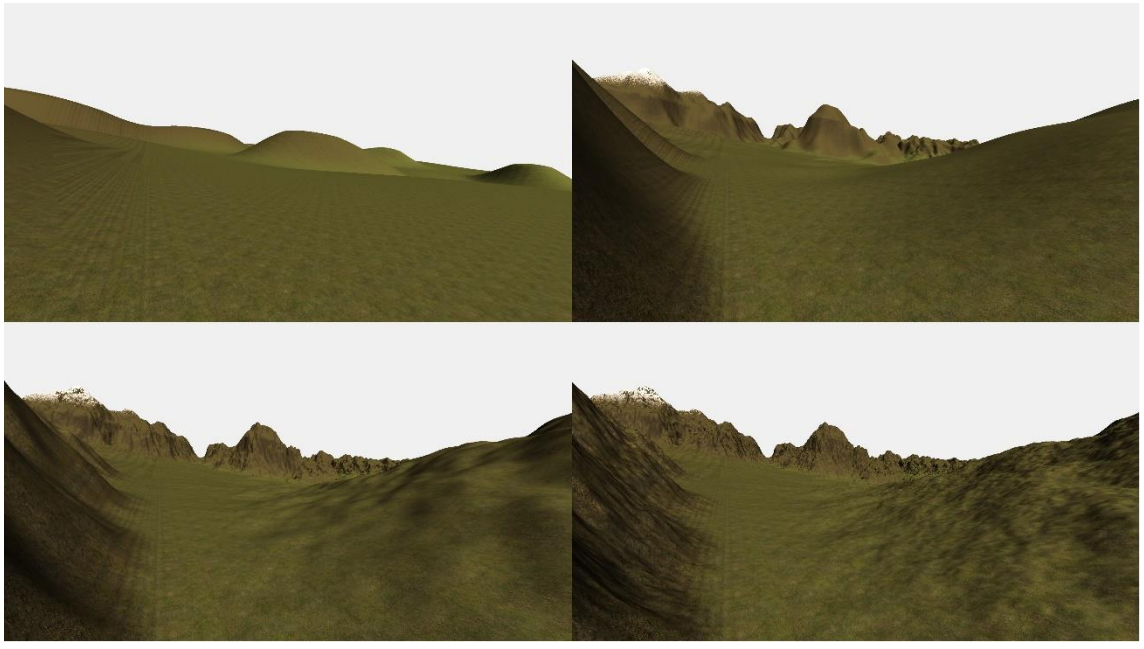

Figura 12. Variación de octavas: una octava (arriba izquierda), cinco octavas (arriba derecha), ocho octavas (abajo izquierda), doce octavas (abajo derecha).

El mayor problema de la generación de ruido mediante este algoritmo es que el terreno conseguido resulta excesivamente rugoso e imposible de recorrer, por lo que es necesario aplicar filtros que permitan un cierto control de los valores recibidos, en el caso de la figura 12, se ha aplicado un filtro que reduce la amplitud de los valores inferiores a cero y suma una cantidad fija a los valores mayores a 100.

Mediante el uso de distintos filtros se pueden conseguir resultados muy variados y curiosos, dando al terreno formas más adecuadas a la función que se le va a dar, como pistas para carreras, superficies con cráteres de aspecto lunar o zonas tácticas para juegos de guerra.

Se ha añadido la posibilidad de crear un filtro a partir de una imagen, logrando así un control muy alto y sencillo para crear mapas con formas concretas como laberintos, pistas de aterrizaje o carreteras (apéndice 2). Este filtro en concreto da la posibilidad de crear mapas que se adapten a las necesidades del usuario de una manera muy rápida y realmente sencilla.

Texturas e iluminación

Una vez asignada su posición a cada vértice, únicamente queda seleccionar el color de los fragmentos, para ello el shader de fragmento utiliza las texturas asignadas y la información que le proporciona el shader de vértices sobre la posición, inclinación y distancia a la cámara de cada fragmento.

El algoritmo de coloreado utilizado para el ejemplo presentado con cuatro texturas (figura 13) es el siguiente:

1. *Se añade ruido a la normal para dar un aspecto más uniforme.*
2. *Se calculan las coordenadas de textura en función de las coordenadas "x" y "z" de la posición.*
3. *Se asigna el color correspondiente a sus coordenadas de la textura 1.*
4. *Si la componente "y" de la normal es menor que cierto valor, se mezcla el color con la textura 2, con las cantidades en función de la componente "y" de la normal.*
5. *Si la componente "y" de la normal es mayor que cierto valor, se mezcla el color con la textura 3, con las cantidades en función de la componente "y" de la normal.*
6. *Si la componente "y" de la posición es mayor que cierto valor (con ruido) y la componente "y" de la normal es mayor que otro valor, se mezcla el color con la textura 4, con las cantidades en función de la componente "y" de la normal y de la altura y se aumenta el valor del reflejo que produce el material, que en otro caso es 0.*
7. *Se aplica al color obtenido el modelo de iluminación de Phong.*
8. *Se mezcla el color obtenido con el color de la niebla en función de la distancia del fragmento a la cámara, obteniendo así el color final.*

Algoritmo 2. Combinación de colores y texturas en el terreno.

Para suavizar el cambio entre texturas se puede añadir un nivel de ruido bajo a cada parámetro, dando un aspecto más realista.

Figura 13. Fragmento de las texturas utilizadas en el shader de fragmento, ordenadas según se usan en el algoritmo especificado anteriormente.

Este algoritmo se puede adaptar fácilmente a las necesidades de cada ocasión, si se desea crear un desierto, una vez cambiados los parámetros del ruido, únicamente haría falta cambiar las texturas del algoritmo por distintas texturas de tierra o, en caso de solo querer una, abreviar el algoritmo quitando las mezclas de color. Así también se puede crear fácilmente una costa o un paisaje nevado (figura 14).

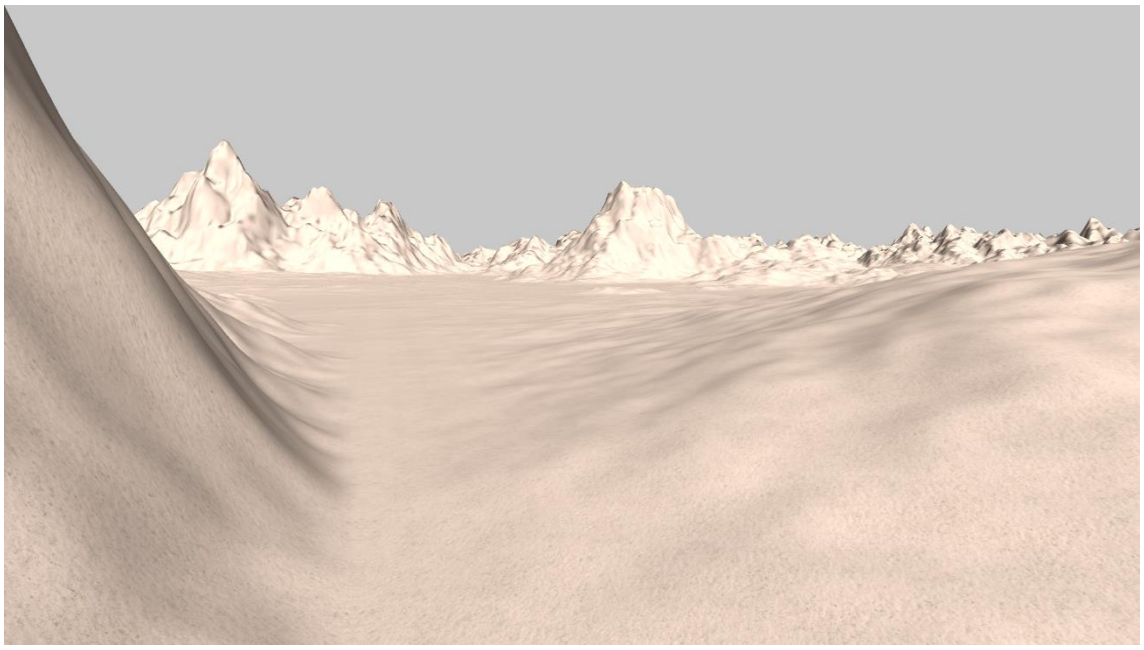

Figura 14. Escena nevada conseguida únicamente mediante el cambio de todas las texturas por una textura nevada.

Vegetación

Una vez construido el terreno sobre el que se desarrollará la acción lo siguiente que se requiere es la vegetación, ya que sin ella únicamente se podrían representar ciertos paisajes desérticos. La vegetación da un aspecto más completo y natural al entorno, además de proporcionar elementos con los que interactuar.

Para representar la vegetación se usan principalmente las técnicas comentadas a continuación:

- Billboards: la técnica del billboard consiste en sustituir el objeto a representar por un plano vertical con una imagen suya (figura 15) y rotar el plano de manera que siempre quede perpendicular al observador. Cuando se trata de objetos pequeños, en vez de rotar un plano, se colocan dos o más de ellos cruzados. Aunque parece una técnica muy chapucera, cuando se trata de una gran cantidad de objetos pequeños, o de objetos grandes muy lejanos, el resultado es tan bueno como el del modelado. La gran ventaja de esta técnica es la pequeña cantidad de vértices necesaria para representar cada objeto.
- Modelado: la técnica de modelado es la más tradicional, consiste en modelar el objeto y pasarlo tal cual a la tarjeta gráfica, el resultado, generalmente es mucho mejor, ya que permite un nivel de detalle mucho mayor, a pesar de ello se evita siempre que es posible, ya que cada modelo puede tener cientos o miles de vértices.

Para crear la vegetación se han considerado dos componentes por separado, hierba y árboles, la primera creada mediante billboards y la segunda mediante modelado. La primera incluye, además de hierba, algunas flores y se podría adaptar fácilmente a arbustos, setas o cualquier planta pequeña. La segunda componente consiste únicamente en árboles, pero también se podrían incluir en ella los arbustos, flores y otras plantas pequeñas a las que se desee añadir más detalle.

Figura 15. Dos de las texturas usadas para representar la hierba (izquierda) y las flores (derecha).

Hierba

La hierba ha sido generada mediante un conjunto de briznas separadas entre sí de manera regular, cada brizna de hierba consiste en dos planos verticales perpendiculares que se cortan en el eje central (figura 16). Todos estos planos se unen en una única geometría, lo que aumenta la velocidad de transmisión a la tarjeta gráfica, donde más adelante se calculará una nueva posición y orientación para cada brizna.

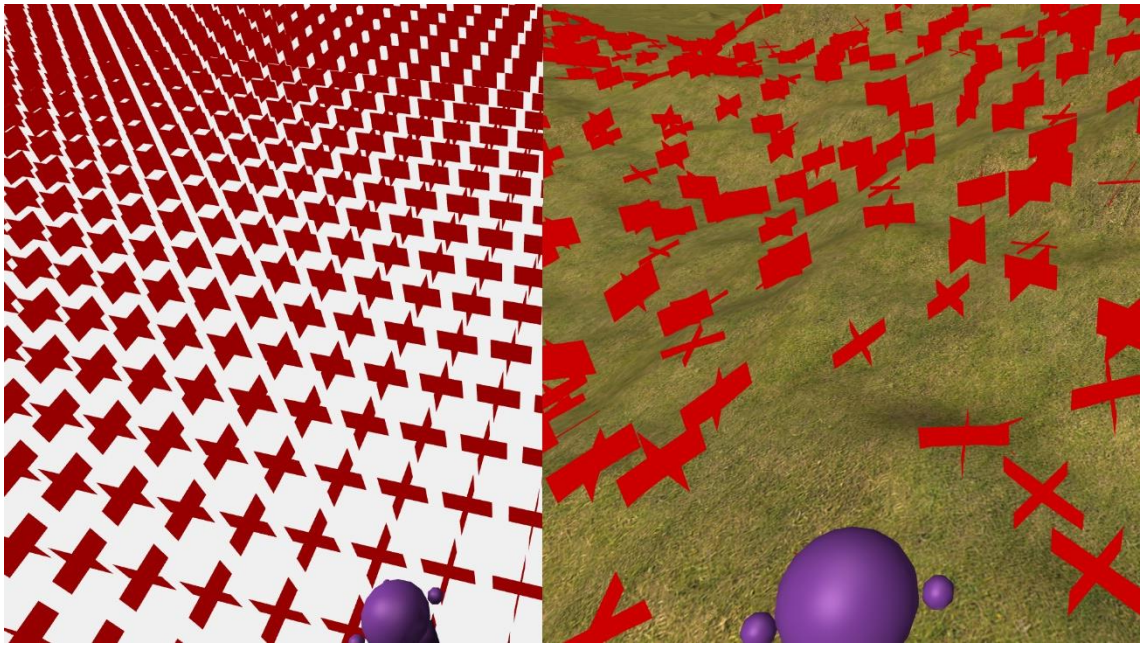

Figura 16. Paneles de hierba antes (izquierda) y después (derecha) de pasar por el shader de vértices.

La malla que compone la hierba ocupa el espacio suficiente como para que no se vean zonas sin hierba y se mueve con el usuario para que todas las zonas estén pobladas, pero como la malla es bastante amplia, y la unión de la hierba con el terreno es bastante disimulada, es muy difícil notar tanto la aparición de nuevas briznas al trasladar la malla como la desaparición de las hierbas que pasan a estar más lejos. Aun así, el hecho de estar compuesta únicamente de planos y formar una sola geometría hace posible que la aplicación siga funcionando con miles de briznas.

Una vez definido el mallado que pasará a la tarjeta gráfica, se realiza el siguiente algoritmo para cada vértice en el shader de vértices:

1. *Se calcula el centro C de la brizna a la que pertenece este vértice.*
2. *Se calculan una serie de ruidos utilizando C como posición del ruido para que sea el mismo para todos los vértices de una brizna.*
3. *Usando uno de estos ruidos se calcula que textura se va a utilizar para la brizna que le corresponde y se notifica al shader de fragmento.*
4. *Se calcula otro ruido distinto en función de C y de a qué plano de la brizna pertenece.*
5. *Se escala la posición en función de uno de los ruidos calculados, consiguiendo así unas briznas más altas y anchas que otras, rompiendo la uniformidad.*
6. *Se traslada el vértice a las coordenadas que le corresponderían si su centro fuera (0,0,0) para rotarlo en función del ruido calculado y se devuelve a su sitio.*
7. *Se traslada el vértice en el plano XZ en función del ruido calculado, quedando el centro en la posición C2.*
8. *Se calcula la normal del terreno en C2 y si su componente "y" no es suficientemente alta se descarta en el shader de fragmento.*
9. *Se suma a la componente "y" de su posición la altura correspondiente al terreno en sus coordenadas.*

Algoritmo 3. Cálculo de la posición y descarte de las briznas de hierba.

El algoritmo que determina el color de cada fragmente es sencillo, se asigna el color correspondiente a sus coordenadas de textura y a la textura seleccionada en el shader de vértices y se combina su color con el que corresponde al terreno en esa posición en función de la altura, para que la zona más alta se vea de su color y la más baja se funda con el color del suelo. De esta manera se consigue evitar la sensación de que se han pegado las plantas al suelo y parece que realmente salen de él, también se consigue así disimular el hecho de que realmente solo hay planos con una textura dibujada. Una vez calculado el color del fragmento se aplica el modelo de iluminación de Phong usando

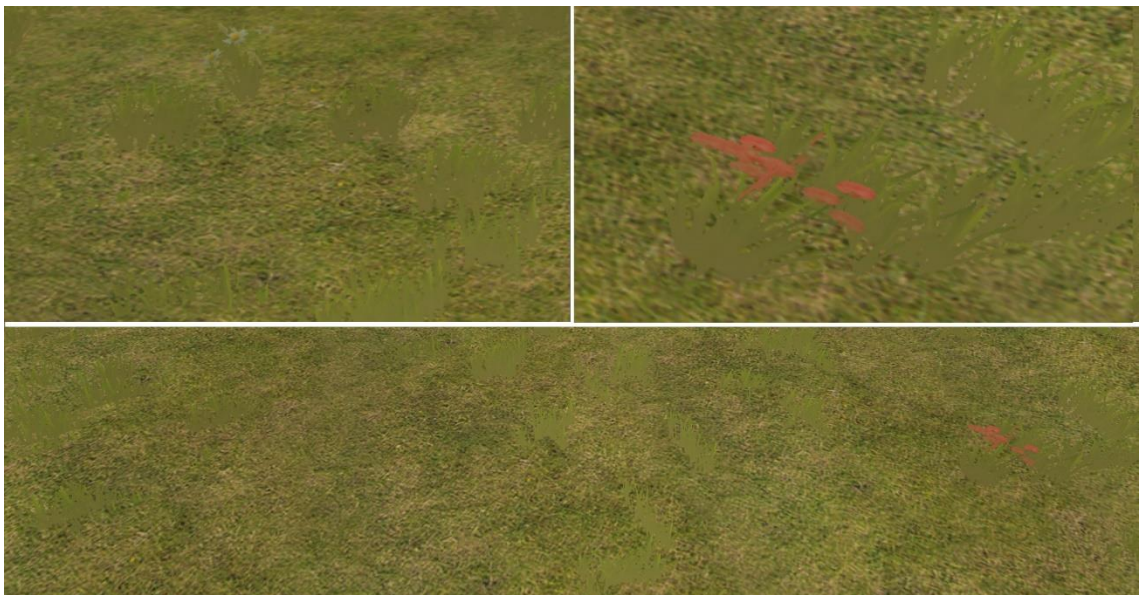

Figura 17. Resultado del terreno tras añadir la hierba (arriba izquierda), las flores (arriba derecha) y vista panorámica de éstas (abajo).

como normal el vector (0,1,0) afectado ligeramente por un ruido, para que no quede demasiado uniforme.

El resultado es el de una hierba frondosa y bien disimulada con el suelo y en la que resulta difícil ver que se usa la técnica de billboard (figura 17).

Árboles

Para crear los árboles se han utilizado varios modelos, cada uno con alrededor de 10.000 vértices, lo que significa que no se puede cargar una cantidad muy grande de ellos, para resolver esto se han utilizado únicamente ocho modelos, los cuales se han escalado, mezclado y rotado, de esta forma se logra disimular su repetición.

Dado que un árbol tiene una cantidad muy grande de hojas y esto no podría representarse fácilmente, ni ser procesado suficientemente rápido, se han utilizado planos doblados con texturas de una manera parecida a la técnica de billboard usada en el dibujado de las hierbas (figura 18). Esto da la posibilidad de manejar un modelo con miles de hojas de manera realista y sin tener que generar demasiados vértices.

Para la simulación de la rugosidad del tronco y las ramas se han utilizado, aparte de las texturas con imágenes de corteza, texturas de modificación de normales, con las que se ha variado la iluminación levemente, dando un aspecto de relieve sin tener que aumentar drásticamente el número de polígonos dibujado (figura 18).

Figura 18. Árbol antes (izquierda) y después (derecha) de aplicar las texturas (centro).

Una vez preparados los distintos modelos de árboles se ha procedido a su colocación. Para ello se ha creado una rejilla regular, de la que se puede variar su amplitud y densidad durante la ejecución. Para que esta rejilla no se quede centrada en una zona, se desplaza con el usuario según ciertos parámetros de manera que los árboles que permanecen en ella no se muevan de su sitio pero los antiguos desaparezcan y se creen nuevos, al igual que se hace con la malla de hierba.

Tras cada movimiento de rejilla se ejecuta el siguiente algoritmo que regula dónde y cómo se coloca cada árbol, además de controlar el descarte de ciertos árboles y qué modelo se utiliza:

1. *Se calculan dos vectores de dos coordenadas de ruido con distintos parámetros.*
2. *Utilizando uno de ellos se calcula que modelo de árbol se usará y el escalado que se le aplicará.*
3. *Se calcula la nueva posición del árbol en función del lugar donde se centra la malla, su posición local en la malla y el otro vector de ruido.*
4. *Se calcula la rotación del árbol combinando los dos vectores de ruido.*
5. *Se calcula la normal correspondiente al terreno en la posición del árbol, si su componente "y" es mayor que cierto parámetro se continua, si es menor se descarta el árbol.*
6. *Se calcula la altura correspondiente a la posición del árbol y si no es excesiva, se crea el objeto con un puntero a la geometría del modelo y se transforma de acuerdo a los parámetros calculados durante el algoritmo.*

Algoritmo 4. Cálculo de la posición y descarte de los árboles.

Con este algoritmo se consigue un aspecto bastante natural en la colocación de los árboles, aunque lo realmente interesante es ajustar los parámetros en función de la zona, de manera manual o mediante una función de ruido, con ello se conseguiría en un mismo escenario tener zonas boscosas, amplias llanuras o lugares intermedios como el que se puede ver a continuación (figura 19).

Figura 19. Escena resultante tras añadir los árboles.

Sombras

Una vez situada la escena, un paso importante para la integración de los objetos en esta y para dar un mayor realismo a las luces es el dibujado de sombras. El algoritmo más fiable y sencillo para dibujarlas consiste en el trazado de rayos, que se basa en la comprobación de si existe algún objeto entre cada pixel y la fuente de iluminación, en este caso, dado el gran número de objetos a comprobar y al requisito de mantener un elevado número de fotogramas por segundo, este método resulta inviable. Por ello se ha utilizado el algoritmo de shadow mapping, para lo que se han marcado dentro del programa los objetos que deben proyectar sombra y los que deben recibirla, por ejemplo, se ha decidido que la hierba no debe proyectar sombra, ya que da resultados más convincentes, sin embargo, todos los objetos la reciben.

El algoritmo de shadow mapping consiste en los dos siguientes pasos:

1. *Se dibuja la escena con los objetos que proyectan sombra situando la cámara en la posición de la fuente de luz, lo que se conoce como coordenadas de la luz. En este caso, como es una luz direccional, se usa una cámara ortográfica. El resultado de este dibujo es una textura que almacena la distancia de cada pixel dibujado a la fuente de luz, es decir el buffer de profundidad.*

2. *Tras ello se dibuja la escena comprobando, tras mover los objetos a las coordenadas de la luz, si cada objeto está más lejos o más cerca que el objeto que fue dibujado en el paso anterior, comparándolo con el buffer de profundidad. Los objetos que estén más cerca o que no reciban sombra se dibujan iluminados, el resto se dibujan sin tener en cuenta la luz que proyecta la sombra.*

Algoritmo 5. Cálculo de la iluminación mediante shadow mapping

Este algoritmo es muy rápido, el único problema es que la calidad de sus resultados está limitada por el tamaño de la textura utilizada como buffer de profundidad, por suerte el tamaño de las texturas que se pueden pasar a una GPU es cada vez mayor, el límite para el equipo usado en la construcción del generador de entornos es de 16.384x16.384, lo que le da capacidad para crear las sombras de un entorno muy amplio con muy buena calidad.

La librería gráfica utilizada facilita el uso de esta técnica, ya que está implementado el primer paso para luces puntuales y direccionales y el segundo para los shaders predefinidos, aunque la adaptación para los shaders creados para el ejemplo es bastante sencilla. El único problema encontrado para esto ha sido el de crear la sombra de las hojas de los árboles, ya que la implementación de esta técnica utiliza un shader básico que no admite texturas para el primer paso del algoritmo, por lo que quedan sombreados planos en vez de hojas. Para solucionarlo se ha creado un material nuevo y se ha sustituido en las hojas para este paso (figura 20).

Figura 20. Comparación entre las sombras creadas con el algoritmo por defecto (izquierda) y las creadas tras cambiar el material de las hojas (derecha).

Una vez creada la escena, la librería en la que se basa en generador permite variar los parámetros de la luz y la sombra de manera que se pueda equilibrar el tamaño del área en sombras y la calidad de éstas. Aunque el área sombreada es grande y desde la posición de la cámara apenas se aprecia que zonas no están sombreadas, se ha ligado la posición de la luz direccional con la posición del usuario para que esta le persiga por la escena y nunca deje de ver la zona en sombras.

A pesar de calibrar adecuadamente el tamaño de la zona en sombras, en determinados momentos se puede apreciar cómo avanza la zona sombreada y la diferencia con el área sin sombrear, la solución obvia es la de ampliar el tamaño de la zona en sombras, pero no se quiere perder tanta calidad en las zonas cercanas, que son las más importantes. Para solucionar esto la herramienta ha incorporado la técnica de sombreado en cascada, ayudando a ampliar la zona en sombras a costa de reducir su calidad en las zonas alejadas, esto no se ha incorporado ya que de momento no funciona perfectamente y da algunos problemas.

Horizonte

La escena resultante de los pasos anteriores resulta bastante completa, pero tiene un fallo muy llamativo, está pintada sobre un lienzo en blanco, quedando recortada contra algo que no se parece en nada al cielo al que estamos acostumbrados. Ya que esto da un aspecto realmente artificial al entorno en el que trabajamos se han llegado a dos soluciones posibles, limitar el campo de visión con niebla y dibujar el fondo del mismo color que la niebla o dibujar un cielo mediante texturas lo suficientemente alejado como para que siempre quede tras los objetos dibujados.

En cuanto a resultados no existe un método mejor que otro, ya que representan situaciones diferentes, el caso de la niebla resulta más limitado, aunque muy apropiado para el caso de escenas nocturnas o con mucha niebla. La utilización de texturas da la

posibilidad de cambiar fácilmente el clima combinando diferentes texturas con cambios de iluminación.

Niebla

Este método resulta el más sencillo, ya que únicamente consiste en calcular en el shader de vértices la distancia del vértice a la cámara y pasarla a una función que calcula el efecto de la niebla de manera exponencial y teniendo en cuenta un parámetro que se le pasa al shader para regularla. El valor del efecto de la niebla se pasa al shader de fragmento, donde se mezcla el color final con el de la niebla en proporción al valor recibido.

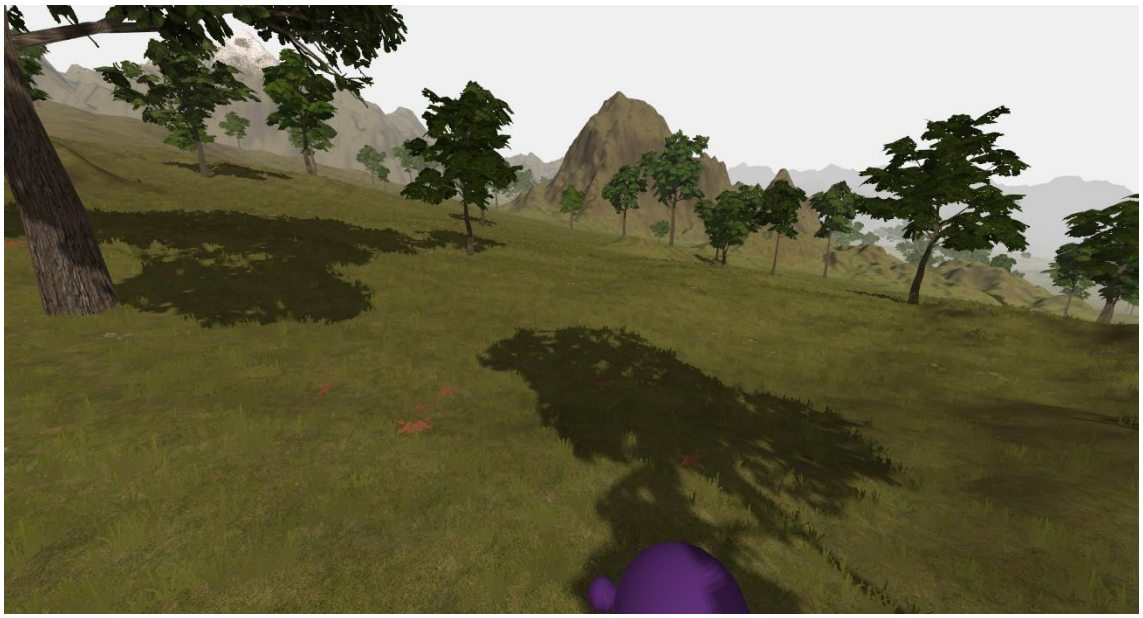

Figura 21. Efecto de la niebla en la escena.

El resultado obtenido (figura 21) es agradable a la vista, ya que difumina la escena logrando disimular el recorte contra el cielo y consiguiendo que los detalles más lejanos sean menos relevantes, centrando la atención del usuario en las zonas en las que mejor se ha dibujado el terreno y en las que se han focalizado nuestros esfuerzos.

Cielo

Para la creación del cielo se ha utilizado la técnica de skybox, que consiste en colocar una caja con texturas pegadas en su interior de manera que formen un cielo sin discontinuidades. Esta caja debe estar suficientemente lejos como para que no tape ningún objeto ni el terreno, pero no tan lejos como para tener que ampliar la distancia máxima de la cámara, por ello se ha hecho de modo que avance con el usuario y así nunca se acerque demasiado.

La mayor ventaja de esta técnica es que el cambio de clima se puede conseguir combinando la posición, intensidad y color de la luz con la modificación de las texturas, incluso se pueden crear las texturas durante la ejecución de manera automática y lograr con ello simular el movimiento del Sol o de las nubes.

Figura 23. Efecto horizonte fundiendo el terreno lejano con el cielo (izquierda), efecto niebla fundiendo todos los objetos con la textura del cielo (derecha).

Aunque el resultado mejora bastante con esta técnica, cuando se logra ver una zona alejada, se ve el terreno recortado contra el cielo, sin lograr el aspecto de un horizonte real. Para ello se ha creado un efecto similar al de la niebla anterior, pero asignando a la componente alfa del color de los fragmentos el valor del efecto de la niebla. Si se regula este efecto para que la niebla afecte únicamente a fragmentos muy lejanos se consigue que se mezcle el terreno con el cielo formando un horizonte realista, mientras que si se eligen las texturas adecuadas para el cielo y se regula la niebla para que aparezca más cerca, aplicando esto a todos los objetos de la escena salvo la caja que simula el cielo, se consiguen buenos efectos de niebla que varían según la dirección en la que se mire (figura 23).El resultado obtenido es bastante bueno como simulación de horizonte y es mejor cuanto mayor sea el terreno y más alejemos este efecto, el efecto niebla consigue fundir todos los objetos gradualmente dando un aspecto no excesivamente realista con algunas texturas, pero si se crean y controlan éstas en la aplicación, se pueden conseguir efectos muy variados e interesantes.

Agua

Cuando se crea el agua de una aplicación interactiva se debe decidir, antes de comenzar con la implementación, de qué manera se situará el agua. Dado el alto nivel de preparación que requiere crear una compleja red de ríos, océanos y lagos, teniendo en cuenta que se desea poder cambiar en cualquier momento la morfología del terreno, únicamente quedan dos opciones:

- Situar el agua por inundación, es decir, seleccionar los puntos de donde parte el agua e inundar los puntos vecinos progresivamente hasta alcanzar el nivel deseado. Este método requiere un tiempo cómputo no demasiado largo tras cada cambio de terreno.
- Fijar un nivel a partir del cual toda la escena estará inundada.

Se ha elegido el segundo método porque no requiere ningún cálculo cuando se cambia la forma del terreno ni cuando se inicia el generador. Además, su mayor ventaja es que puede ser representado por un plano, que se alterará en la tarjeta gráfica dando el aspecto de un lago u océano, dependiendo del terreno que lo rodee. Este plano se moverá en todo momento junto al usuario, para que nunca deje de ver el agua donde deba verla.

Para la simulación de agua se pueden seguir diversos algoritmos en función de la rigurosidad y flexibilidad deseada, en este caso se ha usado uno relativamente sencillo, que representa con bastante fiabilidad la imagen de un lago en el que el agua refleja los demás objetos y el cielo, con únicamente el movimiento que provoca una suave brisa y cierta transparencia.

El objetivo más básico a la hora de generar agua es el de mezclar el color reflejado, el color del agua y el color que se ve a través de ella (figura 24).

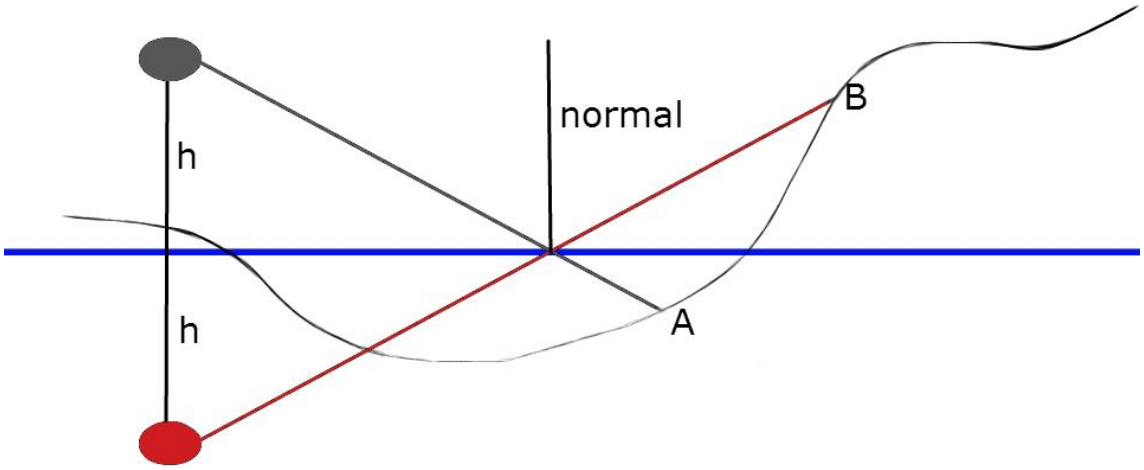

Figura 24. Diagrama de dibujado del agua. Se pueden ver la cámara y el rayo trazado al punto correspondiente del terreno (en gris), la cámara reflejada con el rayo reflejado (en rojo) y el agua (en azul). Y los puntos del terreno de los cuales se mezcla el color (A y B).

Para lograr una representación fiable, sin perder sencillez ni ralentizar la aplicación se ha seguido el siguiente algoritmo:

1. *Inversión de la cámara respecto del plano "y".*
2. *Renderizado de la escena sin el plano del agua en una textura que llamaremos reflejo.*
3. *Renderizado de toda la escena asignando la textura reflejo al material del agua.*

Algoritmo 6. Fases del proceso de renderizado del agua con transparencia, reflejos y olas.

Este proceso resulta muy sencillo, pero la parte importante se encuentra dentro de shader de fragmento del agua, que ejecuta el siguiente algoritmo:

1. *Generación del vector ruido en función de la posición en coordenadas del mundo, una variable que avanza según el tiempo que pasa y una textura en la que se almacenan normales con forma de pequeñas olas.*
2. *Generación de vector distorsión del reflejo en función del ruido y la distancia al observador.*
3. *Asignación del color en función de la textura de reflejo y las coordenadas de pantalla del fragmento distorsionadas mediante el vector distorsión. Esto da el efecto de distorsión del reflejo.*
4. *Mezcla con el color del agua.*
5. *Aplicación de la iluminación de Phong utilizando como normal el vector ruido. Esto da el efecto de iluminación de las olas.*
6. *Asignación del valor de la transparencia del agua para que el renderizador mezcle su color con el de los objetos que tenga detrás.*

Algoritmo 7. Proceso interno de coloreado del agua.

Mediante estos algoritmos se ha conseguido un efecto muy similar al del agua (figura 25), pero sin modificar la posición de los vértices, aunque logrando un buen efecto de pequeñas olas. Esta solución presenta dos pequeños problemas, se ha eliminado el concepto de refracción, por lo que no es del todo realista, aunque no se aprecia a simple vista y se ha aumentado el número de renderizados de uno a dos, es decir, aunque parezca poco, se ha duplicado la necesidad de dibujar la escena. A pesar de ello no se ha aumentado drásticamente el tiempo entre fotogramas, por lo que se puede utilizar.

Figura 25. Masa de agua con reflejos imperfectos y ondas en movimiento (izquierda) y ampliación de la zona donde se encuentra el avatar (derecha).

Los parámetros del agua que se pueden controlar son la distorsión producida por las olas, la velocidad de movimiento de estas, el color base del agua y la transparencia del material.

Escena final

Para presentar el resultado final del generador de entornos se han creado dos escenas con distinto filtro en el mapa de alturas (apéndice 3), el objetivo de éstas es que se puedan explorar y ver los detalles y zonas que se generan de manera automática, destacando que no se ha encontrado, en el tiempo que se ha dedicado a explorar ninguna zona similar a otra vista con anterioridad.

Estas escenas son altamente configurables durante la ejecución mediante los controles comentados en el apartado "módulo de entrada de datos y elementos básicos de una escena".

Los elementos que se pueden encontrar en ambas escenas son:

- Terreno formado por 886.057 vértices.
- Una luz direccional que provoca sombras y otra ambiental.
- Avatar.
- Hierba formada por 62.500 briznas y 500.000 vértices.
- Árboles formados por ocho modelos y 2.916 posibles árboles, de los que la mayoría son descartados antes de ser mandados a la GPU.
- Una cámara perspectiva y una ortogonal para formar las sombras.
- Una caja con texturas de cielo.

En una de ellas, se ha añadido además agua, para que se pueda comparar en las mismas condiciones la ligera ralentización que provoca esta.

Estructura del software

El software se ha dividido en dos partes, el software común a toda escena y el propio de cada una. En el propio de cada una se ejecutan las partes usadas del software común, es decir, controla que no se ejecuten capas no deseadas o añade capas extra. También es donde se define la página web donde se ejecutará el programa. Consiste en el fichero que crea la página web y el fichero en JavaScript que se utiliza para iniciar y ejecutar el programa.

El software común consta de los siguientes ficheros:

- character: funciones y creación del avatar, se encarga entre otras cosas de la animación.
- common: funciones y variables comunes necesarias para cualquier ejemplo.
- grass: funciones de creación de la malla de hierba.
- plane: funciones dedicadas a la creación de la geometría del terreno.
- simplexHeight: funciones dedicadas a la generación de ruido en la CPU, se utiliza para hallar la altura e inclinación del terreno en ciertos puntos.
- skybox: funciones dedicadas a la creación del cielo.
- trees: funciones dedicadas a la creación, actualización y carga de modelos de los árboles.

- water: funciones dedicadas a la creación del agua, a su actualización y a su renderizado.
- Shaders:
    - grass: shader encargado del renderizado de la hierba.
    - leaf: shader encargado del renderizado de las hojas de los árboles.
    - simplexHeight: shader encargado del renderizado del terreno.
    - tree: shader encargado del renderizado del tronco y las ramas de los árboles.
    - water: shader encargado del renderizado del agua una vez obtenida la textura del reflejo.

Con esta división se ha logrado que la creación de nuevas escenas sea sencilla y no tenga dificultad el añadido o eliminado de capas, pudiendo quitar los árboles, la hierba o, como en uno de los ejemplos, añadiendo el agua.

El peso total del programa es de 17,4MB, suponiendo las texturas 2,47MB y los modelos 13,6MB.

Conclusión

Mediante estos pasos se ha conseguido un sistema que genera entornos de aspecto natural de una manera eficiente y con escaso tiempo de carga. El hecho de estar construido como diferentes módulos facilita la integración de módulos nuevos o el descarte de algunos de ellos, pudiendo ser un generador fácilmente ampliable o modificable.

Los resultados han sido bastante buenos para todas las capas aunque no se ha simulado ningún proceso natural en la creación del mapa de alturas ni en la distribución de plantas y agua, ya que esto requiere mucho tiempo de preproceso, cosa que se quiere evitar. Aun así se ha conseguido una simulación adecuada, ya que a nivel local no se nota esta falta, es decir, ningún observador que se encuentre dentro del terreno se dará cuenta de que la distribución de plantas es similar cerca y fuera del agua.

Pruebas y resultados

Para comprobar el funcionamiento del entorno de manera objetiva se ha utilizado un ordenador con las siguientes especificaciones:

- CPU: AMD FX-6300, con 6 núcleos a 3.5GHz.
- GPU: Gigabyte GV-R779OC-1GD, con una memoria de 1GB.
- RAM: 8GB DDR3.
- SO: Windows 7.

Las pruebas realizadas han comprobado el tiempo de carga del programa y el número medio de fotogramas por segundo en las dos escenas presentadas en el apartado "escena final". Tras las pruebas se ha comprobado que la actualización de las posiciones y alturas de los árboles supone una importante ralentización de la aplicación, por lo que se han aislado los datos de estos momentos y analizado por separado. Se han obtenido, tras tres minutos de uso los siguientes resultados en el navegador Google Chrome, versión 43:

Google Chrome		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	40	35
FPS medios resto del tiempo	55	50
Tiempo de carga	38 segundos	40 segundos

Tras el tiempo de carga, la aplicación tarda aproximadamente 10 segundos en comenzar a funcionar de manera fluida, para evitar este problema podría aumentarse el tiempo de carga de manera artificial, evitando que el usuario tenga una experiencia incompleta ese tiempo.

Los resultados presentados son los que se han obtenido utilizando el navegador Google Chrome, para el navegador Mozilla Firefox, versión 38, se ha obtenido un tiempo de carga ligeramente menor y un número de FPS no tan bueno, pero sin llegar a ser un problema.

Mozilla Firefox		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	35	33
FPS medios resto del tiempo	50	47
Tiempo de carga	32 segundos	33 segundos

Para contrastar estos resultados con los de un ordenador no preparado para ejecutar aplicaciones gráficas, se han realizado las mismas pruebas utilizado un ordenador portátil con las siguientes especificaciones utilizando las mismas versiones de los navegadores:

- CPU: Intel Core I3-4005U, con 4 núcleos a 1.7GHz.
- GPU: integrada Intel HD Graphics 4400.
- RAM: 8GB DDR3.
- SO: Windows 8.1.

Google Chrome		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	2	2
FPS medios resto del tiempo	10	8
Tiempo de carga	115 segundos	123 segundos
Mozilla Firefox		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	2	2
FPS medios resto del tiempo	9	6
Tiempo de carga	102 segundos	105 segundos

Como estos resultados no son admisibles se ha probado a adaptar la escena a las posibilidades de este ordenador, variando el número de briznas de hierba y de árboles y el número de vértices del suelo. La reducción del número de brinzas únicamente ha resultado en pequeños cambios en el tiempo de carga y la del número de árboles ha resultado en una mejora no muy grande de los FPS en el momento de la actualización de su posición. Por ello se ha optado por reducir el número de vértices del suelo drásticamente, obteniendo los mejores resultados con 78.905 vértices, frente a los 886.057 que se usaban en las otras escenas. Esto se ha conseguido regulando el nivel de detalle de las zonas cercanas, la velocidad a la que se pierde detalle con la distancia y el tamaño del mapa del suelo. Tras esta adaptación se han conseguido unos resultados óptimos.

Google Chrome		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	30	28
FPS medios resto del tiempo	50	47
Tiempo de carga	63 segundos	64 segundos

Mozilla Firefox		
Escena	**Montañas**	**Agua**
FPS medios en actualización de árboles	27	26
FPS medios resto del tiempo	48	45
Tiempo de carga	55 segundos	58 segundos

Como se puede ver, la comparación entre navegadores da un resultado similar en todos los casos y las diferencias entre la escena sin agua y la escena con agua son muy pequeñas.

Realmente el número de fotogramas por segundo depende más de Three.js y del propio WebGL que de la aplicación en concreto ya que, salvo en la carga y la actualización de los árboles, casi todo el trabajo es llevado por estas librerías. El verdadero objetivo de esta prueba es la demostración de que el entorno generado, a pesar de ser bastante completo y extenso, puede funcionar a buena velocidad en un navegador.

Tras ver estos resultados se puede deducir que en una máquina con mejores prestaciones se podría alcanzar una amplitud del terreno suficientemente grande como para que no se pueda observar el final del entorno que se genera o con mayor nivel de detalle en el terreno.

Los resultados que se han obtenido son el fruto de la adaptación de las escenas a las prestaciones del ordenador donde se han probado, ajustando los elementos de manera que se obtenga el mejor resultado visual sin perder fluidez. Aunque de la manera en que está construido el generador la adaptación a ordenadores de menor o mayor capacidad es muy sencilla, incluso se podría programar para modificarlas en tiempo de ejecución, aunque este proceso ralentizaría mucho el tiempo de carga.

Conclusiones

Objetivos alcanzados

Durante este trabajo se han alcanzado en mayor o menor medida todos los objetivos propuestos, se ha obtenido, como se pretendía, un generador de entornos completo, modificable durante su ejecución, de aspecto natural y con un funcionamiento fluido. Por otro lado, no se ha conseguido que ciertos elementos sean realmente naturales, como la distribución de las plantas o del agua.

Se ha conseguido sin dificultad la creación del módulo de control del usuario y la creación del avatar, ya que estos objetivos son los que menos dificultad presentan.

En cuanto al terreno, se ha conseguido formar una geometría extensa y con buen nivel de detalle. El aspecto que presenta es bastante natural, ya que la iluminación y las texturas aplicadas facilitan esto. Sin embargo, el hecho de no simular ningún fenómeno natural ni basarse en mapas de alturas reales provoca que el nivel de realismo no sea tan alto como se podría haber conseguido de esta manera, aunque esto habría aumentado enormemente el tiempo de preproceso, lo que habría hecho imposible su modificación en tiempo de ejecución.

La capa de vegetación ha sido un éxito en el apartado de la hierba, ya que se ha logrado que sea muy frondosa y con el añadido de más texturas se puede hacer mucho más variada, aunque no resulta necesario. En el apartado de los árboles se ha logrado un buen resultado, aunque el hecho de basarse en modelos lo ha limitado. La capa de árboles ha resultado la parte de la aplicación que más tiempo requiere durante la ejecución, lo que puede suponer un lastre para aplicaciones de mayor tamaño.

El agua formada presenta un aspecto realista y muy integrado en el entorno, apenas ralentizando la aplicación al basarse en un único plano horizontal. Se ha obviado el efecto de la refracción, lo que le resta realismo, pero es apenas perceptible y habría requerido otro proceso de renderizado e integración de texturas en el agua como el del reflejo. No se ha podido hacer una capa de ríos por requerir demasiado tiempo de preproceso, pero, una vez fijados los parámetros del terreno, podría haber sido un elemento muy interesante.

En las texturas que forman el cielo se ha conseguido una buena integración con un aspecto de horizonte bastante acertado, aunque el hecho de usar texturas pregeneradas supone que no se puede cambiar el clima ni la inclinación del Sol. Esto se podría solucionar con una sencilla generación de texturas en tiempo de ejecución.

La unión de estos elementos ha formado las escenas como estaba planeado y se ha conseguido mantener una buena velocidad de carga y ejecución, ya que las escenas se pueden adaptar con facilidad al ordenador donde se ejecutarán.

A pesar de haber hecho la aplicación para funcionar en navegadores web, no se ha conseguido que funcione en todas las plataformas, ya que algunas funciones de Three.js y de WebGL no funcionan en todos los navegadores y solo están disponibles para

funcionar en ordenador. Esto se debe a que la tecnología utilizada aun no da el soporte necesario para ciertos dispositivos como smartphones y tablets, aunque se prevé que esto se solucione pronto. Por lo que se puede concluir que se ha conseguido una buena portabilidad entre ordenadores, ya que no requiere ningún tipo de instalación a parte del navegador donde se ejecuta, aunque la portabilidad a dispositivos móviles queda pendiente de la evolución de esta tecnología.

Por lo tanto, se puede considerar que se han cumplido casi todos los objetivos del proyecto de manera bastante rigurosa, aunque ha quedado espacio para ampliaciones y pequeñas mejoras, especialmente en cuanto a terreno y vegetación.

Líneas abiertas

El campo de la generación de contenido gráfico es muy amplio y completo, utilizando ciertos métodos se puede llegar a generar todo de manera automática, incluidos los modelos y las texturas, teniendo en cuenta esto se pueden proponer varias líneas de trabajo para ampliar el proyecto que se ha realizado. A continuación se comentan las principales.

La ampliación más interesante que se presenta al acabar este proyecto es la variación de los parámetros del terreno, la vegetación y el agua según su situación. Si se decidiesen estos parámetros en función de ciertas funciones de ruido, como el simplex, de manera continua y suave, se podría conseguir un mundo con más variación, pasando de desiertos a oasis y de playas a selvas, sin tener que hacer ningún preproceso ni preparación manual. Esto lograría un resultado mucho más completo y variado.

La implementación de un motor de simulación física puede ampliar enormemente la utilidad de esta herramienta, dando mayor facilidad para la implementación de simuladores, videojuegos o animaciones.

Las texturas utilizadas para el cielo pueden ser formadas de manera procedural y modificadas durante la ejecución para simular el clima y el paso del tiempo, esto no es muy complicado, ya que se puede generar el cielo como un degradado de color que cambie con la posición del Sol y nubes creadas mediante ruido simplex que varíe con el tiempo.

Los árboles y arbustos pueden ser generados de manera procedural en poco tiempo, se puede lograr imitando la herramienta SnappyTree (Brunt, 2012). SnappyTree está escrito en WebGL utilizando la librería GLSE, similar a Three.js, esta herramienta recibe los parámetros de grosor, altura, ramificación, etc. del árbol deseado y lo genera rápidamente, con unos resultados muy buenos.

Otra ampliación muy interesante es el movimiento de las plantas por aire, la generación de viento se puede hacer de manera sencilla sin mucha rigurosidad, ya que al verse únicamente su efecto, no pueden juzgarse fácilmente sus fallos a no ser que sean muy grandes. Una vez generado el viento, únicamente habría que mover los vértices adecuados de las plantas de acuerdo a él. Esto se puede hacer en el shader de vértices sin demasiada complicación.

La generación de caminos y objetos creados por el hombre, como casas o puentes, no es muy complicada y podría dar pie a la formación de carreteras y pueblos sencillos. Esto conseguiría acercar el proyecto a su aplicación en juegos de carreras o de aventuras.

Por último se podría crear un módulo de control de animales que poblasen el terreno, con algunos modelos y ciertos comportamientos básicos, como hacer que huyan del usuario, se puede conseguir un efecto muy completo y animado.

Posibles usos de la herramienta

Este tipo de herramienta puede ser utilizado como base para múltiples productos y modificado rápidamente para servir a distintos fines de las maneras que se han comentado en el apartado anterior.

La utilización más directa de esta herramienta es sencillamente como juego de exploración, ya sea como videojuego o para ofrecer un entretenimiento mientras se hace ejercicio. Añadiendo un sencillo motor de física se puede ampliar como juego de carreras, que se puede incluir en simuladores de conducción. La versatilidad que le da el cambio de terrenos mediante variación de parámetros podría ser un extra para evitar el aburrimiento.

Otra utilización interesante puede como banco de pruebas para algoritmos de construcción de carreteras o emplazamientos como ciudades o pueblos, ya que se pueden generar distintos tipos de terreno en los que probar sus resultados.

Con el añadido de motores físicos más complejos se puede imitar el efecto del viento y el terreno en la expansión de incendios, utilizándolo como simulador para el entrenamiento de equipos de extinción de incendios.

Otra opción que se puede estudiar es la el añadido de un sencillo sistema para el modelado del terreno y el emplazamiento de plantas u obstáculos, pudiendo así preparar fácilmente entornos en los que se desarrollen juegos estrategia o de guerra, en el sentido de videojuegos de disparos en primera persona. Este uso es muy común en la actualidad, por lo que sería una salida muy interesante.

También puede usarse como simulador de ventanas en habitaciones interiores, pudiendo así dar un aspecto más abierto a estas, una ventaja clara de usar una herramienta como esta, en vez de imágenes fijas o vídeos, es que se puede modificar el ambiente seleccionado, incluso crear uno nuevo, en cuestión de minutos. Si se aplica la ampliación de creación de texturas para el cielo, se puede simular el tiempo del lugar donde se encuentra la habitación, el de otro sitio o cualquier clima a petición del usuario.

Por último, una aplicación directa, sencilla e interesante por su utilidad comercial es el uso del entorno como fondo para animaciones, ya que con distintos filtros en el cálculo del mapa de alturas se pueden conseguir muchos tipos de terreno, incluso terrenos inverosímiles para aplicaciones creativas, como la simulación de otro universo con distintas leyes físicas o planetas con distinta fuerza de gravedad.

Bibliografía

Belhadj, F. a. (2005). Modeling Landscapes with Ridges and Rivers: Bottom Up. *GRAPHITE '05*, 447–450.

Benes, B. a. (2001). Layered Data Representation for Visual Simulation of Terrain. *Proceedings of the 17th Spring Conference on Computer Graphics*, 80–86.

Brunt, P. (2012). SnappyTree.

Cabello, R. (2010). Three.js.

Deussen, O. H. (1998). Realistic Modeling and Rendering of Plant Ecosystems. *SIGGRAPH '98*, 275–286.

Finkenzeller, D. a. (2008). Semantic Representation of Complex Building Structures. *Computer Graphics and Visualization*.

Gamito, M. a. (2001). Procedural Landscapes with Overhangs. *10th Portuguese Computer Graphics Meeting*, 33–42.

Hammes, J. (2001). Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering. *DEM '01: Proceedings of the First International Symposium on Digital Earth Moving*, 98–111.

Interactive Data Visualization Inc. (2000). SpeedTree.

K., P. (1985). An Image Synthesizer. *Computer Graphics*, 287-296.

Müller, P. Z. (2007). Image-based Procedural Modeling of Facades. *SIGGRAPH '07*.

Peytavie, A. G. (2009). Arches: a Framework for Modeling. *Eurographics 2009 Proceedings*.

Richard Garriott. (1980). Akalabeth.

Ruben M. Smelik, T. T. (2010). Declarative Terrain Modeling for Military Training Games. *International Journal of Computer Games Technology*.

Stachniak, S. a. (2005). An Algorithm for Automated Fractal Terrain Deformation. *Computer Graphics and Artificial Intelligence*, 1: 64–76.

Apéndices.

Apéndice 1. Imágenes

Con el cambio de las texturas de suelo por una textura de cráteres y la del cielo por una textura de cielo nocturno, la adaptación de la luz y la eliminación de plantas, junto a un filtro que invierte el terreno a partir de cierta altura para simular cráteres, se consigue un efecto parecido al del suelo lunar en cuestión de pocos minutos (figura 26).

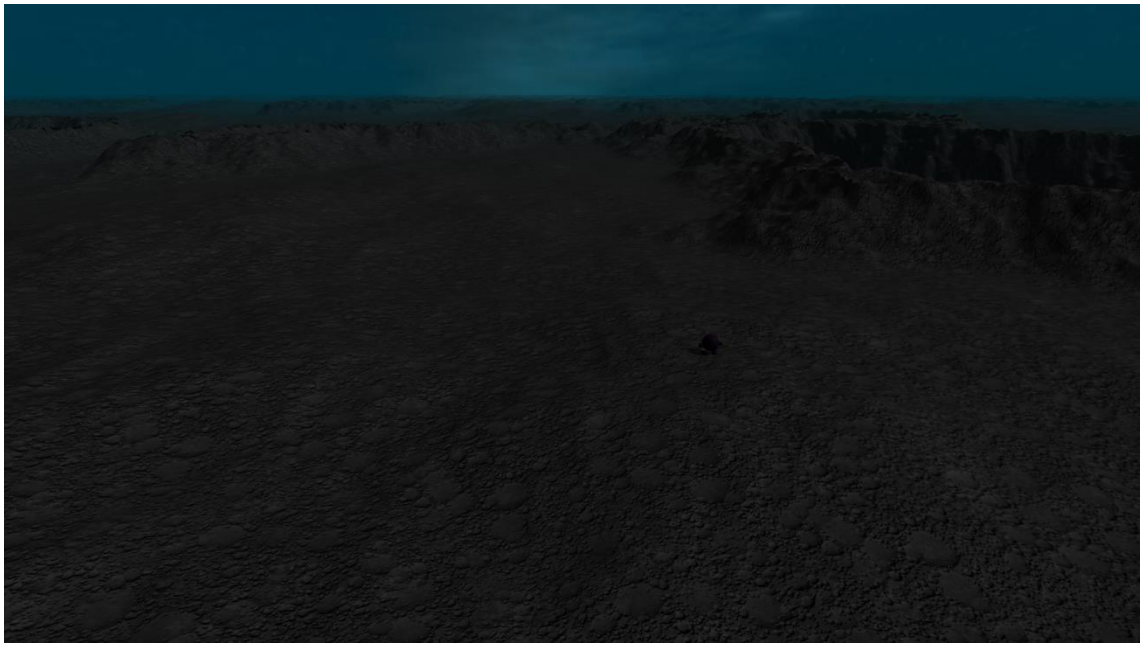

Figura 26. Superficie lunar.

Definiendo en una textura dos regiones (blanco y negro) se puede adaptar la función que genera el ruido para que en cada región utilice distintos parámetros, consiguiendo una adaptación fácil a nuestras necesidades (figura 27).

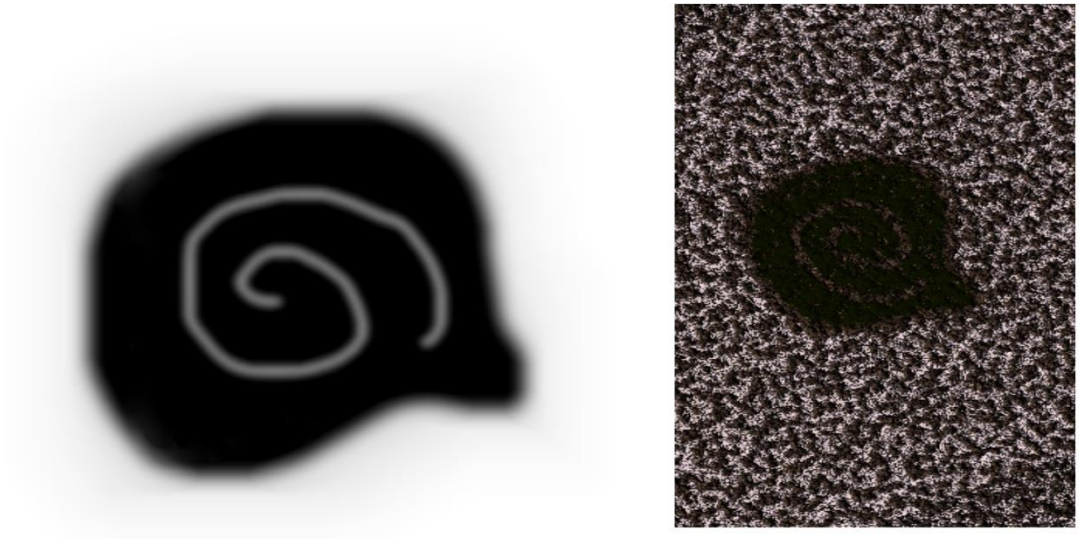

Figura 27. Regiones definidas en una textura (izquierda) y su resultado (derecha) en vista zenital.

Combinando la distorsión del reflejo, la transparencia y el color del plano que define el agua se consigue un efecto muy parecido al de un lago con poco viento en el que se reflejan los árboles, el terreno y las nubes (figura 28).

Figura 28. Reflejo de un árbol.

Apéndice 2. Vídeos y pruebas

Para comprobar el funcionamiento de la herramienta se puede acceder a http://personales.alumno.upv.es/edvilval/ donde están las escenas antes comentadas para probar y tres videos de ellas, para poder ver la herramienta en acción si el equipo con el que se está viendo el trabajo no soporta la herramienta o requiere escenas de menor peso.

Apéndice 3. Licencia MIT

Copyright (c) <año> <titular del copyright>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.