Máster en Computación Gráfica, Realidad Virtual y Simulación

Trabajo Fin de Máster

Generación de terreno Procedural, Biomas y Ciudades para Realidad Virtual en Unity

Autor: Rafael Polope Contreras

Tutor: Álvaro San Juan Cervera

Junio 2024

Resumen:

Este trabajo presenta una propuesta para la generación de terreno infinito de manera procedural incluyendo generación de biomas y poblaciones para el motor Unity en tiempo real, además permitiendo la compatibilidad con la realidad virtual haciendo uso del sistema DOTS de Unity junto a otras varias técnicas de optimización.

La generación del terreno estará parametrizada para permitir control al usuario.

Palabras Clave: Generación Procedural, Biomas, Poblaciones, Realidad Virtual, Job System, Optimización, Parametrización.

Abstract:

This work presents a proposal for the procedural generation of infinite terrain, including biome and population generation, for the Unity engine in real-time. Additionally, it enables compatibility with virtual reality by leveraging Unity's DOTS system along with various optimization techniques.

The terrain generation will be parameterized to allow user control.

Keywords: Procedural Generation, Biomes, Populations, Virtual Reality, Job System, Optimization, Parametrization.

Agradecimientos:

Índice general

I. GUÍA DE LA MEMORIA

Este es el prefacio de la memoria. A continuación, se presenta una breve descripción de los capítulos que componen esta memoria:

Capítulo 1. INTRODUCCIÓN. En la introducción se hace una introducción a la generación procedural y su influencia en la industria de los videojuegos, destacando su evolución y su impacto. También se destacan las motivaciones que han impulsado este proyecto y algunas de los objetivos que se pretenden llevar a cabo con esta propuesta.

Capítulo 2. PLANTEAMIENTO DEL PROBLEMA. Como se indica en el propio nombre, se indican los obstáculos que plantea la generación procedural, la distinción de biomas, optimización temporal, la continuidad del terreno, distribución de poblaciones y objetos, así como el realismo de los resultados.

Capítulo 3. OBJETIVOS. Se indican los objetivos que se pretenden alcanzar en el proyecto y cómo se pretende conseguirlos haciendo una diferenciación entre objetivos generales y parciales.

Capítulo 4. ESTADO DEL ARTE. Aquí se expondrá el estado del arte, realizando un análisis de las soluciones para generación procedural, estudiando herramientas ya existentes en el mercado con objetivos similares a la propuesta de este proyecto, se analizan métodos y técnicas relacionadas con la generación de terrenos, biomas y ciudades que abordan los temas más principales y secundarios, así como técnicas para la optimización y mejora del rendimiento general, además de factores claves para la compatibilidad con la realidad virtual. También se sacarán conclusiones de la investigación realizada.

Capítulo 5. DESARROLLO DE LA SOLUCIÓN. En este capítulo se detalla el análisis, diseño e implementación de la todos los componentes, sistemas y algoritmos empleados para la implementación en Unity de la propuesta de este TFM.

Capítulo 6. ANÁLISIS DE RESULTADOS. Finalmente se recogen todos los resultados de las pruebas establecidas para la evaluación de los algoritmos que crean los terrenos de la herramienta. Se describe el conjunto de parámetros empleados y las métricas establecidas para la evaluación de la variable de interés y para la validación de los resultados, tanto visuales como de rendimiento. Se analizan los resultados obtenidos y se identifican los factores y limitaciones que han podido influir en ellos.

Capítulo 7. CONCLUSIONES Y LÍNEAS FUTURAS. Por último, se extraen las conclusiones a partir del diseño, desarrollo y experimentación del método propuesto para la elaboración de la herramienta. Se describen futuras líneas de trabajo que

complementen el desarrollo alcanzado y se indica la aplicación de este proyecto en el ámbito del desarrollo de contenido multimedia.

II. MEMORIA

Capítulo 1

Introducción

1.1. Introducción

En este apartado haré una contextualización del tema de este TFM en la industria de los videojuegos actualmente y daré una visión general de la estructura del proyecto, indicando brevemente de qué partes se compondrá y cuál será el proceso de creación del mismo.

Durante los últimos años, y más concretamente durante la última década, el desarrollo de la industria de los videojuegos ha sido testigo de una evolución enorme, dado el auge del consumo de videojuegos y todo tipo de productos multimedia y digitales.

Esta tendencia, que se ha agudizado en los últimos años aún más, ha provocado que las experiencias que ofrecen los videojuegos sean cada vez más completas, más cautivadoras e inmersivas, dando lugar a desafíos técnicos cada vez mayores. Como consecuencia de estos cambios, se han producido adaptaciones desde el punto de vista técnico por parte de los desarrolladores quienes han tenido que buscar nuevas técnicas y métodos para generar contenido que cumpla las expectativas y requisitos de la audiencia y consumidores, y no sólo esto, los avances técnicos provocan una carrera y competencia entre los distintos desarrolladores y compañías por crear los mejores resultados y esto ha llevado la innovación a un nivel aún mayor.

Dentro de las nuevas técnicas y métodos que se han implementado en el desarrollo de videojuegos en los últimos años entra la Generación de terreno procedural, tema sobre el que trata esta TFM, y en el que se ha intentado realizar una herramienta con la que obtener resultados de gran calidad de manera eficiente y convincente.

Uno de los motivos por el que la generación procedural de terreno a ganado popularidad en este auge en la generación de videojuegos ha sido la popularización de los juegos de mundo abierto. Generar terrenos continuos y sin interrupciones en los límites de dichos mundos notables ha generado experiencias que han sido recompensadas por la audiencia. Juegos como No Man's Sky o Diablo son prueba del reconocimiento que han logrado algunos títulos que han usado esta técnica. Pero ante todo, Minecraft es el juego que sin duda ocupa el primer puesto en el uso de la generación procedural, siendo una referencia gracias a su éxito cosechado en multitud de juegos que pretenden emplear esta técnica.

Parte del éxito de este tipo de juegos es la renovación del terreno de juego, creando experiencias diferentes cada vez de manera ilimitado, lo que puede suponer una fuente de entretenimiento en sí mismo. Si además estos espacios de juego generados, son me-

dianamente realistas, cuentan con diversidad de paisajes, climas, simulaciones físicas y demás componentes que contribuyen a crear una sensación inmersiva, se está dando lugar a un activo diferencial en la jugabilidad y en la competitividad del juego frente a los demás. Por otro lado, la influencia de la generación procedural de terrenos no se limita únicamente a la creación de entornos y espacios. También se destaca en otros aspectos de la industria de los videojuegos como la generación de ciudades completas en juegos de mundo abierto, la creación de misiones y contenido diverso que aumenta la rejugabilidad o la generación de personajes. La música y los efectos de sonido también se benefician de la generación procedural, adaptando la banda sonora y la atmósfera del juego en tiempo real para acompañar la acción.

Por las razones mencionadas es que la generación procedural se ha convertido en un activo muy importante en la última generación de videojuegos, pero las razones mencionadas no son la únicas que han contribuido a ello. Además de la generación de la malla del terreno, la generación de texturas y elementos artísticos, suponen un ahorro de tiempo significativo de tiempo y espacio de almacenamiento, lo que colateralmente permite una mayor inversión en otros campos del desarrollo que ya no tienen que ser destinada a la generación del 'playground'.

La estructura de este proyecto organiza en un establecimiento de los objetivos a alcanzar, que en este caso son la generación del propio terreno, la generación de biomas (con sonidos, vegetación y cambios en el aspecto del terreno y orografía), la generación de espacios habitados, como aldeas o ciudades y su compatibilidad con realidad virtual, por lo que un rendimiento óptimo es fundamental para poder alcanzar los objetivos mencionados. Para ello se hará uso del sistema DOTS de Unity tratando de aprovechar lo más posible la capacidad de la CPU del equipo y permitir la generación en tiempo real y compatible con RV. También se hará una investigación y análisis previos de las tecnologías, técnicas y aplicaciones ya existentes que tienen un fin similar al de este proyecto para hacer uso de ellas en el diseño y la implementación de este TFM. En cuanto al análisis, cabe mencionar que se hará un análisis temporal y económico del coste del proyecto.

Por último se llevará a cabo la implementación realizando un diseño UML a partir del cual producir el código. Se realizarán pruebas de rendimiento y se obtendrán conclusiones de ellas. Además, se realizarán valoraciones sobre futuros trabajos o ampliaciones de este proyecto.

Cabe mencionar para finalizar que además se implementará un pequeño gameplay a modo de demostración de cómo funciona el resultado obtenido en el ámbito de la gamificación. Para esto se implementará un agente que navegará por el terreno mostrando los diferentes aspectos que se desarrollan en este TFM, como la eficiencia en la generación, los biomas y poblados, la distribución de los elementos que se instancian, la vegetación, etc.

1.2. Motivación

Como ya he mencionado en la introducción, la industria del videojuego exige cada vez más productos más convincentes que satisfagan la demanda de los consumidores. Dentro del industria de los videojuegos existen diferentes categorías de productoras de videojuegos dependiendo de su tamaño y de la calidad técnica y visual de sus productos. La principal motivación de este proyecto es la creación de un asset que permita a los estudios con menos recursos poder realizar contenido de mundo abierto y de calidad,

con una atmósfera inmersiva con la que crear mundos que transmitan experiencias, que permitan una buena jugabilidad y con los que poder llevar a cabo los trabajos en la imaginación de los desarrolladores, que junto con el desarrollo de otras tecnologías como la IA en combinación, se puedan lograr trabajos de gran calidad para dar alcance a un mayor número de gente a producir los resultados que deseen. Además de esto, también supone un reto desde el punto de vista técnico conseguir incluir todas estas funcionalidades y características en una herramienta y hacerlo de tal manera que sirva para ser compatible con tecnologías en auge como la RV, con la cual se alinea en la búsqueda de experiencias inmersivas.

La búsqueda de soluciones técnicas innovadoras y la aspiración de contribuir al crecimiento de la comunidad de desarrolladores son las motivaciones que guían este esfuerzo. La generación procedural de terrenos en Unity tiene el potencial de transformar la forma en que se crean mundos virtuales, y este proyecto se esfuerza por lograr precisamente eso.

Capítulo 2

Planteamiento del problema

Para el desarrollo de una herramienta de las características descritas se plantean diferentes cuestionas a resolver para cada uno de los apartados que se pretenden implementar en este proyecto.

2.1. Problemáticas en la Generación de Terreno

En la generación de la malla del terreno aparecen varias problemáticas a abordar que describo a continuación:

- La creación de chunks de terreno conforme se mueve el jugador.
- La coherencia entre distintos chunks, es decir, que no hayan discontinuidades entre chunks.
- Los chunks han de almacenarse de manera eficiente para poder acceder a ellos rápidamente cuando se deban volver a mostrar.
- Implementación de LOD.
- Implementación de la herramienta para modificar los vértices del terreno a posteriori.
- Mantener la concordancia entre chunks con diferente nivel de detalle mediante teselación.
- Texturización del terreno generado mediante shaders tripalanares.
- Adecuación de los nombres para un uso lo más sencillo posible y mejorar la UX.

2.2. Problemáticas en la Generación de Biomas

Para la generación de biomas se han de tener en cuenta los siguientes aspectos:

- Manejo de los algoritmos de generación para usar aquellos que produzcan resultados que mejor se ajusten a cada tipo de bioma por su orografía característica.

- La diferenciación de texturas entre mallas de terreno pertenecientes a diferentes biomas contiguos para que haya discontinuidades muy notables.
- Los cambios en la parametrización para ajustar la orografía del terreno a su bioma.
- La generación de assets propios para cada bioma y su distribución, como rocas o vegetación.
- Generación de flora y fauna (agentes autónomos) coherente (en sitios correctos y con una distribución correcta).
- Generación de sonidos para cada tipo de biomas con transiciones suaves.
- Ajuste mediante teselación entre mallas de chunks pertenecientes a diferentes biomas.
- Presencia de cuerpos de agua como lagos, mares o ríos han de ser considerados a la hora de generar los terrenos de cada bioma.
- Instanciación eficiente de los assets mencionados mediante el uso de estructuras de datos o algoritmos específicos.

2.3. Problemáticas en la Generación de Poblaciones

En cuanto a la generación de poblaciones se han considerado que hay que prestar atención a los siguientes factores:

- La distribución de edificios ha de ser coherente y regida por ciertas reglas.
- Debe haber un diversidad visual entre los edificios, por lo que se deberán distribuir las diferentes constricciones de manera que cree un resultado realista.
- Las poblaciones deben ser coherentes con el entrono donde se implanten.

2.4. Problemáticas en la Compatibilidad con Realidad Virtual

Por último, para la compatibilidad con RV, se necesitará integrar las siguientes funcionalidades.

- Uso de DOTS de Unity para paralelización del código.
- Prevención de los *'memory leaks'* para su rendimiento óptimo en tiempo de ejecución, etc.
- Optimización del cálculo de colisiones y físicas para un mejor rendimiento.
- Optimización del ambiente inmersivo para una mejor experiencia de juego.
- Ajuste de la escala de los elementos.

Capítulo 3

Objetivos

3.1. Objetivos Generales y Específicos

3.1.1. Objetivos Generales

El objetivo general de este proyecto es desarrollar una herramienta de generación procedural de terreno en Unity que permita a los desarrolladores de videojuegos crear mundos convincentes e inmersivos, con una alta calidad y realismo, dinámicos y que sea compatible con la realidad virtual (RV).

3.1.2. Objetivos Específicos

Para la consecución del objetivo general se han de conseguir los siguientes objetivos parciales:

- Generación de una malla 3D de manera procedural y manejo de esta mediante el uso de varios algoritmos de generación y estructura de datos.
- Parametrización mediante interfaz de usuario.
- Implementación de herramienta de edición del terreno generado.
- Texturización de la misma mediante shaders acorde a su nivel de altura y al bioma que representa.
- Instanciación de elementos correspondientes al bioma.
- Generación de sonidos, vegetación y poblaciones acordes a los biomas.
- Optimización de la generación del terreno, las instanciaciones y cálculo de físicas mediante DOTS de Unity.
- Adaptación de la escala del contenido generado y optimización de la inmersividad.

Capítulo 4

Estado del arte

4.1. Introducción

En esta sección se hará una contextualización y se profundizará sobre qué es la generación procedural, realizando un análisis e investigación de aquellos antecedentes, técnicas, algoritmos e implementaciones previas, así como se analizarán otras herramientas que tengan que ver con la generación de mapas de manera procedural, tanto para Unity como para otro tipo de motores o plataformas, y se realizará una comparación entre estas y el trabajo realizado en este TFM.

Se investigará sobre los distintos algoritmos de generación de mapas de alturas, de biodiversidad, de teselación, de reducción de geometría para LOD, sobre el uso de Job System (tipos de Jobs y cuáles son más apropiados), modelos de distribución probabilística para instanciación de assets, técnicas de simulación de agentes y climas, de modelación en base a masas de agua, sobre métodos de optimización de la inmersividad y más.

4.2. Contextualización

4.2.1. Definición y Conceptos Básicos

La generación procedural o generación por procedimientos, también conocida como PTG (Procedural Content Generation) es una técnica que consiste en la creación de datos a través de algoritmos en tiempo real en lugar de generar estos datos de manera manual.

Esta técnica se utiliza habitualmente en gráficos por computador, simulación y videojuegos. En el ámbito de la los gráficos se utiliza para la creación de texturas o geometría.

En videojuegos se utiliza en compresión y reutilización de datos, escalabilidad o mejorar la rejugabilidad. Concretamente, en el ámbito de videojuegos la generación procedural se usa para generación de niveles para juegos estilo 'Rogue', generación de terrenos y mapas para juegos de mundo abierto, diálogos, instanciación de objetos, etc.[1].

También se ha utilizado para otros diversos fines, como cine para el desarrollo de un gran número de objetos similares con pequeñas variaciones que dan lugar a staff decorativo y efectos visuales. Un ejemplo de sus uso en el cine es el Señor de los Anillos, donde se usó esta técnica para la generación masiva de soldados. [2]

Las razones más comunes para el uso de la generación procedural incluyen: archivos de menor tamaño, mayor cantidad de contenido que se pueden crear manualmente o la inclusión de aleatoriedad para experiencias de juego más sorprendentes. [2]

Cabe mencionar que a menudo se confunde el térmico generación aleatoria con generación procedural, que no son equivalente. La generación aleatoria está englobada en la generación procedural, pero carece guías ni restricciones, por lo que puede producir resultados indeseados y anárquicos.

Por último, para mencionar las ventajas y desventajas que supone esta técnica mencionaré algunos pros y contras:

- **Pros:**
  - Archivos de menor tamaño.
  - Menor esfuerzo para agregar gran cantidad de contenido.
  - Jugabilidad menos repetitiva, por tanto mejor rejugabilidad.
  - Mayor eficiencia y resultados coherentes.
  - Produce resultados reproducibles usando los mismos parámetros.
- **Contras:**
  - Más difícil visualizar y depurar los algoritmos durante su creación.
  - Dificultad para cambiar el algoritmo una vez construido.
  - Sin un conjunto definido de reglas, la generación procedural puede volverse muy desordenada y potencialmente romper el juego.
  - Puede llevar más tiempo crear un algoritmo que la creación manual.

4.2.2. Historia y Evolución

La generación procedural de contenido para videojuegos se remonta al propio comienzo de la industria de los videojuegos. En la década de 1970, Don D. Worth utilizó algoritmos simples en su juego de Beneath Apple Manor para crear mazmorras en este RPG. Posteriormente, en 1980, se lanzó 'Rogue'que también era un juego de mazmorras con creación de niveles. Este juego, además, fue referencia para la generación posterior de juegos con las siguientes características:

- Jugabilidad por turnos.
- Niveles generados proceduralmente.
- Muerte permanente (sin funcionalidad de carga/guardado).

Que pasaron a llamarse juegos de tipo Rogue. [3]

Los primeros videojuegos gráficos fueron frenados por las fuertes limitaciones de memoria. Esto provocó que el contenido, como mapas, fuera generado algorítmicamente ya que simplemente no había espacio suficiente para almacenar una gran cantidad de niveles preconstruídos y arte de los niveles.

Figura 4.1: Mapa generado del videojuego Rogue primera versión ASCII.

Existen varios ejemplos de generación procedural en los primeros juegos. Por ejemplo, Ákalabeth'(1980) de Richard Garriott utiliza semillas para generar el mundo, permitiendo a los jugadores regresar a un mundo dado utilizando el mismo número de semillas. Otro ejemplo fue 'The Sentinel', en el cual se almacenaron supuestamente diez mil niveles diferentes en solo unos pocos kilobytes. Élite', otro juego que hizo uso el PCG, originalmente planeaba tener un número masivo de galaxias, pero solo se seleccionaron ocho debido al temor de que un universo tan vasto resultara poco creíble para los jugadores. Otros ejemplos notables incluyen 'Rescue on Fractalus'(1985), que emplea fractales para crear montañas alienígenas, y 'River Raid'(1982) de Activision, que utiliza números pseudoaleatorios para generar obstáculos en un laberinto desplazable. [2]

Ya en la década de los 90, el artículo 'Solid Texturing'de Darwin R. Peachey, publicado en 1985 [4], fue una de las primeras publicaciones que trataba explícitamente sobre la generación de contenido procedural, donde se proponía una técnica similar al mapeo de normales actual para que las texturas bidimensionales pareciesen tridimensionales. De este artículo surgieron nuevas publicaciones que daba lugar a la generación procedural como campo de investigación dado que estos trataban sobre generación de terreno [5] o diseño de plantas mediante fractales [6].

Un ejemplo de los resultados de iniciar la investigación en generación procedural fue la herramienta RenderMan, de Pixar, la cual servía para generar texturas, materiales y primitivas simples para usarlas en animación, por lo que la generación procedural transgredió los videojuegos y se empezó a usar en más ámbitos.

Posteriormente en la década de los 2000 y con el aumento de la investigación, las técnicas para la generación mejoraron y se empezaron a usar en juegos AAA donde el PCG se usaba como un elemento concreto del juego, no por obligación debido a la falta de recursos como en épocas anteriores. Algunos juegos de estos años que usaron esta técnica fueron: Command and Conquer: 'Red Alert 2'(2000), 'Diablo'(1996) y 'The Elder Scrolls II: Daggerfall'. [3]

En 2004, el grupo '.theprodukkt'lanzó un shooter llamado '.kkrieger'. Este juego, que ganó el primer lugar en la competencia de juegos de 96k en Breakpoint en abril de 2004 no llegó a ser lanzado, permaneció como una demo. El objetivo de este juego fue demostrar la capacidad de generar un videojuego con un tamaño extremadamente reducido, generando texturas, modelos, animaciones, música y niveles usando con herramientas procedurales. Los desarrolladores decidieron publicar sus herramientas en forma del editor .werkkzeug. [7]

Como hemos visto, a comienzos del nuevo milenio, el almacenamiento dejó de ser un

Figura 4.2: Muestra del juego .kkrieger. Tamaño del archivo: 96 kb

problema y la creación de escenarios se volvió más sencilla, en gran parte gracias a los nuevos motores gráficos como Unreal Engine 4 o Unity3D. Esto, junto con la disminución de la popularidad del género Roguelike, frenó el progreso de los métodos de generación procedural.

Sin embargo, el lanzamiento de 'Minecraftén 2011 cambió esto completamente, ya que su popularidad volvió a destacar la generación procedural y su otra gran virtud, la variedad de contenido y rejugabilidad. En Minecraft, cada partida crea un mundo completamente diferente al anterior, lo que permite a los jugadores volver a jugar el juego y experimentar nuevas experiencias.

Minecraft supuso un punto de inflexión para la generación procedural en los videojuegos, siguiendo aún vigente a día de hoy, demostrando que los juegos basados en rejugabilidad, mundo abierto y generación procedural con elementos cambiantes y dinámicos están más viva que nunca.

4.3. Antecedentes

La generación procedural de contenido ha sido fundamental en el desarrollo de entornos virtuales y videojuegos, permitiendo la creación dinámica de terrenos, biomas, ciudades y otros elementos de manera eficiente y escalable. En este apartado, se examinarán las herramientas y trabajos previos relacionados con la generación procedural, centrándose principalmente en las aplicaciones para motores gráficos como Unity y Unreal Engine.

4.3.1. Herramientas para Unity y Otros Motores

Para crear terrenos en Unity existen diversos assets, disponibles tanto como asset del asset store como softwares independientes. Aquí una lista con descripciones de algunos de los softwares más utilizados en la actualidad categorizados por sus funcionalidades y propósito:

Generación de Terrenos y Paisajes

- **Terrain Generator:** Esta herramienta de la asset store de Unity permite la generación paramétrica de terrenos complejos mediante cálculo masivo en GPU, los cuales se pueden modificar mediante un sistema de nodos pudiendo añadir simulaciones físicas realistas. También cuenta con un sistema de generación de biomas,

pudiendo mezclar biomas ya generados. Actualmente es un asset de pago con dos ediciones: Indie y Pro. [8]

- **MapMagic:** Herramienta diseñada para Unity de pago que se puede integrar importándolo en el propio Unity. Es un generador de mapas basado en nodos que cuenta con módulos adicionales que permiten colocar objetos y generar biomas y caminos. Los nodos de esta herramienta representan generadores de terreno (ruido, voronoi, etc.) u objetos (mezcla, curva, erosión, etc.) cuya acción conjunta permite crear los terrenos. Además, existe una paquete adicional: MapMagic2 que cuenta con módulos adicionales que añade más funciones. Además de ser integrable con Unity, si se desea integrar en el workflow con otros motores se pueden exportar los terrenos generados para ser usados en estos adaptando los datos a los formatos específicos de cada motor. [9]

- **Terrain Composer 2:** Terrain Composer 2 es otra herramienta del asset store de Unity que consiste en un generador de 'tiles'de terreno basado en nodos mediante GPU que utiliza un sistema basado en capas.

  Algunas otras características destacables son la exportación de mapas de altura, de texturas, color y normales; su integración con RTPv3 (Relief Terrain Pack Version 3", un conjunto de herramientas para Unity que proporciona una serie de shaders y funciones avanzadas para crear terrenos realistas), biblioteca completa de ruido, incluyendo ruidos erosionados, entre otras. Esta herramienta también es de pago pero con un precio menor que las anteriores. [10]

- **Terragen:** Terragen es un software utilizado para construir, renderizar y animar entornos naturales realistas, mediante la importación de da6tos de alturas o mediante el uso de un editor de terreno con funciones procedurales integradas para lograr terrenos a escala global con un grado detalles enorme. Este software permite tener control, además de sobre el terreno, sobre el clima, ríos, lagos, océanos, etc. y también ofrece control de los shades utilizados para terrenos, texturas, desplazamientos de micropolígonos, nubes y distribución de objetos con formato nativo TGO u OBJ que pueden haber sido modelados en otros softwares.

  Otras características reseñables de esta herramienta es que posee un renderizador híbrido de micropolígonos y trazado de rayos, especialmente optimizado para manejar grandes desplazamientos y paisajes de gran escala. Un sistema de iluminación global a éscala planetaria'para simulación de sistemas planetarios. Una atmósfera fotorrealista con control sobre las nubes con efectos volumétricos y '2.5D'. Opciones de animación de parámetros y editor de nodos para control de texturas y sombreados. Por último es integrable con otros motores mediante la exportación de sus modelos en alta resolución. [11]

- **Gaia:** Este asset de pago de la asset store de Unity permite generar terrenos y paisajes para móviles, realidad virtual, consolas y escritorio tanto fotorrealistas como lowpoly. Cuenta con un sistema de edición de terrenos permitiendo un workflow artístico y procedural en la generación de paisajes. También cuenta con un sistema de generación de biomas, aldeas y de configuración de viento, agua y efectos, incluyendo efectos de sonido para dar una mayores sensación de inmersión. Múltiples sistemas de cielo e iluminación (incluido el momento del día), sistema meteorológico con soporte para lluvia y nieve, sistema de agua y shaders de alto rendimiento. Es un software acelerado por GPU con tiene soporte integrado con URP y HDRP.

Otra característica muy destacable de este asset es el conjunto de assets de muestra con los que viene, el cual se puede personalizar añadiendo o quitando assets de esta colección pudiendo generar los diferentes biomas en base a estos assets. Además, cuenta con una versión pro que añade aún más funcionalidades y activos. [8] [12]

- **Gaea:** Gaea es un software independiente especializado en la generación procedural de terrenos y paisajes, incluyendo formas geológicas, simulaciones de erosión, nieve y ríos, herramientas para edición de rocas y generador de texturas basado en nodos. Este software posee un sistema basado en nodos para realizar simulaciones. Esta herramienta presenta un flujo de trabajo no destructivo, lo que permite iterar sobre los paisajes para actualizar los terrenos generados. Con este software se pueden crear formas de manera procedural utilizando primitivas e importar modelos texturizados y convertirlos en terrenos. Una característica destacable de este software es su algoritmos de erosión, el cual posee flexibilidad para brindar control artísticos sobre la apariencia. Además de los terrenos también están sus biomas, pudiendo recrear ecosistemas complejos con datos relevantes como zonas en conflicto, nieve, agua y más.

  Este software permite también la creación de texturas sofisticadas utilizando mapas de datos, además de la personalización de los mismos junto con la exportación de máscaras, incluidos mapas PBR para poder texturizar en otros motores. Los terrenos generados en este motor son exportables con alta calidad gracias a su soporte para formatos universales junto a su generación de LOD automáticos. [13]

- **World Creator:** World Generator es otro software independiente de esta lista para generación de terreno procedural. Este software permite generar terrenos haciendo uso de cálculo en GPU de cualquier tamaño con diferentes grados de detalle. Además, posee herramientas de edición y escultura del terreno pudiendo no solo modificar su forma sino su mapa de colores. También permite crear ríos, montañas, carreteras y demás accidentes geográficos, así como efectos de erosión realista, entre otras características. Al igual que las demás herramientas anteriores, también permite la generación de biomas y cuenta con una gran librería de ellos predefinida.

  Este software cuenta con us propio motor de render con Ray-tracer en GPU. Además, es compatible universalmente, por lo que los terrenos generados en este software se pueden integrar con otros motores como Unity, Unreal Engine, Blender, etc. mediante la exportación de los terrenos generados en diversos formatos. Este software profesional de pago cuenta con su propia documentación y tutoriales, lo que lo hace muy completo. [14]

- **World Machine:** World Machine es otro software independiente que permite la creación de terrenos con diferentes aspectos, desde realistas hasta terrenos estilizados, ello posible gracias a su generación basada en fractales mediante un workflow de nodos que permite la edición del mismo con herramientas propias. También cuenta con sistemas de agua para crear efectos geológicos realistas, ríos, lagos, etc.; sistema de texturizado de materiales con soporte para renderizado basado en física (PBR) y exportación de terrenos como mapas de altura, malla y los datos de materiales, los cuales se pueden exportar en una variedad de formatos. También se pueden exportar escenas completas con GLTF.

  Este software, también profesional de pago posee diversas funcionalidades y utilidades para la generación de terrenos con compatibilidad con motores como Unity y Unreal Engine, o renderizadores 3D como Blender, Maya o Cinema4D. [15]

- **Vue:** Este software es un paquete de entornos profesional que cuenta con varios tipos de licencia: Creator, Professional y Enterprise, utilizado para la creación de entornos y paisajes. Cuenta con herramientas para artistas para la generación procedural de vegetación y terrenos, pudiendo crear incluso planetas. A través sus herramientas de pintura manual de los terrenos, se puede lograr un mayor control sobre la apariencia final del terreno, que en combinación con ruidos y fractales procedurales da lugar a resultados realistas. Además, VUE cuenta con un modelo atmosférico que simula condiciones de iluminación realistas. VUE además proporciona una gran biblioteca de assets de plantas para crear escenas de vegetación variadas. Este software se puede integrar con motores como Unity o Unreal mediante plugins y permite la exportación de sus escenas o activos en una amplia variedad de formatos. [16]

Herramientas Procedurales Generalistas

- **Houdini:** Houdini es uno de los softwares más famosos en la industria de los VFX dada su increíble versatilidad ya que se puede utilizar para producir prácticamente de todo, y los terrenos no son una excepción. Si bien no es un software específico para este fin, la versatilidad que posee permite dar lugar a terrenos generados algorítmicamente, físicamente realistas, editables y exportables. En Houdini los terrenos se pueden generar utilizando varios métodos, pero el más común es el uso de nodos y operadores procedurales que combina la generación de ruidos con operaciones matemáticas que esculpen el terreno. El terreno se puede generar mediante la combinación de fractales de terreno junto con el uso de máscaras para lograr efectos detallados y realistas, produciendo fracturas, erosiones, texturas mediante capas de ruido y nodos de deformación. Además, Houdini cuenta con herramientas de escultura para personalizar los terrenos generados y se pueden repartir assets con la distribución de objetos mediante nodos procedurales.

  Houdini es una herramienta de tal magnitud, que a pesar de no ser específica para generar este tipo de assets permite la producción de terrenos totalmente personalizables y con alto grado de realismo, con capacidad de exportación a otros softwares como Unreal. Si bien es un software generalista que no permite la generación en tiempo real, al igual que los otros softwares independientes, merece estar en esta lista. [17] [18]

Herramientas de Generación de Entornos Urbanos

- **EasyRoads3D:** EasyRoads es un asset gratuito disponible en la asset store de Unity, cuya función es la generación de carreteras de manera eficiente en Unity. Si bien no se puede decir que EasyRoads sea una herramienta procedural, pues la creación de carretas se realiza de manera manual, es un asset muy utilizado para la creación de entornos en terrenos que sí han podido ser generados proceduralmente de manera rápida y que en combinación con scripts propios, sí puede ser usado de manera automatizada. Es por esto que ha sido incluida en esta lista, aunque sea como mención complementaria. [19]

- **CityGen3D:** CityGen3D es otro asset de pago de la tienda de Unity utilizado para la generación de ciudades a partir de datos sobre carreteras, árboles, edificios, etc. utilizando información de OpenStreetMas. Esta herramienta permite la importación

de mapas de altura del terreno, generación automática de carreteras, edificios procedurales con niveles automáticos de detalle (LOD), generación de biomas. Otras características de este asset es su compatibilidad con diferentes render pipelines de Unity, como el High Definition Render Pipeline (HDRP), Universal Render Pipeline (URP) y el Built-In pipeline, lo que la hace aún más completo. [20] [21]

- **Building Generator:** Este asset de bajo costo en la tienda Unity es un generador de edificios como su propio nombre indica que permite construir y pintar edificios utilizando parte premodeladas que ensambla y asigna materiales seleccionados generando una malla única combinada. [22]

  No es una herramienta procedural en sí al igual que pasaba con otras anteriores pero que puede jugar un papel muy útil en la generación de ciudades para mapas procedurales, sumado a los mecanismos procedurales en su funcionamiento que parece tener, se ha incluido en esta lista.

- **CityEngine:** CityEngine es un software de modelado 3D avanzado para crear entornos urbanos enormes, interactivos e inmersivos en menos tiempo que las técnicas de modelado tradicionales. Las ciudades que crees utilizando CityEngine pueden basarse en datos SIG del mundo real o mostrar una ciudad ficticia del pasado, presente o futuro. Este software permite crear ciudades en 3D y diseñar entornos urbanos creando modelos gracias a sus herramientas de edición. Permite traer activos para construir un contexto 3D alrededor y exportar el trabajo para llevarlo de nuevo a un software de visualización o motor como Unity o Unreal. Además, permite automatizar los flujos de trabajo con código procedural y Python. [23]

Herramientas de Generación de Vegetación

- **Vegetation Studio Pro:** Este también es un asset gratuito de la tienda de Unity añade funcionalidades a su versión anterior (Vegetation Studio). Este software, al igual que EasyRoads3D suele usarse sobre terrenos ya generados para agilizar y realizar la población del terreno con vegetación de manera eficiente. Algunas de las características de esta herramienta son la creación de máscaras de biomas, funciones por lotes para ajustar configuraciones de biomas, y la creación de áreas de máscara para la vegetación, entre otras.

  Aunque no es una herramienta procedural en sí misma, es una herramienta que se integra con la generación de terrenos procedurales permitiendo la creación y manejo de biomas, por lo que es una herramienta habitual a la hora de generar paisajes con biomas, tema sobre el que trata el trabajo de este proyecto [24]. Al igual que no el asset anterior, esta son las razones para incluirla en esta lista.

- **SpeedTree:** SpeedTree es un software independiente profesional de pago con integración en Unity. Esta herramienta se utiliza para el modelado de vegetación permitiendo la creación vegetación realista de forma rápida y eficiente, por lo que es un asset valorado por su uso en la generación de biomas escenarios naturales. Esta herramienta ofrece capacidades de modelado manual pero también de generación procedural, permitiendo a los usuarios crear variaciones de la vegetación con control sobre parámetros específicos. Algunas de las características de la herramienta en las que se emplean técnicas procedurales son procedurales: randomización, colisión de hojas, Mesh anchors, Clusters, Divisiones de la mesh, Control de la forma, color de los vértices, manipulación de secciones de la mesh, UV tiling, entre otras. [25]

4.3.2. Comparativa Entre las Distintas Herramientas y Este Trabajo

En el apartado anterior se han mostrado diferentes herramientas y assets, algunos de ellos únicos de la asset store de Unity, otros independientes con capacidad de exportar sus activos o con compatibilidad mediante plugins, y otros que poseen integración con alguno de los motores aunque se pueden usar de manera independiente también. Hay softwares que permiten la generación de terreno junto con biomas y/o aldeas, incluso fenómenos climáticos, otros a parte de la generación de terreno tienen compatibilidad con realidad virtual, otros son herramientas que permiten la modelación de los paisajes generados de manera automática o automatizable, que generan fenómenos atmosféricos y climáticos para construir escenas realistas, etc.

Todas las herramientas mencionadas cumplen con propósitos que se pretenden llevar a cabo en el proyecto de este TFM, pero la propuesta de este trabajo reúne características de estos softwares para crear una herramienta diferente, capaz de generar terrenos realistas, no en una fase de preprocesado para generar la escena, sino en el propio tiempo de ejecución, mientras el jugador se mueve, aprovechando al máximo la capacidad de cómputo de la CPU haciendo uso del sistema DOTS como principal innovación de esta propuesta. Además de este factor diferencial, en este proyecto no sólo se crearán terrenos en tiempo real con una tasa de refresco que permita una buena jugabilidad, sino que se generarán biomas, se hará distribución de assets correspondientes a la vegetación y demás elementos geográficos y se generaran asentamientos y poblaciones para dar un resultado más realista. Todo ello acompañado de compatibilidad con realidad virtual, efectos de sonido, fenómenos climáticos y edición del propio terreno generado. Este conjunto de funcionalidades, permiten suplir las diferencias de calidad con los resultados de aquellos que permiten una generación más detallada y realistas desde el punto de vista físico, así como de todos aquellos que mejoran los resultados producidos por la herramienta que se propone, pero no permiten que se utilice en tiempo de ejection, de manera procedural y con compatibilidad con realidad virtual dadas las experiencias que permite desarrollar en tiempo de ejecución.

Todas estas características de la solución que se proponen, añadido a que es un proyecto que pretende mejorar la calidad de los desarrollos de estudios con poca capacidad económica o indie teniendo un bajo precio o directamente la gratuidad en un hipotéticos caso de que saliera al mercado, hacen de esta herramienta un producto diferente y relevante dentro del mercado de herramientas y softwares destinados a la generación de terrenos, entornos y paisajes.

Los pros y contras de esta propuesta podrían resumirse de la siguiente manera:

- **Pros:**
  - Generación procedural de terrenos, biomas y elementos de entorno de buena calidad que mejoran la rejugabilidad.
  - Integración Realidad Virtual para experiencias inmersivas.
  - Más asequible que algunas herramientas comerciales.
- **Contras:**
  - Dada la cantidad de funcionalidades que integra la curva de aprendizaje puede ser elevada.

- Limitación en la calidad de los resultados dada la generación en tiempo real.
- Se requerirá de equipos con prestaciones mínimas para que sean capaces de producir resultados destacables con buena tasa de refresco.

4.4. Algoritmos y Técnicas más Utilizados y Aplicaciones

EbDavid S. Ebert identifica las siguientes características que deben estar presentes de las técnicas de generación procedural en su artículo "Texturing and Modelling - A Procedural Approach. Morgan Kaufmann. 2003"[26]

- **Abstracción:** Lo algoritmos deben generar resultados centrándose en el propósito general y dar resultados completo y no centrarse en los detalles.
- **Control Paramétrico:** Los parámetros se definen para que correspondan con un comportamiento específico.
- **Flexibilidad:** Para imitar la esencia de algo del mundo real, no es necesario una recreación totalmente explícita, lo que permite mayor flexibilidad en su implementación.

4.4.1. Generación de Terreno

En este apartado se analizarán las técnicas y métodos más utilizados para generar mapas de alturas siguiendo la clasificación que se hace en [27] y se nombrarán algunos nuevos métodos siguiendo técnicas más novedosas.

La clasificación de técnicas generativas que se utilizan es la siguiente:

Métodos Estocásticos

Estos métodos, basados en el uso de parámetros, intentan generar resultados aleatorios, los cuales se pueden aplicar recursivamente para incrementar el detalle. Estos métodos se caracterizan por su rapidez y eficiencia -una de las razones principales por las que se ha decidido emplear esta técnica en este proyecto-. Aunque estos métodos son rápidos, no se pueden controlar la posición de las características de los terrenos generados (cañones, valles, etc.), que se generarán en posiciones aleatorias.

- **Fractales:** Estos métodos consisten en generar figuras autosemejantes mediante métodos iterativos o recursivos. Dentro de esta categoría quedan incluidos tanto los fractales caóticos como los ruidos para generar mapas de alturas. Algoritmos como:
  - **Ruido Perlin y Simplex:** El ruido Perlin es un tipo de ruido suave y continuo basado en la generación de valores pseudo-aleatorios desarrollado por Ken Perlin en la década de 1980. El ruido Simplex es una variación de este también desarrollado por Perlin en 2001, el cual genera mejores resultados visuales y en menor tiempo. Ambos métodos se pueden aplicar en varias capas con diferentes valores de frecuencia para generar un mapa de alturas fractal con detalles en diferentes escalas. [28]

Figura 4.3: Ruido de Perlin simple.

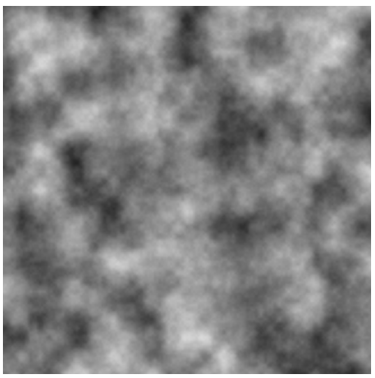

Figura 4.4: Ruido de Perlin fractal.

- **Ruido Worley:** El ruido de Worley es otro tipo de ruido que se genera en base a los diagramas de Voronoi, el cual produce patrones con estructuras celular que se genera dividiendo el espacio en celdas basadas en la distancia a un conjunto de puntos. A menudo se utiliza para generar terrenos con características como cuevas, crestas o túneles. [29]

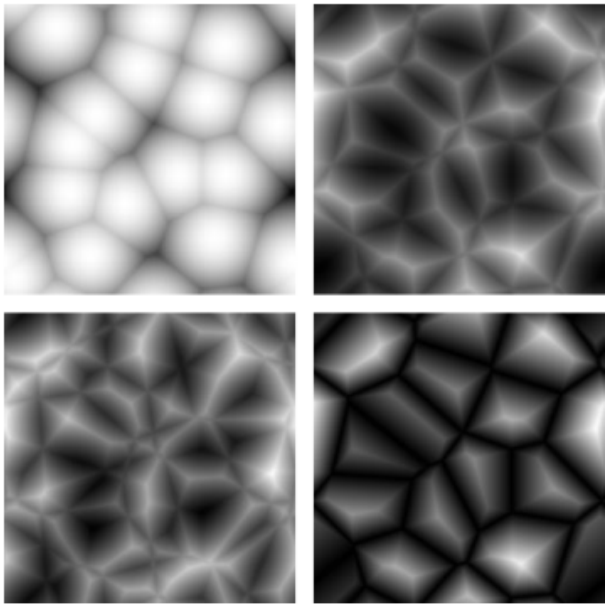

Figura 4.5: Ejemplos de diagramas de Voronoi con coeficientes $c_1 = -1$ (arriba izq.), $c_2 = 1$ (arriba der.), $c_3 = 1$ (abajo izq.), $c_1 = -1$ y $c_2 = 1$ (abajo der.).

- **Algoritmo de desplazamiento del punto medio:** En este método se genera terreno fractal a partir de un segmentos y, en cada iteración, dividir cada segmento en dos partes. La altura de los nuevos vértices se ajusta mediante un valor de desplazamiento aleatorio inversamente proporcional al nivel de recursividad en el que se esté aplicando el desplazamiento, creando así características fractales. [30]
- **Algoritmo diamante-cuadrado:** El algoritmo diamante-cuadrado consiste en dividir un cuadrado inicial con valores aleatorios en las esquinas en subcuadrados , donde la altura de los nuevos vértices se ajusta en función del promedio

de los vértices originales. Este proceso se repite varias veces, lo que da como resultado un terreno con una apariencia fractal. [31]

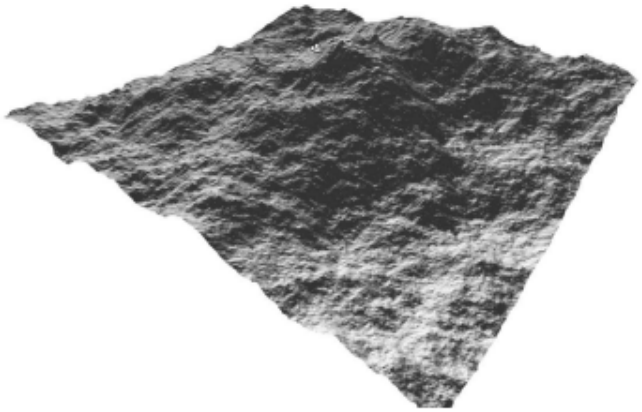

Figura 4.6: Terreno generado con el algoritmo de desplazamiento del punto medio

- **Gramáticas:** Las gramáticas son reglas para la creación de lenguajes formales en los cuales secuencias en crecimiento se aplican a un estado inicial, en [32] estas reglas se utilizan para deformar terrenos y así imitar las acciones de la erosión. Un ejemplo de método que utiliza las gramáticas son los L-Systems, los cuales se utilizan para generar fractales generando patrones a partir de reglas simples. Los Sistemas L fueron introducidos y desarrollados en 1968 por el biólogo y botánico teórico húngaro Aristid Lindenmayerc.

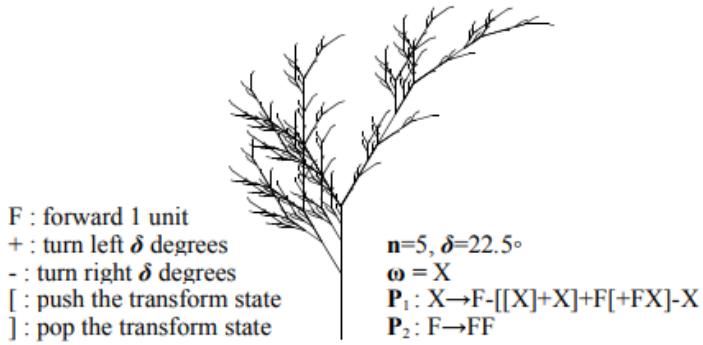

Figura 4.7: Formación de árboles a través de un intérprete de sistemas-L.

- **Tiling:** Esta técnica consiste en dividir el terreno en áreas de menor tamaño. Una de estas técnicas y la más conocida son los Diagramas de Voronoi, los cuales dividen la superficie del terreno en superficies con forma celular.

  Con esta técnica se permite usar diferentes parámetros para cada división -o celda en caso de los diagramas de Voronoi- de manera que se pueda romper la regularidad producida por algoritmos como los basados en ruido produciendo transiciones entre distintas áreas más naturales.

  Tiling solo permite la división del terreno, por lo que es común que se utilice junto a otros algoritmos para la generación de terrenos como en el método descrito en [33]

- **Parametrización:** Estas técnicas giran en torno a tener instrucciones de construcción que son controladas por el usuario, quien ajusta sus parámetros. La variabilidad de los resultados está limitada por el número de parámetros controlables. En [34] se presenta un método basado en la parametrización de tiles. En [35] se propone un método para crear cordilleras, los parámetros son altitud, pendiente y área de influencia.

Métodos de Simulación

Los métodos de simulación física pueden utilizarse en la generación procedural de terrenos para crear terrenos más realistas emulando fenómenos naturales y efectos de procesos físicos como la erosión, la deposición y la meteorización en el terreno, lo que da como resultado un terreno más realista. Algunos de los algoritmos de los métodos simulación más comunes son:

- **Geomorfológicos:** La erosión es uno de los principales factores de la formación del terreno, y estos enfoques se centran en la simulación de formas realistas. Suelen emplearse posteriormente a otros métodos de generación -como los métodos fractales- para añadir una capa de realismo.

  Dentro de esta categoría se pueden incluir los siguientes métodos:

  - **Basados en hidrología**: Los algoritmos basados en hidrología simulan el flujo del agua sobre el terreno, teniendo en cuenta factores como la pendiente, la lluvia y la evaporación. Estos algoritmos pueden utilizarse para crear terrenos con redes de ríos realistas, lagos y otras características de agua [36] [37]. En [38] se describe un modelo de generación de terrenos a partir de las masas de agua del mismo.
  - **Basados en erosión**: Dentro de los algoritmos de erosión existen dos principales: la erosión térmica y la hídrica, en los cuales se simulan los efectos de la erosión causada por el agua, el viento en el terreno o el desprendimiento de partes de terreno dado un ángulo de inclinación o talud, lo que da como resultado terrenos con características como valles, crestas y cañones. Como se ha mencionado anteriormente, estos suelen utilizarse en conjunto con otras técnicas como funciones de ruido o fractales. [39] [40]. Esta técnica se puede observar también en [33].
  - **Choque de placas tectónicas:** El levantamiento de terreno debido a la simulación de choques entre placas tectónicas también entra en esta categoría. Este proceso eleva todo el terreno con el tiempo provocando la creación de sierras. En [41] se emplea este método para para simular los procesos tectónicos que dan forma al terreno a gran escala y la deformación del terreno generando elevaciones y depresiones.

- **Ecosistemas:** Los métodos basados en ecosistemas, también llamados biomas, alteran la apariencia del terreno provocando que por ejemplo, dos terrenos con la misma apariencia geomorfológica luzcan de manera totalmente distinta debido a la presencia de vegetación u otros elementos ambientales que definan ecosistemas distintos.

  La simulación de ecosistemas para modelar la apariencia del terreno a menudo se lleva a cabo mediante sistemas multicapa como en [42], donde se genera el terreno

siguiendo un pipeline de varias capas calculando en ellas factores como la humedad, temperatura y el viento para simular las condiciones del terreno. En [43] se realiza un proceso similar, donde en función de la humedad y exposición al sol se genera vegetación.

Algunas de los métodos más empelados son:

- **Modelos Hidrológicos** Son modelos que simulan la influencia de masas de agua y las cuencas hidrográficas para la distribución de los diversos biomas junto con efectos adicionales como la erosión. El artículo [38], en su método describe también la distribución de biomas basada en cuerpos de agua durante la generación.
- **Diagrama de Voronoi**: Este enfoque permite la generación de terrenos con diversos biomas basándose en los diagramas de Voronoi para su distribución. En el paper [33] se emplea este método.
- **Reglas Geográficas**: Este método emplea reglas y parámetros como la latitud para asignar biomas de manera realista. En el método propuesto en [34], además de emplear diagramas de Voronoi, tiene en cuenta la latitud para controlar el tipo de biomas generados.
- **Distribución de Altura:** Este enfoque simula biomas con diversidad en función de la altitud de los terrenos generados. En el paper de Jacob Olsen [33] de nuevo, se emplea este método para generar diversidad en los biomas.
- **EDPCG (Experience-Driven Procedural Content Generation):** En [44] se describe el algoritmo EDPCG, el cual se emplea para ajustar los modelos generativos en base a la experiencia del usuario. Aunque no es un algoritmo o método estrictamente circunscrito a la generación de biomas específicamente en el artículo se nombra como método para la generación de contenido, lo que puede incluir la instanciación de assets o elementos, lo cual está íntimamente relacionado con el modelado de biomas, y puede ser un método a considerar en esta clasificación.

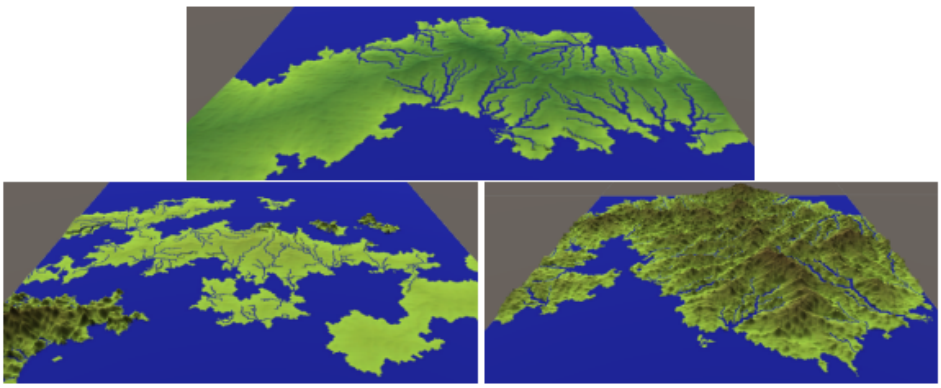

Figura 4.8: Terreno generado basado en cuerpos de agua [38]

- **Basados en Agentes:** Estos métodos no se basan en eventos geológicos, sino en la acción de los llamados agentes, que por lo general se dividen en constructivos y destructivos. Doran y Parberry en su artículo [45] definen cinco tipos de agentes: agentes de línea costera que crean masas terrestres iniciales; agentes de suavizado que

crean llanuras; agentes de playa que nivelan las áreas costeras; agentes de generación de montañas; y agentes de creación de ríos. La creación y el suavizado se realizan de manera aleatoria. El aplanamiento y la creación de montañas son aleatorios pero también incluyen reglas para cierto control.

Métodos de Aprendizaje

Estos métodos usan información del mundo real, generalmente en forma de imágenes o DEMs (Digital Elevation Models) y tratan de imitar el aspecto del mundo real.

- **Basados en Ejemplos:** Como se describe con mayor detalle en [27], existen diversas aproximaciones para los métodos de este tipo. Existen aproximaciones que copian pequeñas partes de las imágenes del mundo real sobre un boceto y se crean splines que den continuidad entre los parches. En otros métodos el terreno se representa como un árbol donde las el nivel de detalle incrementa a medida que se desciende en él, siendo la parte con mayor nivel de detalle las partes provenientes de imágenes del mundo real. Otros crean diccionarios de *átomos*, los cuales representan información de elevación, vegetación, luz y agua los cuales se obtienen de la descomposición de parches de imágenes y la combinación de estos es lo que produce los terrenos.

  - **Wave Function Collapse**: Este algoritmo consiste en la generación procedural extrayendo patrones a partir de datos de entrada. Para el funcionamiento de este algoritmo, se toma una muestra de entrada, por ejemplo un mapa de bits, y esta se divide en "tiles". A partir de esta división se extraen reglas de vecindad, para determinar que tiles pueden aparecer unos al lado de otros y cuales no. El algoritmo comienza generando un tile aleatorio y a partir de este se generan el resto, además, implementa un sistema de backpropagation cuando se llega a un estado donde no se pueden generar más tales que cumplen las reglas de vecindad para regresar a un estado donde sí y probar con distintas combinaciones, iterando este proceso hasta que se complete la generación. [46] [47]

    Este algoritmo es reconocido en áreas como la generación de niveles, texturas y mapas. Aunque es un algoritmo que puede aplicarse a muchos aspectos de este trabajo tales como la generación de biomas o poblaciones se ha decidido incluirlo en este apartado dada su generalidad, ya que puede ser una buena opción para generar una base consistente y utilizar otros algoritmos más específicos para la generación de biomas o poblados.

    Para el caso de la generación de terrenos podría empelarse creando tiles correspondientes a los distintos tipos de terreno, como: montañas, colina, prado, etc. y establecer reglas de vecindad entre estos para generar un terreno. Estos terrenos tendrían sus propias propiedades de alturas y se haría una interpolación entre sus valores para generar una mesh continua y más suavizada.

Además de este método, las redes neuronales y la inteligencia artificial también han sido empleadas para la generación de terrenos, y dada la naturaleza de estas técnicas cabría clasificarlas en esta categoría. En el artículo [48] se presenta un análisis comparativo de modelos generativos para la generación de terrenos en videojuegos de mundo abierto, abarcando Redes Generativas Adversarias (GAN), Autoencoders Variacionales (VAEs), Redes Neuronales Convolucionales (CNNs) y Generación de

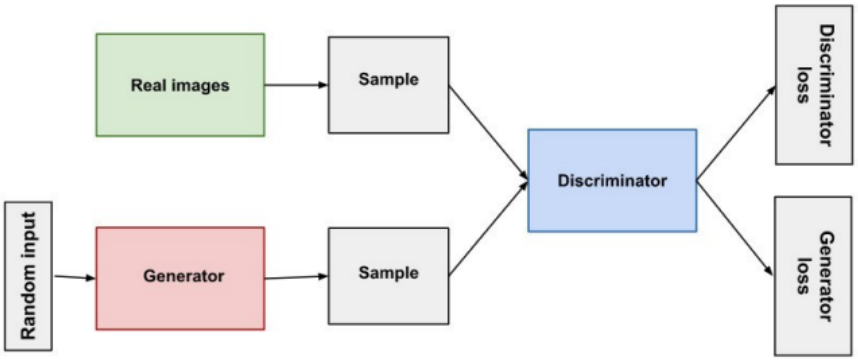

Figura 4.9: Esquema de funcionamiento de una GAN

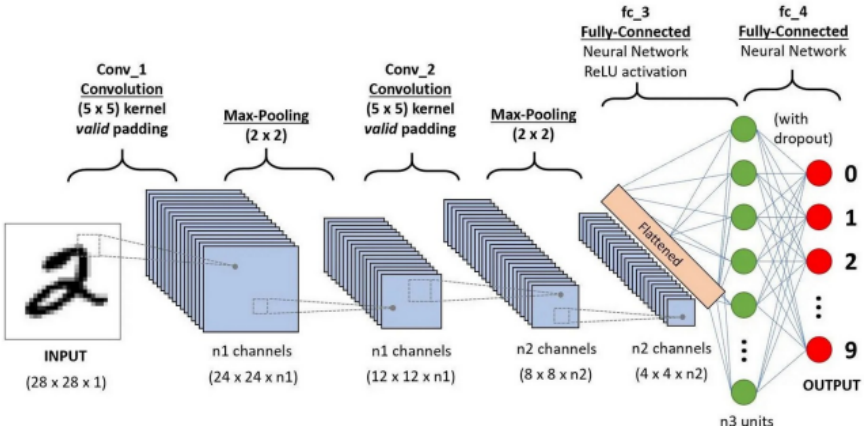

Figura 4.10: Esquema de funcionamiento de una CNN

Contenido Procedimental (PCG). El objetivo de este es evaluar la eficacia y adecuación de estos enfoques en la generación de terrenos diversos y realistas. Aquí se describen algunos de los modelos descritos:

- **Redes Generativas Adversariales (GAN):** A través de las GAN, dos redes neuronales, una generadora y una discriminadora, compiten entre sí para producir terrenos realistas y detallados, mediante el descarte que las características del terreno generado, moldeando la forma del resultado.
- **Redes Neuronales Convolucionales (CNN):** Las CNN extraen patrones espaciales de los datos de terrenos que se le pasan como entrada y generan terrenos a partir de estos patrones adquiridos para generar resultados similares pero innovadores.
- **Autoencoders Variacionales (VAE):** Los VAEs son un tipo de red neuronal capaz de aprender un modelo probabilístico de los datos de un terreno que se use como entrada para posteriormente generar poder replicar terrenos con características similares.

- **Basados en Búsqueda:** Muchos de estos métodos están basados en algoritmos evolutivos, los cuales están basados en IA que imitan el comportamiento de la evolución biológica [49]. No obstante el uso de estos algoritmos evolutivos presentan problemas en la codifican el terreno en cromosomas, el diseño de la funciones de ajuste y la variación del terreno, lo que produce como resultado terrenos demasiado planos comparados con los reales.

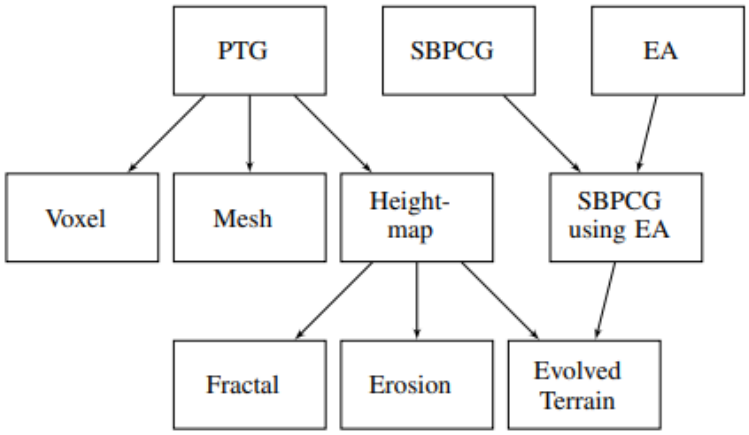

Figura 4.11: Diagrama de relación entre los algoritmos generación procedural (PTG), basados en búsqueda (SBPCG) y evolución (EA)[50]

- **Generación Procedural Inversa:** En estos métodos se extraen parámetros para generar terrenos a partir de ejemplos, para luego recrear las características del terreno de ejemplo usando los parámetros aprendidos de estos. Un caso de este método es el que se da en [51], donde se propone aprender sobra la densidad de vegetación en un área, para a recrear una brocha que instancie vegetación usando esta densidad obtenida.

Métodos basados en Bocetos

Estos métodos consisten en crear terrenos a partir de una información específica del usuario sobre la forma que este tendrá y sus características. Estos métodos mejoran el control sobre el terreno generado dando al usuario la habilidad de dar aspecto al terreno a través de interfaces como si se dibujaran. La desventaja de estos métodos es el menor realismo que logran en comparación con otros métodos.

- **Boceto 2D:** Este método consiste en generar trazas 2D que representan las siluetas de las montañas que se quieren hacer visibles desde el ángulo de la cámara.
- **Boceto 3D:** En lugar de hacer bocetos 2D para pasarlo a 3D mediante reconstrucción, se trabaja directamente con vectores 3D desde el comienzo o curvas en algunos casos. Una vez las referencias 3D están colocadas se produce la voxelización de los vectores que los transforma en aproximaciones discretas dando así estructura al terreno.

Families	Stochastic	Simulation	Learning	Sketches
Techniques	Fractals	Geomorphological	Example-based	2D sketch
	Grammars	Ecosystem	Search-based	3D sketch
	Tiling	Agent-based	Inverse procedural generation	
	Parametrization			

Figura 4.12: Clasificación general de las técnicas de generación según [27]

4.4.2. Generación de Poblaciones

La generación de ciudades, poblaciones o asentamientos conlleva la creación de edificios con organización siguiendo algún patrón y de caminos o carreteras. Para ello se han llevado a cabo diversas implementaciones de recreación de patrones de carreteras y generación de edificios pudiendo variar en en función del estilo arquitectónico.

Existe una gran diversidad de patrones que se pueden dar en la generación de carreteras (lo cual constituye un factor primordial en la generación de ciudades), y estos patrones son producto de numerosos factores. Las poblaciones pueden contener un gran número de patrones prevalecieres en distintas zonas, lo que las hace objetos anárquicos en cierta medida y por tanto difíciles de modelar.

En este apartado se hará una revisión de los modelos de generación de ciudades siguiendo el estudio realizado por George Kelly y Hugh McCabe en [52], donde se hace una evaluación de los resultados obtenidos por los distintos métodos en base al realismo, escala, variabilidad, entrada necesaria, eficiencia, control y aplicabilidad en tiempo real.

Los modelos son los siguientes:

Diseño Grid y Primitivas Geométricas :

Este método presentado por Stefan Greuter en [53][54] propone una solución para generación de ciudades en tiempo real, la cual se aplicó en una aplicación llamada *Undiscovered City* donde se crea una red de carreteras usando un diseño de grid sobre un el cual se posicionan las construcciones generadas usando una combinación de primitivas geométricas. Las carreteras son creadas en base a un tamaño regular de los bloques que serán las parcelas donde se construirán lo edificios, el cual se puede ajustar de manera global. En cuanto a la generación de construcciones, se crea la geometría usando el concepto de combinación de primitivas geométricas para formar las secciones de los edificios.

En cuanto a la evaluación de este modelo, los resultados producen patrones muy regulares de edificios dada su distribución en diseño de gird dando una apariencia muy homogénea, que sumado al aspecto que dan los edificios, los resultados se podrían considerar realistas pero no convincentes. En cuanto a escala, este sistema no parece tener limitación para la generación más allá de la limitación del tamaño de los números enteros para la generación de coordenadas de los bloques. A pesar de que la geometría para cada construcción es diferente, la cantidad de variación que se produce es insuficiente para emular ciudades reales. La aplicación mencionada que se realizó usando este método es standalone, por lo que no se necesitan datos de entrada. Aunque los tamaños de bloque se pueden ajustar, la generación no es interactiva y se realiza de manera automática. Por último, en cuanto a su aplicación en tiempo real, este sistema está diseñado par aplicaciones en tiempo real, por lo que se pueden renderizar grandes ciudades en hardware de equipos comunes a buenas tasas de refresco gracias a optimizaciones como el cacheado de geometría y el frustrum culling.

Sistemas L:

Yoav Parish y Pascal Muller en su artículo "Procedural Modeling of Cities"[55] presentaron su aplicación, de la cual ya se habló en las herramientas existentes relacionadas con este trabajo, por lo que sigue estando vigente. La aplicación que presentaron es CityEn-

gine, que consiste en un conjunto de componentes, incluyendo generación de carreteras, construcción de edificios y fachadas que se unen para formar una pipeline para generación de ciudades. Los Sistemas L son la técnica clave en torno a la que gira la generación procedural de este modelo.

CityEngine usa una forma extendida de los Sistemas L llamados *Auto-sensitivos-L* para construir redes de carreteras de manera que se tenga en cuenta el crecimiento existente para su generación. La generación de carreteras se lleva a cabo usando dos tipos de reglas: *Objetivos Globales*, que definen los tipos de carreteras (principales y secundarias), su patrón geométrico y el criterio de elevación que han de seguir, y *Restricciones Locales* las cuales determinan que las carreteras se creen en áreas legales en base a condiciones.

Las construcciones usando Sistemas L como el que se proponen se generan en diferentes fases: definir emplazamiento de las construcciones, crear la geometría de los edificios y generar la textura de las fachadas, usando diferentes Sistemas L que permiten generar diferentes tipos de edificios, los cuales son determinados por la zona del mapa.

En la discusión de este modelo se determina que los Sistemas L proveen un método efectivo para la generación realista de paisajes urbanos con áreas verdes incluidas, a pesar de que las construcciones resultan ser un poco básicas. En cuanto a entradas de datos, se requiere como mínimo un mapa geográfico con información sobre elevación, vegetación y los límites acuáticos, por lo que el tamaño de este input es el principal factor que puede limitar el tamaño de las ciudades generadas. Se pueden generar un rango notable de redes de carreteras con distintos patrones y también variedad de edificios, aunque en un rango algo limitado. En cuanto al control, parece estar sujeto a la cantidad de imágenes de mapas que puedan ser pasados como datos de entrada. Aunque la generación es eficiente, pudiendo crear grandes redes de carreteras en menos de dies segundos, no permite la generación en tiempo real dado que no implementa técnicas de optimización como culling de geometría o LOD.

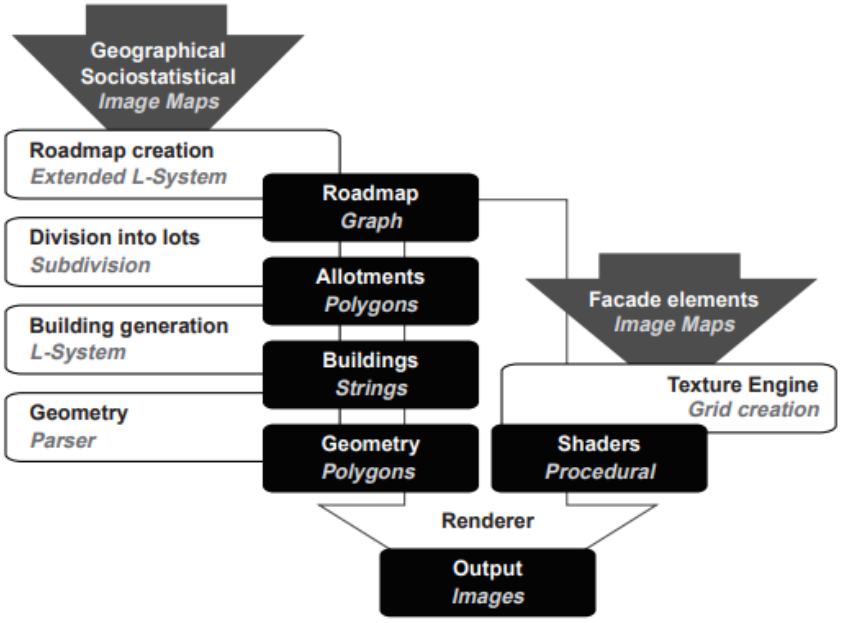

Figura 4.13: Diagrama de la estructura del algoritmo para CityEngine obtenido de [55]

Simulación basada en Agentes:

Este método nombrado anteriormente -al igual que los Sistemas L- en la generación procedural de terrenos también tiene aplicación para la generación de ciudades. En [56] se presenta una solución llamada *CityBuilder* que simula ciudades usando agentes como desarrolladores, autoridades planificadoras y constructores de carreteras. Además de simular la generación de la ciudad, este modelo también simula el crecimiento y desarrollo de la misma a lo largo del tiempo.

Para la generación de carreteras se definen dos tipos de agentes: *Extendedores*, los cuales buscan extender la red de carreteras acorde a la densidad de carreteras desde la que se parte, proximidad a intersecciones y desviación del punto de salida. Y los *Conectores*, quienes chequean la distancia para ir de un punto a otro dentro de un radio para crear nuevas redes en caso de que la distancia sea mayor de un umbral. Las carreteras son creadas acorde a un patrón de grid, aunque la desviación de este parámetro puede especificada mediante parámetros.

En la generación de edificios se realiza la interacción de varios tipos de agentes, pero son principalmente los llamados *Desarrolladores*, quienes cumplen el rol de desarrolladores urbanos obteniendo permisos de planificación, comprando y vendiendo parcelas de tierra. Existen distintos tipos de desarrolladores definidos: residenciales, comerciales e industriales. En este sistema se crean carreteras y se define el uso de las parcelas mediante la interacción de los agentes mencionados, lo que se usa para determinar el tipo de edificios que ocuparán las parcelas, pero no se generan ni geometría de edificios ni texturas.

En la evaluación de ste método se puede decir que a pear de no producir construcciones los mapas generados parecen realistas, aunque la escala de estos es limitada, siendo similar a la de una villa o pueblo pequeño, pero no ciudades. Permite variedad de zonas, pudiendo ser estas residenciales, comerciales o industriales, y se necesitan mapas de altura del terrenos y el nivel del mar como entradas mínimas para determinar las áreas legales donde los edificios serán emplazados. Además incorpora una GUI que permite modificar parámetros de manera interactiva con un sistema de painting . Los aspectos negativos de este método son que es un algoritmo computacionalmente intensivo y consume mucho tiempo teniendo en cuenta la escala de los resultados que produce, además de que no hay si quiera consideraciones de ejecución de este método en tiempo real.

Generación basada en Plantillas:

Este es un método alternativo a los anteriores propuesto en el artículo [57] donde la idea básica es generar un sistema donde una plantilla de red de carreteras es aplicada sobre un mapa geográfico como un plano y las carreteras son moldeadas siguiendo restricciones locales.

En este modelo la red de carreteras es representativa de la distribución de población, la cual viene dada por una plantilla basada en población implementada usando diagramas de Voronoi, donde los puntos de densidad poblacional son la entrada para generar dichos diagramas. Existen varias plantillas que sirven como patrones de crecimiento de la red de carreteras, que son las plantillas: *Modo raster, Modo Radial y Modo Mixto*. Estas plantillas definen únicamente el patrón ideal de la red, pero estas se adaptan siguiendo restricciones locales que hacen que se eviten obstáculos como zonas de agua y de alta elevación.

Con este método se reflejan patrones encontrados en ciudades, pero no alcanzan la

complejidad ni escala de las redes reales de ciudades. Aunque se pueden combinar patrones, solo se pueden combinar dos, lo que resulta insuficiente. Las plantillas de población y de crecimiento de carreteras y sus deformaciones ne base a las restricciones ofrecen variabilidad limitada. Al ser un método basado en plantillas requiere varias entradas, como mapas de alturas, geográficos y de densidad de población.

El método no ofrece control dado que se basa en información estática y no hay interacción con el usuario. No hay información sobre la eficiencia del método y por tanto tampoco sobre su aplicación en tiempo real.

Figura 4.14: Plantilla basada en población (arriba izq.), Modo Radial (arriba der.), Modo Raster (abajo izq.), Modo Mixto (abajo der.).

Gramáticas de separación:

El método que se discute a continuación es el presentado en [58], donde se propone la generación realista de edificaciones a través de un tipo de gramática llamada *gramática de separación*, las cuales están basadas en el concepto de forma. Aunque este es un método de generación de construcciones, no de ciudades, y dado que este TFM va a ser desarrollado en Unity, que es un motor de videojuegos, no de modelado, no tiene aplicación. No obstante, es una técnica empleada en la generación de ciudades a nivel general y se ha creído conveniente nombrarla como parte del estado del arte de este campo, aunque no se analizará tan en profundad como los anteriores métodos.

Este método genera construcciones usando objetos llamados *formas básicas*, las cuales son parametrizadas, etiquetadas y dotadas de atributos simples. Se parte de un estado inicial y se realizan transformaciones en un proceso iterativo usando reglas de una base de datos, las cuales separan las construcciones en fachadas, las fachadas en secciones estructurales y estas secciones en componentes como ventanas y demás. Los atributos de las formas desde las que se parte son propagados a los componentes y gramáticas de control pueden cambiar las propiedades de las formas básicas para aplicar diferentes conceptos de diseño espacial.

Este método produce edificios muy realistas y con gran variedad que incluso permite recrear estilos arquitectónicos. El numero de edificios generables con este sistema es

limitado y no llega a la escala de una ciudad. Se requiere una entrada significativa -una base de datos con reglas y atributos que modelen los edificios. El algoritmo, a pesar de ser complejo es eficiente permitiendo incluso la exploración en tiempo real, no obstante el número de edificios mostrables al mismo tiempo es limitado. También ofrece control a través de la modificación de las reglas directamente de la base de datos.

Generación Procedural de Poblados en Tres Pasos:

Emilien en su artículo "Procedural Generation of Small Cities in Three Steps"[59] propone un método para generar villas o pueblo pequeños de tres pasos que recrea los pequeños asentamientos europeos que utilizan las características del terreno. El método consta de tres pasos: el primero es un proceso iterativo basado en mapas de interés para generar semillas de asentamiento y los caminos que los conectan, teniendo en cuenta el hecho de que un nuevo camino atrae a nuevos asentamientos y nuevos asentamientos conduce a una extensión de los caminos. El segundo paso es un método de conquista anisotrópico que segmenta la tierra en parcelas alrededor de las semillas de asentamiento. Finalmente se introduce una *gramática de forma abierta (open shape grammar)* para generar geometría 3D que se adapte a la pendiente local y terreno local. Esto se consigue creando primero los pisos y el techo y después añadiendo elementos de la fachada. Las *Open Shape Grammars* incorporan mecanismos de adaptación mediante restricciones como la no colisión con el suelo, alienación de elementos y demás reglas.

Este método permite crear de manera poblados con formas realistas comparadas con las de las formas de las villas reales, con un número de vecinos y de esquinas similares, además de una generación realista de edificios gracias al método basado en *Open Shape Grammars*. Solo requiere la información del terreno como mapa de alturas y cuerpo de agua como entrada para la generación y permite el control a través del suo de mapas de interés para crear las semillas de asentamiento y las reglas para generación de construcciones. La escala de los resultados no excede el tamaño de pequeñas villas como se indica en su objetivo y no hay datos del rendimiento, por lo que no se puede evaluar correctamente su uso en aplicaciones de tiempo real.

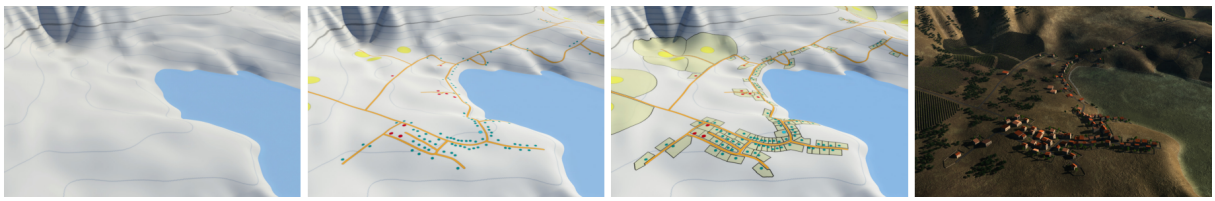

Figura 4.15: Muestra del funcionamiento del método: dado un terreno se genera el esqueleto del poblado (caminos and semillas de asentamiento), después se añaden las parcelas y finalmente las geometrías de las casas.

4.4.3. Métodos Complementarios

En este apartado se discutirá sobre métodos que ayudan en la generación de los elementos troncales de este trabajo pero que no son temas principales. Aún así estas técnicas pueden jugar un papel crucial en el realismo y coherencia visual, por lo que son de importante relevancia y su rendimiento también juega un papel clave, ya que pueden procesos fundamentales como la instanciación de objetos y la corrección de bordes

Métodos para Simplificar la Geometría:

Dado que se deberá optimizar el rendimiento lo máximo posible, la reducción de geometría es un paso indispensable debido al ahorro en cálculo que supone. Existen varios métodos para simplificar la geometría de los objetos de una escena:

- **Colapso de bordes:** El colapso de bordes implica la fusión de vértices adyacentes, lo que reduce el número de bordes en la malla. Este método es preferible en modelos geométricos [60].
- **Colapso de aristas:** El colapso de arista es un método similar al anterior, pero en este caso se elimina una arista manteniendo la cantidad de vértices. Este método es preferible en modelos orgánicos [60].
- **Agrupación en clústeres de vértices:** La agrupación en clústeres de vértices divide la malla en celdas y reemplaza cada celda con un vértice central o promedio, lo que simplifica la estructura [60]. Rossignac y Borrel proponen un método en [61] que consiste en colocar una caja delimitadora alrededor del modelo original y dividirla en una cuadrícula, agrupando los vértices que caigan dentro de la misma celda y actualizando las caras del modelo. Aunque es un método efectivo para la simplificación, la calidad de los resultados parece ser baja y no tampoco permite un gran control sobre la cantidad de cara resultantes.
- **Simplificación basada en curvatura:** Este método utiliza la curvatura de la superficie para identificar y conservar las características importantes de la geometría, permitiendo así realizar simplificaciones conservando la mayor fidelidad y las características de la original. En [62] se describe un método para el remallado de rostros humanos que se basa en alinear los bordes de la superficie de la malla utilizando análisis del tensor de curvatura de esta para ajustarla de manera anisotrópica. Uno de los algoritmos que corresponde con este tipo de métodos es el algoritmo de Ramer-Douglas-Peucker (RDP) para reducir el número de puntos utilizados en la aproximación de una curva. La forma inicial del algoritmo fue propuesta independientemente en 1972 por Urs Ramer, en 1973 por David Douglas y Thomas Peucker, y algunos más en la siguiente década. Este algoritmo también es conocido con el nombre de algoritmo de Douglas-Peucker [63].
- **Decimación de vértices:** Estos métodos es otro método de reducción de vértices y aristas de la malla. En el método presentado en [64] se describe un algoritmo que selecciona de manera iterativa un vértice para eliminar, elimina todas las caras adyacentes y vuelve a triangular el agujero resultante.
- **Métrica de error cuadrático:** La métrica de error cuadrático es el método que se utiliza para minimizar la desviación entre la malla simplificada y la original, colapsando aristas o vértices teniendo en cuenta la orientación de la cara usando [60].

En este subapartado cabe mencionar los trabajos de Jain, Aryamaan y Sharma en su artículo "Learning Based Infinite Terrain Generation with Level of Detailing"[65] donde proponen un framework generativo basado en aprendizaje para generar terrains infinitos, con un enfoque novedoso también basado en aprendizaje para terrenos con nivel de detalle, el cual integra un método de renderizado basado en quadtrees y completado de detalles con imágenes para terrenos con super resolución.

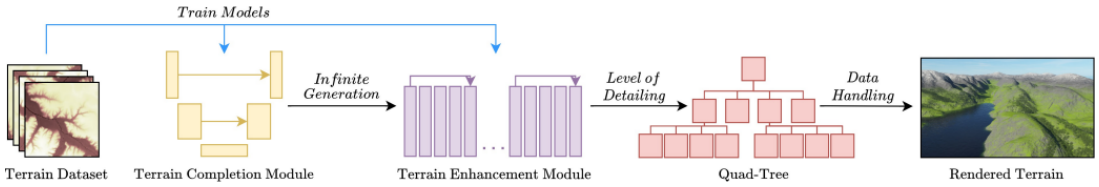

Figura 4.16: Descripción general del framework presentado en "Learning Based Infinite Terrain Generation with Level of Detailing"[65]

También el de Jian Wu, Yuanfeng Yang, Shengrong Gong y Zhiming Cui en su artículo .ª New Quadtree-based Terrain LOD Algorithm"[66], donde se expone un nuevo algoritmo de LOD de terreno que utiliza un quadtree, el cual puede satisfacer la necesidad de renderizado en tiempo real para datos de terreno masivos, y un nuevo algoritmo de eliminación de grietas.

Modelos de Distribución de Objetos:

- **Basados en Puntos de Interés:** Estos modelos toman puntos concretos y centran la mayor distribución de objetos entorno a ellos. Estos puntos de interés pueden ser determinados por mapas sirviendo de entrada para el sistema de distribución. También se pueden determinar los puntos en base a ciertos criterios o características del entorno donde se distribuyan. En [55] se hace uso de mapas de distribución de población extrayendo de estos los puntos para generar diagramas de Voronoi con los que se producirá la generación de edificios.

- **Basados en Probabilística:** Una de las maneras más comunes de hacer distribuciones de objetos es siguiendo distribuciones probabilísticas. La distribución uniforme o la de Poisson son las más utilizadas. Existen diversos artículos como [42], [67] o [68] donde se emplea el muestreo basado en alguna de estas distribuciones para la instanciación de activos en generación procedural. En los artículos mencionados se usa concretamente el muestre de disco de Poisson, donde se busca que haya la mayor distancia posible entre un par de objetos evitando agrupaciones pero produciendo una distribución menos regular que la uniforme. El término "disco"se refiere a las formas que se utilizan para definir la separación mínima entre los objetos.

- **Basados en Gramáticas o Reglas:** De nuevo, esta técnica vuelve a aparecer, esta vez como modelo de distribución. En [42] se combina la distribución de disco de Poisson, con el uso de reglas para modelar qué especies deben aparecer más frecuentemente o más cerca unas de otras.

- **Basados en Análisis de Terreno:** Algunos sistemas utilizan permiten distribuir objetos de acuerdo a las características topográficas del terreno. Por ejemplo en el paper "Procedural Generation of Villages on Arbitrary Terrains"[59], los objetos son distribuidos de acuerdo a las características del terreno.

Métodos de Texturizado:

En generación procedural es importante tener en cuenta el realismo del terreno generado (en caso de que se pretenda realismo) no solo en base a la estructura y topografía del terreno generado, sino en los colores y aspecto que presenta. Para ello, las técnicas

de texturización son una parte importante a tener en cuenta. Aquí se presentan algunas técnicas habitualmente utilizadas [69].

- **Procedural:** Para crear texturas mediante programación utilizando algoritmos y funciones matemáticas. Esta técnica se utiliza a menudo para crear patrones complejos y texturas naturales como nubes, rocas o agua.
- **Basada en Imágenes:** Consiste en aplicar imágenes en modelos 3D para crear detalles de superficie realistas.
- **Pintura :** El uso de software de pintura digital se puede utilizar para crear texturas estilizadas o artísticas.
- **Stenciling:** Al aplicar texturas selectivamente a áreas específicas de un modelo 3D usando máscaras.
- **Mezclado de Texturas:** Para combinar múltiples texturas para crear detalles de superficie complejos como óxido o suciedad. Dentro de esta categoría podríamos encuadrar los **shaders triplanares**, los cuales permiten combinar varias texturas con transiciones suaves entre ellas, y son ampliamente utilizados para generación procedural.
- **Mapeo de Normales:** Simula la apariencia de golpes, abolladuras o cualquier otro tipo de detalle en una superficie plana mediante el uso de una imagen 2D que codifica las normales de superficie.
- **Mapeo de Desplazamiento:** Consiste en agregar detalles geométricos a la superficie de un modelo 3D mediante el uso de una imagen en escala de grises para deformar la geometría de la superficie.
- **Tiling:** Compone texturas de patrones que se pueden repetir sin problemas en una superficie grande como un suelo o una pared.

4.4.4. Optimización y Rendimiento

Técnicas y Algoritmos de Optimización

Dado que la herramienta que se pretende desarrollar debe funcionar en tiempo real, es conveniente implementar técnicas que permitan ahorrar el mayor tiempo de computación y memoria posible con el fin de obtener un rendimiento óptimo. A continuación se presentan varias técnicas ampliamente utilizas para la optimización de código.

- **Culling:** El culling es una técnica ampliamente utilizada en gráficos que consiste en ocultar elementos no visibles en la escena para optimizar la carga de trabajo reduciendo el número de polígonos que procesar. Existen diversos tipos de culling como el *Frustum Culling*, el cual omite todos aquellos objetos que queden fuera del *frustum*, el cual representa el espacio 3D de visión de la cámara, el *Occlusion Culling* que omite aquellos objetos o polígonos que se encuentran no visibles a la cámara aunque sí estén dentro del espacio de visión de esta y otros tipos de culling como el *Camera Culling* que se asocia más con propiedades de la cámara como su ángulo, distancia a esta, etc, más que con el capo de visión.

- **Nivel de detalle:** El nivel de detalle o LOD, es otra técnica de optimización que se utiliza a menudo en gráficos para reducir la complejidad de la geometría de una escena en función de la distancia a al cámara. A mayor distancia a la cámara, menor número de polígonos y por tanto, menor definición tendrán los objetos que implementen esta técnica. Los objetos podrán tener de esta manera diferentes geometrías, cada una de ellas asociada a un LOD distinto que se mostrará en base a la distancia a la cámara.

- **Chunking:** Chunking es una técnica que permite la generación de geometría por lotes en lugar de tener que crearla de una vez. Esta técnica es ampliamente utilizada en generación de terreno procedural, por lo que atañe directamente al tema de este proyecto, y es utilizada ampliamente en juegos como Minecraft para la generación de sus mundos. Con esta técnica se lleva a cabo la carga y descarga dinámica de estos lotes la cual se puede combinar a su vez con otras técnicas de optimización como el LOD, aunque también puede conllevar otras implicaciones que añaden cierta carga computacional como el ajuste de los bordes entre los distintos chunks en algunos casos.

- **Instanciación con GPU:** La creación de instancias con GPU es un método de optimización que genera múltiples copias de una mesh con el mismo material en una única llamada de dibujo. Es útil para dibujar cosas que aparecen varias veces en una escena, por ejemplo, árboles o arbustos. La creación de instancias con GPU genera mallas idénticas. Para agregar variación y reducir la apariencia de repetición, cada instancia puede tener diferentes propiedades, como Color o Escala. Esta técnica es compatible con el sistema de Iluminación Global Baked de Unity y con los shaders estándar de Unity. Al utilizar instanciación en GPU, se mejora significativamente el rendimiento al reducir la carga de la CPU y optimizar la renderización de elementos repetitivos en la escena. [70]

- **Paralelización:** Consiste en realizar la ejecución del código en distintos hilos aprovechando el máximo rendimiento de la CPU al hacer que trabajen todos los núcleos sin saturar el hilo principal. En Unity existe el JobSystem, del cual hablaremos más tarde para la paralelización y ejecución de código paralelo de manera segura. Además de JobSystem también cabe mencionar las corrutinas, la cuales permiten ejecutar código en distintos hilos de manera asíncrona, no interrumpiendo la ejecución del hilo principal.

- **Diseño en capas:** Hacer un diseño de la lógica en capas puede ayudar a reducir la complejidad y mejorar la mantenibilidad y escalabilidad del código. En Unity esto puede llevarse a cabo mediante el uso de prefabs, los cuales pueden derivar unos de otros, heredando la lógica asociada a ellos.

- **Profiling:** El uso de herramientas de manera metódica que permitan la detección de cuellos de botella, puntos de ajuste o de optimización también es crucial para poder obtener un rendimiento óptimo. En Unity existen diversas herramientas que permiten el ajuste de las partes que afecten o ralenticen la ejecución:

    - **Profiler:** Esta herramienta está integrada en el propio editor de Unity y provee información valiosa acerca del rendimiento de los componentes que intervienen en la ejecución en CPU. Se usa para hacer diagnósticos de los problemas de rendimiento y optimización.

- **Graphics Debugger:** Esta herramienta depura la aplicación y permite analizar el rendimiento en la GPU para detectar posibles problemas en el renderizado o los shaders.
- **Unity Remote:** Con esta herramienta se pueden probar y depurar las aplicaciones directamente en dispositivos para optimizar el rendimiento en dispositivos como móviles o tablets, inclusive gafas de RV.

4.4.5. Optimizaciones para la Compatibilidad con Realidad Virtual

Para el desarrollo en Realidad Virtual, hay ciertas consideraciones adicionales que se deben tener en cuanta para una experiencia óptima en términos de rendimiento e inmersividad. A continuación se detallan algunas de ellas.

Optmizaciones de Rendimeinto Específicas

Ya se han nombrado varias técnicas para la mejora del rendimiento, aún así, caben destacar algunas más específicas especialmente importante ciado se trata de aplicaciones en RV. [71] [72]

- **Optimización de assets:** Dado que en realidad virtual se requiere de una renderización doble de la escena para cada visor, las escenas deben tener una cantidad de polígonos muy cuidado, y texturas optimizadas, de lo contrario se verá afectado el frame rate.
- **Minimización de efectos de post-procesado:** De nuevo, la doble renderización es la causante de la necesidad de minimizar los efectos de post-procesado. Debido a la alta carga de trabajo que ya presenta esta tarea para la tarjeta gráfica, los efectos aumentarían dicha carga pudiendo ralentizar la tasa de refresco.
- **Tipos de iluminación:** Las luces estáticas y Mapas de Luces para proyectos de RV son la opción preferible debido a que es la opción de iluminación más económica de renderizar. Si se necesita usar iluminación dinámica, se debe limitar la cantidad de luces dinámicas a lo mínimo posible. Esta técnica es utilizable para cualquier tipo de aplicación, al igual que las anteriores descritas, pero en RV cobran especial importancia dado su impacto en el rendimiento.
- **Efectos visuales (VFX):** Algunos trucos de VFX, como usar Texturas SubUV para simular fuego o humo, no se mantienen muy bien cuando se ven en RV. En muchos casos, se necesitará usar mallas estáticas en lugar de partículas 2D para simular efectos como explosiones o estelas de humo. Los efectos de campo cercano, o efectos que ocurren muy cerca de la cámara, funcionan bien en RV, pero solo cuando están compuestos de partículas de meshes estáticas.
- **Evitar la transparencia:** En gráficos 3D, renderizar la transparencia es extremadamente costoso, porque la transparencia generalmente debe reevaluarse por frame para asegurarse de que nada haya cambiado. Debido a esta reevaluación, renderizar la transparencia en RV puede ser tan costoso que su costo supera sus beneficios.

Estrategias para mejorar la Inmersividad

- **Mecánicas de locomoción adecuadas:** Elegir mecánicas que sean cómodas e intuitivas para el usuario, considerando las experiencias físicas de la RV.
- **Escala del entorno:** Mantener la consistencia en las métricas de los elementos del entorno es fundamental para tener una experiencia inmersiva, ya que los usuarios de RV perciben la escala con más precisión que en entornos de pantalla plana.
- **Uso de audio:** Con sonido ambiental adecuado se puede mejorar la experiencia inmersiva mejorando la sensación de presencia en el entorno que se genere.
- **Limitar el mareo por movimiento:** El mareo por movimiento que puede afectar a los usuarios durante experiencias inmersivas. La siguiente lista describe algunas buenas prácticas para limitar el malestar que los usuarios pueden experimentar en la realidad virtual:
  - *Mantener la frecuencia de refresco:* Los bajos fps pueden causar mareos, por lo que es fundamental conseguir un buen frame rate.
  - *Mantener a los usuarios en control de la cámara:* Las cámaras cinematográficas, o cualquier cosa que tome el control de los movimientos de la cámara lejos del jugador, contribuyen al malestar del usuario en experiencias inmersivas. Se deben evitar efectos de cámara, como el movimiento de la cabeza y el temblor de la cámara, ya que pueden provocar malestar si el usuario no los está controlando.
  - *El FOV debe coincidir con el dispositivo:* El valor del FOV (field of view o campo de visión) se establece a través del SDK del dispositivo y la configuración interna, y coincide con la geometría física del casco y las lentes. Por esta razón, el FOV no debe ser modificable por el usuario. Si se cambia el valor del FOV, el mundo puede parecer distorsionado al girar la cabeza, lo que puede provocar mareo.
  - *Luces y colores más tenues, y evitar el desenfoque:* Al diseñar elementos para RV, es posible que necesites usar luces y colores más tenues de lo normal. La iluminación fuerte y vibrante en RV puede provocar desorientaciones. El uso de tonos más fríos y luces más tenues puede ayudar a prevenir el malestar del usuario. Esto también ayuda a prevenir el desenfoque entre áreas brillantes y oscuras en la pantalla.
  - *La velocidad de movimiento no debe cambiar:* Los usuarios deben comenzar a toda velocidad en lugar de acelerar gradualmente hasta la velocidad máxima.
  - *Evitar los efectos de post-procesado que afecten significativamente lo que ve el usuario:* Evita los efectos posteriores como la Profundidad de Campo y el Motion Blur para prevenir el malestar del usuario.

Limitaciones Debido a la Realidad Virtual

Dada las características de la realidad virtual, se deben tener en cuenta ciertas técnicas que podrían no ser efectivas ni tener los resultados esperados cuando se utilizan en un dispositivo para RV. Aquí se presentan algunas limitaciones: [73]

- **Iluminación global en espacio de pantalla:** Las técnicas en espacio de pantalla pueden generar diferencias entre las pantallas para los dos ojos en el HMD. Estas diferencias pueden causar molestias al usuario. Además de esto, la iluminación por píxel puede producir algunos artefactos que serán más evidentes en RV y también puede no tener en cuenta correctamente la profundidad de los elementos a la hora de iluminar, lo que puede dar una sensación menos inmersiva.
- **Ray Tracing:** Las aplicaciones de RV que utilizan trazado de rayos actualmente no pueden mantener la resolución y los fps necesarios para una experiencia de RV cómoda.
- **Interfaz de usuario 2D o Sprites tipo Billboard:** La interfaz de usuario 2D o los sprites tipo billboard no son compatibles con el renderizado estéreo ya que no se ven bien.
- **Mapeo de normales:** Cuando se ven mapas de normales en objetos en RV no tienen el mismo efecto dado que el mapeo normal no tiene en cuenta tener una pantalla binocular o paralaje de movimiento. Como resultado, los mapas normales a menudo se ven planos cuando se ven con un dispositivo RV. Es más conveniente usar mapas de paralaje en su lugar, aunque son específicamente recomendables para usar en caminos empedrados o superficies con detalles finos.

DOTS Unity

En esta apartado se hablará sobre el el sistema DOTS de Unity compuesto por el Job System junto con el Burst Compiler y ECS (Entity Component System). Se comentarán los tipos de Jobs que hay y se analizará cuáles pueden ser los más propicios para la generación procedural.

- **JobSystem:** El JobSystem, es un sistema que permite crear código multihilo de manera segura aprovechando todos los núcleos, haciendo que mejore el rendimiento de manera considerable. A continuación se describen los tipos de Jobs que existen [74]:
  - **IJob:** Ejecuta una única tarea en un hilo, en un job thread. Se utiliza este tipo de Job cuando no se necesita ejecución en paralelo.
  - **IJobParallelFor:** Ejecuta una tarea en paralelo. Cada job thread que se ejecuta en paralelo tiene un índice exclusivo para acceder de forma segura a los datos compartidos entre job threads. Es útil por cuando se pretende ejecutar una tarea en paralelo a un conjunto de datos, como por ejemplo, aplicar la misma operación a múltiples elementos.
  - **IJobParallelForTransform:** Ejecuta una tarea en paralelo. Cada job thread que se ejecuta en paralelo tiene un nodo transform exclusivo de la jerarquía de transformaciones en el que operar. Este job es útil cuando se necesita, por ejemplo, actualizar la posición de varios objetos en paralelo.
  - **IJobFor:** Igual que IJobParallelFor, pero permite programar el job para que no se ejecute en paralelo.
- **Burst Compiler:** El burst compiler es un compilador de Unity que traduce el código IL/.NET bytecode a código nativo altamente optimizado usando LLVM. Se

puede utilizar como un paquete de unity e integrarlo a través del Package Manager como cualquier otro paquete.[75]

- **ECS:** ECS para Unity (Entity Component System) es un marco orientado a datos compatible con GameObjects. Se podría considerar como un nuevo paradigma de programación orientado a datos que consta de tres partes principales: [76]
  - **Entidades:** las entidades o cosas que pueblan el programa.
  - **Componentes:** los datos asociados con sus entidades, pero organizados por los datos en sí y no por entidad.
  - **Sistemas:** la lógica que transforma los datos del componente de su estado actual a su siguiente estado.

El JobSystem de Unity se puede combinar con el ECS y el Burst Compiler haciendo que el ECS reduzca el tiempo de cálculo de cada subproceso organizando los datos de manera más compatible con la caché y que el Burst optimice el código para reducir aún más el tiempo de trabajo, lo que permite aumentar el rendimiento enormemente. [77]

4.5. Análisis de la investigación

4.5.1. Aspectos Clave Abordados en el Estado del Arte

En el estado del arte se ha realizado principalmente el estudio y análisis de las herramientas previas y antecedentes con fines similares los que se pretenden llevar a cabo en este TFM. Se han estudiado softwares de Unity e independientes, incluso con compatibilidad con otros motores que realizan generación de terrenos de manera procedural con simulación de biomas e incluso de fenómenos atmosféricos con un alto grado de realismo, junto con softwares para la generación de poblaciones y asentamientos.

Tras el estudio de los antecedentes se hizo una revisión de las técnicas más utilizadas y novedosas empleadas para la generación de terreno procedural en base a diversos estudios, artículos y sitios web.

Desde la generación de los propios terrenos con diversos métodos a las técnicas de optimización que se pueden llevar a cabo para mejorar el rendimiento de las aplicación que los implementen (incluido el sistema DOTS de Unity, del cual se hará uso en esta propuesta) y otros métodos complementarios involucrados en las técnicas y métodos más generales, pero necesarios para obtener buenos resultados.

En este estado del arte también se ha estudiado los distintos métodos para generación de ciudades y poblaciones, así como los aspectos relacionados con la compatibilidad con la realidad virtual, sus limitaciones y factores a tener en cuenta relacionados con las generaciones que se pretenden llevar a cabo.

4.5.2. Conclusiones

Del estudio realizado se pueden sacar diversas conclusiones. Comenzando por los métodos de generación de terreno, se han estudiado diferentes tipos, pero solo algunos son opciones reales para esta propuesta dadas las limitaciones derivadas de ser en tiempo real

y compatible con realidad virtual. Las técnicas que se podrán tener más en cuenta son aquellas que permiten la generación paralela.

Los métodos basados en simulación para generar mapas de alturas tienen un coste computacional demasiado elevado en la mayoría de los casos para permitir una buena compatibilidad con la generación en tiempo real y la RV. Los métodos basados en aprendizaje requerirían de un entrenamiento previo de las IAs que trabajarían para generar el terreno, y otros algoritmos como el Wave Function Collapse no dan un resultado tan realista como otros métodos. En el caso de los generados por bocetos, los resultados que producen no se ajustan a los resultados buscados en esta propuesta y la metodología que siguen presenta dificultades para implementarla de manera paralela. Los métodos estocásticos son la propuesta más valorarle para ser implementada dado su bajo coste computacional y su fácil paralelización. En concreto, los métodos fractales son una solución ampliamente utilizada, siendo la opción elegida en la mayoría de métodos estudiados para la generación del terreno.

Tanto los métodos fractales basado en ruido como en los algoritmos recursivos de diamante-cuadrado o desplazamiento del punto medio son opciones válidas siendo muy habituales en los softwares y papers estudiados anteriormente, como se ha mencionado ya. En cuanto a la generación de biomas, se ha discutido sobre el coste de los algoritmos de simulación en tiempo real, no obstante, el paper "AutoBiomes: procedural generation of multi-biome landscapes"[42], el cual se ha mencionado en el estado del arte en varias ocasiones, presenta un método apto para la generación en tiempo real, empleando tenencias de simulación en una versión simplificada abaratando su coste. Aunque el método que proponen está pensado para ser utilizado en tiempo real, que da por ver si será compatible con RV y se podrá ofrecer un buen rendimiento. Aun así, este paper ofrece un enfoque similar al que se pretende llevar a cabo en este trabajo, por lo que se tomará muy en cuenta su modelo para servir como base del que se desarrolle finalmente y se tendrán muy en cuenta las tenencias utilizadas dado que ya es una solución probada con fines parecidos y resultados satisfactorios para sus desarrolladores.

Para la generación de poblaciones, se han estudiado varios métodos y sólo un par parecen ser los más válidos para esta propuesta. El método basado en diseño de grid y primitivas geométricas es una opción compatible con generación en tiempo real, aunque los resultados que ofrece parecen ser demasiado regulares, no obstante, estos aunque no son convincentes, permiten generar ciudades de manera realista, pero presenta el inconveniente de que conlleva el modelado de edificios, y Unity no permite el modelo de geometría. El método basado en tres pasos es un modelo que también es compatible con la ejecución en tiempo real, pero de nuevo presenta el problema del modelado, por lo que se deberá buscar una alternativa para la generación de construcciones que sea compatible con las capacidades de Unity.

Se ha hablado de métodos complementarios para la simplificación de geometría, distribución de objetos y texturizado. Todos estos procesos se deberían llevar a cabo cuando se implementen los métodos de generación principales del terreno, biomas y poblaciones. De los métodos de simplificación de geometría el método que más se ajusta dada las limitaciones para tratar con geometría es la decimación de vértices, donde se deberá llevar a cabo una omisión de ciertos vértices sin modificar la geometría. En cuanto a la distribución de objetos, todas las técnicas son candidatas y se pueden combinar, aunque como se ha dicho antes, las técnicas empleadas en Autobiomes pueden tener cierta preferencia, que en este caso se trataría de la distribución de disco de Poisson. Para la texturización, se contemplan el uso de texturas generadas proceduralmente y el mezclado de varias texturas,

mediante shaders triplanares, dado que como se mencionó, es una técnica ya empleada en generación de terrenos procedurales.

Por último se ha hablado de diversas técnicas de optimización y de factores clave a tener en cuenta para RV. En este aspecto se tratarán de implementar el mayor número posible de estas para mejorar el rendimiento al máximo. Técnicas como el culling, LOD, paralelización o chunking deberán ser puestas en práctica de manera obligatoria, dado que como se vio en las limitaciones provocadas por la compatibilidad con RV, una de las causas de mareo es debido a bajas tasas de refresco, por lo que la optimización queda en un total primer plano. Otras de las limitaciones de la RV como es la doble renderización y la sobrecarga en la GPU que producen los efectos de post-procesado debido a este factor, es un motivate para dejar fuera del proyecto recreación de fenómenos atmosféricos y otro tipo de recursos que hagan uso de la tarjeta gráfica y puedan ocasionar latencias.

Capítulo 5

Desarrollo de la solución

5.1. Análisis

En esta sección se realizará una planificación temporal detallada del proyecto, teniendo en cuenta las actividades necesarias para su desarrollo y los plazos asociados.

5.1.1. Análisis de Requisitos

Requisitos Funcionales

1. **Generación de terreno infinito:** El sistema debe ser capaz de generar chunks ilimitados de terreno de manera procedural en tiempo real, permitiendo a los usuarios especificar los parámetros como altura del terreno, escala de frecuencia, nivel del mar, etc.

2. **Generación de biomas:** El terreno generado debe tener regiones diferenciadas que representen distintos ecosistemas.

3. **Personalización de biomas:** Permitir al usuario personalizar los biomas generados, como la distribución de la vegetación, tipos de vegetación, texturas, etc.

4. **Generación de poblaciones:** Sobre el terreno deberán aparecer poblaciones en base a las características del terreno, siendo estas mayores o menores según las características del terreno donde se generen.

5. **Parametrización de las generaciones:** Tanto el terreno, como los biomas y poblaciones deben ser controlables por el usuario mediante parámetros en mayor o menor grado.

6. **Distribución de elementos:** A la hora de generar biomas se deben distribuir elementos ambientales como vegetación, rocas u otros objetos ambientales.

7. **Detección de colisiones:** El sistema deberá implementar colisiones para que el usuario pueda moverse sobre la superficie del terreno sin atravesar el suelo o las pendientes gracias al sistema de físicas de Unity y su detección de colisiones.

8. **Texturización:** El terreno debe ser texturizado con el mezclado de varias texturas.

9. **Texturas configurables:** Al igual que los algoritmos de generación, las texturas también han de poder ser configuradas por el usuario, permitiendo establecer qué aspecto se quiere en cada nivel de altura del terreno.

10. **Optimización mediante DOTS:** El sistema debe estar optimizado usando el sistema DOTS de Unity.

11. **Técnicas de optimización:** Además del sistema DOTS, se deben implementar otras técnicas de optimización como culling y LOD.

12. **Sistema de sonido:** El sistema debe añadir sonidos adaptados al entorno donde se encuentre el usuario.

13. **Exportación de terreno:** Proporcionar la capacidad de exportar el terreno generado en un formato compatible con otros programas o motores de juego.

14. **Integración de assets externos:** Permitir la integración de assets externos, como modelos 3D de edificios, plantas o rocas, texturas o sonidos.

Requisitos No Funcionales

1. **Continuidad del terreno:** El terreno no debe presentar discontinuidades y debe generar chunks de terreno sin que haya una transición notable de uno a otro.

2. **Distribución de elementos no uniforme:** En la naturaleza es raro que las plantas y otros elementos del paisaje se distribuyan de manera uniforme, por lo que se evitará esta distribución.

3. **Realismo visual:** El terreno generado debe ser realista, incluyendo detalles naturales y accidentes geológicos.

4. **Usabilidad:** El sistema debe ser intuitivo, con parámetros nombrados de manera que no causen confusión en el usuario y expresen de manera clara su función.

5. **Rendimiento:** Se debe ser capaz de generar terrenos en tiempo real con una buena tasa de refresco.

6. **Compatibilidad con RV:** El sistema debe permitir generar terrenos en RV proporcionando una buena experience inmersiva.

7. **Control de mareos:** El sistema no debe propiciar la aparición de mareos debido a su mal funcionamiento cuando se use en RV.

8. **Control de vértices y polígonos:** Los assets que se integren deberán cumplir ciertas condiciones para que no entorpezcan el rendimiento mínimo del sistema.

9. **Escala del terreno y elementos:** La generación debe ser a escala 1:1 para dar una sensación de inmersividad real.

10. **Intensidad baja del sonido:** Dado que el sonido tendrá fin ambiental, un sonido de alta intensidad distorsionaría la experiencia.

5.1.2. Análisis Temporal

Desglose de Tareas

1. **Estudio preliminar:** Esta tarea consiste en hacer una recopilación de los requisitos y objetivos a cumplir, así como la planificación y estudio de la viabilidad del proyecto del estado del arte referente a este proyecto.

2. **Análisis:** En el análisis se analizarán los requisitos que se exigirán a este proyecto, se estudiarán los componentes y partes del sistema que se implementará, así como la integración que estos tendrán y se analizarán los comportamientos y resultados que se deben producir durante su funcionamiento.

3. **Diseño:** Esta tarea se centra en cómo se va a construir el sistema esclareciendo los componentes y definiendo especificaciones usando para ello diferentes diagramas.

4. **Implementación:** La tarea que consiste en la programación de la propia propuesta y la realización de las pruebas pertinentes, así como sus posibles mejoras y extensiones.

5. **Resultados y conclusiones:** En esta tarea se hará una revisión de la solución desarrollada y sus resultados, analizando las diferentes pruebas que muestran cuáles son sus capacidades, obteniendo conclusiones sobre su desempeño y estableciendo posibles líneas de trabajo futuro.

6. **Documentación:** La documentación es la tarea que recoge las actividades de elaboración de la memoria, la presentación y de video demostrativo, que documenta el proceso de elaboración del proyecto y su funcionamiento.

Estimación de Tiempos

Para el cálculo temporal de cada tarea se realizará un cálculo estimado basado en la siguiente fórmula:

$$te = to + \frac{4tm + tp}{6} \tag{5.1}$$

Donde te es el tiempo estimado, to el tiempo optimista, tp es el tiempo pesimista y tm es el tiempo probable.

Una vez determinados los tiempos estimados del total del proyecto por los expertos, se hará un cálculo ponderado del mismo en base a unos pesos que tendrá cada estimación según la fórmula:

$$te = exp1 \times 0{,}5 + exp2 \times 0{,}35 + exp3 \times 0{,}15 \tag{5.2}$$

Donde $exp1$ es el tiempo estimado calculado con la fórmula 5.1 del experto 1, $exp2$ el tiempo estimado del experto 2 y $exp3$ el tiempo estimado del experto 3.

Los expertos y sus pesos en el cálculo final del tiempo estimado son los siguientes [1]:

-

Una vez calculados los tiempos estimados por los expertos hay que aplicar los pesos para calcular las ponderaciones. El tiempo estimado ponderado queda tal que:

[1] Todos los tiempos serán medidos en días

Tarea	Descripción	T.Optimista	T.Pesimista	T.Estimado
1	Análisis preeliminar			
1.1	Objetivos y metodlogía			
1.2	Planteamiento de problemáticas			
1.3	Estado del arte			
1.4	Conclusiones del análisis preeliminar			
1.5	Aprobación del análisis preeliminar			
2	Análisis preeliminar			
2.1	Determinación de requisitos			
2.2	Análisis temporal			
2.3	Análisis económico			
2.4	Análisis de viabilidad			
2.5	Estudio de casos de uso			
2.6	Modelado de los procesos			
2.7	Aprobación del análisis preeliminar			
3	Diseño			
3.1	Diseño de la arquitectura			
3.2	Diseño de los componentes			
3.3	Verificación del diseño			
4	Implementación y pruebas			
4.1	Programación de la solución			
4.2	Pruebas			
4.3	Aplicación de mejoras			
4.4	Aprobación de la implementación			
5	Resultados y conclusiones			
5.1	Evaluación de resultados			
5.2	Conclusiones de los resultados			
5.3	Análisis de trabajo futuro			
6	Documentación			
6.1	Presentación del trabajo			
6.2	Elaboración de la memoria			
6.3	Vídeo demostrativo			
	TOTAL			

Figura 5.1: Estimación de ..., primer experto.

Tarea	Descripción	T.Optimista	T.Pesimista	T.Estimado
1	Análisis preeliminar			
1.1	Objetivos y metodlogía			
1.2	Planteamiento de problemáticas			
1.3	Estado del arte			
1.4	Conclusiones del análisis preeliminar			
1.5	Aprobación del análisis preeliminar			
2	Análisis preeliminar			
2.1	Determinación de requisitos			
2.2	Análisis temporal			
2.3	Análisis económico			
2.4	Análisis de viabilidad			
2.5	Estudio de casos de uso			
2.6	Modelado de los procesos			
2.7	Aprobación del análisis preeliminar			
3	Diseño			
3.1	Diseño de la arquitectura			
3.2	Diseño de los componentes			
3.3	Verificación del diseño			
4	Implementación y pruebas			
4.1	Programación de la solución			
4.2	Pruebas			
4.3	Aplicación de mejoras			
4.4	Aprobación de la implementación			
5	Resultados y conclusiones			
5.1	Evaluación de resultados			
5.2	Conclusiones de los resultados			
5.3	Análisis de trabajo futuro			
6	Documentación			
6.1	Presentación del trabajo			
6.2	Elaboración de la memoria			
6.3	Vídeo demostrativo			
	TOTAL			

Figura 5.2: Estimación de ..., segundo experto.

Tarea	Descripción	T.Optimista	T.Pesimista	T.Estimado
1	Análisis preeliminar			
1.1	Objetivos y metodlogía			
1.2	Planteamiento de problemáticas			
1.3	Estado del arte			
1.4	Conclusiones del análisis preeliminar			
1.5	Aprobación del análisis preeliminar			
2	Análisis preeliminar			
2.1	Determinación de requisitos			
2.2	Análisis temporal			
2.3	Análisis económico			
2.4	Análisis de viabilidad			
2.5	Estudio de casos de uso			
2.6	Modelado de los procesos			
2.7	Aprobación del análisis preeliminar			
3	Diseño			
3.1	Diseño de la arquitectura			
3.2	Diseño de los componentes			
3.3	Verificación del diseño			
4	Implementación y pruebas			
4.1	Programación de la solución			
4.2	Pruebas			
4.3	Aplicación de mejoras			
4.4	Aprobación de la implementación			
5	Resultados y conclusiones			
5.1	Evaluación de resultados			
5.2	Conclusiones de los resultados			
5.3	Análisis de trabajo futuro			
6	Documentación			
6.1	Presentación del trabajo			
6.2	Elaboración de la memoria			
6.3	Vídeo demostrativo			
	TOTAL			

Figura 5.3: Estimación de ..., tercer experto.

Figura 5.4: Tiempos estimados ponderados.

Además de los tiempos ponderados también se ha decidido calcular otros datos de interés como son la desviación típica σ^2 y la varianza σ para determinar la desviación de los valores respecto del promedio.

$$\sigma^2 = \left(\frac{tp - to}{6}\right)^2 \tag{5.3}$$

$$\sigma = \sqrt{\sigma^2} \tag{5.4}$$

Gracias a estos valores podemos hacer un cálculo de cuál podría ser el tiempo de realización del proyecto en base a las estimaciones de los expertos con determinados niveles de confianza. Calcularemos el tiempo que se tardaría en realizar el proyecto según la estimación de los expertos con un 90 % de confianza y con un 99 % de confianza usando la distribución normal estándar para cálculo de probabilidades de variables aleatorias continuas. La fórmula para el cálculo de probabilidad es la siguiente:

$$Z = \frac{X - \mu}{\sigma} \tag{5.5}$$

Con un 90 % de confianza, mirando en la tabla el valor de Z es 1,2815:

$$1,2815 = \frac{X - 154,20}{57,83}; \quad X = 228,3$$

Con un 99 % de confianza, mirando en la tabla el valor de Z es 2,3263:

$$2,3263 = \frac{X - 154,20}{57,83}; \quad X = 288,73$$

Según los cálculos, con un 90 % de confianza, el proyecto debería terminarse en días y con un porcentaje del 99 %, el proyecto debería terminarse en 288 días como máximo.

Diagrama de Gantt

Para visualizar la estimación temporal que se ha hecho de las tareas se ha realizado un diagrama de Gantt. El diagrama se puede ver en la imagen del anexo.

5.1.3. Análisis Económico

Costes Directos

Costes de hardware En este apartado se calculará el costo del equipo necesario para desarrollar este proyecto, así como su amortización. El material requerido para este proyecto incluye un ordenador portátil con las especificaciones definidas en el apéndice, así como un ratón, ya que es necesario para navegar por el entorno 3D de la interfaz de Unity.

El costo de la computadora portátil es de 1000 euros aproximadamente, con una amortización máxima de 5 años según la Agencia Tributaria [78], el del ratón es de 15 euros y las gafas de RV 500 euros, con un tiempo máximo de amortización de 5 años ambos también al tratarse de útiles, herramientas o equipos para el tratamiento de la

información y sistemas informáticos. De los tres componentes del equipo se han utilizado al 100 % durante el proyecto el ordenador portátil y el ratón, mientras que las gafas se han usado en las pruebas y mejoras hechas de la fase de pruebas para la compatibilidad con RV, por lo que se estima que se han usado entorno a un 20 % y un 10 % del total del proyecto, considerando la media: 15 % como su % de uso aproximado final. Con estos datos, el costo amortizado se calcula de la siguiente manera:

$$\text{Coste amortizado} = \text{Días de uso} \times \text{ \% de uso} \times \text{Precio de amortización por día}$$

Donde el precio de amortización por día se calcula como:

$$\text{Precio de amortización por día} = \frac{\text{precio}}{\text{vida útil}}$$

Dado que el tiempo de vida útil de un ordenador portátil se estima en aproximadamente 5 años y el de un ratón en 3 años, mientras que el de las gafas de RV no tienen una estimación estima entre 2 y 5 años, por lo que las estimaremos en 3,5 años. Por lo tanto, el cálculo del precio de amortización se realiza de la siguiente manera:

Material	Vida útil (días)	Precio de amortzación/día
Ordenador	1825	0,55
Ratón	1095	0,013
Gafas de RV	1277	0,39

Figura 5.5: Amortización por día.

Por tanto las amortizaciones del equipo queda así:

Material	Días de uso	Precio de amortzación/día	%uso	Coste Amortizado
Ordenador	x	0,55	100%	
Ratón	x	0,013	100%	
Gafas de RV	x	0,39	15%	

Figura 5.6: Costes amortizados del equipo.

El ordenador tiene una amortización de x/€ por día y el ratón de y/€ por día. Lo que supone un total de z/€ amortizados por día.

Costes de software Son los costes asociados a los programas necesarios para el desarrollo del proyecto. En este caso, todos los recursos software utilizados han sido gratuitos, ya que Unity es un software gratuito, se ha usado VS Code como editor de código, el cual también es gratuito, así como Git y Trello. Los softwares con los que se ha gestionado el proyecto para creación de diagramas han sido softwares online gratuitos y para las presentaciones se han utilizado los documentos que ofrece Google. Los costes en software por lo tanto han sido 0.

Costes Indirectos

En cuanto a los costes indirectos, se necesitaría al menos de un lugar de trabajo, luz, agua e internet que suponen unos gastos mensuales adicionales:

Concepto	Coste/mes
Local de trabajo	200
Agua	40
Luz	50
Internet	30
	320

Figura 5.7: Costes indirectos.

Dado que el tiempo necesitado en este proyecto ha sido de X días, lo que equivale a no se cuantos meses, por tanto:

$$\text{meses que dura el proyecto} \times \frac{\text{costes indirectos}}{\text{mes}} = \text{costes indirectos totales}$$

Finalmente, los costes totales del proyecto quedan tal que:

Costes directos	
Costes indirectos	
Coste total	

Figura 5.8: Costes totales del proyecto.

En resumen, el proyecto ha tenido una duración de X días, Y horas y un coste total de Z €.

El coste temporal del proyecto es debido en su gran parte al trabajo de investigación previo realizado, pues el tema de este proyecto ha sido estudiado desde hace décadas generando gran cantidad de artículos y material que revisar. También la inexperiencia en el uso de tecnologías como el sistema DOTS de Unity que ha requerido un proceso de aprendizaje para adaptar el código a la estructura de este sistema. Además, las pruebas necesarias para comprobar la correcta integración y funcionamiento de todos los componentes y su adaptación a la realidad virtual, sin ser un desarrollador experto en ella, ha sido otros de los factores claves que justifican el coste temporal del proyecto.

5.1.4. Análisis de Viabilidad

Viabilidad Técnica:

Viabilidad Económica

Viabilidad Legal

Análisis de Riesgos

5.1.5. Diagramas de Análisis

Diagrama de Casos de Uso

Otros Diagramas de Análisis

5.1.6. Conclusiones del Análisis

5.2. Diseño

En esta sección se detallará el diseño del sistema, incluyendo la arquitectura general, la especificación de componentes y módulos, así como los diagramas de diseño pertinentes.

5.2.1. Arquitectura del Sistema

Descripción de la Arquitectura a Alto Nivel

Componentes Principales

Interacción Entre los Componentes

5.2.2. Especificación de Componentes

Descripción de Cada Componente

Se detalla la función y el propósito de cada componente del sistema, así como sus características principales.

Dependencias Entre los Componentes

5.2.3. Diagramas de Diseño

Diagrama de Clases

Diagrama de Secuencia

Diagrama de Componentes

5.2.4. Conclusiones del Diseño

Resumen de las Decisiones de Diseño

Implicaciones para la Implementación

5.3. Implementación

En esta sección se detallará el proceso de implementación del sistema, incluyendo la descripción de las tecnologías utilizadas, el desarrollo de los componentes y la integración de los mismos.

5.3.1. Tecnologías Utilizadas

Lenguajes de Programación

Frameworks y Bibliotecas

Herramientas de Desarrollo y Gestión de Código

Entornos de Desarrollo Integrado (IDE)

5.3.2. Desarrollo de los Componentes

Implementación de la arquitectura

Documentación de código

Lógica de la integración de los componentes

5.3.3. Pruebas

Pruebas Funcionales

Pruebas de Rendimiento

Pruebas de Usabilidad

5.3.4. Mejoras Aplicadas

Mejoras Funcionales

Mejoras de Rendimiento

Mejoras de Usabilidad

Capítulo 6

Análisis de Resultados

6.1. Análisis de Resultados

Evaluación de la Generación de Terreno

Comparación de Biomas Generados

Evaluación de las Poblaciones

Desempeño del Sistema en Tiempo Real

Compatibilidad con Realidad virtual

Experiencia del Usuario

Limitaciones y Áreas de Mejora

Capítulo 7

Conclusiones

7.1. Conclusiones

7.1.1. Logros y Contribuciones

7.1.2. Cumplimiento de Objetivos

7.1.3. Lecciones Aprendidas

7.1.4. Trabajo Futuro

7.1.5. Conclusión General

Apéndice A

Apéndice

A.1. Cronograma de fases del desarrollo

A.1.1. Cronograma

En el siguiente diagrama de Gantt se detalla el diseño de la duración de las tareas.

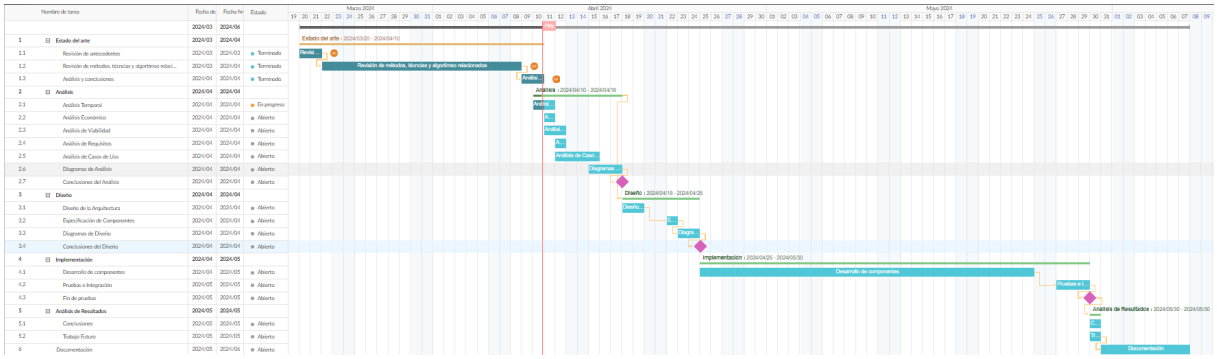

Figura A.1: Diagrama de Gantt.

A.1.2. Especificación del Equipo

Ordenador Portátil

- **Procesador:** 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
- **RAM instalada:** 16,0 GB (15,8 GB usable)
- **Tipo de sistema:** Sistema operativo de 64 bits, procesador basado en x64
- **Memoria en disco:** 500 GB (476 GB usable)
- **Tarjeta Gráfica:** Nvidia GeForce RTX 3060 Laptop (6GB de memoria de video dedicada GDDR6)
- **Sistema operativo:** Windows 11 Home.

Gafas de Realidad Virtual

- **Pantalla**: Dos pantallas LCD (1832x1920 píxeles por ojo).
- **Procesador**: Procesador Qualcomm Snapdragon XR2.
- **Memoria RAM**: 6 GB.
- **Almacenamiento**: 128 GB.
- **Conectividad**: Wi-Fi 6 integrado y Bluetooth 5.0.
- **Audio**: Altavoces integrados y conector de audio de 3.5 mm para auriculares.
- **Seguimiento**: 6 grados de libertad (6DoF).
- **Controladores**: Dos controladores de movimiento Oculus Touch 2.0.
- **Batería**: Batería de ion de litio recargable con duración de hasta 2-3 horas de juego.
- **Peso**: Aproximadamente 503 gramos.
- **Sistema Operativo**: Oculus Quest OS (basado en Android).

Bibliografía

[1] Kent Pawson. Procedural generation: An overview. https://kentpawson123.medium.com/procedural-generation-an-overview-1b054a0f8d41, 2024.

[2] Wikipedia. Generación procedimental, 2024.

[3] Timothy Roden and Ian Parberry. From artistry to automation: A structured methodology for procedural content creation. In Matthias Rauterberg, editor, *Entertainment Computing – ICEC 2004*, pages 151–156, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[4] Darwyn R. Peachey. Solid texturing of complex surfaces. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '85*, pages 279–286, New York, NY, USA, 1985. ACM.

[5] Stephen Lee-Urban. Procedural content generation, 2016.

[6] Peter E. Oppenheimer. Real time design and animation of fractal plants and trees. *ACM SiGGRAPH Computer Graphics*, 20:55–64, 1986.

[7] Wikipedia. .kkrieger, 2024.

[8] Vista pro - graph based terrain generator gpu. https://assetstore.unity.com/packages/tools/terrain/vista-pro-graph-based-terrain-generator-gpu-264414, 2024. Unity Asset Store.

[9] Mapmagic world generator. https://assetstore.unity.com/packages/tools/terrain/mapmagic-world-generator-56762, 2024. Unity Asset Store.

[10] Terrain composer 2. https://assetstore.unity.com/packages/tools/terrain/terrain-composer-2-65563, 2024. Unity Asset Store.

[11] Planetside. What is terragen? https://planetside.co.uk/terragen-overview/, 2024.

[12] Procedural Worlds. Gaia pro - all in one scene creation. https://www.procedural-worlds.com/products/professional/gaia-pro/, 2024.

[13] QuadSpinner. Quadspinner.com official website for gaea. https://quadspinner.com/, Unknown.

[14] World creator - the real-time terrain generator and landscape generator. https://www.world-creator.com.

[15] World machine. https://www.world-machine.com/features.php. Accessed: 2024-02-08.

[16] e-on Software. Vue: Overview. https://info.e-onsoftware.com/vue/overview, 2024.

[17] SideFX. Landscapes in houdini for unreal. https://www.sidefx.com/docs/houdini/unreal/landscape/index.html, 2023. Accessed: March 14, 2024.

[18] Creative Bloq. Creating stunning landscapes in houdini. https://www.creativebloq.com/how-to/create-stunning-landscapes-in-houdini, Year. Accessed: March 14, 2024.

[19] Unity Asset Store. Easyroads3d pro v3. https://assetstore.unity.com/packages/tools/terrain/easyroads3d-pro-v3-469, 2024. Version 3.2.3f3.

[20] Unity Asset Store. Citygen3d - unity asset store. https://assetstore.unity.com/packages/tools/terrain/citygen3d-162468, 2021.

[21] CityGen Technologies Ltd. Citygen3d website. https://www.citygen3d.com/, 2022.

[22] Building generator. Unity Asset Store, 2024.

[23] Esri. Software avanzado de diseño de ciudades en 3d esri cityengine. https://www.esri.es/es-es/arcgis/productos/esri-cityengine/overview, 2023.

[24] Unity Asset Store. Vegetation studio pro extensions, 2022.

[25] Unity. Speedtree. https://unity.com/es/products/speedtree, January 2024.

[26] D. Ebert, F. Kenton Musgrave, D. Peachey, et al. *Texturing and Modeling - A Procedural Approach*. Morgan Kaufmann, 2003.

[27] Luis Oswaldo Valencia-Rosado and Oleg Starostenko. Methods for procedural terrain generation: A review. In *Advances in Intelligent Systems and Computing*, Cham, 2019. Springer.

[28] Khan Academy. Ruido perlin. 2024.

[29] Wikipedia. Worley noise, 2024.

[30] Midpoint displacement algorithm.

[31] A. Krista Bird B. Thomas Dickerson C. Jessica George. Diamond-square algorithm. https://web.williams.edu/Mathematics/sjmiller/public_html/hudson/Dickerson_Terrain.pdf.

[32] I. Marák. Procedural generation of landscapes with water bodies using artificial drainage basins, 1997.

[33] Jacob Olsen. Realtime synthesis of eroded fractal terrain for use in computer games. *Department of Mathematics And Computer Science (IMADA), University of Southern Denmark*, October 2004.

[34] Kazimierz Choroś and Jacek Topolski. Parameterized and dynamic generation of an infinite virtual terrain with various biomes using extended voronoi diagram. *Journal of Universal Computer Science*, 22(6):836–855, 2016.

[35] Hua Fei Yin and Changwen Zheng. A practical terrain generation method using sketch map and simple parameters. *IEICE Transactions on Information and Systems*, E96.D(8):1836–1844, 2013.

[36] Terrain generation using procedural models based on hydrology.

[37] Hydrology-based terrain generation.

[38] Roland Fischer, Judith Boeckers, and Gabriel Zachmann. Procedural generation of landscapes with water bodies using artificial drainage basins. 2022.

[39] Erosion and other algorithms in chunk-based terrain.

[40] Terrain erosion - 3 ways.

[41] G. Cordonnier, J. Braun, M.-P. Cani, B. Benes, É. Galin, A. Peytavie, and É. Guérin. Large scale terrain generation from tectonic uplift and fluvial erosion. *Comput. Graph. Forum*, 35:165–175, 2016.

[42] Autobiomes: procedural generation of multi-biome landscapes. 2024.

[43] Guillaume Cordonnier, Jean Braun, Marie-Paule Cani, Bedrich Benes, Eric Galin, Adrien Peytavie, and Eric Guérin. Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Trans. Graph. TOG*, 36(134), 2017.

[44] Mario Alberto Rivera Legarreta, Mónica Pérez Castañeda, Héctor Adrián Díaz Furlong, and Alberto Roman Flores. Análisis de los algoritmos para generación de ambientes virtuales aleatorios. *Revista de Análisis de Algoritmos y Estructuras de Datos*, 4(1), Septiembre-Diciembre 2016.

[45] Michael Doran and Ian Parberry. Controlled procedural terrain generation using software agents. *IEEE Trans. Comput. Intellig. and AI in Games*, 2(2):111–119, 2010.

[46] mxgmn. Wavefunctioncollapse. https://github.com/mxgmn/WaveFunctionCollapse, Year.

[47] ProcJam. Generating worlds with wave function collapse. https://www.procjam.com/tutorials/wfc/, 2024.

[48] Ishaan Srivastava. A comparative analysis of generative models for terrain generation in open-world video games. *Journal of High School Science*, 2024. Submitted: October 16, 2023, Revised: version 1, November 30, 2023, version 2, January 21, 2024, Accepted: February 1, 2024.

[49] Cognizant. Algoritmo evolutivo. https://www.cognizant.com/es/es/glossary/evolutionary-algorithm, 2024. Accessed: 2024-03-29.

[50] William L. Raffe, Fabio Zambetta, and Xiaodong Li. A survey of procedural terrain generation techniques using evolutionary algorithms. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[51] A. Emilien, U. Vimont, M.-P. Cani, P. Poulin, and B. Benes. Worldbrush: interactive example-based synthesis of procedural virtual worlds. *ACM Trans. Graph. TOG*, 34(4):106, 2015.

[52] George Kelly and Hugh McCabe. A survey of procedural techniques for city generation. *The ITB Journal*, 7(2):87–97, 2006.

[53] Stefan Greuter, Nigel Stewart, Jeremy Parker, and Geoff Leach. Undiscovered worlds - towards a framework for real-time procedural generation of virtual worlds. In *MelbourneDAC 2003 Proceedings, ACM Press*, pages 120–127, 2002.

[54] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of 'pseudo infinite'cities. *Proceedings of GRAPHITE 2003, ACM Press*, pages 87–95, 2003.

[55] Yoav I H Parish and Pascal Mueller. Procedural modeling of cities. *Computer Graphics Laboratory ETH Zurich*, 2001.

[56] Thomas Lechner, Ben Watson, Uri Wilensky, and Martin Felsen. Procedural city modeling. *Computer Graphics Laboratory ETH Zurich*, 2003.

[57] J. Sun and G. Baciu. Template-based generation of road networks for city modeling. *Proceedings of GRAPHITE 2002*, pages 87–95, 2002.

[58] Peter Wonka, Michael Wimmer, Francois Sillion, and William Ribarsky. Instant architecture. *ACM Transactions on Graphics*, 22(3):725–732, 2003.

[59] Arnaud Emilien, Adrien Bernhardt, Adrien Peytavie, Marie-Paule Cani, and Eric Galin. Procedural generation of villages on arbitrary terrains. *Vis. Comput.*, 28(6-8):723–738, 2012.

[60] Er. Isha K. ¿cuáles son algunos de los escollos comunes y las mejores prácticas para la simplificación de malla en la representación 3d? https://www.linkedin.com/advice/0/what-some-common-pitfalls-best-practices-mesh-simplification?lang=es&originalSubdomain=es, 2023.

[61] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

[62] Marco Jinete, Flavio Prieto, and Augusto Salazar. Metodología de remallado basado en la curvatura y orientado a modelos 3d del rostro humano. *Revista Ingenierías Universidad de Medellín*, 14(26):208, enero-junio 2015.

[63] Urs Ramer, David H. Douglas, and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10:112–122, 1973.

[64] William J Schroeder, Jonathan A Zarge, and William E Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, 1992.

[65] Aryamaan Jain, Avinash Sharma, and K S Rajan. Learning based infinite terrain generation with level of detailing. *Computer Science*, 1:1–10, 2022. IIIT Hyderabad, India; Inria, Universite Cóte dÁzur, France; IIT Jodhpur, India.

[66] Jian Wu, Yuanfeng Yang, Shengrong Gong, and Zhiming Cui. A new quadtree-based terrain lod algorithm. *J. Softw.*, 5:769–776, 2010.

[67] Procedural generation of landmasses and terrain. 2024.

[68] Cristina Gasch Garcia. *PROCEDURAL GENERATION OF NATURAL ENVIRONMENTS*. PhD thesis, Universitat Jaume I, January 2021.

[69] CG Wire. Texturing and shading in animation: Definition, process & challenges, 2024.

[70] Unity. Gpu instancing. https://docs.unity3d.com/Manual/GPUInstancing.html, 2024-04-02. Accessed: April 05, 2024.

[71] Optimización del rendimiento en realidad virtual y aumentada nativa - fastercapital. https://fastercapital.com/es/tema/optimizaci%C3%B3n-del-rendimiento-en-realidad-virtual-y-aumentada-nativa.html, 2024. Accessed on April 05, 2024.

[72] Cómo optimizar el rendimiento de la realidad virtual para narrativas complejas. https://es.linkedin.com/advice/0/how-can-you-optimize-vr-performance-complex-narratives-5oohf?lang=es, Unknown Year. Accessed on Unknown Date.

[73] Virtual reality best practices. https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/XRDevelopment/VR/DevelopVR/ContentSetup/, 2024. Accessed on January 26, 2024.

[74] Jobs overview. https://docs.unity3d.com/Manual/job-system-jobs.html. Accessed: 2024-04-13.

[75] Unity Technologies. Burst user guide. https://docs.unity3d.com/Packages/com.unity.burst@1.2/manual/index.html, 2023. Accessed: 2024-03-29.

[76] Unity Technologies. Ecs for unity. https://unity.com/ecs, 2024. Accessed: 2024-03-29.

[77] What is a job system? unity blog. https://blog.unity.com/engine-platform/what-is-a-job-system, 2024. Accessed: March 20, 2024.

[78] 3.5.4 tabla de amortización simplificada, March 2024. Tabla de amortización simplificada de la Agencia Tributaria para edificios y otras construcciones, instalaciones, mobiliario, enseres y resto del inmovilizado material, maquinaria, elementos de transporte, equipos para tratamiento de la información y sistemas y programas informáticos, útiles y herramientas, ganado vacuno, porcino, ovino y caprino, ganado equino y frutales no cítricos, frutales cítricos y viñedos, olivar.