UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Interface de abstração de algoritmos de geração procedural de terrenos para jogos através da parametrização de biomas

LUIZ FERNANDO DA SILVA SOTERO

Orientador: Geber Lisboa Ramalho

JULHO/2016

Luiz Fernando da Silva Sotero

Interface de abstração de algoritmos de geração procedural de terrenos para jogos através da parametrização de biomas

Monografia apresentada à graduação de Ciência da Computação da Universidade Federal de Pernambuco como requisito para a conclusão do curso de graduação

LUIZ FERNANDO DA SILVA SOTERO

Orientador: Geber Lisboa Ramalho

JULHO/2016

AGRADECIMENTOS

Gostaria de agradecer primeiramente aos meus familiares, o suporte que me foi dado durante toda a minha vida, principalmente em relação à minha educação, que permitiu que eu chegasse a esta etapa de conclusão da graduação.

Agradeço aos meus colegas de trabalho que permitiram que eu dedicasse o tempo necessário para realizar este trabalho mesmo em momentos complicados dentro da empresa. Agradeço aos meus colegas da faculdade, por todos os bons momentos de distração e por toda a experiência compartilhada ao decorrer da graduação. Gostaria de agradecer também a todos os meus amigos de fora da faculdade por todos os momentos de descontração que me permitiam aguentar a pressão e a carga de trabalho que o curso me fornecia.

Agradeço à minha namorada por todos os bons momentos, pela força que me foi dada e pela compreensão da minha dedicação para concluir esta etapa da minha vida. Sem o seu apoio e cobrança, eu não conseguiria terminar este trabalho com o escopo e qualidade que eu gostaria.

Aos professores do Centro de Informática, agradeço por terem se dedicado para transmitirem o máximo de conhecimento possível para a nossa turma. Sem vocês, seria impossível ter a qualidade do conhecimento que eu hoje possuo sobre os mais diversos assuntos relevantes à computação. Agradeço especialmente ao professor Geber, meu orientador neste trabalho, que me orientou e esteve sempre disponível para resolver meus problemas e me guiar ao melhor caminho.

RESUMO

A Geração procedural de terrenos para jogos é uma técnica que envolve na sua maior parte programação, entretanto os resultados devem ser gráficos e validados por uma equipe de artistas ou editados, de alguma forma, pela equipe gráfica. Diversos jogos dependem da criação automática de conteúdo por conta da redução de mão de obra necessária e de possibilidades de escalabilidade e adição de variedade nos estágios. Um problema associado com esse processo é o da grande diferença entre estes dois domínios (artístico e programação) fazendo com que a interação das duas áreas possa ser bastante confusa e complicada. Um conceito bastante explorado por jogos recentes é o da interação de jogadores com ecossistemas e biomas, mas as ferramentas atuais de geração procedural de terreno que podem ser licenciadas ou públicas não dão suporte para tal funcionalidade. Empresas, geralmente, possuem seus editores de terrenos próprios, ou utilizam os editores fornecidos por algum motor gráfico e fazem suas modificações para atingir seus resultados esperados na geração de conteúdo aleatório. Este trabalho propõe uma interface para artistas para a criação de *inputs* em algoritmos de criação procedural através da parametrização de biomas que possam ser incorporados em outras ferramentas e algoritmos.

Palavras-chave: geração procedural, terrenos, jogos, motores gráfico

ABSTRACT

Procedural generation for terrains is a technique composed on its majority by programming, but the results must be graphical and validated by a graphics team or edited by them. A variety of games depends on the automated content generation to reduce the staff needed and to help with scalability and variety of game levels. A problem with this process is the big gap between the two areas (programming and arts), which can lead to confusing and complicated interaction if not done correctly. A concept that is been explored by recent games is the interaction between the player and the ecosystem and biomes, but the current open or licensed tools for procedural terrain generation do not have many features for that. Companies tend to have their own terrain editors software or to utilize a graphical engine's one and modify them to reach their goals within generating procedural. This project proposes an interface for artists for creating inputs for a procedural terrain generation tool by breaking biomes into parameters that can be attached to a game engine or to a third party program.

Palavras-chave: procedural generation, terrains, games, game engines

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Neste capítulo serão abordados os principais aspectos deste trabalho, através de uma breve introdução do assunto e da relevância do tema dentro da indústria de jogos na qual este projeto se baseia. Também serão apresentados objetivos gerais, específicos da aplicação desenvolvida e a estrutura deste trabalho.

1.1 Contextualização e motivação

Em computação, geração procedural é um método de criação de dados que utiliza algoritmos para a criação de conteúdo de forma contrária ao da criação manual. Na área de jogos e computação gráfica, este método é muito utilizado para a criação de texturas, mapas, terrenos, objetos utilizáveis por jogadores, objetos que compõem uma cena de forma estática, posicionamento de inimigos e até mesmo música e efeitos sonoros [41].

Os primeiros jogos de computadores que utilizavam recursos gráficos eram muito limitados quanto ao uso de memória. Estas restrições alavancaram a criação de conteúdo, como terrenos, de forma automática por algoritmos em tempo de execução [1].

Estas técnicas procedurais podem também introduzir aleatoriedade, criando elementos e comportamentos menos previsíveis dentro de um determinado jogo e então reduzindo a necessidade da produção excessiva de conteúdo de forma manual[3]. Um dos exemplos mais famosos que utilizou esta técnica é o jogo *Rogue (AI Design* 1980*)*. Este jogo marcou a sua época pela forma da geração de cavernas representadas graficamente por caracteres ASCII de forma automática e aleatória, criando, como resultado, uma experiência de que o jogo poderia ser jogado quase sempre de uma forma diferente.

Desde sua introdução, a geração procedural de conteúdo se tornou muito utilizada para se gerar salas, texturas, corredores, inimigos e outros demais elementos que possam formar a base de um jogo, aumentando a sua complexidade sem que se perdesse muito tempo criando um mundo gigante de forma manual e exaustiva, além de reduzir drasticamente a necessidade pelo uso de memória[2]. Sendo assim, a redução do tamanho

dos arquivos, uma vez que estes só precisam guardar os algoritmos responsáveis por gerar o conteúdo, foi uma arma poderosíssima para a indústria quando introduzida.

Atualmente, videogames estão instalados dentro da cultura e rotinas de muitos. De acordo com um relatório apresentado pela *Entertainment Software Association,* 59% dos Americanos jogavam videogames em 2013 e consumidores gastaram cerca de $21,5 bilhões nesta indústria[4]. Por conta do crescimento nas vendas e da capacidade de processamento de smartphones, é cada vez mais comum ver pessoas jogando no seu aparelho celular pessoal. De acordo com este mesmo relatório, jogos casuais, como *Angry Birds*, chegam a ocupar uma fatia de 46% dos jogos jogados em plataformas móveis. Uma característica muito comum deste gênero é o uso de geração procedural de estágios para se criar uma experiência de jogabilidade muito grande e dinâmica, fazendo com o que o jogador tenha sempre algum estágio novo ou diferente para jogar.

Por conta do grande poder computacional dos computadores e videogames atualmente, empresas buscam cada vez mais alcançar qualidade gráfica impecável nos seus jogos. Muitos títulos da atualidade não só possuem gráficos super-realistas como também mundos exploráveis cada vez maiores o que faz com que estas empresas acabem crescendo cada vez mais para incorporar a grande quantidade de profissionais para produzir todo o conteúdo necessário. Uma entrevista com o diretor e presidente da Rockstar North feita pelo site *Develop* revelou vários pontos interessantes sobre o funcionamento da empresa. Ao ser questionado sobre o número de profissionais que trabalharam na produção de *Grand Theft Auto V*, o presidente Leslie Benzies disse que provavelmente mais de 1.000 profissionais chegaram a trabalhar no título[5]. *Grand Theft Auto V* teve sua produção iniciada em 2008 e seu primeiro lançamento foi em 2013, seguido por um segundo lançamento em 2014 para os videogames da nova geração e em 2015 para *Microsoft Windows*. Ranqueado número 1 em vendas em 2013, o título mostrou o quanto os jogadores da atualidade prezam por um mundo vasto, explorável com conteúdo rico e gráficos surpreendentes.

Apesar do grande sucesso de títulos como *Grand Theft Auto V*, é possível perceber como a produção de conteúdo afeta o tempo de desenvolvimento de um jogo. Conteúdo feito de forma manual pode acabar com a dinamicidade da jogabilidade, uma vez que para cada possível combinação de elementos que compõem um mapa, objetivo ou uma cena, é necessária a produção de um conteúdo específico para aquela problemática. Na entrevista feita pelo site *Develop*, o presidente Bezies afirma que grande parte do tempo gasto na produção se deu pela grande atenção aos detalhes no conteúdo gerado de forma manual.

Algumas empresas estão elevando o patamar da geração automática de conteúdo. O jogo *Elite: Dangerous*, por exemplo, simula um cenário de aventura e exploração espacial que possui um universo de 400 bilhões de sistemas criado de forma procedural em uma escala real [6]. O jogo foi lançado em 2014 e não é o único deste gênero que busca quebrar os limites da geração automática de elementos incorporáveis a jogos em tempo real.

A geração de todo esse conteúdo de forma automática pode também gerar problemas. Dependendo das técnicas utilizadas e da clareza das regras estabelecidas, pode ser muito difícil se garantir que todo conteúdo criado possa ser utilizado no jogo em questão, ou até mesmo que os terrenos e mundos criados possuam um nível de dificuldade adequado para a jogabilidade desejada, existe, também, a possibilidade de que o material construído de forma procedural não corresponda aos requisitos de qualidade impostos pelo time [3]. Um outro possível problema que merece atenção é de que a utilização de técnicas complexas pode estar atrelada a um alto custo computacional que pode impactar diretamente a experiência do jogador, principalmente se não forem executas corretamente.

1.2 Objetivos

Uma dificuldade ao incorporar estes tipos de técnicas em um jogo é a de que a programação dos algoritmos é feita por um time, e os resultados, geralmente gráficos, devem ser validados por um time diferente. A conversa entre estes domínios, se não feita de forma eficaz, pode causar mais danos para um projeto do que benefícios em questão de tempo e esforço. Ferramentas de geração procedurais de conteúdo geralmente são escritas pelas empresas com o objetivo de trabalhar em cima das regras definidas pelos jogos a

serem criados por elas. Dependendo da escala dos títulos e da qualidade visual estimada pela empresa, é possível que várias ferramentas sejam criadas para serem utilizadas por diversas equipes diferentes para que elas possam, juntas, criar a versão final de algum objeto ou mundo explorável.

Sendo assim, este trabalho propõe explorar a possibilidade de criação de uma ferramenta que possa melhorar a interação entre profissionais de algumas das áreas distintas que trabalham neste mesmo processo de criação de conteúdo de forma procedural. Pelo motivo do conceito de geração de conteúdo procedural ser muito amplo, este trabalho terá como objetivo central o de estudar e implementar uma ferramenta que auxilie a criação de terrenos para jogos, com foco na geração procedural, edição manual e visualização do terreno final através da projeção de um mapa de biomas na malha, funcionalidade pouco explorada atualmente pelas ferramentas que dominam o mercado.

O trabalho tem como objetivos específicos:

I. Explicar as diversas técnicas utilizadas para a geração procedural de conteúdos
II. Apresentar quais ferramentas são utilizadas na atualidade para a criação de terrenos de forma procedural.
III. Fazer um levantamento de possíveis funcionalidades para uma ferramenta de criação procedural através de um grupo de estudo.
IV. Criar um algoritmo simples de geração procedural de terrenos.
V. Elaborar o algoritmo para incluir a geração procedural de regiões de biomas.
VI. Criar uma camada de interface por cima do algoritmo para que as funcionalidades possam ser validadas por um grupo de estudo.

1.3 Estrutura do trabalho

Este trabalho é composto por 6 capítulos. No capítulo 2 apresentaremos alguns tipos de geração procedural de conteúdo e suas técnicas. No capítulo 3, analisamos o estado da

arte na geração de terrenos para jogos. O capítulo 4 detalha a criação da ferramenta de geração procedural de terreno feita neste trabalho e a escrita de suas funcionalidades. Por fim, os capítulos 5 e 6 descrevem resultados alcançados pela ferramenta e possíveis soluções para os problemas encontrados além de sugestões de trabalhos futuros envolvendo o assunto.

2. GERAÇÃO PROCEDURAL DE CONTEÚDO

Este capítulo tem como objetivo analisar as diversas questões relacionadas à geração de conteúdo para jogos de forma procedural e quais destas questões serão abordadas dentro deste trabalho.

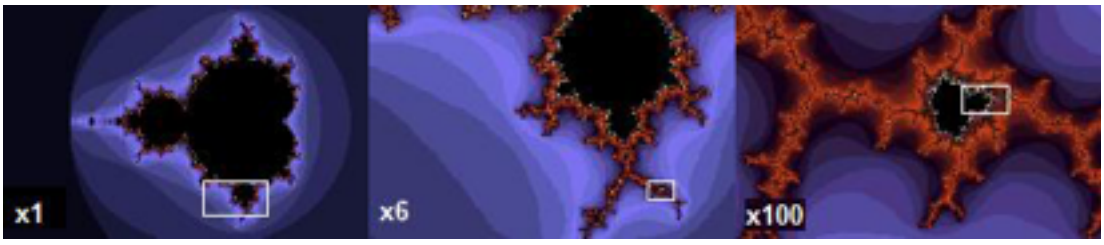
2.1 Introdução

A geração de conteúdo de forma automática para jogos possui vários desafios que variam dependendo das regras do jogo impostas pelo game designer. Sendo assim, esta tarefa pode se tornar bastante complexa e deve ser feita com atenção, uma vez que ela afeta diretamente a experiência de jogadores e a qualidade do produto final. Para uma empresa de jogos, é importante conhecer quais técnicas podem solucionar quais problemas e então aplica-las, ou adaptá-las, para a realidade do projeto atual.

2.2 Ruídos e Fractais

Um fractal é um fenômeno natural ou um conjunto matemático que exibe um padrão repetitivo independente de escala. Caso a repetição seja exatamente a mesma, o padrão é chamado de autossimilar. Fractais também podem ser muito parecidos em diferentes escalas e também estão atrelados à ideia de um padrão de detalhe que se repete. Um exemplo de conjunto que representa um fractal visto em escalas diferentes pode ser visto na imagem abaixo.

Figura 1 Conjunto Mandelbrot em diversas escalas

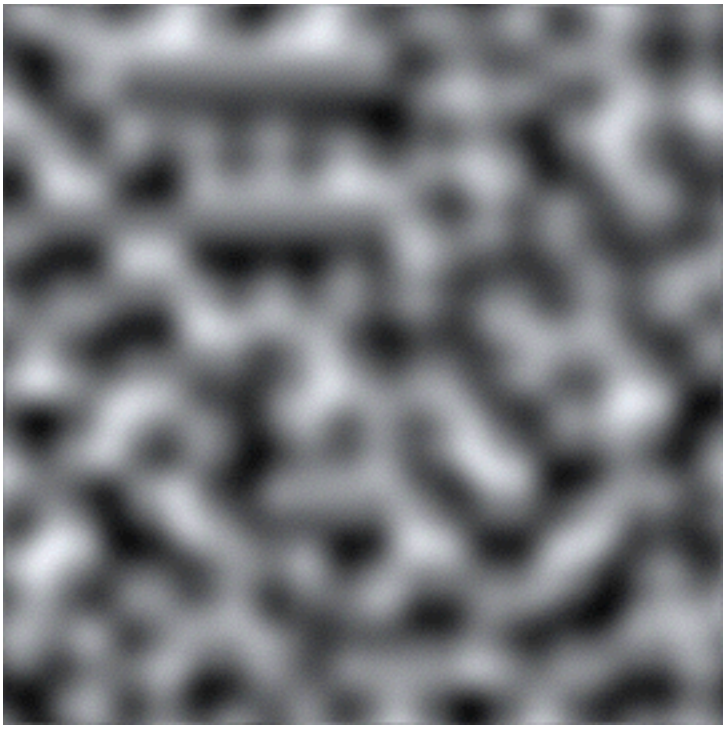


Diversas aproximações de fractais podem ser encontradas na natureza com características autossimilares quando observadas até um certo nível de escala. A conexão entre fractais e folhas, por exemplo, está atualmente sendo utilizada para descobrir o quanto de carbono se tem em uma árvore[7]. Outros fenômenos da natureza com características

semelhantes são as de conexões de rios que formam uma malha fluvial, crateras, árvores, ondas do oceano, detalhes em montanhas e a linha que define a costa de um continente ou ilha. Por conta da existência destas aproximações fractais em fenômenos naturais, diversas técnicas de geração de terrenos de forma procedural utilizam algoritmos de criação de fractais para compor partes de seus modelos finais e criar terrenos com uma grande similaridade com os reais.

A utilização de fractais está muito conectada a geração de detalhes perceptíveis, uma outra técnica utilizada para gerar superfícies, texturas e variações em padrões que se assemelham a realidade é a de utilização de algum tipo de ruído. *Value noise* é um tipo de ruído muito utilizado na computação gráfica como uma primitiva para se gerar texturas procedurais. O método para criação deste tipo de ruído consiste em criar uma grade de pontos com valores aleatórios associados a cada vértice. Em seguida, aplica-se uma função de ruído que interpola os valores dos vértices ao redor de um dado vértice [8]. Para várias aplicações, diversas oitavas destes ruídos podem ser somadas para se criar uma forma de ruído fractal, um dos maiores exemplos é o ruído de Perlin (*Perlin Noise*), ilustrado na figura a seguir.

Figura 2 ***Perlin Noise*** *com diversas oitavas somadas*



2.3 Texturas

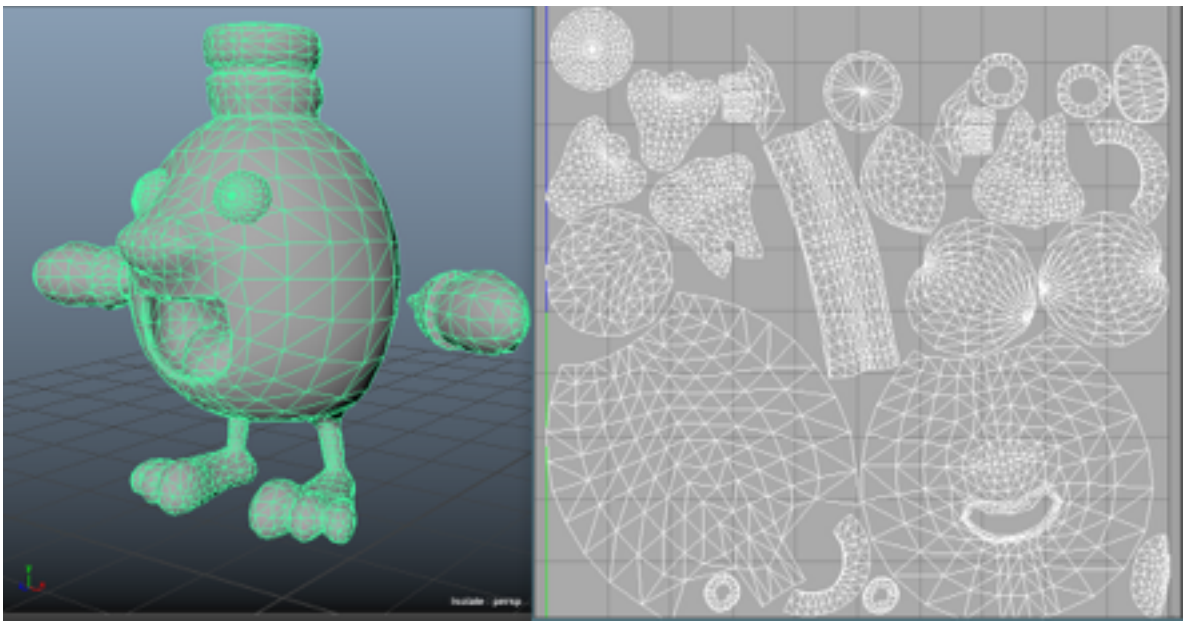
A criação procedural de texturas é uma das tarefas associadas a criação de conteúdo de forma procedural. Uma textura procedural é uma imagem gerada por computador utilizando um algoritmo que tem como objetivo representar uma superfície real ou criar uma representação fiel de elementos naturais como madeira, mármore, granito, rochas[9].

Geralmente, o visual final é alcançado utilizando-se ruídos fractais e funções de turbulência que são utilizadas como representações matemáticas da aleatoriedade encontrada na natureza[9].

O processo de mapeamento de texturas para objetos 3D se dá através do mapeamento de pixels para posições no objeto. Texturas, na área de jogos geralmente existem dentro de um plano 2D em que cada pixel possui uma coordenada UV (U para o eixo horizontal e V para o vertical). Sendo assim, uma textura aplicada no espaço UV pode ser facilmente mapeada para o objeto 3D e utilizada para renderização, entretanto, é necessário que se crie um mapa no qual cada vértice do modelo 3D possua uma coordenada

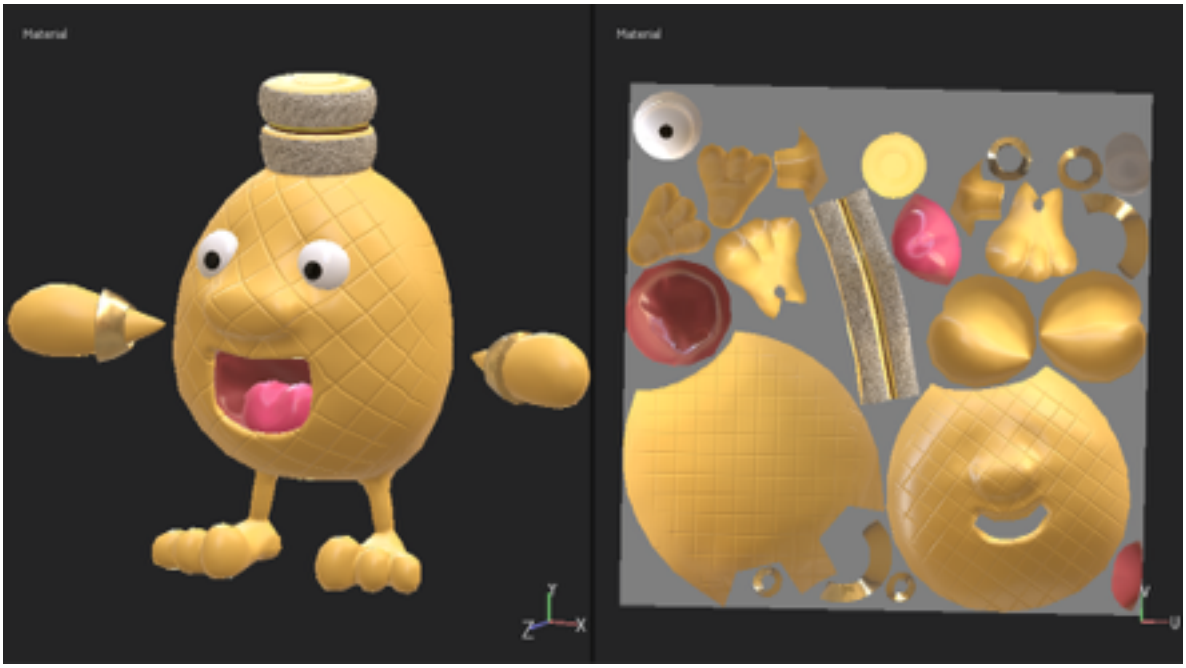
no plano UV, como na figura abaixo. Caso duas faces de uma malha tenha uma interseção de UVs a projeção da textura possivelmente trará resultados indesejáveis.

Figura 3 Mapeamento de modelo 3D nas suas coordenadas UVs



Tendo um mapa criado e associado a uma malha, é possível se aplicar uma textura que será projetada diretamente sobre o modelo 3D, vide figura 4.

Figura 4 Textura aplicada sobre plano UV e projetada em um objeto 3D



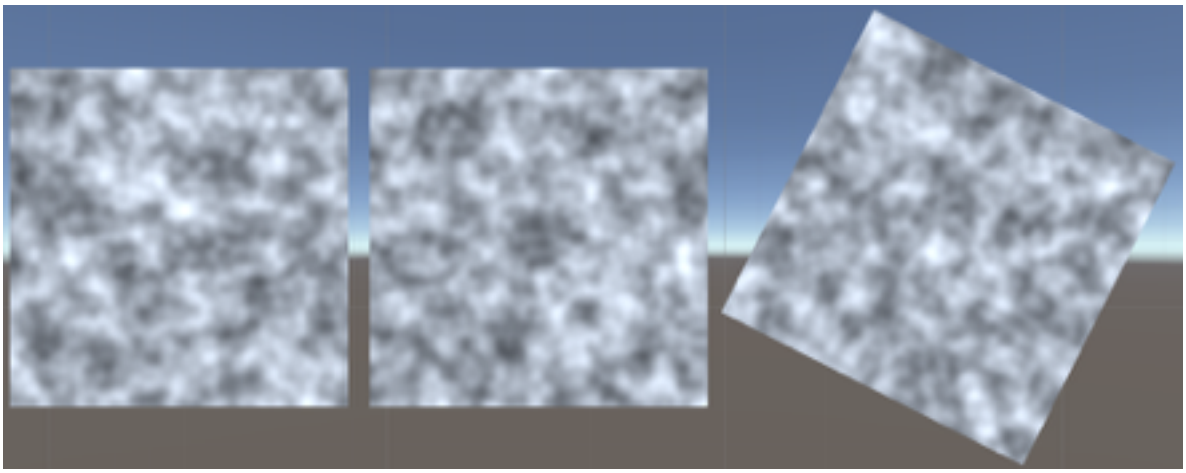
Para o processo de criação de texturas de forma procedural, o caminho inverso também é possível. Contrário ao processo de se inserir uma textura em um mapa para que ela possa ser projetada em um objeto, a técnica conhecida como *Solid Texturing* consiste em utilizar a localização de vértices em um objeto como índices para se extrair alguma informação ou como parâmetros de funções que retornam atributos como resultados [10].

Sendo assim, é possível se utilizar as coordenadas *x,y,z* de objetos no plano tridimensional para gerar informação em um ou mais pixels nas coordenadas UVs, assim se criando uma textura de forma automática. É importante perceber que as coordenadas podem estar escritas em relação ao objeto ou em relação a origem do mundo e que o tipo de coordenada a se utilizar está, geralmente, atrelado a qual tipo de resultado se deseja atingir na geração de texturas [9][10]. Ao invés de se editar imagens para que sua projeção final no modelo seja ótima e que sua repetição não se torne óbvia, este processo consegue mapear pixels visíveis no modelo 3D evitando distorções no resultado final[9].

Programas como o *Substance Painter* (*Allegorithmic*) possuem geradores de ruídos integrados que podem criar texturas que podem ser projetadas diretamente no modelo 3D

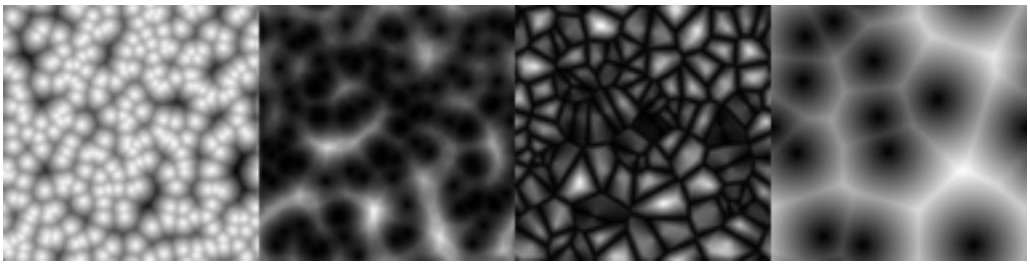
sem distorções utilizando técnicas como a *Solid Texturing*. Um simples exemplo de um ruído Perlin gerado através da leitura de coordenadas no espaço tridimensional pode ser visto na imagem a seguir.

Figura 5 ***Perlin noise*** *gerado através da leitura das coordenadas de pixels de um plano*



Uma outra técnica comum para a geração de texturas de forma automática é conhecida como *Cellular Texturing*. A técnica consiste em se gerar pontos espalhados no espaço 2D de forma aleatória e então, para cada pixel, se verificar sua posição relativa ao par de pontos mais próximos para então se escolher uma cor [9][11]. Esta técnica é muito comum para se produzir texturas que se assemelham a células, como o padrão de escamas. Não é necessário a utilização de ruídos para a geração destes tipos de textura, mas eles são geralmente utilizados para se criar um visual mais aproximado de algo existente na natureza. Exemplos de textura geradas utilizando esta técnica podem ser abaixo.

Figura 6 Texturas geradas utilizando-se a técnica de ***Cellular Texturing***



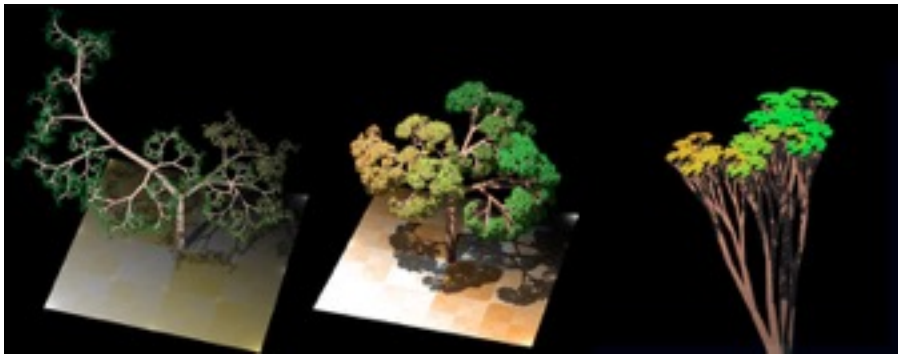
2.4 Malhas

Modelagem procedural é um conjunto de técnicas, geralmente inclusas em algum tipo de ferramenta de modelagem 3D, que tem como objetivo criar objetos tridimensionais a partir de um conjunto de regras, contrário ao da edição através de *inputs* de usuários. *L-Systems*, Fractais e *Generative modeling* são algumas das técnicas de geração procedural que podem criar modelos tridimensionais através de algoritmos.

Lindenmayer system ou L-Systems é um sistema de redução e um tipo de gramática formal que consiste em um alfabeto de símbolos que podem ser utilizados para se criar cadeia de caracteres (*strings*), uma coleção de regras de produção que expandem símbolos em outros símbolos e *strings* maiores, uma *string* de axioma inicial e um mecanismo de tradução das cadeias de caracteres para estruturas geométricas[11]. A técnica criada por Aristid Lindenmayer em 1956 na universidade de Utrecht tinha como objetivo descrever o comportamento de células de plantas e modelar o processo de crescimento e desenvolvimento de plantas. L-systems também foi utilizada para modelar a morfologia de uma variedade de organismos[12] e pode ser usada para gerar fractais autossimilares.

A natureza recursiva das regras leva a criação de estruturas autossimilares. Modelos de plantas e algumas formas orgânicas são simples de se definir através da gramática proposta e com o aumento da quantidade de interações do algoritmo recursivo, os modelos apresentam uma característica de crescimento e tornam-se mais complexos. Para se gerar imagens ou modelos 3D, é necessário que os símbolos utilizados no alfabeto correspondam a elementos ou funções de desenho em uma tela, textura ou mundo 3D. A imagem abaixo foi gerada com um *L-system* que utiliza uma variação do fractal *Dragon Curves*.

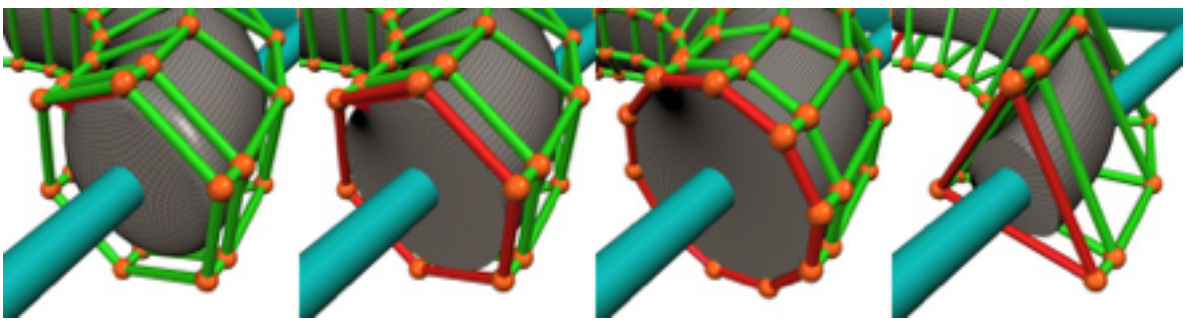
Figura 7 Modelos 3D de árvores geradas com L-systems



Generative modeling language ou GML é uma simples linguagem de programação de descrição de objetos e formas tridimensionais complexas. Esta linguagem segue o paradigma de "*Generative modelling*" definido como uma mudança no paradigma de como se descreve formas. Uma forma passa a ser descrevida por uma sequência de passos de processamento ao invés do resultado final após a aplicação de operações [13]. A GML é uma implementação concreta da abordagem generativa. Sua principal funcionalidade é de ser uma linguagem de computação completamente funcional e que pode ser usada com eficiência para gerar arquivos de descrição de objetos e formas [13].

A GML é muito parecida com a *PostScript* (Adobe) mas sem os operadores 2D existentes, ela provê, no entanto, diversos operadores para gerar modelos 3D. Unida com seu motor gráfico feito em *OpenGL*, a linguagem se integra como um ambiente de criação e visualização de modelos para unir a criação e a visualização interativa da modelagem 3D, a ferramenta *GMLStudio* é a mais comum para se trabalhar com a linguagem. A figura abaixo mostra o resultado da utilização da GML para a produção de tubos de forma procedural e a sua visualização.

Figura 8 Tubos gerados de forma procedural utilizando GML



Algumas *Game Engines* da atualidade também possuem formas de se criar malhas de forma procedural. A Unity 5 e Unreal Engine 4, por exemplo, possuem métodos para se definir a posição de vértices e como a *engine* deve criar triângulos utilizando estes vértices criados e ainda é possível modificar as malhas construídas em tempo de execução[15][16]. Com essas ferramentas, dentro ou fora de *game engines*, empresas tem inserido cada vez mais conteúdo aleatório dentro de seus jogos. O estúdio de jogos Frima Studio tem utilizado o poder da ferramenta *Houdini* (*Sidefx*) para criar conteúdo procedural para os seus jogos [14]. O jogo Chariot (Frima Studio) teve todos os seus *levels*, desde o design até a construção dos blocos de colisão construídos diretamente utilizando as ferramentas de geração de conteúdo procedural da *Houdini*. Uma imagem do jogo pode ser vista abaixo.

Figura 9 Chariot (Frima Studio) tem seu mapa gerado inteiramente de forma procedural

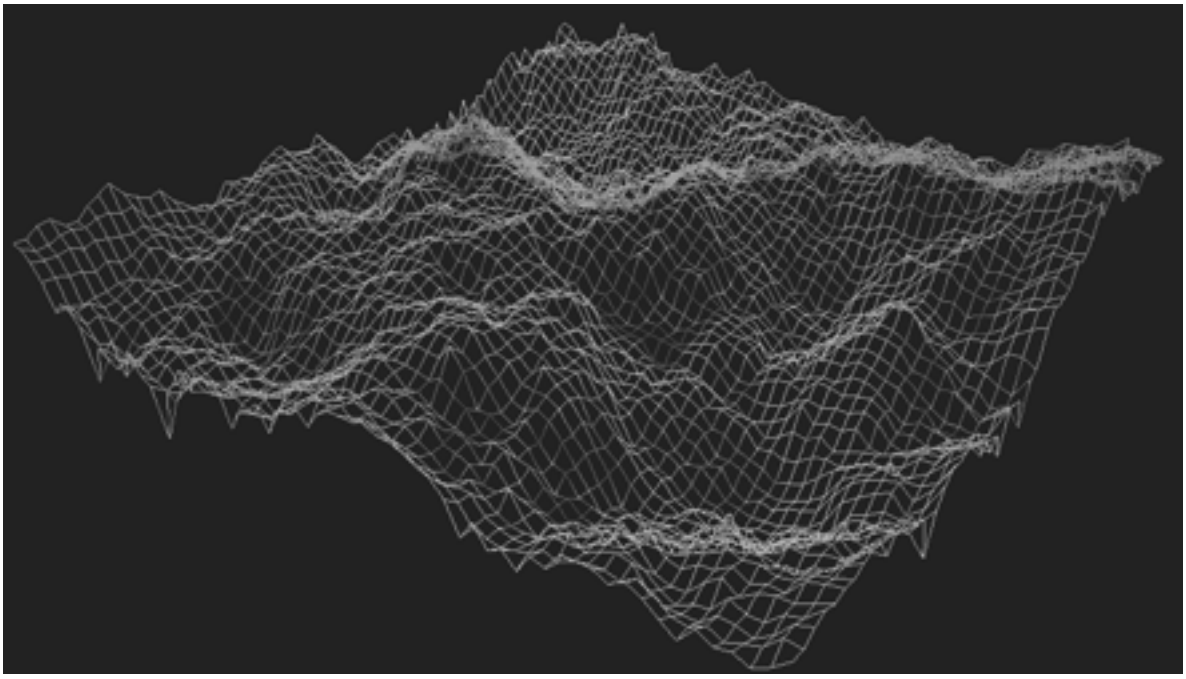
2.5 Terrenos

A geração procedural de terrenos talvez seja a técnica mais comum quando se diz respeito a geração de conteúdo automático. Está atividade não está só ligada a jogos, diversos pesquisadores já utilizaram técnicas nesta área para a simulação de efeitos de erosão, crescimento de árvores, formação de planetas, criações de malhas fluviais[17] e principalmente na indústria de filmes e efeitos visuais.

Entretanto, a indústria de jogos tem explorado bastante a criação de terrenos de forma automática e os resultados são os mais variados possíveis. O ponto principal a se entender é de que a definição de um terreno pode variar de acordo com um jogo, enquanto um jogo trata cavernas exploráveis como o terreno em que o jogador poderá interagir, outros criam mapas com montanhas, oceanos, rios e florestas. É importante entender que a criação do terreno está relacionada a malha que compõe o solo e as estruturas presentes, não necessariamente incluindo árvores, pedras e objetos 3D que compõem a cena.

Terrenos Fractais são um bom exemplo de geração de terrenos de forma procedural. Este tipo de malha é gerado utilizando um algoritmo estocástico para a produção de fractais que imitam comportamentos naturais de superfícies. O objetivo não é o de criar fractais, mas sim de criar uma aleatoriedade na superfície que exiba comportamentos de fractais [18]. Como descrito na seção 2.2 deste trabalho, muitos fenômenos naturais possuem algum tipo de comportamento autossimilar que pode ser modelado através de superfícies fractais [19], um exemplo de uma malha perturbada por ruídos que apresentam comportamento fractal pode ser visto a seguir:

Figura 10 Terreno fractal



A união de superfícies fractais com texturas procedurais, que apresentam variações baseadas no ângulo da superfície e outros fatores, pode produzir terrenos muito semelhantes a biomas reais encontrados na natureza. Entretanto, confirmar que superfícies naturais se comportam de uma forma fractal ainda é objetivo de estudos. Terrenos naturais possuem muitas variações de comportamentos baseado em alturas, biomas, regiões específicas e escalas diferentes. Isso quer dizer que calcular um fractal que representa, no geral, uma região natural pode trazer resultados não esperados [20]. Por conta de todas essas variações e possíveis complicações, funções fractais simples nem sempre são apropriadas para a geração de terrenos de forma procedural, uma abordagem mais sofisticada com resultados melhores é a de se utilizar técnicas multifractais, que mesclam funções fractais em diferentes escalas e regiões e que conseguem modelar o comportamento de terrenos naturais com uma maior precisão[21]. A imagem a seguir mostra um simples terreno gerado com técnicas multifractais utilizando o ruído *Perlin*:

Figura 11 Terreno multifractal utilizando ruidos Perlin

2.6 Posicionamento de objetos e composição de cenas

A abordagem procedimental para geração de conteúdo não está apenas atrelada a criação de objetos, existe também a necessidade de como e onde posicionar elementos chave da jogabilidade.

Não basta para um jogo ter o seu mapa gerado de forma automática, porém completamente aleatória (sem seguir regras conectadas a jogabilidade esperada). O posicionamento correto, ou natural, de objetos podem construir um ambiente muito mais rico para o jogador, além possibilitar que os objetos possam ser de fato alcançados. Jogos que dependem bastante da geração e posicionamento correto de objetos e terrenos são os jogos de infinitos ciclos, este tipo de gênero não segue a ideia comum de se atingir um objetivo específico em um estágio, mas trazem o conceito de que o jogador deve durar o

máximo possível dentro de um determinado estágio, podendo ser, potencialmente, infinito. [22]

Jogos que pertencem a esse gênero, por terem *levels* potencialmente infinitos, dependem da inserção de obstáculos e objetos em tempo de execução para que a experiência do jogador não seja interrompida por telas de carregamentos ou latências e quedas de *framerate* [22]. Sendo assim, é extremamente necessário se criar um algoritmo moldado por regras que regem a jogabilidade para inserir mapas, obstáculos e objetos de forma procedural para se compor a cena final.

Atualmente, existem trabalhos publicados na academia que propõem soluções para o problema de posicionamento de elementos que compõem uma cena e mapas jogáveis. Alguns trabalhos buscam alinhar as necessidades de um game designer com a ideia de progressão de dificuldade[23], enquanto outros utilizam algoritmos de otimização e busca para se criar estágios controlados por funções que mapeiam a dificuldade e diversão[24]. É importante lembrar que cada jogo possui suas regras de jogabilidade associadas e que as ferramentas disponíveis na atualidade podem não suprir todas as necessidades de um título específico, evidenciando a necessidade da possibilidade de extensão das ferramentas e plataformas atuais para que designers possam criar as *constraints* e regras complementares.

2.7 Resumo

Este capítulo apresentou alguns dos tipos de conteúdo em jogos e técnicas para produzi-los de forma procedimental. Algumas questões sobre compromisso ao adotar tais técnicas foram expostas e entender como lidar com elas pode se tornar um trabalho complicado. É importante lembrar que cada jogo possui o seu conjunto de regras e que essas regras estão alinhadas a jogabilidade desejada e a qualidade e coerência estética esperada.

A seção 2.5 deste trabalho, que fala de terrenos procedurais, abre um espaço muito amplo para estudos e experimentações, uma vez que é possível se validar o resultado com exemplos reais e testar a modelagem matemática e estatística de fenômenos reais além de que o terreno gerado pode ser utilizado em diversas áreas de aplicações diferentes. É

importante entender que não se existem requisitos mínimos padronizados para a geração de terrenos, cada terreno depende exclusivamente do objetivo que a equipe queira alcançar e pelo motivo de que esta área se torna muito ampla, empresas vem trabalhado para criar ferramentas com objetivos específicos, sejam elas criar terrenos realistas que se assemelham aos terrenos encontrados na natureza, ou criar em formatos hexagonais que ajudam diretamente na criação de um gênero de jogos específico. Levando isso em consideração, esse trabalho terá como foco este assunto e levantará um estudo sobre o que já existe na atualidade em questão de geração de terrenos de forma procedural no próximo capítulo para que alguma técnica possa ser desenvolvida e integrada com ferramentas no futuro que auxilie criadores de jogos a ter mais controle sobre os terrenos gerados.

O estudo a seguir levará em conta a existência de uma interface amigável para usuários, a possibilidade de geração de conteúdo em tempo de execução (*online*), a possibilidade de geração dos terrenos em *background* (*offline*) e com o objetivo de criar terrenos que se assemelham a terrenos naturais.

3. ESTADO DA ARTE NA GERAÇÃO DE TERRENOS PARA JOGOS

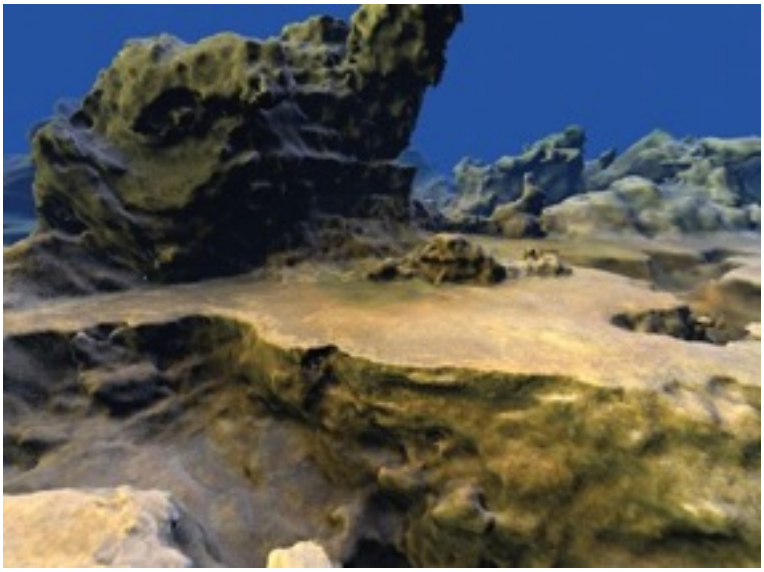
Este capítulo tem como objetivo apresentar as ferramentas atuais utilizadas para a criação de terrenos de forma procedural na indústria de jogos que são públicas ou possível de se comprar uma licença para uso. Este capítulo terá como foco apenas a criação das malhas dos terrenos sem se atrelar muito ao contexto de aplicação de vegetação e objetos para a geração de ecossistemas ou composição de cenas.

3.1 Introdução

Geralmente, terrenos são gerados através de um mapa de alturas pela CPU que são renderizados pela GPU, entretanto a natureza da CPU não é a mais adequada para se gerar terrenos complexos porque dependem de uma alta necessidade de paralelização de tarefas [25]. Além disso, mapas de altura não conseguem gerar características interessantes de terrenos como cavernas. Para se gerar terrenos com alta complexidade em tempo de execução, é muito interessante procurar uma abordagem utilizando diretamente a GPU.

Utilizando capacidades do DirectX10 como o *geometry shader[26], stream output[27]* e renderização para texturas 3D, é possível gerar blocos de terrenos complexos de forma extremamente rápida. A união destes blocos pode gerar um terreno extenso e extremamente detalhado. A figura a seguir demonstra um terreno criado inteiramente através da GPU utilizando o DirectX10.

Figura 12 Terreno gerado inteiramente na GPU

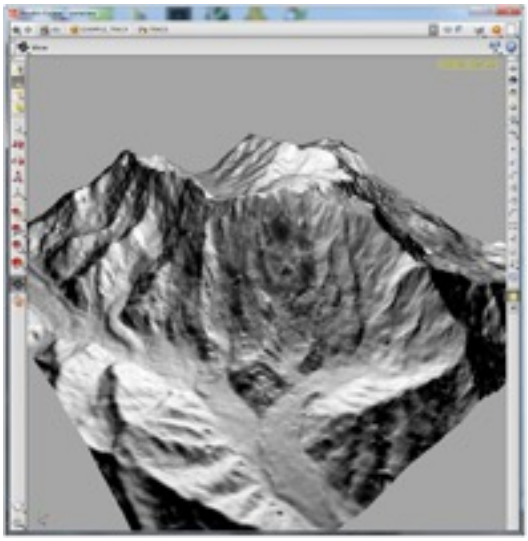


Com todas as possíveis variações e técnicas utilizando tanto a GPU quanto CPU, diversas ferramentas têm sido criadas por empresas para auxiliar desenvolvedores de jogos a terem seus terrenos gerados das mais variadas formas.

3.1 Ferramentas

A *Eletronic Arts* tem trabalhado com abordagens procedurais por um bom tempo utilizando profissionais extraídos da indústria de efeitos especiais [14]. Em 2011, no desenvolvimento de um *reboot* do jogo SSX, a EA (*Eletronic Arts*) passou a adotar um pensamento de utilizar técnicas procedurais para trabalhar no título. Por conta das ferramentas e técnicas desenvolvidas, a empresa conseguiu aumentar a quantidade de estágios disponíveis no jogo em mais de dez vezes [14] sem a necessidade de se fazer um recrutamento excessivo de novos funcionários para trabalhar no cronograma existente. Para o jogo em questão, a *Eletronic Arts* explorou uma ferramenta encontrada dentro da *Houdini* chamada *Mountain Main* que foi criada para gerar pistas de *snowboarding* através da utilização de dados mapeados por satélites extraídos de um projeto da NASA [14]. Uma imagem da ferramenta pode ser vista a seguir:

Figura 11 ***Mountain Man****, software de criação de montanhas utilizando dados de satélites*

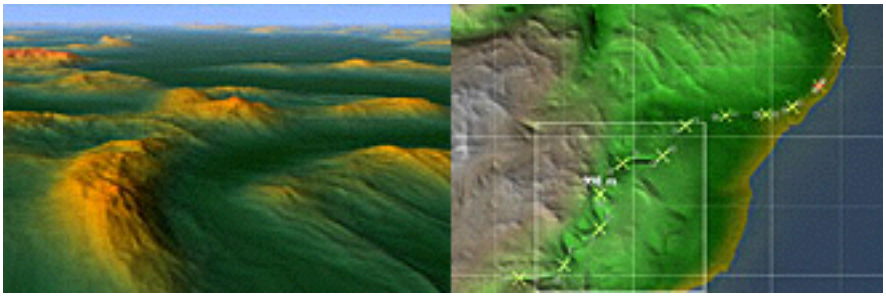


A ferramenta não só foi criada para gerar terrenos, e sim para gerar inteiros mapas para o jogo. As pistas de esqui eram também geradas dentro da *Mountain Man* e diversas regras eram aplicadas para tornar o mapa completamente jogável. A união desta e outras ferramentas possibilitou que a *Eletronic Arts* pudesse criar todos os estágios, incluindo conteúdo posicionado dentro dos mapas, para o título SSX em tempo recorde [28].

World Machine é uma ferramenta de criação de terrenos com simulações de fenômenos naturais e edição interativa que tem como objetivo a produção de terrenos realísticos rapidamente [29]. Diversos artistas, game designers e empresas de jogos como *Eletronic Arts*, *Microsoft Game Studios* e empresas de efeitos especiais tem utilizado esta ferramenta para a produção de seus trabalhos [30].

A natureza interativa da ferramenta *World Machine* permite que o usuário escolha a forma de se criar o mundo disponibilizando a possibilidade de se guiar a construção do *layout* do terreno através da pintura de mapas internamente na aplicação ou com a importação de arquivos externos como pode ser visto na figura 13 a seguir. É possível também deixar que a ferramenta crie inteiramente o terreno de forma procedural.

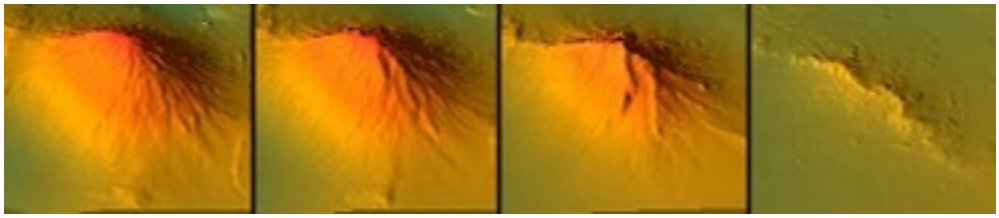
*Figura 13 Influência inserida (esquerda) e pintada (direita) utilizando a **World Machine***



World Machine utiliza um processo não destrutivo da utilização de uma rede de nós que criam os detalhes do terreno. É possível modificar um nó específico e ver a sua mudança afetando toda a rede em seguida. A ferramenta chama estes nós de *devices* que definem ações no terreno. Esta rede controla tudo do terreno, abrangendo a criação de cânions, montanhas com muita erosão, planícies e a junção de todos os elementos.

Esta rede de nós permite que a adição de elementos específicos possa ser controlada por um nó ou a união de alguns nós, a ideia é que com este processo o usuário possa criar lógicas que definem certos comportamentos e reutilizá-los em outros projetos ou partes de um mesmo projeto. Por conta deste tipo de *workflow* o usuário pode ver suas mudanças na rede em tempo real no terreno final como exemplificado na imagem abaixo.

*Figura 14 Visualização de mudanças em tempo real da **World Machine***



Uma das funcionalidades existentes na ferramenta é a da poderosa simulação de erosão. Processos como simulação de interação com vento, água e outros fenômenos caóticos permitem a criação de terrenos com um alto grau de semelhança a paisagens reais.

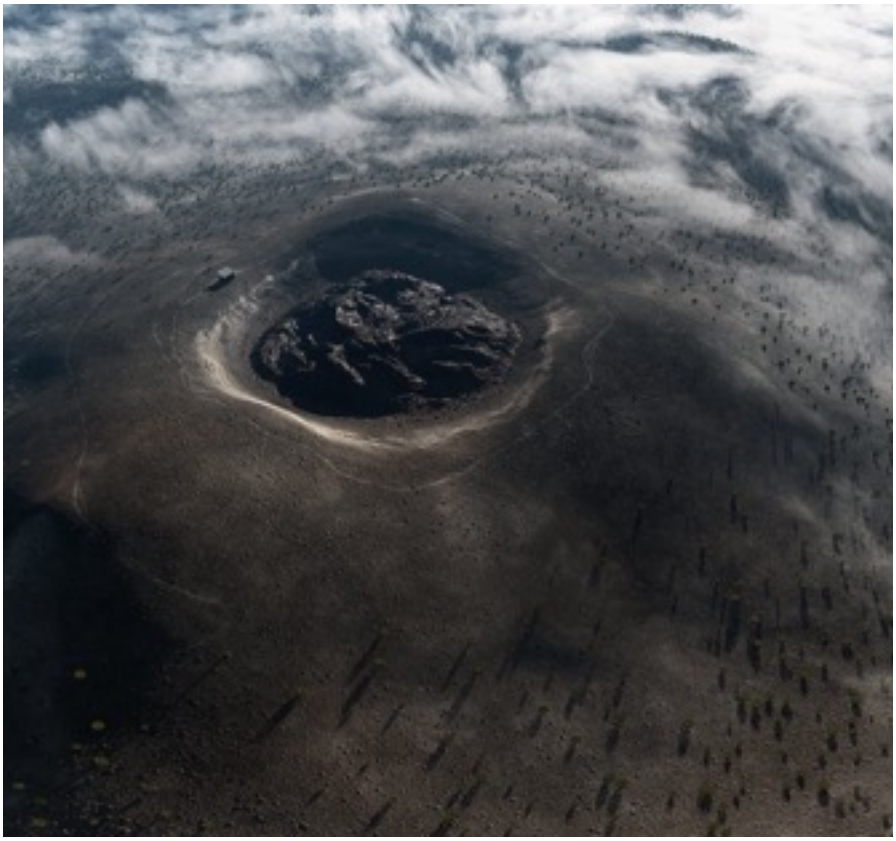
Modelagens de rios e simulações de precipitação permitem a simulação de milhões de anos de erosão de acordo com os desenvolvedores da ferramenta [31].

A ferramenta também permite a utilização de texturas para formações rochosas, grama, terra e erosão que adicionadas à possibilidade de que usuário pode também escolher simulações de iluminação e atmosfera é possível se criar uma pré-visualização do que pode ser o terreno final direto na *viewport* do programa, que pode ser visualizada tanto em uma perspectiva 3D quanto a partir de uma visão ortográfica para a visualização das texturas que serão exportadas.

Apesar da *World Machine* ser uma ferramenta muito complexa e com um vasto leque de funcionalidades, ela não está integrada em nenhuma ferramenta de renderização ou *engine* de jogos, tendo como necessidade suportar diversos modos de exportação dos mapas de detalhes, erosão e alturas o que limita algumas possíveis formações naturais como cavernas e arcos que terão que ser criados utilizando outra ferramenta para compor a cena.

Terragen (*Planetside software*) vem sido muito utilizado pelas indústrias do cinema e televisão. A ferramenta já foi utilizada em grandes títulos do cinema como *Star Wars: the force awakens*, *The Martian*, *Kingsman: The secret service*, *Ender's Game* e *Elysium* e da televisão como no documentário *Drain the Ocean* da *National Geographic Channel.* Alguns jogos têm sido criados também utilizando esta ferramenta, grandes exemplos são *Battlefield 1942* (DICE), *Battlefield Vietnam* (DICE) e *Serious Sam* (Croteam), *Terragen* funciona tanto como uma ferramenta para geração de terrenos como para a renderização realista dos terrenos gerados pelo *software* como pode ser visto na imagem abaixo.

*Figura 15 Terreno gerado e renderizado na **Terragen***

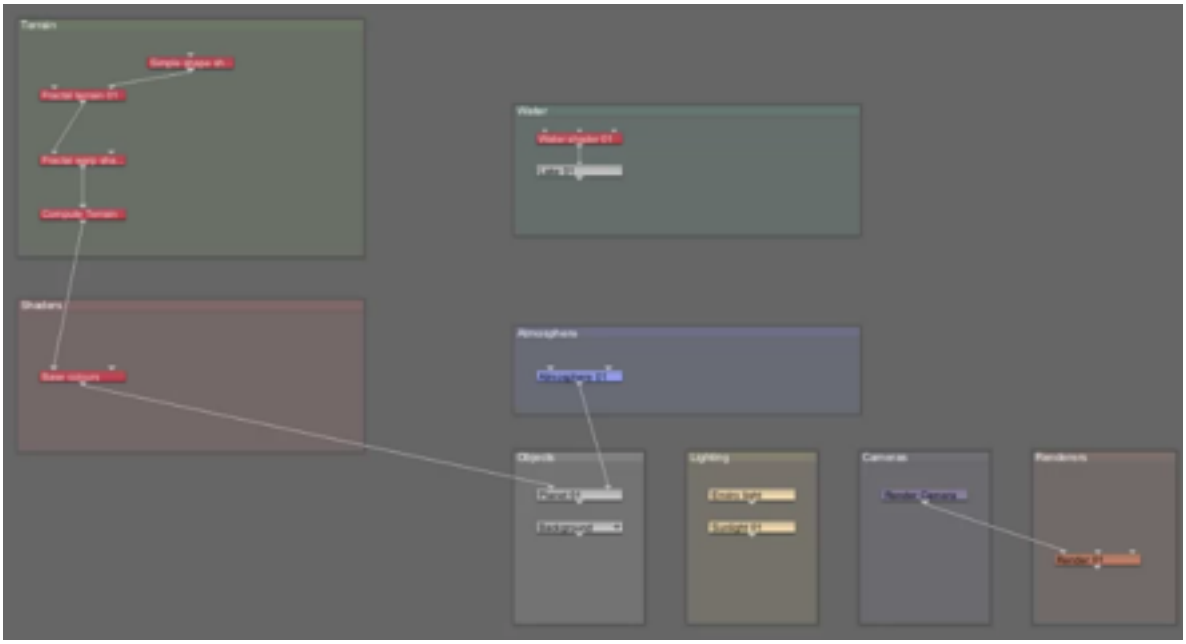


A criação de terrenos dentro da *Terragen* permite que o usuário cria e manipule os terrenos tanto utilizando mapas de alturas quanto funções procedurais. É possível importar mapas de alturas para replicação de terrenos reais ou protótipos de terrenos feitos em outras ferramentas. É possível unir vários tipos de terrenos em apenas um e utilizar as funções procedurais para gerar vários tipos de detalhes que podem variar desde pequenas pedras a montanhas gigantes. A ferramenta também trabalha em diversos níveis de escala, é possível se construir planetas e trabalhar na visualização da atmosfera com a criação de nuvens, quanto trabalhar em uma escala mínima ao ponto de se visualizar plantas pequenas com um terreno super detalhado ao redor [32].

Terragen funciona ao redor do terreno gerado por um mapa de alturas, é possível se editar os parâmetros que regem os fractais e ruídos responsáveis pela criação do mapa de alturas inicial, após a inserção do mapa de alturas, é possível se trabalhar na adição e

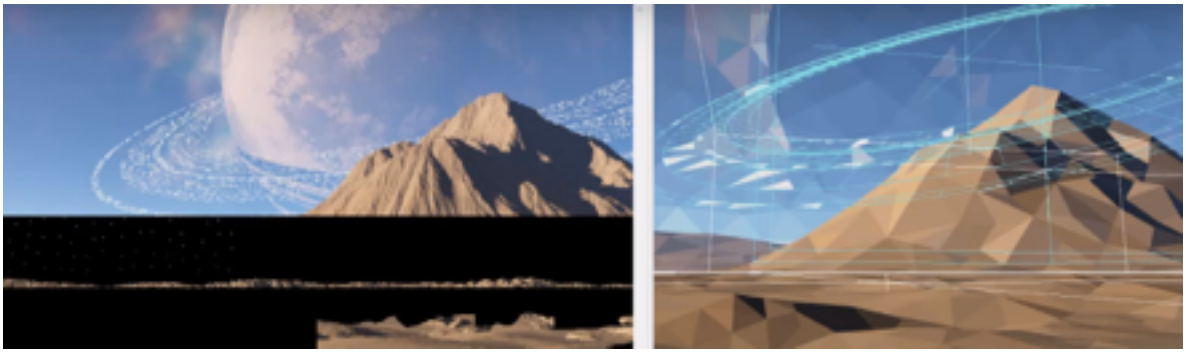
remoção de mais detalhes com a simulação de fractais e utilização de *displacement maps* [33]. Esta ferramenta também funciona utilizando o processo de rede de nós para se trabalhar na formação de detalhes procedurais de forma não destrutiva. Por conta da utilização deste *workflow* é possível se criar terrenos mutifractais por apenas conectar nós na rede de geração do terreno como pode ser visto na figura 16.

Figure 16 Rede de nós de geração de Terreno e Shaders no Terragen



Contrária ao *World Machine*, *Terragen,* possui uma pré-visualização do terreno em tempo real, mas necessita de um renderizador para mostrar o resultado final de todos os detalhes na cena, um exemplo da diferença entre a pré-visualização e o processo de renderização em andamento pode ser visto na imagem a seguir.

Figura 17 Pré-visualização e Processo de renderização em andamento da Terragen



Assim como a *World Machine, Terragen* também funciona sem integração com *engines* de jogos atualmente e por conta disso, para a utilização destes terrenos gerados dentro da ferramenta, é necessário que os mapas de altura, máscaras e texturas sejam exportados para uso dentro de *engines* como *Unity 5* e *Unreal Engine 4*.

VUE (e-on software) é uma outra ferramenta de geração de terrenos muito utilizada na indústria de jogos e cinema. Diversos títulos da indústria cinematográfica utilizaram esta ferramenta, como: Jogos Vorazes, Os vingadores, Como treinar o seu Dragão, G.I. Joe, Nárnia e as Aventuras de TinTin[34]. Apesar da VUE não ser diretamente feita para suportar jogos (assim como *World Machine* e *Terragen*), ela possui as mesmas funcionalidades das demais de exportação de mapas para utilização dentro de *engines*.

A VUE trabalha na preparação para render de terrenos infinitos de forma rápida e escalável, abrangendo todas as escalas que variam desde uma visualização próxima a planetária. Uma funcionalidade chave da plataforma é a da tecnologia Solid3D (e-on Software) que permite esculpir terrenos em tempo real utilizando pincéis 3D para modelar o terreno para criar qualquer forma desejável [35]. Com essa ferramenta é possível se criar cavernas, rios, Cânions, e alguns tipos de topologia impossíveis de se alcançar com o uso de mapas de alturas apenas. É possível também personalizar o comportamento dos pincéis que serão utilizados para esculpir o terreno permitindo a reutilização configurações para outros projetos.

A tecnologia Solid3D também possui algoritmos para subdivisões locais, permitindo que o usuário da ferramenta consiga criar detalhes em regiões específicas sem a necessidade de subdivisão e aumento da quantidade de polígonos em todo o terreno para adicionar detalhe em uma pequena parte. Isso permite que artistas se preocupem mais na parte criativa sem ter que se importar muito com detalhes técnicos que possam afetar o terreno no geral[35]. A imagem a seguir demonstra o processo citado acima, apenas uma região do terreno rochoso possui detalhe o suficiente para terem rachaduras.

Figura 18 Subdivisões localizadas da tecnologia ***Solid3D***

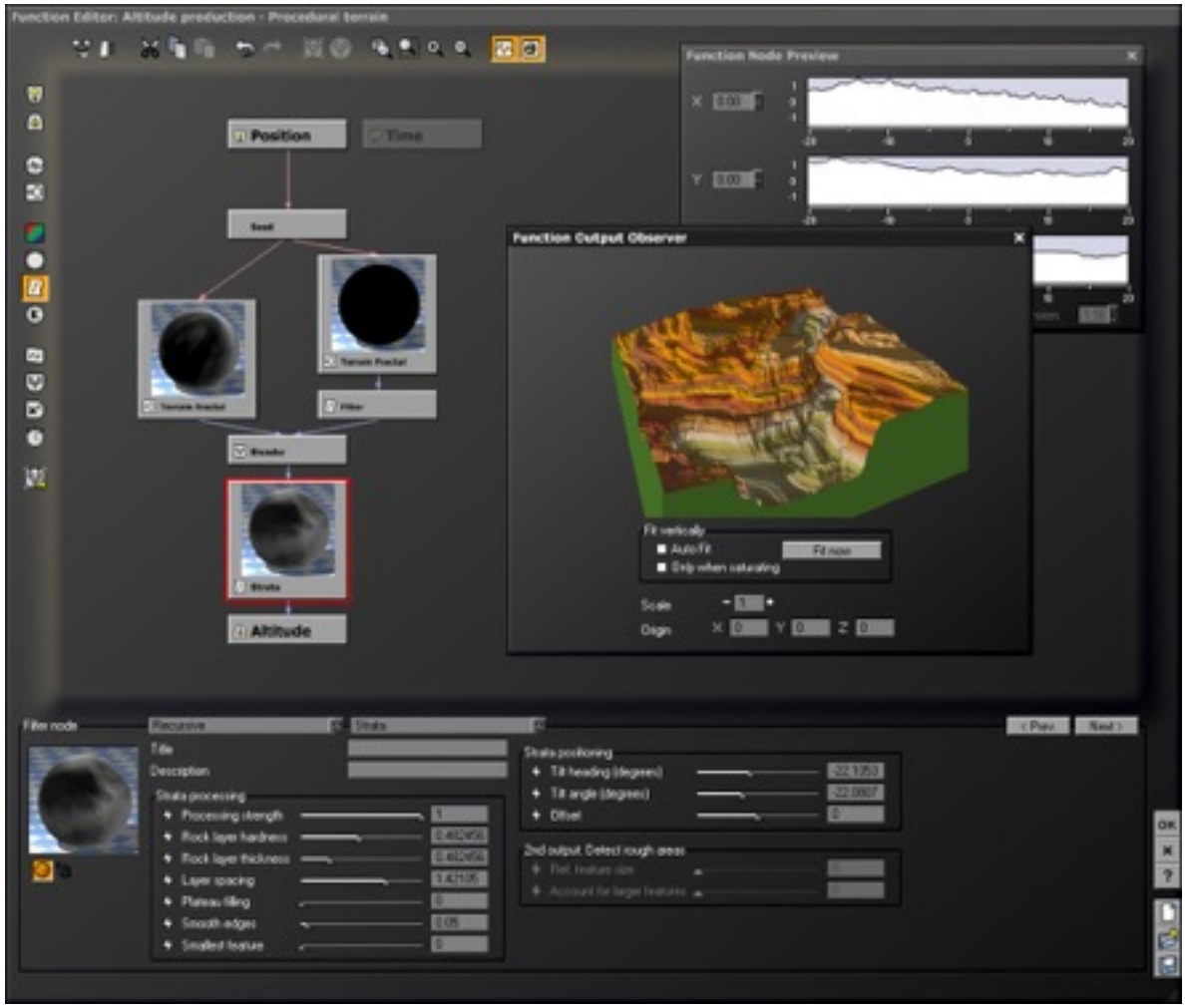
Esculpir é um processo não destrutivo dentro da VUE e funciona junto com a geração procedural infinita. Também é possível esculpir estradas, passarelas e cavernas. É possível também utilizar um carimbo para adicionar detalhes específicos. Uma funcionalidade que auxilia bastante o processo de esculpir e resolução de detalhes é o da retopologia automática, essa técnica recria a topologia da malha otimizando a distribuição dos polígonos de acordo com a necessidade. *Splines* e efeitos também podem ser utilizados na malha para se criar detalhes específicos, como por exemplo uma cordilheira que segue um caminho.

VUE utiliza o mesmo *workflow* não destrutivo de uma rede de nós para a criação do terreno que permite uma edição profunda utilizando um vasto leque de ruídos, fractais e funções matemáticas que definem fenômenos naturais e geológicos, a interface para a

geração do terreno pode ser vista na figura 19. Também é possível extrair zonas específicas de um terreno gerado e trabalhar especificamente nele, utilizando máscaras e processos de textura que podem ser reutilizados em outras regiões.

VUE também é uma ferramenta poderosíssima de renderização e possui funcionalidades atreladas a *shaders*, materiais e texturas que podem criar um visual realista através da sobreposição de materiais, *constraints* para materiais baseadas em ângulos, altitudes, detalhes, mudanças de direção da malha.

Figura 19 Interface de criação de nós da VUE



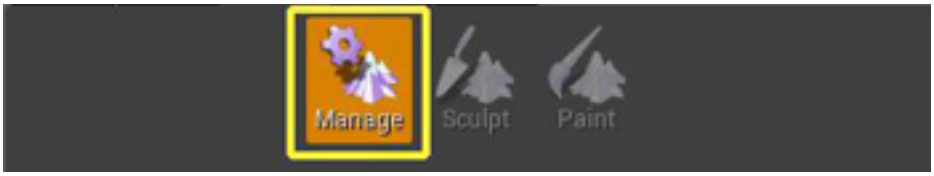
Assim como as demais ferramentas listadas até agora, a VUE não possui uma forma de se trabalhar com as *game engines* atuais. A exportação do terreno e dos mapas é possível e pode ser utilizado dentro da Unreal Engine 4 e Unity 5 por exemplo, mas a não integração com a ferramenta de terreno das *engines* pode causar uma perda de performance caso o terreno não tenha sido preparado com o objetivo de ser usado em jogos. Entretanto, a VUE possui plug-ins para algumas ferramentas de edição e criação de conteúdo 3D como o Maya (Autodesk) e 3DS Max (Autodesk) que permitem uma melhor interação entre ferramentas

3.2 Game Engines Level Editors

A *Unreal Engine* 4 (*Epic Games*) possui sem próprio criador de terrenos embutido *engine*. Com o objetivo de ajudar desenvolvedores a criarem terrenos dentro da própria *engine* a Epic Games criou um editor em tempo real que utiliza o mesmo processo de criação das demais etapas dos jogos dentro da UE4. A ferramenta chamada de *Landscape Tool*, é capaz de criar terrenos massivos que, integrado a sua *engine*, permite que a otimização do terreno seja feita de forma apropriada em um vasto leque de *devices* suportados [36]. Por conta da flexibilidade da ferramenta o usuário é capaz de criar um terreno do zero utilizando a coleção de pincéis e modos de pintura ou importar um mapa de alturas gerados em ferramentas externas, como as citadas na seção 3.1 deste trabalho.

A ferramenta de *Landscape* possui alguns modos de visualização e criação que podem ser vistos na figura 20. Cada modo é responsável por uma característica do terreno, sendo o primeiro modo o da geração das propriedades iniciais do terreno, como tamanho, resolução e subdivisões, o segundo é responsável por disponibilizar todas as ferramentas de escultura de terrenos e, por último, a de pintura e sobreposição de materiais diferentes.

*Figura 20 Modos de gerenciamento da ferramenta **Landscape** da **Unreal Engine 4***



Na primeira aba (gerenciamento) algumas configurações podem afetar diretamente a experiência do jogador. As opções que envolvem resolução e quantidade de segmentos e secções lida diretamente com a quantidade de vértices e faces que estarão disponíveis naquele terreno, esta quantidade de faces associada com uma população de objetos para compor a cena, caso não sejam otimizados, podem provocar quedas de *framerate* e enormes tempos de compilação do terreno.

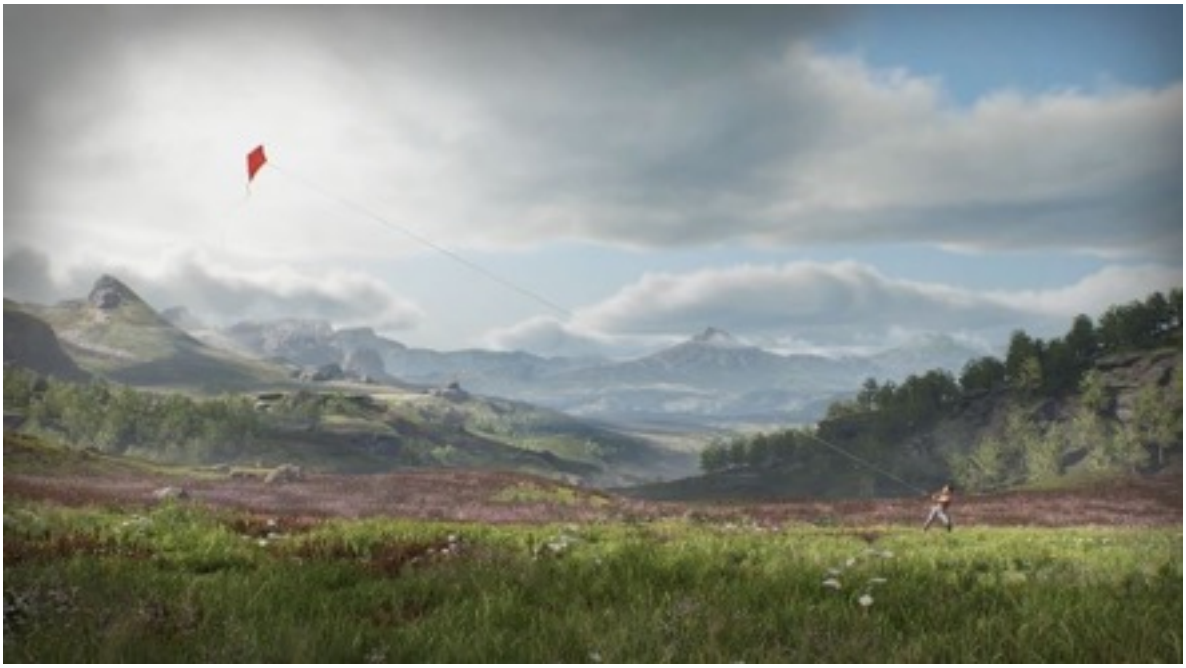
O processo de esculpir o terreno é simples e intuitivo. O usuário tem algumas ferramentas que podem ser utilizadas para manipular o terreno e criar as superfícies como desejado. Um pequeno problema é a da falta de métodos procedurais para se gerar um terreno inicial que o usuário possa trabalhar por cima. Atualmente, a ferramenta de *Landscape* só possui suporte para a criação e edição manual do terreno sem nenhum suporte para o uso de fractais ou ruídos que possam auxiliar na criação de detalhes de erosão e de adição de realismo de forma automática.

Apesar da falta de processos automáticos de manipulação do terreno algumas das ferramentas disponíveis na plataforma permitem a criação de aleatoriedade e erosão. Uma das ferramentas disponíveis no processo de escultura é o de erosão, a ferramenta possui algumas propriedades que podem ser modificadas para se criar tipos diferentes de fenômenos através de simulações de erosões termais. Existe também a ferramenta de Erosão hídrica, que é responsável por criar simulações de erosão causadas por chuvas. Uma outra ferramenta é a de ruído, na qual o usuário pode adicionar ruído em um local específico utilizando o pincel desejado, as configurações destas ferramentas na ordem apresentada podem ser vistas na imagem a seguir. Estas ferramentas auxiliam o processo artístico, permitindo que usuários possam alcançar resultados mais realistas, apesar da falta da geração automática destes detalhes.

Figura 21 Propriedades das ferramentas de erosão e ruído

A *Unreal Engine* 4 é uma *engine* de jogos e por conta disto possui a característica de renderização em tempo real. Isso faz com que o usuário veja, a todo tempo, como o terreno estará visível para o jogador em tempo de execução, um problema associado a esta abordagem é que a edição pode se tornar lenta caso a cena comece a se tornar complexa e custosa para o computador na qual a *engine* está rodando. Para compor a cena, a UE4 também possui um esquema para a criação de materiais que podem ser aplicados ao terreno na forma de pintura. Cada material é feito utilizando uma rede de nós para a criação do material final, e o terreno tem um esquema na qual o usuário pode construir definições de como os materiais para terrenos interagem entre si na superfície. Diversos jogos feitos na UE4 têm tirado proveito desta ferramenta poderosa. O terreno que pode ser visto na imagem a seguir foi criado pela *Epic Games* utilizando inteiramente a ferramenta de *Landscape* para divulgar a liberação da *Unreal Engine* 4 de forma gratuita. Diversos títulos têm utilizado a Unreal Engine para sua produção como os jogos da Série Bioshock, Gears of Wars, Fornite, Kingdom Hearts III e o mais novo título da Epic Games: Paragon.

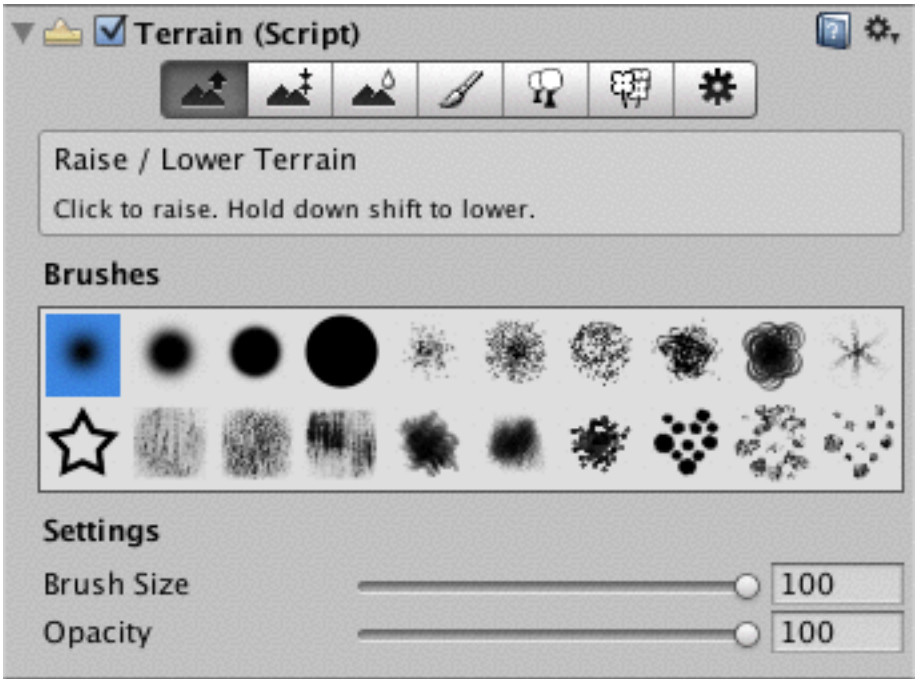
Figura 22 Terreno do curta "A boy with a kite" feito pela Epic Games



A *Unity* 5 também possui uma ferramenta de criação e edição de terrenos com objetivos semelhantes aos da *Unreal Engine* 4. Por conta da Unity ser uma *engine* para jogos, toda a geração do terreno é feita de forma otimizada para que não existam quedas de *framerate* para os jogadores enquanto navegam ou visualizam os terrenos gerados dentro da ferramenta.

A ferramenta *Terrain* da *Unity* cria uma malha que pode ser manipulada através da utilização de diversos pincéis que alteram diretamente o mapa de alturas do terreno. Uma figura da interface de seleção de pincéis e modificação de seus parâmetros pode ser visto na imagem abaixo.

*Figura 23 Parâmetros de pincéis utilizáveis para edição de terrenos da **Unity 5***



Os pincéis para edição de alturas permitem que o usuário crie mudanças nas alturas de forma artística. Quando se usa o pincel para pintar no terreno, a área contida dentro do pincel terá um acréscimo ou decréscimo na altura de acordo com as propriedades selecionadas par ao pincel. Com essas ferramentas é possível construir montanhas, vales, cânions e também suavizar o terreno para criar transições suaves entre regiões. Unity 5 também possui uma forma de importação de mapa de alturas em tons de cinza que permite a transformação da textura importada em um terreno que a representa de forma fiel, desta forma, usuários podem até utilizar imagens de satélite para a criação de seus terrenos.

Diferentemente da Unreal Engine 4, Unity 5 não trabalha com uma rede de nós que produz materiais e texturas finais. A ferramenta Unity funciona utilizando a programação de *shaders* e utilização de texturas para se gerar a parte de renderização. Sendo assim, é possível se criar camadas e aplicar texturas específicas a cada camada que podem ser pintadas diretamente no terreno, um problema associado a esta técnica é que as mudanças devem ser feitas diretamente na textura através de uma ferramenta externa ou *scripts* que rodam dentro da Unity 5. Um segundo problema associado a este processo de texturização

é de que é necessário o aprendizado de uma linguagem de programação específica de *shaders* utilizada pela Unity 5 para se criar materiais complexos para Terrenos em que, por conta de erros de compilação serem fáceis de se cometer, visualizar as mudanças feitas no código no terreno final pode ser custoso e demorado [37].

Muitos títulos têm sido criados utilizando a Unity 5, como por exemplo Kerbal Space Program, Cities Skylines, Firewatch, Gang Beasts e Broforce.

Oposto a Unreal Engine 4, a *Unity* 5 não possui métodos de geração e simulação de erosão ou fenômenos naturais nem a modificação de terrenos através do uso de ruídos. Entretanto, diversas empresas criaram plug-ins e ferramentas que podem ser instaladas diretamente na Unity 5 que podem suprir as necessidades da ferramenta padrão de criação de terrenos.

World Creator (BiteTheBytes) é uma ferramenta externa com uma versão para a Unity 5 que permite a adição de gerações procedurais para a lógica de terreno. Com esta ferramenta é possível se criar terrenos com o uso de um rico leque de ferramentas disponíveis todas integradas diretamente com a *Unity 5 Game Engine*. A ferramenta conta com diversos filtros para geração de fenômenos naturais como simulações de erosões e utilização de ruídos para geração de aleatoriedade nas superfícies. A ferramenta também possui diversos filtros para gerações de características específicas de terrenos baseados na realidade, como a produção de desertos, cânions, montanhas e planícies.

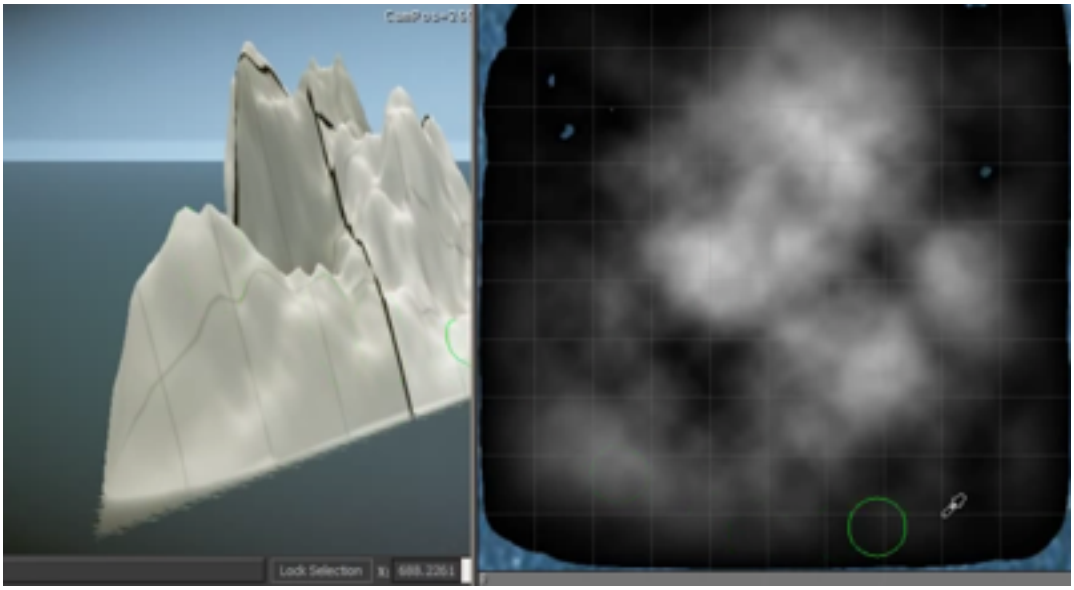
Apesar da não visualização em tempo real do terreno sendo gerado na versão padrão da ferramenta, é possível ter essa funcionalidade na versão *pro* da ferramenta. Além da adição de algumas outras funcionalidades, como a que permite a união de diversos mapas com resoluções diferentes para se criar um só terreno com detalhes diferentes. Esta ferramenta também tem a sua versão *standalone* com uma interface gráfica que não depende da Unity 5. Alguns títulos têm utilizado a *World Creator* para criar os terrenos dos seus jogos por ela fornecer um maior controle e ferramentas de edição do terreno com uma maior variedade.

A interface da ferramenta acoplada à Unity 5 pode ser vista na imagem a seguir

Figure 24 ***World Creator UI***

CryEngine, diferentemente das outras *engines* citadas até então, possui a capacidade de gerar um terreno inicial de forma procedural. Apesar dos parâmetros para a geração procedural não ser muito rica, ela permite a criação de um terreno inicial de forma rápida e interessante que se adequa a resolução escolhida. O resultado da geração procedural é exportado para uma textura em tons de cinza que serão lidos pela *engine* como um mapa de alturas, um exemplo desta geração pode ser visto na imagem a seguir:

Figure 25 ***CryEngine****, terreno gerado a partir de um mapa de alturas procedural*



Mesmo com sua abordagem procedural para *layout* inicial também é possível se importar mapas de alturas para a sua geração. Após este primeiro passo, o usuário pode utilizar alguns pincéis para a modificação das alturas e criação de superfícies de uma forma mais artística, assim como nos outros editores acima. Diferente das outras *engines*, todos os pincéis na *CryEngine* podem ter um ruído associado para criar uma dinamicidade na forma geral em tempo de execução[38].

Um conceito interessante é de que a ferramenta de criação e edição de terrenos da Cryengine possui a possibilidade de se criar terrenos a partir de *Voxels*, um conceito que vem sido muito explorado em jogos do gênero *sandbox*. Uma outra ferramenta interessante é a de cavar buracos, assim como na *Unreal Engine* 4, essa ferramenta de buracos possibilita que usuários criem buracos no seu terreno para a inserção de outros objetos, como um sistema de cavernas. Um exemplo de um buraco inserido em um terreno na *CryEngine* 3 pode ser visto na imagem 26 a seguir.

Figura 26 Buraco inserido na malha do terreno através da ferramenta de buracos da CryEngine

A *CryEngine* foi apresentada pela primeira vez através do jogo Far Cry (2004) que trouxe gráficos deslumbrantes de ilhas paradisíacas e trouxe um ar fresco para o gênero de tiro em primeira pessoa com os seus terrenos impressionantes. A *engine* foi depois aprimorada e mostrou o seu potencial mais uma vez com o jogo Crysis (2007) que foi um jogo com gráficos impressionantes e muito bem otimizados, o título foi utilizado como benchmark para comparação entre componentes de computadores por muitos anos.

3.3 Resumo

Apesar de todas as ferramentas citadas acima serem capazes de produzir terrenos super-realistas com a utilização de diversas ferramentas, não existe nenhuma ferramenta que trabalhe biomas e ecossistemas. Todas elas possuem alguma forma de se posicionar vegetação, árvores, pedras utilizando estratégias diferentes. A *Vue*, por exemplo, é capaz de gerar um inteiro ecossistema de florestas utilizando ruídos como máscaras de posicionamento das árvores, e algumas ferramentas, como a *Terragen* é capaz de simular o crescimento de árvores.

A inclusão de biomas em uma ferramenta de geração procedural de terrenos tem benefícios associados à agilidade de prototipação de um terreno através da possibilidade de construção mais acelerada de biomas (áreas ecológicas com fauna e flora específica) quanto pela possibilidade de parametrização de regiões para a aplicação de texturas, objetos e efeitos que estejam conectados a regras de jogabilidade impostas pelos desenvolvedores de jogos.

Por conta da falta de ferramentas que trabalhem a criação de forma procedural de biomas, este trabalho terá como objetivo incorporar esta funcionalidade junto com a geração de terrenos de forma procedural. Esta implementação pode ser vista no próximo capítulo.

Este capítulo listou algumas das ferramentas disponíveis para o público de graça ou através da aquisição de uma licença para uso utilizadas hoje em dia na criação de jogos. Inicialmente foi listado algumas das ferramentas utilizadas na indústria de jogos e cinema que funcionam separadas de qualquer *engine* para jogos, mas que possuem processos de exportação de mapas que possam ser usados na maioria das *game engines.* As ferramentas apresentadas foram a Mountain Man, Terragen, VUE e World Machine.

A segunda parte desta seção trabalhou em demonstrar o poder dos editores de terreno existentes dentro das *game engines* da atualidade. *Unity 5*, *Unreal Engine 4* e *CryEngine* foram as *engines* apresentadas nesta seção e um pequeno detalhamento de como suas ferramentas funcionam foi feito.

Com o estudo e entendimento destas ferramentas, é possível ter uma ideia do que é necessário se ter em uma ferramenta de criação de terrenos para jogos. A próxima seção deste trabalho apresentará como um estudo foi feito para identificar quais ferramentas são interessantes dentro destas apresentadas e como é possível fazer a combinação delas para se criar um protótipo de uma ferramenta que trabalhe mais os conceitos procedurais para a geração de terrenos e biomas.

4. FERRAMENTA DE CONSTRUÇÃO DE TERRENOS PROCEDURAIS

A ferramenta criada para este projeto partiu de um estudo feito com pessoas ligadas a área de produção de jogos na atualidade. O objetivo inicial deste trabalho é o de priorizar uma lista de funcionalidades que possam existir ou serem incorporadas em uma ferramenta já existente para a criação, parametrização e interação com os parâmetros responsáveis pela geração de Biomas.

4.1 Funcionalidades essenciais para a ferramenta

A primeira necessidade deste trabalho é de entender quais funcionalidades uma ferramenta de geração procedural de terrenos deve ter. De acordo com o estudo feito e apresentado no capítulo anterior sobre ferramentas externas, algumas das funcionalidades foram listadas como necessárias para a continuação deste trabalho.

Estas funcionalidades são fundamentais para que conceitos relacionados a geração e modificação de biomas de forma procedural possa acontecer, pelo fato de que se deve existir um terreno para que os biomas possam ser projetados nele. É importante que este terreno possa ser manipulado por uma nova geração de parâmetros ou pela edição manual do mapa para que resultados diferentes possam ser explorados assim como nas ferramentas estudadas.

Idealmente, o usuário deve ser capaz de escolher valores para os parâmetros e que estes valores estejam sempre associados a mesma saída para que os detalhes alcançados possam ser repetidos em diferentes terrenos, o conjunto destes valores formará a semente da geração procedural.

Também é ideal que a ferramenta tenha uma integração direta com uma interface gráfica que possa ter o terreno gerado validado em tempo real pelos usuários. Sendo assim, a implementação deste trabalho levará em conta a possibilidade de implementação direta dentro de alguma *engine* para jogos ou extensão de uma ferramenta externa já existente.

4.2 Funcionalidades desejáveis na ferramenta

As funcionalidades citadas na seção anteriores não são suficientes para se criar um terreno natural e realista. Apenas a possibilidade de se criar ruídos e da utilização de técnicas multifractais podem não ser suficientes para se criar um terreno de alta fidelidade, como explicado nos capítulos 2 e 3 um grande fator para o realismo de terrenos gerados de forma procedural é o da possibilidade de simulações de fenômenos naturais que criam características específicas em terrenos como os causados por erosões termais, hídricas e eólicas.

Um outro grande fator tanto para a geração de terrenos mais reais quanto para a melhor representação de biomas e ecossistemas é o da possibilidade de inserção de vegetação. Todas as ferramentas listadas na seção anterior deste trabalho possuem alguma técnica para a inserção de vegetação, seja ela interna às *engines* quanto as ferramentas externas.

Sendo assim, algumas características e ferramentas podem ser adicionadas para uma melhor versão de uma ferramenta de geração de terrenos:

1. Adição de filtros de simulação de fenômenos naturais.
2. Adição de subdivisões locais para melhor detalhamento de áreas específicas.
3. Adição de filtros para gerações de características específicas como cordilheiras, cânions e rachaduras.
4. Sobreposição de terrenos com informações diferentes para criar terrenos mais ricos.
5. Posicionamento de características do terreno de acordo com Splines.
6. Posicionamento de vegetação e objetos de forma procedural de acordo com as regiões de biomas.
7. Posicionamento de vegetação e objetos de forma manual.
8. Modificação artística da delimitação dos biomas.

4.3 Priorização de funcionalidades

Por conta do tempo disponível e pelo objetivo final deste trabalho, que é o de fornecer uma possível ferramenta para a geração procedural e edição de biomas, uma listagem priorizada foi estabelecida para garantir a evolução deste projeto de acordo com o cronograma estabelecido. A listagem definida nesta seção servirá como um sumário para a seção seguinte que explicará como cada etapa da ferramenta foi escrita e quais variações dela foram validadas. Para a atividade de priorização, um grupo de estudantes e profissionais foi formado para a validação deste trabalho. A lista priorizada pode ser vista a seguir:

1. Geração de mapas de alturas de forma procedural com ajustes de parâmetros.
2. Escultura do terreno (Edição do mapa de alturas de forma manual)
3. Possibilidade de alteração de resolução e tamanho do terreno.
4. Possibilidade da adição de cores e texturas no terreno gerado.
5. Adição de filtros de simulação de efeitos naturais (erosão e características específicas de terrenos)
6. Posicionamento de vegetação e objetos de forma manual
7. Posicionamento de vegetação e objetos de forma procedural (de acordo com um mapa de biomas ou ecossistemas)
8. Modificação artística de biomas e ecossistemas
9. Subdivisões locais para detalhamento local.
10. Sobreposição de terrenos com informações diferentes.

Esta listagem foi priorizada de acordo com a comparação das listas priorizadas de cada indivíduo do grupo, totalizando seis listagens (incluindo a do autor deste trabalho). A implementação de cada uma das funcionalidades será descrita na próxima seção e é

importante entender que o desenvolvimento de uma funcionalidade específica facilita o desenvolvimento de uma outra que pode estar em um ponto diferente da lista priorizada e por isso seja desenvolvida em conjunto. As funcionalidades serão desenvolvidas dentro da janela de tempo existente de acordo com o cronograma estabelecido na proposta deste trabalho.

4.4 Implementação da ferramenta

Esta seção descreverá todo o processo de implementação da ferramenta, detalhando cada funcionalidade e explicando os algoritmos utilizados em cada etapa, além do relatório de todas as iterações feitas sobre a sua validação até o momento. A ferramenta passou por mudanças na interface dentro da plataforma definida para acoplar novas funcionalidades de acordo com a introdução de cada uma e um processo iterativo com o mesmo grupo utilizado na tarefa de priorização das funcionalidades.

4.4.1 Definição de plataforma

Para se criar a ferramenta com as funcionalidades listadas na seção anterior é necessário a escolha da plataforma na qual a geração de terrenos será construída. *VUE* e *Terragen*, atualmente, não possuem nenhuma forma de se criar extensões ou plug-ins para trabalhar com as funcionalidades através de uma *API* ou *SDK*. A *World Machine*, em contrapartida, possui a habilidade de adição de plug-ins escritos em C++ através do uso do pacote PDK oferecido pela própria empresa criadora da *World Machine*. *Unity 5* e *Unreal Engine 4*, por outro lado, são completamente abertas para a os usuários explorarem diversas funcionalidades e tipos de códigos por conta de serem *game engines* e é possível se construir plug-ins ou até mesmo estender o código fonte para que a *engine* possa incorporar as funcionalidades desejadas.

Por conta destas facilidades apresentadas pelas *engines* de jogos atuais, somando com a quantidade de material disponível de ensino sobre o uso de suas APIs, foi decidido a utilização da *Unity 5* como plataforma na qual a ferramenta proposta neste trabalho será

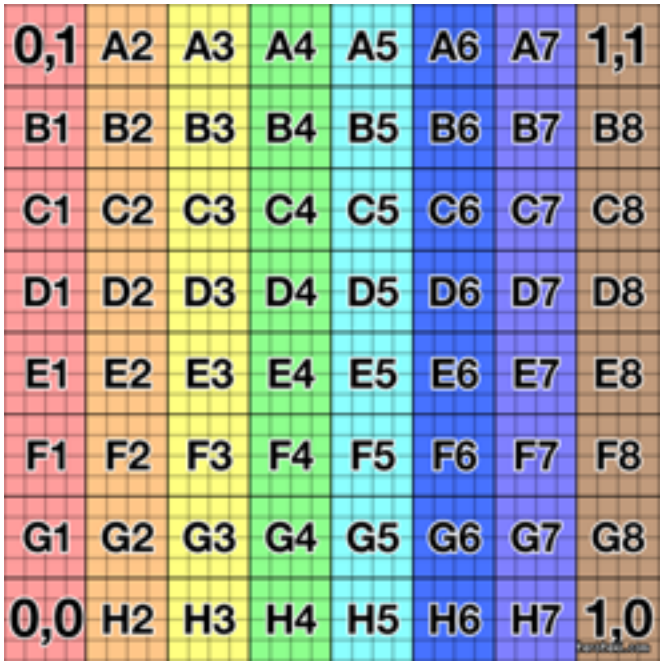
utilizada. A *Unity 5*, contrária a *Unreal Engine 4*, não só tem todas as facilidades listadas como é uma ferramenta aberta ao público desde 2005 e vem sido muito explorada desde então pela indústria de jogos. Sendo assim, a ferramenta proposta neste trabalho será escrita completamente em C# utilizando a API da *Unity 5* e a sua linguagem de escrita de *shaders*. A *Unreal Engine 4* se tornou gratuita a pouco tempo e, por conta disso, possui pouquíssimo material disponível pela comunidade quando comparada a Unity 5, a Cry Engine, pelo motivo de ser paga, também passa pela mesma situação e por isso não foram consideradas para a criação da ferramenta.

4.4.2 Implementação do gerador de ruídos

O core da ferramenta depende de um gerador de ruídos que possa ser utilizado de forma modular pela plataforma. A característica modular permite que futuras versões da plataforma insiram novos tipos de ruídos ou modifiquem os atuais para se atingir resultados diferentes na geração do terreno.

Toda a geração de alturas do terreno dependerá, inicialmente, da leitura de um mapa de alturas que será alimentada pelo gerador de ruídos através da criação de texturas. Uma textura, como explicada no capítulo 2 deste trabalho, existe dentro de um espaço bidimensional expressado através de coordenadas UVs que pode ser visualizado na imagem a seguir.

Figura 27 Textura de tabuleiro projetada em coordendas UVs

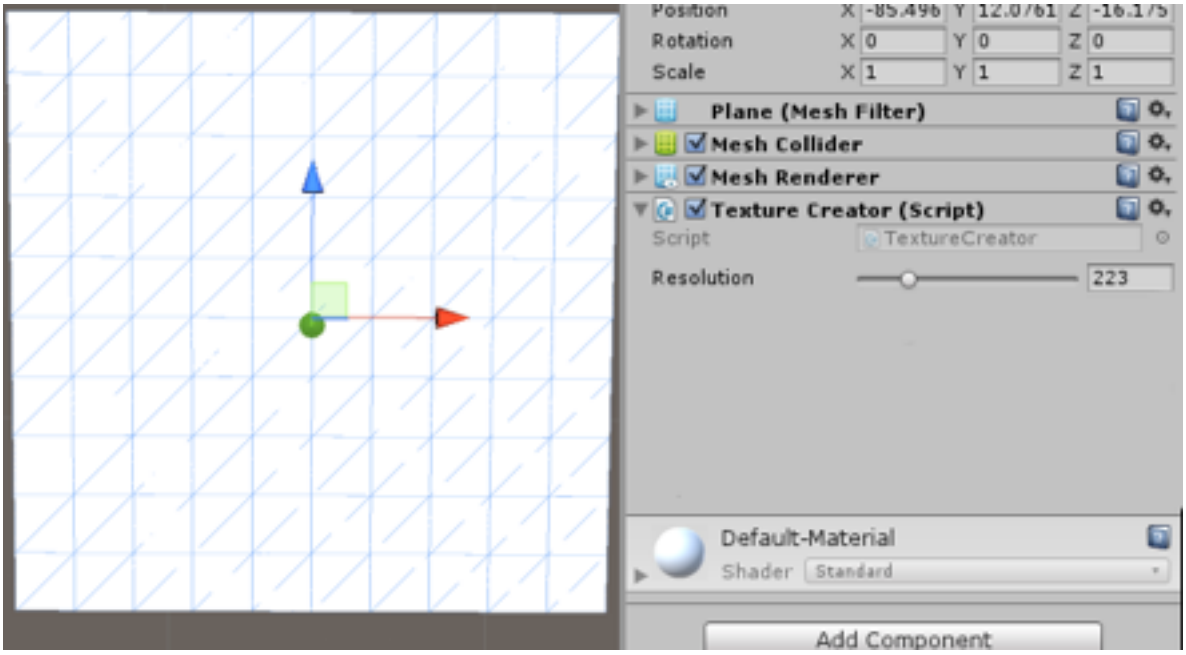


Essa textura tem uma estrutura muito comum utilizada na indústria que é a de se ter um tabuleiro igualmente espaçado e distribuído através das coordenadas UVs. Uma textura como esta ajuda a detecção de projeções erradas e distorções de forma visual. Importante notar que as coordenadas U e V estão normalizadas em um intervalo de 0 a 1, sendo (0,0) o ponto inferior esquerdo e (1,1) o ponto superior direito. Esta mesma orientação é utilizada dentro da Unity 5 e este conhecimento é primordial na criação de texturas.

A API da *Unity* disponibiliza uma forma simples de se criar uma textura através da utilização da classe Texture2D. Com esta classe, é possível se criar a representação de uma textura com uma largura e altura específica que podem ser passadas como parâmetros no seu construtor. Sendo assim, é interessante que o usuário tenha controle sobre a resolução da textura final, uma vez que texturas maiores possuem uma possibilidade de expressão de detalhes maior mas trazem um custo computacional adequado. Sendo assim, a funcionalidade de resolução de terrenos estará atrelada a resolução da textura gerada pelo gerador de ruídos, já que a amostragem das alturas do terreno partirá desta textura.

Para se criar uma textura e ver o progresso de sua criação, um plano foi criado com o *shader* padrão da *Unity* e com uma textura que será modificada em tempo real pelo script de geração de ruídos, como pode ser visto na imagem a seguir.

Figura 28 Parâmetro de resolução do gerador de ruídos



O parâmetro de resolução (atualmente limitado no range de 2 a 1024) é responsável pela criação de objetos da classe **Texture2D** de resolução adequada ao valor selecionado. Sendo assim, uma resolução de 1024 produzirá uma imagem com 1024*1024 pixels, permitindo que possamos iterar sobre cada pixel e adicionar uma cor específica usando a função **setPixels** da API. É importante lembrar que como as coordenadas de cada pixel se encontra em um espaço normalizado, para uma resolução de 1024 cada pixel terá uma sua coordenada dentro deste espaço com uma distância relativa de 1/1024. Nos resta apenas entender como criar um ruído e aplicar sua lógica a cada pixel da textura desejada e teremos uma geração de texturas de forma procedural.

Para evitar que o resultado da textura não tenha uma transição suave entre os pixels para resoluções baixas, a textura será criada com o filtro **Trilinear** automaticamente através do uso da API da Unity.

Inicialmente dois tipos de ruídos serão tratados, *Value noise* e *Perlin noise*. *Value noise* possui uma lógica simples de implementação que é a de usar um número aleatório para cada posição fornecido por um gerador de números aleatórios e este valor é então interpolado baseando-se nos valores dos pixels vizinhos. Sendo assim, este trabalho criou um gerador de *ruído* que particiona o espaço ocupado pelo plano e traduz a coordenada de cada pixel a um valor aleatório, isso faz com que exista a necessidade da leitura de cada ponto no mundo, podendo-se extrair suas coordenadas em um espaço 1D, 2D ou 3D. Para isso, precisamos de uma opção para a definição da dimensão que estamos olhando como parâmetro.

Para a geração de *Value noise* em um espaço 1D, traduziremos sua posição global através de uma função de arredondamento e então multiplicando o valor da coordenada pela cor branca. Este comportamento nos trará a divisão do espaço da textura em listras pois só estamos olhando uma das dimensões das coordenadas. Entretanto, para uma melhor visualização do resultado, introduziremos um parâmetro de frequência *F*, este parâmetro particiona o espaço da textura em *F* pedaços menores para que seja possível ver a repetição do padrão.

Figura 29 ***Value Noise*** *1D com resoluções 10, 25 e 256*

A função criada para geração destes valores de cores utiliza um dicionário com 256 valores que converte as coordenadas globais dos pixels em índices, retornando um valor de

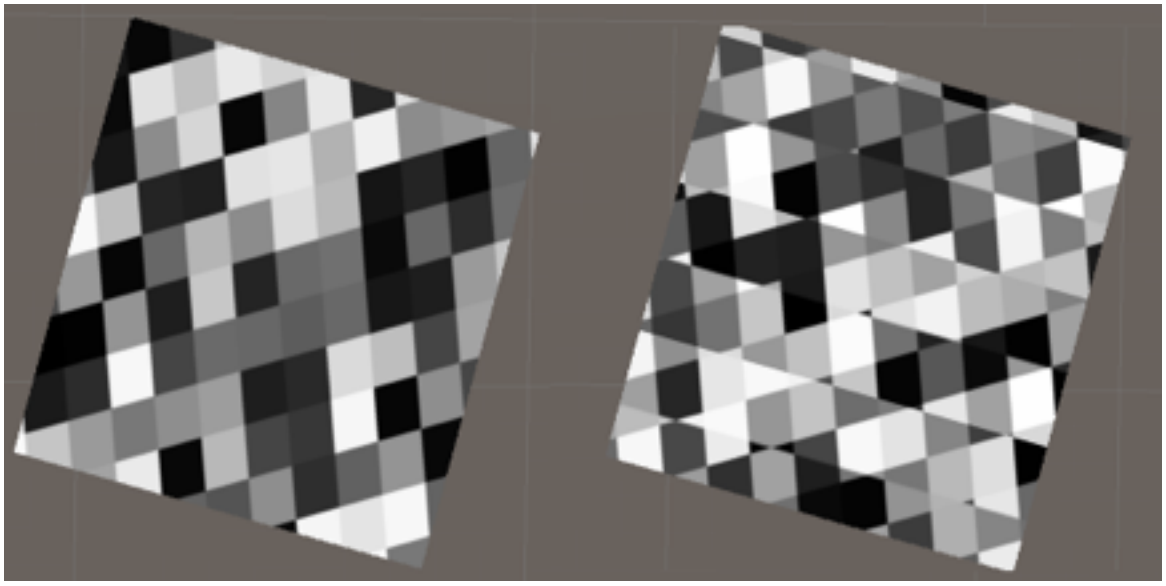
hash que é utilizado multiplicado ao valor da cor branca (1,1,1) na *Unity* para geração do valor da cor final e é então interpolado entre os pixels adjacentes dependendo da dimensão.

Para se aumentar a quantidade de dimensões da função de geração de ruído *Value*, é necessário tirar a amostragem da posição de cada pixel olhando não só uma de suas coordenadas. Sendo assim, a função desenvolvida utiliza o *hash* da coordenada x adicionado a coordenada y como índice para duas dimensões e o *hash* do resultado das duas dimensões adicionado da terceira dimensão como índice do dicionário como pode ser visto nas fórmulas a seguir.

1. 1D =
2. 2D =
3. 3D =

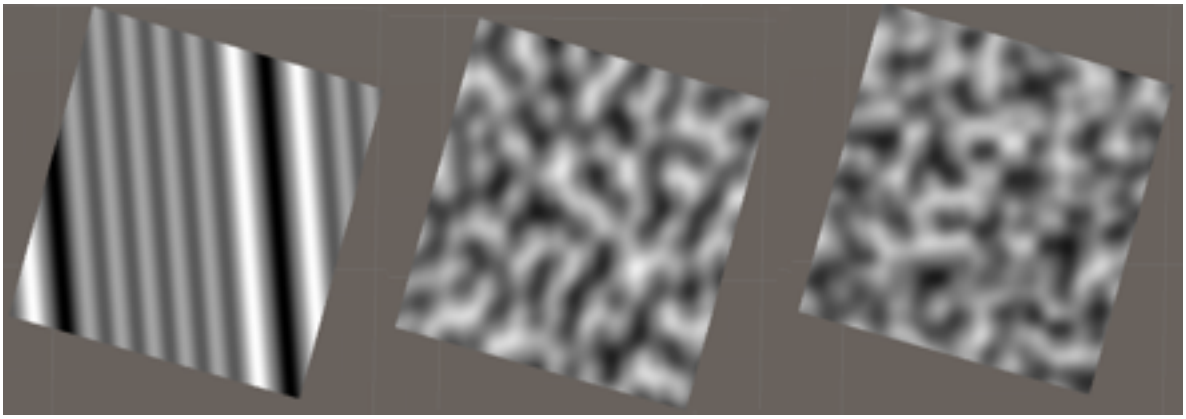
A escrita da função de geração de ruídos *Value* está disponível no código fonte do trabalho. Os resultados da amostragem das dimensões 2D e 3D podem ser vistas na imagem a seguir. Note que a modificação da rotação não afeta o ruído gerado no espaço bidimensional porque sua coordenada é ignorada na amostragem.

Figura 30 Amostragem de coordenadas para criação de ruído no espaço 2D e 3D



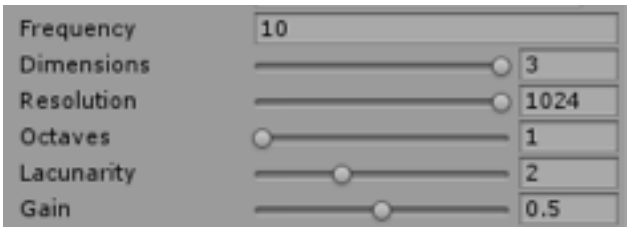
O ruído de *Perlin* é um ruído de gradiente, muitas vezes confundido pelo ruído *Value*. A implementação deste trabalho utiliza direções de gradientes para cada dimensão permitida (1D, 2D e 3D) e interpola estas direções para a produção da textura final. Para dimensões maiores do que 1D é necessário o produto **Dot** entre vetores para se adquirir um gradiente resultante. A aplicação direta das funções geradoras de *Perlin noise* podem ser vistas na imagem a seguir para cada dimensão suportada.

Figura 31 ***Perlin*** *1D, 2D e 3D*



O gerador de ruídos deste trabalho também possui a funcionalidade de geração de ruídos fractais. Para se adequar a necessidade de um nível de detalhamento maior de terrenos é possível somar ruídos com parâmetros diferentes a uma mesma textura para se ter um mapa de alturas mais rico. O processo de geração de ruídos fractais depende da combinação de ruídos e utiliza alguns parâmetros para a amostragem de pixels na textura final, cada novo ruído terá a nomenclatura de oitava e a sua soma será influenciada por uma alteração na amplitude e frequência. A alteração na frequência é comumente chamada de lacunaridade enquanto a mudança de amplitude de Ganho, os parâmetros de geração de ruídos fractais podem ser vistos na imagem a seguir:

Figura 32 Parametros para geração de ruídos fractais



A aplicação destes parâmetros para a geração de ruídos fractais pode ser vista nas imagens abaixo.

Figura 33 ***Perlin noise*** *com frequência base 5 e oitavas variando de 1 a 4 com Lacunaridade 2 e Ganho 0.5 (linha de cima) e Lacunaridade 5 e Ganho 0.8 (linha de baixo)*

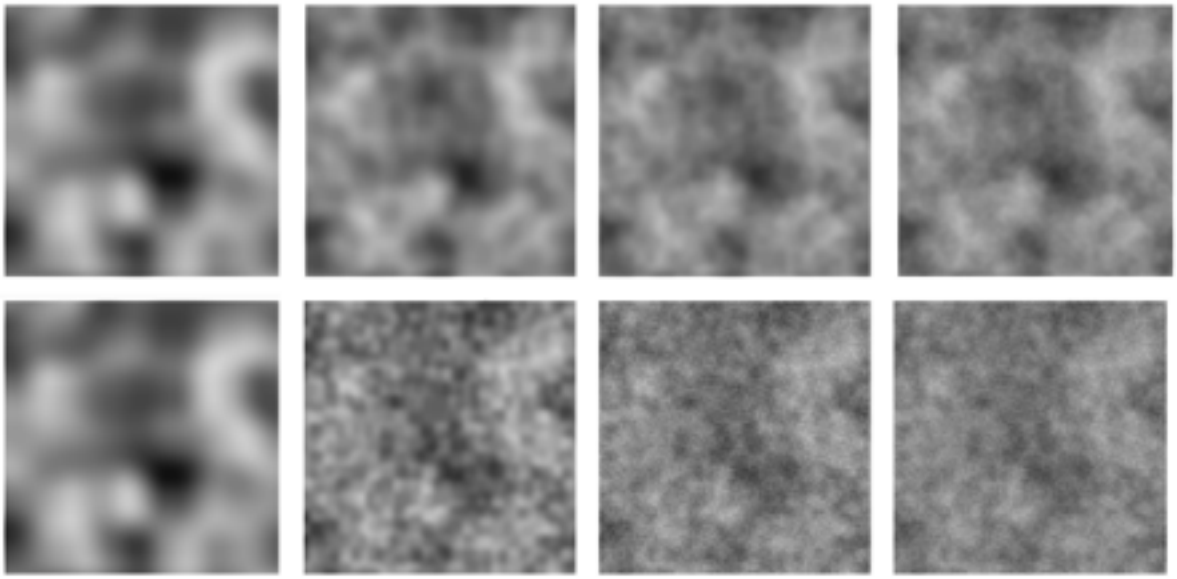
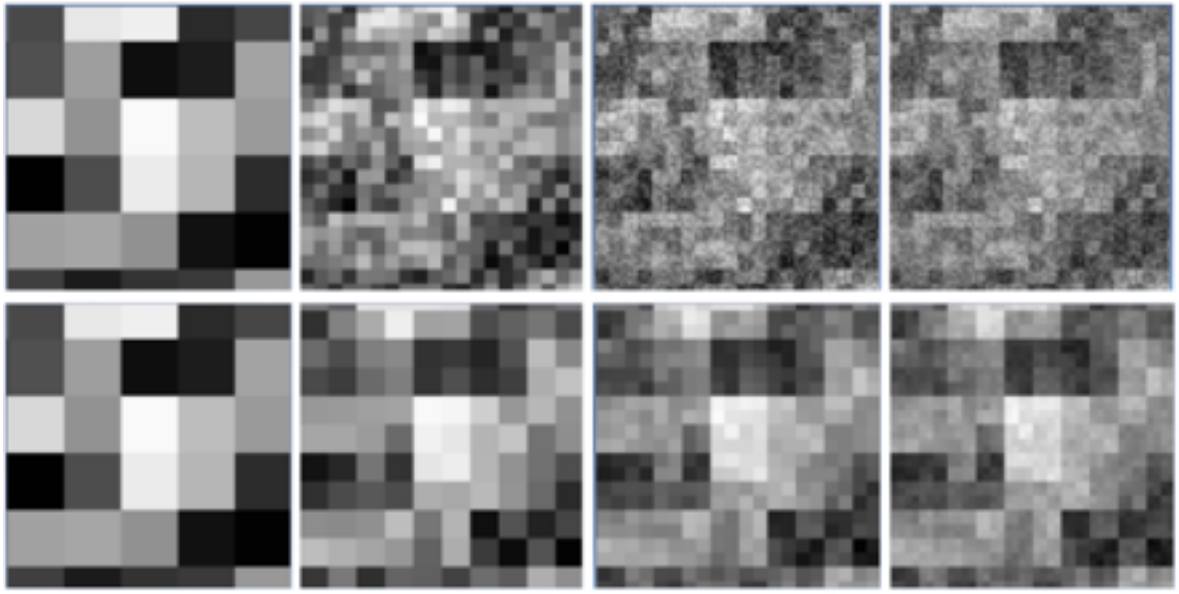


Figura 34 ***Value noise*** *com frequência base 5 e oitavas variando de 1 a 4 com Lacunaridade 2 e Ganho 0.5 (linha de baixo) e Lacunaridade 5 e Ganho 0.8 (linha de cima)*



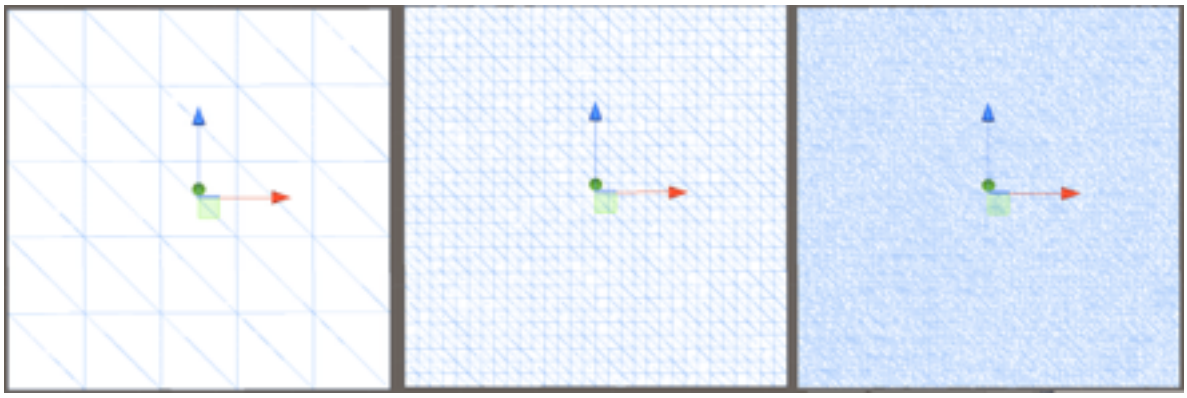
4.4.3 Implementação da malha de terreno inicial

O próximo passo necessário para a criação de terrenos é a da criação de uma malha que tenha os seus vértices acessíveis para que o gerador de terrenos possa modificar a posição dos seus vértices como desejar.

Para implementar tal funcionalidade, a Unity 5 disponibiliza um componente chamado **MeshFilter** que permite que informações de malhas sejam codificadas. A criação de malhas de forma procedural no Unity depende da descrição de quais vértices formam triângulos através da definição da propriedade **vertices** do componente **Mesh** existente em um **MeshFilter**. Esta propriedade é definida como um *array* de inteiros que referencia posições em um *array* de vértices, este *array* de vértices, por sua vez, é composto por valores de **Vector3** (ponto no espaço 3D dentro da *Unity*).

Para a simplificação da implementação o terreno seguirá a forma de um quadrado que pode ser subdividido em menores quadrados de tamanhos iguais para se aumentar a resolução do terreno. A resolução *R* está diretamente conectada a quantidade de vértices encontradas na malha de tal forma que o quadrado responsável pela delimitação da área total do terreno será subdividido em *R*R* quadrados conectados como visto na imagem a seguir.

Figura 35 Resolução afetando diretamente a malha procedural



Um pequeno problema associado a utilização de uma malha para a criação do terreno é que a *Unity* 5 atualmente não permite que uma única malha possua mais de 65 mil vértices, o que é um problema quando se trata da geração de terrenos. Uma solução para tal problema seria o de criar secções juntas pelas suas bordas, mas que estejam separadas em malhas diferentes. Para a simplificação deste projeto essa abordagem não será utilizada, limitando a resolução do terreno para um valor máximo de 254.

4.4.4 Perturbação da superfície baseada no gerador de ruídos

Por conta da limitação de resolução de uma malha para 65 mil vértices a perturbação da malha será limitada por texturas de até a mesma resolução para reduzir a quantidade de interpolações que teriam que ser feitas iterações por pixels desnecessários na leitura de texturas por conta da diferença entre o possível número de vértices da malha e os de pixels da textura.

Sendo assim, o mesmo valor do parâmetro resolução será utilizado para alimentar o gerador de ruídos. Por conta da modularidade do gerador de ruídos, a sua utilização depende apenas da instanciação de um objeto de sua classe. Sendo assim, A perturbação da malha pode ser feita através de uma iteração pelos vértices do terreno e passando suas coordenadas para o gerador de ruído, o valor retornado é então utilizado diretamente para mover os vértices da malha para cima. Exemplos de perturbação da malha podem ser vistos nas imagens a seguir

*Figura 36 Perturbação da malha para Ruído de **Perlin 3D** com resolução 26, 1 Oitava e Frequência 1.3*

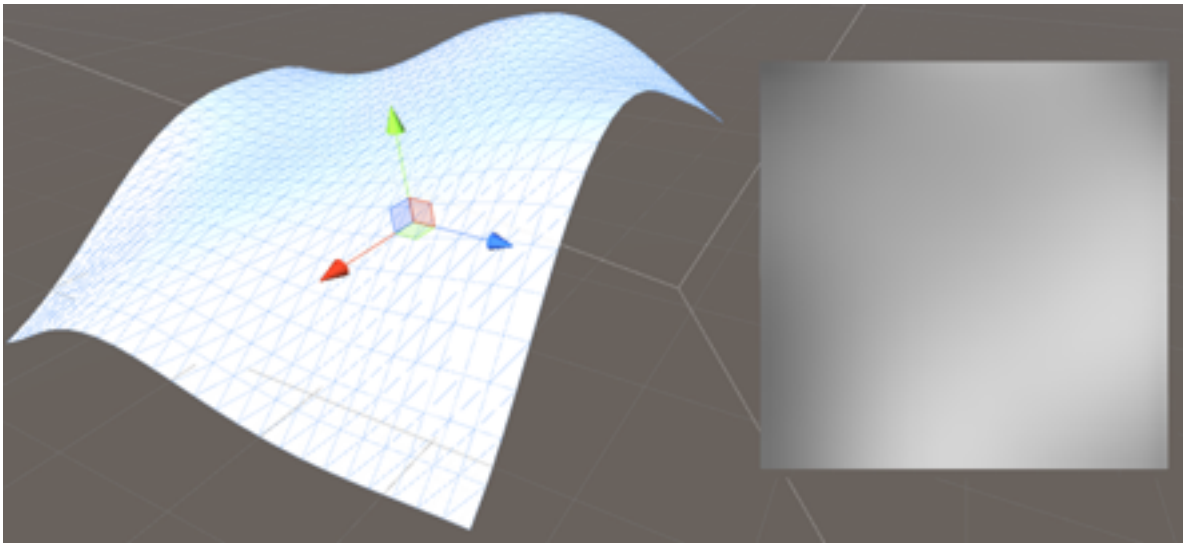
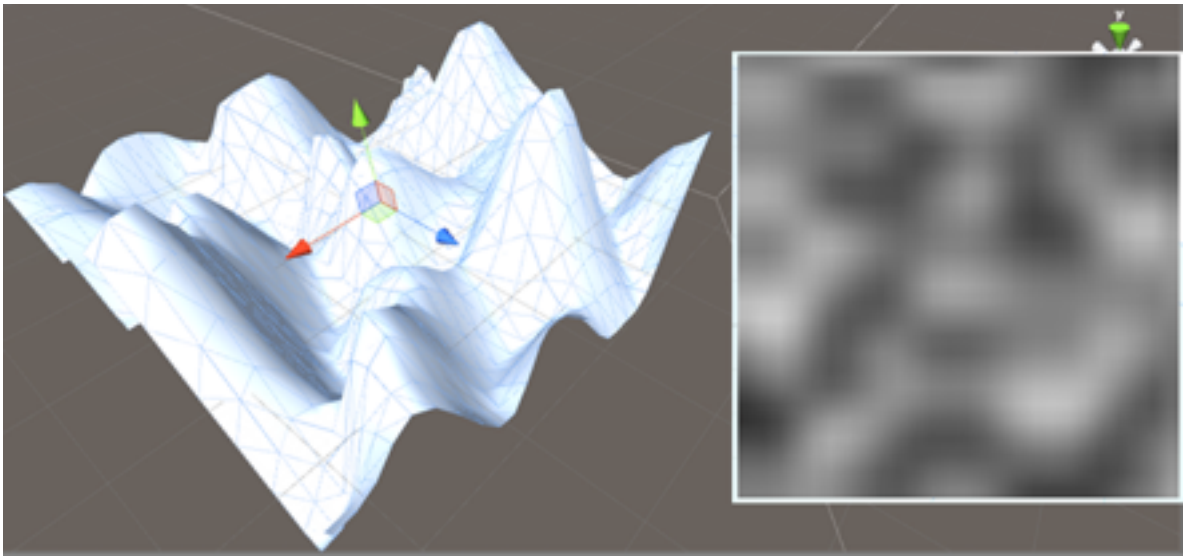


Figura 37 Perturbação da malha para Ruído de Perlin 3D com resolução 26, 1 Oitava e Frequência 4.52



4.4.5 Escultura do terreno

Existem duas abordagens básicas para se trabalhar a escultura do terreno. A primeira seria a de afetar diretamente a forma do terreno através de funções matemáticas para alterar o posicionamento dos vértices da malha em tempo de execução. A segunda abordagem seria a de alterar apenas o mapa de alturas, através da modificação direta da textura em tons de

cinza gerada pelo gerador de ruídos. Este projeto utilizará a segunda abordagem para a escultura.

A primeira parte para criar esta funcionalidade é a de espelhamento da textura do mapa de alturas toda vez que existir uma modificação. Isto pode ser feito de uma forma simples através de um inspetor que chama a função de recriação ou atualização da malha toda vez que existir uma alteração, este controle pode ser feito diretamente pelo código que permite a modificação da textura.

Para alterações nos parâmetros do gerador de ruídos, um inspetor de UI permite que alterações nos valores reflitam no terreno gerado em tempo real, uma simples mudança na frequência ou nos valores de Lacunaridade e Ganho modificam o terreno em tempo de execução dentro da *engine*. Entretanto, não se existe uma forma padrão de se editar texturas de forma artística sem envolver código dentro da *Unity* 5.

Para se criar a possibilidade de esculpir o terreno através da pintura de um mapa de alturas, este projeto utilizou um recurso de texturas de renderização existente dentro da *engine* que permite que uma câmera renderize a sua visualização em uma textura.

A estratégia funciona da seguinte forma:

1. Adicionar uma câmera ortogonal que visualize um plano com a textura do mapa de alturas inicial.
2. Renderizar a visualização desta câmera em uma textura (**RenderTexture**)
3. Projetar, com **raytracer**, a posição do mouse do usuário através da movimentação do **Sprite** de um pincel pela malha.
4. Ao detectar o clique do usuário, instanciar um **Sprite** de um pincel colorido de acordo com a intensidade escolhida pelo usuário na frente da câmera de renderização.
5. Recalcular a malha utilizando a textura gerada pela câmera como novo mapa de alturas.

A preparação para a cena é simples e pode ser descrita com as imagens a seguir. A **figura 38** mostra o posicionamento da câmera em relação ao plano e sua visualização no canto superior esquerdo. A **figura 39** mostra como elementos inseridos entre a câmera e o plano são vistos pela câmera em contraste com o que acontece na *viewport* da *engine.*

Figura 38 Câmera de renderização de texturas para pintura do mapa de alturas

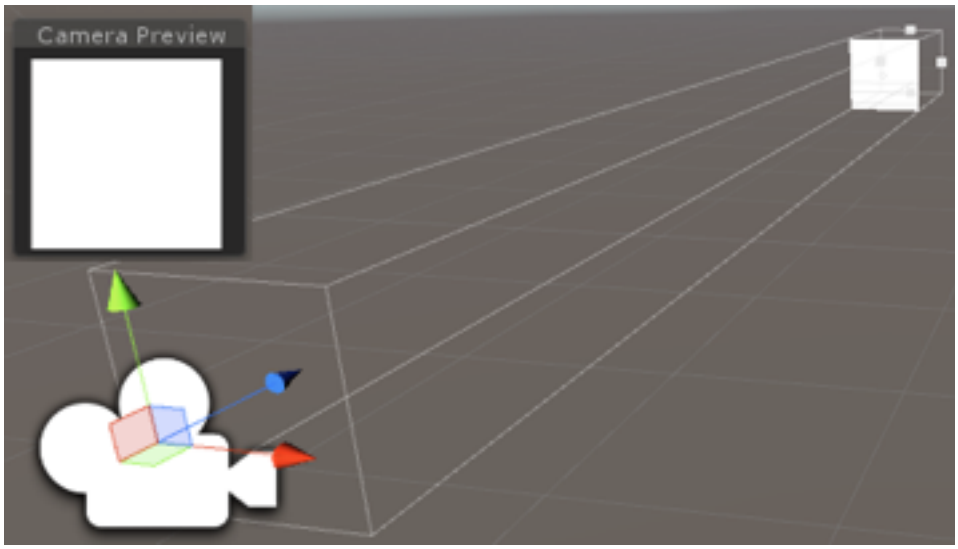
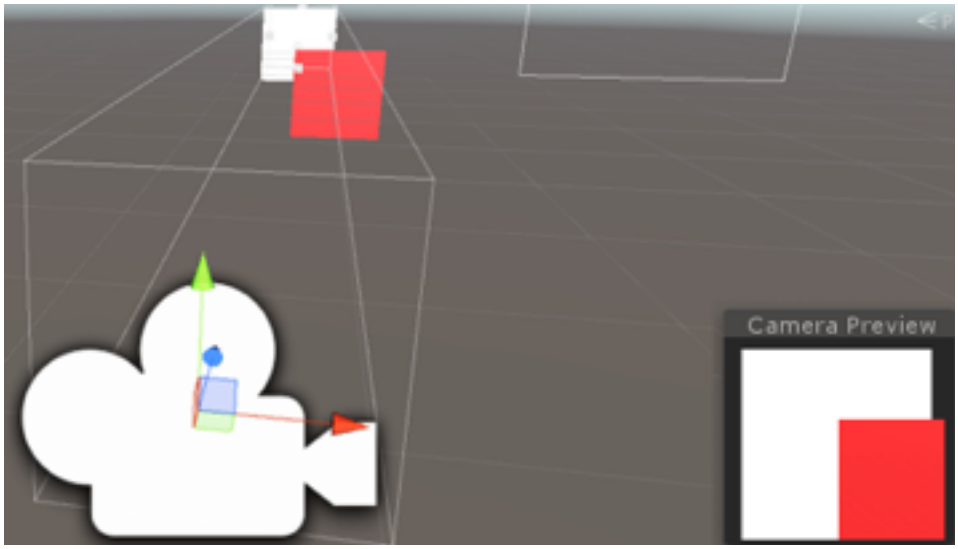


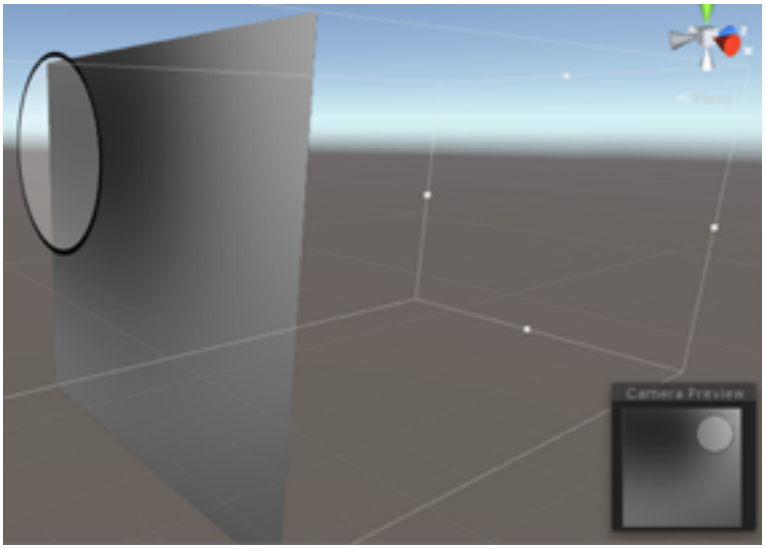
Figure 39 Inserção de um plano colorido entre a câmera e plano alvo cria uma textura planificada



Utilizando esta estratégia de cena, é necessário que se converta os pontos de colisão dos raios traçados pela posição do mouse em coordenadas UVs para se projetar os pincéis

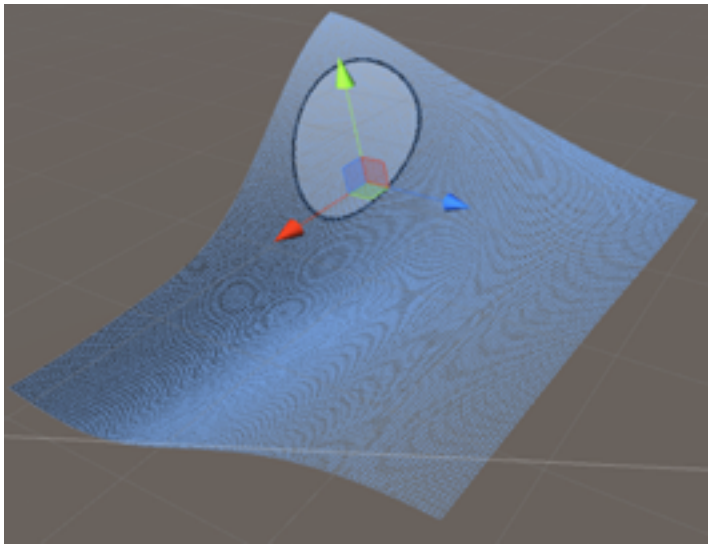
de forma correta entre a câmera e o plano final. O processo é simples, basta apenas normalizar as coordenadas das quinas do plano para se encontrarem no mesmo intervalo das coordenadas UVs. A Colisão com um **MeshCollider** aplicado ao terreno gerado é capaz de dar a posição da colisão, que poderá ser traduzida diretamente a uma coordenada UV. Um exemplo de movimentação de mouse sobre o plano pode ser visto a seguir.

*Figura 40 Posição do mouse detectada pelo **RayTracer** movimenta círculo indicativo do pincel do usuário*



Com isso, no momento em que o usuário movimenta o seu mouse sobre a superfície de uma malha definida é possível renderizar a posição do pincel diretamente na malha. O exemplo da figura abaixo mostra como a textura criada pela câmera na figura 40 reflete na visualização da malha do terreno.

Figura 41 Pincel projetado na malha do terreno através da textura criada pela câmera de renderização

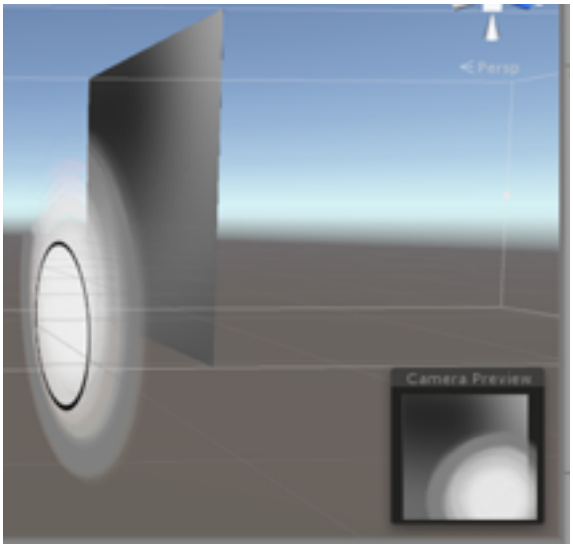


O próximo passo necessário é o de possibilitar que usuários pintem a textura com valores em tons de cinza que possam ser analisados diretamente pelo código de atualização da malha. A estratégia é a mesma, instanciar **Sprites** entre a câmera e o mapa de alturas para que eles possam ser vistos pela câmera de renderização e aplicadas à textura de mapa de alturas.

Para a escolha da intensidade da pintura (que será convertida para tons de cinza) e o tamanho do pincel, dois *sliders* foram criados que podem ser modificados diretamente pelo usuário, o código de pintura verifica o evento de clique do mouse e então captura os dados dos *sliders* para instanciar o pincel de forma correta, como pode ser visto na figura 42 abaixo.

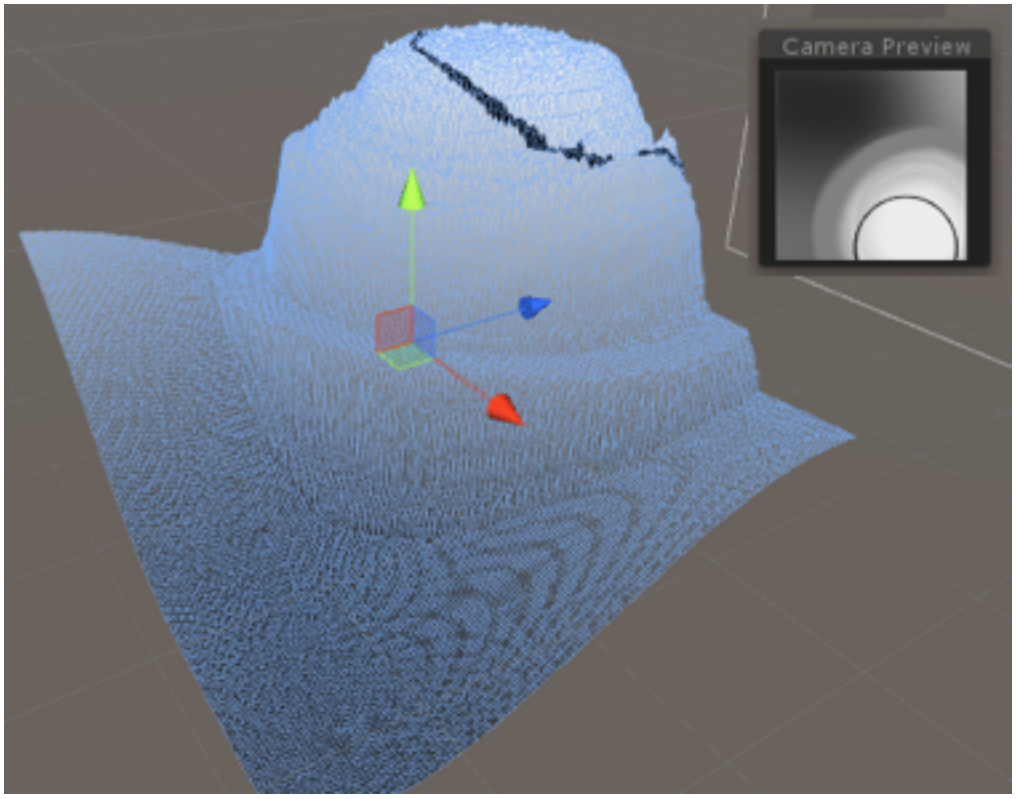
Para evitar uma queda de performance de acordo com o tempo de uso da ferramenta por conta das várias instâncias do **Sprite** do pincel, a cada 100 instâncias o código do pintor salva a textura de renderização em um arquivo, coloca este resultado como textura base do plano alvo e limpa todas as instâncias dos **Sprites** usadas até agora.

*Figura 42 Instâncias do **Sprite** do pincel geradas após cliques do usuário e o resultado final na textura de renderização*



Tendo a textura editada de forma artística, o código do pintor chama a função de atualização das alturas da malha, que reflete diretamente as mudanças feitas pelo o usuário, como pode ser visto no comparativo da textura de renderização e a malha final a seguir.

Figura 43 Modificação no mapa de alturas representadas na malha do terreno



A atualização da malha é feita em tempo real e é totalmente reativa ás ações do usuário mas pode apresentar uma queda de performance de acordo com os valores do parâmetro utilizado para a resolução da malha final do terreno.

4.4.6 Possibilidade de adição de cores ou texturas

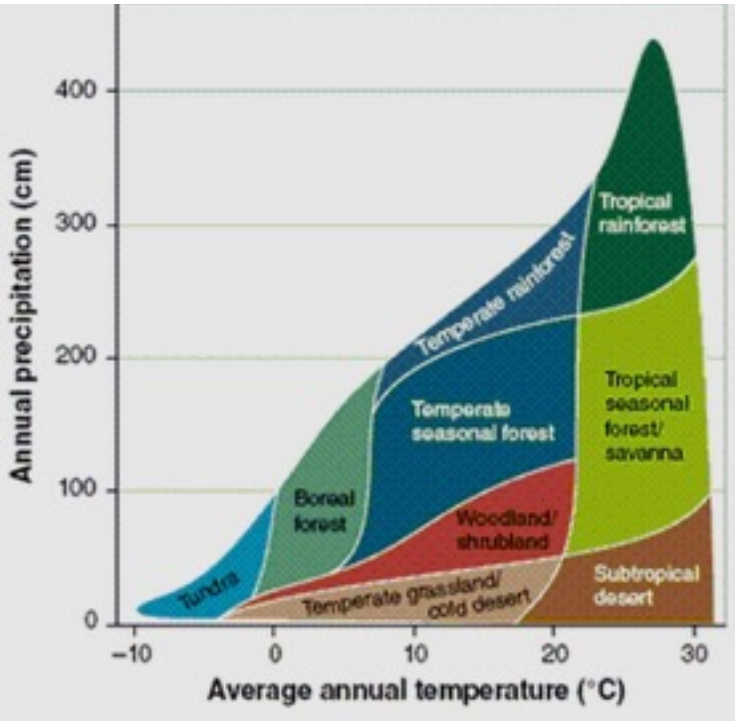
O *shader* padrão existente na *Unity* 5 permite que texturas e cores sejam aplicadas diretamente a objetos de forma simples. Os atributos expostos pelo *shader* permite a criação de materiais que podem suprir a necessidade de coloração do terreno. Exemplos dessa coloração podem ser visto nas figuras 36, 37 e 43.

4.4.7 Parametrização de biomas

A parametrização e geração de regiões de biomas é o verdadeiro foco deste trabalho. As observações dos biomas da natureza revelaram uma correlação com fenômenos naturais como a quantidade de precipitação anual e a temperatura média da região para a existência

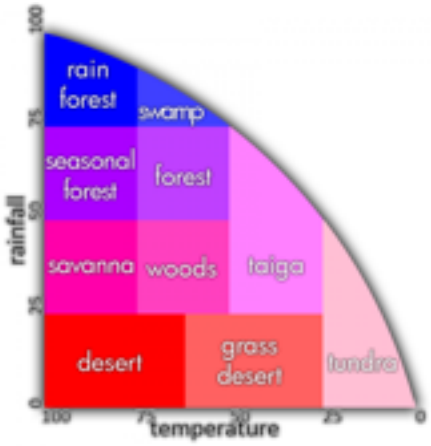
de um bioma específico em uma determinada região. De fato, Whittaker classificou biomas de acordo com esses dois fatores [39], o seu esquema pode ser visto na imagem a seguir.

Figura 44 Classificação de biomas por Whittaker



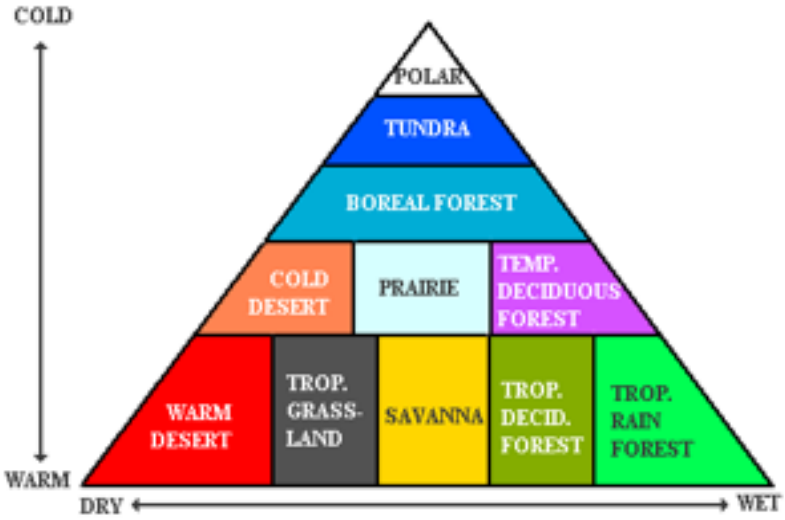
A página na internet da *Wiki* do jogo *Minecraft* possui fatos interessantes de como os biomas são espalhados pelo jogo, incluindo um gráfico explicando os fatores principais que levavam a geração das regiões de biomas nas versões mais antigas do jogo [40], o gráfico pode ser visto a seguir.

Figura 45 Gráfico de biomas do jogo Minecraft nas versões antigas



Com base nestes gráficos, este trabalho decidiu utilizar um modelo simplificado para parametrização de alguns dos biomas existentes na natureza com base nos fatores de temperatura e humidade ou precipitação como visto na figura a seguir.

Figura 46 Parametrização de biomas escolhida pelo trabalho

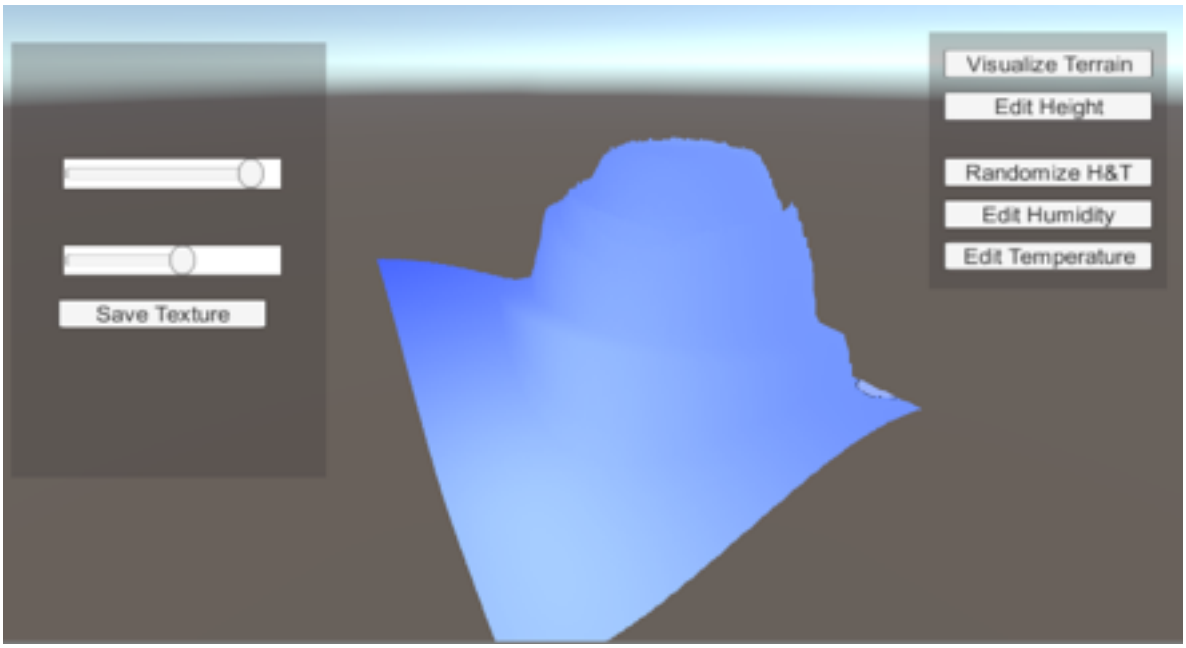


4.4.8 Geração de biomas através da leitura de parâmetros

Com os biomas definidos, o próximo passo é o da geração de mapas de temperaturas e mapas de humidade ou precipitação. Uma funcionalidade importante é de que exista a possibilidade de geração dos biomas acontecer de forma procedural, sendo assim, esta ferramenta utiliza novamente o gerador de ruídos explicado na seção 4.4.2 para a geração dos mapas.

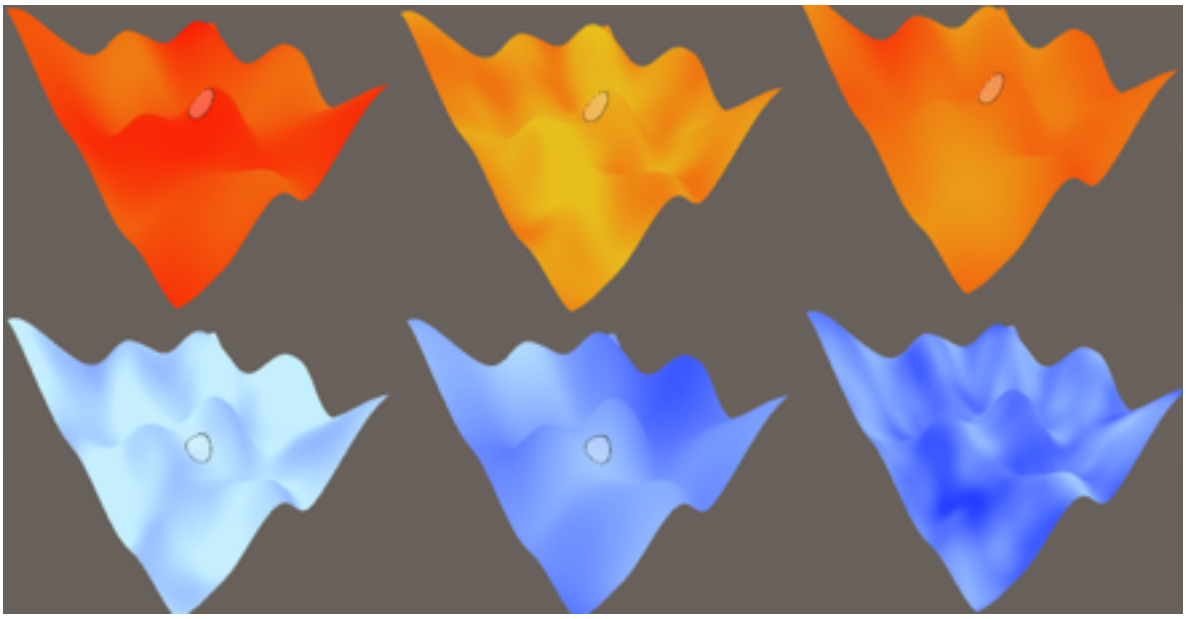
O resultado inicial da geração destes mapas de forma aleatória é extraído do atual mapa de alturas do terreno, mas colorido com um degrade que traga um maior apoio visual ao tipo de mapa sendo editado. Para isso, uma interface simples foi criada para a escolha dos canais de edição do terreno, os possíveis canais são os de edição do mapa de alturas, edição do mapa de humidade edição do mapa de temperaturas. A interface final pode ser vista na imagem a seguir. A mudança nos modos de visualização apenas troca qual textura está sendo aplicada neste momento sobre a malha, desta forma é possível se ver quais regiões específicas estão afetando a malha.

Figura 47 Interface para escolha da edição dos mapas de altura, temperatura e humidade com a edição de humidade ativa



O próximo passo para a geração dos mapas de temperatura e humidade de forma procedural é o de permitir que o usuário insira uma aleatoriedade nos mapas sem que eles imitem a distribuição do mapa de alturas. Por conta disso, uma função foi atrelada ao botão **Randomize H&T** que altera valores do gerador de ruídos e faz uma reiteração sobre as texturas de humidade e temperatura. Exemplos podem ser vistos nas imagens a seguir.

Figura 48 Aleatoriedade da geração dos mapas de temperaturas e Humidade

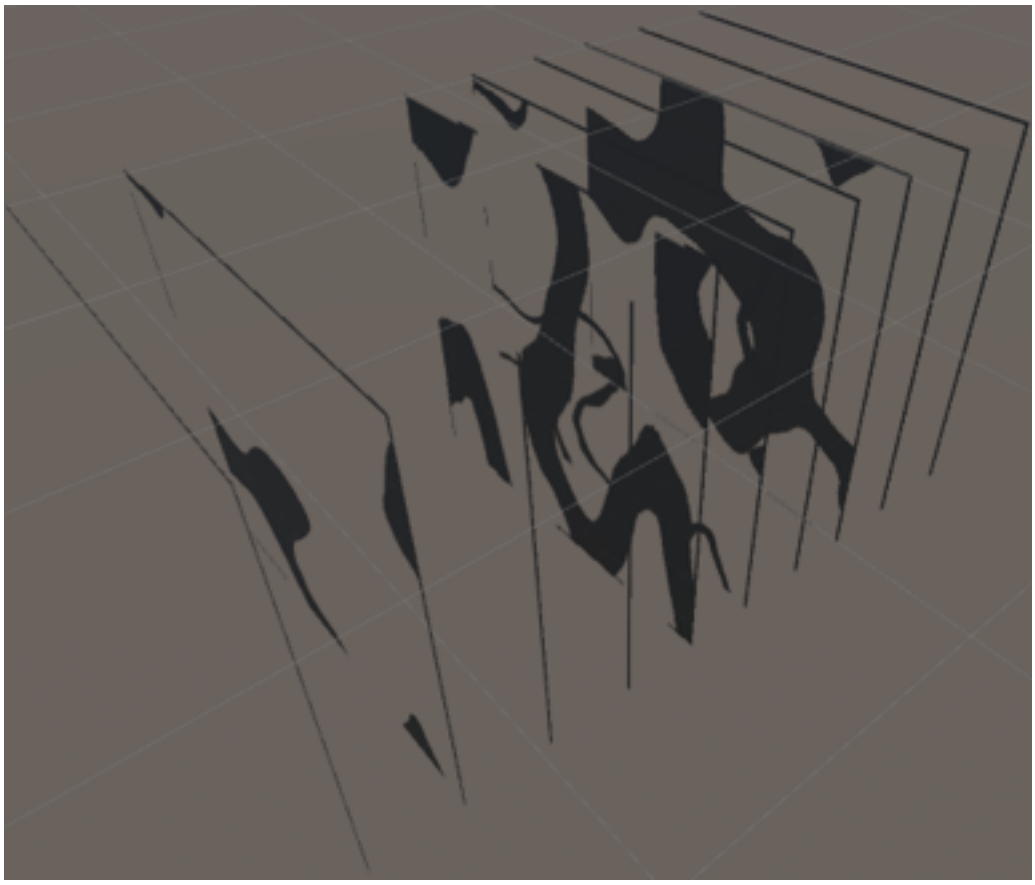


Para que biomas sejam gerados utilizando estes mapas é necessário fazer uma verificação do valor de um pixel em cada textura para um dado vértice da malha. O botão **Visualize Terrain** da interface é responsável por fazer esta verificação de posições e com isso gerar um mapa que define cada bioma.

Para gerar a delimitação dos biomas, uma textura foi gerada para cada bioma. A função que checa, para um dado pixel, o bioma no qual se encontra, procura qual a textura responsável por aquele bioma e a preenche com um valor binário, sendo a cor transparente para não pertencente a região e a cor preta opaca para um pixel pertencente ao bioma. Cada região para um determinado mapa de temperatura e humidade podem ser vistas na figura 49 como texturas de máscara, que podem ser utilizadas para mascarar exatamente qual bioma

abrange qual área do terreno. O alinhamento de todas as texturas observada por uma câmera de renderização pode então criar uma textura final que pode ser aplicada a malha do terreno.

Figura 49 Máscaras geradas para cada bioma específico



Cada uma destas texturas é utilizada como uma máscara para outras texturas serem aplicadas na malha final do terreno. Como exemplo, este trabalhou utilizou texturas gratuitas encontradas no site CGTextures, para a simulação de um material específico para cada terreno. Cada um dos biomas (Polar, Tundra, Floresta Boreal, Deserto Gelado, Pradarias, Floresta Temperada, Deserto Quente, Planícies, Savanas, Floresta Decídua Tropical e Floresta Tropical) recebeu uma destas texturas e o resultado do mapeamento final pode ser projetado diretamente na malha para uma visualização direta do usuário através da aplicação da textura de renderização criada pela câmera que observa o

alinhamento das texturas dos biomas. O resultado desta textura e sua aplicação direta no terreno podem ser vistos nas figuras a seguir.

Figura 50 Máscaras finais dos biomas com texturas aplicadas. Textura final renderizada pela câmera de renderização

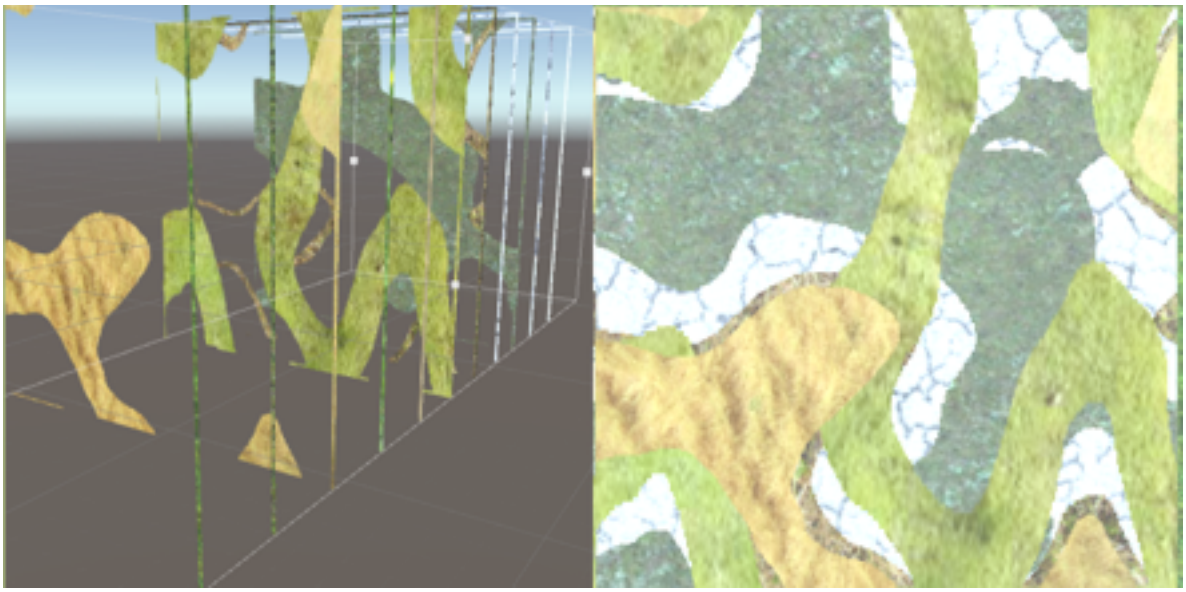
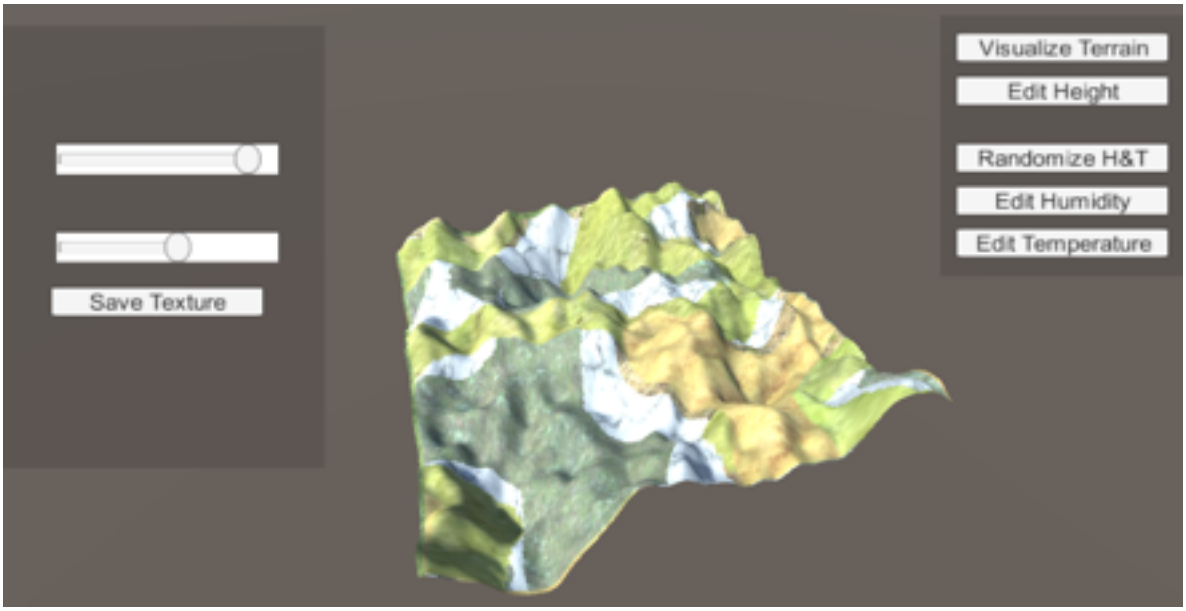


Figura 51 Textura final aplicada ao terreno



4.4.9 Edição de biomas de forma artística

A estratégia para a edição dos biomas de forma artística é a mesma para a edição do mapa de alturas. Câmeras de renderização foram utilizadas em projeções dos mapas de temperatura e humidade para se criar instâncias de pinceis na frente dos planos para que a textura renderizada possa então ser reavaliada pela função de geração de biomas. Apesar da visualização dos mapas de temperatura e humidade serem coloridos, para que a lógica de leitura dos mapas esteja correta, uma conversão para tons de cinza é feita em cada mapa e então aplicada uma textura temporária para leitura. Edições nos mapas de temperatura, humidades e seus efeitos no terreno final podem ser vistos a seguir. Os valores dos tons de cinzas estão invertidos para melhor representar a tonalidade esperada de cada cor.

Figura 52 Edições nos mapas de Humidade, Temperatura, suas conversões para tons de cinza e mapa de alturas

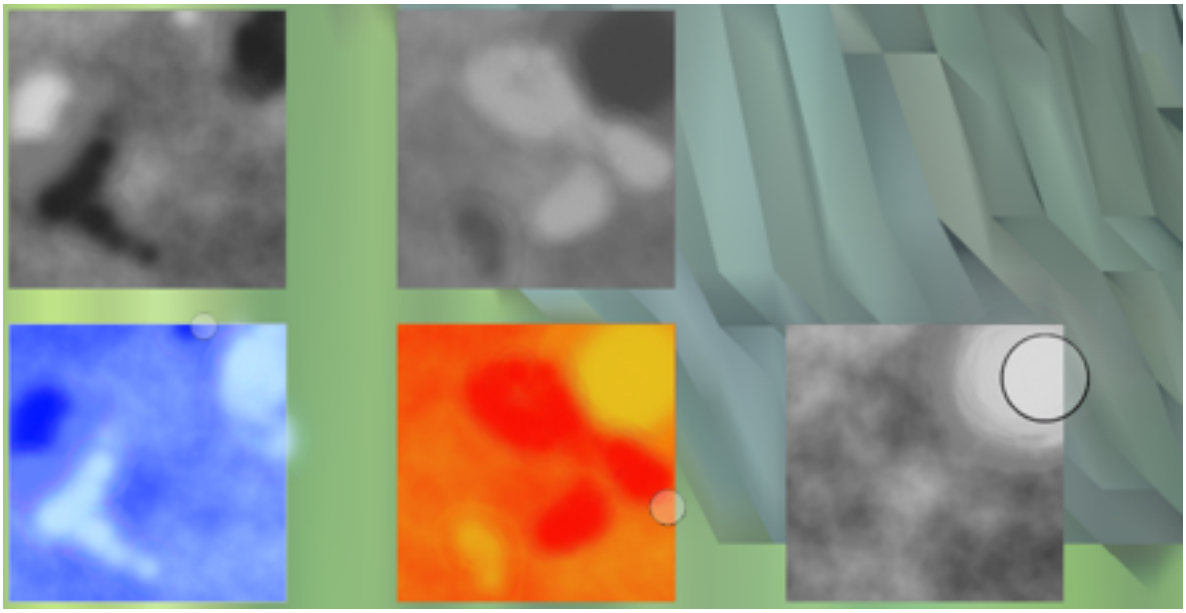
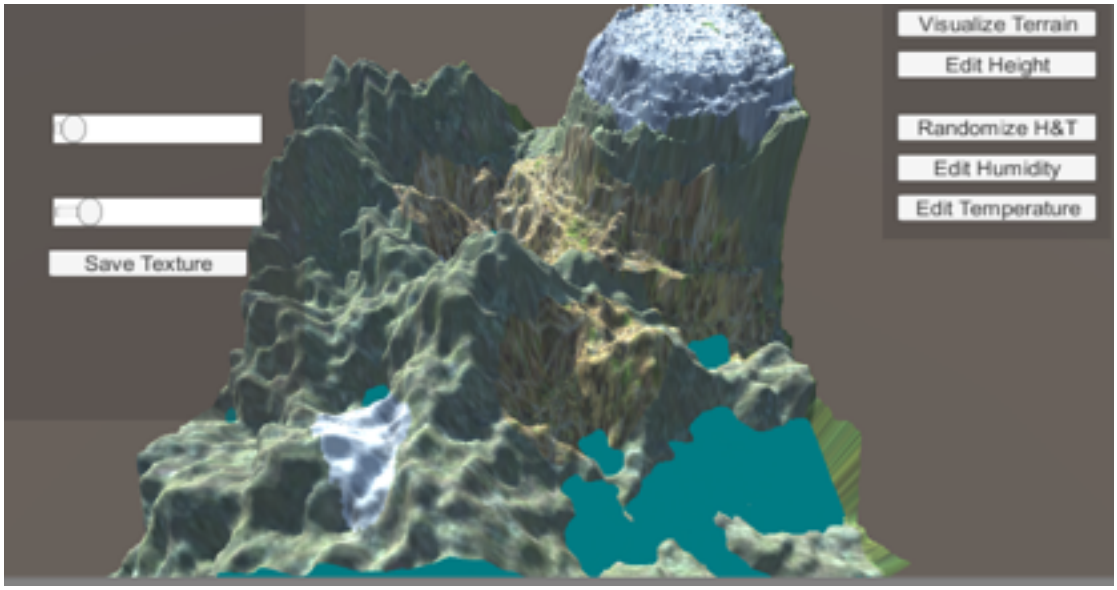


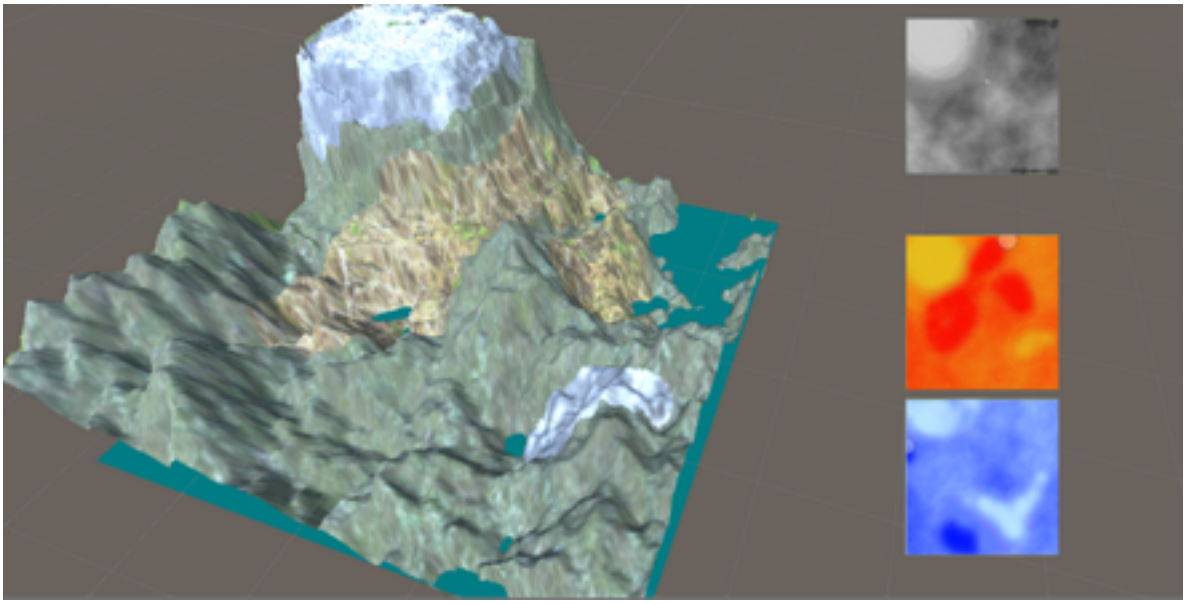
Figura 53 Terreno com delimitações de biomas final gerado através de edições



4.5 Iterações da interface e validação

Após a finalização das funcionalidades listadas até esta seção, uma pequena validação da ferramenta foi feita com o mesmo grupo de 5 pessoas que auxiliaram no processo de priorização na qual duas interfaces foram testadas. A primeira interface foi a utilizada até então durante a escrita deste trabalho que consiste em modificar os parâmetros através do inspetor de propriedades do *Unity 5* e a edição artística feita diretamente no terreno dependendo do modo de visualização atual. A segunda interface foi criada com o objetivo da visualização do terreno final sempre visível e os mapas eram pintados na lateral direita do mapa. A primeira interface pode ser vista na figura 53 e a segunda interface pode ser vista na imagem a seguir.

Figura 54 Segunda interface testada



Estas validações encerraram a primeira iteração de desenvolvimento da ferramenta, as validações levantaram questionamentos e possíveis melhorias que serão apresentadas na seção seguinte referente aos resultados.

4.6 Resumo

Este capítulo descreveu todo o processo de criação da ferramenta, desde a sua concepção e o estudo que levou a definição das funcionalidades que seriam implementadas e os detalhes técnicos da criação de cada parte da ferramenta.

O capítulo começa com uma breve descrição do processo de priorização de funcionalidades, seguido da descrição e detalhamento de um gerador de ruídos utilizado para gerar ruídos *Perlin, Value* e Fractais. Em seguida, foi apresentado como a ferramenta utiliza a textura gerada pelo gerador de ruídos para perturbar a superfície do terreno como um mapa de alturas e também foi descrito o processo de como se modificar esta textura para se esculpir diretamente o terreno.

Da seção 4.4.7 em diante, este trabalho descreveu o processo de parametrização de biomas e como foi criado o mapa de regiões de biomas final que foi convertido para uma

textura de biomas para ser aplicado diretamente sobre a malha do terreno juntamente com a explicação de como a ferramenta possibilita a edição e geração aleatória dos parâmetros que geram esse mapa. Por fim, este capítulo apresentou uma pequena validação feita da primeira iteração da ferramenta.

5. Resultados

O objetivo principal deste trabalho era o de conseguir inserir uma lógica para criação e modificação de regiões de biomas claramente estabelecidos por parâmetros o qual foi implementada e explicada durante o capítulo 4 deste trabalho.

O grupo apresentado nas seções 4.3 e 4.5 é um grupo formado por 5 pessoas atuantes na área de desenvolvimento de jogos no qual um dos seus integrantes é professor de criação de jogos em uma escola de artes digitais no Recife e os outros quatro são estudantes ou atuaram na criação de jogos nos últimos dois anos. Composto por 4 homens e 1 mulher, o grupo possui um intervalo de idades dos 20 aos 26 anos e foi responsável pela validação da ferramenta criada e explicada neste trabalho. Esta validação revelou diversos pontos importantes para uma possível evolução da ferramenta e implantação dos conceitos aqui trabalhos em outras ferramentas.

Inicialmente, o processo de validação partiu de uma entrevista semiestruturada para a validação da importância do trabalho na parametrização de biomas para ferramentas de geração de terrenos. Em seguida, foi feita uma pesquisa estruturada para se apurar a viabilidade de uso da ferramenta e sua expansão futura.

5.1. Pesquisa semiestruturada

Este processo de validação foi focado em se criar perguntas abertas para discussão dos conceitos explorados neste trabalho. O foco das perguntas era o da validação da importância de biomas e podem ser vistas na listagem a seguir:

1. Você conhece algum jogo que utilize biomas diferentes em um mesmo terreno?
2. Qual o impacto destes biomas nestes jogos? (jogabilidade e visual)
3. Você já criou algum terreno ou jogo que necessitasse da criação de biomas?
4. Existia alguma dificuldade na criação destes biomas?
5. Como se criava estes biomas na ferramenta que você utilizou?

6. Era intuitiva e fácil de usar para a geração e diferenciação dos biomas?
7. Você acha que a possibilidade de geração de biomas de forma automática é algo que pode ser aproveitado por artistas e criadores de jogos?

O primeiro resultado que pode ser retirado das respostas é de que todos os integrantes conheciam e já jogaram jogos que existem diversos biomas em um mesmo mapa, o exemplo mais citado foi o do jogo Minecraft. Um dos jogos citados por 3 dos integrantes, o Rust, é um jogo de sobrevivência que depende bastante dos biomas, uma vez que a jogabilidade é alterada de acordo com a preparação do personagem do jogador para cada bioma.

O segundo resultado extraído da entrevista é de que apenas dois dos cinco entrevistados já tinham trabalhado com um jogo que utilizasse este conceito de biomas, ambos através da *Unreal Engine* 4. Um deles achou que a parte visual dos biomas era de fácil criação através das ferramentas disponíveis na *Unreal Engine* 4, mas que mapear a codificação das regiões com os gráficos era uma tarefa difícil e que inviabilizou a continuidade do projeto. O outro respondeu que teve dificuldades tanto na criação visual quanto na programação das regiões.

O terceiro e último resultado desta pesquisa é o de que todos os entrevistados vêm potencial na exploração da criação de biomas e ecossistemas de forma automática, mas que o público alvo de ferramentas com este objetivo, possivelmente, é o de criadores de jogos em que a jogabilidade é afetada pela navegação dos jogadores através de biomas diferentes.

Esta pesquisa não busca resultados numéricos e sim a validação de que o assunto explorado neste trabalho é de interesse da comunidade, mesmo que através de uma amostragem pequena de profissionais.

5.2 Pesquisa estruturada

O objetivo desta segunda pesquisa é o da validação da ferramenta criada e apresentada neste trabalho como possível início de uma ferramenta completa de criação de terrenos de forma procedural ou como possível adição ou extensão para aplicações já

existentes. Os resultados a seguir foram extraídos de um questionário para o mesmo grupo de 5 pessoas citado acima em que algumas perguntas foram feitas para ambas as interfaces apresentadas na seção 4.5 deste trabalho.

Quanto a navegabilidade da ferramenta (interface 1 e 2), 80% dos entrevistados acharam que para as funcionalidades presentes a interface e controles eram satisfatórios.

Quanto a interação com o gerador de ruídos, 100% dos entrevistados acharam que as formas de interação com os parâmetros eram satisfatórias.

Quanto a visualização do resultado do gerador de ruídos, 60% dos entrevistados acharam a visualização satisfatória. Os problemas associados foram os de lentidão na visualização da troca de alguns parâmetros e outros em relação a navegabilidade.

Quanto a pintura do mapa de alturas, 100% dos entrevistados acharam a resposta do processo de escultura satisfatória, entretanto todos os entrevistados acharam os recursos disponíveis para o processo de escultura insuficientes para se criar terrenos completos.

Quanto a visualização dos mapas de humidade e temperatura, para a interface 1 60% dos entrevistados acharam a visualização satisfatória. Os problemas relatados pelos outros 40% foram relacionadas a dificuldade de enxergar os volumes do terreno enquanto visualizavam a textura dos mapas. Para a interface 2, 80% dos usuários acharam a visualização satisfatória. Os problemas relatados pelos demais incluem a dificuldade em mapear regiões do mapa para regiões do terreno.

Quanto a modificação dos mapas de humidade e temperatura, para a interface 1 20% dos entrevistados acharam a interface satisfatória para a edição dos mapas, o mesmo resultado foi recebido na interface 2. Os problemas associados a esta atividade são relacionados à falta de recursos para se fazer pinturas mais interessantes, à interface pobre e simples para alteração dos pincéis e à dificuldade de visualização dos volumes do terreno durante o processo de pintura.

Esta pesquisa revelou que, apesar do trabalho ter criado uma ferramenta para a interação com a criação de biomas, a ferramenta gerada ainda é muito precária para ser

utilizada de fato em um processo de criação de jogos. Na sua atual versão, não foi feita uma integração a *game engine* e diversos recursos da *engine* não poderiam ser utilizados na malha final do terreno, além de que seria impossível se comparar com as ferramentas existentes no mercado atualmente, principalmente pela não possibilidade de criação de terrenos muito detalhados em alta resolução, mas que os conceitos trabalhados em biomas podem ser aplicados por ferramentas já existentes.

6. Conclusão

Este trabalho teve como produto final o estudo e criação de uma ferramenta de geração procedural de terrenos que funciona por cima de módulos de geração procedural de texturas.

Através do estudo de ferramentas existentes na atualidade para a geração procedural de terrenos percebeu-se que não se existiam formas para a geração de múltiplos biomas para um terreno de forma procedural. Biomas são conceitos muito trabalhados por alguns jogos, como o jogo *Minecraft*, e uma forma de se criar regiões específicas para biomas é algo que deve ser codificado completamente pelos desenvolvedores do jogo na atualidade. Com base neste estudo, a ferramenta criada neste trabalho teve como objetivo parametrizar alguns biomas encontrados na natureza através de aproximações gráficos reais de correlação entre fatores climáticos e a existência dos biomas no planeta e criar as regiões no terreno de forma automática e então desenvolver uma forma de interagir com estes parâmetros diretamente dentro da ferramenta.

Duas pesquisas foram feitas com um pequeno grupo de profissionais e estudantes que atuam na criação de jogos para a validação do conceito de se incluir métodos para se trabalhar diretamente com biomas dentro das ferramentas atuais de geração de terrenos de forma procedural. As pesquisas trouxeram resultados importantes que apontam tanto que existe espaço para se trabalhar na exploração deste conceito atualmente e que biomas, de

fato, são essenciais para alguns jogos. Além de mostrar que uma ferramenta pode ser criada para prototipação e validade de funcionalidades relacionadas a este tema.

6.1 Limitações e dificuldades

A criação de uma ferramenta completa de geração procedural de terrenos é muito custosa e complexa. Times desenvolvem suas ferramentas e as disponibilizam através da compra de licenças ou de forma gratuita com o pagamento de royalties baseado no lucro do produto desenvolvido por cima dela, caso não sejam criadas especificamente para jogos.

Geração procedural lida com conceitos matemáticos complexos que requerem tempo de estudo e experimentação para se atingir bons resultados, além disso, grandes barreiras para o desenvolvimento desta ferramenta foram as da necessidade de aprendizado de APIs, linguagens de programação e ferramentas, uma vez que a exploração de ferramentas é um processo demorado e custoso.

6.2 Sugestões para trabalhos futuros e melhorias

Toda a ferramenta foi criada sem assincronia entre as atualizações de texturas e malha, o que causa uma perda na performance. A programação utilizando a GPU também pode acelerar a ferramenta a níveis que se torne possível a criação de simulações de fenômenos naturais, pinturas com partículas e filtros de erosão e possa competir com ferramentas já existentes no mercado. Uma grande necessidade de se trabalhar utilizando a GPU e paralelização é de possibilitar que a ferramenta tenha uma performance elevada e traga uma pré-visualização de boa qualidade do terreno final em tempo real.

A criação de subdivisões locais é algo que pode afetar diretamente a satisfação dos usuários da ferramenta por possibilitar a inserção de detalhes em áreas específicas.

Inserção de leitura de mapas de alturas nos formatos mais utilizados pela indústria também é algo benéfico que pode ser adicionado à ferramenta.

A integração com as ferramentas de edição de terrenos da *Unity 5* ou *Unreal Engine 4* é algo que pode aquecer a comunidade de criação de conteúdo de forma procedural e se tornar padrão dentro de versões futuras da *engine*.

Todas as funcionalidades foram escritas utilizando leitura e escrita de texturas, que é um processo um pouco demorado, é possível converter boa parte do código gerado para linguagens de *shaders* que podem trabalhar com vários passes para substituir as operações feitas sobre texturas.

A implementação das funcionalidades listas no capítulo 4 que não foram finalizadas seria um acréscimo necessário para que a ferramenta possa competir com as demais existentes no mercado.

Adicionar a possibilidade de inserção, edição e remoção de biomas de forma manual é uma funcionalidade que possa tornar o sistema de biomas mais rico dentro da ferramenta.

Criar uma ferramenta de texturização do terreno através de uma interface não destrutiva é algo que pode ser trabalhado de forma isolada e acrescentar muito valor a ferramenta atual.

7. Referências

[1] GameSpy: Rise Of The Roguelikes: A Genre Evolves - Page 1. Pc.gamespy.com. Disponível em: <http://pc.gamespy.com/pc/ftl-faster-than-light/1227287p1.html>. Acesso em: 13 maio 2016.

[2] BBC NEWS | Business | Cost headache for game developers. News.bbc.co.uk. Disponível em: <http://news.bbc.co.uk/2/hi/business/7151961.stm>. Acesso em: 10 mar. 2016.

[3] Gamasutra: Tanya Short's Blog - Level Design in Procedural Generation. Gamasutra.com. Disponível em: <http://www.gamasutra.com/blogs/TanyaShort/

20140204/209176/Level_Design_in_Procedural_Generation.php>. Acesso em: 13 jul. 2016.

[4] Essentials facts about the computer and video game industry. 1. ed. [s.l.: s.n.], 2014. Disponível em: <http://www.isfe.eu/sites/isfe.eu/files/attachments/esa_ef_2014.pdf>. Acesso em: 7 jun. 2016.

[5] Develop. Develop-online.net. Disponível em: <http://www.develop-online.net/studio-profile/inside-rockstar-north-part-2-the-studio/0184061>. Acesso em: 7 jun. 2016.

[6] Our Entire Galaxy Re-created In Elite: Dangerous. CONTROL500. Disponível em: <http://ctrl500.com/tech/how-frontier-managed-to-re-create-our-entire-galaxy-in-elite-dangerous/>. Acesso em: 4 jun. 2016.

[7] Fractals - Hunting the Hidden Dimension. [s.l.]: NOVA PBS, 2008.

[8] Value Noise and Procedural Patterns: Part 1 (Creating a Simple 1D Noise). Scratchapixel.com. Disponível em: <http://scratchapixel.com/lessons/procedural-generation-vritual-worlds%20/procedural-patterns-noise-part-1/creating-simple-1D-noise>. Acesso em: 6 maio 2016.

[9] EBERT, David S., MUSGRAVE, F. KentonPeachey, Darwyn et al. Texturing & Modeling: A procedural approach. 2. ed. [s.l.]: AP Professional, 1998.

[10] SolidTextures. Ohio State Computer Science and Engineering. Disponível em: <http://web.cse.ohio-state.edu/~parent/classes/681/Lectures/13.SolidTextures.pdf>. Acesso em: 9 abr. 2016.

[11] Aristid Lindenmayer, Mathematical models for cellular interaction in development. J.Theoret. Biology, 18:280—315, 1968.

[12] Rozenberg, GrzegorzSalomaa, Arto. Mathematical Theory of L Systems. Orlando, FL: Academic Press, 1980.

[13] Generative Modeling. Generative-modeling.org. Disponível em: <http://www.generative-modeling.org/>. Acesso em: 3 jul. 2016.

[14] Davidson, Kim. Sponsored: Go Procedural - A Better Way to Make Better Games. Gamasutra.com. Disponível em: <http://www.gamasutra.com/view/news/233899/Sponsored_Go_Procedural__A_Better_Way_to_Make_Better_Games.php>. Acesso em: 12 jun. 2016.

[15] Procedural Mesh Generation - Epic Wiki. Wiki.unrealengine.com. Disponível em: <https://wiki.unrealengine.com/Procedural_Mesh_Generation>. Acesso em: 10 maio 2016.

[16] Technologies, Unity. Unity - Manual: Procedural Mesh Geometry. Docs.unity3d.com. Disponível em: <https://docs.unity3d.com/Manual/GeneratingMeshGeometryProcedurally.html>. Acesso em: 10 maio 2016.

[17] Génevaux, Jean-David, Galin, ÉricGuérin, Éric et al. Terrain Generation Using Procedural Models Based on Hydrology. SIGGRAPH, 2013. Disponível em: <http://arches.liris.cnrs.fr/publications/articles/SIGGRAPH2013_PCG_Terrain.pdf>. Acesso em: 2 jul. 2016.

[18] Mandelbrot, Benoit B. The fractal geometry of nature. New York: W.H. Freeman, 1983.

[19] T, Cham. Advances in Multimedia Modeling. In: 13th International Multimedia Modeling Conference. Singapore: [s.n.], 2007.

[20] Is the Fractal Model Appropriate for Terrain?. http://scribblethink.org/. Disponível em: <http://scribblethink.org/Work/caseagainstfractals.pdf>. Acesso em: 8 abr. 2016.

[21] Jense, Hans. Dynamic Terrain Generation Based on Multifractal Techniques. In: High Performance Computing for Computer Graphics and Visualisation. 1. ed. [s.l.: s.n.], 1995, p. 186-203.

[22] Junior, Alberto Rodrigues. FERRAMENTA PARA GERAÇÃO PROCEDIMENTAL DE NÍVEIS EM JOGOS INFINITOS. Graduado, Universidade Federal de Pernambuco, 2014.

[23] Sorenson, Nathan Pasquier, Philippe. Towards a Generic Framework for Automated Video Game Level Creation. School of Interactive Arts and Technology, 2016.

[24] Smith, Gillian, Treanor, MikeWhitehead, Jim et al. Rhythm-Based Level Generation for 2D Platformers. University of California, Santa Cruz, .

[25] GPU Gems 3 - Chapter 1. Generating Complex Procedural Terrains Using the GPU. Http.developer.nvidia.com. Disponível em: <http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html>. Acesso em: 8 jul. 2016.

[26] Geometry-Shader Object (Windows). Msdn.microsoft.com. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/bb509609(v=vs.85).aspx>. Acesso em: 8 jul. 2016.

[27] Getting Started with the Stream-Output Stage (Windows). Msdn.microsoft.com. Disponível em: <https://msdn.microsoft.com/en-us/library/windows/desktop/bb205122(v=vs.85).aspx>. Acesso em: 8 jul. 2016.

[28] Electronic Arts - SSX | SideFX. Sidefx.com. Disponível em: <https://www.sidefx.com/stories/electronic-arts-ssx/>. Acesso em: 2 maio 2016.

[29] World Machine : 3D Terrain Generation. World-machine.com. Disponível em: <http://www.world-machine.com/>. Acesso em: 23 abr. 2016.

[30] Learn about users of WM. World-machine.com. Disponível em: <http://www.world-machine.com/about.php?page=testimonial>. Acesso em: 23 abr. 2016.

[31] Features. World-machine.com. Disponível em: <http://www.world-machine.com/about.php?page=features>. Acesso em: 23 abr. 2016.

[32] User, Super. Scale. Planetside.co.uk. Disponível em: <http://planetside.co.uk/scale-tour>. Acesso em: 20 abr. 2016.

[33] Terrain and Water - Planetside Software Wiki. Planetside.co.uk. Disponível em: <http://www.planetside.co.uk/wiki/index.php?title=Terrain_and_Water>. Acesso em: 27 abr. 2016.

[34] e-on software - Blockbusters Use Vue. E-onsoftware.com. Disponível em: <http://www.e-onsoftware.com/showcase/spotlights/movies.php>. Acesso em: 30 abr. 2016.

[35] e-on software - VUE Complete 2015. E-onsoftware.com. Disponível em: <http://www.e-onsoftware.com/products/vue/vue_2015_complete/?page=terrains>. Acesso em: 30 abr. 2016.

[36] Creating Landscapes. Docs.unrealengine.com. Disponível em: <https://docs.unrealengine.com/latest/INT/Engine/Landscape/Creation/>. Acesso em: 30 abr. 2016.

[37] Technologies, Unity. Unity - Manual: Textures. Docs.unity3d.com. Disponível em: <https://docs.unity3d.com/Manual/terrain-Textures.html>. Acesso em: 30 abr. 2016.

[38] Modifying Terrain - CRYENGINE Manual - Documentation. Docs.cryengine.com. Disponível em: <http://docs.cryengine.com/display/SDKDOC2/Modifying+Terrain>. Acesso em: 30 abr. 2016.

[39] Whittaker, Robert H., Botanical Review, Classification of Natural Communities, Vol. 28, No. 1 1962, p. 1–239.

[40] Biome – Official Minecraft Wiki. Minecraft.gamepedia.com. Disponível em: <http://minecraft.gamepedia.com/Biome>. Acesso em: 2 abr. 2016.

[41] Togelius, Julian. Procedural Content Generation: Goals, Challenges and Actionable Steps. Julian Togelius. Disponível em: <http://julian.togelius.com/Togelius2013Procedural.pdf>. Acesso em: 17 jul. 2016.