UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SIMULAÇÃO DE DINÂMICA DO RELEVO ATRAVÉS DA TRANSFORMAÇÃO DE MAPAS DE ALTURA

GUILHERME DIEGOLI NETO

BLUMENAU
2017

GUILHERME DIEGOLI NETO

SIMULAÇÃO DE DINÂMICA DO RELEVO ATRAVÉS DA TRANSFORMAÇÃO DE MAPAS DE ALTURA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dalton Solano dos Reis, M.Sc. - Orientador

BLUMENAU
2017

SIMULAÇÃO DE DINÂMICA DO RELEVO ATRAVÉS DA TRANSFORMAÇÃO DE MAPAS DE ALTURA

Por

GUILHERME DIEGOLI NETO

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof(a). Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof(a). Maurício Capobianco Lopes, Dr. – FURB
Membro:	Prof(a). Aurélio Faustino Hoppe, M.Sc. – FURB

Blumenau, 6 de julho de 2017

Dedico este trabalho a todos os meus grandes amigos que, mesmo separados por tempo ou espaço, expressaram o apoio necessário para que eu alcançasse todas essas realizações.

AGRADECIMENTOS

Agradeço à toda minha família e amigos, tanto próximos quanto distantes, que me acompanharam e deram apoio durante o desenvolvimento deste trabalho. Agradecimentos especiais aos professores Dalton Solano dos Reis e Maurício Pozzobon pelo seu interesse e por suas orientações que contribuíram muito para a conclusão deste trabalho.

“Sem paixão você não tem energia; sem energia, você não tem nada. Nada grandioso no mundo foi alcançado sem paixão!”

Donald J. Trump

RESUMO

A paisagem do planeta Terra encontra-se em constante mudança devido à ação de diversos processos de dinâmica de relevo, sendo que muitos destes carregam consequências para a sociedade, sendo de interesse o seu entendimento. Este trabalho apresenta o desenvolvimento de uma aplicação que permite a criação e visualização de uma paisagem virtual, a criação de algoritmos que representem processos de dinâmica de relevo em tempo real e a obtenção de informações sobre uma paisagem a qualquer momento. As paisagens são representadas computacionalmente como matrizes de alturas e os algoritmos são baseados nos algoritmos de transformação de imagens. São analisados dois algoritmos de simulação dos processos de erosão térmica (deslizamentos) e erosão hídrica utilizados em geração procedural de terreno e então são realizadas diversas adaptações nos algoritmos para permitir parametrizações baseadas na profundidade do solo, tipo de superfície e sua humidade. A aplicação destes algoritmos apresenta resultados que podem ser esperados de uma ocorrência real, embora de uma forma altamente simplificada. Embora a escolha da plataforma de renderização tenha impactado o desempenho final da aplicação, o modelo de simulação se mostrou viável, com possibilidade de expansões ou até mesmo a criação de algoritmos similares não explorados neste trabalho.

Palavras-chave: Dinâmica de relevo. Erosão. Simulação. Mapa de altura.

ABSTRACT

The landscape of planet Earth is constantly changing due to the action of several geomorphological processes, many of those carrying consequences for society, justifying the need for better understanding them. This work presents the development of an application that allows the creation and visualization of a virtual landscape, the creation of algorithms that represent geomorphological processes in real time and the obtaining of information about the landscape at any moment. The landscapes are represented computationally as height matrices, and the algorithms are based on image transformation algorithms. Two algorithms are used to simulate the processes of thermal erosion (landslides) and hydraulic erosion and then several adaptations are made in the algorithms to allow parameterizations based on soil depth, surface type and its humidity. The use of these algorithms presents results that can be expected from a real occurrence, although in a highly simplified way. Even though the choice of rendering platform has impacted the final performance of the application, the simulation model proved feasible, with the possibility of expansion or even creation of similar algorithms not explored in this work.

Key-words: Geomorphological processes. Erosion. Simulation. Heightmaps.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A paisagem do planeta Terra está em constante mudança. Esse processo deve-se a atuação de diversas forças da natureza sobre os componentes que formam a paisagem. Tais forças podem ser classificadas como internas (endógenas, provenientes de dentro das camadas da Terra) e externas (exógenas, provenientes de fatores externos ao solo). Exemplos de forças internas são o vulcanismo e o tectonismo, enquanto exemplos de forças externas são a chuva, vento e rios (FRANCISCO, 2017). A intensidade de cada força sobre a paisagem pode variar de região para região. Um ambiente desértico sofrerá muito mais alterações devido ao vento do que à chuva, que por sua vez terá muito mais efeito em um ambiente subtropical.

Certos processos de alteração da paisagem mais imediatos, como deslizamentos de terra e erosão em menor escala, podem causar prejuízos sociais e econômicos, sendo desejável a sua prevenção ou redução dos efeitos. Um exemplo da capacidade de destruição destes processos pode ser verificado analisando-se a enchente de novembro de 2008 na cidade de Blumenau, quando ocorreram diversos deslizamentos que alteraram drasticamente a paisagem em vários pontos da cidade (RIBAS, 2015).

> O evento meteorológico extremo de 2008 [...] pode ser compreendido como a associação de dois cenários predisponentes à manifestação generalizada das instabilidades em taludes e encostas naturais registradas na área de estudo. O primeiro deles resulta de um acumulado de precipitações contínuas a partir do mês de julho e que se intensificaram a partir de outubro daquele ano [...]. O segundo cenário passa a se configurar a partir do dia 18 de novembro, com o ápice nos dias 22 e 23, quando os totais diários registrados ficaram torno de 250 mm de chuva. O acumulado mensal resultou em 1.001,7 mm, superando em 6 vezes a média histórica. (POZZOBON, 2013, p. 23).

A fim de reduzir ou evitar os prejuízos causados por esses eventos, foram desenvolvidos diversos métodos para identificar situações de risco. Mesmo regiões que nunca sofreram eventos no passado podem vir a sofrer acidentes no futuro, devido a condições induzidas pelo homem, como alterações na topologia natural (HIGHLAND; BOBROWSKY, 2008, p. 59).

No entanto, a maioria das técnicas de análise dos riscos de deslizamentos ainda depende de estudos de campo realizados por pessoas qualificadas ou equipamento sofisticado, o que nem sempre se encontra disponível. Até mesmo as ferramentas computacionais utilizadas para a simulação de volume da massa de deslizamento e estabilidade de encostas podem ser complicadas demais para o entendimento da população (HIGHLAND; BOBROWSKY, 2008, p. 56-61). Embora isso seja um resultado da complexidade necessária da simulação para providenciar uma previsão confiável, assim como nas outras técnicas a

presença de um especialista ainda é necessária para traduzir os dados coletados em informações para a população (HIGHLAND; BOBROWSKY, 2008, p. 62).

Tendo em vista os métodos comentados, este trabalho apresenta o desenvolvimento de um modelo de simulação para deslizamentos de terra (assim como outros processos de alteração da paisagem) que possa ser utilizado para representar possíveis eventos reais de uma forma simplificada suficiente para o entendimento por usuários sem conhecimento técnico. Tal programa poderia ser utilizado como ferramenta auxiliar em estudos e programas educacionais, além de servir como base para o desenvolvimento de programas mais elaborados, capazes de realizar simulações mais precisas.

1.1 OBJETIVOS

O objetivo deste projeto é verificar a aplicação de algoritmos computacionais para a representação de fenômenos reais de dinâmica do relevo, por meio do desenvolvimento de uma aplicação 3D.

Os objetivos específicos são:

a) desenvolver uma aplicação que permita a visualização e possível edição de uma paisagem real ou virtual através do motor de terreno da plataforma Unity;
b) estudar, elaborar e aplicar algoritmos de transformação sobre os dados do relevo que representem, dentro de um grau razoável de acurácia, os processos que ocorrem sobre o ambiente e alteram a paisagem;
c) analisar a viabilidade do formato do algoritmo dos pontos de vista de desenvolvimento, resultados e desempenho;
d) coletar, a partir dos dados do relevo, informações que sejam de interesse para estudos de dinâmica de relevo.

1.2 ESTRUTURA

Na fundamentação teórica serão analisados dois programas com funcionalidades similares às que foram apresentadas acima. Também serão apresentados conhecimentos básicos da área de geologia e computação que serão utilizados para guiar o desenvolvimento da aplicação.

No desenvolvimento serão apresentados os detalhes técnicos da aplicação, incluindo especificações, técnicas, ferramentas e os diversos algoritmos utilizados durante o processo. Por último, serão apresentados os resultados dos testes e as conclusões que podem ser tiradas, assim como possíveis extensões do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados e discutidos conceitos básicos relacionados aos processos de dinâmica de relevo e representação computacional de relevo que foram necessários para o desenvolvimento. Também serão analisados dois programas com funcionalidades similares à aplicação desenvolvida.

2.1 COMPOSIÇÃO DO RELEVO

Para o desenvolvimento de uma simulação capaz de representar os processos de dinâmica de relevo com um determinado grau de acurácia, é necessário compreender quais elementos formadores do relevo podem afetar estes processos. Para as simulações que serão desenvolvidas neste trabalho, serão consideradas a camada da rocha-matriz, a concentração de água no solo e a superfície da paisagem.

O Sistema Brasileiro de Classificação de Solos (EMPRABA, 2006, p. 32) classifica o solo como "uma coleção de corpos naturais, constituídos por partes sólidas, líquidas e gasosas, tridimensionais, dinâmicos, formados por materiais minerais e orgânicos". O solo pode ser dividido em camadas (horizontes) de acordo com variações em sua composição e estrutura. A partir de uma determinada profundidade, o material gradualmente passa de solo para um material de propriedades rochosas. Nas condições climáticas do Brasil, é comum encontrar solos que alcançam 200 cm de profundidade (EMBRAPA, 2006 p. 32).

O solo se forma através das ações do intemperismo sobre o substrato rochoso, transformando-o no material granuloso que passa a ser considerado solo (CAPUTO, 1988 p. 14). Quando esse material permanece no local de origem, é classificado como solo residual, e se verifica uma transição gradual entre os horizontes do solo e o substrato rochoso. Quando o solo é transportado para um local diferente da origem, passa a ser classificado como solo sedimentar (CAPUTO, 1988 p. 15).

A composição do solo é provavelmente o fator mais importante a ser considerado na análise dos processos de dinâmica do relevo. Particularmente em casos de deslizamentos de encostas, uma das suas principais causas é a saturação de água no solo que sofre a alteração, que por sua vez pode ser influenciada por diversos eventos externos, como chuvas intensas, inundações e até mesmo eventos causados por ação humana, como vazamento de tubulações (HIGHLAND; BOBROWSKY, 2008 p. 41).

Finalmente, a cobertura da superfície do solo é outro fator importante ao analisar a ocorrência de eventos como deslizamentos de encostas. Em situações onde o solo encontra-se exposto ao ambiente, espera-se uma maior ocorrência de fenômenos como infiltração de água

e erosão. Já solo coberto por plantas de pequeno porte, como grama, ganham resistência aos processos erosivos. A presença de flora de maior porte, como arbustos e árvores, tanto reforça a resistência do solo através de suas raízes (HIGHLAND; BOBROWSKY, 2008 p. 119) quanto reduz os efeitos do ambiente (forças físicas do vento, sol e chuva) (HACKSPACHER, 2011 p. 55).

2.2 RENDERIZAÇÃO DE TERRENO NA PLATAFORMA UNITY

Unity é uma plataforma de desenvolvimento de aplicativos 2D e 3D. Dentre as diversas funcionalidades fornecidas por sua biblioteca, pode ser encontrado o motor de terreno do Unity. Em tempo de execução, a renderização do terreno é otimizada para maior eficiência (Figura 1), enquanto que no editor, uma série de ferramentas facilitam a criação e edição de terrenos (UNITY TECHNOLOGIES, 2016).

Com exceção do posicionamento de árvores e as propriedades gerais do terreno, todas as ferramentas de edição de terreno do Unity funcionam por meio de pincéis, com forma, tamanho e opacidade configuráveis. A utilização destas ferramentas é similar a de um software de pintura, com a aplicação dos pincéis sobre o objeto de terreno (UNITY TECHNOLOGIES, 2016).

Figura 1 – Exemplo de terreno renderizado na plataforma Unity

Fonte: Unity Technologies (2016).

Para editar o relevo, o Unity disponibiliza três ferramentas: elevação e rebaixamento, definição de altura e suavização. A ferramenta de elevação e rebaixamento permite que o usuário altere a altitude do relevo com base no pincel selecionado. A ferramenta de definição de altura permite selecionar uma altitude fixa e desenhá-la no terreno, sendo útil para criar uma trilha nivelada, por exemplo. Por último, a ferramenta de suavização rateia as alturas sob o pincel, reduzindo inclinações extremamente acentuadas. O Unity também permite fazer a

exportação ou importação do relevo por meio de mapas de altura[1] (UNITY TECHNOLOGIES, 2016).

Além destas ferramentas, o editor também disponibiliza ferramentas para aplicação de texturas, colocação de árvores, grama e outros adereços, e definição de zonas de vento para fins estéticos. Também podem ser alteradas diversas propriedades relacionadas à renderização e simulação do terreno durante a execução (UNITY TECHNOLOGIES, 2016).

Embora seja possível alterar as propriedades do terreno em tempo de execução, as ferramentas de edição só se encontram disponíveis no editor Unity nativamente, o que pode ser um empecilho no desenvolvimento da aplicação, já que necessitaria da utilização do editor Unity para a configuração do terreno. Alternativamente, poder-se-ia desenvolver edição de terreno própria para o software proposto ou permitir a importação de mapas de altura, que podem ser editados por programas externos.

2.3 REPRESENTAÇÃO COMPUTACIONAL DO RELEVO

Antes de iniciar o desenvolvimento da simulação, é necessário entender as técnicas utilizadas para se armazenar e representar um relevo em uma aplicação computacional. Conforme visto na seção anterior, a plataforma Unity utiliza mapas de altura (*heightmaps*) como forma de armazenamento das informações de relevo.

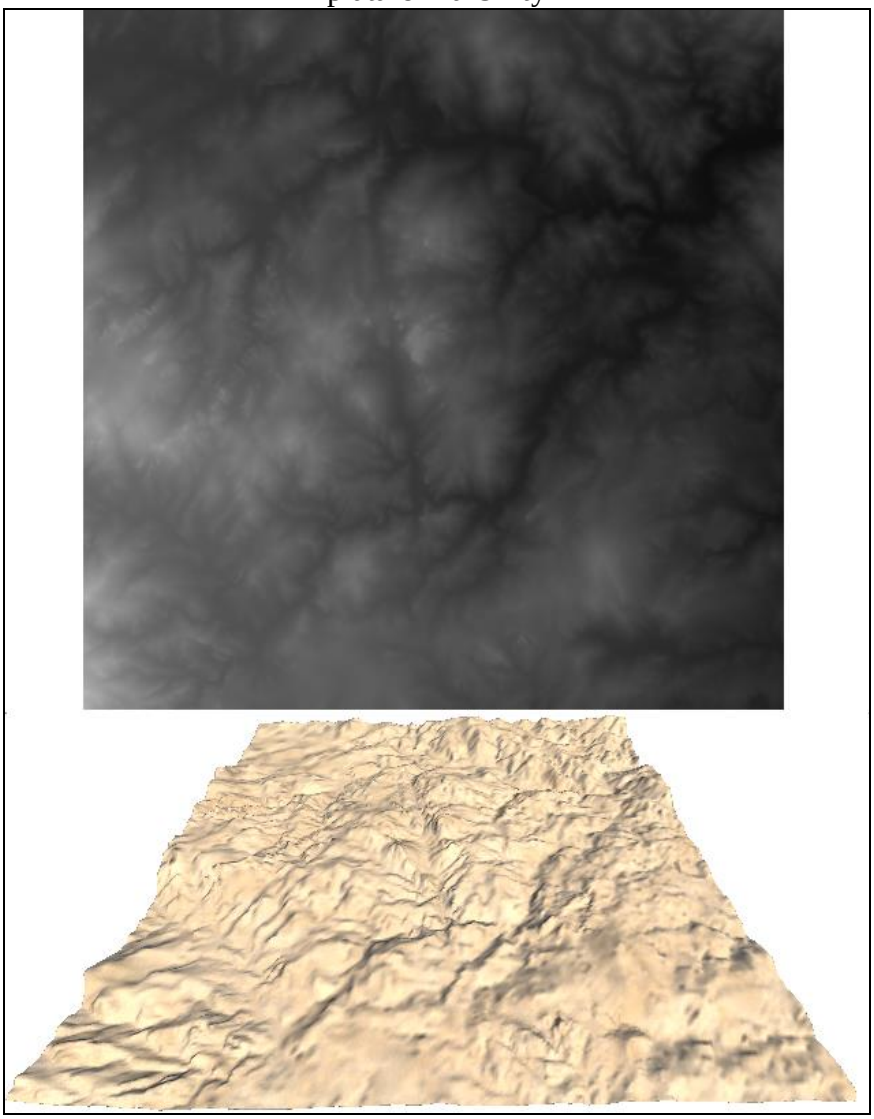
Um mapa de altura consiste nos valores de altura do relevo, organizados em uma matriz, de tal forma que essas alturas podem ser mapeadas a suas respectivas coordenadas (x,y). Essa matriz pode ser visualizada como uma imagem bitmap em escala de cinza, onde cada valor de altura corresponde ao valor entre preto e branco de um pixel na imagem. Essa característica tem a vantagem de oferecer uma visualização compreensível do terreno na forma de bitmap e possibilita o uso de editores de imagem para criar ou modificar um relevo (SNOOK, 2003 p. 120).

Na plataforma Unity, os objetos de terreno utilizam arquivos no formato RAW como entrada. Estes arquivos são então carregados na memória como matrizes de valores de ponto flutuantes de zero a um. No entanto, é possível construir uma matriz equivalente manualmente ou a partir de um arquivo de imagem em tempo de execução e carregá-la no objeto de terreno, como foi realizado na figura 2. É importante ressaltar que, embora o Unity aceite mapas de resoluções variadas, a matriz deve ter sempre um formato quadrado. Entretanto, o Unity

[1] Representação de um relevo por meio de uma imagem em escala de cinza.

permite que o objeto de terreno seja distorcido para ter uma visualização retangular (UNITY TECHNOLOGIES, 2016).

Figura 2 – Exemplo de mapa de altura seguido pelo objeto de terreno correspondente na plataforma Unity



Fonte: Elaborado pelo autor.

2.1 CRAFTSCAPE

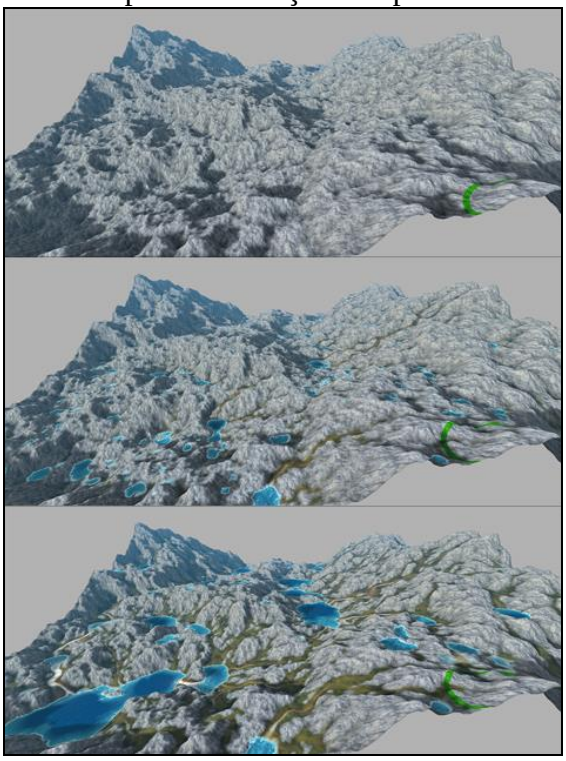
A aplicação WebGL Crafscape é um simulador de dinâmica de relevo inspirado pelo jogo de computador From Dust (BOESCH, 2011). A aplicação simula erosão hidráulica sobre um terreno inicialmente rochoso. O terreno é representado por uma malha hexagonal, o que permite uma representação mais fiel do que uma malha de grade tradicional. Também há a simulação de corpos de água no terreno, feita através de um simples algoritmo baseado na diferença entre alturas. Com base nessa funcionalidade, é realizada a simulação da erosão hidráulica, convertendo-se o terreno rochoso em solo com base no fluxo da água e

transportando-se o solo na direção da corrente, criando um processo de escavação por onde a água passa (BOESCH, 2011).

Para gerar os corpos de água no terreno, a aplicação simula o processo de precipitação, lentamente submetendo o relevo como um todo ao processo de erosão, enquanto a água se acumula em pontos específicos e passa a escorrer para as partes mais baixas, formando rios e vales. Para evitar uma sobrecarga da paisagem com corpos de água, a aplicação também simula o processo de evaporação, gradativamente removendo a água da paisagem. Esses dois processos, assim como o processo de erosão em si, podem ser desabilitados pelo usuário se desejado. A aplicação também permite que o usuário adicione ou remova manualmente rocha, solo ou água à paisagem (BOESCH, 2011).

Embora a simulação da erosão hidráulica e dinâmica de fluidos tenha um resultado satisfatório (Figura 3), a aplicação não trata de outros processos relevantes para uma análise mais completa da dinâmica do relevo, como a erosão eólica e grandes deslizamentos de terra. As opções de parametrização disponíveis também são limitadas, não possibilitando a alteração de valores específicos de precipitação e evaporação, ou a definição da cobertura do solo (BOESCH, 2011).

Figura 3 - Exemplo de execução do aplicativo Craftscape



Fonte: Boesch (2011).

2.2 FROM DUST

From Dust (UBISOFT, 2011) é um jogo de computador projetado por Eric Chahi. O jogo, lançado em julho de 2011, possui uma simulação complexa de dinâmica de relevo em tempo real, tratando processos como erosão hidráulica, deslizamento de areia e formação de dunas, crescimento de cobertura vegetal (Figura 4) e até ciclos de marés e vulcanismo (Figura 5) (UBISOFT, 2011).

Figura 4 - Exemplo de paisagem afetada pela erosão no jogo From Dust

Fonte: Ubisoft (2011).

Figura 5 – Conceito de vulcanismo do jogo From Dust

Fonte: Ubisoft (2011).

O jogo coloca o jogador no papel de um “sopro” invocado por uma tribo primitiva, com o poder de alterar a paisagem coletando e depositando os materiais que a formam, como areia, água e magma. O objetivo do jogo é proteger a tribo de desastres naturais e ajudá-la em sua migração, desviando o curso de um rio para permitir a travessia, por exemplo. O jogo também possui um objetivo opcional de cobrir a maioria do terreno com vegetação (UBISOFT, 2011).

O terreno é modelado e a simulação computada através de uma grade, podendo alcançar de 100.000 a 200.000 células para calcular por frame. Por ter um custo alto de processamento, a simulação foi desenvolvida utilizando uma linguagem de baixo nível similar

a VS Assembly, o que permitiu que a simulação tirasse o maior proveito possível da memória cache do processador, resultando em uma simulação mais rápida (CHAHI, 2011).

Embora a simulação seja robusta, ela continua sendo apenas mais uma funcionalidade do jogo. As paisagens são pré-definidas e compactadas e os agentes alteradores do relevo encontram-se em poucos locais específicos e fixos, como uma nascente de água ou um vulcão. A maioria dos efeitos acaba sendo causado pelas ações diretas do jogador (UBISOFT, 2011).

3 DESENVOLVIMENTO

Nesta seção estão detalhados os requisitos, especificação e o processo de desenvolvimento da aplicação proposta.

3.1 REQUISITOS

O simulador proposto neste trabalho deve:

a) permitir a visualização de uma representação de uma paisagem virtual em 3D, com câmera ajustável (Requisito Funciona - RF);
b) permitir a criação e edição de paisagens virtuais, com informações sobre o relevo da rocha e do solo, e a cobertura do solo (RF);
c) permitir a importação do relevo de uma paisagem a partir de arquivos de mapa de altura (RF);
d) permitir a parametrização de informações relativas aos agentes alteradores da paisagem (RF);
e) simular em tempo real as alterações sobre uma paisagem virtual, permitindo a verificação de fenômenos como deslizamentos e erosão (RF);
f) calcular e exibir as áreas mais afetadas após a simulação com base em mapas de dispersão, projetados na própria paisagem (RF);
g) permitir a visualização de estatísticas relacionadas à simulação, como a quantidade de pontos de deslizamento e a massa do solo deslocado (RF);
h) salvar os dados de uma paisagem ou simulação no disco rígido do computador em que o simulador estiver executando (RF);
i) executar em desktop (Requisito Não Funcional - RNF);
j) ser desenvolvido usando a plataforma Unity para visualização e linguagem C# para scripts (RNF).

3.2 ESPECIFICAÇÃO

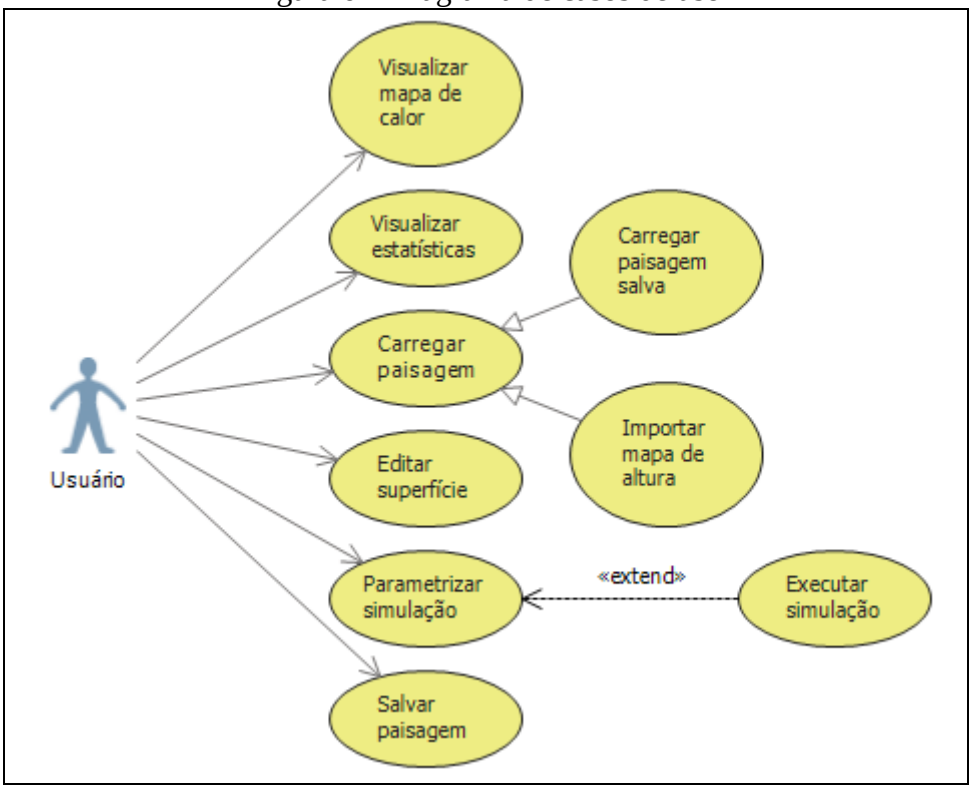
Nesta seção encontram-se uma descrição da estrutura do programa, assim como os diagramas de classes e casos de uso.

3.2.1 CASOS DE USO

A figura 6 descreve os principais casos de uso do programa. Após a inicialização, o usuário pode a qualquer momento optar por ativar a visualização de um mapa de calor, visualizar a tabela de estatísticas, carregar uma paisagem (tanto através de um arquivo salvo

anteriormente quanto através da importação de um mapa de altura), editar os tipos de superfície da paisagem, parametrizar a simulação e salvar a paisagem para acesso posterior. Ao parametrizar a simulação, o usuário também poderá ativar ou desativar a execução da mesma. Como o programa inicia com uma paisagem vazia, normalmente o usuário carregará uma paisagem antes de realizar qualquer um dos outros casos de uso, embora estejam disponíveis.

Figura 6 – Diagrama de casos de uso



Fonte: Elaborado pelo autor.

3.2.2 DIAGRAMA DE CLASSES

Nas Figuras 7 a 11 estão representados os diagramas das classes, divididas por *namespace*. Para se construir os diagramas foi utilizada a ferramenta Visual Studio 2013. Cada *namespace* representa uma tela do programa, com exceção do *namespace* `Utility`, que contém algoritmos e informações de uso geral.

O *namespace* `TerrainView` (Figura 7) representa a tela principal da aplicação, onde pode ser visualizada a simulação e acessadas as outras telas. A classe `TerrainControl` controla a execução da simulação, assim como o acesso aos dados do terreno e a atualização de seus objetos 3D respectivos. A classe `GameControl` providencia acesso às operações comandadas por outras telas, como a importação de um mapa de altura e atualização das

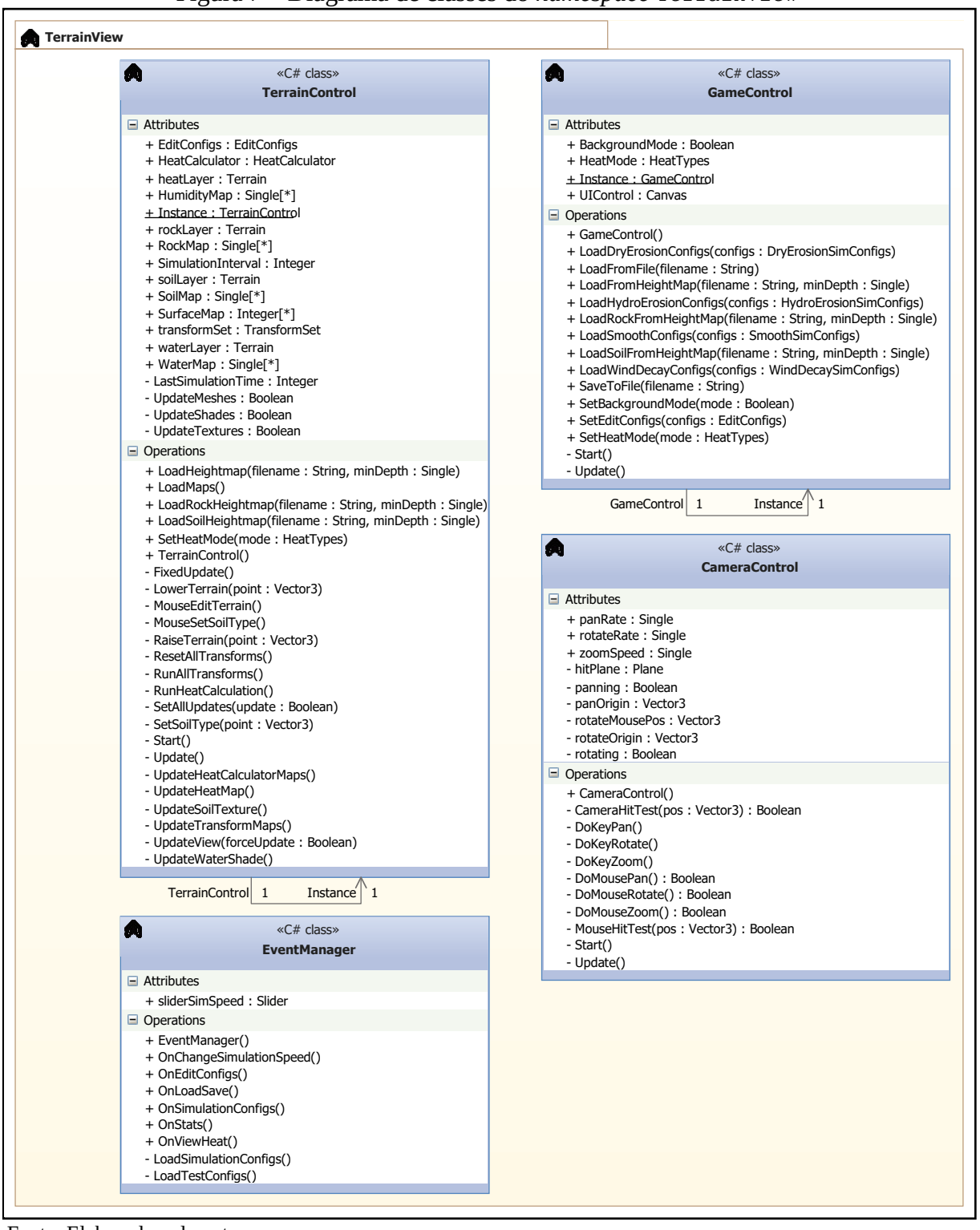
configurações de simulação. As classes `EventManager` e `CameraControl` são responsáveis pelos eventos da interface de usuário e controle da visualização, respectivamente.

O *namespace* `Utility::TerrainAlgorithm` (Figura 8) contém os algoritmos de transformação desenvolvidos. Cada algoritmo herda da classe `TerrainTransform`, que contém uma referência aos dados usados nas transformações. Estes algoritmos são agrupados na classe `TransformSet`, a qual permite acesso geral à todas as classes de algoritmos suportados pela aplicação. Uma instância dessa classe é instanciada e utilizada na classe `TerrainControl`. Cada algoritmo também possui sua classe de parâmetros específica. O enumerador `TransformIndex` representa todos os algoritmos presentes no código (incluindo algoritmos inativos). As funções delegadas `LocalTransform` e `LocalTransformEx` são utilizadas para generalizar as operações locais de transformação para cada célula do mapa de altura.

O *namespace* `Utility::HeatAlgorithm` (Figura 9) contém apenas a classe `HeatCalculator`, que é responsável por efetuar os cálculos de mapas de calor, e o enumerador de tipos de mapa de calor `HeatTypes`. Outras entidades presentes no *namespace* `Utility` estão descritas na Figura 10. Por exemplo, enumeradores para direções cardinais e tipos de superfície, uma classe de extensão ao enumerador de direções `DirectionsEx` e uma classe para configurações de edição de superfície `EditConfigs`.

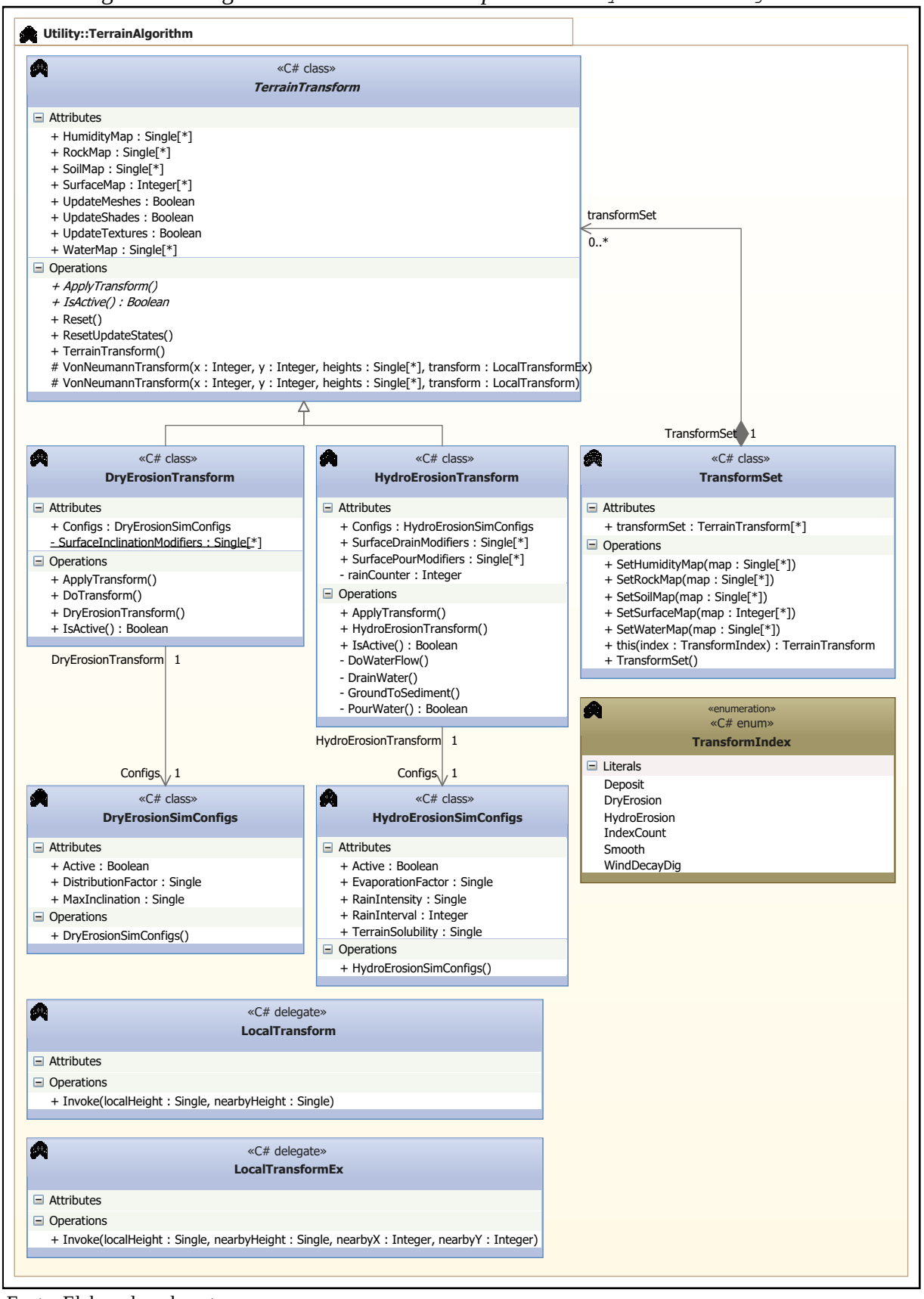
Os *namespaces* `SimulationConfigsScreen` (Figura 10), `LoadSaveScreen`, `ViewScreen` e `EditConfigsScreen` (Figura 11) representam as telas de configuração de simulação, carga e salvamento dos dados da paisagem, visualização de mapas de calor e configuração de edição, respectivamente. As estruturas destes conjuntos são similares, com uma classe `UIControl` responsável por agrupar e acessar os controles de entrada da tela, e uma classe `EventManager` responsável pelos eventos chamados pelo usuário. Uma exceção a este formato é o *namespace* `ViewScreen`, onde a classe `UIControl` também é responsável pelo único evento da tela. No *namespace* `SimulationConfigsScreen` (representa uma tela mais complexa) os controles encontram-se agrupados nas classes `DryErosionConfigsControl` e `HydroErosionConfigsControl`, representando os parâmetros para erosão térmica e hidráulica, respectivamente.

Figura 7 – Diagrama de classes do *namespace* `TerrainView`



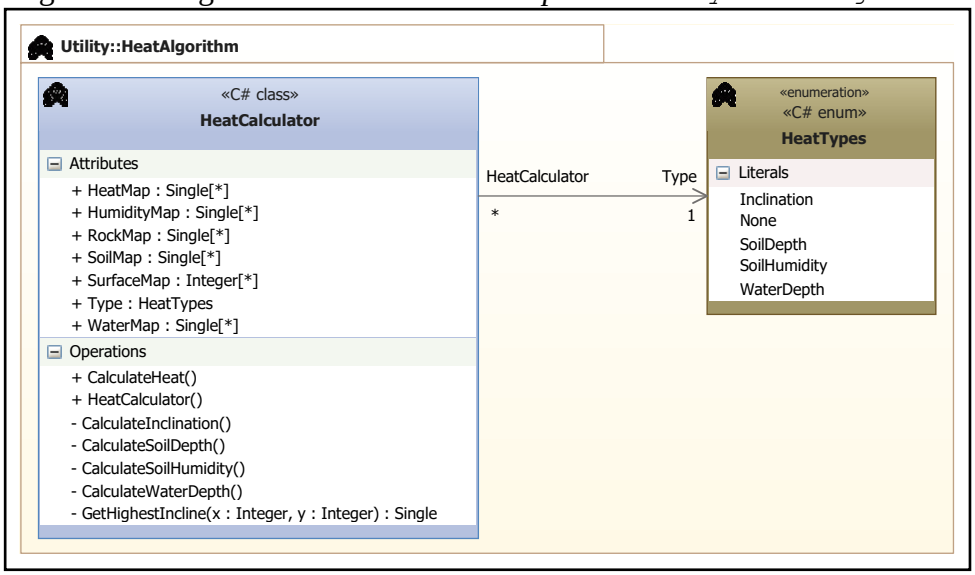
Fonte: Elaborado pelo autor.

Figura 8 – Diagrama de classes do *namespace* `Utility::TerrainAlgorithm`



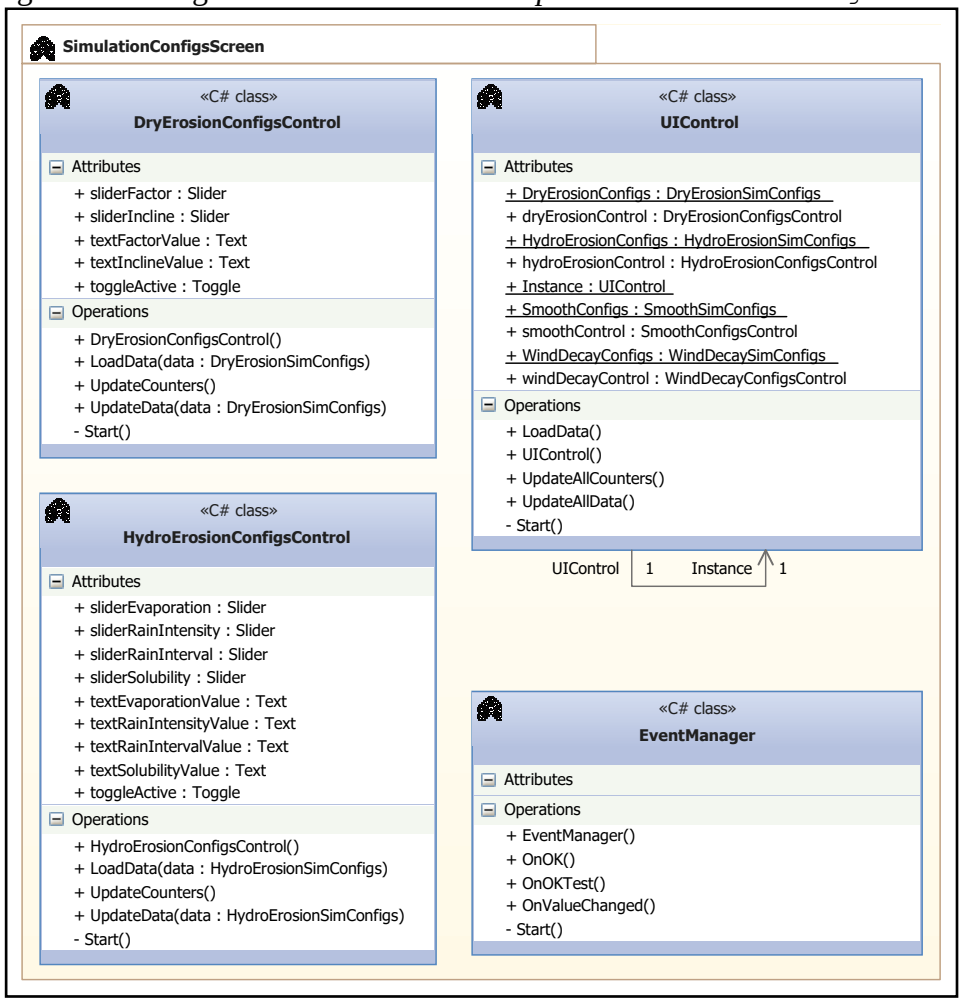
Fonte: Elaborado pelo autor.

Figura 9 – Diagrama de classes do *namespace* `Utility::HeatAlgorithm`



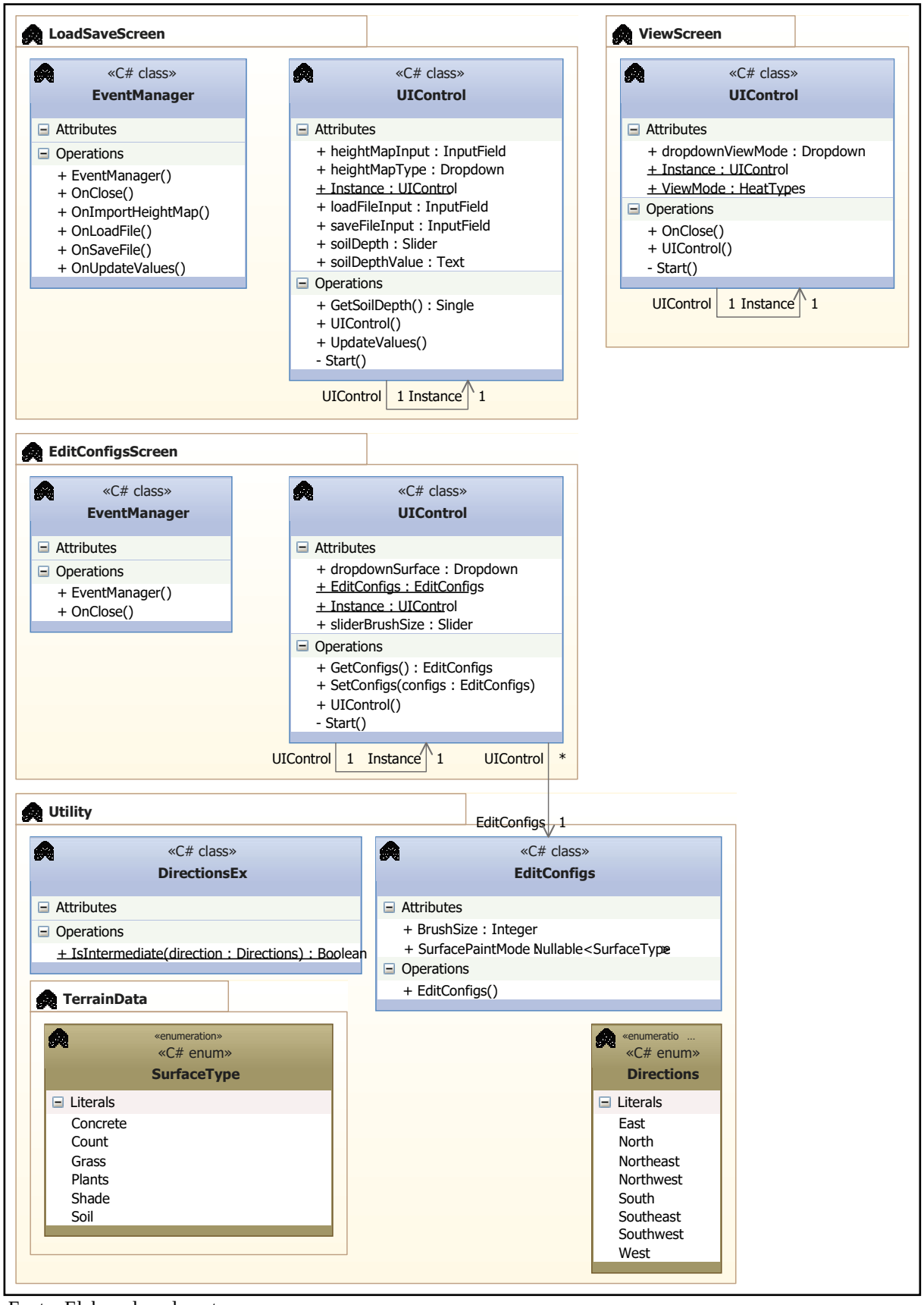
Fonte: Elaborado pelo autor.

Figura 10 – Diagrama de classes do *namespace* `SimulationConfigsScreen`



Fonte: Elaborado pelo autor.

Figura 11 – Diagrama de classes dos *namespaces* `LoadSaveScreen`, `ViewScreen`, `EditConfigsScreen` e `Utility`



Fonte: Elaborado pelo autor.

Nota-se que muitas das classes descritas contêm uma instância estática de si própria (*singleton*) denominada `Instance`. Isto foi feito para facilitar o relacionamento entre as classes, já que estas estão anexadas a objetos Unity e não podem ser instanciadas entre si. Também vale ressaltar que a aplicação contém classes não apresentadas nos diagramas. Estas classes representam algoritmos parciais que são discutidos na seção 3.3, mas encontram-se inativos na aplicação final e, portanto, foram desconsideradas na geração dos diagramas.

3.3 IMPLEMENTAÇÃO

Nesta seção estão descritas as ferramentas utilizadas para o desenvolvimento, a técnica de transformação de mapas de altura, e o detalhamento dos algoritmos de erosão térmica e hidráulica, assim como as adaptações e otimizações que foram realizadas sobre os mesmos. Também serão descritos os testes e resultados que foram obtidos ao final do processo de implementação.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

No desenvolvimento da aplicação foram utilizadas as ferramentas Unity 5.5.2f1 personal e Microsoft Visual Studio Ultimate 2013 12.0.30723.00 Update 3. Foi utilizado também o plugin Microsoft Visual Studio Tools for Unity 2.3.0.0. Para o desenvolvimento dos scripts foi utilizada a linguagem C# com referências à biblioteca Unity.

A seguir, descrevem-se as fórmulas que foram utilizadas como base no desenvolvimento da aplicação, bem como as adaptações que foram realizadas com base nas pesquisas realizadas anteriormente.

3.3.1.1 ALGORITMOS DE TRANSFORMAÇÃO DE RELEVO

Tendo em vista que no Unity 3D os dados de altura de um objeto de terreno são armazenados no formato de *heightmap*, podem-se aplicar os conceitos utilizados na filtragem de imagens no desenvolvimento de algoritmos que transformam o relevo. Um dos algoritmos mais simples utilizados em imagens é o *box blur*, descrito no Quadro 1. Este algoritmo altera cada valor da matriz para a média aritmética do mesmo e seus vizinhos, resultando na redução de ruído e desfocagem (*blurring*) da imagem como um todo (CHANDEL; GUPTA, 2013, v. 3 ed. 10 p. 198).

Na primeira aplicação do algoritmo de *box blur* sobre os dados de terreno do Unity, foi considerada inicialmente uma matriz retangular de 3x3 (conhecida como elemento estruturante) centralizada sobre a célula (denominada semente) que será atualizada. As

alterações não são feitas imediatamente no terreno em alteração, sendo salvas em uma matriz secundária que substituirá a matriz inicial no término da execução. Desta forma, evita-se que alterações feitas em um ponto afetem alterações nos próximos pontos.

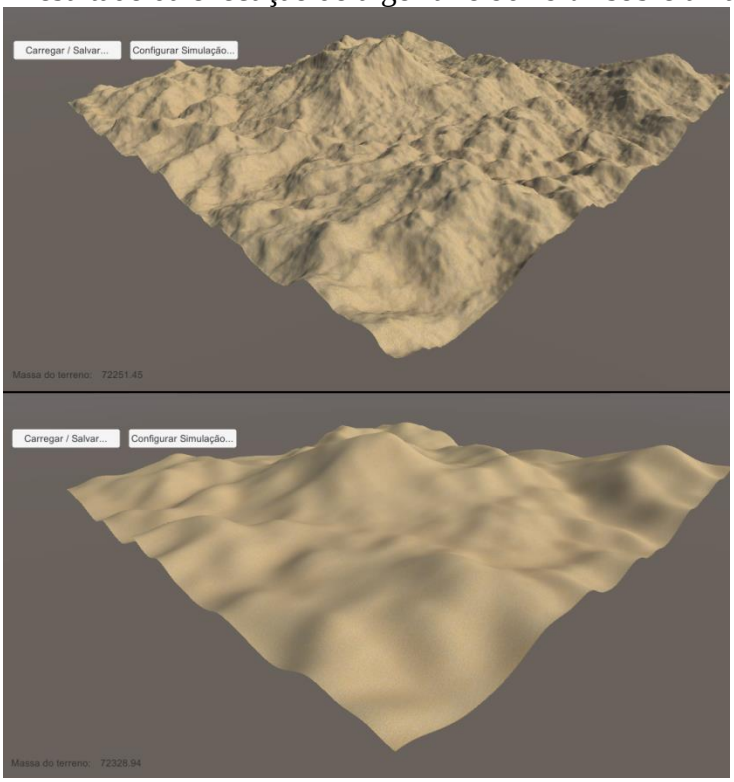
Quadro 1 – Algoritmo *box blur* na linguagem C#

```
private void BoxBlur(float[,] matrix)
{
    int maxX = matrix.GetLength(0);
    int maxY = matrix.GetLength(1);

    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y < maxY; y++)
        {
            float sum = 0;
            int count = 0;

            for (int relX = -1; relX <= 1; relX++)
            {
                int absX = x + relX;
                if (absX < 0 || absX >= maxX)
                    continue;

                for (int relY = -1; relY <= 1; relY++)
                {
                    int absY = y + relY;
                    if (absY < 0 || absY >= maxY)
                        continue;

                    sum += matrix[absX, absY];
                    count++;
                }
            }

            if (count > 0)
                matrix[x, y] = sum / count;
        }
    }
}
```

Fonte: Elaborado pelo autor.

A execução do algoritmo elimina pequenas características do relevo e suaviza inclinações, um resultado que pode ser comparado ao assentamento de areia solta (Figura 12). Foi desenvolvida então a possibilidade de parametrização de certos componentes do algoritmo, inicialmente a área da matriz considerada no cálculo da média, assim como um "fator de transformação", que permite reduzir a alteração do terreno a cada iteração do algoritmo.

Figura 12 - Resultado da execução do algoritmo *box blur* sobre uma paisagem



Fonte: Elaborado pelo autor.

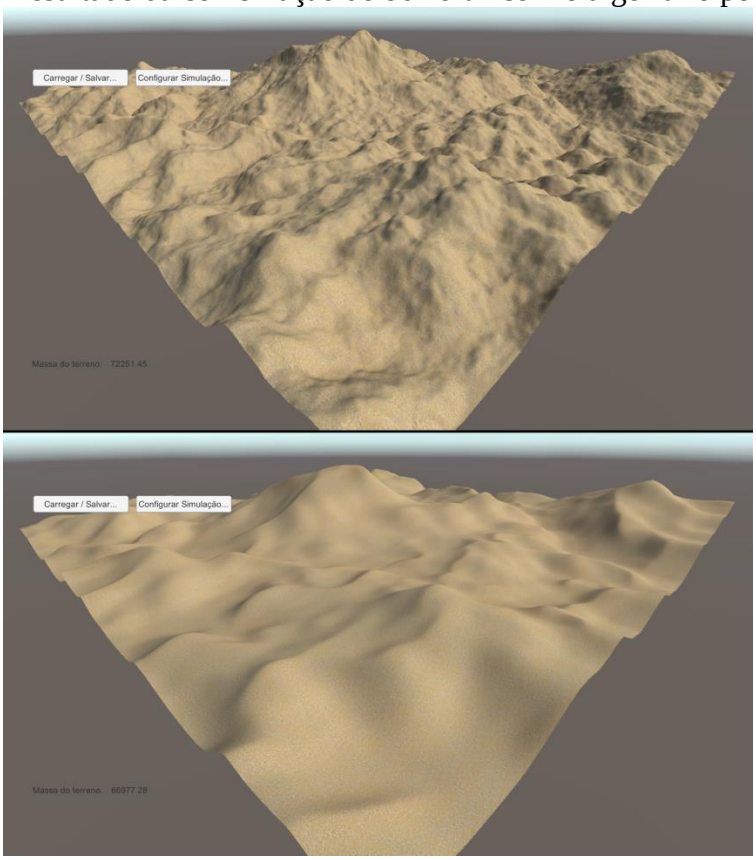
Para avaliar as possibilidades que esse formato de algoritmo providencia, foi realizada uma alteração no primeiro algoritmo, cujo resultado pode ser visualizado no Quadro 2. Ao invés de considerar a matriz de vizinhos completa na aplicação da média, é considerada apenas uma parte da mesma, descartando-se células ao norte na aplicação da média. Além disso, são descartadas alterações que elevem partes do terreno, permitindo apenas o rebaixamento dos pontos de altura do mesmo. O resultado é uma transformação similar ao efeito de um vento sul-norte varrendo uma paisagem arenosa, causando o desgaste do lado sul dos morros. Aplicando-se os dois algoritmos simultaneamente, verifica-se a formação de estruturas similares a dunas (Figura 13).

Quadro 2 – Algoritmo personalizado com base no *box blur*, na linguagem C#

```
public void WindDecay(float[,] matrix)
{
    int maxX = matrix.GetLength(0);
    int maxY = matrix.GetLength(1);

    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y <maxY; y++)
        {
            float sum = 0;
            int count = 0;

            for (int relX = -1; relX <= 1; relX++)
            {
                int absX = x + relX;
                if (absX < 0 || absX >= maxX)
                    continue;

                // Considerar apenas vizinhos em coordenadas Y iguais ou
                // inferiores ao ponto central
                for (int relY = -1; relY <= 0; relY++)
                {
                    int absY = y + relY;
                    if (absY < 0 || absY >= maxY)
                        continue;

                    sum += matrix[absX, absY];
                    count++;
                }
            }

            if (count > 0)
            {
                // Apenas efetuar a alteração caso o novo valor seja
                // inferior ao atual
                float avg = sum / count;
                if (avg < matrix[x, y])
                    matrix[x, y] = avg;
            }
        }
    }
}
```

Fonte: Elaborado pelo autor.

Figura 13 - Resultado da combinação do *box blur* com o algoritmo personalizado

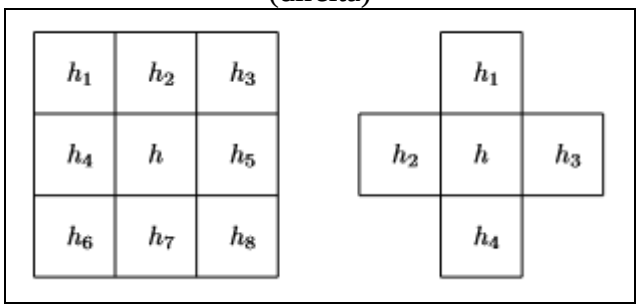


Fonte: Elaborado pelo autor.

3.3.1.2 OTIMIZAÇÕES

A fim de reduzir o número de operações executadas a cada iteração do algoritmo, foram realizados alguns ajustes ao seu formato. Os algoritmos de transformação apresentados até agora realizavam operações utilizando uma vizinhança de 8 células, também conhecida como vizinhança Moore. Os algoritmos a seguir foram desenvolvidos utilizando-se uma vizinhança de 4 células (Vizinhança Von Neumann). A diferença entre os dois tipos de vizinhança pode ser vista na Figura 14. Nota-se que a vizinhança Von Neumann não realiza operações nas células em posições diagonais. Por utilizar uma vizinhança mais limitada, esta diferença causa um impacto negativo na qualidade da simulação, mas no escopo da aplicação proposta, esse impacto pode ser ignorado para fins de desempenho.

Figura 14 – Representação visual das vizinhanças de Moore (esquerda) e Von Neumann (direita)



Fonte: Olsen (2004).

Com a utilização da vizinhança de Von Neumann em todos os algoritmos, a quantidade de operações efetuadas sobre células vizinhas é reduzida pela metade. Por ser uma vizinhança mais simples, optou-se por escrever um código linear ao invés de utilizar loops, como estava sendo feito para vizinhanças Moore. O código resultante pode ser visto no Quadro 3.

Quadro 3 – Algoritmo linear de transformação utilizando a vizinhança Von Neumann

```
public void VonNeumann (int x, int y, float[,] matrix)
{
    // "Transform" representa as operações de transformação da matriz
    if (x != 0)
    {
        Transform( (x - 1), y);
    }
    if (y != 0)
    {
        Transform(x, (y - 1));
    }
    if (x != heights.GetLength(0) - 1)
    {
        Transform( (x + 1), y);
    }
    if (y != heights.GetLength(1) - 1)
    {
        Transform(x, (y + 1));
    }
}
```

Fonte: Elaborado pelo autor.

Durante os testes, foi verificado que a execução dos algoritmos de transformação causa um grande impacto no desempenho da visualização 3D. Embora a maior parte deste impacto seja causada pelos algoritmos em si, as rotinas de atualização da visualização 3D agravam ainda mais este impacto. Inicialmente, a cada execução das transformações, era realizada a atualização de todos os componentes que formam a visualização. Para tentar melhorar o desempenho, estes componentes foram separados em `mesh`, `texture` e `shade`. O componente `mesh` representa a estrutura tridimensional do relevo, o componente `texture` representa a

aparência da superfície e o componente `shade` representa a tonalidade da superfície (que é definida pela umidade do solo).

Nos algoritmos de transformação foram disponibilizadas variáveis correspondentes a estes componentes que podem ser ativadas em pontos específicos do algoritmo, para indicar que uma atualização no componente é necessária. A rotina de atualização então apenas realizará as atualizações necessárias. Após esta alteração, foi apresentado um pequeno ganho de desempenho em alguns casos, principalmente quando a estrutura do terreno se estabiliza e alterações na malha deixam de ser necessárias. No entanto, essa alteração não resultou em ganhos de desempenho visíveis durante a execução normal da simulação, quando todas as rotinas de simulação são necessárias.

3.3.1.3 ALGORITMOS DE EROSÃO

Para simular os efeitos da erosão no terreno foram utilizados os algoritmos detalhados em Olsen (2004, p. 5-6) como base. Estes algoritmos são direcionados à geração de relevo procedural, mas a sua implementação é realizada de uma maneira similar aos algoritmos de transformação vistos no capítulo anterior.

O primeiro algoritmo visa representar os efeitos da erosão térmica, que causa a queda de material em encostas e o seu acúmulo na base. O algoritmo atinge esse efeito através da movimentação de material de pontos mais altos para vizinhos mais baixos quando a diferença entre os dois ultrapasse um determinado valor. A quantidade de material a ser movimentada é inicialmente obtida através da fórmula `(factor * (diff - talus))`, onde `factor` representa um fator limitante de movimentação, `diff` representa a diferença entre as alturas, e `talus` representa a diferença máxima permitida. No entanto, como pode haver múltiplos vizinhos que satisfaçam essa condição, é necessário realizar a distribuição do material movimentado de forma proporcional às inclinações correspondentes, através da fórmula estendida `(factor * (maxDiff - talus) * (diff / sumDiff))`, onde os novos valores `maxDiff` e `sumDiff` representam a maior diferença encontrada entre os vizinhos e a soma das diferenças que ultrapassam o limite `talus`, respectivamente (OLSEN, 2004, p. 5-7). O algoritmo final está representado no Quadro 4, com a aplicação da fórmula em destaque. Segundo Olsen (2004, p. 6), o valor ideal para `factor` é 0,5, tendo em vista que valores maiores podem causar oscilações no processo de movimentação de solo e valores menores deixam o processo mais lento.

Quadro 4 – Algoritmo de erosão térmica na linguagem C# com a fórmula de distribuição de solo em destaque

```
public void DryErosion(float[,] matrix, float talus, float factor)
{
    int maxX = matrix.GetLength(0);
    int maxY = matrix.GetLength(1);
    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y < maxY; y++)
        {
            float maxDiff = 0;
            float sumDiff = 0;
            // Primeiro loop é necessário para totalizar as informações
            // necessárias para calcular as alterações
            VonNeumann(x, y, matrix,
                (int relX, int relY)
                {
                    float diff = matrix[x, y] – matrix[relX, relY];
                    if (diff > maxDiff)
                        maxDiff = diff;
                    if (diff > talus)
                        sumDiff += diff;
                }
            );
            // Se este valor for zero, o ponto está estabilizado
            if (sumDiff == 0)
                continue;
            float inclinationDifference = (maxDiff - talus);
            // Segundo loop é onde são feitas as alterações
            VonNeumann(x, y, matrix,
                (int relX, int relY)
                {
                    float diff = matrix[x, y] - matrix[relX, relY];
                    if (diff > talus)
                    {
                        float move = factor * (maxDiff - talus) * (diff /
                        sumDiff);
                        matrix[relX, relY] += move;
                        matrix[x, y] -= move;
                    }
                }
            );
        }
    }
}
```

Fonte: Elaborado pelo autor.

O segundo algoritmo simula a erosão a partir de forças hidráulicas, ou seja, o deslocamento de material causado pela precipitação e escoamento da água. A primeira parte do algoritmo efetua uma simulação simplificada da hidrografia do relevo através de um mapa auxiliar de volumes de água `water`, cujos valores de altura da superfície podem ser obtidos através de `(water[x,y] + matrix[x,y])`. A simulação da ação da chuva é realizada através da soma de um valor de precipitação `pour` a todos os valores em `water`. Então, é realizada a erosão, por meio da subtração de uma parcela proporcional à quantia de água presente sobre

cada célula dos valores de altura em `matrix`, de acordo com um fator de solubilidade `solubility`.

Após a precipitação, é realizado o escoamento da água através de um algoritmo similar ao utilizada na erosão térmica, que visa nivelar completamente as diferenças de alturas ao invés de se basear em um limite. A fórmula para distribuição é `(Min(water[x, y], (surface - avg)) * (diff / sumDiff))`, onde `water` é a matriz de volumes de água, `surface` representa o nível da superfície no local, `avg` é a média das alturas das superfícies vizinhas inferiores ao local, `diff` é a diferença entre as superfícies e `sumDiff` é a soma destas diferenças para todos os vizinhos inferiores. O algoritmo completo está representado no Quadro 5, com a fórmula em destaque.

Após o escoamento, ocorre a drenagem da água, através da subtração de um percentual `drain` de cada valor em `water`. Durante esta operação, são adicionados aos valores `matrix` os valores contidos na quantidade de água removida, seguindo a mesma proporção utilizada anteriormente. Os algoritmos de precipitação, erosão e drenagem estão representados no Quadro 6.

A aplicação destes algoritmos mostrou resultados satisfatórios (Figuras 15 e 16), mas não provou ser suficiente para uma simulação mais representativa da realidade. Propõe-se então uma expansão destes algoritmos, através da inclusão de parâmetros que não foram considerados em sua concepção inicial.

Quadro 5 – Algoritmo de escoamento de água na linguagem C# com a fórmula de distribuição de água em destaque

```
private void WaterFlow(float [,] matrix, float[,] water)
{
    int maxX = matrix.GetLength(0);
    int maxY = matrix.GetLength(1);
    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y < maxY; y++)
        {
            if (water[x, y] < 0) continue;
            float surface = matrix[x, y] + water[x, y];
            float avg = 0;
            int count = 0;
            float sumDiff = 0;
            VonNeumann(x, y, matrix,
                (int relX, int relY)
                {
                    float localSurface = matrix[relX, relY] + water[relX,
                    relY];
                    float diff = surface - localSurface;
                    if (diff < 0) continue;
                    sumDiff += diff;
                    avg += localSurface;
                    count++;
                }
            );
            // Se este valor for zero, a água está estabilizada
            if (sumDiff == 0) continue;

            avg /= count;
            VonNeumann(x, y, matrix,
                (int relX, int relY)
                {
                    float localSurface = matrix[relX, relY] + water[relX,
                    relY];
                    float diff = surface - localSurface;
                    if (diff < 0) continue;
                    float delta = Math.Min(water[x, y], (surface - avg)) *
                    (diff / sumDiff);
                    water[relX, relY] += delta;
                    water[x, y] -= delta;
                }
            );
        }
    }
}
```

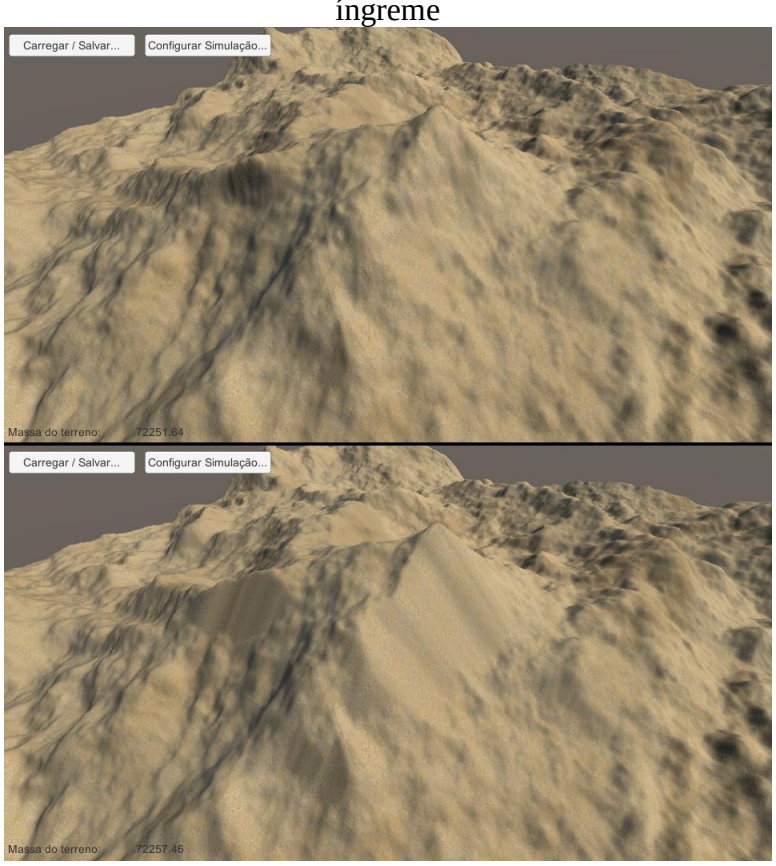
Fonte: Elaborado pelo autor.

Quadro 6 – Algoritmos de erosão hidráulica na linguagem C#

```
// Algoritmo de precipitação
private void PourWater(float[,] water, float pour)
{
    for (int x = 0; x < water.GetLength(0); x++)
    {
        for (int y = 0; y < water.GetLength(1); y++)
        {
            water[x, y] += pour;
        }
    }
}

// Algoritmo de remoção do solo
private void Dissolve(float[,] matrix, float[,] water, float solubility)
{
    for (int x = 0; x < matrix.GetLength(0); x++)
    {
        for (int y = 0; y < matrix.GetLength(1); y++)
        {
            matrix[x, y] -= solubility * water[x, y];
        }
    }
}

// Algoritmo de drenagem
private void DrainWater(float[,] matrix, float[,] water, float solubility,
float drain)
{
    for (int x = 0; x < matrix.GetLength(0); x++)
    {
        for (int y = 0; y < matrix.GetLength(1); y++)
        {
            float delta = waterVolume - (water[x, y] * drain);
            water[x, y] -= delta;
            matrix[x, y] += solubility * delta;
        }
    }
}
```

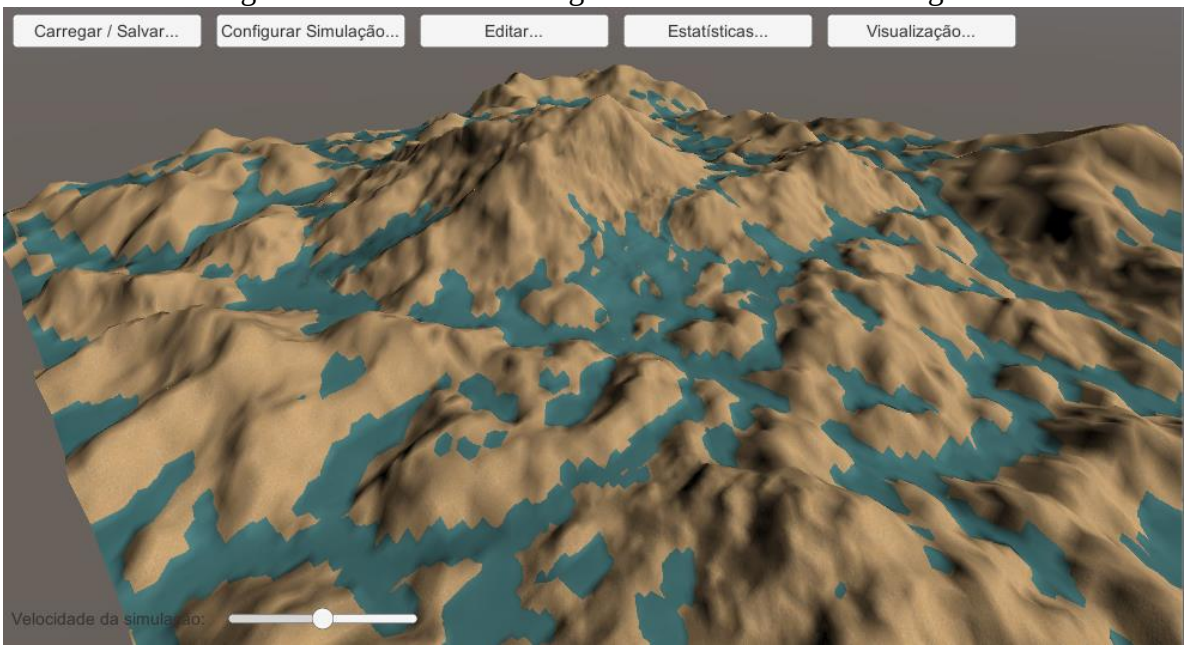
Fonte: Elaborado pelo autor.

Figura 15 - Resultado da execução do algoritmo de erosão térmica, com a queda do solo mais íngreme



Fonte: Elaborado pelo autor.

Figura 16 – Resultado do algoritmo de escoamento de água



Fonte: Elaborado pelo autor.

3.3.1.4 INCLUSÃO DA CAMADA DE ROCHA

Para aproximar a simulação de uma situação real, passa a se considerar um segundo objeto de terreno com as mesmas dimensões do terreno original, mas com um mapa de altura

distinto. Este novo mapa representará a estrutura da camada de rocha presente abaixo do solo. Tendo as matrizes de altura `soil` e `rock`, pode se calcular uma quantidade de solo presente nas coordenadas (x, y) através de `(soil[x, y] – rock[x, y])`. Nota-se que a altura do solo em qualquer célula deve ser igual ou superior à altura da rocha, senão haveria a possibilidade de existirem células contendo quantias negativas de solo. Essa regra deverá ser levada em consideração na adaptação dos algoritmos a seguir.

Tendo em vista o algoritmo de erosão térmica desenvolvido anteriormente, é necessário reformular o cálculo de distribuição do solo entre as células vizinhas que estão propensas a receber material. A quantia movimentada para cada vizinho passa a ser multiplicada por um fator limitante `limiter`, que é calculado através da fórmula vista no Quadro 7, onde `soilVolume` representa a quantidade de solo na célula central e `sumDelta` contém a quantidade total de solo que seria movimentado desconsiderando-se a fórmula de distribuição. A fórmula de distribuição do solo passa então a ser `(factor * (maxDiff – talus) * (diff / sumDiff) * limiter)`.

Quadro 7 – Fórmula para obtenção do valor `limiter`

```
if (Movimentacao > VolumeSolo)
{
    limiter = VolumeSolo / Movimentacao;
}
else
{
    limiter = 1;
}
```

Fonte: Elaborado pelo autor

Quanto ao algoritmo de erosão hidráulica, é realizada uma simples verificação na conversão de solo em sedimento. Quando a quantidade a ser convertida for maior do que a quantidade de solo presente na célula, é convertida apenas esta quantidade, conforme destaque no Quadro 8. Nota-se que, como o algoritmo atual assume uma concentração uniforme de sedimento por toda a água, equivalente a quantidade máxima de solo convertida, que essa limitação causará um lento aumento da quantidade final de solo na paisagem.

Quadro 8 – Algoritmos de erosão hidráulica com limite de quantidade de solo em destaque

```
// Algoritmo de remoção do solo
private void Dissolve(float[,] soil, float[,] rock, float[,] water, float
solubility)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            float delta = solubility * water[x, y];
            soil[x, y] -= Math.Min(delta, soil[x, y] - rock[x, y]);
        }
    }
}
```

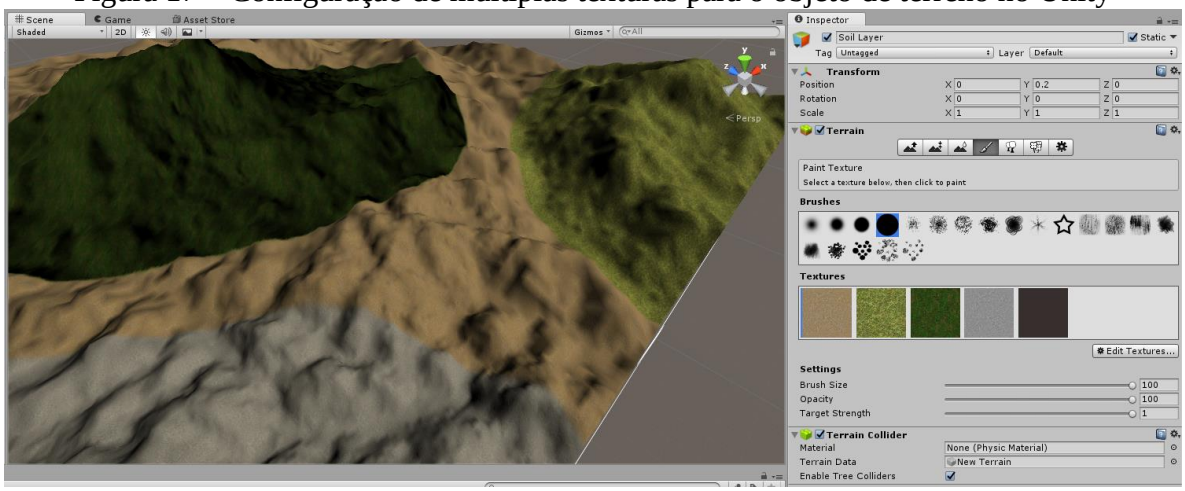
Fonte: Elaborado pelo autor.

3.3.1.5 INCLUSÃO DAS SUPERFÍCIES DE TERRENO

Os dados de superfície da paisagem são armazenados em uma matriz `surface` similar à utilizada para as alturas, mas ao invés de valores decimais são armazenados valores inteiros correspondentes a uma enumeração dos tipos de superfície suportados pela aplicação. No objeto de solo foram incluídas texturas adicionais para cada superfície, de forma que a opacidade da textura em um determinado ponto da paisagem possa ser acessada por meio das coordenadas x e y, mais o valor da superfície no ponto em questão (Figura 17). As texturas utilizadas foram obtidas a partir do pacote incluído no tutorial de Dvornik (DVORNIK, 2013).

Os tipos de superfície adotados na aplicação são: solo exposto, grama, floresta e concreto (pavimento). Como a informação de superfície não pode ser deduzida de um mapa de altura convencional, foi desenvolvida uma opção na aplicação para permitir que o usuário desenhe a superfície com o mouse.

Figura 17 – Configuração de múltiplas texturas para o objeto de terreno no Unity



Fonte: Elaborado pelo autor.

Em relação à erosão térmica, os dados de superfície são utilizados para modificar a inclinação máxima permitida em um determinado ponto da paisagem. Nota-se que a

inclinação é representada pela diferença entre alturas vizinhas (e não por valores de ângulo). Para cada ponto avaliado pelo algoritmo, a inclinação máxima é multiplicada por um valor modificador correspondente à superfície do mesmo. Esses valores (descritos no Quadro 9) visam representar a resistência à movimentação vista em solos com vegetação devido ao enraizamento, de forma arbitrária. Superfícies de concreto são ignoradas no algoritmo, pois se assume que a estrutura do pavimento impede qualquer queda de material.

Quadro 9 – Valores modificadores definidos para cada tipo de superfície

Tipo de Superfície	Inclinação Máxima	Acúmulo de Água	Drenagem de Água
Solo exposto	100%	100%	100%
Gramado	110%	80%	80%
Floresta	120%	40%	80%
Pavimento	200%	100%	0%

Fonte: Elaborado pelo autor.

Essa validação é apenas feita sobre o ponto central e não sobre os vizinhos, por questões de desempenho. Isso significa que a superfície na região mais alta definirá a quantidade de material que será transportado para as regiões mais baixas. Para simular a destruição de vegetação/estruturas em deslizamentos, as células que recebem material de células mais altas têm sua superfície alterada para solo exposto. Desta forma, o valor de inclinação máxima `talus` deixa de ser constante para toda a paisagem, sendo utilizado em seu lugar um valor `localTalus`, calculado através da fórmula (`talus * SurfaceModifier(surface[x, y])`), onde a função `SurfaceModifier` retorna o modificador de inclinação para a superfície respectiva. Verifica-se que os modificadores das superfícies de gramado, floresta e concreto resultam em maiores limites de inclinação, e consequentemente, encostas mais estáveis.

Em relação à erosão hidráulica, o tipo de superfície modifica a quantidade de água da chuva acumulada e a quantidade de água absorvida em cada célula. Como a movimentação de sedimentos depende destes eventos, o tipo de superfície também afeta a alteração do relevo, embora de uma forma indireta.

No acúmulo de água, apenas um percentual da quantidade total de água proveniente da chuva passa a ser considerado dependendo da superfície, conforme valores no Quadro 9. A redução no acumulo em regiões de gramado e floresta visa representar o amortecimento que a vegetação proporciona sobre a água da chuva, o que reduz o efeito da mesma sobre o solo. A drenagem da água é modificada da mesma maneira, neste caso os valores baseiam-se na permeabilidade de cada superfície, e por influenciar na quantidade de água drenada, também afetam indiretamente a quantidade de sedimento que cada célula irá a receber.

Em relação à simulação da água, células com quantidades de água superiores a 25% da altura máxima permitida pelo objeto do terreno têm sua superfície alterada para solo, para simular a destruição causada pela enxurrada.

3.3.1.6 INCLUSÃO DA UMIDADE RELATIVA DO SOLO

Dados referentes à umidade relativa do solo são armazenados em uma matriz `humidity` com o mesmo formato dos mapas de altura do solo e rocha. Os valores de ponto flutuante podem variar de 0 (solo absolutamente seco) a 1 (solo com concentração máxima de água).

A variação da umidade do solo está relacionada à simulação de chuva e dinâmica hídrica do algoritmo de erosão hídrica. Quando a água é infiltrada no solo, além da adição de sedimento, também é adicionada a quantidade de água que foi removida do mapa de águas ao mapa de umidade do solo, na proporção de 1:1, através de uma adaptação no algoritmo de drenagem de água destacada no Quadro 10.

Quadro 10 – Algoritmo final de drenagem de água com efeito na humidade do solo destacada

```
// Algoritmo de drenagem
private void DrainWater(float[,] soil, float[,] water, int[,] surface,
float[,] humidity, float solubility, float drain)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            float delta = waterVolume - (water[x, y] * drain);
            delta *= drainMods[surface[x, y]];
            water[x, y] -= delta;
            soil[x, y] += solubility * delta;
            humidity[x, y] += delta;
            // Limitar a umidade máxima a 1
            if (humidity[x, y] > 1) humidity [x, y] = 1;
        }
    }
}
```

Fonte: Elaborado pelo autor.

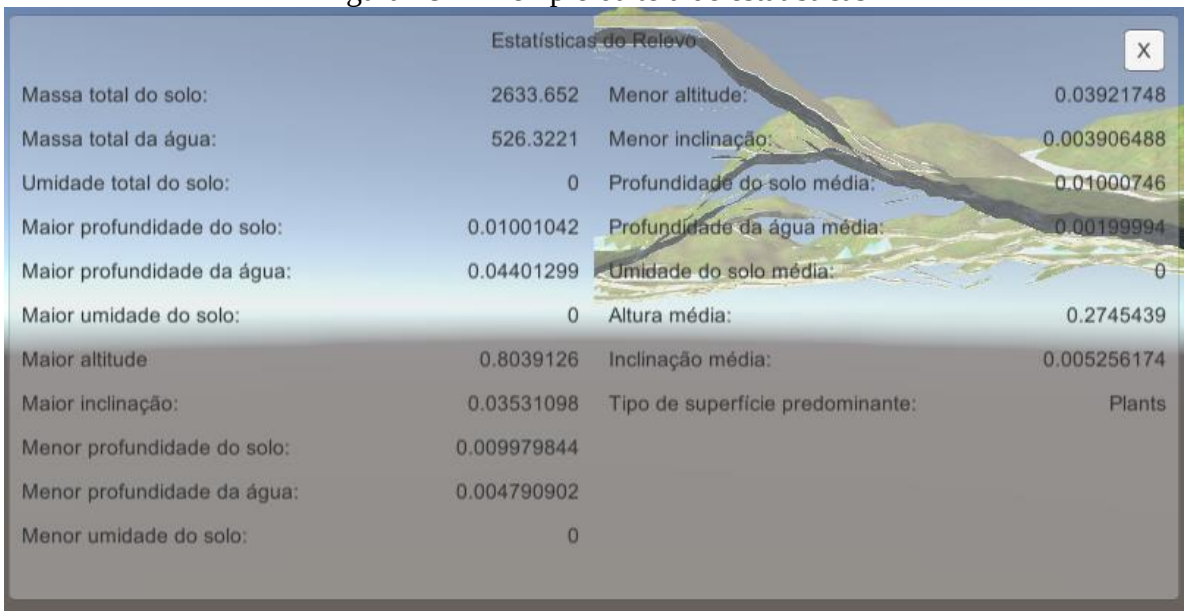
Esses dados são utilizados na erosão térmica, onde o valor de umidade de cada célula modifica a inclinação máxima suportada na mesma. Para células com umidade zero, a inclinação é a mesma, enquanto células com umidade máxima têm sua inclinação máxima reduzida pela metade. Valores intermediários resultam em uma modificação proporcional, conforme a nova fórmula de obtenção da inclinação máxima `((talus * SurfaceModifier(surface[x, y])) - ((humidity[x, y] * talus) / 2))`. As versões finais completas de cada algoritmo podem ser visualizadas no Apêndice A.

3.3.1.7 VISUALIZAÇÃO DE ESTATÍSTICAS

Para possibilitar uma análise numérica dos resultados da simulação, foi adicionada uma opção de visualização de estatísticas na tela principal, que pausa a simulação e exibe diversos dados numéricos relativos ao estado da paisagem no instante em que a opção foi selecionada, conforme Figura 18. A coleta de estatísticas é realizada através de um loop similar ao que ocorre nas transformações, no qual os dados são acumulados. As informações coletadas são:

a) volume total do solo;
b) volume total da água;
c) volume total de umidade no solo;
d) maior/menor profundidade de solo;
e) maior/menor profundidade da água;
f) maior/menor valor de umidade no solo;
g) maior/menor altitude;
h) maior/menor inclinação;
i) profundidade média do solo;
j) profundidade média da água;
k) altitude média;
l) inclinação média;
m) tipo de superfície mais presente.

Figura 18 – Exemplo da tela de estatísticas



Fonte: Elaborado pelo autor.

Em relação aos dados de inclinações, estes são coletados de cima para baixo, ou seja, para cada ponto analisado são coletados os valores de inclinação relativos aos vizinhos mais baixos que o ponto central. Isso é necessário para garantir que todas as inclinações sejam analisadas, independente de sua orientação, e ao mesmo tempo evitando que os valores se cancelem ao considerar as inclinações em ambos os sentidos (cima/baixo).

3.3.1.8 VISUALIZAÇÃO DE MAPAS DE CALOR

Embora a representação "natural" seja suficiente para que se tenha uma análise superficial da paisagem, informações como a profundidade do solo ou pequenas variações na umidade do solo não são tão facilmente visualizadas neste caso. Para retificar isso, foram criadas opções de visualização de mapas de calor.

O mapa de calor consiste em um quarto objeto de terreno na cena principal do programa com duas texturas mapeadas, verde e vermelho. Este objeto fica inativo até o momento em que a visualização de mapa de calor é ativada. Neste momento, o mapa de calor é ativado e as outras camadas (solo, rocha e água) são desativadas.

A camada de calor utiliza o mesmo mapa de altura da camada de solo, imitando o relevo da paisagem na visualização natural. No entanto, a sua textura é definida por uma matriz de calor, que por sua vez tem uma estrutura de dados similar aos mapas de altura utilizados nas outras camadas. Os valores deste mapa de calor são utilizados para definir a variação de textura sobre o terreno, onde valores iguais à zero ou um são representados por uma textura completamente verde ou vermelha, respectivamente, e valores intermediários são representados por uma combinação proporcional das duas cores.

Tendo em vista este comportamento, os dados do mapa de calor podem ser calculados ou até mesmo diretamente extraídos dos outros dados disponíveis. Foram disponibilizadas visualizações de mapas de calor referentes à profundidade do solo, profundidade da água, umidade do solo e inclinação. Vale notar que, embora seja feita a extração direta de todos esses dados (exceto inclinação) para o mapa de calor, optou-se por multiplicar os dados originais ao transferi-los para o mapa de calor. Tal adaptação tem a vantagem de ressaltar a visualização quando os valores envolvidos são muito baixos, mas faz com que a diferença entre valores extremos não possa ser visualizada, já que os valores do mapa de calor truncam em 1. Como situações com valores extremos são muito raras, optou-se por manter essa adaptação, multiplicando os valores por 10.

A visualização das inclinações (cujo algoritmo está descrito no Quadro 11) é um caso a parte. Assim como no cálculo de estatísticas, as inclinações são sempre avaliadas de cima

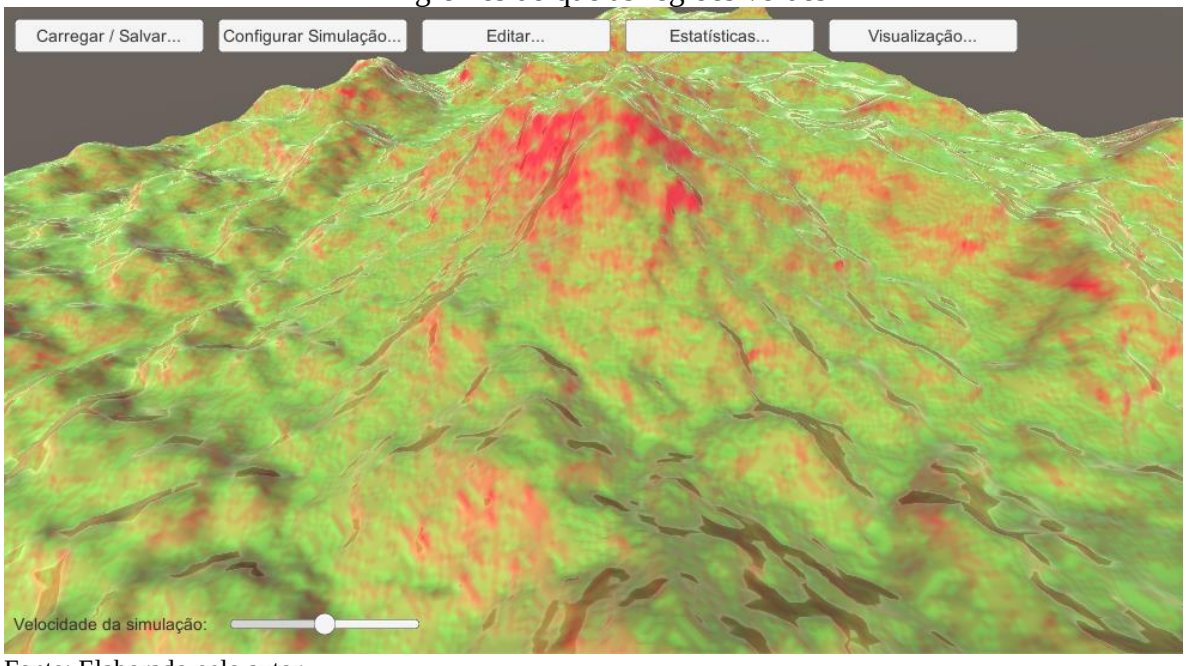
para baixo. Para obter o valor correspondente no mapa de calor, é extraído o valor da maior inclinação para baixo (diferença entre ponto central e vizinho), que é multiplicado por 50. Um exemplo de um mapa de calor resultante pode ser vista na Figura 19.

Quadro 11 – Algoritmo gerador do mapa de altura para inclinação do relevo

```
private void InclinationMap(float[,] soil, float[,] map)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            // A função HighestInclination retorna a maior inclinação
            // para baixo dentre os vizinhos do ponto (x, y)
            float value = HighestInclination(soil, x, y);

            // Multiplica-se o valor por 50 para destacar diferenças
            value *= 50;
            // Limitar o valor final a um valor entre 0 e 1
            if (value < 0) value = 0;
            else if (value > 1) value = 1;
            map[x, y] = value;
        }
    }
}
```

Fonte: Elaborado pelo autor.

Figura 19 – Visualização das inclinações. As regiões vermelhas apresentam inclinações mais íngremes do que as regiões verdes



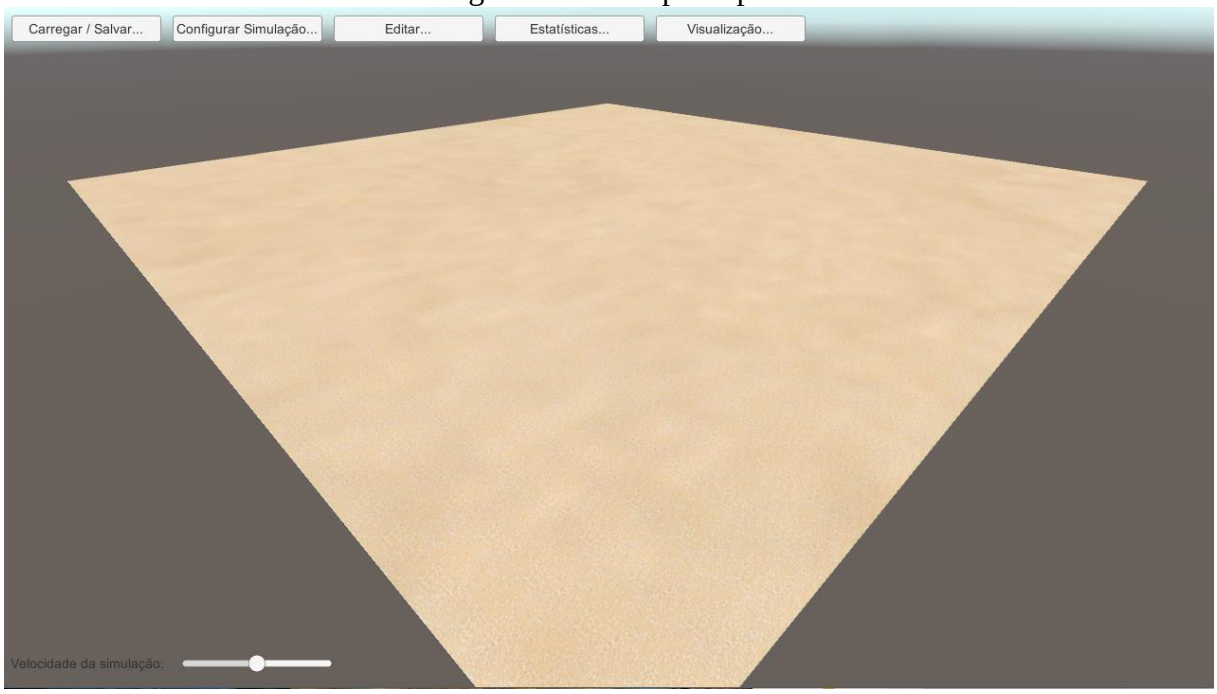
Fonte: Elaborado pelo autor.

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Ao se iniciar a aplicação, esta apresenta a visualização de um terreno sem características (Figura 20), assim como acessos aos menus do programa na parte superior. Estes menus são: carregar e salvar, configurar simulação, editar, estatísticas e visualização.

No canto inferior esquerdo encontra-se um controle deslizante para a velocidade da simulação (intervalo entre as execuções dos algoritmos).

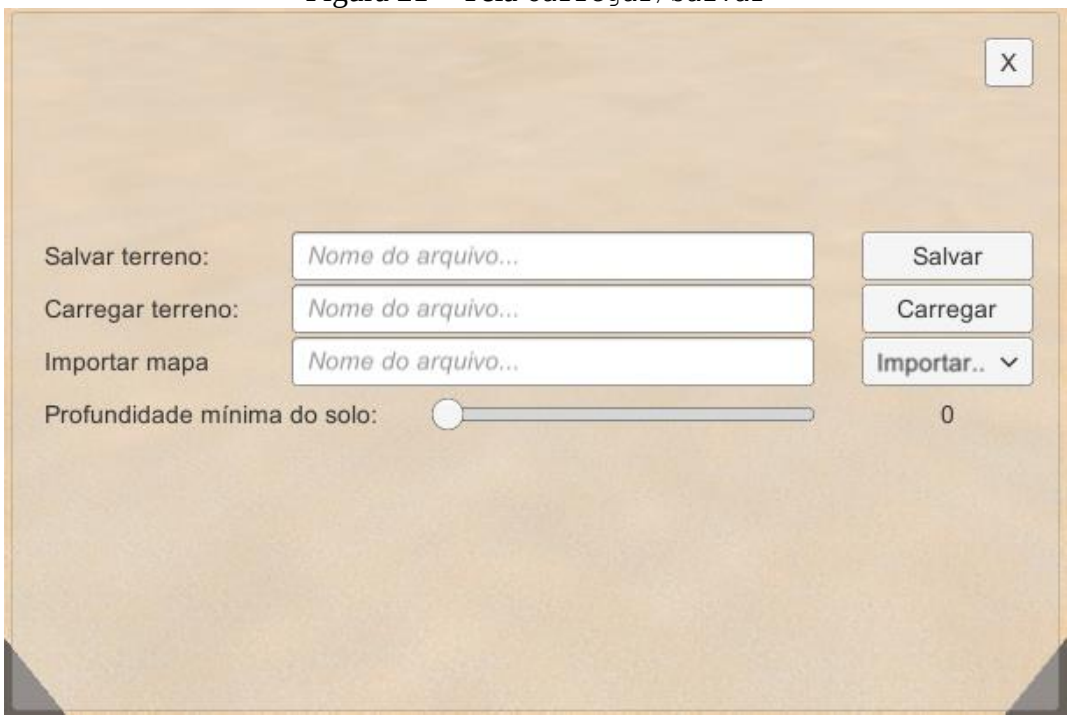
Figura 20 – Tela principal



Fonte: Elaborado pelo autor.

O menu `Carregar/Salvar` abre a tela relacionada (Figura 21), na qual há os campos para carregamento, salvamento e importação de mapas de altura. Estas ações são realizadas através do preenchimento do campo respectivo com o caminho completo ou relativo do arquivo desejado, seguido pelo clique no botão ao lado do campo. No caso da importação de mapa de altura, o botão oferece três opções: aplicar o mapa apenas à camada de solo ou rocha, ou aplicá-lo a ambas as camadas. O controle deslizante abaixo do campo de importação define qual a profundidade mínima do solo que deve ser permitida ao se importar o mapa de altura. Ao se importar um mapa de solo, a camada de rocha será podada onde a profundidade do solo for insuficiente, enquanto que na importação do mapa de rocha, o solo será levantado onde sua profundidade não for suficiente.

Figura 21 – Tela `Carregar/Salvar`

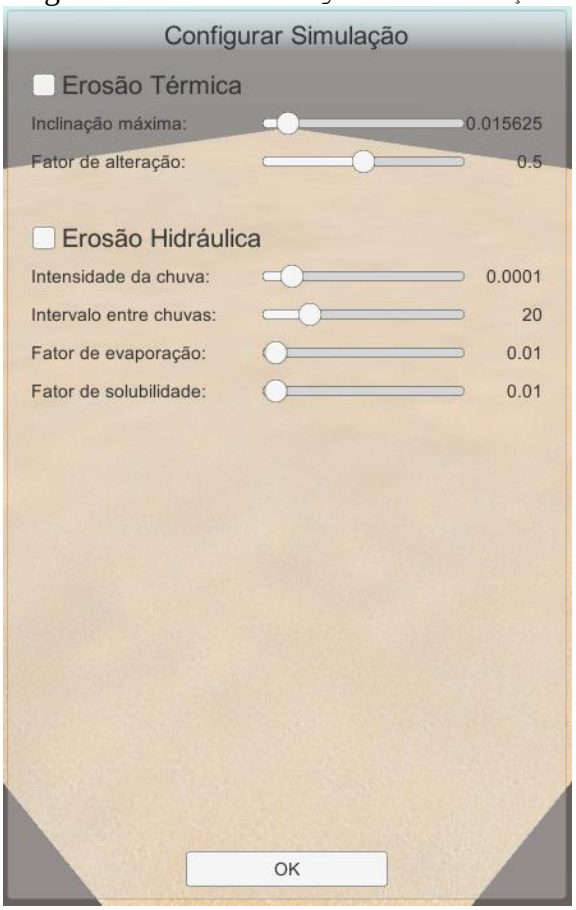


Fonte: Elaborado pelo autor.

A tela `Configurar Simulação` (Figura 22) contém as opções de ativar ou desativar as duas simulações desenvolvidas (erosão térmica e hidráulica) através de caixas de checagem. Cada simulação também possui seus parâmetros definidos por controles deslizantes.

Em relação aos parâmetros da erosão térmica, a inclinação máxima refere-se ao valor de `talus`, ou seja, a maior diferença entre alturas permitida pelo algoritmo, com valores maiores permitindo inclinações mais acentuadas. O fator de alteração é um multiplicador aplicado a todas as movimentações, sendo que valores maiores permitiram movimentações mais bruscas de solo.

Figura 22 – Tela `Configurar Simulação`



Fonte: Elaborado pelo autor.

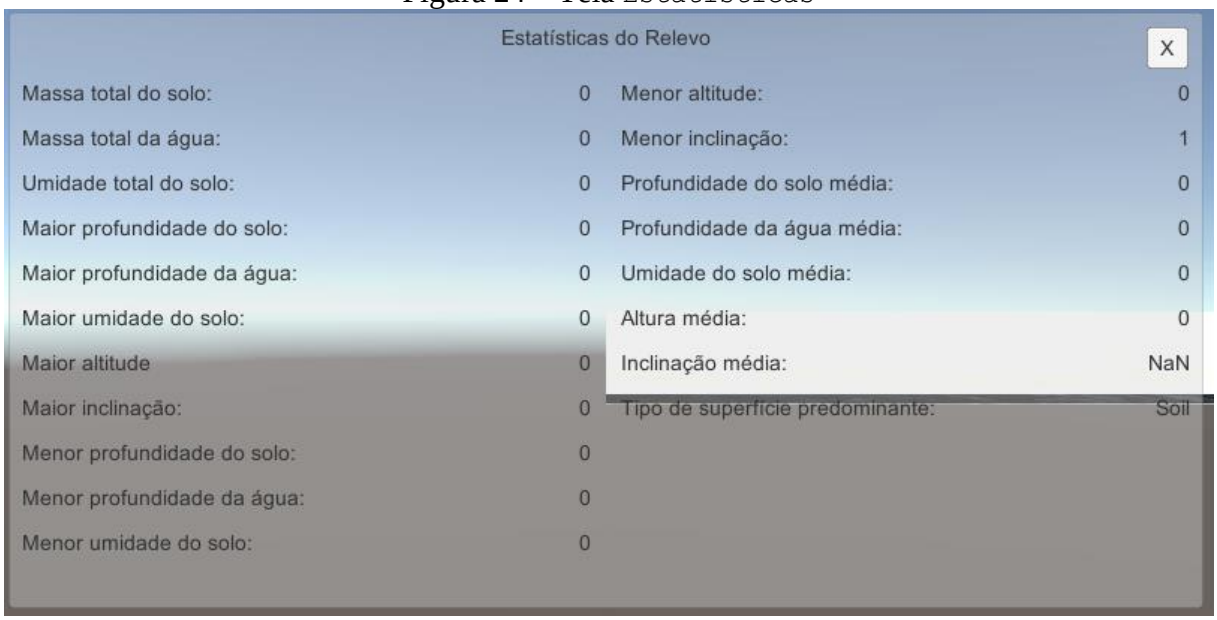
Na erosão hidráulica, os dois primeiros parâmetros referem-se ao algoritmo responsável por adicionar água da chuva à paisagem. A intensidade refere-se ao volume adicionado a cada iteração da chuva e o intervalo refere-se ao período entre chuvas, medido em loops da simulação. O fator de evaporação refere-se ao percentual de água que será evaporada/drenada à cada iteração e o fator de solubilidade refere-se à proporção de solo que será desgastada pela água durante a chuva.

Em relação às outras telas, a tela de edição (Figura 23) contém as opções utilizadas na pintura da superfície da paisagem (seleção de tipo de superfície e tamanho do "pincel" utilizado). Ao retornar à visualização, o usuário pode alterar a superfície clicando na área da paisagem desejada, caso tenha selecionado um pincel. A tela de estatísticas (Figura 24) exibe estatísticas sobre o estado atual da paisagem (a descrição destas estatísticas pode ser encontrada na subseção 3.3.1.6). A tela de visualização (Figura 25) permite selecionar um modo de exibição de mapa de calor (descritos na seção 3.3.1.7).

Figura 23 – Tela `Editar`

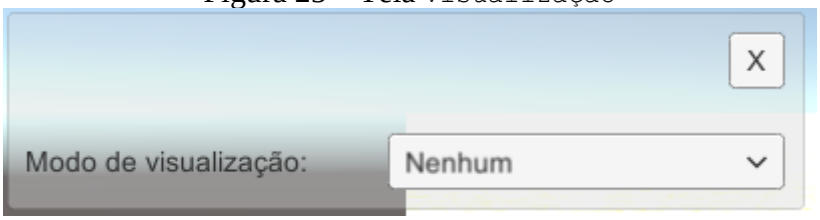
Fonte: Elaborado pelo autor.

Figura 24 – Tela `Estatísticas`



Fonte: Elaborado pelo autor.

Figura 25 – Tela `Visualização`



Fonte: Elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

A seguir serão descritos os processos de testes sobre a geração de estatísticas e a simulação, assim como os resultados e as conclusões iniciais que podem ser tiradas a partir destes.

3.4.1 GERAÇÃO DE ESTATÍSTICAS

A finalidade dos testes a seguir é verificar a integridade da geração do quadro de estatísticas. Para realizar estes testes, foram criadas duas imagens em escala de cinza para serem utilizadas como mapas de altura. Ambas as imagens possuem resolução de 129x129 e formato PNG.

A primeira imagem, denominada "*Split*" (Figura 26, esquerda), consiste em um painel dividido ao meio por uma linha de cor cinza 50%, com metade completamente branca e a outra metade completamente preta. O mapa resultante consiste em duas zonas planas de áreas idênticas, representando as alturas máxima e mínima permitidas pela aplicação. As duas zonas são divididas por uma inclinação de 0,5 (valor este representando a diferença entre alturas) que se estende por 3 células (inferior, uma intermediária, e topo). A segunda imagem, denominada "*Slope*" (Figura 25, direita), consiste em um gradiente em escala de cinza vertical. Nota-se que o gradiente utilizado apresenta pequenas imperfeições.

Figura 26 – Arquivos de imagem utilizados nos testes de geração de estatísticas

Fonte: Elaborado pelo autor.

Avaliando-se as estatísticas geradas pelo mapa "*Split*" (Tabela 1), é possível perceber uma pequena diferença entre os valores de maior e menor inclinação, a qual pode ser atribuída à imprecisão dos cálculos com valores do tipo `float`. Já os valores de altura e inclinação média em 0,5 estão dentro do esperado, tendo em vista a divisão exata do terreno entre alturas de 0 e 1, assim como a inclinação definida em 0,5. Vale ressaltar que, no cálculo da inclinação média, não são consideradas áreas planas.

Nos resultados do mapa "*Slope*" (Tabela 1), destaca-se a altura média, que permaneceu próxima de 0,5. Importante ressaltar que o gradiente, embora centralizado, não possui o alcance completo de 0 a 1. O alcance real pode ser definido através da maior e menor altitude (~0,15 a ~0,86).

Tabela 1 – Estatísticas geradas nos testes utilizando arquivos de imagem

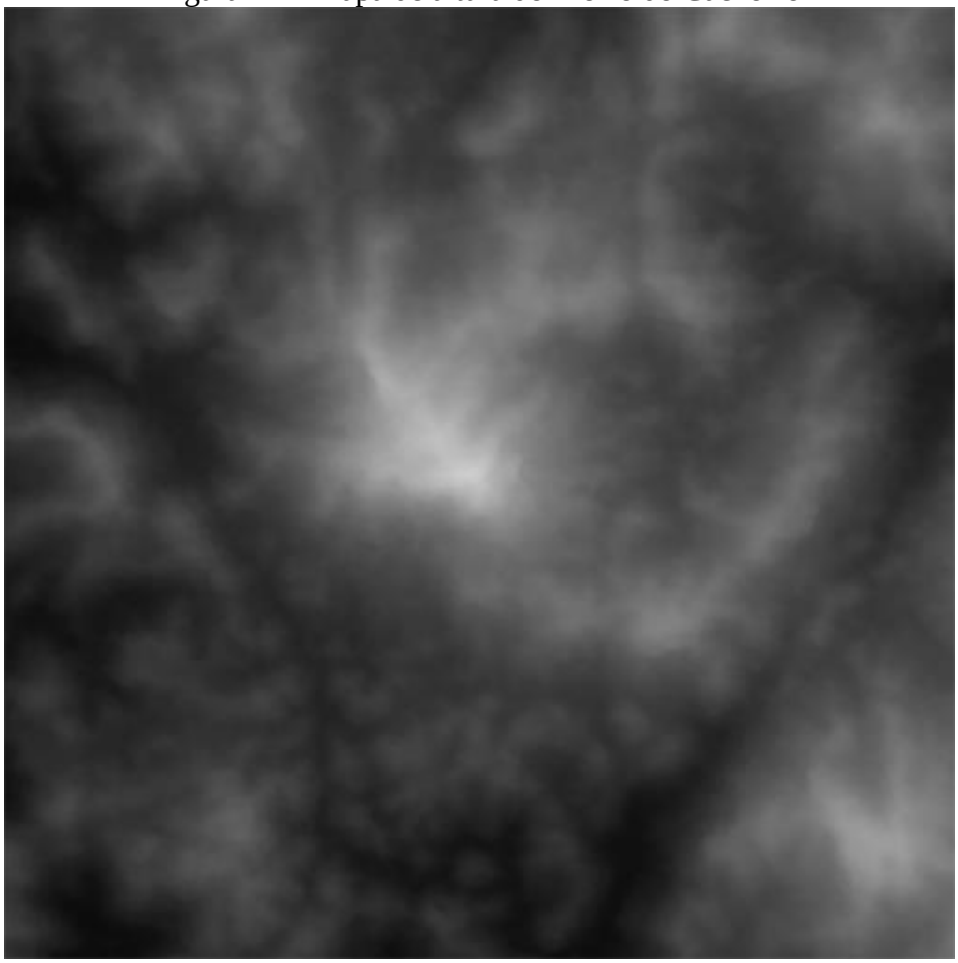
Estatística	*Split*	*Slope*
Massa total do solo	166,3978	166,3978
Maior profundidade	0,01	0,01000001
Menor profundidade	0,00999999	0,00999999
Profundidade média	0,009999269	0,009999269
Maior inclinação	0,5843138	0,01176473
Menor inclinação	0,4156862	0,003921568
Inclinação média	0,5	0,005269003
Altura média	0,500654	0,5109465

Fonte: Elaborado pelo autor.

3.4.2 EROSÃO TÉRMICA

Em seguida, foram efetuados testes sobre o algoritmo de erosão térmica. Para efetuar estes testes, foi utilizado um mapa de altura do Morro do Cachorro (Figura 27), situado ao norte do município de Blumenau. Este mapa de altura foi obtido utilizando a ferramenta online terrain.party. Antes de iniciar a simulação, foram geradas estatísticas sobre a paisagem no seu estado inicial para comparação (Tabela 2).

Figura 27 – Mapa de altura do Morro do Cachorro



Fonte: Elaborado pelo autor.

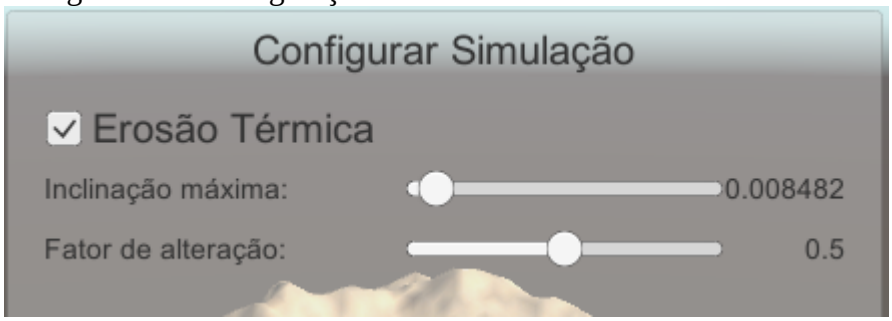
Tabela 2 – Estatísticas geradas nos testes do Morro do Cachorro

Estatística	Inicial	Erosão térmica	Erosão hidráulica
Massa total do solo	5267,897	5267,249	5113.614
Maior profundidade	0,02000001	0,05004317	0,020511866
Menor profundidade	0,01999998	5,364418e-07	0,000664562
Profundidade média	0,02001716	0,0200147	0,01943091
Maior inclinação	0,03529412	0,03535414	0,03532404
Menor inclinação	0,003921568	7,450581e-09	7,450581e-09
Inclinação média	0,005250997	0,00490757	0,00357193

Fonte: Elaborado pelo autor.

Para executar a simulação, foram selecionados os parâmetros vistos na Figura 28. Deve-se notar que caso a inclinação máxima não seja inferior a maior inclinação do terreno, não haverá movimentações. Permitiu-se então a execução da simulação até que as movimentações de solo tenham cessado em sua maioria.

Figura 28 – Configurações de erosão térmica utilizadas no teste

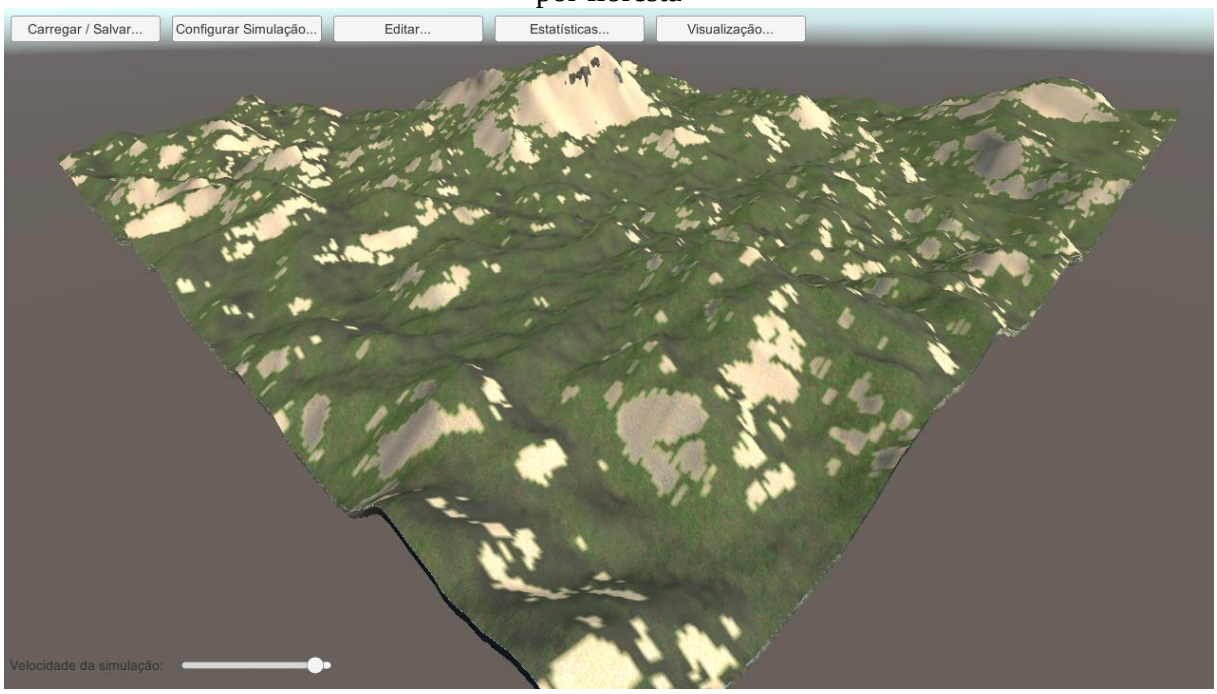


Fonte: Elaborado pelo autor.

Analisando-se as estatísticas resultantes (Tabela 2), podem-se perceber os efeitos da simulação sobre a paisagem. Tanto a massa total quanto a profundidade média do solo permaneceram relativamente constantes, enquanto os valores de profundidade máxima e mínima foram alterados, indicando que houve a movimentação de solo presente na paisagem. Em relação às inclinações, enquanto as máximas permaneceram similares, a inclinação mínima sofreu uma redução considerável. A inclinação média também foi reduzida, mas não alcançou o valor parametrizado de inclinação máxima, como seria o esperado. Isso ocorre porque há uma quantidade limitada de solo para ser movimentado na paisagem, sendo insuficiente para eliminar todas as inclinações superiores ao valor definido.

Adicionalmente, foi efetuado um teste de erosão térmica com a paisagem toda coberta por floresta. O comportamento da simulação de destruir a superfície onde ocorre movimentação resulta em uma paisagem onde os locais de deslizamento são facilmente visíveis, conforme a Figura 29.

Figura 29 – Resultado da execução do teste de erosão térmica sobre uma paisagem coberta por floresta

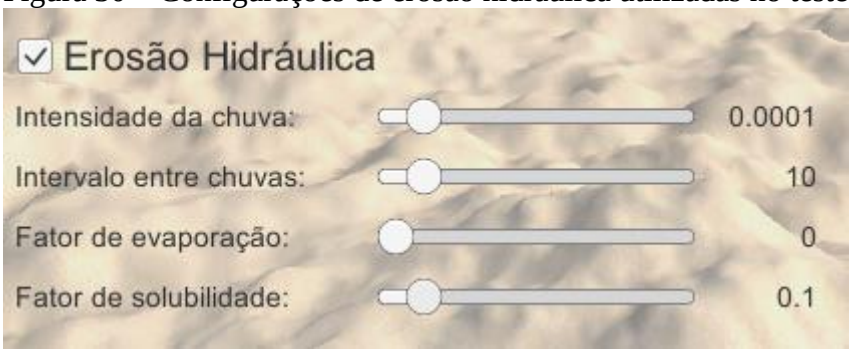


Fonte: Elaborado pelo autor.

3.4.3 CHUVA E EROSÃO HIDRÁULICA

Os testes do algoritmo de erosão hidráulica foram efetuados sobre o mesmo mapa inicial do Morro do Cachorro utilizado anteriormente. Inicialmente, foram utilizadas as configurações na Figura 30, e permitiu-se que a água acumulasse até formar pequenas reservas, conforme visto na figura 31.

Figura 30 – Configurações de erosão hidráulica utilizadas no teste

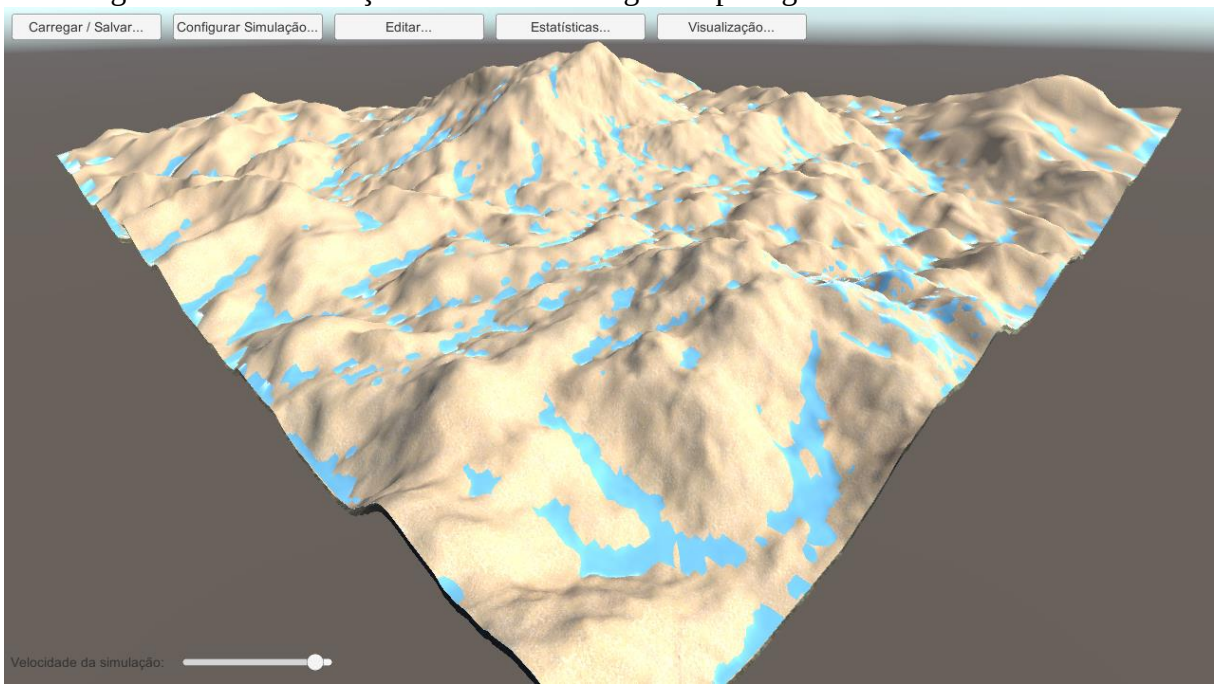


Fonte: Elaborado pelo autor.

Após o acúmulo da água, foi alterada a configuração `Fator de evaporação` para 0,1, permitindo assim que a água infiltrasse no solo e depositasse o sedimento nela dissolvido. Após a infiltração da maioria da água, foram extraídas as estatísticas da paisagem (Tabela 2). Percebe-se uma pequena redução na massa total e profundidade média do solo, devido à

parcela que se encontra dissolvida na água ainda presente na paisagem. Embora tenha ocorrido uma redução notável na menor profundidade, não houve o aumento proporcional da maior profundidade que se notou na erosão térmica. Isso se deve ao fato de a erosão hidráulica distribuir o material removido das regiões mais altas de forma mais abrangente nas regiões mais baixas, em contraste com o movimento mais direto da erosão térmica. Também nota-se que, embora os valores máximo e mínimo de inclinação sejam similares aos resultados da erosão térmica, a média no caso da erosão hidráulica é inferior.

Figura 31 – Visualização do acúmulo de água na paisagem do Morro do Cachorro



Fonte: Elaborado pelo autor.

4 CONCLUSÕES

A aplicação resultante do desenvolvimento visto neste trabalho permite a visualização, carga e salvamento de mapas de altura, alteração de sua superfície, execução de dois algoritmos de transformação baseados em processos de dinâmica de relevo reais e opções de visualização do estado do relevo a qualquer momento.

A simulação, embora ainda esteja longe de oferecer uma representação detalhada e precisa dos processos que ela representa, resulta em uma visualização condizente com a realidade. A atualização do relevo em tempo real possibilita uma visualização dos processos mais acessível do que simuladores tradicionais. No entanto, essa funcionalidade também causa um grande impacto no desempenho da aplicação e limita a complexidade das transformações executadas. A escolha da plataforma Unity praticamente eliminou a necessidade de se desenvolver um motor de renderização de terreno, mas ela não se encontra otimizada para uma aplicação que necessite de atualizações frequentes no relevo, tendo uma pequena pausa na visualização durante estas atualizações que resulta em execução lenta ao se selecionar maiores velocidades de simulação.

A visualização da paisagem virtual apresenta desempenho satisfatório enquanto a simulação estiver inativa. A câmera pode ser controlada com o mouse ou teclado, permitindo movimentos de translação, rotação em torno do ponto de foco e aproximação/afastamento, oferecendo uma possível visualização de qualquer ponto da paisagem. A representação da paisagem é simples, tendo em vista que a qualidade gráfica não foi um dos objetivos principais do projeto.

A captura de estatísticas oferece informações detalhadas que permitem ao usuário avaliar as alterações que ocorreram durante a simulação. No entanto, os valores exibidos não são refinados, o que pode dificultar o seu entendimento. Já os mapas de calor não oferecem dados com o mesmo nível de detalhamento, mas permitem uma visualização da informação localizada sobre a paisagem, sendo de mais fácil entendimento.

De uma forma geral, a aplicação comprova que existe viabilidade do uso de algoritmos de transformação de matriz como simulação de processos de relevo em tempo real, embora com seus limites. Este formato de simulação é de simples desenvolvimento e pode ter aplicações didáticas ou lúdicas, onde a aparência e abrangência da simulação tenham maior prioridade que sua precisão.

4.1 EXTENSÕES

Por utilizar a plataforma Unity, que não se encontra otimizada para o tipo de aplicação desenvolvida, esta apresenta problemas de desempenho durante a simulação. A transferência dos algoritmos neste trabalho desenvolvidos para uma plataforma mais adequada à renderização de terreno, ou até mesmo o desenvolvimento de uma plataforma própria, poderia resultar em uma simulação com melhor desempenho. Também existe a possibilidade de integração dos algoritmos com outras aplicações que façam uso de mapas de altura.

Outra possibilidade de extensão é o desenvolvimento de mais algoritmos de transformação, com base nas técnicas descritas neste trabalho. Outros processos de dinâmica de relevo, como os causados por vulcanismo e terremotos, poderiam ser traduzidos para um algoritmo de transformação e integrados à aplicação.

Finalmente, existem várias oportunidades para aprimoramentos na experiência de usuário e visualização, para se criar uma aplicação mais acessível. Uma importante funcionalidade para ser adicionada seria a existência de ferramentas de edição da paisagem dentro da própria aplicação, o que reduziria a dependência de ferramentas externas para a obtenção e alteração de mapas de altura.

REFERÊNCIAS

BOESCH, Florian. **WebGL GPU landscaping and erosion.** Basel, Suiça, 2011. Disponível em: <http://codeflow.org/entries/2011/nov/10/webgl-gpu-landscaping-and-erosion/>. Acesso em: 22 ago. 2016.

CAPUTO, Homero Pinto. **Mecânica dos Solos e Suas Aplicações:** Fundamentos. 6. ed. Rio de Janeiro: LTC Editora, 1988. 234 p.

CHAHI, Eric. **Eric Chahi on From Dust, Peter Molyneux and what's next.** [S.I], 2011. Disponível em: <http://www.eurogamer.net/articles/2011-11-03-eric-chahi-on-from-dustpeter-molyneux-and-whats-next-interview>. Acesso em 30 out. 2016. Entrevista concedida a Wesley Yin-Poole.

CHANDEL, Ruchika; GUPTA, Gaurav. Image Filtering Algorithms and Techniques: A Review. **International Journal of Advanced Research in Computer Science and Software Engineering.** Shoolini University, India, v. 3, n. 10, 0. 198-202, out. 2013.

DVORNIK, Kostiantyn; **Unity 4 Cool & Sweet Terrain Tutorial with FREE assets.** Kharkiv, Ucrânia. 2013. Disponível em: <http://kostiantyn-dvornik.blogspot.com.br/2013/01/unity-4-cool-sweet-terrain-tutorial.html>. Acesso em: 7 nov. 2016.

EMBRAPA. **Sistema Brasileiro de Classificação de Solos**. 2. ed. Rio de Janeiro: EMBRAPA-SPI, 2006. 306 p.

FRANCISCO, Wagner de Cerqueria e. **Agentes formadores do relevo**. [S.I]: Brasil Escola, 2017. Disponível em <http://brasilescola.uol.com.br/geografia/agentes-formadores-relevo.htm>. Acesso em: 21 mar. de 2017.

HACKSPACHER, Peter Christian. **Dinâmica do relevo:** quantificação de processos formadores. São Paulo: Editora Unesp, 2011. 146p.

HIGHLAND, Lynn M.; BOBROWSKY, Peter. **The landslide handbook:** A guide to understanding landslides. Reston, Virginia: U.S. Geological Survey Circular 1325, 2008. 129p.

OLSEN, Jacob. **Realtime Procedural Terrain Generation:** Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. 2004. 20 f.

POZZOBON, Maurício. **Análise da suscetibilidade a deslizamentos no município de Blumenau, SC:** Uma abordagem probabilística através da aplicação da técnica pesos de evidência. 2013. Tese (Doutorado em Ciências Florestais) - curso de Pós-Graduação em Engenharia Florestal, Setor de Ciências Agrárias, Universidade Federal do Paraná, Curitiba. 137 f.

RIBAS, Fernanda. Sete anos após a tragédia de 2008, Blumenau ainda convive com cenário de deslizamentos. **Jornal de Santa Catarina**, [S.I], 21 nov. 2015. Disponível em: < http://jornaldesantacatarina.clicrbs.com.br/sc/geral/noticia/2015/11/sete-anos-apos-a-tragedia-de-2008-blumenau-ainda-convive-com-cenario-de-deslizamentos-4911837.html>. Acesso em: 21 mar. de 2017.

SNOOK, Greg. **Real-Time 3D Terrain Engines Using C++ and DirectX 9**. Hingham, Massachusetts: Charles River Media, 2003. 362 p.

UBISOFT ENTERTAINMENT. **From Dust.** [S.I.], 2011. Disponível em: <https://www.ubisoft.com/pt-BR/game/from-dust/>. Acesso em: 25 ago. 2016.

UNITY TECHNOLOGIES. **Unity manual.** [S.I.], 2016. Disponível em: < http://docs.unity3d.com/Manual/index.html>. Acesso em: 11 set. 2016.

APÊNDICE A – Versões finais dos algoritmos de erosão térmica e hidráulica.

Nos quadros a seguir se encontram as versões finais dos algoritmos de erosão térmica e erosão hidráulica, com a inclusão dos parâmetros de camada de rocha, tipos de superfície e humidade do solo.

Quadro 12 – Algoritmo completo de erosão térmica na linguagem C#

```
float[] inclinationMods = { 1.0f, 1.1f, 1.2f, 2.0f };

public void DryErosion(float[,] soil, float[,] rock, int[,] surface,
float[,] humidity, float talus, float factor)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            float soilVolume = (soil[x, y] - rock[x, y]);
            float maxDiff = 0;
            float sumDiff = 0;
            float sumDelta = 0;
            float localTalus = talus * inclinationMods[surface[x, y]];
            localTalus -= (humidity[x, y] * talus) / 2;

            // Primeiro loop é necessário para totalizar as informações
            // necessárias para calcular as alterações
            VonNeumann(x, y, soil,
                (int relX, int relY)
                {
                    float diff = soil[x, y] - soil[relX, relY];
                    if (diff > maxDiff)
                        maxDiff = diff;
                    if (diff > localTalus)
                    {
                        sumDiff += diff;
                        sumDelta += factor * (diff - localTalus);
                    }
                }
            );
            // Se este valor for zero, o ponto está estabilizado
            if (sumDiff == 0)
                continue;
            // Limitar a quantidade de material ao volume do solo
            float limiter = 1.0f;
            if (sumDelta > soilVolume)
                limiter = soilVolume / sumDelta;
            float inclinationDifference = (maxDiff - talus);
            // Segundo loop é onde são feitas as alterações
            VonNeumann(x, y, soil,
                (int relX, int relY)
                {
                    float diff = soil[x, y] - soil[relX, relY];
                    if (diff > localTalus)
                    {
                        float move = factor * (maxDiff - localTalus) *
                        (diff / sumDiff) * limiter;
                        soil[relX, relY] += move;
                        soil[x, y] -= move;
                        // Locais que recebem queda de material têm sua
                        // superfície destruída
                        surface[relX, relY] = 0;
                    }
                }
            );
        }
    }
}
```

Fonte: Elaborado pelo autor.

Quadro 13 – Algoritmo completo de escoamento de água na linguagem C#

```
private void WaterFlow(float [,] soil, float[,] water)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            if (water[x, y] < 0) continue;
            float surface = soil[x, y] + water[x, y];
            float avg = 0;
            int count = 0;
            float sumDiff = 0;
            VonNeumann(x, y, soil,
                (int relX, int relY)
                {
                    float localSurface = soil[relX, relY] + water[relX,
                    relY];
                    float diff = surface – localSurface;
                    if (diff < 0) continue;
                    sumDiff += diff;
                    avg += localSurface;
                    count++;
                }
            );
            // Se este valor for zero, a água está estabilizada
            if (sumDiff == 0) continue;

            avg /= count;
            VonNeumann(x, y, soil,
                (int relX, int relY)
                {
                    float localSurface = soil[relX, relY] + water[relX,
                    relY];
                    float diff = surface – localSurface;
                    if (diff < 0) continue;
                    float delta = Math.Min(water[x, y], (surface - avg)) *
                    (diff / sumDiff);
                    water[relX, relY] += delta;
                    water[x, y] -= delta;
                }
            );
        }
    }
}
```

Fonte: Elaborado pelo autor.

Quadro 14 – Algoritmos completos de erosão hidráulica na linguagem C#

```
float[] pourMods = { 1.0f, 0.8f, 0.4f, 1.0f };
float[] drainMods = { 1.0f, 0.8f, 0.8f, 0.0f };

// Algoritmo de precipitação
private void PourWater(float[,] water, int[,] surface, float pour)
{
    for (int x = 0; x < water.GetLength(0); x++)
    {
        for (int y = 0; y < water.GetLength(1); y++)
        {
            water[x, y] += pour * pourMods[surface[x, y]];
        }
    }
}

// Algoritmo de remoção do solo
private void Dissolve(float[,] soil, float[,] rock, float[,] water, float
solubility)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            float delta = solubility * water[x, y];
            soil[x, y] -= Math.Min(delta, soil[x, y] – rock[x, y]);
        }
    }
}

// Algoritmo de drenagem
private void DrainWater(float[,] soil, float[,] water, int[,] surface,
float[,] humidity, float solubility, float drain)
{
    for (int x = 0; x < soil.GetLength(0); x++)
    {
        for (int y = 0; y < soil.GetLength(1); y++)
        {
            float delta = waterVolume - (water[x, y] * drain);
            delta *= drainMods[surface[x, y]];
            water[x, y] -= delta;
            soil [x, y] += solubility * delta;
            humidity[x, y] += delta;
            // Limitar a umidade máxima a 1
            if (humidity[x, y] > 1) humidity [x, y] = 1;
        }
    }
}
```

Fonte: Elaborado pelo autor.