UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SIMULAÇÃO DE DINÂMICA DE RELEVO

GUILHERME DIEGOLI NETO

BLUMENAU
2017

GUILHERME DIEGOLI NETO

SIMULAÇÃO DE DINÂMICA DE RELEVO

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, @@Titulação@@ - Orientador

BLUMENAU
2017

SIMULAÇÃO DE DINÂMICA DE RELEVO

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: @@Preencher membros da banca@@

Presidente:	Prof(a). Nome do(a) Professor(a), Titulação – Orientador, FURB
Membro:	Prof(a). Nome do(a) Professor(a), Titulação – FURB
Membro:	Prof(a). Nome do(a) Professor(a), Titulação – FURB

Blumenau, dia de mês de ano @@data apresentação@@

@@Geralmente um texto pouco extenso, onde o autor homenageia ou dedica o trabalho a alguém. Colocar a partir do meio da página.@@

AGRADECIMENTOS

@@Colocar menções a quem tenha contribuído, de alguma forma, para a realização do trabalho.@@

@@Epígrafe: frase que o estudante considera significativa para sua vida ou para o contexto do trabalho. Colocar a partir do meio da página.@@

@@Autor da Epígrafe@@

RESUMO

@@Resumo@@

@@Palavras chave@@

ABSTRACT

@@Abstract@@

@@Key-words@@

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

@@Listar abreviaturas@@

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.

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, 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). Embora a maioria dos métodos de detecção ainda necessite de ação humana, recentemente tem se visto o uso da computação para efetuar simulações de volume da massa de deslizamento e estabilidade de encostas (HIGHLAND; BOBROWSKY, 2008, p. 61).

Tendo em vista os métodos comentados, este trabalho se propõe a desenvolver 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, embora de uma forma simplificada. 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 estudar 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 (UNITY TECHNOLOGIES, 2016);
b) estudar, elaborar e aplicar algoritmos 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) 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 à que foi acima proposta. 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 desenvolvida, incluindo especificações, técnicas, ferramentas e os diversos algoritmos desenvolvidos durante o processo.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão analisados dois programas com funcionalidades relacionadas à aplicação proposta, assim como serão apresentados e discutidos os conhecimentos necessários para o desenvolvimento da aplicação.

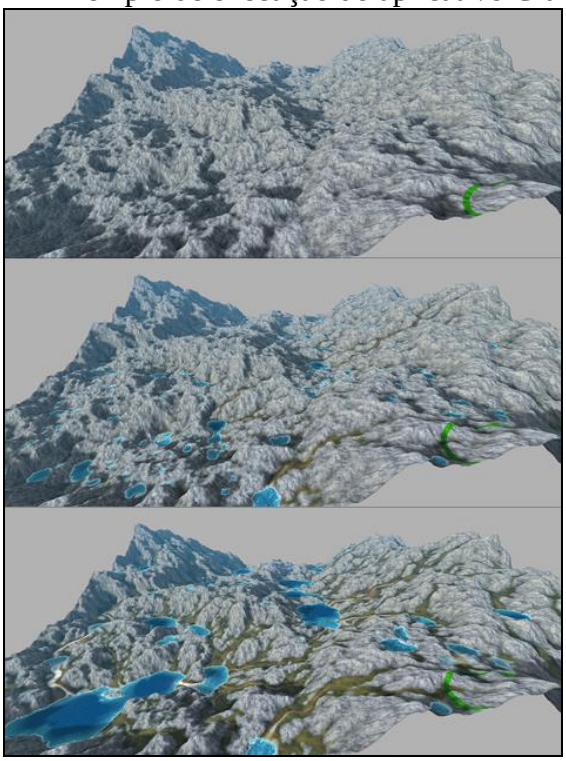
2.1 CRAFTSCAPE

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 1), 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 1 - 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, é notável por sua complexa simulação de dinâmica de relevo em tempo real, tratando processos como erosão hidráulica, deslizamento de areia e formação de dunas, crescimento de cobertura vegetal (Figura 2) e até ciclos de marés e vulcanismo (Figura 3) (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 lava. 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 se encontram em poucos locais específicos e fixos, como uma nascente de água ou um vulcão. A maioria dos efeitos acaba sendo causado pelas ações do jogador (UBISOFT, 2011).

Figura 2 - Exemplo de paisagem afetada pela erosão no jogo From Dust

Fonte: Ubisoft (2011).

Figura 3 – Conceito de vulcanismo do jogo From Dust

Fonte: Ubisoft (2011).

2.3 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 os 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 (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 sobre no seu 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, passando 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 em 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 se encontra exposto ao ambiente, se espera 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)

2.4 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 4), 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).

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.

Figura 4 – Exemplo de terreno renderizado na plataforma Unity

Fonte: Unity Technologies (2016)

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

2.5 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 cinza 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. É importante ressaltar que, embora o Unity aceite mapas de resoluções variadas, a matriz deve ter sempre um formato quadrado. (Entretanto, o Unity permite que o objeto de terreno seja distorcido para ter uma visualização retangular). (UNITY TECHNOLOGIES, 2016)

3 DESENVOLVIMENTO

Nesta seção está detalhado o processo de desenvolvimento da aplicação proposta.

3.1 REQUISITOS

O software proposto neste trabalho deve:

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

3.2 ESPECIFICAÇÃO

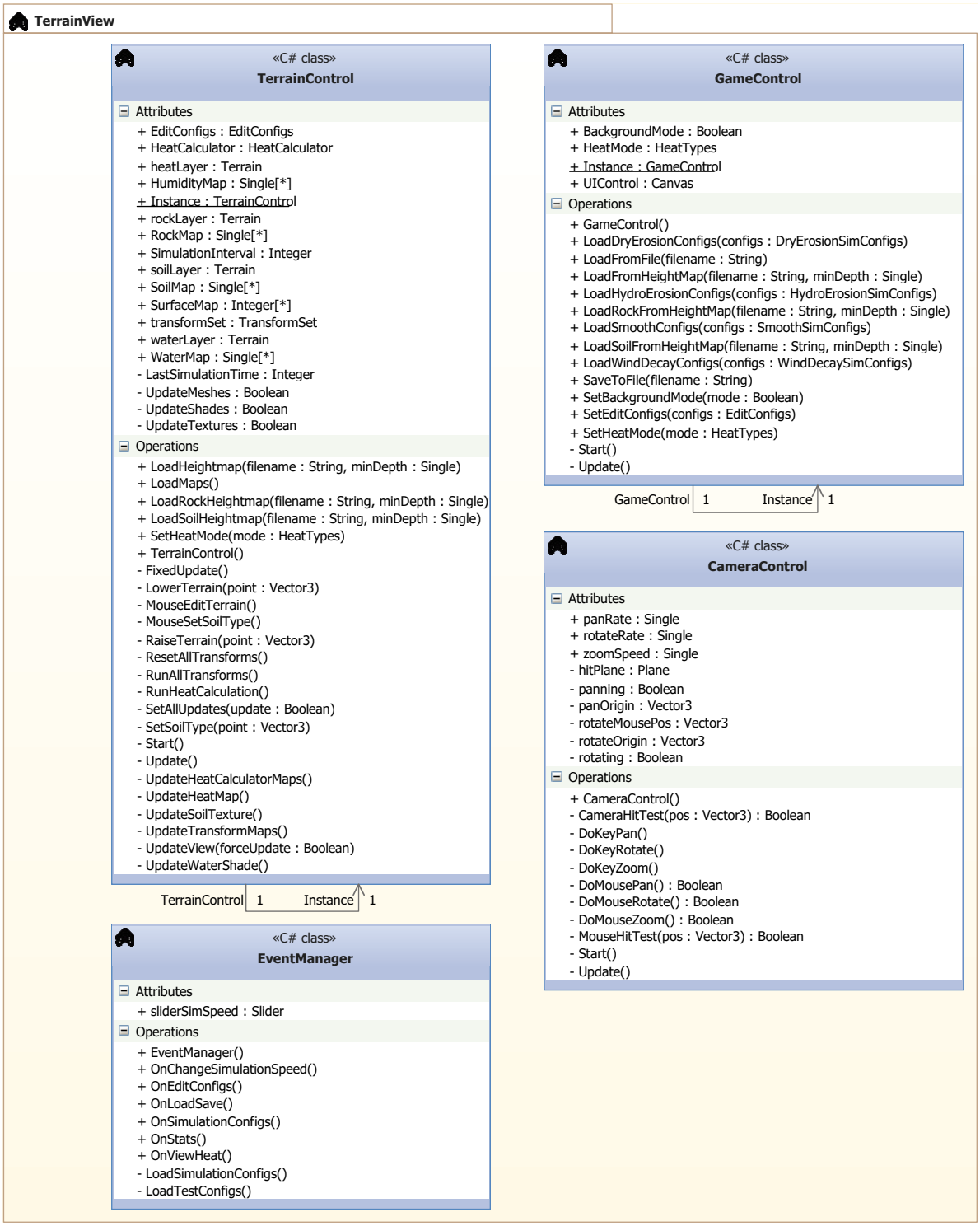
Nesta seção se encontram os diagramas de classe do programa final.

3.2.1 DIAGRAMA DE CLASSES

As Figuras 5 a 9 contém o diagrama das classes do programa, divididas por *namespace*. Para se construir os diagramas, foi utilizada a ferramenta Visual Studio 2013.

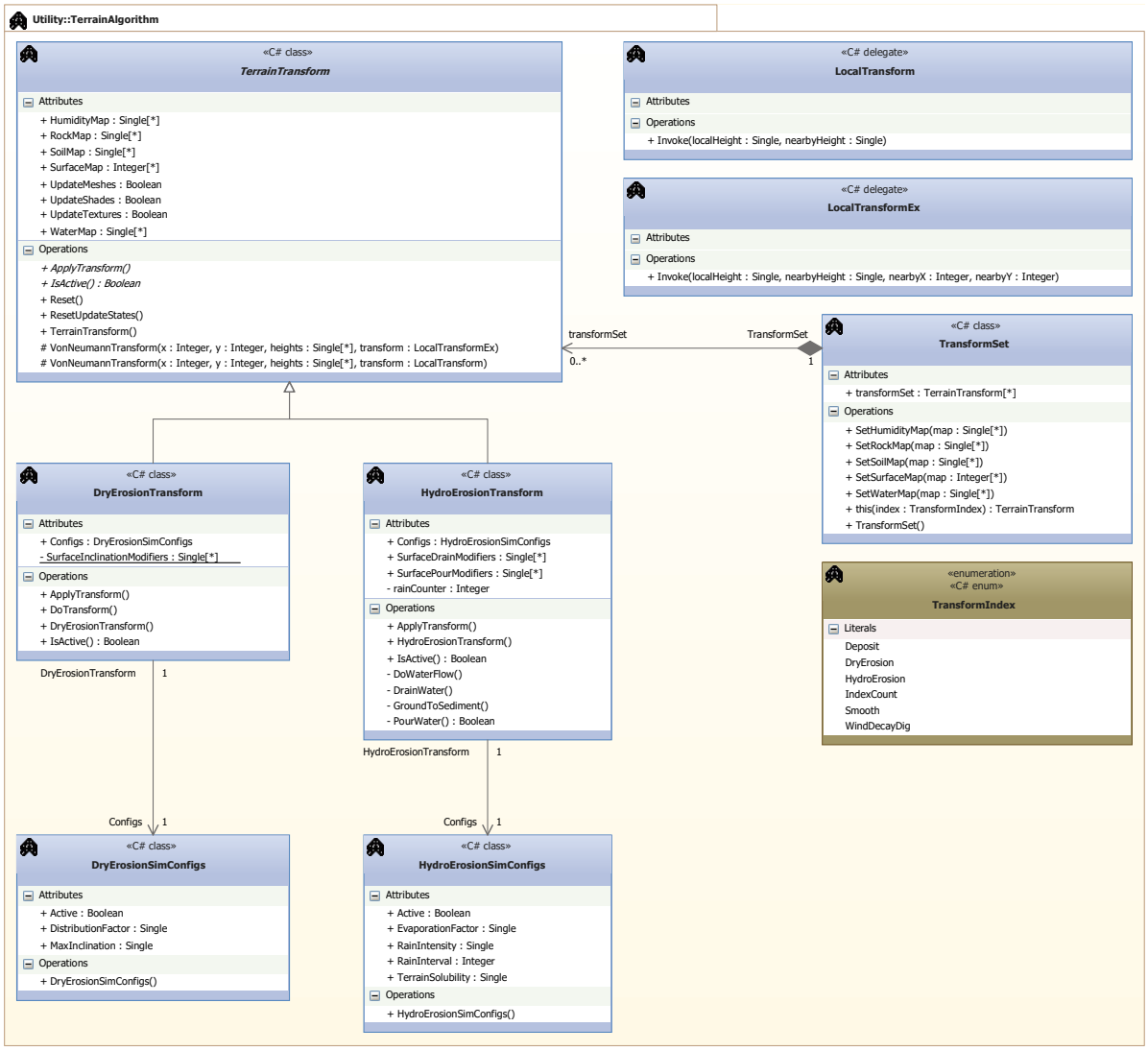
Cada *namespace* representa uma tela do programa, com exceção do *namespace* `Utility`, que contém algoritmos e informações de uso geral.

Figura 5 – Diagrama de classes do *namespace* `TerrainView`



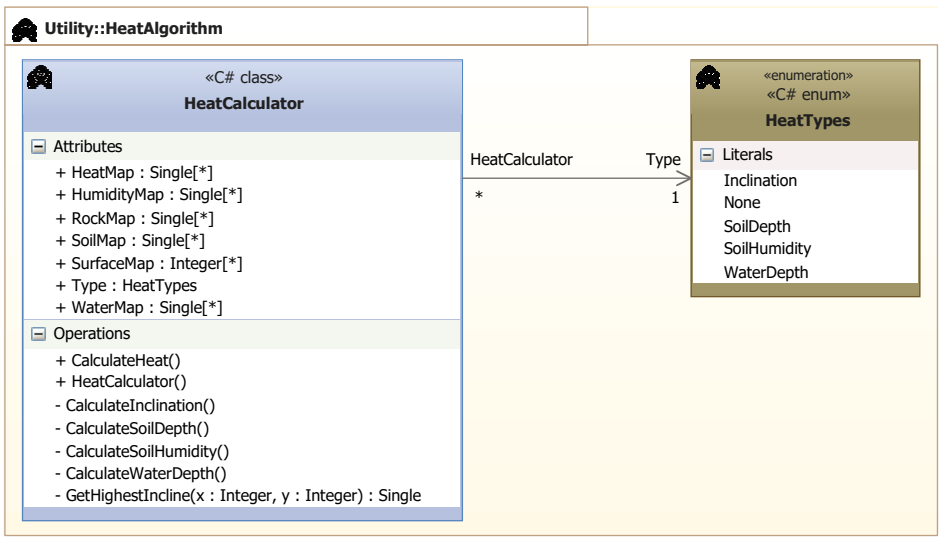
Fonte: Elaborado pelo autor.

Figura 6 – Diagrama de classes do *namespace* `Utility::TerrainAlgorithm`



Fonte: Elaborado pelo autor.

Figura 7 – Diagrama de classes do *namespace* `Utility::HeatAlgorithm`

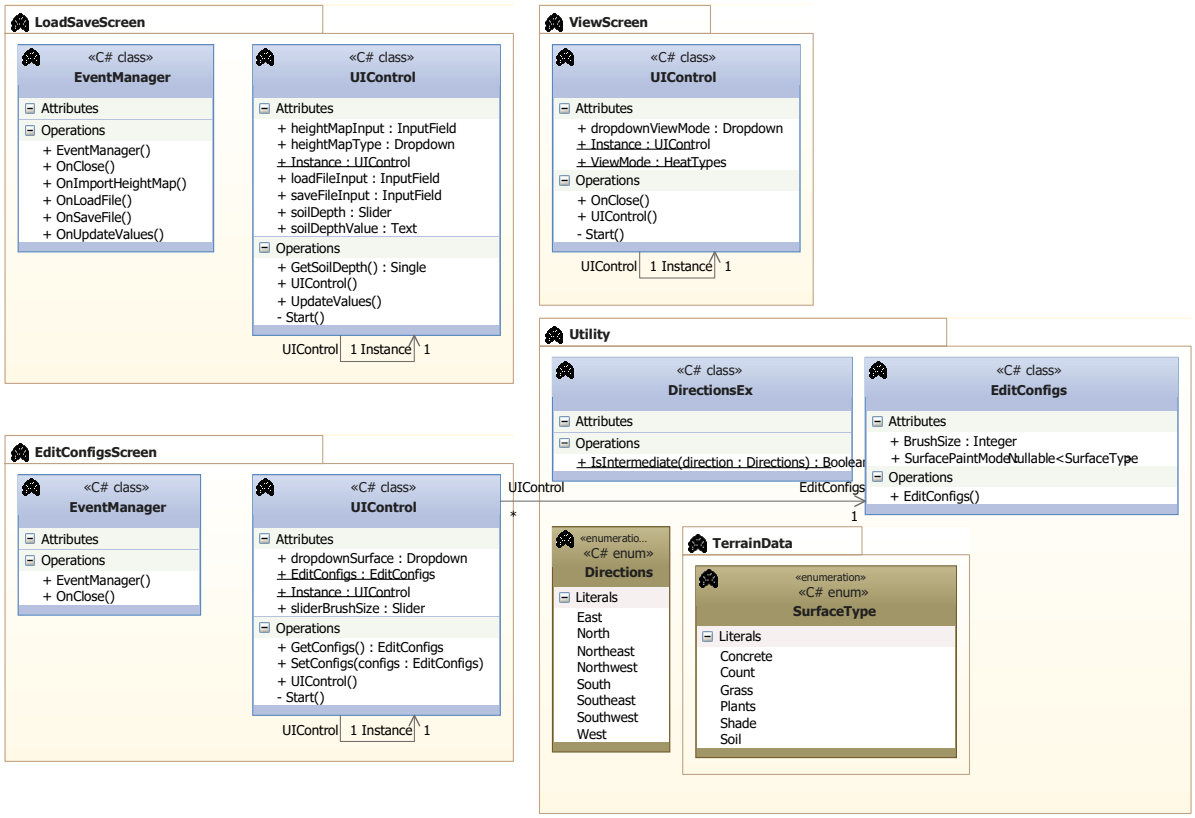


Fonte: Elaborado pelo autor.

Figura 8 – Diagrama de classes do *namespace* `SimulationConfigsScreen`

Fonte: Elaborado pelo autor.

Figura 9 – Diagrama de classes dos *namespaces* `LoadSaveScreen`, `ViewScreen`, `EditConfigsScreen` e `Utility`



Fonte: Elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nesta seção está descrito o processo de implementação do programa proposto.

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.

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Nesta seção se descreve as fórmulas que foram utilizadas como base no desenvolvimento da aplicação, bem como as adaptações que foram realizadas com base nas pesquisas realizadas anteriormente.

3.3.2.1 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 transformem 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 (Figura 10). (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 centralizada sobre a célula 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.

A execução do algoritmo elimina pequenas características do relevo e suaviza inclinações, um resultado que pode ser comparado ao assentamento de areia solta (Figura 11). 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.

Para avaliar as possibilidades que esse formato de algoritmo providencia, foi desenvolvida uma alternativa ao primeiro algoritmo, que pode ser visualizado no Quadro 2. Ao invés de considerar a matriz de vizinhos completa na aplicação da média, são consideradas apenas as células que se encontram na mesma latitude ou abaixo dela. 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 do vento varrendo uma paisagem arenosa a partir de uma direção fixa. Aplicando-se os dois algoritmos simultaneamente, verifica-se a formação de estruturas similares a dunas (Figura 12).

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

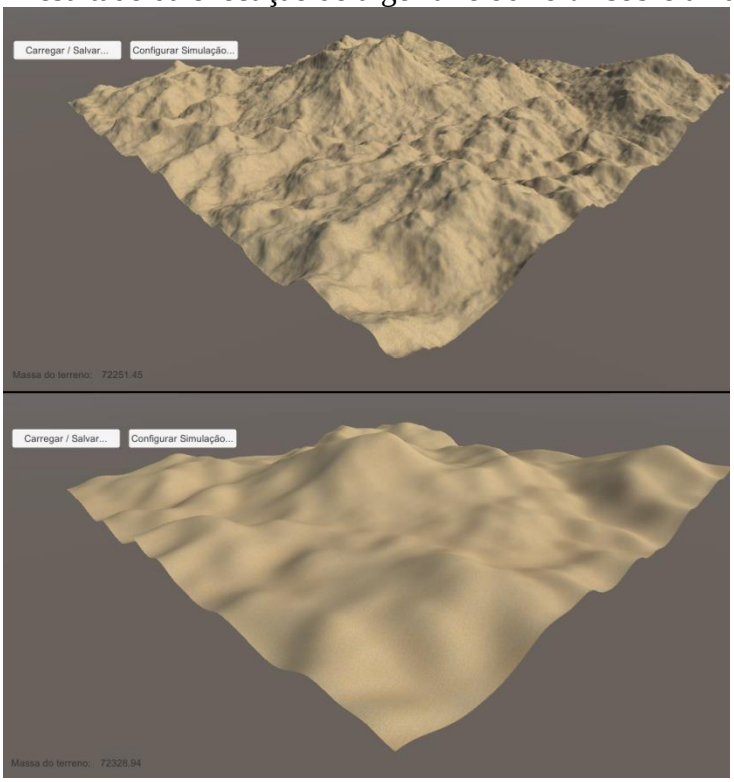
```
private void BoxBlur(float[,] matrix)
{
    int maxX = matrix.GetLength(0);
    int maxY = matrix.GetLength(1);

    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

Figura 10 - Exemplo de utilização do filtro *Box Blur* na ferramenta StylePix

Fonte: Hornil StylePix User Manual

Figura 11 - Resultado da execução do algoritmo *box blur* sobre uma paisagem



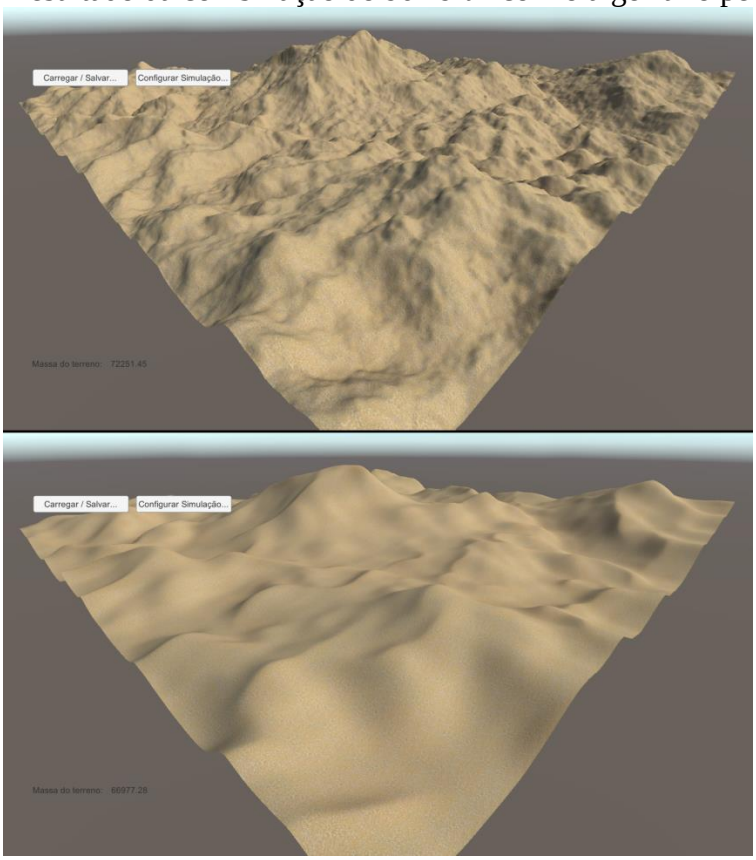
Fonte: Elaborado pelo autor

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



Fonte: Elaborado pelo autor

3.3.2.2 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 alto para vizinhos mais baixos caso a diferença entre os dois ultrapasse um limite pré-definido. 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.

O algoritmo final está representado no Quadro 3, onde o parâmetro `talus` é a inclinação máxima permitida (representada pela diferença entre duas alturas adjacentes) e `factor` é um fator de movimentação de material. Este fator reduz a quantidade movimentada

por iteração do algoritmo, deixando a transformação mais devagar, mas também melhorando a qualidade da simulação. Segundo Olsen (2004, p. 6), o valor ideal para `factor` é 0,5.

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`.

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`. Finalmente, a água é escoada através de um algoritmo similar ao utilizada na erosão térmica, mas que visa nivelar os valores de s através da movimentação de água para os vizinhos inferiores. Este algoritmo está representado no Quadro 4.

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 descritos no Quadro 5.

A aplicação destes algoritmos mostrou resultados satisfatórios (Figuras 13 e 14), 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 3 – Algoritmo de erosão térmica na linguagem C#

```
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
            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;

                    float diff = matrix[x, y] – matrix[absX, absY];
                    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
            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;
                    float diff = matrix[x, y] - matrix[absX, absY];
                    if (diff > talus)
                    {
                        float move = factor * (maxDiff - talus) * (diff /
sumDiff);
                        matrix[absX, absY] += move;
                        matrix[x, y] -= move;
                    }
                }
            }
        }
    }
}
```

Fonte: Elaborado pelo autor.

Quadro 4 – Algoritmo de escoamento de água na linguagem C#

```
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;
            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;
                    float localSurface = matrix[absX, absY] + water[absX,
absY];
                    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;
            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;
                    float localSurface = matrix[absX, absY] + water[absX,
absY];
                    float diff = surface - localSurface;
                    if (diff < 0) continue;
                    float delta = Math.Min(water[x, y], (surface - avg)) *
(diff / sumDiff);
                    water[absX, absY] += delta;
                    water[x, y] -= delta;
                }
            }
        }
    }
}
```

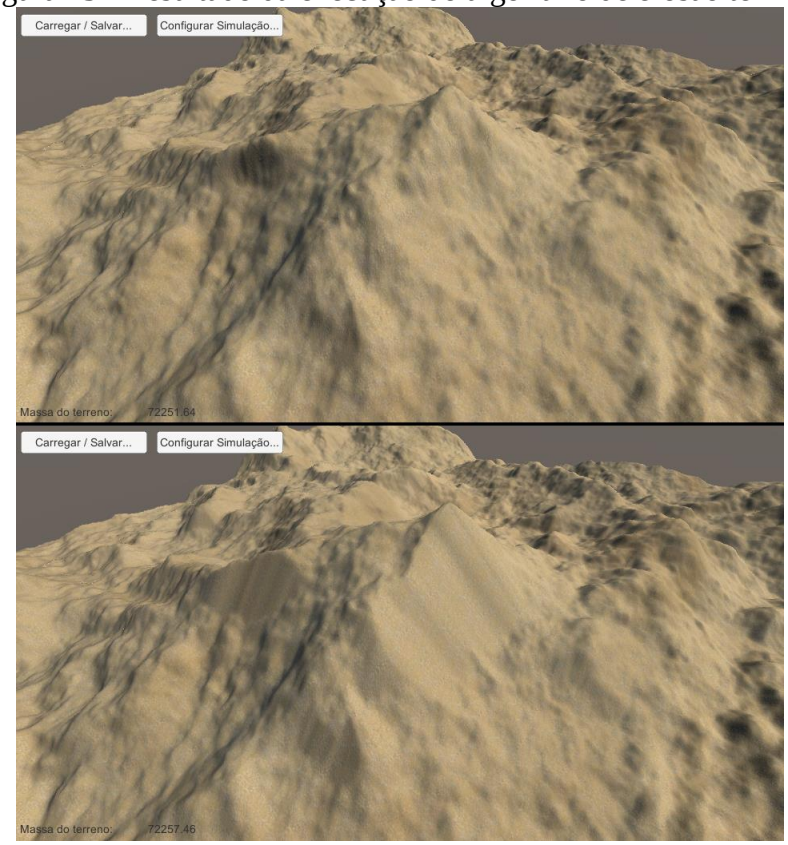
Fonte: Elaborado pelo autor.

Quadro 5 – 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;
        }
    }
}
```

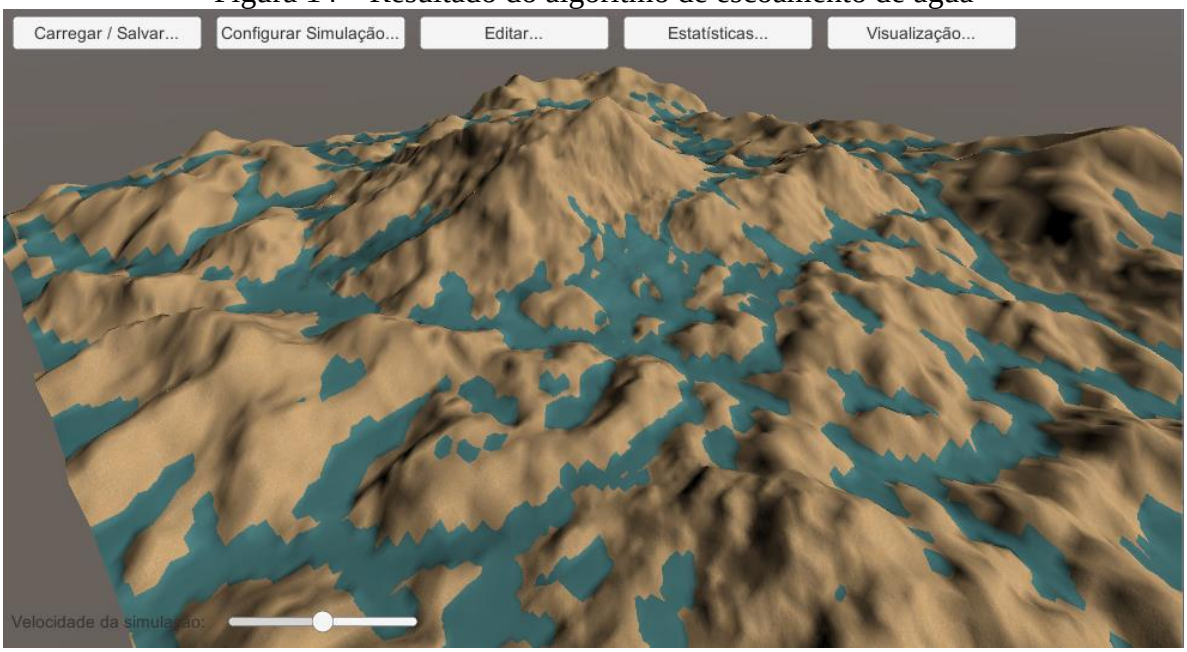
Fonte: Elaborado pelo autor.

Figura 13 - Resultado da execução do algoritmo de erosão térmica



Fonte: Elaborado pelo autor.

Figura 14 – Resultado do algoritmo de escoamento de água



Fonte: Elaborado pelo autor.

3.3.2.3 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`, cujo uso pode ser visto no Quadro 6.

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 (Quadro 7). 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 6 – Adaptação do algoritmo de erosão térmica considerando a camada de rocha

```
public void DryErosion(float[,] soil, float[,] rock, float talus, float
factor)
{
    int maxX = soil.GetLength(0);
    int maxY = soil.GetLength(1);
    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y < maxY; y++)
        {
            float soilVolume = (soil[x, y] - rock[x, y]);

            float maxDiff = 0;
            float sumDiff = 0;
            float sumDelta = 0;
            for (int relX = -1; relX <= 1; relX++)
            {
                // [Código ocultado por motivos de redundância]
                for (int relY = -1; relY <= 1; relY++)
                {
                    // [Código ocultado por motivos de redundância]
                    float diff = soil[x, y] - soil[absX, absY];
                    if (diff > maxDiff)
                        maxDiff = diff;
                    if (diff > talus)
                    {
                        sumDiff += diff;
                        sumDelta += factor * (diff - talus);
                    }
                }
            }

            // [Código ocultado por motivos de redundância]

            // Limitar a quantidade de material movimentada se não houver
            // solo o bastante
            float limiter = 1.0f;
            if (sumDelta > soilVolume)
                limiter = soilVolume / sumDelta;

            for (int relX = -1; relX <= 1; relX++)
            {
                // [Código ocultado por motivos de redundância]
                for (int relY = -1; relY <= 1; relY++)
                {
                    // [Código ocultado por motivos de redundância]
                    float diff = soil[x, y] - soil[absX, absY];
                    if (diff > talus)
                    {
                        float move = factor * (maxDiff - talus) * (diff /
sumDiff) * limiter;
                        matrix[absX, absY] += move;
                        matrix[x, y] -= move;
                    }
                }
            }
        }
    }
}
```

Fonte: Elaborado pelo autor.

Quadro 7 – Adaptação dos algoritmos de erosão hidráulica considerando a camada de rocha

```
// 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, 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);
            water[x, y] -= delta;
            // Está assumindo que a concentração de sedimento na água
            // equivale ao percentual de solubilidade, no entanto podem
            // haver situações onde a quantidade de solo retirada é
            // inferior à este percentual
            soil[x, y] += solubility * delta;
        }
    }
}
```

Fonte: Elaborado pelo autor.

3.3.2.4 INCLUSÃO DAS SUPERFÍCIES DE TERRENO

Os dados de superfície da paisagem são armazenados em uma matriz 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 15).

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.

Em relação à erosão térmica, os dados de superfície são utilizados para modificar a inclinação máxima permitida em um determinado ponto da paisagem. Notar 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 8) visam representar a resistência à movimentação vista em solos com vegetação, devido ao enraizamento. Superfícies de concreto são ignoradas no algoritmo, pois se assume que a estrutura do pavimento impede qualquer queda de material.

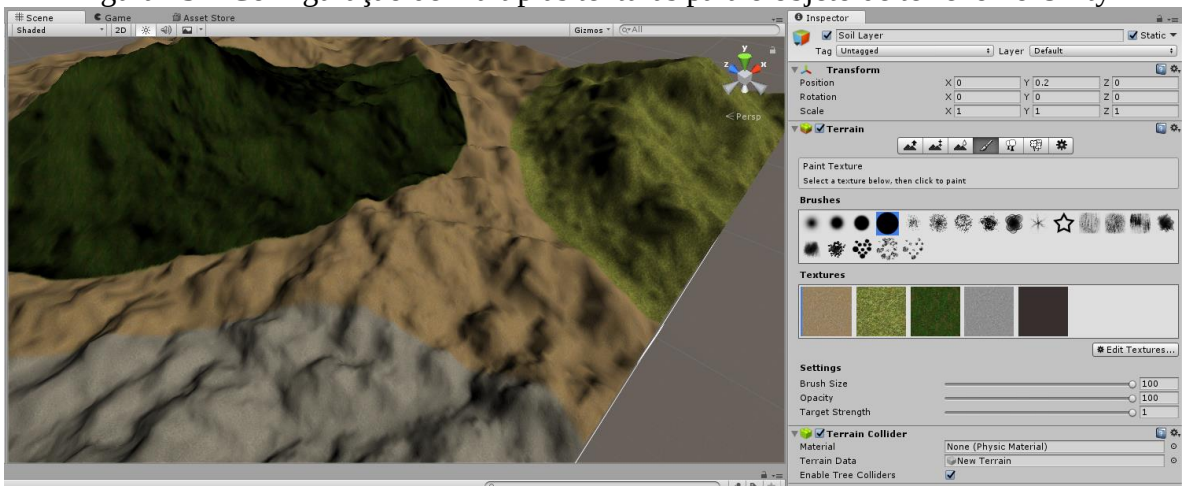
Essa validação é apenas feita sobre o ponto central, e não sobre os vizinhos, por questões de desempenho. Isso significa que a superfície na região mais alta definirá a quantidade de material que será transportado para as regiões mais baixas. Para simular a destruição de vegetação/estruturas em deslizamentos, as células que recebem material de células mais altas têm sua superfície alterada para solo exposto. Todas estas alterações podem ser vistas no Quadro 9.

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 8. 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 se baseiam 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. Estas alterações podem ser vistas no Quadro 10.

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.

Figura 15 – Configuração de múltiplas texturas para o objeto de terreno no Unity



Fonte: Elaborado pelo autor.

Quadro 8 – Valores modificadores das superfícies do terreno

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.

Quadro 9 – Adaptação do algoritmo de erosão térmica considerando superfícies de terreno

```
float[] inclinationMods = { 1.0f, 1.1f, 1.2f, 2.0f };
public void DryErosion(float[,] soil, float[,] rock, int[,] surface, float
talus, float factor)
{
    int maxX = soil.GetLength(0);
    int maxY = soil.GetLength(1);
    for (int x = 0; x < maxX; x++)
    {
        for (int y = 0; y < maxY; 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]];
            for (int relX = -1; relX <= 1; relX++)
            {
                // [Código ocultado por motivos de redundância]
                for (int relY = -1; relY <= 1; relY++)
                {
                    // [Código ocultado por motivos de redundância]
                    float diff = soil[x, y] - soil[absX, absY];
                    if (diff > maxDiff)
                        maxDiff = diff;
                    if (diff > localTalus)
                    {
                        sumDiff += diff;
                        sumDelta += factor * (diff - localTalus);
                    }
                }
            }
            // [Código ocultado por motivos de redundância]
            // Limitar a quantidade de material movimentada se não houver
            // solo o bastante
            float limiter = 1.0f;
            if (sumDelta > soilVolume)
                limiter = soilVolume / sumDelta;
            for (int relX = -1; relX <= 1; relX++)
            {
                // [Código ocultado por motivos de redundância]
                for (int relY = -1; relY <= 1; relY++)
                {
                    // [Código ocultado por motivos de redundância]
                    float diff = soil[x, y] - soil[absX, absY];
                    if (diff > localTalus)
                    {
                        float move = factor * (maxDiff - localTalus) *
(diff / sumDiff) * limiter;
                        matrix[absX, absY] += move;
                        matrix[x, y] -= move;
                        // Locais que recebem queda de material têm sua
                        // superfície destruída
                        surface[absX, absY] = 0;
                    }
                }
            }
        }
    }
}
```

Fonte: Elaborado pelo autor.

Quadro 10 – Adaptação do algoritmo de erosão hidráulica considerando superfícies de terreno

```
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[,] surfacefloat 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 drenagem
private void DrainWater(float[,] soil, float[,] water, int[,] surface,
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;
        }
    }
}
```

Fonte: Elaborado pelo autor.

3.3.2.5 INCLUSÃO DA UMIDADE RELATIVA DO SOLO

Dados referentes à umidade relativa do solo são armazenados em um mapa com o mesmo formato dos mapas de altura do solo e rocha. Os valores de ponto flutuante podem variar de 0 (solo absolutamente seco) à 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 (Quadro 11).

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 fórmula vista no Quadro 12.

Quadro 11 – Algoritmo final de drenagem de água com incremento da umidade do solo

```
// 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.

Quadro 12 – Algoritmo final para obtenção da inclinação máxima na erosão térmica

```
// Loop do algoritmo de erosão térmica

float localTalus = talus * inclinationMods[surface[x, y]];
localTalus -= (humidity[x, y] * talus) / 2;

// Resto do algoritmo de erosão térmica
```

Fonte: Elaborado pelo autor.

3.3.2.6 VISUALIZAÇÃO DE ESTATÍSTICAS

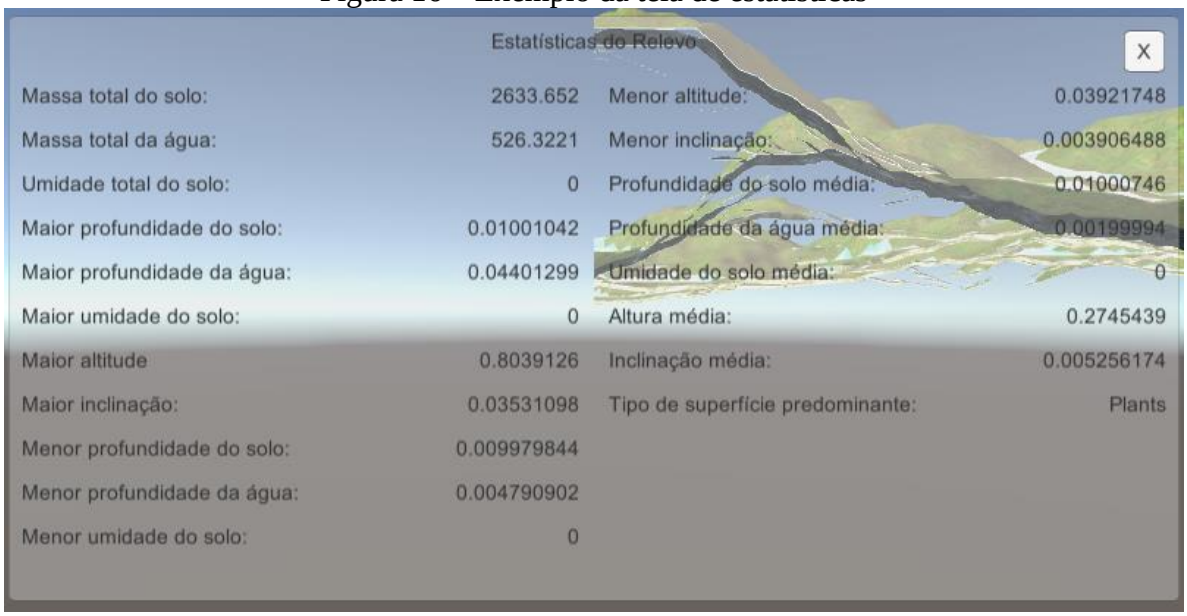
Para possibilitar uma análise numérica dos resultados da simulação, foi adicionada uma opção de visualização de estatísticas na tela principal, que pausa a simulação e exibe diversos dados numéricos relativos ao estado da paisagem no instante em que a opção foi selecionada, conforme Figura 16. A coleta de estatísticas é realizada através de um loop similar ao que ocorre nas transformações, no qual os dados são acumulados. As informações coletadas são:

d) volume total do solo;
e) volume total da água;
f) volume total de umidade no solo;
g) maior/menor profundidade de solo;
h) maior/menor profundidade da água;
i) maior/menor valor de umidade no solo;
j) maior/menor altitude;
k) maior/menor inclinação;

l) profundidade média do solo;
m) profundidade média da água;
n) altitude média;
o) inclinação média;
p) tipo de superfície mais presente;

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).

Figura 16 – Exemplo da tela de estatísticas



Fonte: Elaborado pelo autor.

3.3.2.7 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.

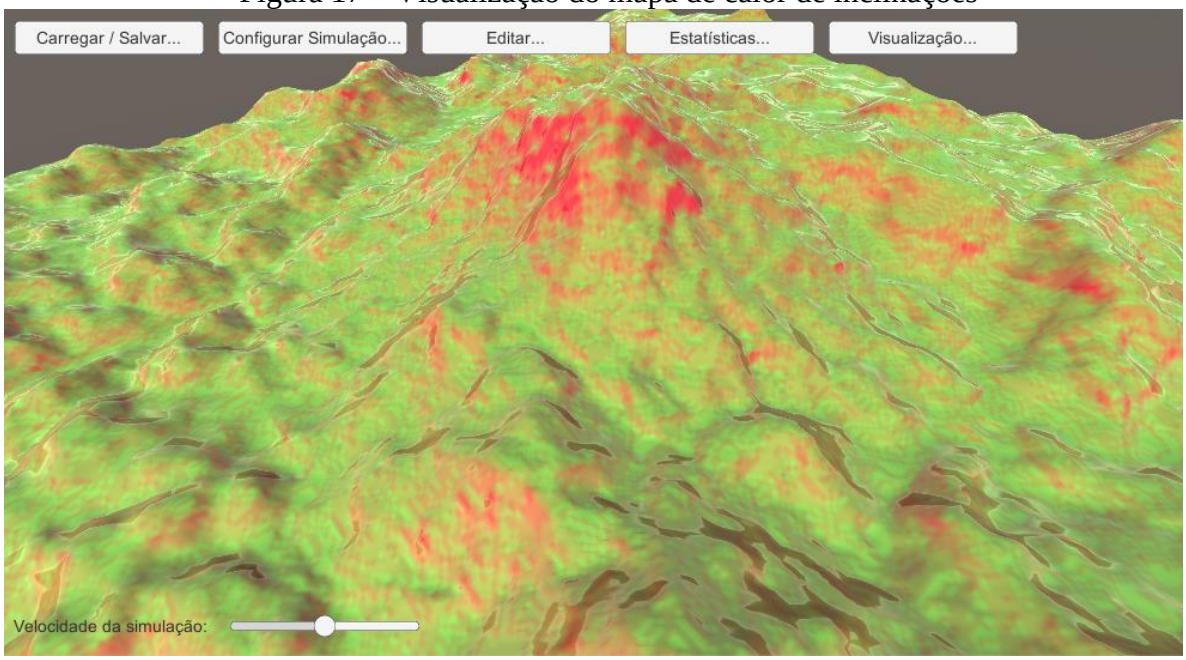
A visualização das inclinações (Quadro 13) é um caso a parte. Assim como no cálculo de estatísticas, as inclinações são sempre avaliadas de cima para baixo. Para obter o valor correspondente no mapa de calor, é extraído o valor da maior inclinação para baixo (diferença entre ponto central e vizinho), que é multiplicado por 50. A visualização do mapa de calor resultante pode ser vista na Figura 17.

Quadro 13 – Algoritmo gerado 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 17 – Visualização do mapa de calor de inclinações



Fonte: Elaborado pelo autor.

3.3.2.8 OTIMIZAÇÕES

Ao longo do desenvolvimento dos algoritmos vistos nos capítulos anteriores, foram realizados vários ajustes a fim de reduzir o número de operações executadas a cada iteração do algoritmo. Os algoritmos de transformação apresentados até agora realizavam operações utilizando uma vizinhança de 8 células, também conhecida como vizinhança Moore. Os algoritmos utilizados no programa foram adaptados para utilizar uma vizinhança de 4 células (Vizinhança Von Neumann). A diferença entre os dois tipos de vizinhança pode ser vista na Figura 18, nota-se que a vizinhança Von Neumann não realiza operações nas células diagonais à central. Esta diferença causa um impacto negativo na simulação caso seja utilizada a vizinhança Von Neumann, mas no escopo da aplicação proposta, esse impacto pode ser ignorado para fins de desempenho.

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 14.

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. Na maioria dos casos, onde todas as atualizações são executadas, o desempenho não apresenta ganho.

Figura 18 – Representação visual das vizinhanças de Moore e Von Neumann

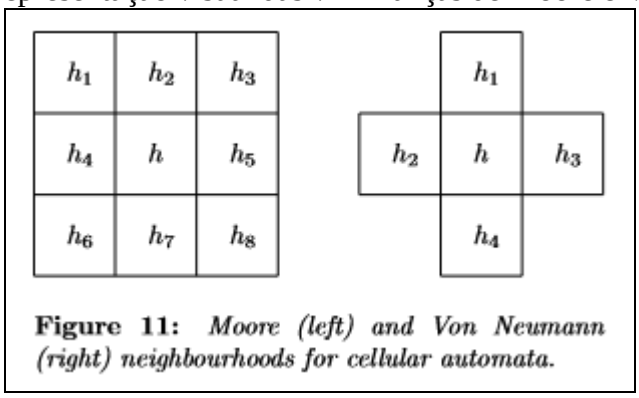


Figure 11: *Moore (left) and Von Neumann (right) neighbourhoods for cellular automata.*

Fonte: Olsen (2004).

Quadro 14 – 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(ref heights[x, y], ref heights[x - 1, y]);
    }
    if (y != 0)
    {
        Transform(ref heights[x, y], ref heights[x, y - 1]);
    }
    if (x != heights.GetLength(0) - 1)
    {
        Transform(ref heights[x, y], ref heights[x + 1, y]);
    }
    if (y != heights.GetLength(1) - 1)
    {
        Transform(ref heights[x, y], ref heights[x, y + 1]);
    }
}
```

Fonte: Elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

@@Análise dos resultados@@

4 CONCLUSÕES

@@Conclusões@@

4.1 EXTENSÕES

@@Extensões@@

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.

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.

HORNIL. **Hornil StylePix User Manual.** [S.I.], [2016?]. Disponível em: <http://hornil.com/kr/docs/stylepix/UserManual/>. Acesso em 9 mai. 2017.

OLSEN, Jacob. **Realtime Procedural Terrain Generation:** Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games. 2004. 20 f. [Tipo???] (Curso???) - University of Southern Denmark. 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.