UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Bruno Torres do Nascimento

EDIÇÃO PROCEDIMENTAL DE TERRENOS VIRTUAIS UTILIZANDO CURVAS DE BÉZIER 3D DE BAIXO GRAU EM GPU

Santa Maria, RS
2020

PPGCC/UFSM, RS NASCIMENTO, Bruno Torres do Mestre 2020

Bruno Torres do Nascimento

EDIÇÃO PROCEDIMENTAL DE TERRENOS VIRTUAIS UTILIZANDO CURVAS DE BÉZIER 3D DE BAIXO GRAU EM GPU

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em 1.03.03.05-7 Processamento Gráfico, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. Cesar Tadeu Pozzer

Santa Maria, RS
2020

Nascimento, Bruno Torres do
Edição Procedimental de Terrenos Virtuais Utilizando Curvas de Bézier 3D de Baixo Grau em GPU / Bruno Torres do Nascimento.- 2020.
60 p.; 30 cm

Orientador: Cesar Tadeu Pozzer
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2020

1. Terrenos virtuais 2. Edição procedimental 3. Curvas de Bézier 4. Computação gráfica 5. Hash espacial I. Tadeu Pozzer, Cesar II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

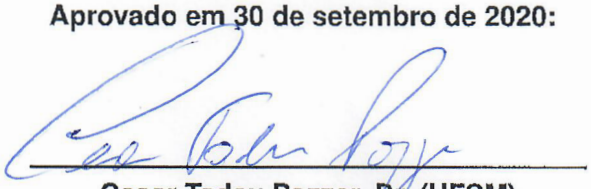
End. Eletr.: brunotn@inf.ufsm.br

Bruno Torres do Nascimento

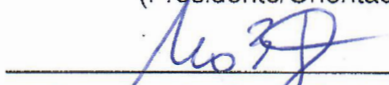
EDIÇÃO PROCEDIMENTAL DE TERRENOS VIRTUAIS UTILIZANDO CURVAS DE BÉZIER 3D DE BAIXO GRAU EM GPU

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em 1.03.03.05-7 Processamento Gráfico, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

Aprovado em 30 de setembro de 2020:



Cesar Tadeu Pozzer, Dr. (UFSM)
(Presidente/Orientador)



Mateus Beck Rutzig, Dr. (UFSM) (videoconferência)

Vinicius da Costa de Azevedo, Dr. (ETH Zürich) (videoconferência)

Santa Maria, RS
2020

DEDICATÓRIA

A Deus e aos meus pais, Marco e Rosane.

AGRADECIMENTOS

Agradecemos ao Exército Brasileiro pelo suporte financeiro através do projeto SIS-ASTROS, dentro do qual este trabalho foi desenvolvido.

RESUMO

EDIÇÃO PROCEDIMENTAL DE TERRENOS VIRTUAIS UTILIZANDO CURVAS DE BÉZIER 3D DE BAIXO GRAU EM GPU

AUTOR: Bruno Torres do Nascimento
ORIENTADOR: Cesar Tadeu Pozzer

A geração procedimental de terrenos virtuais a partir de dados geoespaciais é empregada em diversas categorias de aplicações, como simuladores virtuais, jogos e visualização de dados de Sistemas de Informação Geográfica (SIG), e frequentemente demanda a integração visual de estradas, rios e outras características do terreno. Este trabalho apresenta uma técnica procedimental para terraplenagem de caminhos em terrenos virtuais gerados a partir de Modelos Digitais de Elevação (MDE). Um caminho é uma característica do terreno com um percurso bem definido, como uma estrada ou rio. Os caminhos são carregados a partir de dados vetoriais e convertidos de polilinhas para curvas de Bézier cúbicas 3D com continuidade C^2 e parametrização de suavidade. Em seguida, as curvas cúbicas são divididas e transformadas em curvas quadráticas e enviadas para a GPU onde a nossa técnica processa todo o mapa de alturas em paralelo, calculando as movimentações de terra para criar escavações e aterros no terreno ao longo dos caminhos. A nossa técnica emprega um *hash* espacial utilizando as caixas delimitadoras das curvas para reduzir os cálculos na GPU. A abordagem de se utilizar curvas quadráticas permite que os cálculos de distância de ponto à curva sejam realizados de forma analítica, promovendo uma melhora substancial no desempenho quando comparado às abordagens numéricas, e permitindo que se alcance tempos compatíveis com execução em tempo real. Além disso, a utilização de um conjunto de parâmetros de terraplenagem permite que o usuário controle as dimensões e o perfil dos caminhos esculpidos no terreno, modificando a largura, a inclinação lateral, e a distância de suavização, produzindo caminhos suaves e com aparência natural.

Palavras-chave: Curvas de Bézier. Edição Procedimental. Terrenos Virtuais. Hash Espacial. Terraplenagem.

ABSTRACT

PROCEDURAL EDITING OF VIRTUAL TERRAINS USING LOW DEGREE 3D BÉZIER CURVES ON GPU

AUTHOR: Bruno Torres do Nascimento
ADVISOR: Cesar Tadeu Pozzer

The procedural generation of virtual terrains from geospatial data is used in several categories of applications, such as virtual simulators, games, and visualization of GIS data, and often demands the visual integration of roads, rivers, and other terrain features. This work presents a procedural technique for carving paths on virtual terrains generated from Digital Elevation Models (DEM). A path is a feature on the terrain with a well-defined route, such as a road or river. The paths are loaded from vector data and converted from polylines to 3D cubic Bézier curves with C^2 continuity and smoothness parameterization. Then, the cubic curves are divided and transformed into quadratic curves and sent to the GPU where our technique processes the entire heightmap in parallel, calculating the displacements to create excavations and embankments on the terrain along the paths. Our technique employs a spatial hash using the bounding boxes of the curves to reduce calculations on the GPU. The approach of using quadratic curves allows the calculations of point-to-curve distances to be carried out analytically, promoting a substantial improvement in performance when compared to numerical approaches, and allowing reaching times compatible with real-time execution. In addition, the use of a set of carving parameters allows the user to control the dimensions and profile of the paths sculpted in the terrain, changing the width, the lateral inclination, and the smoothing distance, producing smooth and natural-looking paths.

Keywords: Bézier Curves. Procedural Editing. Virtual Terrains. Spatial Hashing. Earthwork.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DEM	*Digital Elevation Model*
GIS	*Geographic Information System*
MDE	Modelo Digital de Elevação
MDT	Modelo Digital de Terreno
MDS	Modelo Digital de Superfície
SIG	Sistema de Informação Geográfica

LISTA DE SÍMBOLOS

α	Ângulo de inclinação
λ	Distância de suavização
Ω	Fator de suavização
Ψ	Parâmetro linear de terraplenagem
Θ	Intensidade da terraplenagem
w	Largura do caminho

SUMÁRIO

1 INTRODUÇÃO

Terrenos virtuais são componentes importantes em vários tipos de aplicações, como simuladores virtuais, visualização de dados geoespaciais e jogos. Criar terrenos virtuais com estradas, rios e outras características é uma tarefa demorada e pode rapidamente se tornar impraticável de ser feita manualmente conforme a escala do terreno aumenta. Dessa forma, terrenos em grande escala geralmente exigem abordagens procedimentais, que devem ser eficientes e capazes de fornecer resultados visualmente aceitáveis e consistentes.

A modelagem de terrenos virtuais pode ser feita de diferentes maneiras, dependendo da natureza da aplicação e da escala do terreno. Cenas estáticas e menores, como muitas vezes vistas em jogos, geralmente podem ser feitas à mão por um artista, enquanto cenas muito grandes ou dinâmicas exigem uma abordagem capaz de produzir resultados com mínima ou nenhuma interação do usuário. Simuladores de treinamento militar, por exemplo, geralmente precisam ser capazes de gerar ambientes de treinamento completos a partir de conjuntos de dados geoespaciais.

Terrenos virtuais que representam locais do mundo real são frequentemente modelados usando dados de Sistemas de Informação Geográfica (SIG). Dados georreferenciados são ideais para criar cenas de grande escala e geoespacialmente precisas, onde os dados rasterizados são usados para modelar a topografia geral do terreno virtual e os dados vetoriais codificam a localização de características do terreno. Rios, estradas e outros elementos que seguem um trajeto bem definido são representados por sequências de vértices que descrevem polilinhas, enquanto lagos, florestas e limites da cidade são representados por polígonos.

Imagens *raster* georreferenciadas criadas a partir de Modelos Digitais de Elevação (MDE) são comumente usadas para representar as elevações do terreno. Uma das vantagens dessa abordagem é que a imagem pode ser modificada procedimentalmente, permitindo a incorporação de elementos vetoriais na topografia do terreno, e reduzindo a necessidade de edição manual.

Automatizar a incorporação de elementos vetoriais é um desafio importante no processo de criação de terrenos virtuais de grande escala. Especificamente, há uma demanda por métodos procedimentais projetados para esculpir trajetos em terrenos gerados a partir de dados geoespaciais, que, como mencionado anteriormente, são empregados em vários tipos de aplicações. Características do terreno, como estradas e rios, são um elemento intrínseco na maioria dos mundos virtuais e precisam ser posicionados corretamente para obter bons resultados visuais e técnicos. Inconsistências, como uma estrada excessivamente inclinada em uma encosta ou um rio que corre morro acima, podem ser muito visíveis e afetar o realismo da cena ou prejudicar a execução de uma simulação.

A fase inicial de pesquisa deste trabalho gerou uma publicação (NASCIMENTO; POZZER; FRANZIN, 2019) que, posteriormente, foi estendida e melhorada sob vários aspectos. Ao final deste trabalho, esses aspectos são abordados de forma comparativa com o propósito de destacar a evolução da pesquisa.

1.1 OBJETIVOS

O foco principal deste trabalho é apresentar uma técnica procedimental capaz de modificar a topografia de terrenos virtuais para incorporar elementos vetoriais representando rios e estradas de maneira visualmente satisfatória. Essa incorporação é realizada através de um processo de terraplenagem virtual que modifica as elevações do terreno, criando escavações e taludes ao longo do trajeto dos elementos vetoriais para a obtenção de um resultado com uma aparência mais natural.

A técnica proposta visa a execução em tempo real, permitindo ao usuário ou aplicação modificar dinamicamente os parâmetros de terraplenagem e alterar o perfil dos trajetos, favorecendo os aspectos artísticos do processo de criação de terrenos virtuais.

Em dados de SIG, os elementos vetoriais que representam rios e estradas são tradicionalmente definidos por polilinhas e, por isso, apresentam trajetos com cantos e curvas acentuadas que se destacam e os fazem parecer não naturais. Para tornar os trajetos mais suaves e mais realistas, este trabalho tem também como objetivo apresentar um método procedimental para converter polilinhas para curvas de Bézier. Curvas paramétricas permitem a criação de trajetos suaves com resoluções virtualmente infinitas e sem a necessidade de um maior conjunto de dados de entrada.

Aspectos como instanciação e geração procedimental de modelos 3D de estradas ou malhas d'água, texturização e renderização de elementos, bem como simulação de erosão hidráulica *não* estão no escopo deste trabalho.

1.2 MÉTODO PROPOSTO

Neste trabalho, propomos uma técnica procedimental para esculpir trajetos de rios e estradas em terrenos virtuais criados a partir de mapas de alturas. Nossa técnica permite a parametrização das dimensões e perfil dos trajetos, tornando-os nivelados ou inclinados e alterando o resultado visual da sua incorporação ao terreno.

Nossa abordagem consiste em calcular o deslocamento vertical necessário para ajustar a topografia do terreno e armazenar os valores de deslocamento em uma textura separada, mantendo assim os valores originais do mapa de alturas inalterados. Isso tem a

vantagem de permitir que as alterações sejam desfeitas pelo usuário, para que o processo de edição possa ser mais dinâmico. Após a conclusão da edição, o usuário pode escolher aplicar os deslocamentos no mapa de alturas definitivamente. Nossa abordagem foi projetada para aproveitar a GPU e seu poder de processamento paralelo, permitindo a edição em tempo real pelo usuário.

Apresentamos um algoritmo para converter elementos vetoriais de polilinhas para curvas de Bézier cúbicas 3D com continuidade C^2 entre curvas adjacentes.

Em relação ao desempenho, propomos uma abordagem de *hash* espacial usando a caixa delimitadora das curvas de Bézier para reduzir a quantidade de curvas avaliadas na GPU e melhorar o tempo de execução.

Por fim, apresentamos um algoritmo para converter curvas de Bézier 3D cúbicas para quadráticas com continuidade C^1 em duas dimensões, permitindo um ganho ainda maior de desempenho nos cálculos de distância de ponto à curva.

1.3 CONTRIBUIÇÕES

Nossas contribuições são as seguintes:

- um método para esculpir procedimentalmente terrenos virtuais ao longo de trajetos definidos por curvas de Bézier 3D;
- uma abordagem utilizando *hash* espacial para avaliar curvas de Bézier na GPU;
- um algoritmo para converter polilinhas para curvas de Bézier cúbicas 3D;
- um algoritmo para converter curvas de Bézier 3D cúbicas para quadráticas.

1.4 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está estruturado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica. O Capítulo 3 explora trabalhos relacionados. No Capítulo 4, apresentamos uma visão geral da nossa abordagem, que é discutida mais detalhadamente nos Capítulos 5 a 7. O Capítulo 8 trata sobre detalhes de implementação. Finalmente, nos Capítulos 9 e 10, apresentamos e discutimos os resultados alcançados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 DADOS *RASTER*

Dados *raster* são um tipo de representação discreta de informações em um espaço regular. Esse tipo de representação é comumente utilizado para imagens, onde as informações são organizadas em uma matriz regular bidimensional (Figura 2.1), mas também existem representações com três ou mais dimensões, dependendo da aplicação. Estas posições são chamadas individualmente de *pixel* (do inglês *picture element*) no contexto de telas, ou *texel* (do inglês *texture element*) quando se referem a uma textura, e cada posição armazena a cor, ou valor, associada àquela área retangular da imagem. Essa forma de representação de dados é a base de vários formatos populares de arquivos de imagem, como *Bitmap*, *TIFF* e *PNG*.

Uma limitação fundamental desse tipo de representação de dados é que a qualidade e precisão estão limitadas à resolução do conjuntos de dados, e o aumento da resolução é diretamente proporcional a um aumento no uso da memória disponível, o que pode ser um fator impeditivo em algumas situações. No contexto deste trabalho, as informações de elevação do terreno são armazenadas em uma imagem *raster*, representada por uma textura bidimensional chamada de *mapa de alturas*. Ainda, os valores de deslocamento vertical da superfície do terreno também são armazenados em uma textura bidimensional chamada de *mapa de deslocamentos*. Ambas as texturas são armazenadas em GPU durante a execução.

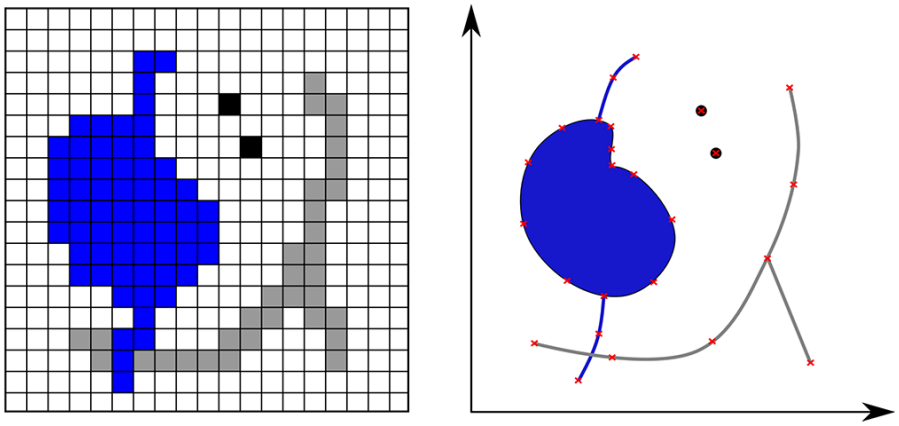


Figura 2.1 – Comparação entre formatos diferentes de representação de um conjunto de dados. Formato *raster* à esquerda e formato vetorial à direita. Adaptado de Engel (2018).

2.2 DADOS VETORIAIS

Outra forma amplamente utilizada para representação espacial de elementos é através do uso primitivas geométricas, como pontos, linhas e polígonos (Figura 2.1). Nesse tipo de representação, os elementos geométricos são avaliados utilizando fórmulas matemáticas que descrevem as primitivas utilizadas, o que possibilita a obtenção de uma resolução virtualmente infinita. Essa característica faz com que dados vetoriais sejam comumente empregados em vários formatos de arquivos de imagem e texto, como *EPS*, *SVG* e *PDF*.

Um das principais vantagens desse tipo de representação é que os objetos podem, em teoria, ser escalados para qualquer tamanho sem perda de precisão ou qualidade. Ainda, dados vetoriais são uma maneira eficiente de representação do ponto de vista de armazenamento, e frequentemente são preferidos ao invés de representações *raster* quando o consumo de memória é um aspecto importante. No contexto deste trabalho, utilizamos dados vetoriais para representar o trajeto de rios e estradas. Esses trajetos são descritos inicialmente por sequências de vértices, representando segmentos de retas consecutivos, chamadas de *polilinhas* e, posteriormente, são convertidos para curvas de Bézier.

2.3 DADOS DE SIG

Dados de um Sistema de Informação Geográfica (SIG) — termo adotado mais amplamente em inglês: *Geographic Information System* (GIS) — conceitualmente são empregados para representar diversos tipos de elementos existentes em uma determinada área geográfica do mundo real e, consequentemente, são georreferenciados. O termo SIG é utilizado em vários contextos e, por isso, acaba recebendo definições distintas. Chrisman (1999) define SIG como uma atividade organizada, pela qual as pessoas medem e representam fenômenos geográficos e então transformam essas representações em outras formas, enquanto interagem com as estruturas sociais. Chrisman (1999) observou, ainda, que o termo SIG passou a simbolizar uma tecnologia, uma indústria e uma forma de trabalho.

As formas mais comuns de representação utilizadas em SIG envolve o uso de dados vetoriais e *raster*, dependendo da natureza do elemento representado. Informações espaciais discretizadas ou geradas a partir de amostragens em intervalos regulares, como elevações do terreno, costumam ser representadas por *raster*, enquanto elementos com limites físicos ou trajetos bem definidos — como rios, estradas, lagos etc. — são normalmente representados vetorialmente. Considerando essas formas de representação de dados, podemos aplicar trivialmente os métodos propostos neste trabalho a conjuntos de dados de SIG.

2.4 MODELOS DIGITAIS DE ELEVAÇÃO

Um Modelo Digital de Elevação (MDE) é uma forma de representação de elevações de um terreno. O termo MDE costuma ser utilizado de forma genérica para se referir a dois tipos de modelos de representação de terrenos: o Modelo Digital de Terreno (MDT), e o Modelo Digital de Superfície (MDS). A diferença principal entre esses modelos é que o MDT é uma representação altimétrica da superfície do terreno em seu estado "puro", desconsiderando prédios, árvores, ou outros elementos. Já o MDS registra a elevação de qualquer elemento sobre a superfície do terreno. Ambos os modelos armazenam os valores de elevação do terreno em imagens *raster*.

Em computação gráfica, esse tipo de imagem é chamada também de mapa de elevações, ou mapa de alturas (em inglês, *heightmap*, ou *height map*), sendo uma técnica amplamente utilizada para modelar terrenos virtuais (SMELIK et al., 2014; FREIKNECHT; EFFELSBERG, 2017; GALIN et al., 2019). Nessa técnica, um campo escalar de valores de elevações do terreno — geralmente armazenado em uma textura — é usado para deslocar os vértices de uma malha de grade regular, modelando a superfície do terreno (Figura 2.2). Neste trabalho, consideramos que o terreno virtual é representado por um mapa de alturas.

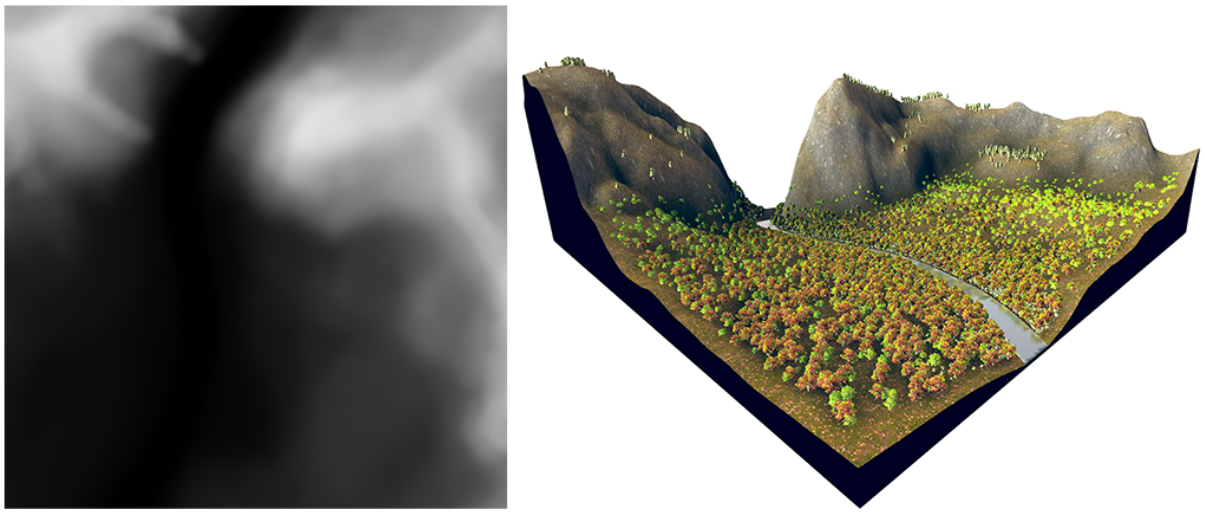


Figura 2.2 – Um mapa de alturas (esquerda) utilizado para representar e renderizar um terreno virtual (direita). Adaptado de Nascimento, Franzin e Pozzer (2018).

2.5 *HASH* ESPACIAL

Um *hash* espacial é uma técnica de particionamento do espaço em n dimensões utilizada para definir agrupamentos de elementos baseados nas suas posições. Essa técnica é bastante empregada para mapear objetos em um espaço bidimensional para uma tabela *hash*. A função usada para fazer este mapeamento é chamada de função *hash*. Em

2D, as áreas de particionamento do espaço são comumente chamadas de células *hash*.

Uma das principais aplicações dessa técnica é para reduzir o número de consultas a objetos seguindo um critério de proximidade espacial. Técnicas de *ray tracing* (LAGAE; DUTRÉ, 2008), *steering behaviour* (POZZER et al., 2014) e renderização de elementos vetoriais (FRASSON; ENGEL; POZZER, 2018) costumam empregar *hashes* espaciais com frequência.

2.6 CURVAS DE BÉZIER

Curvas de Bézier são um tipo de curva paramétrica definida por um polinômio de grau n e construídas a partir de $n + 1$ pontos de controle (LENGYEL, 2012). O primeiro e último pontos de controle são interpolados pela curva, e os demais são aproximados (Figura 2.3). Essas curvas são efetivamente representações vetoriais e, por isso, têm uma precisão absoluta. Historicamente, curvas de Bézier foram desenvolvidas por projetistas de automóveis para descrever a forma de painéis exteriores de carros (BUSS, 2003).

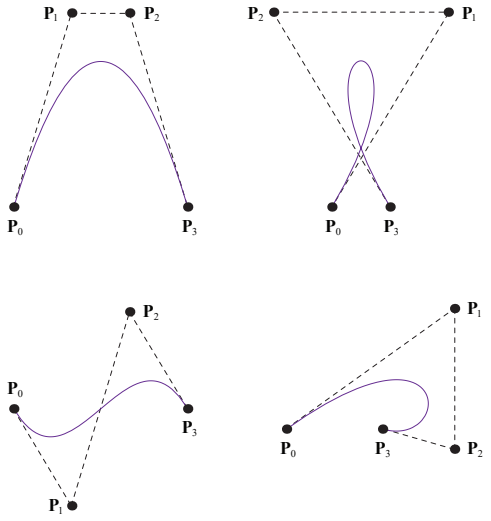


Figura 2.3 – Uma variedade de curvas de Bézier cúbicas com quatro pontos de controle P_0, P_1, P_2 e P_3 (LENGYEL, 2012).

Para se representar caminhos mais longos, ou mais complexos, é comum que várias curvas de Bézier sejam conectadas formando curvas compostas e contínuas. A forma como essa curvas se comportam no seus pontos de conexão determina o seu tipo de *continuidade*. Esse conceito de continuidade é fundamental quando se busca representar caminhos suaves. Neste trabalho, os conceitos de continuidade paramétrica C^1 e C^2 referem-se às curvas compostas onde a primeira e segunda derivadas, respectivamente, são contínuas no ponto de conexão entre duas curvas adjacentes, o que significa uma continuidade tangencial e, consequentemente, curvas suaves (BUSS, 2003).

3 REVISÃO BIBLIOGRÁFICA

A geração procedimental de conteúdo para criação de mundos virtuais é um tópico bastante amplo e que se ramifica em diversas áreas abordando questões mais granulares, desde a geração da topografia de terrenos virtuais (EBERT et al., 2003), distribuição de vegetação (NASCIMENTO; FRANZIN; POZZER, 2018), criação de redes de rios e estradas (SMELIK et al., 2011), até a modelagem individual de plantas (LONGAY et al., 2012), áreas urbanas (MÜLLER et al., 2006) e outros elementos.

Neste trabalho, buscamos identificar na literatura relacionada trabalhos que apresentam técnicas procedimentais para geração de terrenos virtuais e, em especial, abordagens que possibilitem a modificação da topografia do terreno para incorporação do trajeto de rios e estradas a partir de dados vetoriais.

3.1 RIOS

Em sua pesquisa sobre geração procedimental de mundos virtuais, Freiknecht e Effelsberg (2017) notaram que a criação de rios é frequentemente abordada como sendo uma parte integrante do processo de geração do terreno, ou como um passo separado onde o terreno é modificado posteriormente para incluir os rios. Smelik et al. (2014) observaram que algumas das estratégias mais comuns para geração de rios envolve analisar o mapa de alturas do terreno para encontrar potenciais rotas de fluxo da água. Abordagens que provêm um maior controle ao usuário, em geral, não garantem um grau consistente de realismo quanto aos aspectos hidrológicos que governam a existência de rios na natureza e vice-versa. Além disso, a consideração pela capacidade de edição e execução em tempo real não é comum.

Diversos trabalhos apresentam abordagens que aplicam conceitos de hidrologia para simular o fluxo de cursos de água e a erosão hidráulica causada no terreno. Kelley, Malin e Nielson (1988) apresentaram um técnica baseada em modelos empíricos de erosão para simular o efeito de cursos de água sobre na superfície do terreno. Sua abordagem usa uma estrutura de árvore para representar as junções dos cursos de água tributários aos principais (Figura 3.1). Teoh (2008) apresentou uma ferramenta para geração automática de terrenos chamada *WaterWorld*. Essa ferramenta permite ao usuário adicionar corpos d'água que são utilizados para simular a erosão do terreno seguindo um conjunto de parâmetros (Figura 3.2). O efeito da erosão no terreno é simulado através de uma interpolação linear entre a altura do leito do rio e o ponto do terreno considerado.

Génevaux et al. (2013) apresentaram um *framework* para modelagem de terrenos usando conceitos inspirados em hidrologia. A sua abordagem consiste, primeiramente,

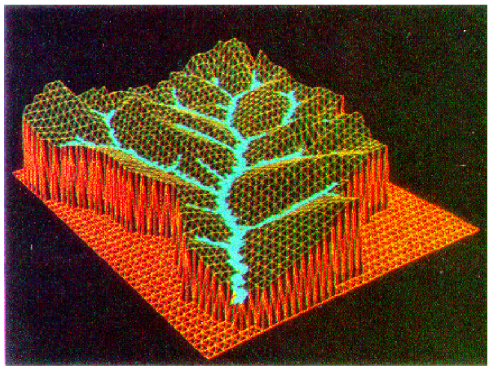


Figura 3.1 – Representação em *wireframe* de um terreno erodido por cursos de água (KELLEY; MALIN; NIELSON, 1988).

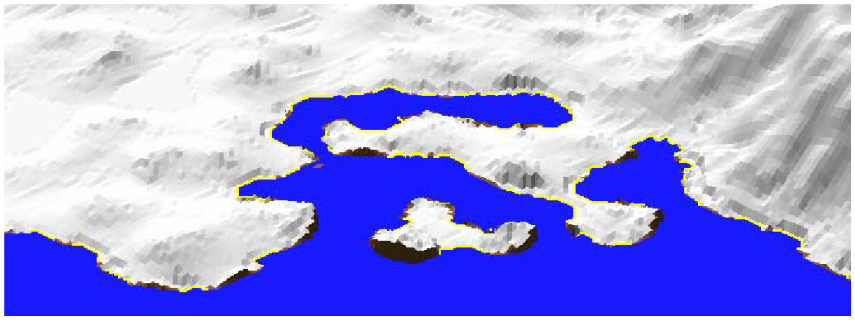


Figura 3.2 – Resultados de uma simulação de erosão litorânea produzindo penhascos e praias (TEOH, 2008).

em gerar uma rede de rios para formar um bacia hidrográfica completa a partir de um esboço inicial do usuário com alguns rios principais. A partir dessa bacia hidrográfica, toda a topografia do terreno é derivada garantindo a consistência e realismo dos aspectos hidrográficos. Diferentemente da maioria das abordagens, os autores representaram o terreno por funções analíticas contínuas, ao invés de dados raster. Cada rio é classificado conforme Rosgen (1994) e atribuído a uma de nove categorias com um perfil de escavação associado, gerando leitos diferentes (Figura 3.3).

Huijser et al. (2010) apresentaram um abordagem para escavar o percurso de um rio no terreno que possibilita ao usuário definir o perfil da seção transversal do rio (Figura 3.4). Perfis com diferentes larguras e formas podem ser definidos e aplicados alternadamente ao longo do percurso, conforme a curvatura do rio, para criar variedade visual. A abordagem proposta entrega ao usuário mais controle, enquanto sacrifica realismo físico, já que não garante que o percurso do rio obedeça a gravidade.

Mais recentemente, Peytavie et al. (2019) abordaram a criação de redes de rios com geometria animada em tempo real simulando a movimentação da água em tipos diferentes de rios, desde fluxos lentos até corredeiras. O *framework* apresentado pelos autores deriva os cursos de água de um bacia hidrográfica a partir de um mapa de alturas representando o terreno, e considera aspectos de hidrologia para obtenção de trajetos mais realistas. Os tipos de rios seguem a classificação de Rosgen (1994), e determinam as características

Figura 3.3 – Rio esculpido em uma paisagem utilizando o método baseado em hidrologia proposto por Génevaux et al. (2013).

do curso de água, como largura e profundidade, que são utilizadas para determinar o perfil da seção transversal utilizada para escavar o terreno (Figura 3.5).

3.2 ESTRADAS

McCrae e Singh (2009) apresentaram um sistema baseado em esboços para o desenho conceitual de redes de caminhos 3D, onde uma curva 3D é criada a partir de um esboço 2D feito pelo usuário, empregando uma abordagem de ajuste de clotóide. Clotóides — também chamadas de espirais de Euler (LEVIEN, 2008) — são um tipo de curva usada popularmente para modelar o traçado de estradas graças a sua característica de apresentar uma variação linear de curvatura. Ainda, o sistema apresentado por McCrae e Singh (2009) é focado em prover funcionalidades que facilitem a interação e permitam ao usuário criar rapidamente, a partir de esboços de curvas, redes de estradas com interseções, viadutos, pontes, túneis e sinalização (Figura 3.6). A geração da geometria das estradas, incluindo os pilares de viadutos e pontes, e túneis é gerada procedimentalmente. A topografia do terreno, no entanto, permanece inalterada e não é modificada para se adequar às estradas.

É comum o emprego de algoritmos de busca de caminhos com heurísticas variadas para criar redes rodoviárias. Galin et al. (2010) propuseram um método automático para geração de estradas utilizando um algoritmo ponderado de busca de caminhos (Figura 3.7). A partir da entrada de um ponto inicial e final, é criado automaticamente um caminho conectando os pontos, onde a trajetória busca minimizar uma função de custo que considera características da cena, como inclinação do terreno, obstáculos naturais, rios, lagos e florestas. Os autores mencionaram que os vértices do terreno são alterados

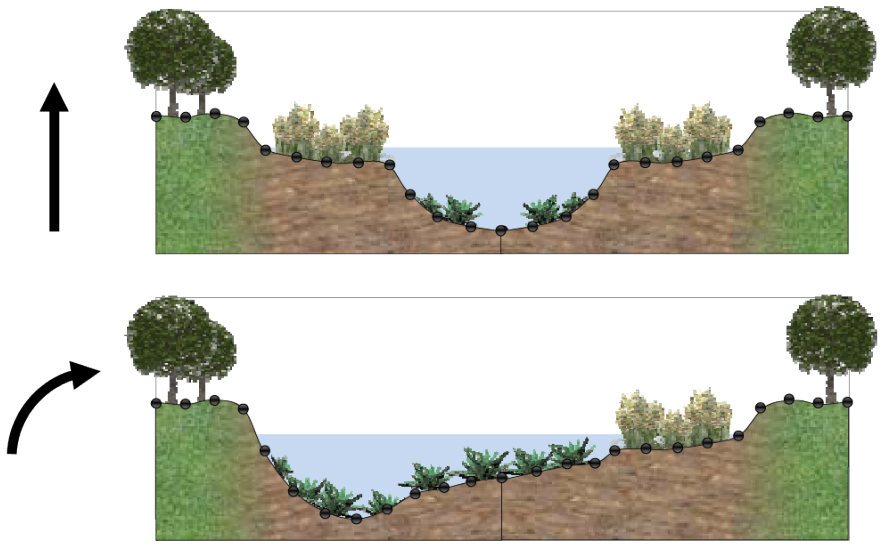


Figura 3.4 – Dois perfis representando a seção transversal de um rio conforme a sua curvatura, indicada pelas setas (HUIJSER et al., 2010).

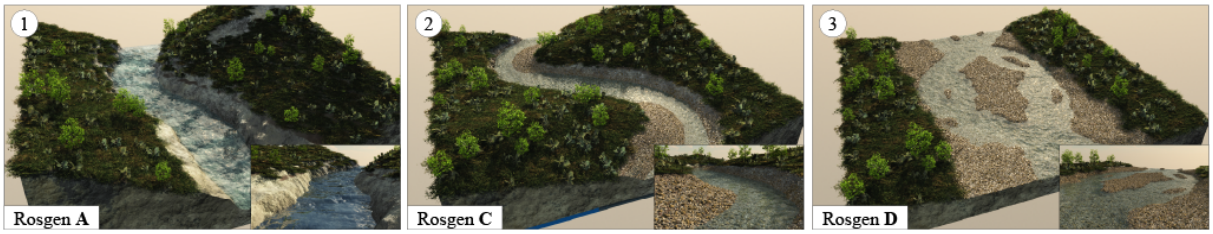


Figura 3.5 – Rios de diferentes tipos, segundo a classificação de Rosgen (1994), gerados pelo *framework* de Peytavie et al. (2019).

aplicando um perfil de seção transversal para esculpir a superfície do terreno ao longo do trajeto da estrada, embora não sejam apresentados tempos de processamento desta tarefa.

A abordagem apresentada por Galin et al. (2011) emprega um algoritmo de geração de grafo geométrico baseado em uma métrica não euclideana, combinado com um algoritmo de mesclagem de caminhos para criar redes rodoviárias hierárquicas combinando diferentes tipos de estradas. O método apresentado fornece controle ao usuário, permitindo manipular a densidade padrão da rede. Além disso, a geometria das estradas é criada automaticamente utilizando modelos 3D parametrizados. A escavação do terreno é mencionada mas não é abordada nesse trabalho.

Zhang et al. (2019) apresentaram um método para modelagem 3D de estradas usando modelos pré-definidos. O método gera as estradas a partir de dados vetorias SIG e analisa a superfície do terreno para criar e posicionar os modelos de estradas de maneira procedimental. A Figura 3.9 mostra um exemplo de modelo gerado utilizando o método descrito. Apesar de considerar a topografia do terreno para a colocação das estradas, a abordagem não esculpe ou altera o terreno de forma alguma.

Vários trabalhos abordam o problema de geração de estradas do ponto de vista

Figura 3.6 – Redes de estradas geradas procedimentalmente (MCCRAE; SINGH, 2009).

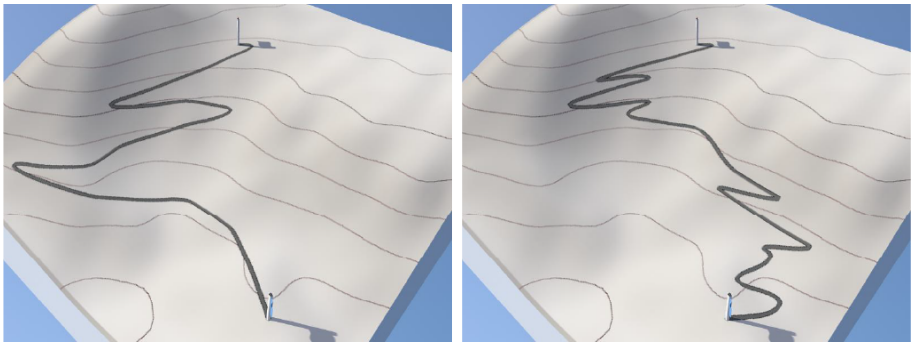


Figura 3.7 – Resultados de buscas de caminhos com diferentes restrições de curvatura (GALIN et al., 2010).

de áreas urbanas, onde existe uma demanda por otimização do *layout* da estradas e restrições específicas quanto ao tamanho de quarteirões e espaço ocupado. Trabalhos apresentados por Parish e Müller (2001) e Kelly e McCabe (2007) focaram na modelagem procedimental de cidades e *layout* de estradas (Figura 3.10); no entanto, esses trabalhos não resolvem o problema de adaptação do terreno.

Applegate, Laycock e Day (2011, 2012) apresentaram um sistema baseado em esboços para projeto de rodovias com o objetivo de criar simulações de tráfego. Nessa abordagem, um esboço criado pelo usuário é convertido para uma curva clotóide (Figura 3.11), de forma similar ao apresentado por McCrae e Singh (2009). O terreno é escavado a longo do trajeto restringindo a altura de seus vértices usando a malha da estrada.

Nguyen, Desbenoit e Daniel (2014) apresentaram uma abordagem para construção de curvas representando estradas com traçados realistas a partir de dados de SIG definidos por polilinhas. A abordagem apresentada estima a possibilidade de junção entre segmentos de estradas próximos para gerar caminhos contínuos (Figura 3.12). É aplicado um método de minimização de erro para garantir um maior nível de fidelidades entre as curvas geradas e os dados de entrada, preservando uma continuidade G^1 entre segmentos de curvas consecutivos. Ainda, os autores não abordaram formas de modificação da topografia do terreno baseada no trajeto das estradas.

Bruneton e Neyret (2008) apresentaram um método para renderizar e preencher

Figura 3.8 – Diferentes etapas do processo de geração de uma rede hierárquica de estradas utilizando o método proposto por Galin et al. (2011).

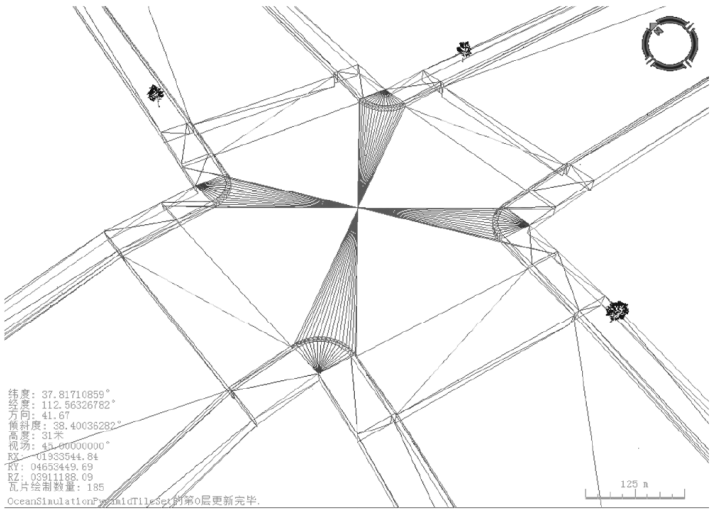


Figura 3.9 – Estrutura da malha de um interseção de estradas gerada procedimentalmente (ZHANG et al., 2019).

terrenos muito grandes com características a partir de dados vetoriais (Figura 3.13). Para representar trajetos de estradas e rios, eles usam curvas de Bézier 2D. Os autores consideram que os dados vetoriais de entrada já são curvas de Bézier e, portanto, não fornecem um método para gerar as curvas, embora seja possível que o usuário altere os pontos das curvas. Ainda, a abordagem apresentada gera uma malha para cada curva, a qual é usada para nivelar o terreno ao longo dos caminhos. Como as curvas 2D não registram altura, elas são restritas à superfície do terreno, ao contrário de nossa abordagem.

Thöny, Billeter e Pajarola (2016) apresentam um trabalho focado na renderização de elementos vetoriais onde é empregado um método de renderização diferida de linhas associado a uma técnica de *hash* espacial para melhorar o desempenho ao trabalhar-se com conjuntos de dados muito grandes. Ainda, Frasson, Engel e Pozzer (2018) apresentam uma abordagem em espaço de tela para renderizar elementos vetoriais em terrenos virtuais. Seu trabalho emprega uma estrutura de dados eficiente usada para acessar os dados vetoriais na GPU. Embora ambos os trabalhos obtenham bons resultados de renderização (Figura 3.14), suas abordagens não alteram a topografia do terreno para se adequar ao elementos vetoriais e nem empregam curvas de Bézier, uma vez que são projetadas para trabalhar com polígonos e polilinhas.

Para suportar a edição de mapas de alturas em tempo real, precisamos de técnicas

Figura 3.10 – *Layout* de estradas gerado utilizando a abordagem apresentada por Parish e Müller (2001).

Figura 3.11 – Estradas geradas a partir de esboços (APPLEGATE; LAYCOCK; DAY, 2011).

muito eficientes para avaliar as curvas de Bézier. Mais especificamente, precisamos de uma abordagem eficiente para calcular a distância de ponto à curva. Vários trabalhos propõem técnicas de redução de graus de curvas de Bézier, com diversas aplicações, como aplicativos de desenho auxiliado por computador e conversão de caracteres de fontes. Trabalhos de Chen e Wang (2002) e Lu e Wang (2006a, 2006b) apresentaram abordagens para reduzir em vários graus as curvas de Bézier. O primeiro lida com restrições de continuidade do ponto final, e o último com a continuidade G^1 e G^2. Essas abordagens são voltadas para curvas de grau geral e não são adequadas para o nosso caso, que é a conversão de curva cúbica em quadrática.

Figura 3.12 – Um rede complexa de estradas reconstruída a partir de dados de SIG utilizando o algoritmo proposto por Nguyen, Desbenoit e Daniel (2014)

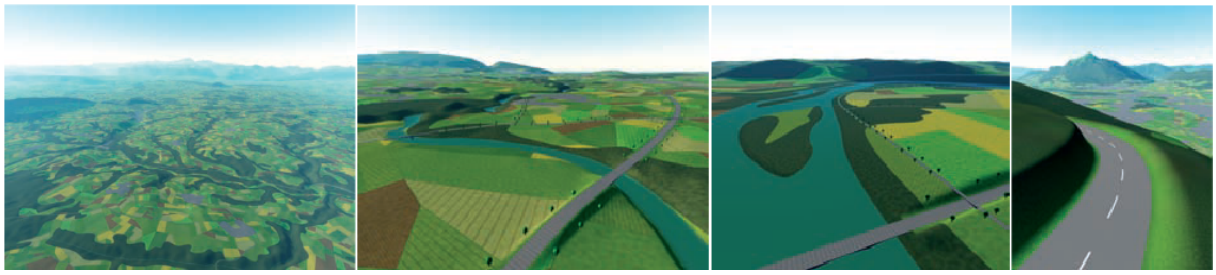


Figura 3.13 – Diferente vistas de um terreno renderizado e populado com elementos utilizando a abordagem proposta por Bruneton e Neyret (2008).

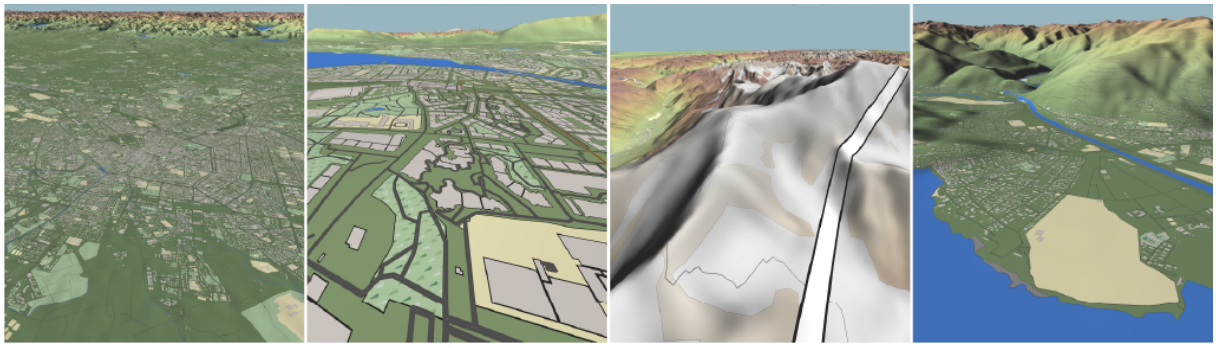


Figura 3.14 – Diferente vistas de um conjunto de dados vetoriais renderizados utilizando a técnica apresentada por Frasson, Engel e Pozzer (2018).

4 VISÃO GERAL DA TÉCNICA PROPOSTA

Esta seção descreve resumidamente todos os passos que nosso método segue para realizar o deslocamento vertical da superfície do terreno (Figura 4.1). Mais detalhes são apresentados nos Capítulos 5 a 7.

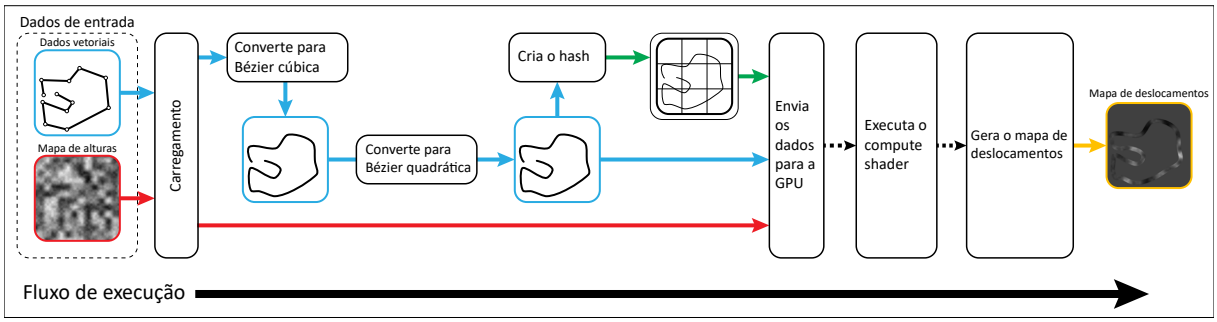


Figura 4.1 – As setas coloridas representam o fluxo de cada tipo de dado. Os dados raster são vermelhos; os dados vetoriais são azuis; o *hash* espacial é verde; e os valores de deslocamento são amarelos.

Primeiramente, os dados *raster* (mapa de alturas) e os dados vetoriais (rios e estradas) são carregados. Os dados vetoriais são representados por listas de vértices (polilinhas) que descrevem os caminhos.

A segunda etapa converte as polilinhas de entrada para curvas Bézier cúbicas 3D. Cada par de vértices que compõe um segmento de linha é utilizado para gerar uma curva com quatro pontos de controle. Os dois pontos de controle extremos são os vértices originais, e os outros dois pontos de controle intermediários são gerados com base na direção relativa entre segmentos de linha adjacentes e seus comprimentos (Seção 5.1).

Na terceira etapa, cada curva de Bézier cúbica é dividida em duas usando o algoritmo de de Casteljau. Cada curva é dividida em seu ponto de inflexão, se houver, caso contrário, seu ponto médio é usado. Em seguida, as curvas cúbicas são convertidas para curvas quadráticas usando o algoritmo apresentado na Seção 5.2.

A quarta etapa é a criação um *hash* espacial 2D para reduzir o número de avaliações de curvas na GPU. A caixa delimitadora de cada curva quadrática é usada para construir o *hash* espacial. Além disso, as estruturas de dados que contêm os dados necessários para os cálculos de deslocamento são criadas nesta etapa e descritos no Capítulo 6.

Na quinta etapa, todos os dados necessários são enviados para a GPU e todo o mapa de alturas é processado em paralelo por um *compute shader*. Cada posição do mapa de alturas é amostrada e subtraída da altura da curva de Bézier que passa sobre ela — ou dentro do alcance. Os valores de diferença são armazenados em um mapa de deslocamentos e podem ser adicionados aos valores do mapa de altura original para obter a altura final do terreno em cada posição. Veja o Capítulo 7.

5 GERAÇÃO DAS CURVAS DE BÉZIER

Conforme mencionado anteriormente, os dados vetoriais de entrada são listas de vértices que representam polilinhas. Este tipo de representação de dados normalmente requer muitos segmentos de linhas — resultando em mais vértices e segmentos de linha menores — para descrever caminhos mais suaves (Figura 5.1). Portanto, para facilitar a parametrização e edição em tempo real, e para tornar os caminhos mais suaves, nosso método emprega curvas de Bézier cúbicas 3D geradas a partir dos dados vetoriais de entrada. Um aspecto essencial das curvas de Bézier é que elas interpolam os pontos de controle nas extremidades, algo importante em alguns tipos de aplicações, como simuladores virtuais militares que empregam navegação automática.

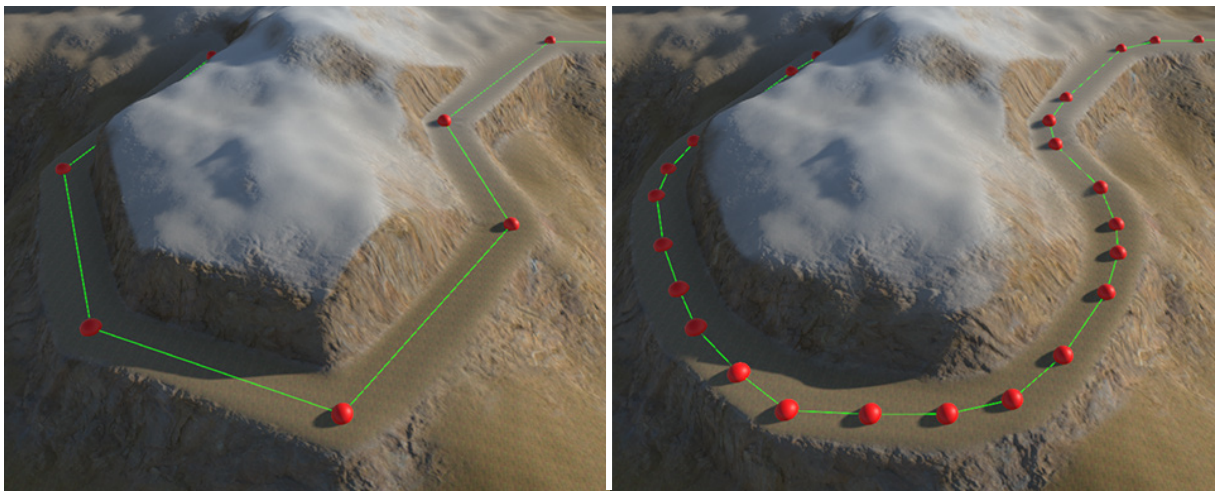


Figura 5.1 – Comparação entre resoluções de uma polilinha. Resolução mais baixa à esquerda; resolução mais alta à direita. (NASCIMENTO; POZZER; FRANZIN, 2019)

Escolhemos empregar curvas cúbicas porque elas oferecem um bom equilíbrio entre controle de edição e custo computacional, e são comumente usadas em muitos tipos de aplicações de design gráfico e modelagem. Além disso, esse tipo de curva pode ser gerado de forma a garantir continuidade C^2 entre curvas adjacentes, sem mudanças drásticas no traçado original do caminho.

5.1 CONVERSÃO DE POLILINHAS

Cada segmento de linha de uma polilinha é convertido em uma curva cúbica. Os dois vértices do segmento de linha tornam-se o primeiro e o último pontos de controle da curva — doravante referidos como âncoras —, e os dois pontos de controle intermediários — doravante referidos como alças — são derivados com base na direção e no comprimento

dos segmentos de linha adjacentes e um fator de suavização Ω. A Figura 5.2 ilustra — em 2D, para melhor visualização — a criação de curvas de Bézier a partir de uma polilinha.

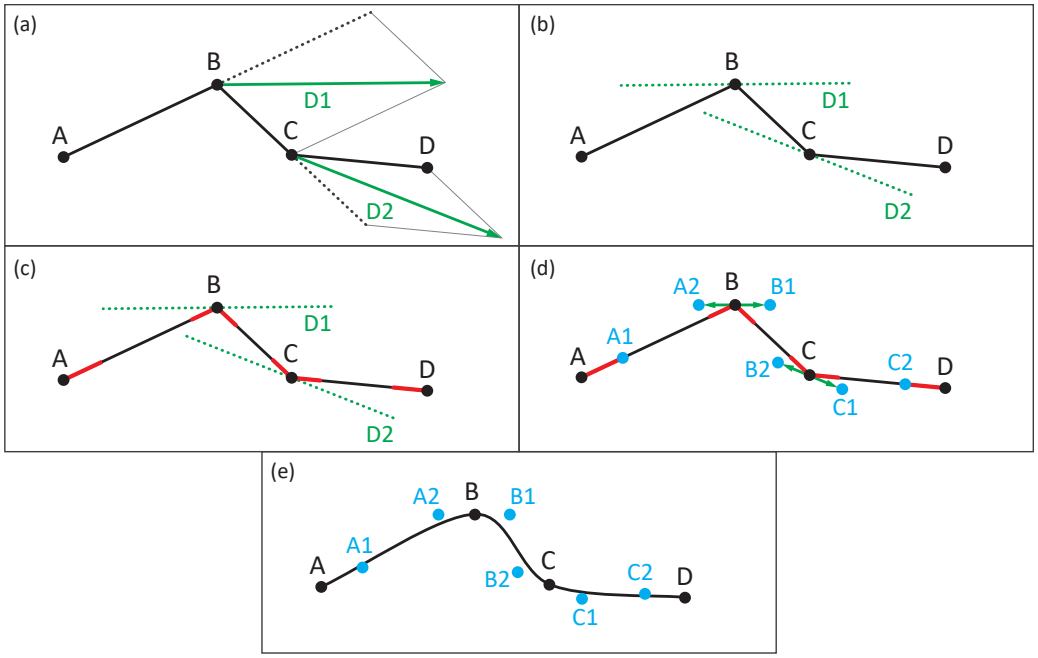


Figura 5.2 – Geração de curvas de Bézier cúbicas a partir de polilinhas. Os pontos pretos são âncoras e os pontos azuis são alças. Os segmentos de linha vermelhos ilustram as distâncias das âncoras nas quais as alças serão colocadas quando $\Omega = 0,5$.

A Figura 5.2.a mostra uma polilinha de entrada, com três segmentos de linha e quatro vértices (A, B, C, D) que se tornarão as âncoras. Os segmentos de linha são tratados como vetores para que possamos realizar operações de soma e multiplicação por um escalar. Para cada âncora compartilhada (B, C), determinamos os vetores de direção $\vec{D1}$ e $\vec{D2}$, nos quais as alças serão colocadas, somamos os dois segmentos de linha adjacentes e normalizamos o resultado (Figura 5.2.b). Essas direções são o que garante a continuidade tangencial.

Depois, para cada âncora, utilizamos metade do comprimento do segmento de linha adjacente mais curto e multiplicamos por um fator de suavização Ω, com $\Omega \in [0, 1]$ (Figura 5.2.c, em vermelho). O resultado é a distância da âncora até as alças adjacentes, e é o que garante a continuidade do C^2.

A seguir, a posição das alças ($A1$, $A2$, $B1$, $B2$, $C1$ e $C2$, na Figura 5.2.d) é determinada multiplicando as direções $\vec{D1}$ e $\vec{D2}$ pelas distâncias calculadas no passo anterior. As alças adjacentes às âncoras nas extremidades ($A1$, $C2$) são projetadas sobre os seus segmentos de linha ($\overline{AB}$, $\overline{CD}$).

Enfim, temos uma sequência de curvas de Bézier cúbicas que representam um caminho. O caminho gerado na Figura 5.2.e consiste em dez pontos de controle e três curvas de Bézier.

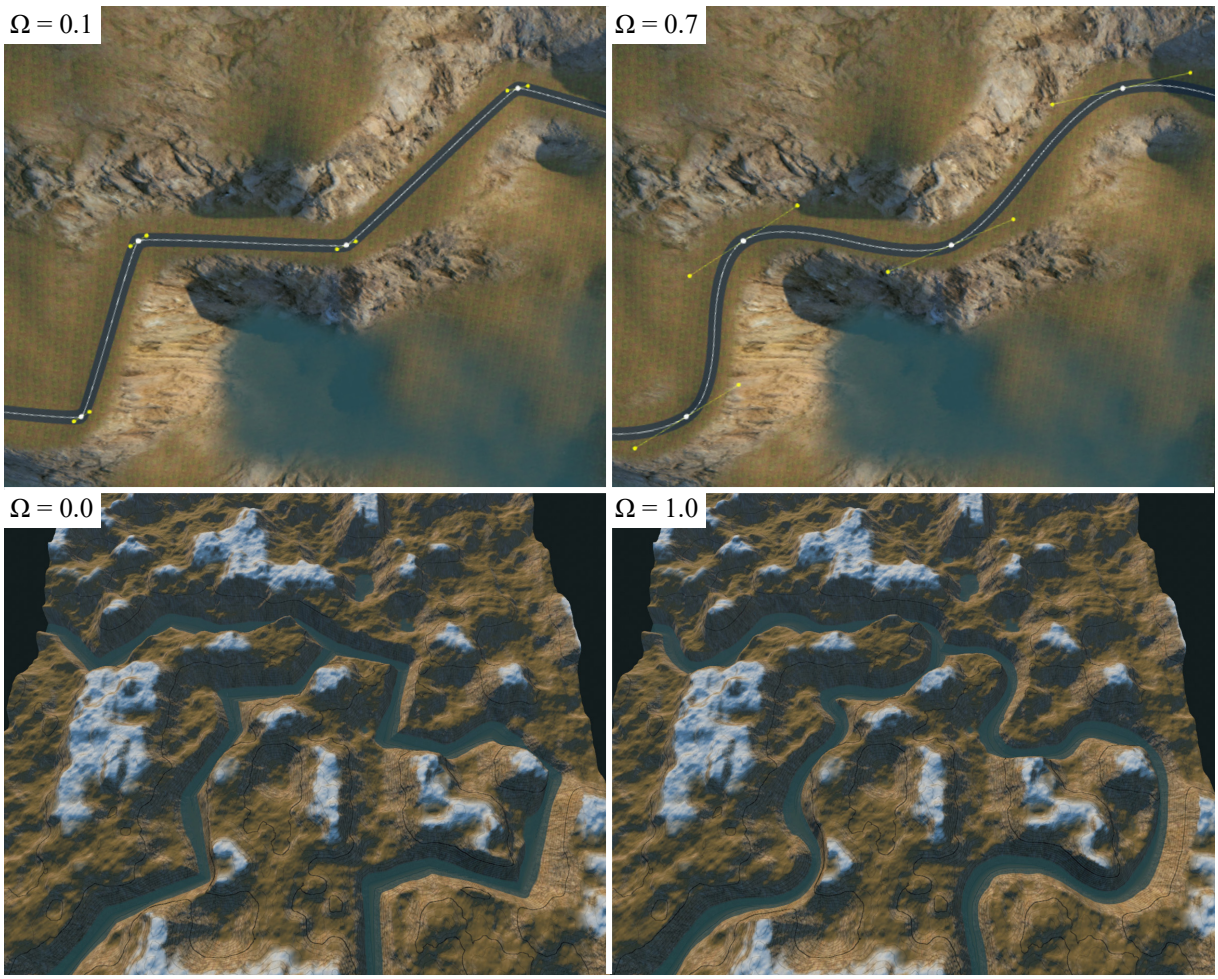


Figura 5.3 – O fator de suavização Ω é usado para reduzir cantos agudos. Na linha superior, as âncoras são brancas e as alças são amarelas.

5.2 REDUÇÃO DE GRAU

Para determinar o valor de deslocamento para cada posição do terreno, primeiro devemos encontrar o ponto mais próximo sobre o caminho. A projeção de pontos em curvas de Bézier é um tópico de pesquisa ainda em aberto e com várias abordagens numéricas iterativas propostas (CHEN et al., 2007), embora a maioria não seja adequada para execução em GPU.

Não há uma solução analítica para esse problema relacionado a curvas de Bézier cúbicas, e métodos numéricos iterativos, em geral, não apresentam um bom desempenho em GPU. Assim, propomos converter as curvas cúbicas para quadráticas — que possuem uma solução analítica — e apresentar um método para realizar essa conversão.

Em geral, uma curva cúbica não pode ser representada exatamente por uma quadrática, então o processo de conversão resulta em uma curva ligeiramente diferente. Em última análise, observamos que a melhora do desempenho obtida com essa abordagem justifica a pequena perda de fidelidade ao traçado original da curva.

Uma das principais diferenças entre uma curva de Bézier cúbica e quadrática é que

a primeira pode mudar sua direção de curvatura dentro do seu intervalo paramétrico, ao contrário da segunda. Em outras palavras, as curvas cúbicas podem ter pontos de inflexão. Uma curva cúbica 3D pode ter até seis pontos de inflexão.

O primeiro passo é projetar a curva 3D no plano XZ, simplesmente removendo o componente Y de cada ponto de controle — neste trabalho, assumimos que o componente Y representa a altura do ponto (ou seja, o eixo Y aponta para cima). Isso reduz a curva para uma curva cúbica planar com dois pontos de inflexão no máximo. Além disso, nosso método para converter polilinhas para curvas (Seção 5.1) garante que as curvas não tenham laços e pontas (a menos que o valor Ω usado extrapole o intervalo definido). Portanto, o número possível de pontos de inflexão cai para um, no máximo.

Uma curva de Bézier cúbica 2D C, com $C(t) = (C_x(t), C_y(t))$, definida por quatro pontos de controle — P_0, P_1, P_2 e P_3 — tem uma equação paramétrica na forma:

$$C(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \tag{5.1}$$

Seus pontos de inflexão são definidos pelas soluções da Eq. (5.2) dentro do intervalo $t \in (0, 1)$.

$$C'(t) \times C''(t) = 0 \tag{5.2}$$

$$C'_x(t)C''_y(t) - C'_y(t)C''_x(t) = 0 \tag{5.3}$$

Agrupando os coeficientes na Eq. (5.1), temos:

$$C(t) = at^3 + bt^2 + ct + d \tag{5.4}$$

$$C'(t) = 3at^2 + 2bt + c \tag{5.5}$$

$$C''(t) = 6at + 2b \tag{5.6}$$

onde:

$$a = -P_0 + 3P_1 - 3P_2 + P_3 \tag{5.7}$$

$$b = 3P_0 - 6P_1 + 3P2 \tag{5.8}$$

$$c = -3P_0 + 3P_1 \tag{5.9}$$

$$d = P_0 \tag{5.10}$$

Substituindo (5.5) e (5.6) na Eq. (5.3) e reorganizando os termos, temos:

$$3(b \times a)t^2 + 3(c \times a)t + (c \times b) = 0 \tag{5.11}$$

A Eq. (5.11) é de segundo grau e podemos encontrar suas raízes usando a fórmula de Bhaskara. Estamos interessados apenas nas raízes reais que se encontram no intervalo $t \in (0, 1)$. Inserindo as raízes na Eq. (5.4), obtemos os pontos de inflexão de C.

O segundo passo é dividir a curva em seu ponto de inflexão usando o algoritmo de De Casteljau (DUNN; PARBERRY et al., 2010). Se a curva não tem inflexão, nós a dividimos em seu ponto médio ($t = 0,5$).

Teoricamente, quanto mais subdividimos uma curva, melhor será sua aproximação quadrática. Porém, observamos que dividir as curvas em duas é suficiente para produzir resultados aceitáveis e, também, evita que o uso de memória aumente drasticamente, já que cada divisão cria três pontos de controle novos. Além disso, mais divisões significa mais curvas para serem avaliadas, o que afeta o desempenho.

A terceira etapa combina as duas alças de cada curva em uma só. Para cada curva cúbica com quatro pontos de controle, duas âncoras A e D, e duas alças B e C, calculamos a interseção dos vetores $\vec{AB}$ e $\vec{CD}$. O ponto de interseção (E) será a única alça de uma nova curva de Bézier quadrática 2D definida pelos pontos de controle A, E e D. Isso garante continuidade C^1 no plano XZ.

Por fim, para transformar as curvas quadráticas de 2D para 3D, devemos adicionar de volta o componente Y de seus pontos de controle. Como as âncoras não devem mudar, simplesmente restauramos seus valores anteriores. Para a nova alça (E), definimos o valor de Y como sendo a média dos componentes Y das alças originais da curva cúbica (B e C).

6 HASH ESPACIAL

Ao calcular o deslocamento do terreno, o nosso método deve determinar a influência que os caminhos têm sobre todas as posições do terreno. Para fazer isso, é necessário calcular a distância de cada posição a todos os caminhos próximos. Para otimizar esse processo, usamos um *hash* espacial bidimensional para particionar o terreno e diminuir a quantidade de avaliações das curvas de Bézier, reduzindo assim o número de cálculos de distância. A tabela *hash* é armazenada em memória linear e emprega uma tabela de pivôs auxiliar. Sua construção é baseada na abordagem proposta por Pozzer et al. (2014). A Figura 6.1 mostra uma visão geral da estruturação dos dados.

6.1 ESTRUTURAÇÃO DOS DADOS

Cada célula *hash* (Figura 6.1.a) tem uma entrada na tabela de pivôs (Figura 6.1.b) que armazena quantas curvas existem na área da célula (ou seja, seu uso U), e o índice inicial I na tabela *hash*. A tabela *hash* (Figura 6.1.c), por sua vez, contém o índice do primeiro ponto de controle de cada curva contida em uma célula *hash*. Como todos os pontos de controle são colocados em sequência (Figura 6.1.d), precisamos indexar apenas o primeiro para avaliar uma curva na GPU.

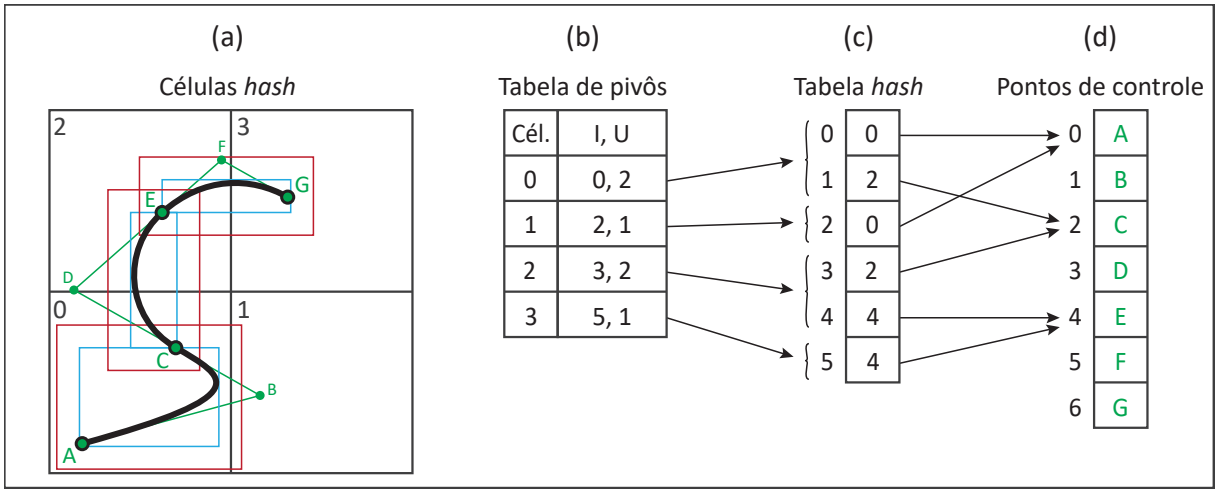


Figura 6.1 – *Hash* espacial usado para reduzir avaliações de curvas. *Hash* com dimensão 2, tendo $2 \times 2 = 4$ células. Caixas delimitadoras originais (em azul) e expandidas (em vermelho) em (a).

6.2 CONSTRUÇÃO DO *HASH* ESPACIAL

O *hash* é construído depois que as curvas são convertidas de cúbicas para quadráticas. Para determinar quais curvas estão contidas em uma célula *hash*, avaliamos se há uma sobreposição entre a caixa delimitadora da curva e a célula (Figura 6.2). Para evitar que curvas próximas sejam ignoradas incorretamente ao realizar o deslocamento, expandimos a caixa delimitadora da curva o suficiente para cobrir sua área de influência. Por exemplo, se um caminho pode influenciar a altura do terreno a uma distância de até 10 unidades, expandimos a caixa delimitadora de cada curva nesse caminho em 10 unidades de cada lado antes de verificar se ela sobrepõe uma célula *hash*. Isso garante que as células *hash* irão conter todas as curvas que influenciam suas áreas.

Além disso, o *hash* é dimensionado — utilizando as caixas delimitadoras das curvas — de forma que cubra a mínima área possível do terreno, evitando que áreas "vazias" sejam cobertas sem necessidade. Na Figura 9.1.b, por exemplo, o *hash* é criado de forma que cubra apenas a região do canto inferior direito do terreno onde as curvas estão concentradas.

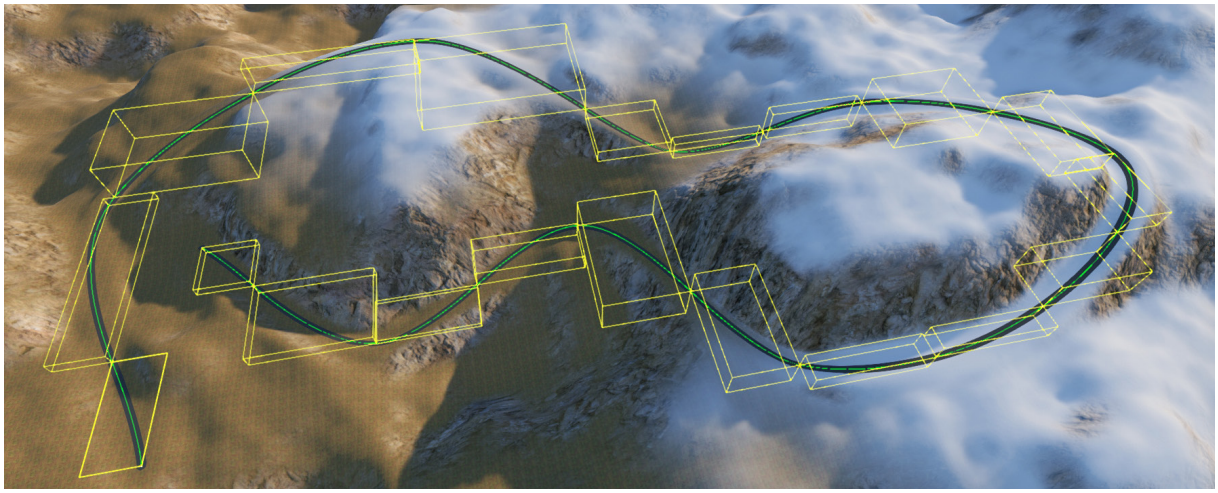


Figura 6.2 – As caixas delimitadoras (em amarelo) das curvas são usadas para criar o *hash* espacial.

7 TERRAPLENAGEM DO TERRENO

O objetivo principal das curvas de Bézier é descrever com precisão a altura do terreno ao longo de um caminho (Figura 7.1). Se uma posição estiver inteiramente em um caminho, sua altura será igual à curva naquele ponto. Se uma posição estiver em algum lugar na área de influência de vários caminhos, sua altura será uma média ponderada das curvas mais próximas, com base em suas distâncias.

Depois que a altura de uma posição é determinada, ela é subtraída do valor do mapa de altura e o resultado é armazenado em um Mapa de deslocamento. Então, ao renderizar o terreno, adicionamos o mapa de altura e os valores de deslocamento para obter a altura final do terreno.

Figura 7.1 – Terreno sem deslocamento (esquerda) e deslocamento total (direita) ao longo de um caminho.

Alternativamente, a altura da curva pode ser cozida diretamente no mapa de altura, sem a necessidade do Mapa de deslocamento. Notamos que é melhor manter o mapa de altura original inalterado, para permitir a edição dos caminhos em tempo real. No entanto, dependendo do propósito, qualquer abordagem pode ser empregada.

Para calcular o deslocamento, a primeira etapa é enviar todos os dados relevantes — pontos de controle, tabela de pivôs, tabela *hash* e parâmetros de terraplenagem — para a GPU. Em seguida, executamos um *compute shader* para processar cada posição do mapa de alturas (i.e., *texel*) em paralelo e armazenar o resultado no mapa de deslocamentos.

Quando esculpimos um caminho no terreno, devemos considerar a largura w do caminho. Além disso, para criar escavações e taludes, é importante aplicar uma distância de suavização λ para evitar transições bruscas e declives acentuados (Figura 7.2).

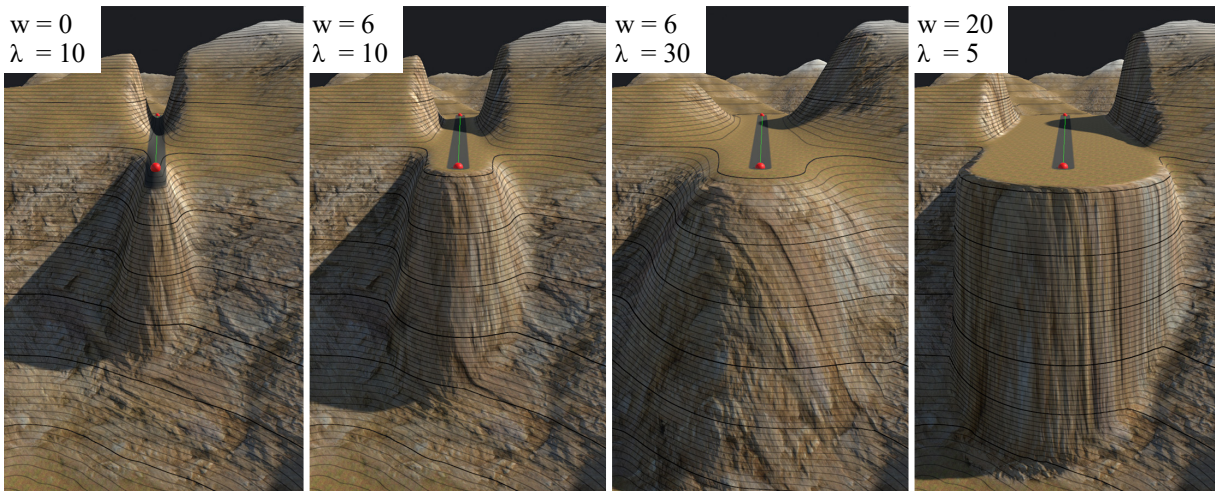


Figura 7.2 – Escavações e taludes criados com valores diferentes para w e λ. Adaptado de Nascimento, Pozzer e Franzin (2019).

7.1 CÁLCULO DE DISTÂNCIA

Para calcular a distância de um ponto P a uma curva quadrática C, com $C(t) = (C_x(t), C_y(t), C_z(t))$, precisamos determinar a projeção de P em C, ou seja, precisamos encontrar o valor de t associado a um ponto Q para o qual $\|QP\|$ é mínimo.

Uma curva quadrática de Bézier definida por três pontos de controle — P_0, P_1, e P_2 — tem uma equação paramétrica na forma:

$$C(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2 \tag{7.1}$$

Subtraindo P da Eq. (7.1), nós obtemos uma equação que representa cada vetor de C a P:

$$V(t) = at^2 + bt + c \tag{7.2}$$

onde:

$$a = P_0 - 2P_1 + P_2 \tag{7.3}$$

$$b = -2P_0 + 2P_1 \tag{7.4}$$

$$c = P_0 - P \tag{7.5}$$

A magnitude dos vetores descritos pela Eq. (7.2) é:

$$\|V(t)\| = \sqrt{V_x(t)^2 + V_y(t)^2 + V_z(t)^2} \tag{7.6}$$

Para encontrar o menor vetor, precisamos apenas da expressão sob a raiz quadrada na Eq. (7.6).

A Eq. (7.7) descreve o quadrado da distância de cada ponto em C até P. Estamos interessados apenas nos mínimos locais no intervalo $t \in [0, 1]$. A Eq. (7.7) tem o seu mínimo local onde $D'(t) = 0$ e $D''(t) \geq 0$.

$$D(t) = V_x(t)^2 + V_y(t)^2 + V_z(t)^2 \tag{7.7}$$

Substituindo (7.3), (7.4), e (7.5) na Eq. (7.7), derivando uma vez, e reorganizando os termos, temos:

$$D'(t) = 4(a \cdot a)t^3 + 3(a \cdot b)t^2 + 2((b \cdot b) + (a \cdot c))t + (b \cdot c) \tag{7.8}$$

Podemos encontrar as raízes da equação cúbica Eq. (7.8) analiticamente usando a fórmula de Cardano, que pode render até três raízes reais. Queremos a raiz no intervalo $t \in [0, 1]$ que produz o menor resultado na Eq. (7.6). Essa será a distância mais curta de P a C. Entrando com a raiz na Eq. (7.1) obtemos a projeção de P em C.

7.2 INCLINAÇÃO LATERAL

Para definir a inclinação lateral de um caminho, associamos, a cada ponto de controle, um valor de inclinação I. Este valor representa a tangente do ângulo α entre o vetor normal do caminho e o eixo Y (Figura 7.3). Geramos procedimentalmente I calculando a curvatura 2D do caminho em $t = 0$, $t = 0,5$ e $t = 1$. Para fazer isso, primeiro projetamos a curva C no plano XZ e, em seguida, usamos a Eq. (7.9) para determinar I para cada ponto de controle. Como as curvas adjacentes podem ter curvaturas diferentes em seus pontos de ancoragem compartilhados, usamos a média das curvaturas nesses pontos. Este processo é executado na CPU antes que os pontos de controle sejam enviados à GPU para deslocamento.

$$I(t) = \frac{C'(t) \times C''(t)}{\|C'(t)\|^3} \tag{7.9}$$

$$\alpha(t) = \arctan(I(t)) \tag{7.10}$$

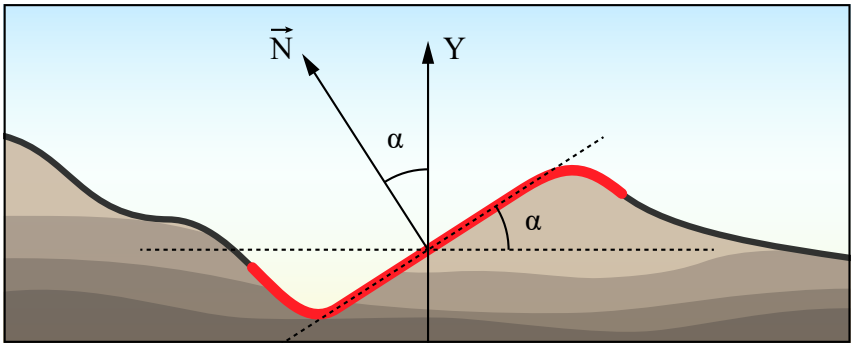


Figura 7.3 – Seção transversal de um caminho inclinado. A valor α representa o ângulo entre o plano do caminho e o plano horizontal. A linha vermelha indica a área do terreno influenciada pelo caminho.

7.3 INTENSIDADE DO DESLOCAMENTO

Para modular a intensidade Θ do deslocamento impresso em uma posição P do terreno por um caminho, primeiro, precisamos determinar a distância 2D d de P para esse caminho (Seção 7.1). Então, usando a Eq. (7.11), calculamos o parâmetro linear Ψ que produz o valor interpolante d, com $d \in [w/2, (w/2 + \lambda)]$ (i.e., interpolação linear inversa). A Figura 7.4.a mostra como Ψ (linha verde) muda ao longo da área de influência do caminho.

$$\Psi = \frac{d - (w/2 - \lambda)}{-\lambda} \tag{7.11}$$

Para calcular Θ, empregamos a função *smoothstep* de grau 5 (Eq. (7.12)) proposta por Perlin (2002), que tem derivadas de primeira e segunda ordem nulas em $x = 0$ e $x = 1$. Isso contribui para uma mesclagem mais suave dos deslocamentos ao terreno. A Figura 7.4.b mostra que Θ é máximo ao longo de w e diminui gradualmente ao longo de λ.

$$\Theta = 6\Psi^5 - 15\Psi^4 + 10\Psi^3 \tag{7.12}$$

7.4 CÁLCULO DO DESLOCAMENTO

Para realizar os cálculos de deslocamento em paralelo na GPU, executamos, para cada *texel* do mapa de alturas, uma *thread* que executa um *compute shader*. Primeiramente, cada *thread* converte a posição do *texel* para uma posição de mundo P e a usa para determinar a célula *hash* correta e acessar as tabelas de pivôs e *hash*.

Para cada curva quadrática C referenciada pela tabela *hash*, calculamos a projeção Q de P em C (Seção 7.1). Em seguida, projetamos o vetor $\vec{PQ}$ no plano XZ e usamos sua magnitude para calcular Θ. Então, a altura do terreno é amostrada do mapa de alturas e subtraída de Q altura para produzir um valor de deslocamento bruto Δh. Finalmente, Δh

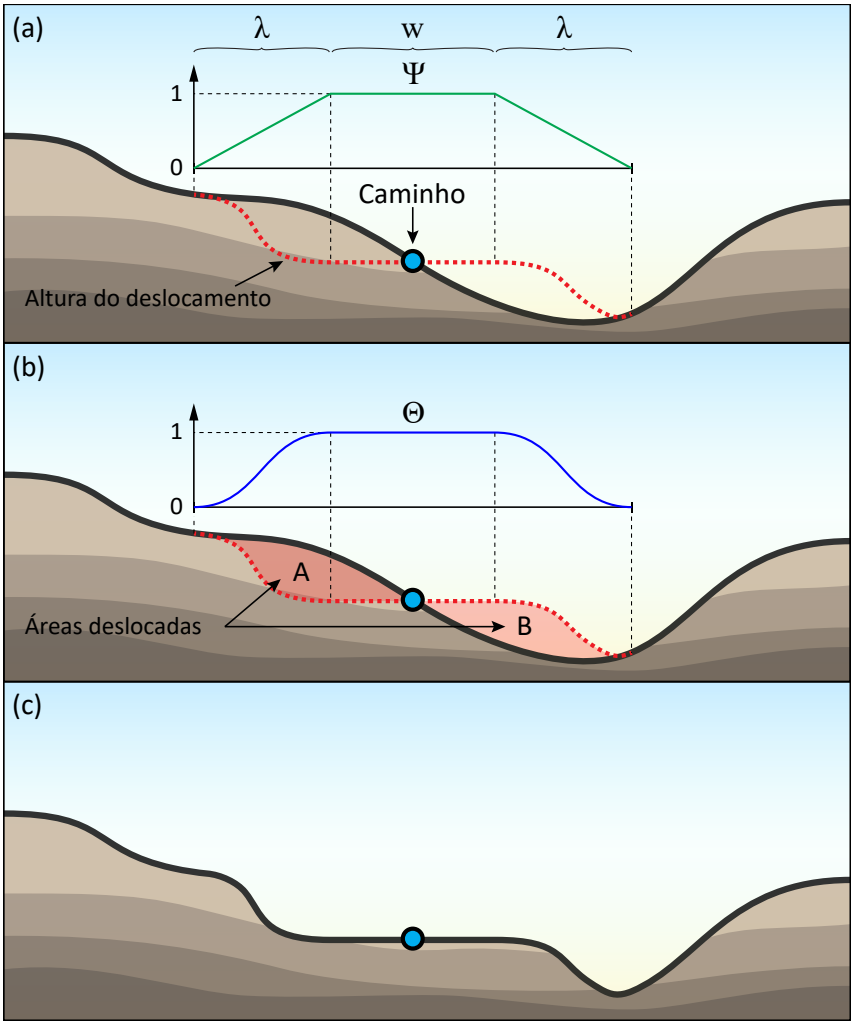


Figura 7.4 – Seção transversal do terreno mostrando a área de influência de um caminho, definida pela largura w e a distância de suavização λ. A linha vermelha pontilhada em (a) indica a altura do terreno após o deslocamento. Em (b), A e B são as áreas que serão escavadas e preenchidas, respectivamente. O perfil do terreno após o deslocamento é visto em (c). Adaptado de Nascimento, Pozzer e Franzin (2019).

é multiplicado por Θ, resultando no valor de deslocamento final, que é então armazenado no mapa de deslocamentos.

8 ASPECTOS DE IMPLEMENTAÇÃO

Para validar a nossa técnica, implementamos um protótipo de aplicação (Figura 8.1) que permite ao usuário criar e remover caminhos interativamente e observar o terreno virtual sendo modificado em tempo real para encorporar o traçado dos caminhos. A aplicação possibilita, ainda, que o usuário possa alterar os parâmetros de terraplenagem, inclinação, além de diversas outras opções que controlam a renderização das curvas de Bézier e do terreno virtual, embora os aspectos de renderização não sejam o foco deste trabalho. Os binários da aplicação são anexos a este trabalho e também estão disponíveis para download [1].

Figura 8.1 – Captura de tela do protótipo implementado para validar a técnica proposta. A aplicação permite ao usuário inserir e remover caminhos e modificar os parâmetros de terraplenagem.

A aplicação implementada apresenta um renderizador de terreno virtual que emprega um *vertex shader* para deslocar os vértices da malha de grade regular usando, como entrada, um mapa de alturas gerado com *Perlin Noise* (PERLIN, 1985) e *fractional Brownian motion* (fBm) (EBERT et al., 2003). Ainda, o usuário pode instanciar um plano d'água para simular a criação de rios.

Considerando a natureza eminentemente gráfica desta aplicação, optamos por utilizar a plataforma Unity como base para a sua implementação. A Unity é uma plataforma para desenvolvimento de jogos já consolidada no mercado e que oferece suporte a uma

[1] <http://www.inf.ufsm.br/~pozzer/path_carving_demo.rar>

variedade de tecnologias utilizadas no desenvolvimento de aplicações gráficas. Em especial, o suporte a *compute shaders* permite a execução de programas diretamente na GPU sem estarem atrelados diretamente ao *pipeline* gráfico normal (Unity Technologies, 2020).

A computação dos deslocamentos do terreno só precisa ser executada uma vez para gerar o mapa de deslocamentos. Uma vez que o mapa está pronto, basta que o seu valor seja somado ao mapa de alturas para que se obtenha a altura final dos vértices da malha do terreno. Ou seja, não é necessário que os cálculos que envolvem a terraplenagem sejam executados continuamente, a cada *frame*, dentro do *pipeline* de renderização. Por isso, optamos por empregar um *compute shader* para computar os deslocamento sob demanda em um estágio separado do *pipeline* de renderização normal.

As etapas de conversão de polilinhas para curvas de Bézier (Seção 5.1), redução de grau das curvas (Seção 5.2) e criação das estruturas de *hash* espacial (Seção 6.2) foram implementadas em C# e são executadas na CPU. As etapas de computação dos deslocamentos e geração do mapa de deslocamentos são executadas, em GPU, por um *compute shader* escrito em *High Level Shading Language* (HLSL) para DirectX 11. Os Apêndices A e B apresentam, respectivamente, o código-fonte do *compute shader* usado para gerar o mapa de deslocamentos e as funções que propriamente amostram as curvas de Bézier e calculam os valores de deslocamento.

As linhas 17 a 19 do Apêndice B mostram a declaração dos *buffers* de dados utilizados para armazenar as estruturas de dados do *hash* espacial (Seção 6.1) e pontos de controle das curvas de Bézier em GPU. A implementação da função que calcula analiticamente a distância de ponto à curva é baseada na fórmula de Cardano (WEISSTEIN, 2002) e pode ser vista nas linha 80 a 124 do Apêndice B.

9 RESULTADOS E DISCUSSÃO

Todos os experimentos e medições foram realizados em um processador Intel Core i7-4790 3,5 GHz, com 24 GB de RAM DDR3 e placa de vídeo NVIDIA GeForce GTX 1070, com 8 GB de VRAM DDR5 e 1.920 núcleos CUDA.

9.1 ANÁLISE DE DESEMPENHO

Realizamos várias medições para avaliar o desempenho de nossa abordagem durante a calculação dos valores de deslocamento do terreno e geração do mapa de deslocamentos em GPU. As medições foram feitas utilizando-se três conjuntos de dados vetoriais de entrada (Figura 9.1). A conversão das polilinhas de entrada para curvas de Bézier e a geração do *hash* espacial são executadas em CPU e não foram abrangidas pelas medições realizadas. O tamanho do terreno considerado foi de 2048×2048, em espaço de mundo. Ainda, os parâmetros de terraplenagem utilizados foram largura $w = 15$ e distância de suavização $\lambda = 15$.

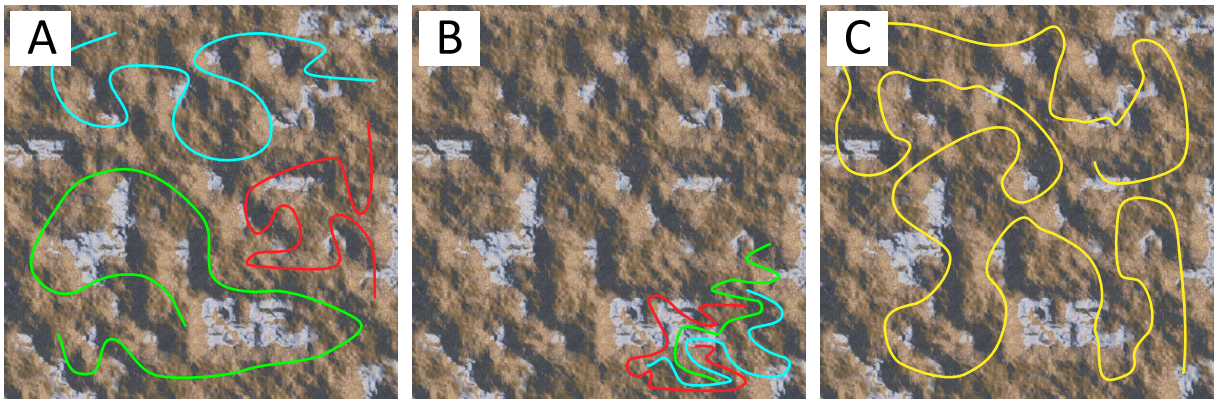


Figura 9.1 – Conjuntos de dados com características diferentes utilizados nas medições de desempenho.

Para fornecer a base para uma análise comparativa, implementamos também abordagens que usam métodos iterativos para calcular as distâncias de ponto à curva — para curvas de Bézier cúbicas e quadráticas. Esses métodos consistem em iterar sobre as curvas subdividindo e aproximando-as por segmentos de linha e calculando a projeção de ponto em linha para cada segmento.

O principal aprimoramento da nossa técnica com respeito ao desempenho está no emprego de um método analítico para calcular as distâncias de ponto à curva. Para quantificar essa melhoria, também comparamos o desempenho da abordagem anterior (NASCIMENTO; POZZER; FRANZIN, 2019) desenvolvida na fase inicial de pesquisa deste

trabalho, a qual empregava um método iterativo usando curvas cúbicas.

Para identificar melhor a contribuição de diferentes fatores para o desempenho do processo de geração do mapa de deslocamentos, realizamos as medições variando cada fator individualmente. Os fatores avaliados foram: a resolução dos mapas (de alturas e deslocamentos); o número de iterações (para os métodos iterativos); as dimensões do *hash*; e o tamanho dos grupos de *threads* do *compute shader*.

A Figura 9.2 apresenta os tempos de processamento para o cálculo de todas as posições em um terreno com diferentes resoluções dos mapas. Como todos os mapas são quadrados, uma resolução de 2048 significa $2048 \times 2048 = 4.194.304$ *texels* (i.e., posições). Essas medições utilizaram o conjunto de dados C (Figura 9.1.c), o qual contém um caminho (em amarelo) com com 198 curvas de Bézier quadráticas que se estendem por toda a área do terreno.

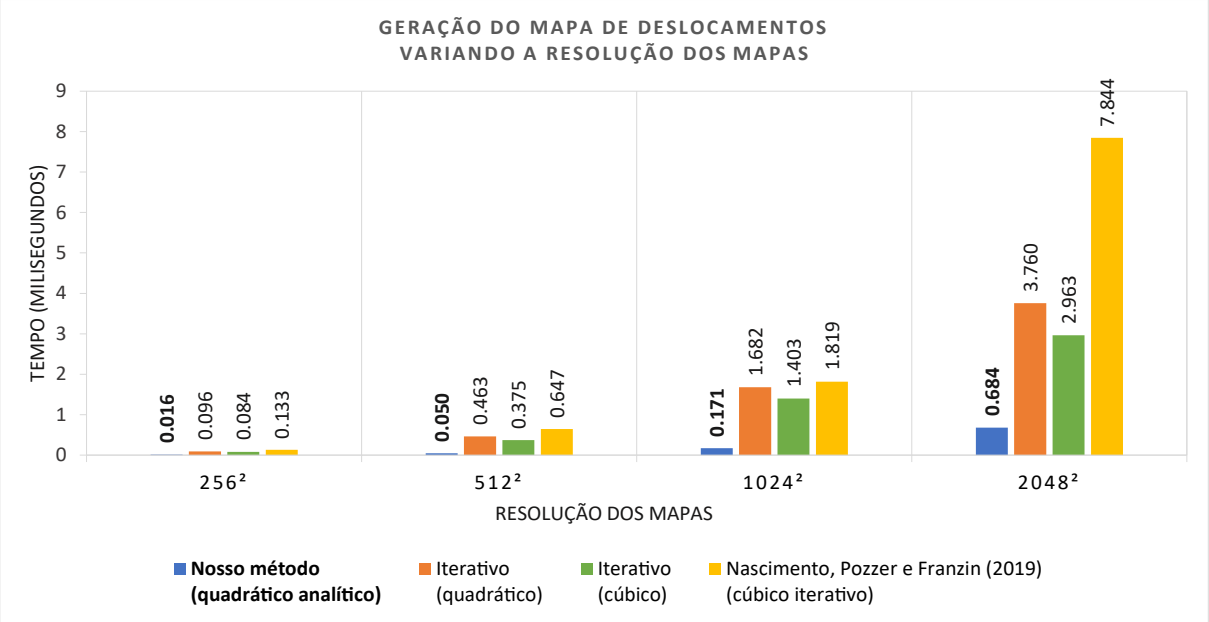


Figura 9.2 – O nosso método apresenta um desempenho superior nos cenários testados, e se mostra menos sensível ao aumento da resolução dos mapas. **Linha de base**: dimensões do *hash* = 16^2; número de iterações = 32.

Conforme esperado, analisando os resultados no gráfico (Figura 9.2), verificamos que a resolução dos mapas influencia fortemente os tempos de processamento dos deslocamentos, uma vez que determina o número de posições processadas. Embora todas as medições apresentadas neste capítulo tenham sido realizadas utilizando mapas que compartilham a mesma resolução, vale ressaltar que isso não é um requisito da nossa técnica. Nestes testes, o nosso método obteve um desempenho 4 a 8 vezes mais rápido do que as abordagens iterativas, e 8 a 12 vezes mais rápido do que a abordagem anterior (NASCIMENTO; POZZER; FRANZIN, 2019).

A qualidade visual dos resultados alcançados pelos métodos iterativos está diretamente ligada ao número de iterações realizadas. Mais iterações significa resultados

mais refinados, mas também afeta o desempenho. A Figura 9.3 mostra que os tempos de processamento aumentam linearmente com o número de iterações. Essas medições utilizaram o conjunto de dados C (Figura 9.1.c).

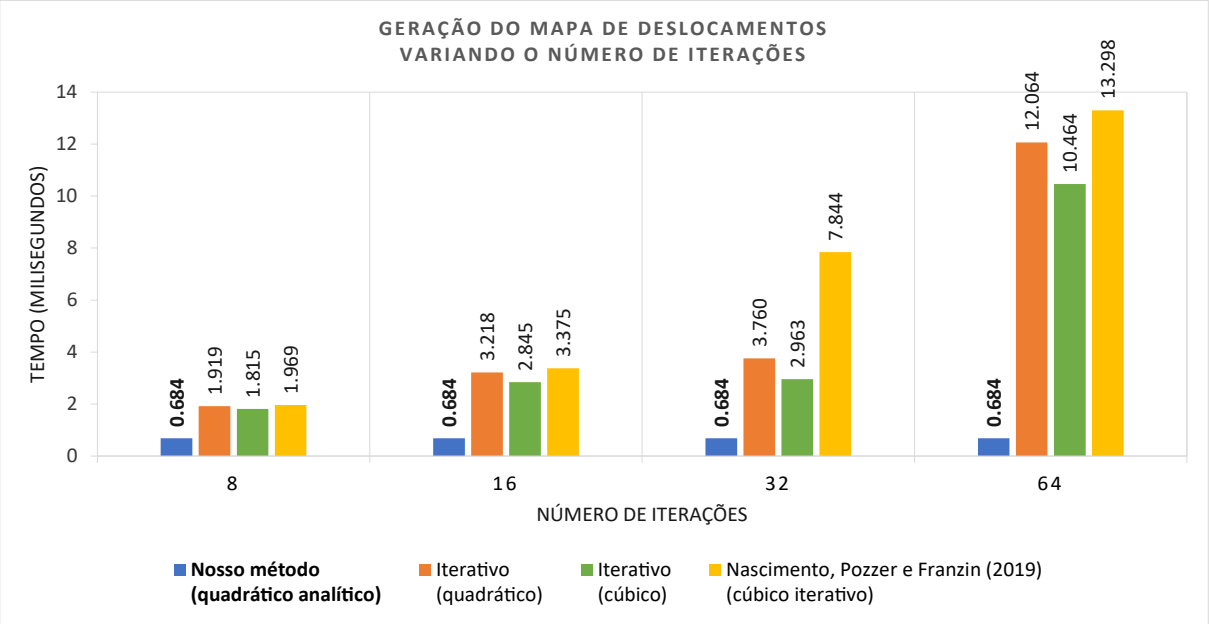


Figura 9.3 – Visto que nossa abordagem é analítica e não itera sobre as curvas, o seu tempo de processamento é mostrado aqui apenas para comparação. **Linha de base**: resolução dos mapas = 2048^2; dimensões do *hash* = 16^2.

Para avaliar o ganho de desempenho causado pelo uso de *hash* espacial, realizamos medições usando diferentes dimensões de *hash* e processando o conjunto de dados C (Figura 9.1.c). O *hash* é bidimensional, portanto, uma dimensão 16 significa $16 \times 16 = 256$ células de *hash*. Os resultados mostrados na Figura 9.4 atestam o impacto positivo do *hash*, o qual agilizou a geração do mapa de deslocamentos de 12 a 42 vezes, para nosso método, naquele cenário. A diferença de desempenho entre a abordagem quadrática iterativa (em laranja) e sua contraparte cúbica (em verde) deve-se principalmente ao fato de que a quadrática processa o dobro de curvas de Bézier.

Um aspecto importante a ser considerando quando se utiliza *compute shaders* é o tamanho dos grupos de *threads*. A versão *Shader Model 5* limita em 1.024 o máximo de *threads* em um grupo. *Threads* em um mesmo grupo seguem o modelo de execução *Single Instruction, Multiple Thread* (SIMT), e rodam executando as mesmas instruções em paralelo de forma sincronizada. Um desvantagem dessa abordagem é que a *thread* "mais lenta" define o tempo de execução de todo o grupo. Por exemplo, se um grupo de *threads* estiver processando uma determinada região do terreno com poucas, ou sem curvas, a execução do grupo será finalizada mais rapidamente. Entretanto, se apenas uma dessas *threads* estiver processando uma posição do terreno correspondente a uma célula *hash* que indexa muitas curvas (em comparação às demais células na região), a execução dessa *thread* irá atrasar todo o grupo.

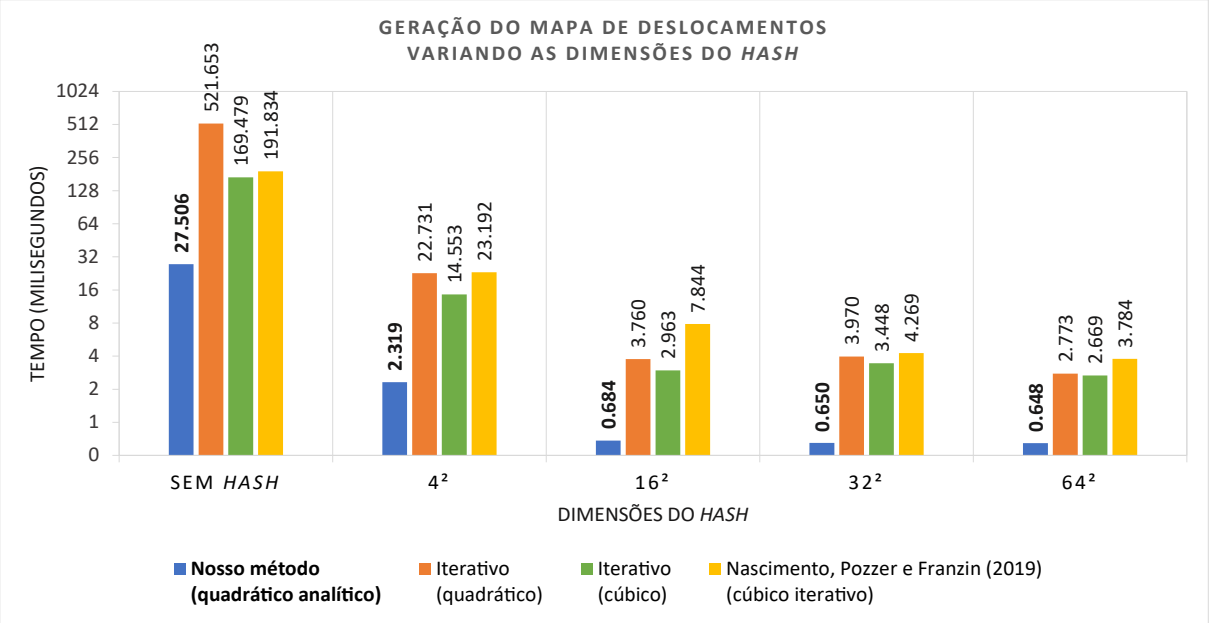


Figura 9.4 – Os tempos são mostrados em um escala logarítmica. Podemos observar que, para o nosso método, não houve um ganho significativo com o uso de um *hash* maior que 16^2. **Linha de base**: resolução dos mapas = 2048^2; número de iterações = 32.

Com o objetivo de tentar identificar a influência do tamanho dos grupos na geração do mapa de deslocamentos, realizamos uma série de medições que avaliam dois cenários de distribuição de curvas no terreno, considerando também a influência do *hash* espacial. O primeiro cenário considera um conjunto de caminhos distribuídos de forma dispersa sobre o terreno (conjunto A, Figura 9.1.a). O segundo cenário utiliza um conjunto que se concentra em um região pequena do terreno (conjunto B, Figura 9.1.b). É importante ressaltas que ambos os conjuntos de dados possuem exatamente a mesma quantidade de curvas de Bézier (3 caminhos, com um total de 84 curvas quadráticas). Apenas a sua distribuição sobre o terreno é que difere. A Figura 9.5 apresenta os resultado das medições realizadas sobre os conjuntos de dados A e B, variando-se o tamanho dos grupos de *threads* e dimensões do *hash* espacial.

Comparando-se os tempos de processamento dos dois conjuntos, verificamos que as curvas dispersas em uma região maior do terreno consumiram um tempo sensivelmente maior do que aquelas agrupadas em um região menor. Isso se deve principalmente a forma como o *hash* espacial é construído (Seção 6.2), em que ele se limita à área retangular mínima ocupada pelo conjunto. Isso faz com que as curvas agrupadas sejam indexadas de maneira mais eficaz pelo *hash*, já que, na prática, ambos os conjuntos (A e B) foram coberto pela mesma quantidade de células *hash*. Ainda, podemos observar que o aumento na dimensão do *hash* reduziu a diferença entre os tempos de processamento dos dois conjuntos.

Ainda, verificamos no gráfico que grupos de *threads* muito pequenos (2x2x1) apresentaram um desempenho significativamente pior do que os demais. Em contrapartida,

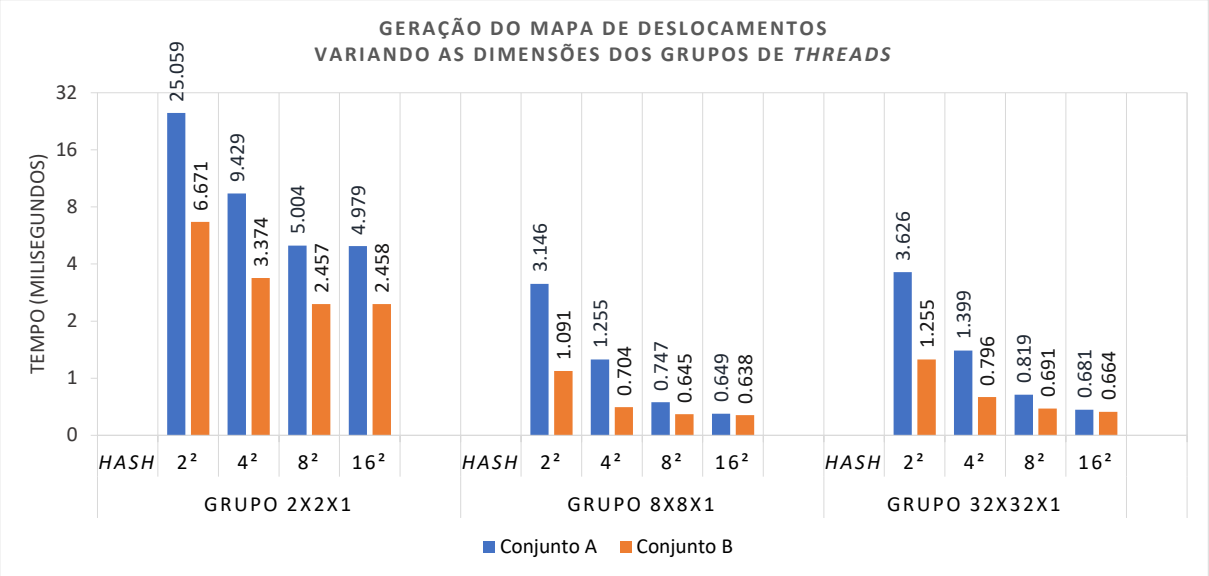


Figura 9.5 – O conjunto B, com curvas concentradas numa região pequena, foi processado mais rapidamente em todos os casos. **Linha de base**: resolução dos mapas = 2048^2.

vemos que aumentar indiscriminadamente o tamanho dos grupos não significa necessariamente um desempenho melhor, já que grupos muito maiores (32x32x1) apresentaram um leve perda de desempenho em comparação aos grupos médios (8x8x1).

9.2 ANÁLISE VISUAL

Como já dito, para avaliar os aspectos visuais de nossa abordagem, implementamos um renderizador de terreno virtual que emprega um *vertex shader* para deslocar os vértices de uma malha do terreno. Na Figura 9.6, podemos observar um mapa de deslocamentos criado a partir de um caminho e usado para esculpir o terreno. Nesse exemplo, ambos os mapas têm uma resolução de 512×512.

Usar um mapa de deslocamentos com uma resolução inferior a do mapa de alturas pode causar artefatos, a menos que a aplicação do mapa deslocamentos seja restrita a uma parte do mapa de alturas com as mesmas dimensões. Isso pode ser útil em uma situação em que apenas uma parte do terreno precisa ser deslocada. Ainda. pode-se criar um atlas de texturas de mapas de deslocamentos para deslocar regiões descontínuas do terreno.

A Figura 5.3 mostra uma estrada e um rio criados usando diferentes valores Ω. Valores mais baixos podem ser usados quando é necessário manter uma fidelidade mais alta à forma original do caminho de entrada. Porém, podemos observar que mesmo um valor máximo não altera drasticamente o caminho e as posições dos vértices originais nunca são alteradas.

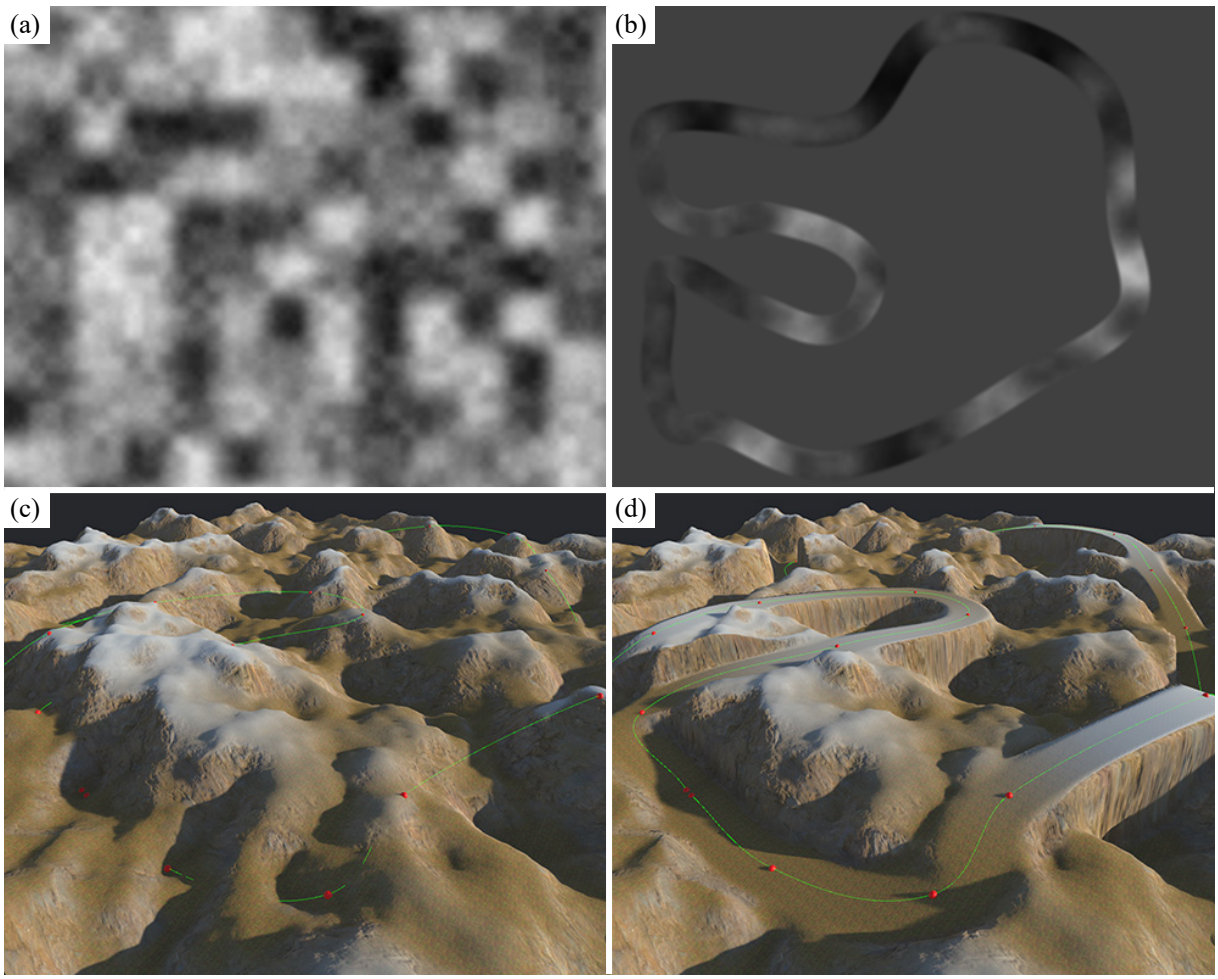


Figura 9.6 – Exemplo de um mapa de alturas de entrada (a), um mapa de deslocamentos (b) e um terreno renderizado sem (c) e com (d) o mapa de deslocamentos. Em (b), os deslocamentos positivos e negativos (ou seja, aterros e escavações) são representados pelos valores mais claros e mais escuros, respectivamente. Adaptado de Nascimento, Pozzer e Franzin (2019).

Os parâmetros de terraplenagem mudam o perfil do caminho e criam efeitos diferentes. A Figura 7.2 ilustra como w e λ são usados para modificar o perfil de um caminho. O perfil também é afetado pela inclinação I do caminho (Figura 9.7). As Figuras 9.8 e 9.9 destacam como o nossa técnica desloca corretamente o terreno, combinando suavemente os caminhos aos seus arredores.

Nossa abordagem, sendo analítica, não requer iterações e subdivisão ou discretização das curvas, e sempre produz resultados tão precisos quanto a resolução do mapa de alturas (Figura 9.10).

Nossa abordagem pode ser usada efetivamente para nivelar o terreno ao longo de um caminho, permitindo a renderização de estradas diretamente no terreno ou a colocação de modelos 3D, como uma ferrovia ou malha de água, conforme exemplificado na Figura 9.11. Veja o vídeo em anexo[1] para mais resultados.

[1] <https://youtu.be/p_NPN3WVPjk>

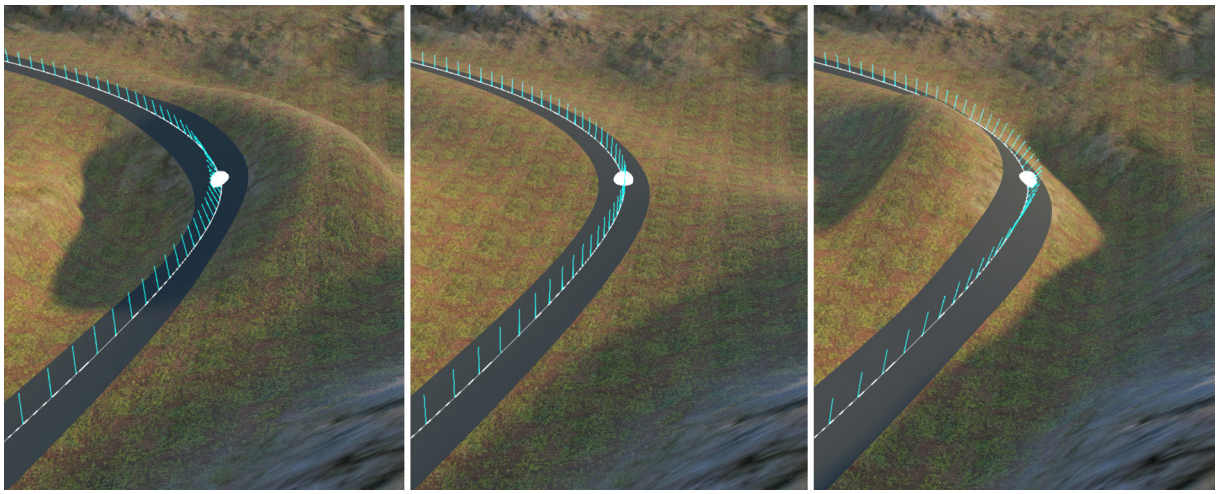


Figura 9.7 – Um trajeto de estrada com inclinações variadas em uma curva. Da esquerda para a direita, I é igual a 1.0, 0.0 e -1.0.

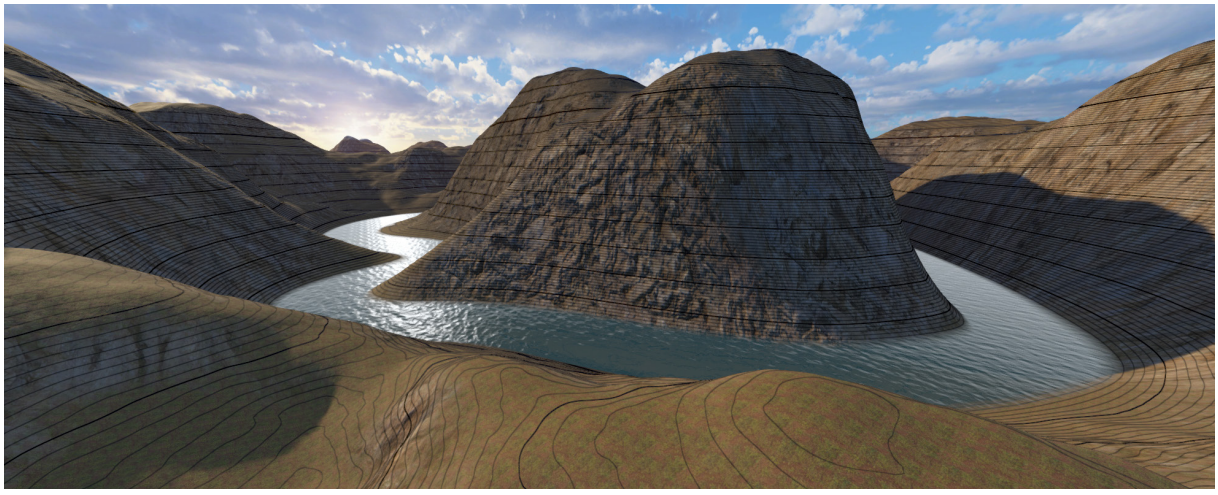


Figura 9.8 – Um caminho de rio escavado pela nossa técnica. Caminhos que passam por montanhas criam cânions quando esculpidos no terreno.

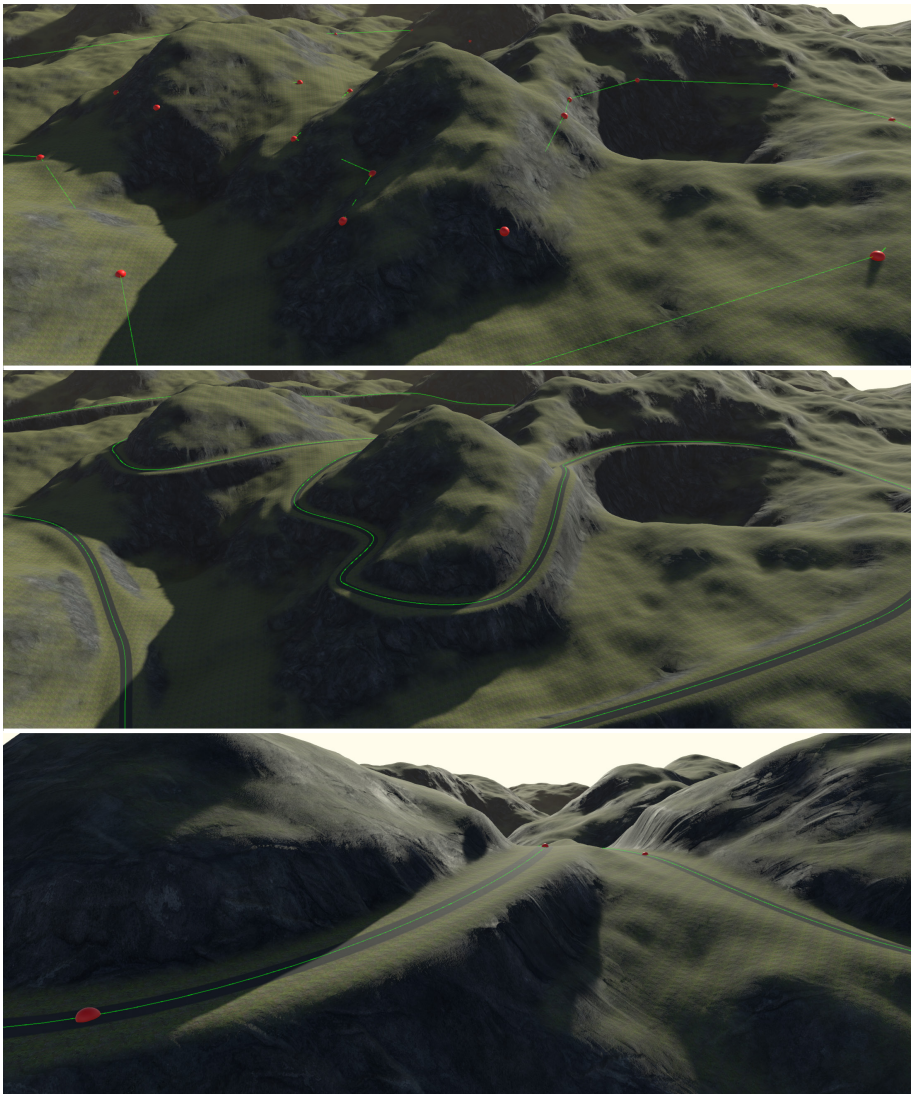


Figura 9.9 – O terreno original e a polilinha de entrada (cima). O caminho criado e esculpido no terreno (meio). Parte do caminho esculpido em detalhe (baixo). (NASCIMENTO; POZZER; FRANZIN, 2019)

Figura 9.10 – Comparação entre a nossa abordagem (esquerda) e uma abordagem iterativa (direita) utilizando 8 subdivisões por curva cúbica.

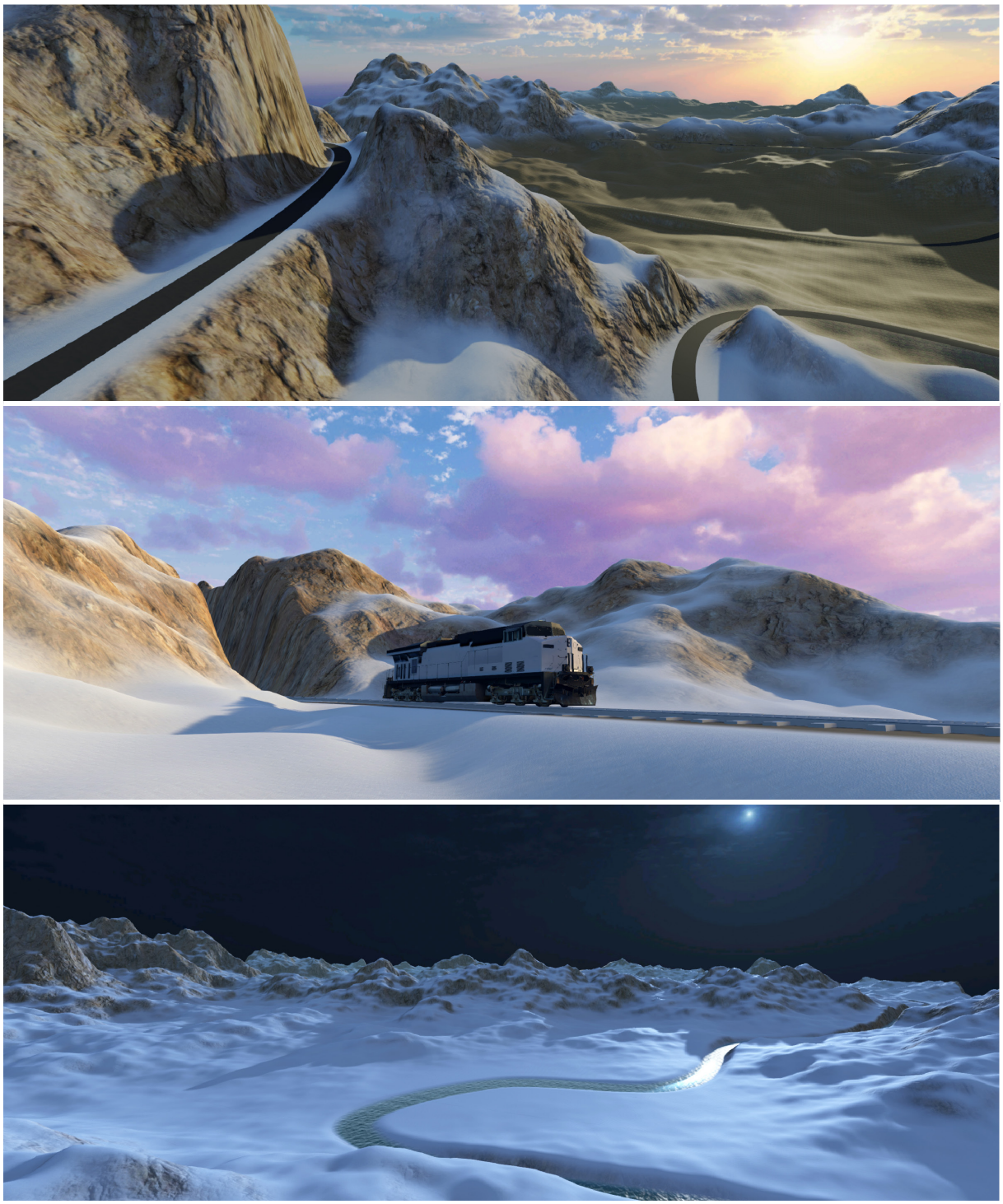


Figura 9.11 – Caminhos de estrada (cima), ferrovia (meio) e rio (baixo) esculpidos em um terreno. (NASCIMENTO; POZZER; FRANZIN, 2019)

10 CONCLUSÃO

Neste trabalho, apresentamos uma técnica para esculpir caminhos procedimentalmente em terrenos virtuais usando curvas de Bézier 3D. Nossa técnica tira proveito da GPU para processamento paralelo e seus resultados de desempenho demonstram que ela pode ser aplicada para edição dinâmica em tempo real e em aplicativos de visualização de dados geoespaciais.

Nossa técnica pode converter polilinhas para curvas de Bézier cúbicas com continuidade C^2 e parametrização de suavidade. As curvas representam os trajetos de rios e estradas, e são usadas para esculpir o terreno, criando escavações e taludes. O perfil dos caminhos é parametrizado, dando ao usuário mais controle sobre a aparência. Os caminhos se mesclam adequadamente com o terreno, sem marcas ou discrepâncias visuais.

A pesquisa desenvolvida ao longo deste trabalho proporcionou o melhoramento no desempenho da técnica e na qualidade visual dos resultado alcançados. A nossa abordagem para a conversão de curvas de Bézier cúbicas para quadráticas, o que permitiu a realização de cálculos analíticos de distância ponto à curva na GPU, melhorando também a precisão visual. Combinando isso com modificações em nossa abordagem de *hash* espacial, fomos capazes de aumentar substancialmente o desempenho de nosso método em relação à geração do mapa de deslocamentos.

Pudemos demonstrar que nossa abordagem de *hash* espacial reduz com sucesso a sobrecarga associada às avaliações de curvas realizadas na GPU e torna os tempos de processamento mais estáveis entre tipos distintos de conjuntos de dados.

Ainda, a associação de um valor de inclinação possibilitou a modificar o perfil dos caminhos, oferecendo ao usuário mais controle artístico e permitindo a criação de estradas mais realistas.

Atualmente, a nossa abordagem não trata corretamente caminhos cruzados ou sobrepostos. No entanto, se os caminhos se intersectam (ou seja, têm a mesma altura no ponto de cruzamento), os resultados são satisfatórios. Outra desvantagem é que se o terreno for muito grande e os caminhos cobrirem apenas uma pequena parte dele, a maior parte do mapa de deslocamentos permanecerá sem uso.

Como trabalho futuro, pretendemos adicionar suporte para o processamento de dados vetoriais de polígonos, de forma que nosso método possa ser usado para esculpir lagos e grandes rios e nivelar áreas urbanas, entre outros tipos de elementos. Além disso, uma otimização maior pode ser alcançada processando pequenas seções de um caminho por vez, favorecendo a edição em tempo real. Por último, pretendemos melhorar o processo de edição adicionando suporte para diferentes valores para os parâmetros de terraplanagem ao longo de um caminho, permitindo ao usuário controlar e modificar ainda mais o perfil dos caminhos.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLEGATE, C. S.; LAYCOCK, S. D.; DAY, A. A sketch-based system for highway design. In: ACM. **Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling**. [S.l.], 2011. p. 55–62.

____. A sketch-based system for highway design with user-specified regions of influence. **Computers & Graphics**, Elsevier, v. 36, n. 6, p. 685–695, 2012.

BRUNETON, E.; NEYRET, F. Real-time rendering and editing of vector-based terrains. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2008. v. 27, n. 2, p. 311–320.

BUSS, S. R. **3D computer graphics: a mathematical introduction with OpenGL**. [S.l.]: Cambridge University Press, 2003.

CHEN, G.-D.; WANG, G.-J. Optimal multi-degree reduction of bézier curves with constraints of endpoints continuity. **Computer Aided Geometric Design**, Elsevier, v. 19, n. 6, p. 365–377, 2002.

CHEN, X.-D. et al. Improved algebraic algorithm on point projection for bézier curves. In: IEEE. **Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)**. [S.l.], 2007. p. 158–163.

CHRISMAN, N. R. What does gis mean? **Transactions in GIS**, Wiley Online Library, v. 3, n. 2, p. 175–186, 1999.

DUNN, F.; PARBERRY, I. et al. **3D math primer for graphics and game development**. [S.l.]: Jones & Bartlett Publishers, 2010.

EBERT, D. S. et al. **Texturing & modeling: a procedural approach**. [S.l.]: Morgan Kaufmann, 2003.

ENGEL, T. A. **Um framework para a distribuição e renderização de plantas em tempo-real para paisagens em larga escala**. 2018. Dissertação (Mestrado), 2018.

FRASSON, A.; ENGEL, T. A.; POZZER, C. T. Efficient screen-space rendering of vector features on virtual terrains. In: ACM. **Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games**. [S.l.], 2018. p. 7.

FREIKNECHT, J.; EFFELSBERG, W. A survey on the procedural generation of virtual worlds. **Multimodal Technologies and Interaction**, Multidisciplinary Digital Publishing Institute, v. 1, n. 4, p. 27, 2017.

GALIN, E. et al. A review of digital terrain modeling. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2019. v. 38, n. 2, p. 553–577.

____. Authoring hierarchical road networks. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2011. v. 30, n. 7, p. 2021–2030.

____. Procedural generation of roads. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2010. v. 29, n. 2, p. 429–438.

GÉNEVAUX, J.-D. et al. Terrain generation using procedural models based on hydrology. **ACM Transactions on Graphics (TOG)**, ACM, v. 32, n. 4, p. 143, 2013.

HUIJSER, R. et al. Procedural natural systems for game level design. In: IEEE. **2010 Brazilian Symposium on Games and Digital Entertainment**. [S.l.], 2010. p. 189–198.

KELLEY, A. D.; MALIN, M. C.; NIELSON, G. M. Terrain simulation using a model of stream erosion. In: **Proceedings of the 15th annual conference on Computer graphics and interactive techniques**. [S.l.: s.n.], 1988. p. 263–268.

KELLY, G.; MCCABE, H. Citygen: An interactive system for procedural city generation. In: **Fifth International Conference on Game Design and Technology**. [S.l.: s.n.], 2007. p. 8–16.

LAGAE, A.; DUTRÉ, P. Compact, fast and robust grids for ray tracing. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2008. v. 27, n. 4, p. 1235–1244.

LENGYEL, E. **Mathematics for 3D game programming and computer graphics**. [S.l.]: Cengage Learning, 2012.

LEVIEN, R. The euler spiral: a mathematical history. **Rapp. tech**, Citeseer, 2008.

LONGAY, S. et al. Treesketch: Interactive procedural modeling of trees on a tablet. In: **SBM**. [S.l.: s.n.], 2012. p. 107–120.

LU, L.; WANG, G. Optimal multi-degree reduction of bézier curves with g2-continuity. **Computer Aided Geometric Design**, Elsevier, v. 23, n. 9, p. 673–683, 2006.

LU, L.-z.; WANG, G.-z. Optimal multi-degree reduction of bézier curves with g1-continuity. **Journal of Zhejiang University-SCIENCE A**, Springer, v. 7, n. 2, p. 174–180, 2006.

MCCRAE, J.; SINGH, K. Sketch-based path design. In: **Proceedings of Graphics Interface 2009**. Toronto, Ontario, Canada: Canadian Human-Computer Communications Society, 2009. (GI 2009), p. 95–102. ISBN 978-1-56881-470-4. ISSN 0713-5424.

MÜLLER, P. et al. Procedural modeling of buildings. In: **ACM SIGGRAPH 2006 Papers**. [S.l.: s.n.], 2006. p. 614–623.

NASCIMENTO, B. T. d.; POZZER, C. T.; FRANZIN, F. P. Procedural editing of virtual terrains using 3d bézier curves. In: IEEE. **2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S.l.], 2019. p. 135–143.

NASCIMENTO, B. T. do; FRANZIN, F. P.; POZZER, C. T. Gpu-based real-time procedural distribution of vegetation on large-scale virtual terrains. In: IEEE. **2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)**. [S.l.], 2018. p. 157–15709.

NGUYEN, H. H.; DESBENOIT, B.; DANIEL, M. Realistic road path reconstruction from gis data. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2014. v. 33, n. 7, p. 259–268.

PARISH, Y. I.; MÜLLER, P. Procedural modeling of cities. In: ACM. **Proceedings of the 28th annual conference on Computer graphics and interactive techniques**. [S.l.], 2001. p. 301–308.

PERLIN, K. An image synthesizer. **ACM Siggraph Computer Graphics**, v. 19, n. 3, p. 287–296, 1985.

____. Improving noise. In: ACM. **ACM transactions on graphics (TOG)**. [S.l.], 2002. v. 21, n. 3, p. 681–682.

PEYTAVIE, A. et al. Procedural riverscapes. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2019. v. 38, n. 7, p. 35–46.

POZZER, C. T. et al. A hash table construction algorithm for spatial hashing based on linear memory. In: ACM. **Proceedings of the 11th Conference on Advances in Computer Entertainment Technology**. [S.l.], 2014. p. 1–4.

ROSGEN, D. L. A classification of natural rivers. **Catena**, Elsevier, v. 22, n. 3, p. 169–199, 1994.

SMELIK, R. M. et al. A survey on procedural modelling for virtual worlds. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2014. v. 33, n. 6, p. 31–50.

____. A declarative approach to procedural modeling of virtual worlds. **Computers & Graphics**, Elsevier, v. 35, n. 2, p. 352–363, 2011.

TEOH, S. T. River and coastal action in automatic terrain generation. In: CITESEER. **CGVR**. [S.l.], 2008. p. 3–9.

THÖNY, M.; BILLETER, M.; PAJAROLA, R. Deferred vector map visualization. In: ACM. **SIGGRAPH ASIA 2016 Symposium on Visualization**. [S.l.], 2016. p. 16.

Unity Technologies. **Compute shaders**. 2020. Acesso em: 18 set. 2020. Disponível em: <https://docs.unity3d.com/2020.1/Documentation/Manual/class-ComputeShader.html>.

WEISSTEIN, E. W. Cubic formula. **Wolfram MathWorld**, Wolfram Research, Inc., 2002. Acesso em: 18 set. 2020. Disponível em: <https://mathworld.wolfram.com/CubicFormula.html>.

ZHANG, X. et al. Template-based 3d road modeling for generating large-scale virtual road network environment. **ISPRS International Journal of Geo-Information**, Multidisciplinary Digital Publishing Institute, v. 8, n. 9, p. 364, 2019.

APÊNDICE A – IMPLEMENTAÇÃO DO *COMPUTE SHADER*

```
1  // Arquivo PathBaker.compute
2
3  #pragma kernel BakePath
4  #define DIV float4(1.0, 1.0/MAX_INCLINATION, 1.0/MAX_DISTANCE, 1.0)
5
6  #include "PathCarving.cginc"
7
8  // Textura do mapa de deslocamentos
9  RWTexture2D<float4> _DisplacementMap;
10 // Informações de dimensões e tamanho de texel da textura
11 float4 _DisplacementMap_TexelSize;
12
13 [numthreads(8, 8, 1)]
14 void BakePath(uint3 id : SV_DispatchThreadID)
15 {
16     // Calcula a UV da thread atual
17     float2 heightmapUV = (float2(id.xy) + 0.5)
18         * _DisplacementMap_TexelSize.xy;
19     // Calcula a posição em espaço de mundo da thread atual
20     float2 worldXZPos = heightmapUV * _TerrainSize.xz;
21     // Avalia a posição atual, calcula os seus valores de altura,
22     // inclinação, distância para curva e intensidade de deslocamento,
23     // e armazena esses valores em um float4, nessa ordem.
24     float4 positionInfo = EvalPosition(worldXZPos);
25     // Utiliza os valores anteriores para calcular o deslocamento
26     // da posição atual
27     float displacement = GetPositionDisplacement(
28         positionInfo, GetTerrainHeight(heightmapUV));
29     // Normaliza os valores e armazena na textura
30     positionInfo.x = ((positionInfo.x / _TerrainSize.y) - 0.5) * 2.0;
31     positionInfo.z = abs(positionInfo.z);
32     positionInfo.w = displacement;
33     positionInfo = (positionInfo * DIV + 1) * 0.5;
34     _DisplacementMap[id.xy] = positionInfo.wzxy;
35 }
```

APÊNDICE B – IMPLEMENTAÇÃO DA TERRAPLENAGEM DE CAMINHOS

```
// Arquivo PathCarving.cginc

#define MAX_FLOAT 3.4e38 /// NEAR max float value
// Valores arbitrários para fins de teste
#define MAX_DISTANCE 100.0
#define MAX_INCLINATION 2.0

#define BLEND_HEIGHT_FUNCTION(c, i) GetBlendedHeight(c, i);
#define APPLY_INCLINATION_FUNCTION(c, i, s) GetInclinedHeight(c, i, s);
#define DIST_FUNCTION(k, p) \
    DistQuadraticAnalytical(_CtrlPts[k], _CtrlPts[k+1], _CtrlPts[k+2], p);
#define CARVE_STRENGTH(dist) smoothstep( \
    _CarvingParameters.x * 0.5 + max(0.001, _CarvingParameters.y), \
    _CarvingParameters.x * 0.5, dist)

// Estruturas de dados do hash espacial
StructuredBuffer<int2> _PivotTable;
StructuredBuffer<int> _HashTable;
StructuredBuffer<float4> _CtrlPts;
int2 _HashDim;
float2 _HashOrigin;
float2 _HashSize;
float2 _HashCellSize;

// Parâmetros de terraplenagem
// x: largura (w) | y: distância de suavização (lambda)
float4 _CarvingParameters;
float4 _InclinationParameters

/////////////// Funções auxiliares

float Cross2D(float2 a, float2 b)
{
    return a.x * b.y - a.y * b.x;
}

```

```
37 float DistLineSeg(float2 a, float2 b, float2 p, out float t)
38 {
39     float2 pa = p - a, ba = b - a;
40     t = saturate(dot(pa, ba) / dot(ba, ba));
41     return length(pa - ba * t);
42 }
43
44 float DistLineSeg(float2 a, float2 b, float2 p)
45 {
46     float t;
47     return DistLineSeg(a, b, p, t);
48 }
49
50 float DistLineSeg(float2 a, float2 b, float2 p, out float t, out float s)
51 {
52     float2 pa = p - a, ba = b - a;
53     // +1: ponto à esquerda | -1: ponto à direita
54     s = sign(Cross2D(ba, pa));
55     s += s == 0; // evita s == 0;
56     t = saturate(dot(pa, ba) / dot(ba, ba));
57     return length(pa - ba * t);
58 }
59
60 ////////////// Funções de Bézier
61 float2 EvalQuadratic2D(float2 p0, float2 p1, float2 p2, float t)
62 {
63     float u = (1 - t);
64     return u*u*p0 + 2*t*u*p1 + t*t*p2;
65 }
66
67 float GetQuadraticBezierSign(float2 a, float2 b, float t, float2 pos)
68 {
69     float2 g = 2.0 * a * t + b; /// gradiente
70     // +1: ponto à esquerda | -1: ponto à direita
71     float s = -sign(Cross2D(g, pos));
72     s += s == 0; // evita s == 0;
73     return s;
74 }
75
```

```
76  // Cálculo da distância para curva de Bézier quadrática
77  // utilizando a fórmula de Cardano.
78  // Adaptado de QUILEZ, Inigo, em:
79  // https://iquilezles.org/www/articles/distfunctions2d/distfunctions2d.htm
80  float3 DistQuadraticAnalytical(float4 p0, float4 p1, float4 p2, float2 pos)
81  {
82      float2 a = p1.xz - p0.xz;
83      float2 b = p0.xz - 2.0 * p1.xz + p2.xz;
84      float2 c = a * 2.0;
85      float2 d = p0.xz - pos;
86      float kk = 1.0 / dot(b,b);
87      float kx = kk * dot(a,b);
88      float ky = kk * (2.0 * dot(a,a) + dot(d,b)) / 3.0;
89      float kz = kk * dot(d,a);
90      float p = ky - kx * kx;
91      float q = kx * (2.0 * kx * kx - 3.0 * ky) + kz;
92      float h = q * q + 4.0 * p * p * p;
93      if( h >= 0.0)
94      {
95          h = sqrt(h);
96          float2 x = (float2(h, -h) - q) / 2.0;
97          float2 uv = sign(x) * pow(abs(x), 1.0/3.0);
98          float t = saturate(uv.x + uv.y - kx);
99          float2 qos = d + (c + b * t) * t;
100
101         float s = GetQuadraticBezierSign(b, c, t, qos);
102         return float3(EvalQuadratic2D(p0.yw, p1.yw, p2.yw, t),
103             s * length(qos));
104     }
105     else
106     {
107         float z = sqrt(-p);
108         float v = acos(q / (p * z * 2.0)) / 3.0;
109         float m = cos(v);
110         float n = sin(v) * 1.732050808;
111         float3 t = saturate(float3(m + m, -n - m, n - m) * z - kx);
112         float2 qos1 = d + (c + b * t.x) * t.x; /// ponto na 1a raiz
113         float2 qos2 = d + (c + b * t.y) * t.y; /// ponto na 2a raiz
114         float sqrD1 = dot(qos1, qos1);
```

```
115         float sqrD2 = dot(qos2, qos2);
116         float4 closest = sqrD1 < sqrD2 ?
117             float4(qos1.xy, sqrD1, t.x) : float4(qos2.xy, sqrD2, t.y);
118         /// 3a raiz pode ser ignorada
119 
120         float s = GetQuadraticBezierSign(b, c, closest.w, closest.xy);
121         return float3(EvalQuadratic2D(p0.yw, p1.yw, p2.yw, closest.w),
122             s * sqrt(closest.z));
123     }
124 }
125 ////////////////////////////
126 
127 int2 GetHashCell(float2 worldXZPos)
128 {
129     if(any(worldXZPos < _HashOrigin) ||
130        any(worldXZPos >= (_HashOrigin + _HashSize)))
131     {
132         return int2(0, 0); /// Fora dos limite do hash
133     }
134     else
135     {
136         int2 cell = (worldXZPos - _HashOrigin) / _HashCellSize;
137         return _PivotTable[cell.x + cell.y * _HashDim.x];
138     }
139 }
140 
141 float GetBlendedHeight(float4 closest, float4 onInfluence)
142 {
143     float avgOnInfluenceH = onInfluence.x / max(0.001, onInfluence.w);
144     float blendingValue = smoothstep(_CarvingParameters.z,
145         _CarvingParameters.w, closest.w);
146     return lerp(avgOnInfluenceH, closest.x, blendingValue);
147 }
148 
149 float GetInclinedHeight(float4 closest, float4 onInfluence, float side)
150 {
151     float inclinationStr = smoothstep(_InclinationParameters.x,
152         _InclinationParameters.y, closest.w);
153     float f = 1.0 / (sqrt(closest.y * closest.y + 1));
```

```
154     float offset = closest.y * closest.z * f *
155         pow(inclinationStr, _InclinationParameters.z);
156     return closest.x - offset * side;
157 }
158 
159 float4 EvalPositionInCell(float2 worldXZPos, int2 hashCell)
160 {
161     if(hashCell.y == 0) return float4(0, 0, MAX_DISTANCE, 0);
162     // x: altura | y: inclinação | z: distância | w: intensidade
163     float4 closest = float4(0, 0, MAX_FLOAT, 0);
164     float4 onInfluence = float4(0, 0, MAX_FLOAT, 0);
165     float closestDistanceSign = 0;
166 
167     // Para cada curva de Bézier indexada pela célula hash
168     for(int i = hashCell.x; i < (hashCell.x + hashCell.y); i++)
169     {
170         // Índice do primeiro ponto de controle da curva
171         int k = _HashTable[i];
172         float4 current;
173         current.xyz = DIST_FUNCTION(k, worldXZPos, _Iterations);
174         float currentDistanceSign = sign(current.z);
175         current.z = abs(current.z);
176         current.w = CARVE_STRENGTH(current.z);
177         onInfluence += current * float4(current.w, current.w, 1, 1);
178         if(current.z < closest.z)
179         {
180             closestDistanceSign = currentDistanceSign;
181             closest.xy = current.xy;
182             closest.z = min(closest.z, current.z);
183             closest.w = CARVE_STRENGTH(closest.z);
184         }
185     }
186 
187     closest.x = BLEND_HEIGHT_FUNCTION(closest, onInfluence);
188     closest.x = APPLY_INCLINATION_FUNCTION(closest, onInfluence,
189         closestDistanceSign);
190     closest.z *= closestDistanceSign;
191     return closest;
192 }
```

```
193
194 float4 EvalPosition(float2 worldXZPos)
195 {
196     return EvalPositionInCell(worldXZPos, GetHashCell(worldXZPos));
197 }
198
199 float GetPositionDisplacement(float4 positionInfo, float terrainHeight)
200 {
201     // Altura normalizada entre [-1, 1]
202     positionInfo.x = ((positionInfo.x / _TerrainSize.y) - 0.5) * 2.0;
203     return clamp((positionInfo.x - terrainHeight) * positionInfo.w,
204         -1.0, 1.0);
205 }
```