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

Tiago Augusto Engel

UM FRAMEWORK PARA A DISTRIBUIÇÃO E RENDERIZAÇÃO DE PLANTAS EM TEMPO-REAL PARA PAISAGENS EM LARGA ESCALA

Santa Maria, RS
2018

Tiago Augusto Engel

UM FRAMEWORK PARA A DISTRIBUIÇÃO E RENDERIZAÇÃO DE PLANTAS EM TEMPO-REAL PARA PAISAGENS EM LARGA ESCALA

Dissertação apresentada ao Curso de Mestrado do Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Mestre em Ciência da Computação**.

Orientador: Dr. Cesar Tadeu Pozzer

Santa Maria, RS
2018

Engel, Tiago Augusto
UM FRAMEWORK PARA A DISTRIBUIÇÃO E RENDERIZAÇÃO DE PLANTAS EM TEMPO-REAL PARA PAISAGENS EM LARGA ESCALA / Tiago Augusto Engel.- 2018.
53 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, 2018

1. Vegetação procedural 2. Renderização 3. Paisagens virtuais 4. Dados vetoriais I. Pozzer, Cesar Tadeu II. Título.

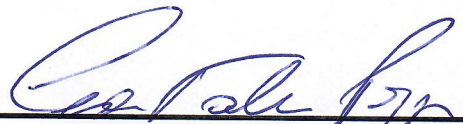
Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Tiago Augusto Engel

UM FRAMEWORK PARA A DISTRIBUIÇÃO E RENDERIZAÇÃO DE PLANTAS EM TEMPO-REAL PARA PAISAGENS EM LARGA ESCALA

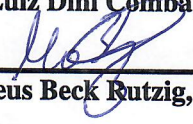
Dissertação apresentada ao Curso de Mestrado do Programa de Pós-graduação em Ciência da Computação, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria(UFSM, RS), como requisito parcial para a obtenção do grau de **Mestre em Ciência da Computação**.

Aprovado em 26 de Fevereiro de 2018:



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

João Luiz Dihl Comba, Dr. (UFRGS)



Mateus Beck Rutzig, Dr. (UFSM)

Santa Maria, RS
2018

AGRADECIMENTOS

Nós agradecemos o Exército Brasileiro pelo suporte financeiro através do projeto SIS-ASTROS.

RESUMO

UM FRAMEWORK PARA A DISTRIBUIÇÃO E RENDERIZAÇÃO DE PLANTAS EM TEMPO-REAL PARA PAISAGENS EM LARGA ESCALA

AUTOR: Tiago Augusto Engel
ORIENTADOR: Dr. Cesar Tadeu Pozzer

A renderização de vegetação em terrenos virtuais é um problema importante em computação gráfica pois as plantas compreendem uma parte predominante das paisagens naturais. Os principais desafios técnicos estão associados à complexidade dos ecossistemas naturais e ao grande número de elementos para distribuir e processar. Em particular, a distribuição de plantas é frequentemente pré-computada, impondo longos tempos de iteração durante a edição e exigindo considerável espaço de armazenamento. Com os avanços das GPUs modernas, é possível distribuir as plantas em tempo de execução. Essa dissertação apresenta um framework em tempo-real para preencher cenários virtuais com grama, arbustos e árvores. Ele é baseado em uma estrutura de dados dependente da visão utilizada para solicitar, gerenciar e renderizar plantas dinamicamente. As entradas do framework são o mapa de altura do terreno e os dados vetoriais que representam restrições naturais e realizadas pelo homem que afetam a distribuição, tais como como estradas, rios e lagos. Um algoritmo baseado em GPU avalia um conjunto de regras de posicionamento utilizando texturas derivadas automaticamente dos dados de entrada para determinar as distribuições de plantas. Os elementos gerados são agrupados e renderizados dinamicamente utilizando instanciação de geometria em GPU. O framework permite a rápida população de vastos terrenos com uma pequena sobrecarga de tempo de execução.

Palavras-chave: Vegetação procedural, renderização, paisagens virtuais, dados vetoriais.

ABSTRACT

A FRAMEWORK FOR REAL-TIME DISTRIBUTION AND RENDERING OF PLANTS ON LARGE-SCALE LANDSCAPES

AUTHOR: Tiago Augusto Engel
ADVISOR: Dr. Cesar Tadeu Pozzer

Rendering vegetation on virtual terrains is an important issue in computer graphics as plants comprehend an integral part of natural landscapes. The main challenges are associated with the complexity of natural ecosystems and the massive number of elements to distribute and render. In particular, plant distribution is often pre-computed, imposing long iteration times during editing and requiring considerable storage space. With the advances of modern GPUs, it is possible to distribute plants at runtime. This dissertation presents a real-time framework to populate virtual scenarios with grass, bushes, and trees. The framework is based on a view-dependent data structure employed to request, manage and render plants dynamically. The inputs of our system are the terrain heightmap and vector features representing natural and human-made distribution constraints such as roads, rivers, and lakes. Based on textures derived automatically from the input data, a GPU-based algorithm evaluates a set of placement rules to determine the plant distributions. The elements generated are dynamically batched and rendered using geometry instancing. The framework allows the quick population of vast terrains with a small runtime overhead.

Keywords: Procedural vegetation, rendering, virtual landscapes, vector data.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A visualização de mundos virtuais em larga escala tem aplicações em diversas áreas, tais como videogames, simuladores virtuais e estudos ambientais. Nesse contexto, a vegetação é frequentemente o elemento dominante das paisagens. A visualização de plantas 3D tem sido objeto de pesquisas científicas há muito tempo devido ao seu impacto visual em mundos virtuais, apresentando desafios quanto ao desempenho de renderização, qualidade visual e de distribuição. Uma paisagem típica pode facilmente conter milhões de plantas, variando em ecótipo, forma, cor, animação, padrão de distribuição e resposta à iluminação.

A distribuição manual de plantas em grandes ambientes é muitas vezes impraticável pois envolve muitas horas de trabalho laborioso. Por outro lado, abordagens totalmente procedurais não permitem controle do usuário ou oferecem pouco dinamismo. Abordagens mistas que combinam métodos manuais e procedurais são as mais comuns, onde a distribuição é armazenada em disco uma vez gerada. Os artistas geralmente utilizam mapas de densidade pintados manualmente para transmitir seu design, definindo áreas que podem conter elementos e atribuindo semânticas a eles (ex.: *perto de um rio*). No entanto, ao criar grandes paisagens, o tempo de pré-cálculo e o armazenamento podem impor limitações. Nesse contexto, várias aplicações podem tirar proveito de ferramentas automatizadas para gerar esses mapas e distribuir plantas em tempo de execução. Os artistas se beneficiam de feedback imediato durante a produção, enquanto as demandas de armazenamento e streaming de disco são reduzidas.

Na criação de cenários baseados no mundo real, informações espaciais como o modelo digital de elevação do terreno, as imagens de satélite e os dados vetoriais são úteis no processo de criação. Em tais aplicações, os dados vetoriais representam características naturais e feitas pelo homem (ex.: estradas, rios, florestas e lagos). Além disso, as informações espaciais representam restrições que afetam a distribuição de plantas (ex.: árvores não crescem nas ruas).

Nesse contexto, propõe-se um framework *semi-automático* para distribuir e renderizar vegetação em mundos virtuais 3D em tempo real. Ele depende dos dados de terreno e vetoriais para gerar restrições de *onde* e *como* a vegetação é distribuída no terreno. O algoritmo de posicionamento avalia as informações derivadas dos dados de entrada para criar distribuições de plantas na GPU em tempo de execução. Utiliza-se os dados vetoriais que representam estradas, rios, áreas de vegetação e lagos para calcular um campo de distância (DF) e um mapa de material procedural (splatmap) durante a execução. Essa informação é utilizada para determinar regiões tais como *próximo de um rio* ou *em uma floresta*, as quais são úteis para a distribuição adequada da vegetação. A solução é integrada ao sistema de terreno, onde uma estrutura de quad-tree que depende da visão é usada para gerar e armazenar a distribuição sob demanda por nó.

A presente pesquisa está direcionada ao desenvolvimento de um simulador virtual para o Exército Brasileiro (projeto Astros 2020). Portanto, o framework compreende restrições adicionais, como física (as árvores representam obstáculos estáticos), espaçamento de elementos (para o trânsito de caracteres e veículos) e o terreno subjacente. No entanto, os métodos des-

critos aplicam-se a jogos 3D e ambientes virtuais em geral. Por fim, mostramos como nossa estrutura pode ser integrada à edição manual.

1.1 OBJETIVOS

O principal objetivo da presente pesquisa é propor e implementar um framework para distribuição e renderização de grama, árvores e arbustos em grandes mundos virtuais em tempo real. Mais especificamente:

- Avaliar soluções modernas para distribuição em tempo real e renderização de vegetação em grandes terrenos virtuais;
- Propor e implementar um sistema escalável em tempo real baseado em GPU para distribuição e renderização de vegetação;
- Integrar a solução a um sistema de terreno, a fim de extrair informações para determinar as posições das plantas;
- Distribuir as plantas de forma plausível;
- Gerenciar o LOD da vegetação.

1.2 CONTRIBUIÇÕES

A principal contribuição dessa pesquisa é um framework baseado em GPU para distribuir a vegetação em tempo real. Demonstramos como associar os dados de terreno e vetoriais para impor restrições à distribuição de plantas. O algoritmo gera informações de material e distâncias de maneira dependente da visão, fornecendo uma solução escalável para visualização de terrenos em larga escala. A distribuição e as texturas de cada nó são geradas em compute shaders assíncronos, aproveitando os recursos crescentes das GPUs modernas. O framework leva em consideração uma ampla gama de tópicos e suas implicações para o projeto de um sistema de vegetação totalmente funcional voltado para uma aplicação real.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

A dissertação está organizada da seguinte maneira: apresenta-se os conceitos básicos relacionados aos dados geográficos e ao pipeline gráfico moderno no Capítulo 2. O Capítulo 3 revisa o estado da arte em relação à distribuição e renderização de plantas; a abordagem proposta para associar as fontes de informação (ex.: os dados de terreno e vetoriais) para a distribuição e a renderização de plantas é descrita no Capítulo 4. O Capítulo 5 mostra os resultados obtidos em relação ao desempenho e à qualidade visual. Finalmente, no Capítulo 6 é feita uma discussão sobre a pesquisa desenvolvida e as direções para trabalhos futuros.

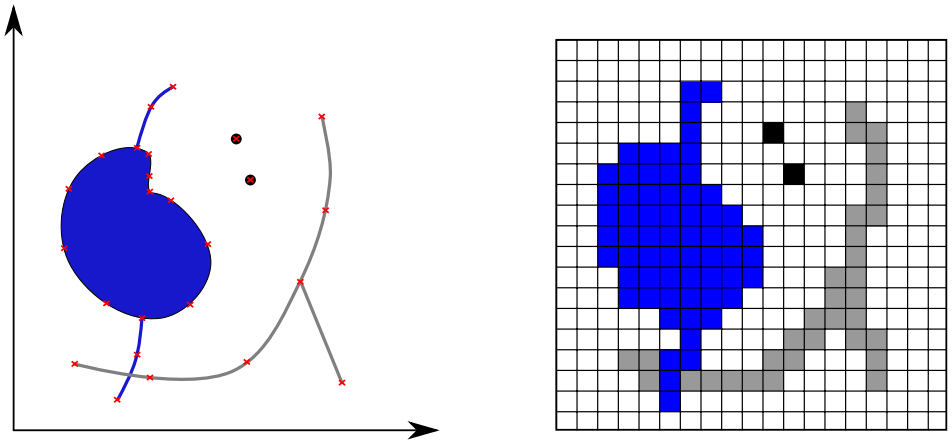
2 FUNDAMENTAÇÃO

Este capítulo apresenta os conceitos relacionados a essa pesquisa. Apresenta-se uma revisão da representação de dados de Sistemas de Informações Geográficas (SIG), que é a principal fonte de dados utilizada na modelagem de cenários reais. Por fim, são discutidos aspectos dos recursos do pipeline gráfico e arquitetura de GPU exploradas nessa dissertação.

2.1 REPRESENTAÇÃO DE DADOS DE SIG

Um Sistema de Informações Geográficas é um sistema de software e hardware projetados para capturar, armazenar e processar dados espaciais (CHRISMAN, 1999). Os objetos geográficos retratam uma abstração da realidade situada dentro de um sistema de referência comum. As ferramentas de SIG também permitem a visualização dos dados, tendo aplicações em diversas áreas, como sistemas de navegação e ferramenta de apoio à tomada de decisão. Os dados são geralmente divididos em várias camadas, cada uma representando uma categoria diferente, como estradas, rios, lagos, vegetação e edifícios. Os principais modelos de dados usados no SIG são o *raster* e o *vetorial*, como ilustrado na Figura 1. Cada um deles é mostrado como uma camada separada, embora eles representem informações diferentes da mesma área.

Figura 1 – Exemplo de dados de SIG. Na esquerda, a representação vetorial (os marcadores vermelhos indicam vértices). Na direita, a mesma localização representada usando um formato raster.



Os dados raster raster são representados por uma matriz regular de valores que descrevem pontos de um campo contínuo (ex.: elevação ou temperatura) ou pixels de uma imagem. Nessa dissertação, refere-se a raster como imagens (ex.: de fontes de satélite ou aéreas) ou Modelo Digital de Elevação (MDE) representando a topografia da superfície do solo (ex.: heightmap). Em geral, o formato raster captura dados contínuos, onde cada ponto na matriz (ou seja,

pixel) pode ser interpolado com vizinhos. O tamanho de cada célula representa a resolução e está diretamente relacionado ao espaço de armazenamento exigido.

O modelo *vetorial* representa objetos geométricos definidos por primitivas como pontos, polilinhas e polígonos. Esse modelo é especialmente útil para denotar objetos em alto nível, como caminhos (ex.: estradas, rios) e áreas (ex.: lagos, área de vegetação e limites em geral). Cada primitiva é definida por vértices que geralmente correspondem a latitude e longitude no globo. Os dados vetoriais têm vantagens em relação à resolução, tamanho de armazenamento e facilidade de codificar e interpretar informações topológicas.

Existem bancos de dados públicos, como o United States Geological Survey (2017) e o OpenStreetMap (2017), que fornecem acesso a dados vetoriais do mundo real. Devido aos grandes conjuntos de dados usados nos SIG, algoritmos automatizados para extrair informação dos MDEs e imagens de satélite são um tópico de pesquisa ativo (CHAI ET AL., 2013; GRUEN E LI, 1995; SOILLE E GRAZZINI, 2007; TARBOTON ET AL., 1991; TUNCER, 2007; TUPIN ET AL., 1998). No entanto, a intervenção humana pode ser utilizada para corrigir possíveis inconsistências e garantir a qualidade dos dados. Em terrenos artificiais, os dados vetoriais podem ser editados manualmente por artistas, gerados proceduralmente (GALIN ET AL., 2010; GÉNEVAUX ET AL., 2013) ou uma mistura de ambos (SMELIK, TUTENEL, KRAKER ET AL., 2010).

2.2 PIPELINE GRÁFICO MODERNO

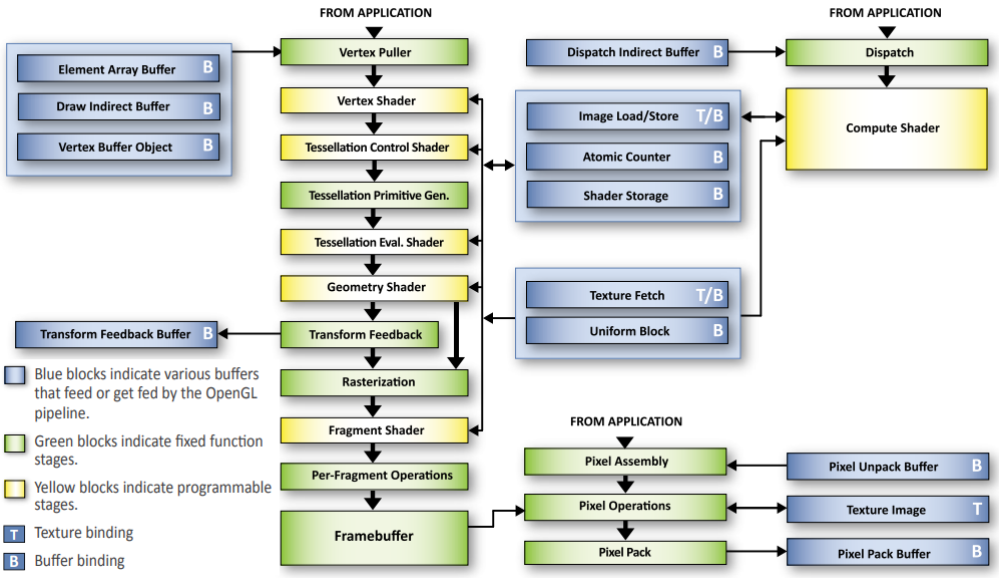
O hardware das Unidades de Processamento Gráfico é, na maioria das vezes, classificado como instrução única, dados múltiplos (SIMD) (FLYNN, 1972), onde múltiplas unidades de processamento executam a mesma operação em vários pontos de dados simultaneamente. A introdução do pipeline gráfico moderno forneceu mais flexibilidade aos desenvolvedores, permitindo o proveito dos recursos avançados da GPU. A Figura 2 resume o pipeline de renderização da API do OpenGL 4.5, cuja arquitetura é semelhante à do Directx11. A compreensão do fluxo de trabalho do pipeline subjacente é essencial no contexto dessa dissertação, visto que a solução proposta utiliza diferentes recursos para distribuir e renderizar a vegetação. Mais especificamente, fazemos uso extensivo de compute shaders assíncronos para diversas finalidades.

Os estágios programáveis do pipeline gráfico são representados pelos blocos amarelos na Figura 2. O primeiro estágio é o shader de vértices, que realiza transformações por vértice individualmente. Os shaders de tesselagem e geometria permitem gerar e transformar a geometria diretamente na GPU. Após a rasterização da geometria, o shader de fragmentos processa cada amostra para executar, por exemplo, coloração e iluminação. Mais detalhes de cada estágio são descritos por Wright et al. (2010).

Os compute shaders operam de maneira diferente dos estágios do pipeline gráfico, estendendo-o para oferecer funcionalidades semelhantes ao CUDA/OpenCL, porém com uma integração

mais próxima com o pipeline gráfico. Enquanto os estágios gráficos são executados em tempos padrão, os de compute são baseados em demanda e despachados explicitamente da aplicação. Além disso, eles são projetados para propósitos gerais e, portanto, não vinculados à geometria ou pixels. A entrada e a saída também são genéricas, permitindo cálculos de uso geral na GPU e os resultados podem ser passados para estágios gráficos regulares ou lidos de volta para a CPU. A Figura 2 ilustra a interoperabilidade, onde os compute shaders podem compartilhar objetos como texturas, contadores e buffers com o pipeline gráfico. Uma descrição completa das funcionalidades é fornecida na documentação do OpenGL (OPENGL WIKI CONTRIBUTORS, 2018).

Figura 2 – Pipeline Gráfico do OpenGL 4.5 (THE KHRONOS GROUP, 2017).



Existem vários tipos de memória disponíveis para uso na GPU, cada uma variando em tamanho e latência, conforme mostrado na Tabela 1. A organização geral da hierarquia de threads e da memória é mostrada na Figura 3. Cada thread tem registradores privados de rápido acesso e memória local que é mais lenta. A memória compartilhada é acessível de todos os elementos de um grupo. A memória global está disponível para todos os blocos, sendo consideravelmente mais lenta que as demais. A memória constante armazena variáveis globais e argumentos do kernel definidos na CPU.

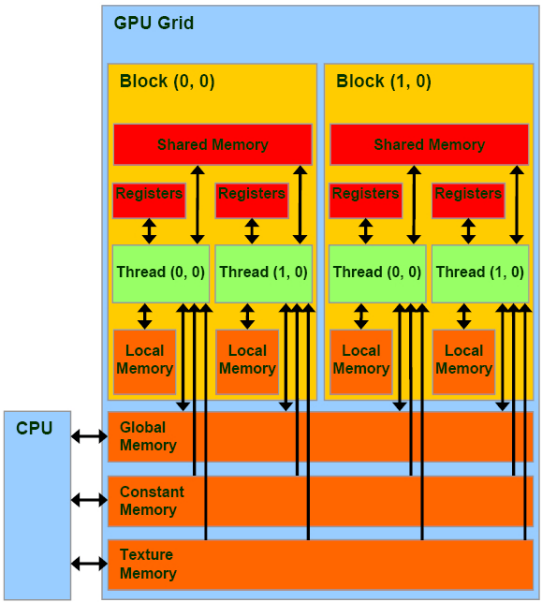
Os compute shaders podem ser despachados diretamente da aplicação usando argumentos explícitos ou esses podem ser passados via um *buffer indireto* de despacho em GPU. O último tem aplicações quando é indesejável o involvimento da CPU no processo, por exemplo, para enviar threads para operar em um buffer de GPU quando a CPU não está ciente de seu tamanho. Similarmente ao CUDA/OpenCL, o desenvolvedor é responsável por determinar o número de grupos de threads (também conhecidos como *blocos*) e as threads por grupo despachadas. A hierarquia de threads é mostrada na Figura 3. Por exemplo, para preencher um array

Tabela 1 – Hierarquia de memória da GPU. Adaptado de (OOSTEN, 2011).

Memória	**Localização**	**Acesso Kernel**	**Acesso Host**	**Escopo**	**Ciclo de Vida**
Registrador	cache	R/W	não	thread	thread
Local	dispositivo	R/W	não	thread	thread
Compartilhada	cache	R/W	não	grupo	grupo
Global	dispositivo	R/W	R/W	aplicação	aplicação
Constante	dispositivo	R	R/W	aplicação	aplicação

de N elementos (um por thread), e assumindo tamanhos de grupo de 16 threads, o número de grupos despachados é definido como $tg_x = ceil(\frac{N}{16})$. O número final de threads é $td = tg_x \cdot 16$. Essa intuição vale para duas e três dimensões de grupos. Nessa dissertação, utiliza-se essa abordagem para gerar as texturas de payload e as distribuições para cada nó em GPU.

Figura 3 – Hierarquia de threads da GPU.



O tamanho crescente e a qualidade visual esperada de aplicações 3D modernas exigem soluções eficientes e a exploração de vários níveis de paralelismo. Aplicações modernas utilizam compute shaders para ex.: culling (HAAR E AALTONEN, 2015; WIHLIDAL, 2016), redução do número de chamadas de desenho (WIHLIDAL, 2016), distribuição de plantas (MUIJDEN, 2017), entre outros. Esses princípios são observados no framework proposto para gerar e refinar o heightmap e outras texturas (Seção 4.3), assim como para a distribuição de plantas (Seção 4.4).

3 TRABALHOS RELACIONADOS

Este capítulo descreve a literatura subjacente relacionada a essa pesquisa. Os níveis de realismo envolvidos na modelagem e distribuição procedural de plantas são descritos. Apresentamos as técnicas de renderização aplicadas à vegetação e os níveis de detalhamento utilizados para representar as plantas.

3.1 MÉTODOS PROCEDURAIS PARA VEGETAÇÃO

Os tópicos envolvidos na distribuição, na renderização, na animação e na iluminação das plantas compreendem especialidades multidisciplinares. A simulação de plantas pode ser dividida em três níveis de resolução (SMELIK, TUTENEL, BIDARRA ET AL., 2014): órgãos individuais, plantas e ecossistemas completos. As abordagens de geração de ecossistema procedural são divididas em *local-para-global* e *global-para-local*, conforme descrito por Lane e Prusinkiewicz (2002).

A modelagem procedural de plantas envolve a geração, com diversos níveis de interatividade do usuário, de modelos 3D realistas de plantas. Os sistemas L (e suas extensões) são as abordagens mais comuns para resolver essa tarefa (DEUSSEN E LINTERMANN, 2005; DEUSSEN, HANRAHAN ET AL., 1998; PRUSINKIEWICZ E LINDENMAYER, 1990). As ferramentas comerciais para modelagem procedural de plantas incluem XFrog, Plant Factory e SpeedTree. No entanto, a presente pesquisa concentra-se em aspectos macro das plantas, por isso a modelagem individual de plantas e a iluminação estão fora do nosso escopo (veja Smelik, Tutenel, Bidarra et al. (2014) para uma revisão sobre esse tópico).

3.1.1 Local-para-global

Em abordagens locais-para-globais, o ecossistema é baseado em simulações individuais de crescimento e competição. O extenso corpo de trabalho nessa área simula a natureza com vários níveis de realismo, considerando diferentes variáveis que afetam a distribuição. Durante a simulação, as plantas envelhecem, morrem e novas sementes são plantadas, efetivamente simulando um sistema dinâmico de interações ao longo de um período de tempo. Esse nível de simulação tem vantagens em relação à distribuição mais consistente e realista, ao custo de tempos de processamento mais longos.

Deussen, Hanrahan et al. (1998) propuseram um sistema onde a distribuição de plantas é de autoria manual, simulação de ecossistema ou uma combinação de ambos. Os autores fornecem um mecanismo para simular a competição por recursos e um sistema L para modelagem de plantas. Lane e Prusinkiewicz (2002) propuseram um algoritmo de lançamento de dardos baseado em um campo de probabilidade. Os autores estenderam o sistema L de Deussen, Hanrahan

et al. (1998) para simular populações de plantas. Essa abordagem foi estendida por Alsweis e Deussen (2006), onde foi proposto o modelo Campo de Vizinhança (FoN), que descreve a área de influência de cada planta.

Beneš et al. (2011) propuseram uma abordagem para a distribuição espacial de plantas em ambientes urbanos onde cada área da cidade recebe um nível de gerenciabilidade. Algumas áreas são deixadas inteiramente selvagens onde ocorre a simulação da competição de plantas, enquanto que para outras áreas são aplicadas regras de distribuição específicas do domínio (ex.: as árvores são colocadas ao longo das estradas principais de acordo com a entrada do usuário). Os autores empregaram a simulação de ecossistema usando algoritmos adaptados de (DEUSSEN, HANRAHAN ET AL., 1998; LANE E PRUSINKIEWICZ, 2002), conforme o grau de gerenciabilidade da área. O algoritmo utiliza o feedback de um módulo de simulação da cidade, de onde informações como a textura de footprint da cidade e a distância até as estradas são usadas para modular o crescimento das árvores. A abordagem se relaciona com a nossa no sentido de que, embora não tenhamos um módulo de simulação de cidade, utilizamos dados geográficos com o mesmo propósito de impor restrições à distribuição de plantas.

Figura 4 – a) fotografia aérea de um bairro em Chicago. b) vizinhança simulada com a abordagem de Beneš et al. (2011).

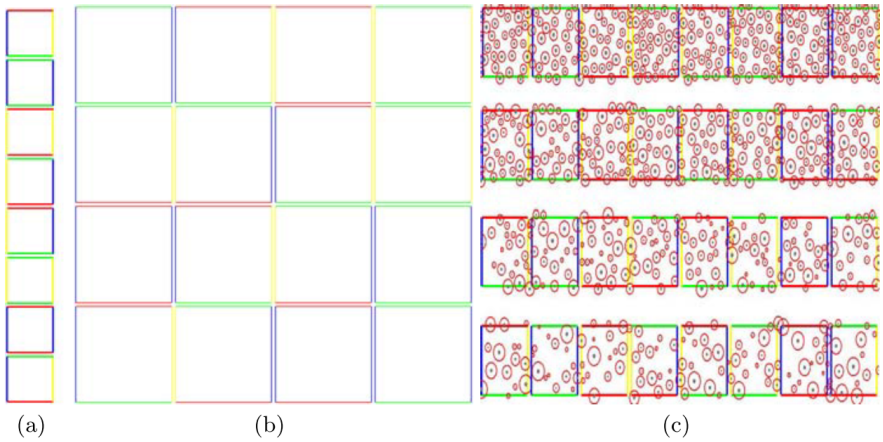
A simulação de ecossistemas é capaz de fornecer resultados realistas, mas ainda não é adequada para propósitos em tempo real, visto que simular cada interação da planta é computacionalmente cara. Em computação gráfica, esses modelos de simulação são simplificados (BRADBURY ET AL., 2015; CH'NG, 2011; CORDONNIER ET AL., 2017; GAIN ET AL., 2017) como uma compensação para permitir interatividade, controle artistístico e desempenho.

3.1.2 Global-para-local

Essa abordagem consiste em inferir o ecossistema através de uma distribuição de densidades de plantas. A distribuição é estática (não simula novas plantas), representando a população de plantas em um ponto específico no tempo. Ao contrário da abordagem local-para-global, essa técnica não compreende um processo de simulação, e a disposição é inferida a partir de regras globais. Essa técnica apresenta algoritmos mais simples que são atraentes pelo seu desempenho, porém os resultados costumam ser menos realistas.

A distribuição de disco de Poisson (PDD) permite uma amostragem uniformemente espaçada com pelo menos um raio definido pelo usuário entre pontos. Bridson (2007) propôs um algoritmo otimizado para gerar as amostras que executam em tempo $O(n)$. Ainda assim, o desempenho pode ser proibitivo para um grande número de amostras. Uma alternativa comum para gerar populações de plantas utiliza Wang tiles associadas a um PDD dentro de cada célula (ALSWEIS E DEUSSEN, 2006; BRUNETON E NEYRET, 2012; ONRUST ET AL., 2015; WEIER ET AL., 2013). Essa abordagem possui alguns recursos atraentes, como baixo consumo de memória e escalabilidade. No entanto, a distribuição não leva em consideração informações espaciais. A Figura 5 demonstra um exemplo de distribuição. Essa abordagem é uma alternativa ao PDD original para permitir a população de grandes áreas com um custo fixo, caso contrário, a PDD pode ser computacionalmente cara.

Figura 5 – Exemplo de Wang tiles com PDD (ALSWEIS E DEUSSEN, 2006). a) conjunto de tiles; b) disposição válida; c) distribuição com diferentes densidades.

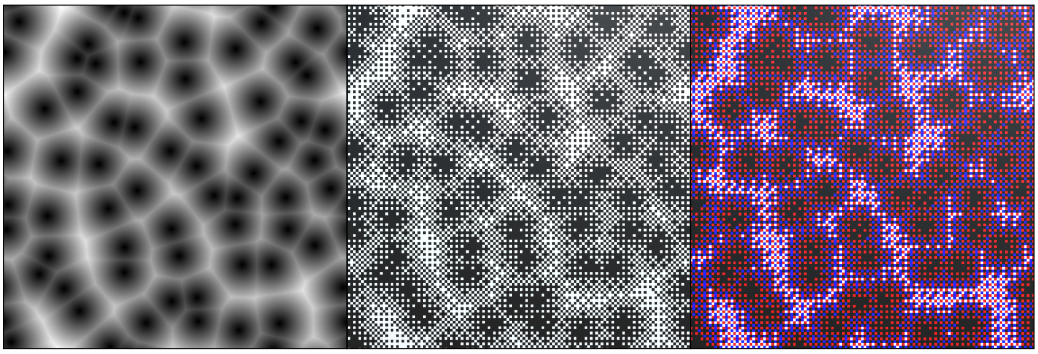


O esquema de criação interativa proposto por Deussen, Hanrahan et al. (1998) se baseia em um mapa de densidade de tons de cinza definido pelo usuário. A distribuição é obtida aplicando o algoritmo de meia tonalidade de Floyd-Steinberg (FLOYD E STEINBERG, 1976) na imagem, onde cada pixel preto representa uma posição da planta. Uma abordagem semelhante foi proposta por Muijden (2017), onde o dithering ordenado é empregado usando a matriz de

Bayer (BAYER, 1973). A Figura 6 mostra um exemplo de dithering aplicado em um mapa de densidade.

Recentemente, uma abordagem baseada em GPU utilizando compute shaders foi introduzida no jogo Horizon Zero Dawn (MUIJDEN, 2017) para distribuição de plantas em tempo real. Os autores apresentam o conceito de *camada* que corresponde a um ecótipo. O terreno é dividido em uma grade uniforme e o algoritmo de distribuição é executado para todas as camadas que potencialmente possuem elementos dentro de cada célula. A distribuição é baseada no algoritmo de dithering ordenado de Bayer (BAYER, 1973) aplicado sobre um mapa de densidade definido pelo artista. Os autores apresentam uma poderosa ferramenta baseada em nós para gerar as distribuições. Embora a abordagem apresente resultados de alta qualidade e algumas idéias gerais sejam fornecidas, ela não apresenta detalhes de implementação pois é uma aplicação proprietária.

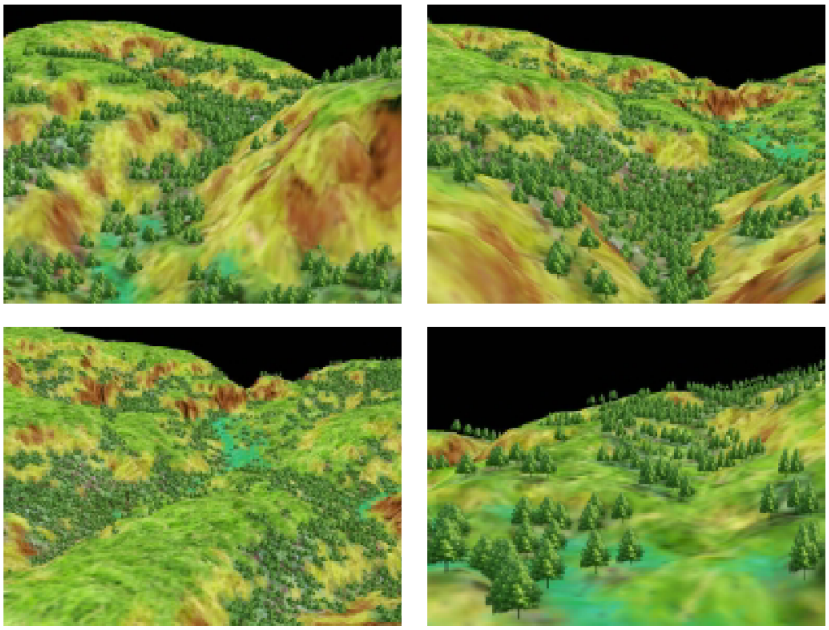
Figura 6 – Exemplo de dithering ordenado usando a matriz de Bayer. Discretização binária (centro) e por intervalos de limiar (direita) aplicadas sobre um mapa de densidade (esquerda).



Hammes (2001) propôs ecossistemas pré-definidos de acordo com a probabilidade de sua existência, dadas as condições ambientais. Sua abordagem considera as características do terreno como elevação, altitude relativa, inclinação e direção da inclinação. O ecótipo mais adequado é escolhido para cada amostra gerada de acordo com as propriedades calculadas do terreno. Os resultados da abordagem proposta são mostrados na Figura 7. A simplicidade da abordagem e independência entre amostras é atraente para fins de tempo real. Um esquema de distribuição similar foi proposto porNetzel e Rohleder (2016), onde os autores usam uma amostragem estocástica estratificada para colocar elementos.

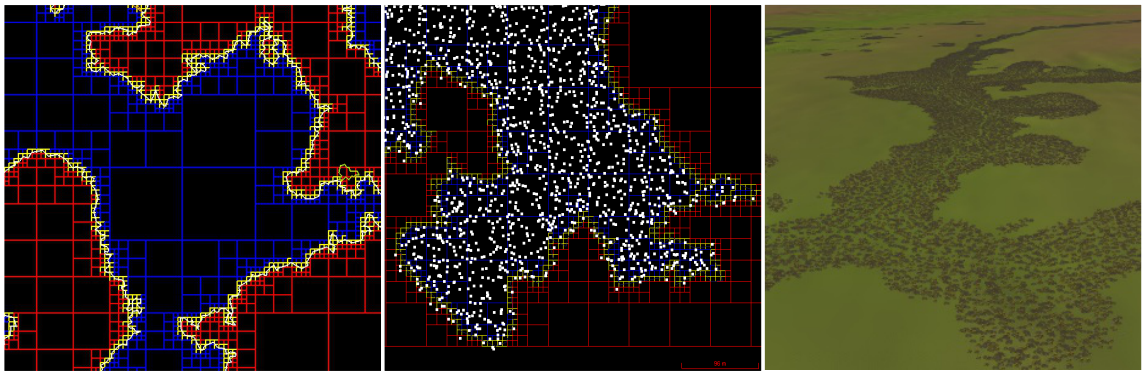
Em uma abordagem anterior, Engel et al. (2016) propuseram um algoritmo baseado em CPU para distribuir árvores em polígonos. O algoritmo aproxima os limites de polígonos com quad-trees até que um número mínimo de arestas seja alcançado. Finalmente, os nós são preenchidos com árvores colocadas aleatoriamente e o conjunto de dados é armazenado em disco. Embora o algoritmo seja relativamente rápido, as dimensões do mundo virtual ocasionam em grandes conjuntos de dados que inviabilizam altos níveis de detalhes. Além disso, a distribui-

Figura 7 – Distribuição proposta por Hammes (2001).



ção não considera as restrições do terreno. A Figura 8 mostra a subdivisão da quad-tree e os resultados obtidos no mundo 3D.

Figura 8 – Distribuição de árvores em polígonos de floresta (ENGEL ET AL., 2016). Nós azuis, amarelos e vermelhos estão totalmente dentro, parcialmente e totalmente fora do polígono, respectivamente.



Uma vantagem típica das técnicas global-para-local é que os algoritmos são mais simples e rápidos, permitindo a população de grandes áreas com baixo processamento. No entanto, elas geralmente desconsideram as regras ecologicamente corretas e contam com a entrada do usuário. Como apontado por Onrust (2015), o uso de fontes de dados reais está atualmente limitado a mapas de altura e mapa de solos. Nesse contexto, o framework propõe a geração de mapas a partir dos dados de SIG associados à área, a fim de ampliar as possibilidades de

tais técnicas. As texturas permitem um novo nível de personalização que usa a semântica de contexto para melhorar a distribuição (ex.: nas proximidades do rio).

3.2 Renderização de Vegetação

Os métodos de representação de plantas em ambientes 3D foram amplamente estudados em computação gráfica. A quantidade de geometria envolvida torna a tarefa desafiadora para propósitos em tempo real, exigindo simplificações em várias frentes, como LOD e iluminação. As principais abordagens são baseadas em representações de geometria, imagem ou volume.

As abordagens baseadas em geometria consistem na renderização de plantas geometricamente modeladas. Essa técnica fornece resultados de alta qualidade, com total efeito de paralaxe, mapeamento de textura e suporte para animação. No entanto, seu uso é normalmente restrito a visualizações próximas à câmera, pois o alto número de triângulos apresenta grandes problemas de desempenho. O impacto é especialmente limitante nas árvores, pois o número de folhas e galhos é muito alto. Por outro lado, abordagens baseadas em geometria podem ser usadas para renderizar grama de visão de perto, onde representações implícitas permitem a geração da geometria diretamente em GPU (Fan et al., 2015; Jahrmann e Wimmer, 2017; Outerra, 2012). Na Figura 9 (a, b, c), uma árvore geométrica com diferentes níveis de detalhamento é exibida.

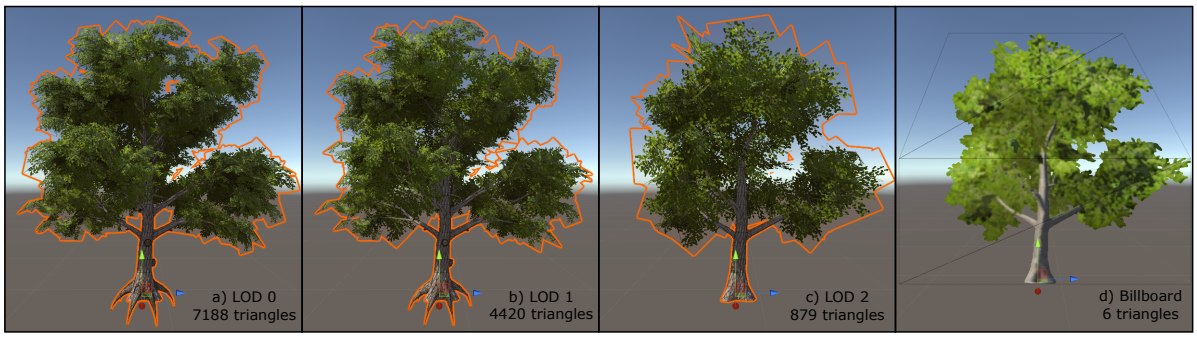
A renderização *baseada em imagens* é mais rápida e retrata uma aproximação do modelo geométrico. A abordagem mais comum é popularmente conhecida como *billboards ou impostores*, em que um quadrado de dois lados é renderizado com uma textura semi-transparente aplicada. As billboards podem ser agrupados em patches (Pelzer, 2004) para reduzir o número de plantas individuais (ex.: uma forma de estrela). Essa representação pode ser usada para renderizar plantas inteiras ou partes individuais, como galhos e folhas. Embora essa técnica seja atraente devido ao seu desempenho, ela fornece resultados indesejáveis em visualizações próximas ou superiores, devido à iluminação de aparência plana e efeitos limitados de paralaxe e animação. A Figura 9 (d) mostra um exemplo de uma árvore de billboard.

Abordagens baseadas em *volume* são uma variação da técnica baseada em imagem proposta por Decaudin e Neyret (2009) onde os autores usam uma textura 3D, podendo representar o efeito paralaxe adequadamente. Para cada fatia da textura 3D, a geometria é renderizada entre os dois planos de recorte e o resultado é armazenado na célula. No entanto, essa solução tem desvantagens significativas relacionadas ao alto uso de memória e à necessidade de estruturas de aceleração durante a renderização. O mecanismo mip-map pode ser usado nesse caso para fornecer LOD.

3.3 Níveis de Detalhamento de Vegetação

A vegetação tradicionalmente usa LODs discretos, onde cada malha de árvore é pré-computada em vários níveis de detalhamento e salva como modelos diferentes. Atualmente, os softwares de modelagem comercial são capazes de exportar automaticamente o modelo em vários LODs. A Figura 9 mostra um exemplo da mesma árvore em diferentes LODs. Enquanto para renderização em tempo real os modelos são simplificados para a melhoria de desempenho, isso não é necessariamente uma restrição para renderizadores off-line. Essa descrição varia de acordo com o renderizador alvo e o orçamento de desempenho disponível.

Figura 9 – Quatro níveis de detalhamento de uma árvore. O modelo foi projetado usando o software SpeedTree e obtido na Unity Asset Store.



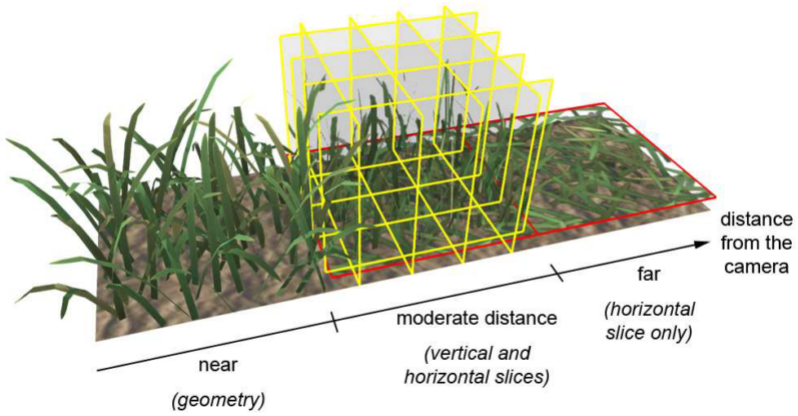
Em sistemas de renderização em tempo real, as árvores e os arbustos contêm vários níveis de detalhes por motivos de desempenho e qualidade. Os níveis mais detalhados contêm apenas geometria, enquanto o meio é um híbrido de geometria e billboards, onde o tronco e os galhos são modelados geometricamente e os billboards representam folhas. Finalmente, os níveis menos detalhados contêm apenas um ou vários billboards em forma de estrela.

Um princípio similar se aplica à grama, entretanto, devido à maior densidade, os billboards são uma escolha comum em sistemas de renderização em tempo real (às vezes dois ou mais billboards entrelaçados em forma de estrela). Por outro lado, vários trabalhos foram propostos usando geometria implícita para renderizar grama em GPU, representando a lâmina de grama como uma curva quadrática de Bézier (Fan et al., 2015; Jahrmann e Wimmer, 2017) avaliada em GPU. No entanto, essas abordagens não têm suporte para modelos definidos por artistas.

Boulanger et al. (2009) propuseram uma abordagem para renderização de grama com iluminação dinâmica e sombras em tempo real usando métodos baseados em geometria e volume. Os autores fornecem um sistema LOD baseado em uma quad-tree que permite uma visão de perto com polígonos texturizados detalhados e visão panorâmica com uma representação

simplificada de volume e fatia de texturas. A abordagem é focada em grama curta e estática e, portanto, lida com animação de forma limitada.

Figura 10 – Níveis de detalhamento da grama na abordagem de Boulanger et al. (2009).



A alternância entre os níveis de detalhamento é realizada de acordo com uma métrica. A mais simples é a distância até a câmera, porém essa permite que objetos menores que um pixel sejam renderizados. Nesse caso, o tamanho do objeto na tela evita o desenho desnecessário de objetos que não contribuam para a imagem final. A transição entre os LODs é importante para evitar efeitos *popping* visíveis e pode ser manipulada usando técnicas como cross-fade (KHARLAMOV E CANTLAY, 2007), fade e alpha-blending (GIEGL E WIMMER, 2007).

4 O FRAMEWORK

4.1 VISÃO GERAL

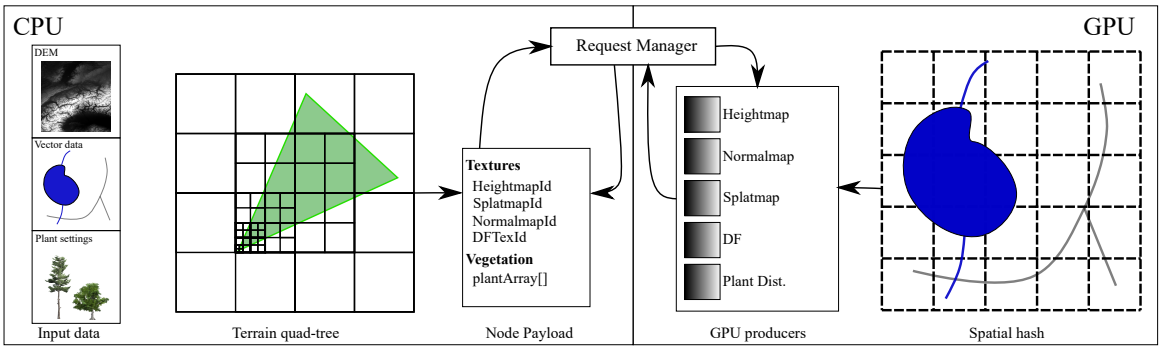
O objetivo dessa pesquisa é desenvolver um framework visando o preenchimento de um terreno virtual com plantas procedurais, considerando as restrições impostas pelas características do terreno e dos dados vetoriais. Esse comportamento pode ser facilmente encontrado na natureza, onde as plantas com afinidade a água crescem nas margens dos rios, por exemplo. As soluções apresentadas tentam replicar um ambiente real semi-automaticamente usando o mapa de altura do terreno e os dados vetoriais como entrada para distribuir a vegetação em tempo real. O framework não aborda a simulação de plantas individuais no sentido de modelagem, iluminação e animação. Ao invés disso, trabalha-se em um nível populacional.

No que diz respeito à distribuição, o objetivo é a colocação plausível de elementos no ambiente ao contrário de simulações ecologicamente precisas. A vegetação natural apresenta frequentemente um comportamento de agrupamento resultante da propagação de sementes (entre outros fatores), o que é indesejável em nossa aplicação uma vez que objetiva-se o trânsito de caracteres e veículos entre as árvores. Além disso, a simulação impõe dependência mútua entre partes do terreno, isto é, a simulação de um nó pode depender de outro. Nesse contexto, as técnicas local-para-global não são adequadas à nossa solução, pois podem levar a etapas de geração arbitrariamente complexas que afetam o desempenho. Além disso, para aplicações como jogos ou simuladores virtuais, a simulação contínua (ex.: crescimento e morte) de plantas é muitas vezes desnecessária.

O framework estende a abordagem de distribuição de plantas de Hammes (2001) para suportar as restrições impostas por dados vetoriais. Seu trabalho leva em conta as características do terreno, como elevação e inclinação, para determinar o ecótipo mais adequado. No entanto, ele não considera outras restrições naturais e realizadas pelo homem. A combinação de informações do terreno e vetoriais fornece restrições adicionais à distribuição da vegetação. Nesse sentido, os polígonos de floresta são utilizados para delimitar onde as árvores são colocadas, enquanto lagos, geleiras e estradas não devem conter plantas. A distribuição (Seção 4.4) avalia os ecótipos individualmente e de forma independente dos outros, o que é atraente para uma implementação da GPU.

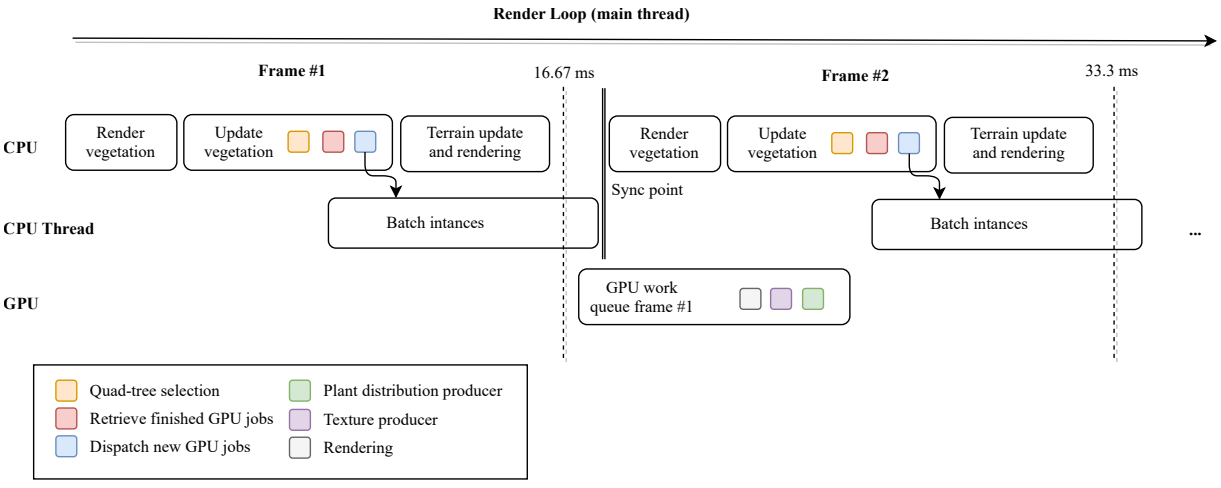
O terreno fornece várias restrições que devem ser consideradas ao distribuir as plantas. Para fornecer as informações do terreno durante a distribuição, e devido à grande escala da paisagem, é usado um algoritmo LOD de terreno com base em uma quad-tree (Seção 4.2). Cada nó quad-tree contém um *payload* que representa o conjunto de texturas (ou seja, heightmap, splatmap e campo de distância) e objetos usados para renderização. Os payloads são gerados em tempo de execução usando compute shaders assíncronos (Seção 4.3). A quad-tree do terreno é utilizada para gerenciar e armazenar a vegetação, conforme ilustrado na Figura 11. As principais

Figura 11 – Organização do framework. Cada nó da quad-tree armazena um conjunto de texturas geradas sob demanda a partir do mapa de altura e dos dados vetoriais da área.



funcionalidades do framework são implementadas na CPU pelo *gerenciador de requisições*, que é responsável por selecionar os nós da quad-tree que devem receber vegetação, solicitar a distribuição baseada em GPU e renderizar os resultados. O fluxo de trabalho de um frame é mostrado na Figura 12. Primeiro, uma travessia na quad-tree determina os nós do terreno dentro do frustum de visão que deve receber vegetação. Para cada um dos nós selecionados, as texturas de payload são geradas (Seção 4.3) e armazenadas na memória da GPU. Uma vez que as texturas estejam disponíveis, o sistema distribui dinamicamente as plantas nos nós selecionados (Seção 4.4) e realiza uma leitura de volta à memória principal depois de concluída. Finalmente, as instâncias geradas são atribuídas aos seus respectivos nós da quad-tree, finalizando assim o processo de distribuição.

Figura 12 – Fluxo de trabalho para distribuição e renderização de vegetação. Considerando um tempo de renderização de 16,67 ms por frame.



O framework lida com arbustos e árvores de forma diferente da grama por vários motivos. Primeiro, para permitir regras de distribuição mais elaboradas sem aumentar o custo por frame, esse processo é executado uma vez por nó para árvores e arbustos. Em segundo lugar, as árvores e os arbustos são lidos de volta à CPU para permitir a colisão física com outros objetos (por exemplo, veículos) e para renderização (seleção de LOD). O processo de leitura para a CPU e loteamento é caro para a grama devido à sua maior densidade. A grama emprega uma abordagem mais simples que é executada diretamente no shader de vértice a cada frame, conforme descrito na Seção 4.5.2.

Um procedimento de *loteamento dinâmico* é executado em cada frame, onde todas as instâncias de árvores e arbustos são agrupados em buffers de desenho por ecótipo e LOD para a renderização, conforme descrito na Seção 4.5. Os lotes são gerados em paralelo, como ilustrado na Figura 12.

4.2 Representação do Terreno

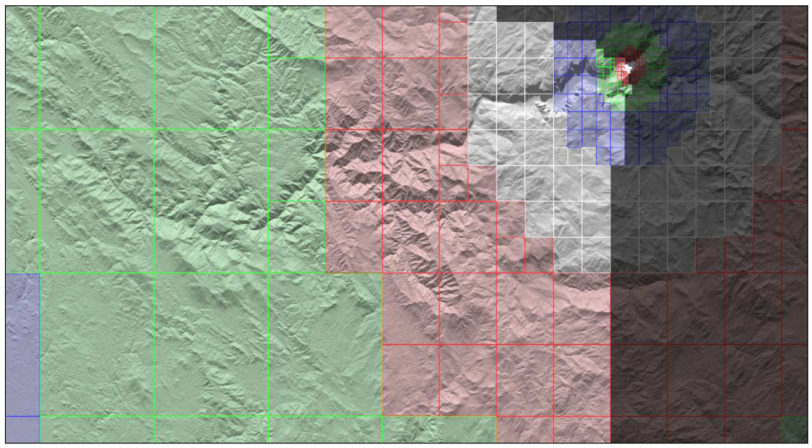
Nessa seção, descrevemos as técnicas usadas para renderização do terreno e as informações armazenadas na quad-tree. Além disso, mostramos como os dados vetoriais são consultados na GPU e as estruturas de subdivisão empregadas para melhoria de desempenho.

4.2.1 Quad-tree do Terreno

A renderização de terrenos virtuais é um tópico bem estabelecido em computação gráfica, portanto a revisão da extensa literatura sobre esse assunto está além do escopo dessa dissertação (ver (Cozzi e Ring, 2011; Mahdavi-Amiri et al., 2015; Pajarola e Gobbetti, 2007) para uma revisão completa sobre esse tópico). Nessa dissertação, utiliza-se o algoritmo de nível de detalhamento dependente da distância (CDLOD) (Strugar, 2009) utilizando uma quad-tree clássica dependente da visão. O autor propõe um sistema de LOD contínuo, onde cada nó da quad-tree contém um mapa de altura usado para deslocar a malha na GPU (vertex shader). A cada nível de LOD é atribuído um intervalo de visão e um intervalo de transição, que são usados pelo algoritmo de seleção, como mostrado na Figura 13. Outros algoritmos que fornecem suporte para amostras de altura e informações normais em GPU (Asirvatham e Hoppe, 2005; Bösch et al., 2009) também podem ser usados. Os nós do terreno são refinados na GPU e detalhes procedurais são adicionados usando o ruído Fractional Brownian Motion (FBM), como proposto por Frasson et al. (2016).

Cada nó armazaena um payload, que é representado por um conjunto de texturas tais como heightmap, normalmap, splatmap e campo de distância. Os payloads são gerados em tempo de execução e armazenados em cada nó correspondente. Uma coleta de memória é executada regularmente para liberar nós que não foram selecionados por um determinado tempo.

Figura 13 – Seleção dos nós da quad-tree (STRUGAR, 2009).



Os nós coletados são guardados em uma cache baseada em pilha para serem reutilizados quando novos nós são solicitados. Além disso, a geração dos payloads é dividida em alguns frames para reduzir o custo da instanciação do nó e fornecer um desempenho mais suave.

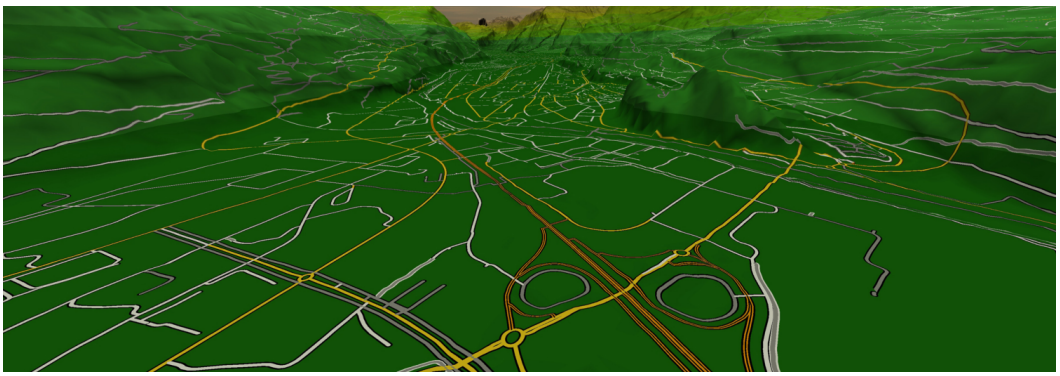
4.2.2 Dados Vetoriais

A renderização do terreno inclui o desenho de recursos naturais e feitos pelo homem, como estradas, rios, lagos e florestas. Esses elementos são codificados em um conjunto de linhas vetoriais e polígonos. Nós nos inspiramos na abordagem de Bruneton e Neyret (2008), que rasteriza texturas dos vetoriais por nó. Em seu trabalho, a textura *footprint* é usada para impor restrições no terreno (ex.: beira de estradas) e a textura de *aparência*, que representa o material da superfície, é mapeada no solo. A principal diferença em relação à nossa abordagem é que os dados vetoriais não são codificados em malhas para gerar os nós. Ao invéz disso, no nosso framework os dados vetoriais são subdivididos e enviados para a GPU, onde as interseções são calculadas usando testes de distância analítica e de ponto em polígono.

A abordagem empregada nessa dissertação é uma versão simplificada da abordagem de renderização de linha proposta por Thöny et al. (2017). Os autores propuseram subdividir os dados vetoriais em uma grade 2D de árvores BVH contendo segmentos de linha. A estrutura é consultada na GPU para renderização, em que cada amostra em espaço do mundo é mapeada para a estrutura de dados, e a cor é aplicada usando a distância do pixel até o segmento de linha mais próximo. A solução suporta apenas dados de linha, como mostrado na Figura 14. Nós estendemos sua abordagem para suportar polígonos, adaptando o nosso trabalho anterior (ENGEL ET AL., 2016) para consultas de ponto-em-polígono baseadas em GPU.

Nessa dissertação, utiliza-se um hash espacial 2D estendido de Pozzer et al. (2014) para indexar linhas e polígonos. A estrutura é consultada na GPU para gerar as texturas dos nós (ver

Figura 14 – Abordagem de renderização de linhas vetoriais proposto por Thöny et al. (2017).



Sec. 4.3).

Para as consultas no hash espacial, o primeiro passo é usar a função hash para determinar a célula na qual a posição atual em espaço de mundo está contida. A partir do id da célula, as linhas relevantes são localizadas para determinar a distância para a linha mais próxima. Para polígonos, um teste de ponto em polígono é realizado retornando a distância até a borda do polígono mais próxima do ponto (caso ele esteja contido). Os dados vetoriais são organizados em *camadas*, cada uma representada por um conjunto de dados vetoriais. As camadas são avaliadas de acordo com seu valor de precedência, que determina a ordem em que os elementos devem ser desenhados. A estratificação é essencial para resolver conflitos de sobreposição durante a travessia.

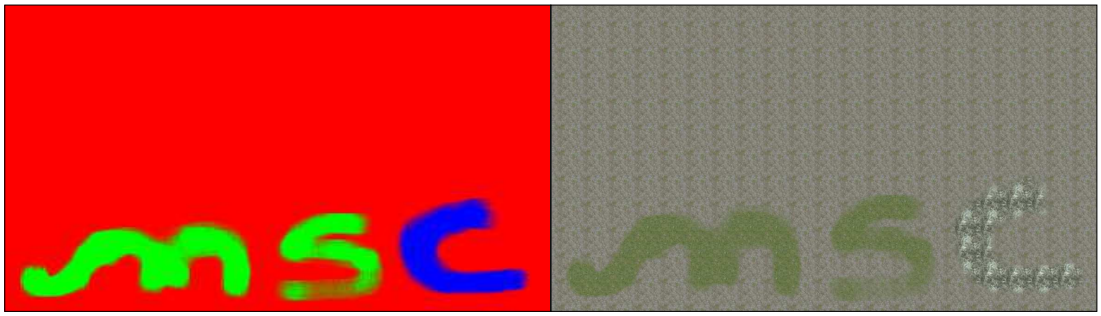
4.3 Geradores de Texturas

Os geradores são compute shaders projetados para gerar várias texturas por nó, tais como o heightmap, normalmap, splatmap e campo de distância. Cada textura do payload tem um respectivo gerador que as calcula em tempo de execução usando as características do terreno e dados vetoriais. O *heightmap* é gerado e refinado a partir do mapa de altura original. Esse gerador consulta os dados vetoriais, deforma o terreno (se necessário) e adiciona detalhes de ruído procedural. O *normalmap* é extraído do mapa de altura usando as derivadas parciais da superfície do terreno.

A textura de *splatmap* indica os materiais e onde eles são usados na superfície do terreno, como mostrado na Figura 15. Nesse exemplo, o splatmap contém três canais (RGB), cada um correspondendo a um material do chão. Essa intuição é extrapolada para suportar texturas de rocha, grama e chão, usando regras procedurais para determinar o material da superfície. Os parâmetros considerados são as características de inclinação, altura e dados vetoriais.

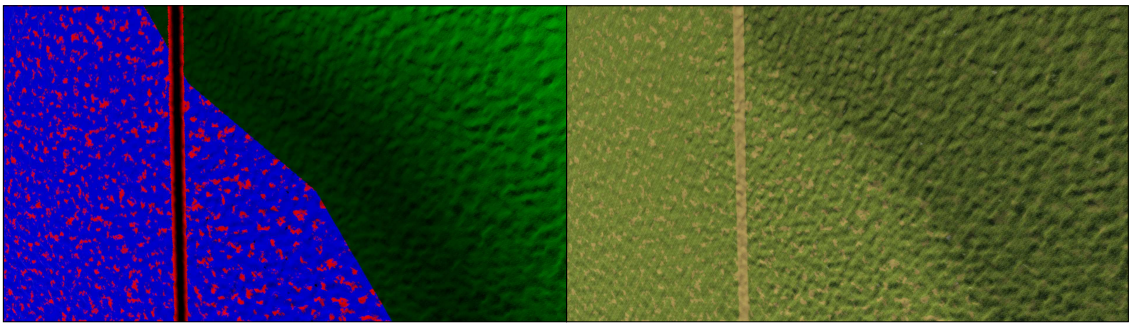
O *campo de distância* (DF) é uma textura adicional que contém informações de dados vetoriais. Cada canal da textura armazena a distância para a linha e polígono mais próximos, e

Figura 15 – Um exemplo de splatmap (esquerda) e do mapeamento desse para a textura do material (direita).



uma máscara indicando o(s) tipo(s) de dados vetoriais presentes. A máscara de bits permite que cada pixel pertença a um ou mais elementos ao mesmo tempo (ex.: dentro de um rio que flui para um lago). Por exemplo, utiliza-se máscaras para identificar várias camadas vetoriais, como florestas, corpos d'água e geleiras. Atualmente, o sistema suporta 32 máscaras diferentes. A textura de DF complementa o splatmap para fornecer informações adicionais para distribuição e renderização. Um exemplo da combinação é mostrado na Figura 16, onde o splatmap procedural é associado com o DF para gerar estradas e fornecer a mistura de material de grama para o solo da floresta. Da mesma forma, as texturas geradas são usadas no processo de distribuição (Seção 4.4). As texturas descritas podem ser associadas a aquelas pintadas manualmente por artistas (embora não abordemos isso no trabalho atual).

Figura 16 – Associação de splatmap e de campos de distância. Uma combinação do splatmap e DF onde uma estrada atravessa um polígono de floresta (esquerda) e o resultado final, onde os respectivos materiais são amostrados e interpolados (direita).



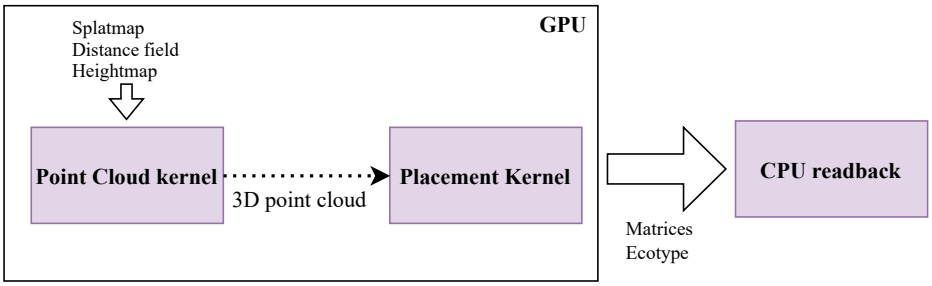
4.4 Distribuição de Plantas

O gerador do ecótipo recebe as configurações splatmap, heightmap e plant para determinar a distribuição adequada. As configurações das plantas representam suas regras de colocação, determinando a probabilidade de cada elemento existir em um determinado local. A colisão com os dados vetoriais é verificada por meio do DF e do splatmap. Similarmente a Muijden (2017), a distribuição é dividida em dois kernels de compute shader: o *núvem de pontos* e o *posicionamento*, como ilustrado na Figura 17.

No kernel de *núvem de pontos*, cada thread calcula uma posição de amostra dentro do nó considerando um deslocamento aleatório. Esse kernel avalia regras globais que definem onde as plantas *não* devem ser colocadas. Com base nessas regras, os pontos são mascarados contra as informações de splatmap, DF e terreno e descartados, se necessário. Por exemplo, este passo rejeita amostras que caem dentro de estradas e lagos. As plantas geradas são colocadas em um buffer de pontos que pode ser compartilhado com outros geradores. Observe que nem todas as threads geram um ponto, pois alguns elementos podem cair em áreas onde a vegetação não cresce. Como tal, a escolha de um buffer de *append* é conveniente para produzir apenas os itens necessários e reduzir as demandas de transferência e processamento de dados.

A saída do kernel de nuvem de pontos é um buffer de pontos 3D que contém apenas os elementos que passaram nos testes de distribuição. No entanto, como a CPU não sabe o tamanho exato do buffer (a menos que um readback seja executado), o tamanho do buffer é copiado e passado para um kernel auxiliar que é despachado para calcular o número de grupos de threads necessários para o kernel de posicionamento. Finalmente, o kernel de posicionamento é enviado *indiretamente* recebendo como argumentos o buffer preenchido pelo kernel auxiliar. Nesse caso, os argumentos de execução (ou seja, número de grupos e threads) para a execução são lidos diretamente na GPU, enquanto a CPU não armazena os valores.

Figura 17 – Processo de distribuição em GPU.



O kernel de *posicionamento* recebe a nuvem de pontos e é responsável por adicionar mais informações para que os elementos possam ser renderizados apropriadamente no mundo virtual. O resultado é uma estrutura (por instância) contendo uma matriz 4x4 completa com

informações de posição, rotação e escala, juntamente com o ecótipo da planta. Os parâmetros calculados para cada planta levam em consideração os valores de ruído fractal, pois são capazes de representar padrões encontrados na natureza (HARGROVE ET AL., 2002). Esse buffer é lido de volta à CPU após a conclusão.

Para determinar o ecótipo de cada ponto, utiliza-se a abordagem de Hammes (2001), onde uma probabilidade baseada em pesos de ecotipos é calculada em pontos no terreno. O ecótipo com a maior chance é escolhido. Utiliza-se regras básicas (valores) para definir a probabilidade de uma planta pertencer a um material específico do solo, faixa de inclinação, faixa de altura e proximidade à água. Os valores de inclinação, altura e distância à água são compostos por uma distância mínima e máxima e um fator de suavização. Para calcular a probabilidade normalizada X de uma variável V (ex.: a inclinação, altura) de estar dentro de um intervalo, primeiro V é amostrado na posição atual, então o valor é calculado como $X = abs((V - midRange)/range)$, e o intervalo é distribuído usando um fator de suavização S aplicado sobre o valor final $X = 0.5^{X^S}$ (HAMMES, 2001). O produto dessas possibilidades dá a probabilidade final de um ecótipo existir em um determinado ponto.

A cadeia de compute shaders descrita (mostrada na Figura 17) permite manter todos os cálculos na GPU e somente realizar leituras para a memória principal quando necessário. Além disso, a leitura é realizada de forma assíncrona assim que os cálculos são concluídos, evitando paradas custosas no pipeline. As instâncias inseridas são armazenadas em seus respectivos nós e loteados para renderização quando o nó é selecionado. O número de nós em que o processo de geração é executado é limitado a um valor máximo por frame para fornecer um desempenho estável.

4.5 RENDERIZAÇÃO

A renderização de plantas apresenta desafios relacionados ao número de chamadas de desenho, a quantidade de geometria e a iluminação. Uma *chamada de desenho* contém todas as informações necessárias para renderizar um ou mais objetos (ex.: estados, texturas, buffers). Essa informação é convertida em uma série de buffers de comandos que a GPU pode executar. No entanto, a sobrecarga da CPU para gerar os buffers de comandos de hardware ainda representa gargalos de desempenho com um grande número de chamadas de desenho. Uma alternativa é o uso de *instanciação em GPU*, que permite desenhar várias instâncias do mesmo objeto com uma única chamada de desenho (WRIGHT ET AL., 2010). Outra abordagem, conhecida como *renderização por agrupamento*, consiste em agrupar várias malhas para formar uma maior (HAAR E AALTONEN, 2015). Como a renderização de plantas depende do desenho de muitas instâncias da mesma malha em cada frame, a instanciação em GPU é a abordagem ideal para processá-las.

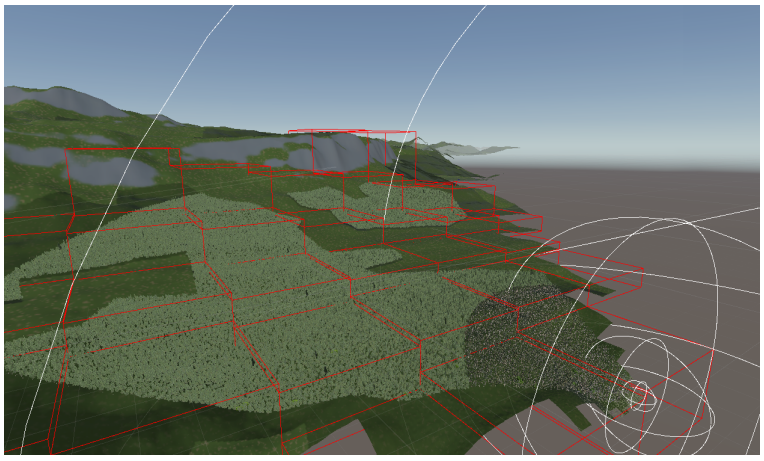
A solução foi implementada em um pipeline de renderização deferred. Os fragmentos

gerados para as plantas são testados com alfa (*alpha-test*) usando um valor de limite α. Nesse caso, o valor alfa do fragmento atual *color.a* é testado em relação ao limite e descartado se $color.a < \alpha$. O teste alfa permite renderizar as plantas sob o paradigma deferred. As seções a seguir descrevem como nossa estrutura lida com a renderização dos diferentes tipos de plantas.

4.5.1 Árvores e Arbustos

O grande número de plantas e a quantidade de detalhes exige LOD eficiente. Outro importante benefício do LOD é a redução do aliasing resultante da rasterização de muitos triângulos no mesmo pixel. O nível apropriado de detalhe é escolhido de acordo com a distância da câmera. Um exemplo de árvore LOD é mostrado na Figura 9, onde cada imagem representa uma geometria simplificada do mesmo modelo. A Figura 18 ilustra as faixas de LOD no mundo virtual.

Figura 18 – Seleção de LOD. Cada LOD recebe uma faixa de distâncias (em branco) da câmera.

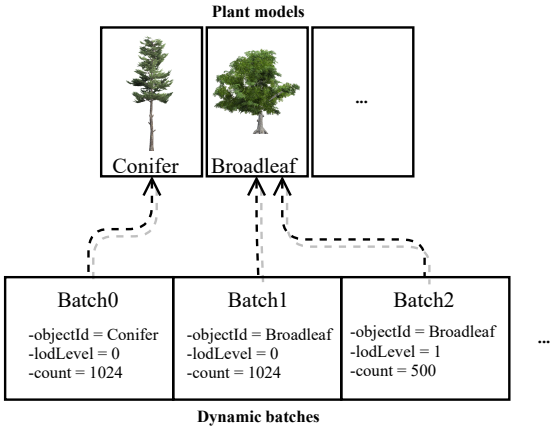


A partir do conjunto de modelos de plantas (ou seja, biblioteca de plantas) recebidas como entrada, as malhas e materiais de cada nível de detalhamento disponíveis são extraídas. O resultado é um array de protótipos de árvore, onde apenas uma instância de cada modelo é instanciada na cena. Assume-se que o material já esteja configurado (ex.: texturas são atribuídas) e contém todas as propriedades necessárias para a renderização. Portanto, o renderizador abstrai a representação da planta (ou seja, geometria ou billboard). A estrutura resultante é:

```
struct {
    int lodLevels;
    Mesh[] meshes;              //tipicamente uma por LOD
    Material[][] materials; //materiais por (sub)malha
} TreePrototype;
```

Para tirar proveito da instanciação em GPU, os objetos devem compartilhar a mesma malha e material. Observando essas restrições, desenvolveu-se um sistema de geração de lotes para montar instâncias de objetos em listas de desenho de acordo com o ecótipo da planta, material e nível de detalhamento apropriado, conforme ilustrado na Figura 19. Como os lotes mudam continuamente (ex.: quando a câmera se move), eles precisam ser atualizados a cada frame.

Figura 19 – Exemplo de lotes de desenho. Cada lote corresponde à várias instâncias de uma planta em um determinado LOD



A função de geração de lotes recebe os nós selecionados do frame atual, lê todos as plantas e gera os lotes apropriados. Para cada LOD de cada planta, arrays de matrizes são preenchidas até que todos os modelos sejam atribuídos. Os lotes são divididos em matrizes de até 1023 elementos (o número máximo de instâncias por desenho). Apesar da simplicidade do algoritmo, ele itera através de um grande número de objetos (normalmente dezenas de milhares), consumindo assim uma quantidade considerável de processamento. O algoritmo foi adaptado para que o lote fosse executado em paralelo entre os frames. Enquanto o frame atual é concluído, o framework gera os lotes de forma assíncrona para serem renderizados no próximo frame. A Figura 12 ilustra a execução paralela e o ponto de sincronização antes do desenho do próximo quadro. Efetivamente, as plantas são processadas um frame mais tarde do que elas são selecionadas. O algoritmo de lotes pode ser facilmente adaptado para trabalhar com modelos genéricos.

4.5.2 Grama

No framework proposto, a grama é renderizada como billboard; sendo assim, a mesma malha é utilizada para todas as instâncias (ex.: um quad de dois lados). Como a grama é muito mais abundante do que as outras plantas, é conveniente contornar a CPU para reduzir a

sobrecarga de desenho. Abordamos isso usando a instanciação *indireta*, onde as posições da planta são calculadas diretamente no shader de vértice. A CPU apenas comanda a GPU para renderizar a grama por nó, usando uma contagem de plantas de tamanho fixo. Ao invés de remover plantas que intersectam dados vetoriais (ex.: em uma estrada), elas são dimensionadas para zero para evitar a geração de fragmentos para as plantas. No entanto, o shader de vértice ainda deve realizar os cálculos para determinar a posição.

As posições de grama são calculadas usando o id da instância da malha, que é o índice da instância da planta particular à qual o vértice pertence. O id da instância é o mesmo para todos os vértices da mesma malha e contém um valor entre zero e o número de elementos enviados ao comando de desenho específico. Como as amostras são avaliadas uniformemente em patches $N \times N$ dentro do nó, o *uv* local (dentro do nó) é obtido como $uv = \frac{0.5}{N} + (\frac{floor(\frac{SV\_InstanceID}{N})}{N}, \frac{SV\_InstanceID \bmod N}{N})$. Com a *uv* da amostra, a posição e tamanho do nó, é possível reconstruir a posição no espaço de mundo. Além disso, a *uv* é usada para avaliar as texturas splatmap e DF.

Para alcançar esse nível de personalização, um shader específico foi desenvolvido para a grama. As texturas de grama são agrupadas em um array de texturas na CPU e selecionadas no shader de fragmento de acordo com uma função de ruído. A distribuição da grama é uniforme e avalia regras mais simples de que são aplicadas a árvores e arbustos, onde a mesma função de ruído é usada para gerar rotação e escala aleatória de plantas de grama. A posição no espaço de mundo da amostra é usada como a semente para as funções de ruído para manter a consistência entre os frames.

5 RESULTADOS

Para avaliar a aplicabilidade do framework em relação ao desempenho e à qualidade visual, um protótipo foi desenvolvido. Como a premissa é de que é viável distribuir as plantas em tempo de execução, o desempenho é avaliado. Além disso, são demonstrados os resultados relacionados à qualidade visual e de distribuição alcançada com a abordagem proposta.

O protótipo foi desenvolvido usando a Unity Engine (UNITY, 2018) e a linguagem de programação C#. Como a engine atualmente não suporta leitura assíncrona da GPU, a biblioteca AsyncTextureReader (SKALSKY, 2017) foi utilizada, fornecendo acesso de baixo nível à API gráfica (DirectX 11) de onde essa operação pode ser executada. A engine não oferece suporte nativo para dados SIG; por isso, uma ferramenta auxiliar foi desenvolvida usando C++/Qt para pré-processar os dados SIG em formatos próprios para facilitar o processo de carregamento dentro da engine.

O cenário de teste foi gerado a partir de um conjunto de dados do mundo real que compreende parte da Suíça em uma área de 227kmx227km de extensão. Os dados vetoriais de polígonos representam florestas, lagos e geleiras, enquanto os conjuntos de dados de linha compreendem estradas e rios. O MDE foi obtido do repositório USGS (UNITED STATES GEOLOGICAL SURVEY, 2017), e os dados vetoriais correspondem ao conjunto de dados *Swiss Map Vector 500* adquirido no Escritório Federal Suíço de Topografia (swisstopo) (SWISSTOPO, 2016). Os modelos arbóreos e as texturas de gramíneas foram obtidos na Unity Asset Store, enquanto a textura da grama e a textura do solo seco foram obtidas de Quixel Megascans. Os resultados foram avaliados em uma máquina com uma CPU Ryzen 1600X 3.2 GHz, 16 GB de RAM e uma GPU GeForce 1070 (8 GB).

5.1 DESEMPENHO

O desempenho foi avaliado em um sobrevoo de 120 s em um percurso de 5,96 km no terreno, enquanto os tempos de renderização foram capturados (mostrado no vídeo acompanhante). A Figura 20 ilustra os resultados obtidos durante a animação e a Tabela 2 mostra as configurações usadas durante os benchmarks. O gráfico mostra os tempos de execução para o caminho com diferentes configurações, considerando os tempos de atualização (U) e de renderização (R): o terreno somente (UR); terreno (UR) com atualização de vegetação (U); e o terreno (UR) e vegetação (UR). Os tempos de atualização são representados pela geração de nós, seleção, e o loteamento dinâmico de plantas (no caso de árvores e arbustos), enquanto os tempos de renderização se relacionam ao desenho, culling, sombra e antialiasing temporal. Além disso, o gráfico mostra a altura (em relação ao terreno) e a velocidade quando as amostras foram obtidas, pois estão diretamente relacionadas aos resultados. A animação permite observar o desempenho quando a velocidade da câmera varia de 42 km/h a 600 km/h e altura de 1 a 300 m. A projeção e recebimento de sombras são ativadas para objetos abaixo de 250 m da

Tabela 2 – Configurações das plantas usadas nos testes. Para cada tipo de planta, mostra as configurações usadas para a distribuição e renderização.

Planta	Alcance de Visão	Tam. de Bloco	Tam. do Nó	Nível da Quad-tree
Trees	2200 m	64x64	444x444 m	9
Bushes	500 m	32x32	222x222 m	10
Grass	100 m	120x120	55x55 m	12

câmera (esses objetos são renderizados várias vezes). As sombras são renderizadas usando o mapa de sombra em cascata integrado da Unity.

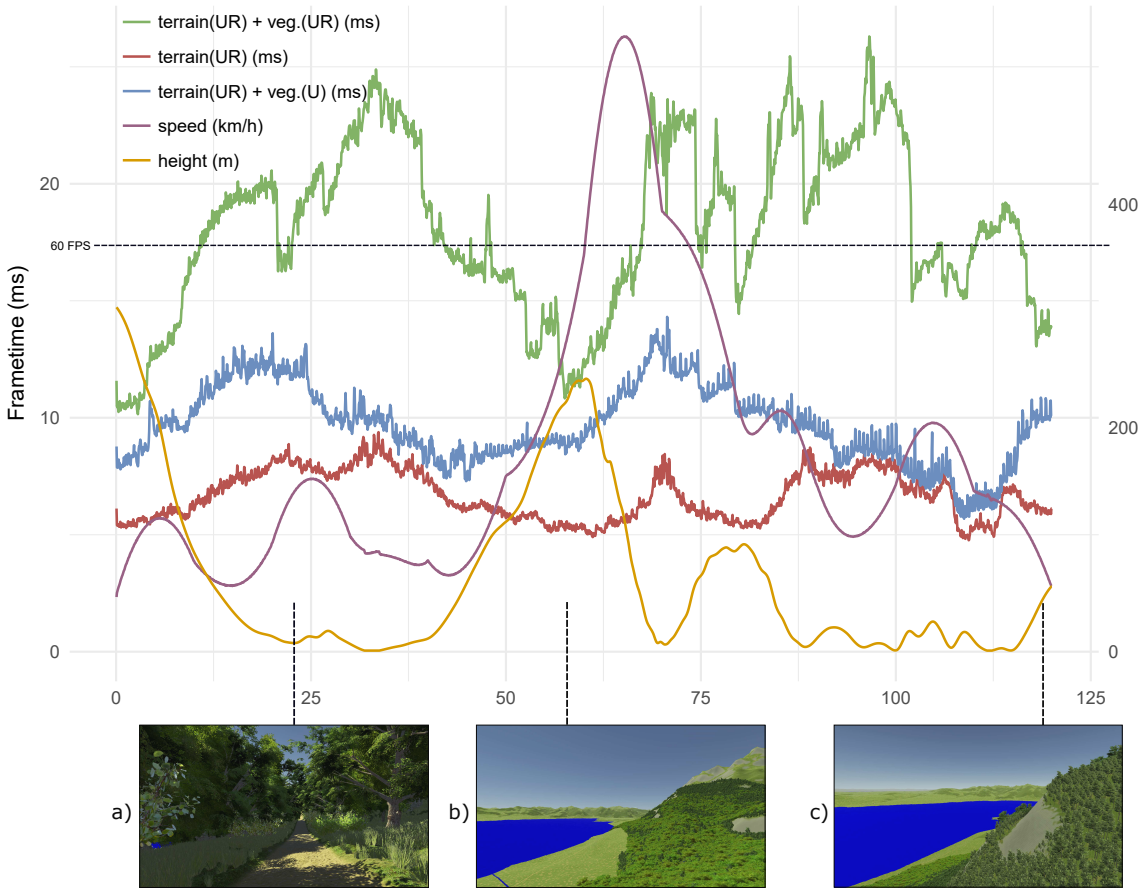
O número de nós gerados por frame para a renderização do terreno e a distribuição da vegetação é um fator impactante no desempenho da aplicação. De fato, quando a câmera se move rapidamente em baixas altitudes, um alto número de níveis da quad-tree deve ser gerado, causando instabilidade no desempenho (Figura 20 (a)). Por outro lado, o efeito é imperceptível ao mover a câmera longe do chão, mesmo em altas velocidades (Figura 20 (b)). Esse comportamento é observado na diferença entre a atualização da vegetação e os tempos de renderização, pois, quando mais próximo do solo, o nível de detalhe é maior e a renderização ocupa uma parte mais significativa do tempo de frame. Por outro lado, em altitudes mais elevadas, mais plantas são visíveis, embora com menos detalhes (Figura 20 (c)).

A distribuição e a renderização da vegetação correspondem, em média, por 10 ms de cada frame. O resto é dividido no terreno e outros custos gerais (ex.: pós-processamento). O benchmark demonstrou que o sistema executa a distribuição e a renderização em tempo real. Os tempos de cada frame permaneceram estáveis mesmo com a leitura das distribuições de plantas para a CPU. No entanto, com velocidades de até 500 km/h, o tempo por frame ficou abaixo de 25 ms. O tempo médio de frames de toda a animação é de 17,9 ms.

Durante o sobrevoo, 60 mil plantas são renderizadas a cada frame (em média), com picos próximos a 90 mil. Cada lote resulta em uma chamada de desenho e os objetos que se encontram em áreas sombreadas requerem um adicional para cada cascata de sombra a que pertencem. Observamos picos de até 12 ms de CPU dedicados ao sistema de loteamento, pois ele escala com o número de elementos. Ele é responsável pela parte mais significativa da atualização da vegetação, pois opera em um grande número de instâncias. Mesmo com a paralelização do método entre frames, a thread principal ainda aguarda até 5 ms até que termine. O número de instâncias de cada lote está atualmente limitado ao tamanho da memória constante (veja Sec 2.2) onde as matrizes de transformação são armazenadas, que neste caso são 64kb (1023 matrizes). Trabalhos futuros devem otimizar o sistema de lotes, pois representa o principal gargalo no framework. Além disso, outros métodos que não estão limitados ao tamanho da memória constante podem ser avaliados, como a instanciação indireta utilizada para a renderização

da grama.

Figura 20 – Desempenho durante o sobrevoo. O gráfico mostra os tempos de frames da animação considerando os tempos de atualização (U) e de renderização (R). O eixo direito representa a velocidade e a altura.



A leitura assíncrona para a CPU tem o benefício de permitir recuperar os dados sem causar bloqueios no pipeline. Em geral, os resultados de distribuição estão disponíveis em dois frames, embora possa levar de 3 a 5 frames quando um grande número de solicitações está sendo processado. Se houver necessidade de obter os resultados instantaneamente (ex.: física), uma leitura bloqueante pode ser realizada. Como a avaliação das amostras de plantas não depende de nada além das configurações das plantas e das texturas de payload que são mantidas na GPU, a abordagem se beneficia muito da implementação da GPU. Abordagens de distribuição mais interativas, como o local-para-global, descritas na Seção 3.1 podem ser complexas de implementar na GPU, pois (em geral) dependem do feedback contínuo da distribuição e têm dependências de vizinhança. A distribuição simplificada usada para a grama não é armazenada ou lida para a CPU, ela é executada em todos os frames e oferece pouca sobrecarga na CPU.

Como os geradores operam em tempo de execução, o seu escalonamento impacta diretamente na estabilidade da exploração do mundo virtual. Vários mecanismos foram adotados

para obter tempos de frames consistentes. O primeiro é armazenar os nós em uma cache para que quando um novo for solicitado, ele retorne um da cache ao invés de sempre alocar um novo. Em segundo lugar, a geração do payload dos nós é dividida em alguns frames, onde o heightmap e o normalmap são os primeiros a serem gerados. Dois frame depois, o splatmap e DF são despachados, e então o nó pode ser selecionado para renderização ou distribuição de vegetação. O número de solicitações de distribuição ativas foi limitado a quatro ao mesmo tempo. Esses mecanismos são essenciais para evitar que a GPU tenha um trabalho excessivo em um frame enquanto fica ocioso em outros.

Outro fator importante é que a linguagem C# tem um sistema coletor de lixo que é executado periodicamente para identificar objetos não referenciados e memória heap livre. Esse sistema geralmente causa oscilação repentina de desempenho quando muitas alocações são executadas ou grandes blocos de memória são liberados. Portanto, o caching de objetos é essencial para a manutenção adequada do desempenho. Para reduzir os efeitos causados pela alteração na velocidade e fornecer tempos de frames mais suaves, a previsão deve ser usada para distribuir as tarefas de geração de maneira uniforme entre os frames.

5.2 Resultados Visuais

A qualidade visual do protótipo foi avaliada usando referências do mundo real. A área recriada está localizada nos Alpes suíços, onde o ecótipo dominante são árvores coníferas nas montanhas com floresta estacional decidual nas áreas mais baixas. O cenário do protótipo contém dois modelos de árvores, dois para arbustos e um para grama. As árvores têm quatro LODs cada, enquanto os arbustos e grama só um. Os modelos são muito detalhados, por exemplo, o modelo Broadleaf, que é mostrado na Figura 9, possui 7188 triângulos em seu nível mais detalhado. O mecanismo LOD desempenha um papel crucial na redução da quantidade de geometria e permite distâncias de visualização mais longas. No entanto, a implementação atual carece de suporte para transições suaves entre os LODs, que devem ser abordados em trabalhos futuros.

A distribuição apresentou resultados plausíveis com um grau médio de controle do usuário. Como pode-se notar nas Figuras 21, 22 e 24, a distribuição está próxima do uniforme dentro de florestas, mantendo espaço para trânsito e respeitando as estradas. Além disso, as árvores que crescem perto de áreas com rochas tendem a ter raízes superficiais e crescem menos que as outras. Em nosso framework, cada ecótipo contém um conjunto de regras de posicionamento definidas pelo usuário. Portanto, depende de um artista para configurar cada planta que pode ser um processo interativo. Esse processo é executado manualmente e pode precisar de alterações para diferentes cenários. Portanto, a falta de uma interface de edição intuitiva é uma limitação da implementação atual.

A aplicação de dados vetoriais para modular a distribuição da vegetação permite uma

Figura 21 – Distribuição de plantas. A referência do mundo real (esquerda) e a mesma área gerada com o nosso framework (direita). Observe a predominância de árvores de folha caduca no sopé e pinheiros nas colinas.

personalização de alto nível. No entanto, dependendo da nitidez dos dados vetoriais, a delimitação pode parecer artificial, já que na natureza não é precisa. Esse efeito pode ser observado quando um curso de rio não segue o caminho correto no mundo 3D, fazendo com que a vegetação faça o mesmo. Na Figura 23 demonstramos como nosso framework permite realizar a edição básica dos dados e como a vegetação se adapta às novas restrições. Outras ferramentas de edição devem ser desenvolvidas para permitir a correção de tais detalhes e para fornecer um controle preciso ao usuário.

Figura 22 – Vista de cima de uma distribuição de plantas. A imagem de referência (esquerda) e a gerada pelo nosso sistema (direita).

As abordagens locais-para-globais geralmente não consideram o fato de que os seres

humanos modificam a paisagem para adicionar estradas, construções e outros objetos. Por outro lado, aqueles que consideram normalmente exigem etapas de pré-processamento offline para calcular os resultados, ou ficam limitados a mapas de densidade pintados por artistas. Nossa estrutura armazena informações semânticas e fornece mais detalhes para personalizar a distribuição em tempo real. Ao editar os dados vetoriais, o terreno, o payload e as distribuições são atualizados instantâneamente para acomodar as alterações. Além disso, a integração do nosso sistema com elementos colocados à mão é simples, pois eles podem ser inseridos em seus respectivos nós.

A resolução das texturas de payload afeta a qualidade visual do terreno. Para os experimentos, foi utilizado um splatmap de 129x129 pixels e uma textura DF, pois eles são mais suscetíveis a aliasing, enquanto o heightmap e o normalmap têm 35 e 33 pixels, respectivamente. As regras de geração de splatmap podem ser personalizadas para melhorar a qualidade visual, porém ele é limitado a 8 canais (ou seja, materiais). A textura do campo de distância é útil para realizar efeitos de interpolação, por exemplo, entre a estrada e a grama, assim como outros efeitos. O splatmap pode ser criado por um artista e integrado ao sistema, onde os nós editados devem ser salvos no disco. As texturas de carga útil requerem, em média, 1,1 GB de memória da GPU (considerando 4k nós na memória). Embora seja uma quantidade considerável de memória, acreditamos ser uma troca valiosa pela modularidade de cada nó. Os buffers de GPU utilizados para as posições de árvores são desalocados após a distribuição da vegetação e, em seguida, as instâncias geradas são armazenadas em seus respectivos nós (na memória principal).

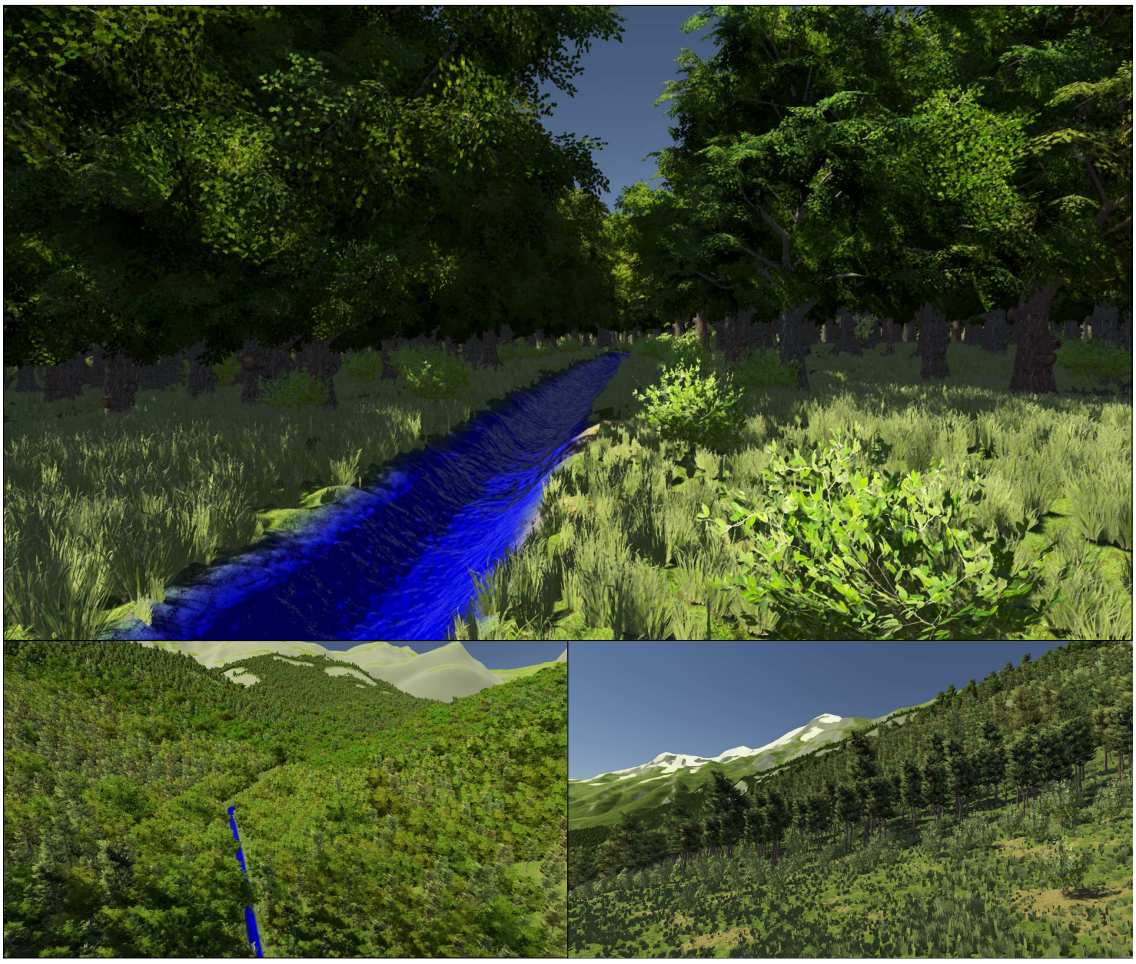
Figura 23 – Edição dos dados de vetoriais. O rio corria ligeiramente para cima (esquerda). Após a edição, o terreno, a grama, as árvores e os arbustos foram atualizados adequadamente (direita).

Uma das vantagens do nosso sistema é de que com o terreno, dados vetoriais e uma biblioteca de plantas, é possível preencher rapidamente o ambiente considerando várias restrições. Além disso, o framework não impõe limitações aos modelos de plantas, sendo projetado para ser genérico e para suportar plantas com vários LODs ou mesmo modelado como simples billboards. As densidades apresentadas na Tabela 2 demonstram um alto número de elementos co-

locados no terreno, o que poderia resultar em problemas de armazenamento se pré-processados. Por exemplo, no caso das árvores, em média existe uma planta por $48m^2$, e cada uma ocupa 72 bytes de memória. Se o terreno fosse coberto 30% por florestas, a contagem final resultaria em mais de 322 milhões de instâncias que exigiriam 21,5 GB de espaço de armazenamento.

Figura 24 – Resultados gráficos. As figuras mostram uma distribuição irrestrita (superior), limitada por rochas (parte inferior esquerda) e por uma estrada cruzando a floresta (parte inferior direita).

Figura 25 – Resultados das regras de distribuição. As árvores de folhas largas e os arbustos verdes são mais propensos a crescer em baixas altitudes, encostas baixas e perto da água (superior, inferior esquerda). Por outro lado, pinheiros predominam em altas altitudes (canto inferior esquerdo, canto inferior direito).



6 CONSIDERAÇÕES FINAIS

A vegetação é um dos elementos mais importantes em paisagens naturais. Apresentamos um framework para a população procedural de mundos virtuais com vegetação. Com uma entrada limitada, o sistema é capaz de distribuir e renderizar as plantas no mundo virtual em tempo real. O framework abrange um amplo escopo de restrições para interagir com o terreno e outros fatores importantes, como estradas e água. Hoje em dia, a disponibilidade de dados SIG permite que os sistemas de distribuição de plantas, ao invés de tentar gerar a vegetação de uma maneira inteiramente procedural, utilizem as características espaciais conhecidas para inferir informações do contexto. Nessa dissertação, demonstramos como os dados vetoriais podem ser utilizados em associação com o terreno para melhorar a distribuição das plantas. Além disso, os mesmos dados podem ser ativamente empregados para visualizar outros componentes do ambiente, como estradas, rios e lagos.

A distribuição de plantas apresentou um desempenho eficiente, aproveitando-se de um fluxo de trabalho baseado em GPU para avaliar e gerar posições de plantas. Em nossa estrutura, cada ecótipo contém um conjunto de regras básicas de posicionamento que são avaliadas durante a distribuição. Os resultados demonstraram uma disposição plausível de elementos no ambiente, seguindo as regras básicas definidas para cada planta e restrições globais de posicionamento. O framework permite densidades altas de vegetação, o que causaria impactos significativos no tamanho do arquivo se ele fosse pré-calculado. A abordagem é integrada ao sistema de terreno e mostra-se adequada para renderizar um grande número de plantas de alta qualidade em mundos 3D.

O framework possui uma ampla gama de aplicações em mundos virtuais 3D em geral. A geração automatizada é um recurso atraente que permite a população rápida de paisagens em grande escala com pouco controle do usuário. Com os recursos de computação das GPUs modernas, os pipelines de autoria tradicionais podem ser reprensados para aproveitar esses recursos e gerar conteúdo em tempo real. Seguimos essa tendência e demonstramos uma aplicação bem-sucedida em relação à distribuição procedural de plantas em ambientes de grande escala.

6.1 TRABALHOS FUTUROS

Nosso framework esboça um fluxo de trabalho para um sistema de vegetação totalmente funcional baseado em GPU, onde implementações modernas podem ser usadas para melhorar áreas mais específicas deixadas fora do escopo dessa dissertação (ex.: iluminação). Mais especificamente, a dinamicidade das populações de plantas deixa espaço para mais pesquisas sobre outros métodos rápidos para a distribuição de plantas baseada em GPU. As principais áreas são:

- **Desempenho:** A seleção de vegetação, a seleção de LOD e a geração de

lotes se beneficiariam de uma implementação da GPU, uma vez que as mesmas operações são realizadas em um grande número de elementos. Para reduzir o número de chamadas de desenho, a abordagem proposta por Haar e Aaltonen (2015) poderia ser avaliada;

- **Regras de distribuição:** Existem vários parâmetros que podem ser expostos para melhorar e expandir os recursos de distribuição (ex.: parâmetros dos recursos vetoriais, padrões de distribuição personalizados). No entanto, para aproveitar esse nível de personalização, uma interface de usuário intuitiva deve ser fornecida. A edição intuitiva é um dos problemas mais importantes, ainda por resolver, na modelagem procedural (SMELIK, TUTENEL, BIDARRA ET AL., 2014). A ferramenta baseada em nós apresentada por Muijden (2017) é uma poderosa ferramenta de edição para artistas;
- **Generalização do framework:** As ideias de distribuição de plantas podem ser extrapoladas para qualquer objeto cujas regras de distribuição possam ser representadas no sistema, como rochas, vida selvagem e galhos de árvores mortas. Muijden (2017) apresentou uma abordagem semelhante em um jogo comercial;
- **Iluminação:** A iluminação das billboards e a transição entre os LODs devem receber atenção para melhorar a qualidade visual. Abordagens como as propostas por Bruneton e Neyret (2012) e Behrendt et al. (2005) fornecem resultados de qualidade com amplos campos de visão e podem ser consideradas.

Referências

Alsweis, M. e O. Deussen. **Wang-tiles for the simulation and visualization of plant competition**. Em: *Advances in Computer Graphics*, pp. 1–11. 2006.

Asirvatham, A. e H. Hoppe. **Terrain rendering using GPU-based geometry clipmaps**. Em: *GPU Gems 2* 2, pp. 27–46. 2005.

Bayer, B. E. **An optimum method for two-level rendition of continuous-tone picutres**. Em: *IEEE International Conference on Communications*. Vol. 50, pp. 11–15. URL: http://ci.nii.ac.jp/naid/10022149628/en/. 1973.

Behrendt, S., C. Colditz, O. Franzke, J. Kopf e O. Deussen. **Realistic real-time rendering of landscapes using billboard clouds**. Em: *Computer Graphics Forum* 24.3, pp. 507–516. DOI: 10.1111/j.1467-8659.2005.00876.x. 2005.

Beneš, B., M. A. Massih, P. Jarvis, D. G. Aliaga e C. A. Vanegas. **Urban ecosystem design**. Em: *Symposium on Interactive 3D Graphics and Games on - I3D '11*, p. 167. DOI: 10.1145/1944745.1944773. 2011.

Bösch, J., P. Goswami e R. Pajarola. **RASTeR : simple and efficient terrain rendering on the GPU**. Em: *Proceedings EUROGRAPHICS Areas Papers, Scientific Visulization* February, pp. 35–42. 2009.

Boulanger, K., S. N. Pattanaik e K. Bouatouch. **Rendering grass in real time with dynamic lighting.** Em: *IEEE computer graphics and applications* 29.1, pp. 32–41. DOI: 10.1109/MCG.2009.14. 2009.

Bradbury, G. A., K. Subr, C. Koniaris, K. Mitchell e T. Weyrich. **Guided Ecological Simulation for Artistic Editing of Plant Distributions in Natural Scenes**. Em: *The Journal ofComputer Graphics Techniques* 4.4, pp. 28–53. 2015.

Bridson, R. **Fast Poisson disk sampling in arbitrary dimensions**. Em: *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*, 22–es. DOI: 10.1145/1278780.1278807. 2007.

Bruneton, E. e F. Neyret. **Real-time rendering and editing of vector-based terrains**. Em: *Computer Graphics Forum* 27.2, pp. 311–320. DOI: 10.1111/j.1467-8659.2008.01128.x. 2008.

Bruneton, E. e F. Neyret. **Real-time realistic rendering and lighting of forests**. Em: *Computer Graphics Forum* 31.2, pp. 373–382. DOI: 10.1111/j.1467-8659.2012.03016.x. 2012.

Ch'ng, E. **Realistic placement of plants for virtual environments**. Em: *IEEE Computer Graphics and Applications* 31.4, pp. 66–77. DOI: 10.1109/MCG.2010.42. 2011.

Chai, D., W. Forstner e F. Lafarge. **Recovering line-networks in images by junction-point processes**. Em: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1894–1901. DOI: 10.1109/CVPR.2013.247. 2013.

Chrisman, N. R. **What Does 'GIS' Mean?** Em: *Transactions in GIS* 3.2, pp. 175–186. 1999.

Cordonnier, G., E. Galin, J. Gain, B. Benes, E. Guérin, A. Peytavie e M.-P. Cani. **Authoring Landscapes by Combining Ecosystem and Terrain Erosion Simulation**. Em: *ACM Transactions on Graphics* 36.4. 2017.

Cozzi, P. e K. Ring. **3D engine design for virtual globes**. 1st. Natick, MA, USA: A. K. Peters, Ltd. 2011.

Decaudin, P. e F. Neyret. **Volumetric billboards**. Em: *Computer Graphics Forum* 28.8, pp. 2079–2089. DOI: 10.1111/j.1467-8659.2009.01354.x. 2009.

Deussen, O. e B. Lintermann. **Digital Design of Nature: Computer Generated Plants and Organics**. Vol. 37, p. 39. DOI: 10.1007/b138606. 2005.

Deussen, O., P. Hanrahan, B. Lintermann, R. Měch, M. Pharr e P. Prusinkiewicz. **Realistic Modeling and Rendering of Plant Ecosystems**. Em: *c* 4.1, pp. 275–286. DOI: 10.1145/280814.280898. 1998.

Engel, T. A., A. Frasson e C. T. Pozzer. **Optimizing tree distribution in virtual scenarios from vector data**. Em: *Proceedings of XV SBGames*. São Paulo - SP - Brazil: SBC, pp. 198–201. URL: http://www.sbgames.org/sbgames2016/downloads/anais/157535.pdf. 2016.

Fan, Z., H. Li, K. Hillesland e B. Sheng. **Simulation and rendering for millions of grass blades**. Em: *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games - i3D '15*. New York, New York, USA: ACM Press, pp. 55–60. DOI: 10.1145/2699276.2699283. 2015.

Floyd, R. W. e L. Steinberg. **An Adaptive Algorithm for Spatial Greyscale**. Em: *Proceedings of the Society for Information Display* 17.2, pp. 75–77. 1976.

Flynn, M. J. **Some computer organizations and their effectiveness**. Em: *Computers, IEEE Transactions on* 100.9, pp. 948–960. 1972.

Frasson, A., T. A. Engel e C. T. Pozzer. **Improving Terrain Visualization Through Procedural Generation and Hardware Tessellation**. Em: *Proceedings of XV SBGames*. São Paulo - SP - Brazil: SBC, pp. 218–221. URL: http://www.sbgames.org/sbgames2016/downloads/anais/157677.pdf. 2016.

Gain, J., H. Long, G. Cordonnier e M.-P. Cani. **EcoBrush: Interactive Control of Visually Consistent Large-Scale Ecosystems**. Em: *Computer Graphics Forum* 36.2, pp. 63–73. DOI: 10.1111/cgf.13107. 2017.

Galin, E., A. Peytavie, N. Maréchal e E. Guérin. **Procedural generation of roads**. Em: *Computer Graphics Forum* 29.2, pp. 429–438. DOI: 10.1111/j.1467-8659.2009.01612.x. 2010.

Génevaux, J.-D., É. Galin, E. Guérin, A. Peytavie e B. Beneš. **Terrain generation using procedural models based on hydrology**. Em: *ACM Transactions on Graphics* 32.4, p. 1. DOI: 10.1145/2461912.2461996. 2013.

Giegl, M. e M. Wimmer. **Unpopping: Solving the image-space blend problem for smooth discrete LOD transitions**. Em: *Computer Graphics Forum* 26.1, pp. 46–49. DOI: 10.1111/j.1467-8659.2007.00943.x. 2007.

Gruen, A. e H. Li. **Road extraction from aerial and satellite images by dynamic programming**. Em: *ISPRS Journal of Photogrammetry and Remote Sensing* 50.4, pp. 11–20. DOI: 10.1016/0924-2716(95)98233-P. 1995.

Haar, U. e S. Aaltonen. **GPU-Driven Rendering Pipelines**. Em: *Siggraph 2015*. URL: http://advances.realtimerendering.com/s2015/aaltonenhaar%7B%5C_%7Dsiggraph2015%7B%5C_%7Dcombined%7B%5C_%7Dfinal%7B%5C_%7Dfooter%7B%5C_%7D220dpi.pdf. 2015.

Hammes, J. **Modeling of Ecosystems as a Data Source for Real-Time Terrain Rendering**. Em: *Digital Earth Moving: First International Symposium, DEM 2001 Manno, Switzerland, September 5–7, 2001 Proceedings*. Ed. por C. Y. Westort. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 98–111. DOI: 10.1007/3-540-44818-7_14. 2001.

Hargrove, W. W., F. M. Hoffman e P. M. Schwartz. **A fractal landscape realizer for generating synthetic maps**. Em: *Ecology and Society* 6.1. DOI: 2. 2002.

Jahrmann, K. e M. Wimmer. **Responsive real-time grass rendering for general 3D scenes**. Em: *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '17*, pp. 1–10. DOI: 10.1145/3023368.3023380. 2017.

Kharlamov, A. e I. Cantlay. **Next-Generation SpeedTree Rendering**. Em: *GPU Gems 3*. Cap. 4. URL: http://http.developer.nvidia.com/GPUGems3/gpugems3%7B%5C_%7Dch04.html. 2007.

Lane, B. e P. Prusinkiewicz. **Generating Spatial Distributions for Multilevel Models of Plant Communities**. Em: *Interface* 2002, pp. 69–80. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.9523. 2002.

Mahdavi-Amiri, A., T. Alderson e F. Samavati. **A Survey of Digital Earth**. Em: *Computers and Graphics (Pergamon)* 53, pp. 95–117. DOI: 10.1016/j.cag.2015.08.005. URL: http://dx.doi.org/10.1016/j.cag.2015.08.005. 2015.

Muijden, J. van. **GPU-Based Procedural Placement in Horizon Zero Dawn - Guerrilla**. URL: https://www.guerrilla-games.com/read/gpu-based-procedural-placement-in-horizon-zero-dawn (acesso em 23/07/2017). 2017.

Netzel, A. e P. Rohleder. **Procedural Content Generation on GPU**. Em: *GPU Pro 2*. Ed. por W. Engel. Cap. 1, pp. 29–36. 2016.

Onrust, B. **Automatic generation of plant distributions for existing and future natural environments using spatial data**. master thesis. TU Delft. URL: http://www.gdmc.nl/publications/2015/Automatic%7B%5C_%7Dgeneration%7B%5C_%7Dplant%7B%5C_%7Ddistributions.pdf. 2015.

Onrust, B., R. Bidarra, R. Rooseboom e J. van de Koppel. **Procedural generation and interactive web visualization of natural environments**. Em: *Proceedings of the 20th International Conference on 3D Web Technology - Web3D '15*, pp. 133–141. DOI: 10.1145/2775292.2775306. URL: http://dl.acm.org/citation.cfm?doid=2775292.2775306. 2015.

Oosten, J. van. **CUDA Memory Model**. URL: https://www.3dgep.com/cuda-memory-model/ (acesso em 25/01/2018). 2011.

OpenGL Wiki contributors. **Compute Shader**. URL: https://www.khronos.org/opengl/wiki/Compute%7B%5C_%7DShader (acesso em 12/01/2018). 2018.

OpenStreetMap. **OpenStreetMap Contributors**. URL: https://www.openstreetmap.org (acesso em 25/11/2017). 2017.

Outerra. **Procedural grass rendering**. URL: http://outerra.blogspot.com.br/2012/05/procedural-grass-rendering.html. 2012.

Pajarola, R. e E. Gobbetti. **Survey of semi-regular multiresolution models for interactive terrain rendering**. Em: *Visual Computer* 23.8, pp. 583–605. DOI: 10.1007/s00371-007-0163-2. URL: http://link.springer.com/10.1007/s00371-007-0163-2. 2007.

Pelzer, K. **Rendering countless blades of waving grass**. Em: *GPU Gems*. Vol. 2. 2. Addison-Wesley, pp. 107–121. URL: https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems%7B%5C_%7Dch07.html. 2004.

Pozzer, C. T., C. A. de Lara Pahins e I. Heldal. **A Hash Table Construction Algorithm for Spatial Hashing Based on Linear Memory**. Em: *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*. ACE '14. New York, NY, USA: ACM, 35:1–35:4. DOI: 10.1145/2663806.2663862. 2014.

Prusinkiewicz, P. e A. Lindenmayer. **The algorithmic beauty of plants**. Em: p. 228. 1990.

Skalsky, M. **AsyncTextureReader**. URL: https://github.com/SlightlyMad/AsyncTextureReader. 2017.

Smelik, R. M., T. Tutenel, R. Bidarra e B. Benes. **A Survey on Procedural Modelling for Virtual Worlds**. Em: *Computer Graphics Forum* 33.JANUARY, n/a–n/a. DOI: 10.1111/cgf.12276. 2014.

Smelik, R., T. Tutenel, K. J. de Kraker e R. Bidarra. **Integrating procedural generation and manual editing of virtual worlds**. Em: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - PCGames '10*. New York, New York, USA: ACM Press, pp. 1–8. DOI: 10.1145/1814256.1814258. 2010.

Soille, P. e J. Grazzini. **Extraction of river networks from satellite images by combining mathematical morphology and hydrology**. Em: *CAIP'07 Proceedings of the 12th international conference on Computer analysis of images and patterns*, pp. 636–644. DOI: 10.1007/978-3-540-74272-2_79. 2007.

Strugar, F. **Continuous Distance-Dependent Level of Detail for Rendering Heightmaps**. Em: *Journal of Graphics, GPU, and Game Tools* 14.4, pp. 57–74. DOI: 10.1080/2151237X.2009.10129287. 2009.

SWISSTOPO. **Swiss Federal Office of Topography**. URL: https://shop.swisstopo.admin.ch/en/products/maps/national/vector/smv500 (acesso em 29/01/2018). 2016.

Tarboton, D. G., R. L. Bras, I. Rodriguez-Iturbet e R. M. Parsons. **ON THE EXTRACTION OF CHANNEL NETWORKS FROM DIGITAL ELEVATION DATA**. Em: 5, pp. 81–100. URL: http://hydrology.usu.edu/dtarb/hp91.pdf. 1991.

The Khronos Group. **The Open Graphics Library**. URL: https://www.opengl.org/. 2017.

Thöny, M., M. Billeter e R. Pajarola. **Large-Scale Pixel-Precise Deferred Vector Maps**. Em: *Computer Graphics Forum* 00.0, pp. 1–12. DOI: 10.1111/cgf.13294. 2017.

Tuncer, O. **Fully automatic road network extraction from satellite images**. Em: *Proceedings of the 3rd International Conference on Recent Advances in Space Technologies, RAST 2007*. IEEE, pp. 708–714. DOI: 10.1109/RAST.2007.4284085. 2007.

Tupin, F., H. Maître, J.-F. Mangin, J.-M. Nicolas e E. Pechersky. **Detection of Linear Features in SAR Images : Application to Road Network Extraction**. Em: *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING* 36.2, pp. 434–453. 1998.

United States Geological Survey. **USGS - Earth Explorer**. URL: https://earthexplorer.usgs.gov/ (acesso em 25/11/2017). 2017.

Unity. **Unity Engine**. URL: https://unity3d.com/%20http://unity3d.com/unity (acesso em 09/02/2018). 2018.

Weier, M., A. Hinkenjann, G. Demme e P. Slusallek. **Generating and Rendering Large Scale Tiled Plant Populations**. Em: *Journal of Virtual Reality and Broadcasting* 10.1. 2013.

Wihlidal, G. **Optimizing the Graphics Pipeline with Compute**. Em: *Gdc 2016*, p. 99. URL: http://frostbite-wp-prd.s3.amazonaws.com/wp-content/uploads/2016/03/29204330/GDC%7B%5C_%7D2016%7B%5C_%7DCompute.pdf. 2016.

Wright, R. S., N. Haemel, G. Sellers e B. Lipchak. **OpenGL SuperBible: Comprehensive Tutorial and Reference**. 5th. Addison-Wesley Professional. 2010.