NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

PEDRO MIGUEL OLIVEIRA VALENTE

Licenciado em Ciências e Engenharia Informática

GERAÇÃO PROCEDIMENTAL CONTROLADA DE VEGETAÇÃO EM MUNDOS VIRTUAIS

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
novembro, 2021

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

GERAÇÃO PROCEDIMENTAL CONTROLADA DE VEGETAÇÃO EM MUNDOS VIRTUAIS

PEDRO MIGUEL OLIVEIRA VALENTE

Licenciado em Ciências e Engenharia Informática

Orientador: Fernando Pedro Reino da Silva Birra
Professor Auxiliar, Universidade NOVA de Lisboa

Coorientador: João Carlos Gomes Moura Pires
Professor Auxiliar, Universidade NOVA de Lisboa

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
novembro, 2021

Geração Procedimental Controlada de Vegetação em Mundos Virtuais

Resumo

A criação de um mundo virtual verdadeiramente imersivo é um tópico recorrente na área da computação gráfica e suas aplicações. Nos últimos anos houve um avanço significativo nesta área com a introdução de sistemas de realidade virtual no mercado doméstico.

Estes avanços tiveram ramificações que se estenderam para além do mundo dos videojogos, por exemplo no combate a incêndios, havendo benefícios em integrar esta tecnologia no treino de operacionais de campo e no fornecimento de dados de ajuda ao combate. A incorporação desta tecnologia evita que os utilizadores tenham de estar fisicamente presentes no local para realizar treinos e facilita a visualização de perspetivas diferentes da zona de combate.

Contudo, criar um ambiente virtual que espelhe a realidade é um processo complexo e potencialmente moroso que envolve a construção de várias camadas de representação. Uma das mais importantes corresponde à camada de vegetação que, no contexto do combate aos incêndios, é determinante para atender e antecipar a propagação dos incêndios.

A fim de resolver este problema é apresentada uma solução que integra dados reais, de produtos gerados a partir de técnicas de deteção remota, com técnicas de geração procedimental. Esta abordagem não tem o fim de atingir uma qualidade foto-realista, mas sim uma distribuição que espelhe a realidade, cujo processo de geração seja o mais automatizado possível e que seja rápido o suficiente para ter benefícios comparativamente à geração manual.

Desta forma, irão ser usados dados de cartas de ocupação do solo, mais precisamente, da COS e do CLC, em conjunto com dados de camadas de alta resolução, técnicas de geração procedimental e o motor de jogos *Unreal Engine 4*. Os resultados serão analisados segundo uma avaliação de performance e análise qualitativa dos dados gerados.

Palavras-chave: Geração procedimental, Modelação de Vegetação, Dados de Deteção Remota, Realidade Virtual, Sistemas de Informação Geográfica, Unreal Engine

Abstract

The creation of an immersive virtual world is a recurring topic in the area of computer graphics. In recent years there have been significant advancements in this area with the introduction of virtual reality systems in the domestic market.

These advances had ramifications that extended beyond the world of video games, for example, there are benefits in integrating this technology in the context of firefighting, in the training of field operatives, and providing data to help combat fires. This incorporation prevents users from having to be present in affected areas to carry out training, and facilitates the visualization of different perspectives of the combat zone.

However, creating a virtual environment that mirrors reality is a complex and potentially time-consuming process that involves building multiple layers of representation. One of the most important is the vegetation layer, which, in the context of firefighting, is crucial to anticipate the spread of fires.

In order to tackle this problem, the presented solution integrates real data form products generated through remote sensing techniques, with procedural generation techniques. This approach doesn't intend to achieve a photo-realistic quality, but rather a distribution that mirrors reality, whose generation process is as automated as possible, and fast enough to insure benefits compared to manual generation.

The solution combines data from land usage maps, from COS and CLC, with data from high resolution layers, and procedural generation techniques on the Unreal Engine 4 game engine. The results will be analyzed according to a performance evaluation, and a qualitative analysis of the generated data.

Keywords: Procedural generation, Vegetation Modeling, Remote Sensing Data, Virtual Reality, Geographic Information Systems, Unreal Engine

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Siglas

CLC	CORINE Land Cover
CNN	Convolutional Neural Network
COS	Carta de Uso e Ocupação do Solo
DEM	Digital Elevation Model
DSM	Digital Surface Model
DTM	Digital Terrain Model
GDC	Game Developers Conference
GPU	Graphics Processing Unit
HRL	High Resolution Layer
SI-MORENA	Sistemas Inteligentes para Monitorização de Recursos Naturais
UE4	Unreal Engine 4
VR	Virtual Reality

1

Introdução

1.1 Motivação

A simulação de mundos virtuais, fidedignos à realidade, enquadra-se num tema de investigação que tem sido estudado ainda antes da criação dos primeiros rasterizadores nos anos setenta [43, 27]. Na atualidade, tecnologias emergentes como a realidade virtual e a realidade aumentada, correspondem a novos passos na direção da criação de um ambiente verdadeiramente imersivo. Dada a juventude destes sistemas, há muito pouca informação no que toca às possibilidades que esta tecnologia pode apresentar [27].

Assim, é de notar que nem todas as aplicações desenvolvidas com realidade virtual em mente correspondem a videojogos, isto advém do facto de que a realidade virtual é uma plataforma ideal para resolver problemas, onde é necessário existir uma noção de presença que não possa ser fornecida através dos meios de visualização tradicional, como por exemplo, ecrãs com duas dimensões. Uma situação muito estudada passa pela criação de representações virtuais de partes do mundo real, as quais podem ser usadas como formas de lazer [34], em situações de treino [9], ou como forma de realizar estudos à distância [33].

Representações do mundo real podem ser usadas no combate a incêndios [1], não só em termos de treino dos operacionais de campo, mas também no fornecimento de dados adicionais que podem servir de ajuda ao combate.

Nestes contextos, os utilizadores podem requerer informação adicional que não pode ser fornecida apenas através de meios tradicionais. Por exemplo, no contexto do treino de combatentes é preciso que estes se encontrem presentes no local para que consigam ter um melhor entendimento da vegetação e do meio que os rodeia, contudo, a realização destas operações de campo são dispendiosas, e muitas vezes desnecessariamente complexas. Caso fosse exequível criar um ambiente virtual com um forte sentido de imersão, então era possível colocar os utilizadores numa situação em que estes conseguiam receber um nível de treino equivalente ao fornecido pelos meios tradicionais, mas sem as inconveniências e custos associados.

Outro exemplo, onde o uso de VR pode resultar numa melhoria comparativamente

às técnicas atuais de combate a incêndios corresponde ao ponto de vista de um operador, durante um incêndio, que não se encontra presente no terreno. Este utilizador necessita ter a maior quantidade possível de informação, de modo a que consiga tomar decisões acertadas. Sendo assim, a noção de presença fornecida por um capacete de realidade virtual pode facilitar o acesso e a visualização imediata de dados que podem ser cruciais no combate.

No entanto, criar um mundo verdadeiramente imersivo que siga dados concretos de forma precisa e representativa do mundo real não é uma tarefa fácil, pois implica um estudo das várias camadas de informação que formam o nosso mundo, desde o terreno, às estruturas de fabrico humano, como por exemplo estradas e edifícios, passando pela própria vegetação que nos rodeia.

A camada de vegetação corresponde a uma das componentes mais cruciais no contexto do combate aos incêndios, não só em termos de representar a propagação dos mesmos, mas também na definição de rotas seguras em zonas de acesso difícil. No contexto desta tese iremos focar-nos na criação desta mesma camada, não com o foco na criação de florestação foto-realista, mas sim representativa da distribuição de vegetação real.

1.2 Contexto

Este projeto integra-se no escopo da iniciativa de investigação SI-MORENA (Sistemas Inteligentes para Monitorização de Recursos Naturais). O objetivo principal consiste em criar um sistema de realidade virtual que consiga fornecer assistência e treino a profissionais de modo a ajudá-los em situações de desastres naturais, com especial foco no combate a incêndios florestais.

Como já foi mencionado anteriormente, a criação de um sistema destes, cobre um conjunto de camadas de informação do mundo real. Assim, um dos pontos fulcrais deste projeto passa pela automatização do processo de transformação de dados reais num mundo virtual, não necessariamente foto-realista, mas sim um cujas representações virtuais sigam os padrões do mundo real.

Realçamos o facto de este trabalho dar continuidade a outros já realizados anteriormente neste projeto. Muito do trabalho focou-se no desenvolvimento das técnicas e dos modelos de geração procedimental, estendendo ferramentas já existentes, enquanto o processo de automatização da geração foi deixado de parte. No contexto da geração procedimental de vegetação, verificou-se que é possível usar ferramentas de geração automatizada fornecendo como *input* sumários da vegetação numa determinada região.

Deste modo, este trabalho surge como um seguimento do trabalho previamente efetuado, e pretendemos agora realizar a construção de um sistema de geração procedimental de vegetação, com foco na automatização, usando como *input* um conjunto de dados georreferenciados, nomeadamente imagens de satélite, cartas de ocupação do solo e fotografias aéreas.

1.3 Definição do problema

Assim, dado o contexto anterior, o problema que pretendemos abordar com esta tese incide no desenvolvimento de um sistema automático, que apoiado num conjunto de dados que descrevem a realidade, realiza uma tradução precisa para a vegetação de um mundo virtual, procurando não uma solução que gere cenários foto-realistas, mas sim representações de alta precisão, onde esta é medida com base na semelhança da distribuição da vegetação no mundo virtual comparativamente à descrita pelos dados.

Como já foi mencionado anteriormente, no contexto deste projeto, ter uma representação fiável da vegetação que siga os padrões reais da vegetação é prioritário, não só para haver representações da propagação de incêndios, como também, na assistência à navegação no combate a um incêndio.

Podemos assim decompor o problema em várias partes: uma focada na automatização da geração de parâmetros de *input* para os modelos procedimentais, tentando fazer uso de produtos derivados de deteção remota, e outra que passa pela otimização e modificação dos modelos de geração procedimental de modo a que estes gerem representações adequadas.

Como iremos ver no capítulo 2, tentar gerar representações realistas usando dados do mundo concreto não é um problema novo. No entanto, não existem soluções publicamente disponíveis, que façam uma combinação de ferramentas de deteção remota com modelos de geração de vegetação. Para além disto, muitos dos modelos de vegetação que usam dados reais encontram-se desatualizados, tendo em conta os avanços mais recentes na área de *remote sensing* [46].

Neste projeto vai ser utilizado o motor de jogos *Unreal 4*, em parte porque este possui um suporte extenso para a construção de aplicações de realidade virtual, mas também, pelo facto de este corresponder a um dos motores *Standard* da indústria [19, 36].

1.4 Contribuições previstas

As contribuições previstas com a realização deste trabalho são as seguintes:

- **Aumentar a precisão na geração da camada de vegetação, fazendo a integração de dados obtidos a partir de ferramentas de deteção remota:** O objetivo principal deste trabalho é aumentar a fidelidade das soluçãos anteriores, não com o fim de atingir o foto-realismo, mas sim uma representação fidedigna de como a vegetação se encontra organizada no ambiente real. Para atingir isto é preciso fazer uma integração de dados obtidos a partir de ferramentas de deteção remota, que nos permitem retirar informação relevante do mundo;

- **Integrar vários tipos de vegetação:** De forma a obter uma representação mais realista da camada de vegetação, é preciso garantir que o processo de geração tem em

conta vários níveis de vegetação. Assim, outro foco deste trabalho é garantir que este integra várias camadas de vegetação;

- **Automatizar do processo de geração:** Um dos grandes focos desta dissertação é garantir que o processo de geração requer a menor intervenção possível por parte do utilizador, de modo a que este processo seja beneficiário em oposição à geração manual;

- **Desenvolver um conjunto de técnicas que possam ser usadas em diversas zonas do território:** Outro objetivo deste trabalho passa por desenvolver um conjunto de técnicas que possam ser usadas para a geração de vegetação diversa, de modo a poderem ser aplicadas em zonas distintas do território;

1.5 Estrutura do documento

Este documento encontra-se dividido em quatro capítulos distintos, cada um contendo a seguinte informação:

- **Introdução:** No capítulo 1 é explorado o problema que se irá abordar, bem como a motivação e o contexto por detrás da realização desta dissertação;

- **Estado da Arte:** O capítulo 2 explora o estado atual da indústria, as várias soluções existentes atualmente no mercado e as tecnologias que vão ser abordadas no decorrer desta tese, desde as técnicas de geração procedimental mais usadas, ao estado atual na área da deteção remota;

- **Abordagem:** No decorrer do capítulo 3 é apresentada a abordagem que foi planeada durante a fase de preparação da dissertação, focando-se em apresentar de forma detalhada o processo de geração que vai ser usado;

- **Implementação:** No capítulo 4 é exposta a implementação da abordagem proposta, detalhando todos as fases da implementação com exemplos concretos;

- **Avaliação:** Por fim, no capitulo 5 é feita uma breve avaliação do sistema desenvolvido segundo um conjunto de métricas.

2

Estado da arte

Ao longo deste capítulo vão ser abordados os vários temas científicos e tecnológicos que são relevantes para esta dissertação, fazendo uma análise detalhada do estado corrente da indústria, com foco nas soluções atuais do mercado e das técnicas que vão ser usadas ao longo do desenvolvimento deste trabalho.

Assim, neste capítulo é feita uma pesquisa das técnicas de geração procedimental mais comuns na indústria 2.1, de seguida, é feita uma análise dos métodos mais usados na geração procedimental de vegetação 2.2. Na secção 2.3 faz-se um balanço dos dados que vão ser usados no desenvolvimento, em conjunto com um estudo de como estes podem ser tratados e usados. Por fim, na secção 2.4 é efetuada uma breve análise do motor de desenvolvimento que vai ser usado no contexto deste trabalho.

2.1 Técnicas de geração procedimental

Como já foi mencionado anteriormente, o grande foco desta tese passa pelo desenvolvimento de um sistema de geração de vegetação que consiga gerar uma representação virtual e que siga padrões de vegetação reais. A geração procedimental, por oposição à geração manual, é um ponto crucial deste projeto, visto que tem um grande foco em automatizar o processo de geração da vegetação de um mundo grande, com a menor intervenção humana possível.

Assim, podemos definir geração procedimental como sendo todos os métodos que geram conteúdo recorrendo a um algoritmo. Como já foi mencionado anteriormente, as principais vantagens desta técnica são a capacidade de automatizar o processo de geração de conteúdo, podendo assim gerar mais conteúdo num menor espaço de tempo.

O processo de desenhar um método de geração procedimental pode ser feito através de um conjunto de passos, como foi descrito ao longo da seguinte apresentação na *GDC* (*Game Develeopr's Conference*) de 2017 [12], onde se começa por fazer uma análise do espaço de soluções que se podem obter com o algoritmo de geração procedimental, ou seja, tenta-se compreender o que se pretende gerar com este algoritmo. De seguida, faz-se um estudo das restrições que vão ser usadas no processo e dos métodos que se adaptam ao

processo de geração de conteúdo. Para tal, é preciso fazer um estudo dos componentes que constituem o objeto a ser criado, o qual pode ser feito através da consulta de especialistas, ou lendo os artigos publicados pelos mesmos. Por fim, itera-se pelos pontos anteriormente mencionados até que se obtenham os resultados pretendidos.

Dois conceitos fundamentais no que toca à geração procedimental de conteúdo têm a ver com o espaço de possibilidades e o espaço de expressividade. O primeiro pode ser definido como a totalidade de objetos distintos que são gerados no contexto do algoritmo escolhido, enquanto o último, corresponde ao espaço de resultados que define o quão distintos os resultados obtidos são.

Com isto, podem-se definir duas formas distintas de classificar os processos de geração procedimental, a primeira tem a ver com a dimensão do espaço de possibilidades gerado, onde os métodos são aditivos, ou seja, estas técnicas são usadas como uma forma de construir conteúdo e consequentemente, aumentam o espaço de possibilidades. Por oposição, os métodos subtrativos, reduzem o espaço de possibilidades, selecionando apenas os melhores resultados.

Por contraste, os métodos de geração procedimental também podem ser classificados de acordo com a ordem de prioridade pela qual a informação de *input* é processada [18]. Se há uma maior ênfase na geração com base em outras estruturas já definidas, agrupando-as de modo a gerar novo conteúdo, então estamos perante um processo de geração que assume uma abordagem *bottom-up* por contraste. Se tivermos um conjunto de valores abstratos que servem de *input* para a geração de um objeto, estamos perante um método *top-down*.

Dado o contexto deste projeto, é importante realçar algumas técnicas de geração procedimental específicas que podem ser relevantes para a geração de vegetação. Por conseguinte, temos os seguintes métodos [12]:

- **Distribuições:** Esta técnica pode ser classificada como sendo um método aditivo que segue uma estrutura *bottom-up*, e tem como principal objetivo gerar uma distribuição matemática de um conjunto de objetos. Pode ser gerada através de processos com diferentes graus de complexidade, que pode ser simplesmente uma grelha de posições com um *offset*, ou algo mais complexo como uma sequência de Halton [44]. Na apresentação de 2017 [12] são mencionados um conjunto de conceitos fundamentais no que toca a geração de distribuições:
  - ***Barnacling*:** Este termo serve para descrever situações onde há um objeto grande que é colocado em conjunto com um grupo de objetos mais pequenos que são instanciados à volta do primeiro;
  - ***Footing*:** Este termo é usado para descrever ocorrências onde dois objetos se intercetam, devendo haver alguma informação visual que informe o utilizador;
  - ***Greebling*:** Este termo tem origem no mundo do *star wars* e serve para descrever a técnica de montar um objeto como sendo uma composição hierárquica de

peças individuais, com o objetivo de aumentar o espaço de expressividade;

Distribuições são muito usadas no contexto de sistemas de geração de vegetação;

- **Paramétricos:** Esta técnica corresponde a um método aditivo estruturado de uma forma *top-down*. Descreve todos os processos de geração que recebem um conjunto de parâmetros de *input*, e que traduzem um conjunto de parâmetros abstratos. Estes dados são passados por um algoritmo que dá como *output* o objeto pretendido;

- **Interpretativos:** Esta técnica é um método aditivo que segue uma estrutura *top-down*, cuja classificação aplica-se a todos os métodos que começam com uma representação simples dos dados e que posteriormente passa por um algoritmo que interpreta estes dados por outros valores. Um exemplo relevante consiste na interpretação de dados reais para gerar representações virtuais, como por exemplo usar mapas reais para gerar vegetação;

- **Simulações:** Esta técnica pode ser classificada como sendo um método aditivo com uma estrutura *bottom-up*, e inclui todas as soluções que recorrem a uma simulação de um ambiente com um conjunto de parâmetros que produz um *output* desejado, por exemplo simulação do crescimento de vegetação;

- **Gerar e testar:** Esta técnica corresponde a um método subtrativo, estruturado de uma forma *top-down* que engloba todas as técnicas que recorrem a uma função de *fitness*. Avalia-se o interesse, ou a qualidade, do objeto gerado, com o objetivo de remover os piores resultados. Na geração de vegetação, por exemplo, pode-se usar uma função que avalie o nível de fidelidade da distribuição gerada, dado um conjunto de distribuições reais;

É importante salientar que muitos destes métodos podem e devem ser combinados de modo a gerar outputs mais variados e interessantes. Um padrão comum passa por realizar a geração em dois passos, primeiro gera-se um conjunto muito grande de soluções recorrendo a um conjunto de métodos aditivos e, de seguida, faz-se uma filtragem do espaço de possibilidades usando técnicas subtrativas.

Assim, é possível verificar que muitas das técnicas anteriores são relevantes no que toca à geração de vegetação e, como iremos ver, muitos dos métodos mais usados neste campo, acabam por corresponder a uma combinação de técnicas de simulação para gerar distribuições realistas da vegetação.

2.2 Geração procedimental de vegetação

No seguimento do que foi referido anteriormente, é importante efetuar uma análise mais aprofundada dos processos de geração procedimental de vegetação mais comuns. É de

notar que a geração de vegetação é uma tarefa complexa e as técnicas que temos disponíveis variam de acordo com *output* que pretendemos obter. Assim, é importante perceber o resultado pretendido, antes de se poder compreender que técnicas é que se podem aplicar.

Um dos aspetos do *output* que distingue as técnicas disponíveis, tem a ver com a dimensão do problema com o qual se está a lidar, isto é, se a geração está a ser feita para mundos grandes [5, 28] ou pequenos [3]. Por norma, a geração de vegetação para mundos de maiores dimensões é um processo mais dispendioso e, consequentemente, muitas das técnicas que podem ser aplicadas a mundos pequenos não escalam para mundos de maior dimensão.

Outro ponto importante passa por fazer uma análise do nível de detalhe que se pretende representar com o *output*. Isto é, que fatores vegetais é que vão ser representados, tendo em conta o desenvolvimento individual de cada planta, ou apenas a distribuição abstrata de como é que as plantas se encontram posicionadas no ambiente.

No contexto deste trabalho, os mundos que estão a ser gerados cobrem grandes áreas florestais e o foco desta componente do projeto é obter distribuições realistas da vegetação, cobrindo diversos tipos de plantas, não se focando na geração individual de cada modelo com absoluta precisão.

Relativamente ao problema de gerar vegetação para uma grande área, podemos classificar os métodos de geração de quatro formas distintas, de acordo com as técnicas de geração procedimental que estes priorizam, mesmo que façam uso de outras técnicas.

Em primeiro lugar, existem modelos de simulação [2.2.1] que englobam todos os modelos e dão prioridade a técnicas de simulação de vegetação usando parâmetros bióticos e abióticos, que corresponde a uma abordagem mais *bottom-up*. De seguida, os modelos probabilísticos [2.2.2] cobrem todos os modelos que geram vegetação guiando-se por distribuições probabilísticas, e aqui estamos perante uma solução mais *top-down*. Os modelos híbridos [2.2.3], correspondem a todos os modelos que usam uma combinação de técnicas probabilísticas com simulações bióticas e/ou abióticas, para gerarem distribuições de vegetação. Por fim, os modelos guiados por dados reais [2.2.4], combinam as técnicas mencionadas anteriormente em conjunto com dados reais, de modo a obterem distribuições realistas da vegetação.

2.2.1 Modelos de simulação

Como já foi mencionado anteriormente, as soluções que dão ênfase à simulação de sistemas ecológicos para gerar a vegetação de um mundo virtual são classificadas como sendo modelos de simulação. Estes modelos geram conteúdo com base numa simulação de um conjunto de parâmetros bióticos e abióticos. A simulação recebe como *input* uma coleção de características que, não só descrevem o estado inicial da simulação, como também podem ditar a forma como a simulação evolui. Estes valores podem corresponder a informação não biológica que caracteriza o ambiente onde a simulação vai decorrer, como por exemplo a temperatura média, a distribuição dos corpos de água no mundo, a inclinação

e a constituição do solo, entre outros. Dito isto, estes valores também podem descrever informação biológica, tal como a distribuição inicial de sementes e plantas no mundo, a forma como as plantas interagem entre si, e outros conjuntos de interações biológicas que podemos verificar num ecossistema. Para além do já mencionado, estes modelos também recebem a janela de tempo onde a simulação vai decorrer.

Após o estabelecimento deste conjunto de características iniciais, a simulação trata cada planta como se fosse uma partícula que interage com outras plantas segundo um conjunto de relações bióticas. A relação mais frequentemente apresentada é a competição por recursos do solo [10]. Com isto, cada planta tem a capacidade de crescer competindo por recursos com as plantas vizinhas, começando a propagar-se no momento em que atingem a maturidade, e eventualmente morrendo devido à falta de recursos, ou de idade, ou por motivos não biológicos.

Há várias técnicas de simulação que são usadas neste tipo de modelos, onde o procedimento mais comum passa pela simulação de iterações radiais [10, 8, 2, 15]. Neste método, para cada planta, é considerado um raio e, sempre que os raios de duas plantas intersetam, ocorre um processo de decisão que determina qual a planta que é colocada.

Tomando como exemplo, a apresentação de Etienne Carrier durante a GDC de 2018 [8], o apresentador refere que o algoritmo desenvolvido usa dois parâmetros por espécie de modo a controlar a colocação da vegetação, atribuindo a cada uma das espécies valores únicos. O primeiro corresponde à viabilidade de uma espécie (Figura 2.1), dado este valor, quando os raios de viabilidade de duas plantas de espécies diferentes se intersetam, é selecionada a planta com um maior valor de viabilidade. O segundo parâmetro chama-se prioridade e detém um comportamento idêntico ao primeiro, ou seja, também se assume um raio e um valor de prioridade por espécie, e sempre que os raios de duas plantas de espécies diferentes se intersetam, é selecionada a planta como maior valor de prioridade, este segundo parâmetro é usado de forma a que plantas com um raio e valor de viabilidade muito alto, possam ter vegetação rasteira na sua vizinhança.

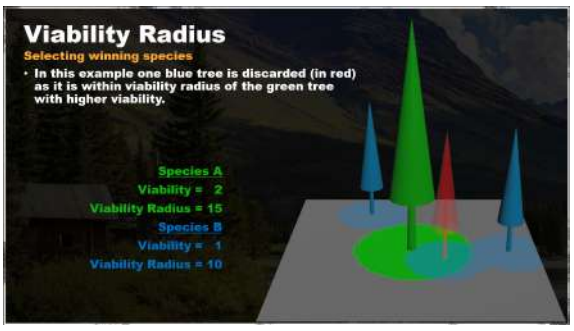


Figura 2.1: Demonstração do funcionamento do sistema de viabilidade de [8].

Por contraste, uma técnica de simulação menos comum passa pela simulação da interação de agentes de natureza não vegetal com a vegetação [4, 11, 14]. Estes agentes podem ser de natureza biológica, predadores ou animais que possam ajudar na proliferação de sementes, como podem ser não biológicos, como por exemplo fogos ou deslizamentos de

solo.

Apesar de ser menos comum, este tipo de simulação continua a apresentar resultados interessantes e biologicamente corretos. Cordonnier et al. [14] complementam o sistema tradicional de simulação do crescimento de vegetação com um conjunto de eventos geomorfológicos que podem ocorrer ao longo da simulação. Estes eventos variam desde a queda de chuva, à mudança da temperatura e de incêndios (Figura 2.2).

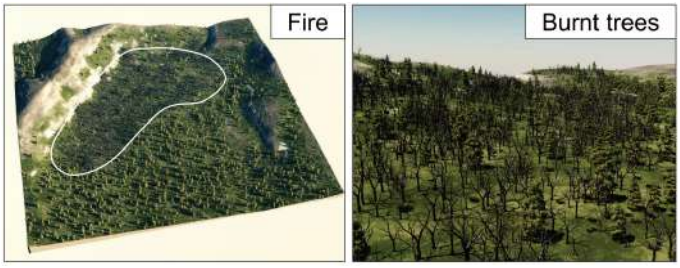


Figura 2.2: Representação das consequências de um fogo causado por utilizador, retirado de [14].

Portanto, a qualidade da simulação depende muitas vezes dos fatores e interações considerados, não só em termos da quantidade de características avaliadas, como também na qualidade das mesmas. Para além disto, a precisão do resultado também vai depender das técnicas usadas para calcular os valores da simulação.

Ch'Ng [11] refere que, para conseguirmos obter um resultado o mais realista possível, devemos considerar o maior número possível de parâmetros de entrada, não só representando características ambientais, mas também ao nível de cada planta individual. No entanto, o processo de simular recorrendo a uma grande quantidade de fatores é dispendioso, tendo assim limitações de performance.

No desenvolvimento da solução apresentada [11], o autor valoriza a implementação dos fatores que afetam o crescimento e a distribuição da vegetação de uma forma significativa, agregando estes fatores usando a média ponderada deles. De seguida, conclui que, de acordo com vários estudos [10], os fatores que mais afetam o crescimento de plantas são características abióticas, mais precisamente a luz solar, a humidade e acesso a água, a qualidade do solo e, em conjunto com estes fatores, a competição entre plantas por território e outros recursos.

Por fim, podemos concluir que os modelos de simulação, apesar de fornecerem um alto nível de realismo, acabam por ter limitações, não só em termos de *performance* computacional, como também ao nível da complexidade estrutural que podem apresentar.

2.2.2 Modelos probabilísticos

Por oposição aos sistemas de simulação, existem modelos probabilísticos, que como já foi mencionado anteriormente, assumem uma abordagem *top-down*, no que diz respeito à geração de conteúdo. Isto quer dizer que em vez de haver um conjunto de atores que seguem uma coleção de regras de propagação, simulando fatores bióticos e abióticos,

recebem como *input* uma representação da distribuição da vegetação e tentam gerar uma representação virtual do mundo de forma a que siga as distribuições fornecidas.

Estes modelos são frequentemente usados no contexto da construção de ferramentas para artistas visuais [20, 8, 16], dado que estes métodos não só são mais leves computacionalmente do que os sistemas de simulação, permitindo assim aos artistas fazerem alterações ao mundo em tempo real obtendo *feedback* imediato, mas também porque fornecem um maior controlo ao utilizador, podendo ser este a definir a forma como a vegetação se encontra distribuída pelo mundo, como por exemplo, através de pincéis de vegetação [20, 16].

No que toca à forma como as distribuições de *input* são geradas e representadas, a grande maioria dos modelos utiliza imagens *raster* de modo a representar mapas de densidade de vegetação (Figura 2.3), onde cada píxel da imagem pode ser mapeado para uma posição do mundo virtual com um valor de densidade associado, se normalizarmos os valores da imagem entre zero (0) e um (1). O mais comum é considerar os píxeis a um, como sendo áreas de alta densidade enquanto os píxeis a zero (0) são zonas sem vegetação.

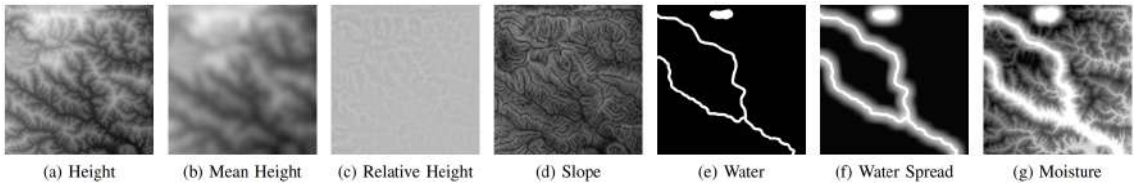


Figura 2.3: Exemplos de mapas de densidade, todos estes mapas são compilados de modo a gerar o mapa (g). Os mapas (a), (d), (e) e (g) influenciam diretamente a distribuição da vegetação, imagem retirada de [29].

Nascimento et al [29] geram mapas de distribuição de vegetação combinando várias imagens que contêm informação sobre um conjunto de parâmetros abióticos. Estas imagens são compiladas diretamente a partir dos dados do mundo já gerado, usando maioritariamente dados de elevação do terreno e do posicionamento de corpos de água, para gerarem o mapa de inclinação do terreno os autores aplicam uma transformação ao mapa de elevação (Figura 2.3).

Adicionalmente, ainda na apresentação da GDC de 2017 [20], é exposto um sistema com uma estrutura hierárquica, em que um mapa de densidade de um nó é propagado para os filhos, posteriormente, este mapa que vem do nó pai pode ser com mapas locais de cada nó de forma a gerar o mapa de densidade final. Permite aos autores definir máscaras gerais que são depois propagadas para o resto da vegetação. Estas máscaras podem conter o posicionamento de estradas e rios, impedindo que cresça vegetação nestas zonas (Figura 2.4).

Relativamente à forma como o mapa de distribuição é interpretado pelo sistema, o processo mais comum passa por fazer um processo de *half-toning* ou *Dithering* da imagem de densidade de vegetação [5, 20].

Continuando no escopo da apresentação já mencionada[20], os autores começaram por

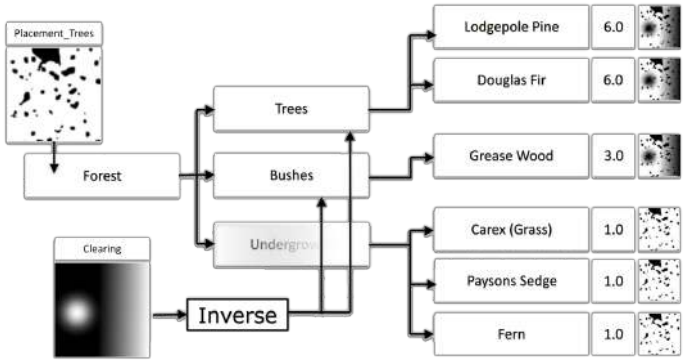


Figura 2.4: Sistema hierárquico de propagação de mapas de densidade apresentado em [20].

usar uma técnica de *dithering* dos mapas de densidade. Este processo consiste em fazer uma binarização das imagens tentando manter ao máximo os tons da imagem inicial. No entanto, as distribuições geradas eram muito uniformes e pouco realistas, tendo os autores optado por usar um padrão que mapeia um conjunto de pontos de acordo com um valor de *input*.

Contudo, os autores depararam-se com outro problema. Como estavam a lidar com várias camadas de vegetação, vão verificar-se imensas colisões entre plantas de camadas diferentes. A solução imediata passa por fazer *read-backs* dos objetos colocados no ambiente e eliminar os que estão a colidir. No entanto, esta solução gera dependências complexas, e o processo de geração, a ocorrer no GPU, torna-se exageradamente complicado. A solução que encontraram funciona apenas com plantas com a mesma pegada, e tem como base uma alteração do processo de *dithering*, onde em vez de considerarem apenas um mapa de densidade como *input*, assumem vários que se encontram organizados em camadas distintas, de seguida, durante o processo de binarização é feito um passo adicional que decide a espécie de planta que é colocada no mundo, de acordo com a camada de vegetação onde se encontra.

No entanto há alternativas, Andújar et al [3] usam uma técnica de *dart-throwing* com uma restrição de distância, onde cada planta é colocada no mundo tentando seguir uma distribuição anteriormente estabelecida.

Com isto, pode-se concluir que este tipo de modelos, apesar de serem mais eficientes computacionalmente, a sua qualidade fica inteiramente dependente da distribuição considerada inicialmente e, para além disto, há o problema adicional de que a vegetação gerada, sem o uso de métodos adicionais, fica restrita a uma só camada.

2.2.3 Modelos híbridos

Até agora foram mencionados dois tipos de modelo distintos: modelos de simulação e modelos probabilísticos. A separação entre estas duas classificações não é tão estrita quanto parece de imediato, podendo haver técnicas que são primariamente probabilísticas e, de

modo a evitarem os problemas associados com os procedimentos selecionados, integram também técnicas de simulação, ou vice-versa.

Por exemplo, Deussen et al. [15] optaram por, simplesmente, criar duas soluções distintas. A primeira faz uso de técnicas de análise de distribuições probabilísticas, à qual eles chamaram de especificação explícita, uma vez que os mapas de densidade são definidos pelos utilizadores. A segunda faz uso de técnicas de simulação, mais precisamente através do uso de *L-systems* [31], à qual eles chamaram de geração procedimental.

Quando um modelo não é facilmente classificado como pertencendo a alguma das classes anteriores, usando uma combinação de técnicas de simulação de crescimento vegetal em conjunto com representações probabilísticas da distribuição da vegetação, pode ser classificado como sendo um modelo híbrido. É comum usar esta combinação de técnicas de modo a tentar corrigir limitações apresentadas pela solução escolhida. Por exemplo, anteriormente foi mencionado que modelos probabilísticos não são indicados para representar várias camadas de vegetação, dado que tradicionalmente este tipo de modelos não detém nenhuma informação adicional sobre a forma como as várias camadas interagem uma com as outras, o que pode resultar em conflitos entre plantas quando são colocadas no ambiente.

Nascimento et al. [29] usam três camadas de vegetação, diferenciadas pela altura das plantas presentes nestas camadas. Estas encontram-se organizadas numa *quad-tree* [32], onde cada camada está associado a uma altura da estrutura de dados e a uma zona no mapa. Na figura 2.5 é possível ver uma representação das várias camadas de vegetação organizadas segundo a *quad-tree*, a camada que contém as plantas de maior dimensão, a *Layer 1* na imagem, está associada aos nós mais altos da árvore, a camada que contém plantas de dimensões médias, a *Layer 2* na imagem, corresponde aos nós intermédios da árvore e, por fim, a camada que representa a vegetação rasteira, a *Layer 3* na imagem, corresponde às folhas da *quad-tree*.

Plantas dentro da mesma camada não se afectam umas às outras, no entanto, como os autores avaliam as camadas de cima para baixo de acordo com a altura dos nós na *quad-tree*, a distribuição das plantas das camadas inferiores é afetada pela distribuição das plantas nas camadas superiores, isto permite aos autores simular a interação entre plantas de diferentes alturas. Eles combinam esta lógica com um parâmetro de distância mínima de modo a garantir que não há colisões entre plantas de camadas diferentes.

Por contraste, outra forma de combinar modelos de simulação com modelos probabilísticos, passa por combiná-los de forma a melhorar a eficiência computacional dos modelos de simulação, reduzindo o tempo de processamento das simulações usando dados de distribuições pré-calculadas. Gain et al. [17] tinham como objetivo principal desenvolver uma ferramenta que permitisse aos utilizadores desenharem ecossistemas de grandes escala que seguissem os princípios biológicos e botânicos do mundo definido. Assim, optaram por conceberem um conjunto de simulações detalhadas em ambientes de pequena escala, cujos resultados lhes permitiu obter dados estatísticos que podem ser usados para representar os ambientes em grande escala. De modo a extraírem dados dos mundos

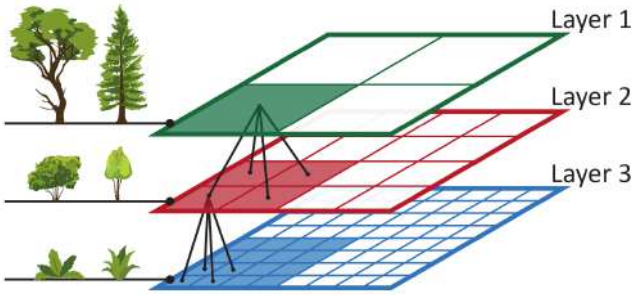


Figura 2.5: Representação das três camadas de vegetação onde as plantas mais altas ocupam a camada de topo, imagem retirada de [29].

resultantes das simulações, os autores usaram uma função que analisa a distribuição de discos de raios diferentes, onde cada um representa a copa de uma planta.

Por fim, conseguimos ver que este tipo de modelos assume um equilíbrio entre classificações mencionadas anteriormente, gerando modelos muito mais complexos que não conseguem atingir os benefícios de usar uma solução puramente probabilística ou simulada. Nunca serão computacionalmente tão eficientes como um modelo mais probabilístico, e ao mesmo tempo os resultados obtidos nunca serão tão realistas como uma simulação complexa.

2.2.4 Modelos guiados por dados reais

Até ao momento, analisamos os principais modelos que podem ser usados para representar a distribuição da vegetação num mundo virtual. No entanto, a grande parte dos modelos mencionados dão ênfase à criação de vegetação para um ambiente virtual, sobre a forma de simulações complexas, ou ferramentas para "pintar"vegetação, sem qualquer ligação a dados reais.

A forma como decorre esta ligação depende em primeiro lugar dos dados a que se tem acesso. Por exemplo, os modelos de simulação utilizam dados como a temperatura média anual e a qualidade do solo, os quais correspondem a dados que podem ser registados através de estudos do ecossistema que se pretende simular, e usados como *input* direto. No entanto, como iremos ver mais à frente, muitos dos dados disponíveis contêm informação limitada do ambiente, não sendo suficiente para criar distribuições verdadeiramente realistas.

Os dados frequentemente usados são ortofotos [30], *DTM*'s (*Digital Terrain Model*) [3], *DEM*'s (*Digital Elavation Model*) e *DSM*'s (*Digital Surface Model*). No entanto, o uso deste tipo de dados apresentam limitações de acordo com a tarefa a realizar. Por exemplo, em zonas de vegetação muito dispersa, tal como zonas urbanas ou de cultivo, a tarefa de deteção de vegetação individual em ortofotos é muito mais simples do que em zonas de floresta densa ou semi-densa, sendo até possível em alguns casos fazer a deteção de plantas individuais a um baixo custo. Por exemplo, Karantzalos et al. [21] conseguem fazer

uma identificação de plantas individuais aplicando um filtro seguido de uma binarização da imagem.

Outro exemplo, onde é possível observar as vantagens em trabalhar com zonas de vegetação esparsas é no artigo de Niese et al. [30] que propõem um método para gerar distribuições de vegetação em zonas urbanas, que faz uso de uma rede-neuronal de mudança de estilo para gerar uma máscara de cobertura de vegetação a partir de ortofotos (Figura 2.6).

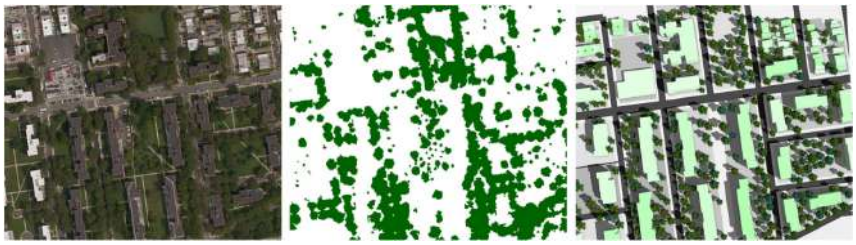


Figura 2.6: É possível ver a ortofoto de *input* à esquerda, a máscara de cobertura gerada pela rede neuronal no meio e por fim, à direita, a distribuição gerada, Imagem retirada de [30].

No entanto, há exemplos de procedimentos que conseguem detetar plantas em zonas de vegetação densa. Xiao et al. [45] combinam ortofotos com *DSM*'s de modo a conseguirem isolar o topo das árvores e fazer uma separação das copas. Também é possível observar estes resultados no trabalho de Andújar et al. [3], onde usaram uma técnica de deteção de copas de árvores que procura por formas de copas numa ortofoto. Estas copas são obtidas deformando uma área de um círculo com parâmetros de ruído. O resultado deste algoritmo é um conjunto de pontos tridimensionais que correspondem ao centro de massa de cada planta projetado no solo.

Outro fator crucial na geração procedimental de vegetação utilizando dados reais tem a ver com a forma como estes dados são usados. A solução mais direta é usá-los de uma forma interpretativa, onde estes são diretamente interpretados pelo modelo de geração [3]. No entanto, outra abordagem possível é usar os dados como sendo um parâmetro regulador da geração, onde o algoritmo de geração corre sem ter conhecimento dos dados reais, e o resultado do modelo é comparado com os dados reais, usando uma função de *fitness*, desta forma os dados são usados como informação de treino do modelo [30].

No artigo previamente mencionado [30], os autores usam as máscaras de cobertura do solo como um parâmetro de regulação da geração de vegetação. Isto é, como o processo de identificação da posição de cada planta individual é extremamente complexo, e ainda não existem soluções ideais para realizar esta tarefa, os autores optaram por limitar o espaço de parâmetros de *input* que o algoritmo de geração de vegetação recebe, estabelecendo padrões predefinidos de distribuição de plantas que se podem observar num ambiente urbano. De seguida, treinaram uma *CNN* (*Convolutional Neural Network*) que recebe como *input* uma máscara de cobertura do solo e dá como *output* um conjunto de parâmetros que podem ser usados pelo algoritmo de geração procedimental de vegetação (Figura 2.7).

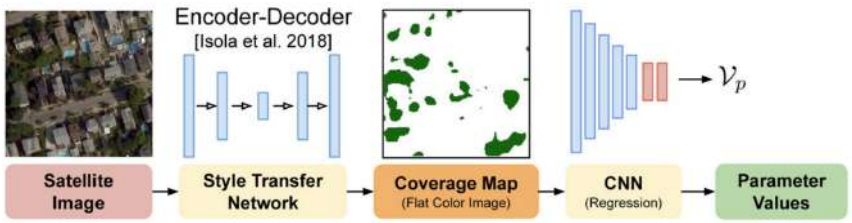


Figura 2.7: Processo de geração de parâmetros de *input* para o modelo gerar a distribuição de uma ortofoto, imagem retirada de [30].

De modo a treinar a *CNN* os autores, geraram um conjunto de dados aleatórios dentro do espaço de parâmetros que queriam explorar, transformando a distribuição da vegetação obtida numa máscara de cobertura da vegetação.

Por fim, podemos concluir que a integração de dados reais no processo de geração procedimental de vegetação é uma área ainda pouco estudada, que não explora muitas das ferramentas mais atuais do mundo da deteção remota, tendo assim potencial para se poder gerar vegetação ainda mais realista que espelha o que vemos no mundo real.

2.3 Coleção e tratamento de dados

É fundamental ter conhecimento dos dados usados na geração procedimental de vegetação, não só de modo a guiar as decisões que vão ser tomadas no processo de construção de uma solução, mas também para compreender os tratamentos necessários a aplicar aos dados recolhidos.

Na secção anterior 2.2, foram analisados vários conceitos de geração procedimental de vegetação, os quais podem ser aplicados neste trabalho. A secção terminou com a exposição de um conjunto de técnicas que fazem uso de dados do mundo real para gerar representações virtuais da vegetação. Os dados frequentemente utilizados por estas técnicas são ortofotos e *DEM*'s, embora não sendo realmente usados diretamente pelos modelos, sofrendo alguma forma de pré-processamento permitindo recolher informação adicional dos mesmos, tal como já foi referido anteriormente. Niese et al. [30] usam ortofotos em conjunto com uma rede neuronal para extrair máscaras de vegetação do ambiente.

No entanto, é pertinente analisar os parâmetros de *input* usados por outros modelos que não recorrem a dados reais. Estes dados, variam de acordo com o modelo considerado, por exemplo, a grande maioria dos modelos probabilísticos usa mapas de densidade sobre a forma de imagens *raster* para representar a forma de como a vegetação se encontra dispersa ao longo de um ambiente. Por contraste, os modelos de simulação recebem como *input*, valores de dados biológicos e não biológicos, que depois são usados no processo de simulação do crescimento e propagação da vegetação. Neste contexto, os fatores mais recorrentes são abióticos, nomeadamente, a luz solar, a humidade, a temperatura, o acesso a água e a qualidade do solo, bem como o fator biótico adicional da competição entre

plantas por território e outros recursos.

A grande maioria dos dados mencionados não podem ser extraídos diretamente do ambiente que se pretende recriar. No caso de mapas de densidade de vegetação, como iremos ver mais à frente, existem camadas de representação que nos permitem extrair informação sobre a densidade de vegetação de uma área específica. No entanto, esta informação é limitada, uma vez que não nos indica as espécies que se encontram presentes numa área específica, apesar de poder ser obtida através de cartas de ocupação do solo, nem o tamanho das árvores dispersas ao longo do ambiente.

Outra forma de obter um mapa de densidade de vegetação, é compila-lo a partir de informação abiótica do mundo. Nascimento et al. [29] constroem um mapa de densidade usando uma combinação de curvas de adaptabilidade de plantas individuais, com mapas que contêm informação abiótica, como a humidade, a inclinação do terreno, a altura, entre outros. Muitos destes dados podem ser obtidos num contexto do mundo real. Por exemplo, a altura pode ser obtida a partir dos dados de elevação do terreno, a inclinação pode ser compilada a partir dos dados de elevação e a humidade pode ser obtida através de camadas de alta resolução que mapeiam este valor. Neste contexto, o único valor que não se consegue obter de imediato corresponde aos dados individuais de cada planta.

Muitos dos dados usados pelos modelos de simulação podem ser extraídos diretamente do mundo real, havendo limitações relativamente à disponibilidade dos mesmos. O maior problema com este tipo de modelos é que, dada a sua natureza, e o facto de serem uma abreviatura do mundo real, o resultado obtido nem sempre corresponde ao pretendido.

Neste seguimento, não só é prudente fazer uma análise dos dados já mencionados, como também das ferramentas e técnicas que estão disponíveis, para fazer o levantamento destes dados. Assim, é pertinente fazer uma análise dos mapas de cobertura e uso do solo de maior interesse nacional a COS, e o CORINE, bem como das camadas de alta resolução que temos disponíveis que também podem ser relevantes.

2.3.1 COS e CORINE

Ao nível do território nacional, há duas ferramentas que nos permitem recolher informação da cobertura e uso do solo: a COS (Carta de Uso e Ocupação do Solo) e o CORINE *Land Cover*. Ambas são cartas de ocupação do solo que detêm informação sobre a organização do terreno. A COS foi desenvolvida no contexto do uso a nível nacional e por isso apenas contém informação de Portugal. Em contraste, o CLC (CORINE Land Cover) é um produto criado pelo programa CORINE, que é uma iniciativa europeia, cuja carta abrange todos os países membros da união europeia.

A COS é um mapa cartográfico de polígonos, onde os dados se encontram guardados na forma de imagens vetoriais, segundo o formato de *shapefiles* [7], onde cada polígono da COS inclui informação sobre uma unidade de ocupação/uso do solo. Estas unidades representam qualquer área de terreno com um valor maior ou igual à unidade mínima

considerada, que neste caso é um hectare, com uma distância entre as linhas do polígono maior ou igual a um valor pré-definido, que neste contexto é de vinte metros. De modo a que seja atribuída uma classe a uma unidade é preciso que esta domine 75% do solo presente na área considerada. Mesmo que haja outras classes dentro desta área as mesmas não podem representar mais de 25% do polígono gerado. A atribuição é feita associando um código representativo da classe ao polígono gerado.

As classes da COS encontram-se organizadas de uma forma hierárquica de acordo com o nível de detalhe que cada classificação, ou seja, as classes de maior detalhe encontram-se contidas dentro das de menor detalhe (Figura 2.8). Isto resulta num sistema de classificação onde a geração de uma classe mais detalhada, vai implicar que esta tem de ser interpretada de acordo com as assunções que foram feitas para as classes onde esta se encontra contida. O código de cada classe representa a posição da mesma na hierarquia de classes. É ainda importante notar que, quando um mapeamento é feito, a classe atribuída corresponde à mais detalhada que pode dada a esse polígono.

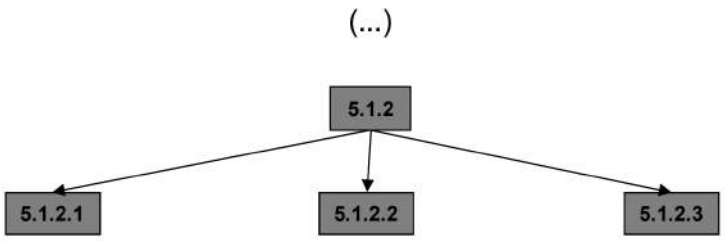


Figura 2.8: Estrutura hierárquica da COS, imagem retirada de [7].

A COS contém informação correspondente a uma série temporal de 5 anos de referência (1995, 2007, 2010, 2015 e 2018) (Figura 2.9), havendo produtos individuais para cada um destes anos. A versão mais atualizada da COS, a de 2018, contém 83 classes no total. Esta versão saiu oficialmente em 2019 [7], e de momento a DGT está a atualizar as versões anteriores da COS com as classes que foram usadas na versão mais recente . Está também a ser construída uma versão da ferramenta mais leve, com menos classes e que usa imagens *raster* por oposição a imagens vetoriais. Outro ponto que é importante mencionar, é que das 83 camadas presentes na versão mais recente da COS, 19 correspondem a superfícies florestais, onde 7 são dedicadas a classificar florestas para uso agrícola, enquanto as restantes 12 são florestas naturais.

Por contraste temos o CLC, que como já mencionado detém informação sobre todos os países da União Europeia. A COS segue os padrões de nomenclatura que foram estabelecidos por esta ferramenta, isto quer dizer que, muitas das classes que se encontram disponíveis numa ferramenta, também podem ser encontradas na outra. No entanto, o CLC tem menos classes que a COS, com um total de quarenta e quatro (44) classes e possui uma área de mapeamento mínima maior do que a COS, vinte e cinco (25) hectares por oposição ao um (1) hectare da COS, com uma distância mínima de cem (100) metros

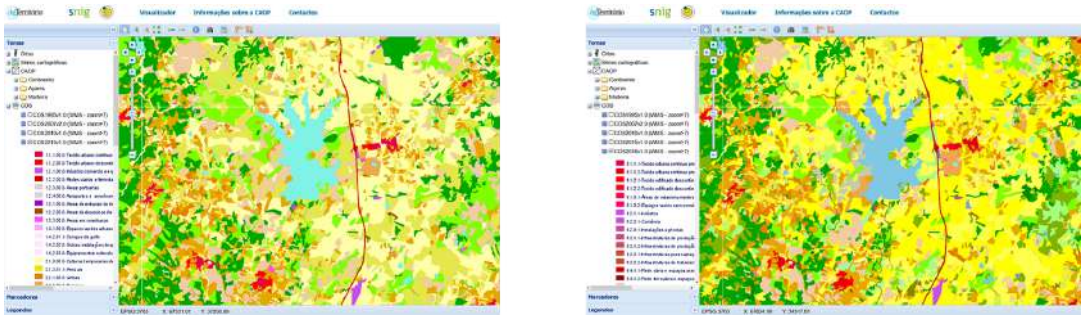


Figura 2.9: Duas representações temporais da COS, à esquerda vê-se uma representação versão de 2015, e à direita vê-se a de 2018, Imagem retirada de [42].

entre linhas. Estes valores são menos precisos porque esta carta de ocupação necessita de cobrir uma área maior do que a COS. Os dados da CLC encontram-se disponíveis sobre a forma de imagens vetoriais e *raster*.

Uma característica única do projeto CORINE, que não se encontra disponível com a COS, tem a ver com mapas de evolução da cobertura do solo. Estes dados fornecem-nos informação relativamente à forma como o terreno evoluiu ao longo do tempo.

Por fim, é possível observar a importância destas ferramentas no que toca ao contexto de determinação do posicionamento e organização da vegetação de um determinado território. No entanto, estas ferramentas, por si só, não nos fornecem informação suficiente, de modo a que seja possível gerar uma representação realista da vegetação.

2.3.2 Camadas de alta resolução

Outro conjunto de dados relevantes, mencionados anteriormente, correspondem às camadas de alta resolução. Estas camadas, também chamadas de temas de alta resolução, surgem no âmbito de um projeto europeu, que tem como objetivo principal complementar a informação já existente sobre a cobertura do solo dos membros da união europeia, mais precisamente, o CLC [13]. Com isto, foram produzidas sete camadas distintas.

Estas camadas são apresentadas sobre a forma de imagens Raster com uma resolução de vinte metros por píxel, isto quer dizer que cada píxel representa uma área de vinte por vinte metros. Estes temas são gerados a partir de um conjunto de dados de satélites que também são usadas para produzir os dados do CLC.

As sete camadas criadas [25, 23, 24, 26] são as seguintes: Grau de Coberto Florestal, Tipo de Folhagem Dominante, Tipo de Floresta, Pastagens, Grau de Impermeabilidade, Zonas Húmidas, Corpos de Água e Índice de Probabilidade de Zonas Húmidas. Destas camadas, as que apresentam maior interesse para este projeto sãos as que estão relacionadas diretamente com a cobertura vegetal do terreno, ou seja as duas primeiras. No entanto, as outras camadas também podem ser importantes na construção de uma simulação complexa de um ambiente.

O grau de cobertura florestal [23] é um produto que fornece um mapeamento das percentagens de vegetação florestal de uma determinada área, isto é, o valor de cada píxel

da imagem representa a percentagem da sua área que se encontra coberta por copas de árvores (Figura 2.10).

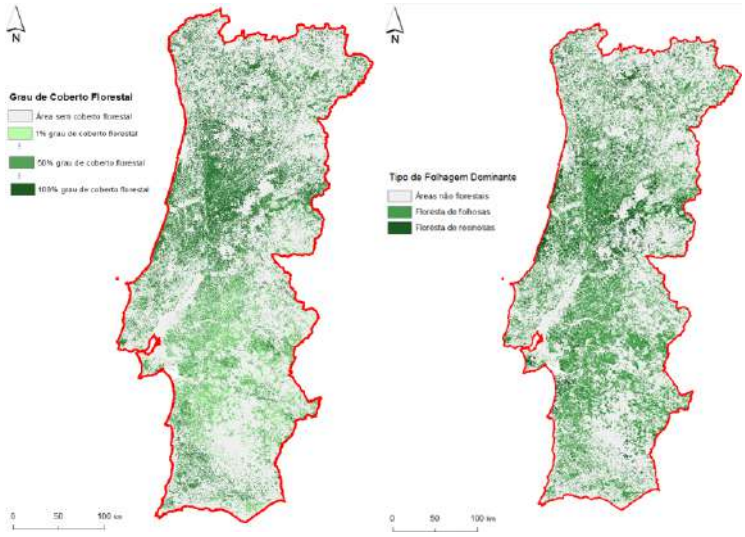


Figura 2.10: Representação do grau de cobertura florestal à esquerda e o tipo de folhagem na direita, imagens retirada de [23].

Outros temas de alta resolução que também são relevantes para este trabalho correspondem ao tipo de folhagem dominante e o tipo de florestas presente, fazendo uma classificação do tipo de vegetação dominante numa determinada área. O método de classificação é feito de forma semelhante ao que foi observado anteriormente com o CLC. A vantagem deste método, relativamente ao anterior, reside na classificação de áreas de vinte por vinte metros, resultando numa imagem que pode ser usada diretamente como máscara na camada do grau de cobertura florestal.

Outro tema de alta resolução importante corresponde à *Grassland* [24]. Esta camada fornece duas classes distintas, uma para classificar zonas com alguma forma de vegetação rasteira, podendo ser relvados ou pequenos arbustos, e outra para representar zonas sem vegetação.

Ainda no âmbito do tema das camadas de alta resolução, é importante referir as camadas existentes que fornecem dados sobre a forma como a informação contida nestas camadas evoluiu ao longo do tempo.Estes dados encontram-se limitados de acordo com a janela temporal em que foram recolhidos.

Por fim, podemos concluir que estes temas de alta resolução, em conjunto com os dados já mencionados anteriormente, têm a capacidade de fornecer informação suficiente para gerar uma representação realista do mundo concreto. No entanto, existem certos dados que não podem ser recolhidos simplesmente a partir dos já existentes, como por exemplo a idade da floresta, apesar de este poder ser inferido a partir das camadas de evolução.

2.4 Unreal

Até agora foram mencionadas inúmeras técnicas e produtos das várias áreas de conhecimento que são abrangidas por este trabalho, desde conceitos relacionados com a geração procedimental de vegetação em mundos virtuais, às várias ferramentas que nos permitem extrair factos ambientais do nosso mundo. No entanto, ainda não foi abordado o contexto onde este trabalho está a ser desenvolvido.

Neste seguimento, é prudente fazer uma análise da plataforma que vai ser usada ao longo deste trabalho, que, como já foi mencionado, é o motor de jogos *Unreal 4*. Este motor foi desenvolvido pela *Epic Games*, e é atualmente uma das ferramentas líder da indústria [36]. Por si só, construir uma ferramenta de geração procedimental de raiz na *UE4* é uma tarefa extremamente complexa que requer um conhecimento aprofundado da ferramenta.

Contudo já existe um conjunto de produtos desenvolvidos que permitem atenuar a dificuldade deste trabalho, mais precisamente ferramentas de geração procedimental de vegetação. Algumas destas ferramentas são desenvolvidas pelos estúdios internos da *Epic Games*, sendo desenvolvidas à volta dos sistemas já existentes na *ue4*, mais precisamente o sistema de *landscape*'s do *unreal*, enquanto que outras foram desenvolvidas por terceiros.

2.4.1 Sistema de Landscapes

Antes de avançar com a explicação das restantes ferramentas é importante mencionar o sistema onde estas se encontram inseridas. O sistema de *Landscapes* da *ue4* foi desenvolvido pela equipa da *ue4* com o objetivo de facilitar o processo de geração do terreno de cenários.

Esta ferramenta foi primariamente construída com artistas em mente e por isso a sua principal forma de interação roda à volta da utilização de pincéis de deformação do terreno, no entanto esta continua a apresentar alguns sistemas que facilitam a sua utilização dentro do contexto de sistemas de geração procedimental, mais precisamente, tem a capacidade de gerar representações do terreno com base em mapas de elevação sobre o formato de *DEM*'s do terreno.

Cada terreno gerado tem associado um conjunto de fatores que determinam a forma como ele é representado no ambiente, destes o que nos interessa mais no contexto desta tese é o material do terreno, que é responsável por determinar a cor com a qual o terreno está a ser renderizado.

Os materiais no *unreal* usam um sistema de nós visuais que é posteriormente compilado para *HLSL*. Os materiais dos terrenos podem usar todos os nós padrão disponíveis em outros materiais, no entanto, há um conjunto de expressões que só podem ser usadas no contexto do terreno do *unreal*. Para o tópico da geração de vegetação as expressões que nos interessam mais são as que estão diretamente relacionadas com a geração das camadas do terreno.

Neste contexto, as camadas presentes na *landscape* têm a capacidade de distinguir as várias zonas que estão a ser representadas no ambiente, guardando os seus resultados sobre o formato de mapas de densidade, os quais são depois usados pelas ferramentas de geração de vegetação para guiarem o processo de geração.

Cada camada tem um *LayerInfoObject* associado, sendo este objeto responsável por guiar a forma como as camadas interagem entre si em situações de interseção de várias camadas. Os pesos de cada camada podem ser definidos usando pincéis no ambiente, ou podem ser diretamente atribuídos através de uma textura.

Algo que também é importante mencionar brevemente é que o sistema de *landscapes* permite que haja uma divisão do terreno em várias *tiles*, isto reduz a quantidade de informação que está a ser carregada e consequentemente também reduz os tempos de renderização melhorando o desempenho dos projetos desenvolvidos.

2.4.2 Procedural Fooliage Tool

A ferramenta mais popular no mundo da *UE4* no que toca ao povoamento de um ambiente virtual com vegetação é a *Procedural Fooliage Tool* [37]. Esta ferramenta foi desenvolvida pela equipa do *Unreal* com o objetivo de estender as ferramentas já existentes de criação de mundos realistas, nomeadamente o sistema de *Landscapes*. É importante realçar que esta ferramenta não foi desenvolvida com o intuito de ser usada em tempo real, atendendo ao facto de que é uma ferramenta pesada em termos de *performance* computacional.

A referida ferramenta usa maioritariamente técnicas de simulação do ambiente e está estruturada à volta de duas classes principais, classes estas que no contexto da *UE4* são chamadas de atores. A primeira é a *Procedural Fooliage Spawner*, que é responsável por instanciar os vários objetos de vegetação no ambiente, e a segunda é a *Static Mesh Fooliage*, que corresponde à representação de um tipo de vegetação do ambiente a ser gerado. Um *spawner* pode possuir vários atores de vegetação sobre a forma de *Static Mesh Fooliage*'s, isto permite que um *spanwer* consiga gerar uma vegetação diversa com vários tipos de plantas. Uma outra classe, também importante para a geração de vegetação, apesar de não ser estritamente necessária, é a *Procedural Fooliage Blocking Volume* responsável por limitar as zonas onde a vegetação pode ser gerada definindo uma área em que os *spawners* não podem colocar plantas.

O processo de geração é controlado através de vários parâmetros presentes nos *spawner*'s e nos atores contidos em cada *spawner*. Destes parâmetros os que afetam mais a geração de conteúdo são os que se encontram presentes nos atores individuais. Estes estão agrupados de acordo com os fatores afetam durante o processo de geração, havendo um grupo de parâmetros para controlar a forma como cada planta é colocada e um grupo para determinar a forma como a planta faz a geração de conteúdo.

Este último conjunto de parâmetros é o que afeta mais a distribuição gerada, havendo valores que permitem controlar a forma como o sistema lida com colisões entre plantas, a maneira como as plantas são agrupadas no mundo e a forma como a vegetação cresce ao

longo do tempo.

Os parâmetros de colisão servem para garantir que duas plantas não são colocadas no mesmo local, usando um sistema de verificações radiais semelhante ao que foi observado na secção 2.2.1. Por outro lado, os parâmetros de agrupamento afetam a distribuição inicial das plantas e a forma como estas se vão propagando ao longo do tempo (Figura 2.11). Por fim, temos os parâmetros de crescimento das plantas os quais determinam a forma como as plantas se desenvolvem no ambiente estabelecido, usando valores como a idade máxima de uma planta e uma curva de crescimento para determinar o tamanho final de cada planta representada.

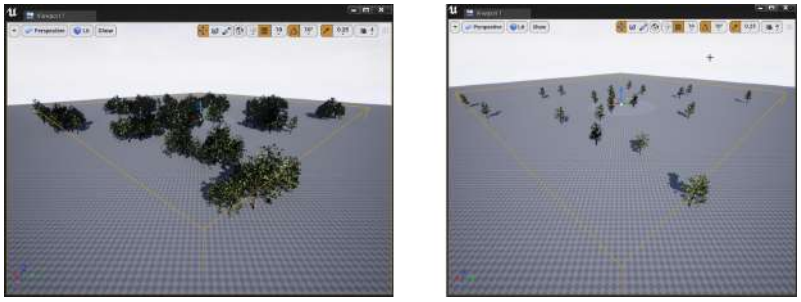


Figura 2.11: Comparação de dois cenários que foram gerados com parâmetros de agrupamento diferentes, imagem retirada de [37].

É importante notar que esta ferramenta tem a opção de funcionar usando o sistema de *landscapes* do *unreal* e permite que haja um controlo mais *top-down* da geração, onde podemos delimitar as zonas onde podemos observar vegetação usando várias camadas da paisagem. Estas camadas são estabelecidas no material da *landscape*, e os pesos das camadas são estabelecidos no editor do motor de jogos, e definem as máscaras de densidade que vão guiar a geração da *Procedural Foliage Tool*. No entanto, esta integração da ferramenta tem imensos problemas que impedem um controlo total da geração usando apenas mapas de densidade.

Com isto, esta ferramenta é muito poderosa no que toca a geração de distribuições de vegetação em mundos virtuais, no entanto, possui algumas limitações. Como se trata maioritariamente de um modelo de simulação possui todas as limitações que podem ser encontradas neste tipo de modelos, sendo muito pesada em termos de desempenho computacional e os *outputs* gerados serem imprevisíveis dado os valores fornecidos, apesar de que isto é parcialmente corrigido usando o sistema de *landscapes* da *ue4*. Adicionalmente esta ferramenta, por si só, não possui a capacidade de representar distribuições de vegetação já feitas sobre o formato raster. Outra desvantagem da *Procedural Fooliage Tool* é que não é uma ferramenta ideal para representar modelos de vegetação que precisem de diferenciar camadas distintas.

2.4.3 Ferramenta Grass

À semelhança da *Procedural Fooliage Tool* a ferramenta *Grass* [38] também foi desenvolvida pela equipa do *Unreal*. Contudo, esta ferramenta encontra-se diretamente ligada ao conjunto de ferramentas de *Landscape*, não podendo ser removida deste contexto.

Esta ferramenta tem como objetivo principal gerar a vegetação rasteira para uma determinada paisagem, havendo a necessidade dos utilizadores definirem uma malha e um material para a mesma. Como esta ferramenta se encontra ligada ao sistema de *Landscape*, é preciso definir um outro material que represente o solo onde a vegetação se encontra. Para além disto, o material associado à malha representativa da vegetação pode ser ligado ao material usado para representar o ambiente, gerando assim uma associação entre o material do terreno e o material da vegetação, o que aumenta o realismo presente no cenário.

Como já foi mencionado anteriormente, este produto foi desenvolvido para o sistema de *Landscape*, que utiliza primariamente pincéis e outras ferramentas de decoração de cenários para gerar os ambientes que se pretendem representar, os quais podem ser usados pelo sistema *Grass* (Figura 2.12).

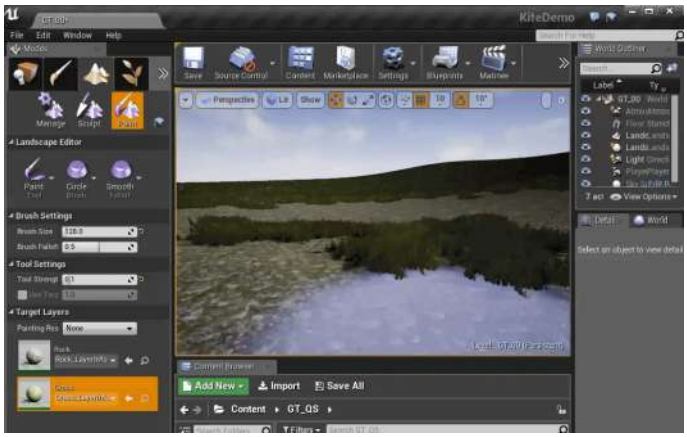


Figura 2.12: Pincéis disponíveis na ferramenta *Grass*, imagem retirada de [38].

Contudo, esta ferramenta apresenta limitações comparativamente à *Procedural Fooliage Tool*. Em primeiro lugar, não pode ser usada fora do contexto do sistema de *Landscapes*, que é mais usado num contexto artístico. Para além disso, é um sistema que não usa qualquer simulação de colisões significando que o uso desta ferramenta fora do contexto para o qual foi desenvolvida iria gerar representações irrealistas.

2.4.4 Ferramentas de terceiros

Existem poucas ferramentas de geração procedimental de vegetação públicas desenvolvidas por terceiros. Contudo, é pertinente referir que uma abordagem muito comum no que toca a geração de conteúdo procedimental para a *UE4*, passa por usar um produto chamado *Houdini*, mais precisamente, a *Houdini Engine* [35] que contém uma integração direta com o motor de jogos.

A *Houdini* corresponde a um conjunto de ferramentas procedimentais, que foram desenvolvidas com artistas em mente. Este produto possui uma interface de programação em nós, que tem como objetivo principal criar *pipelines* de desenvolvimento para gerar conteúdo de uma forma procedimental. A *Houdini Engine* é uma ferramenta usada em conjunto com um motor de desenvolvimento de jogos, tendo suporte para *Unreal* e para o *Unity*. Esta ferramenta está organizada de forma a que do lado do *Hudini* o utilizador tenha acesso a todas as ferramentas que se encontram disponíveis neste produto, podendo criar conteúdo e exportá-lo de forma imediata para o projeto com a qual a *Houdini Engine* se encontra integrada.

Na conferência da GDC de 2018 [22], os apresentadores desenvolveram um sistema que faz uso da *Houdini Engine* em conjunto com uma versão modificada da *Procedural Fooliage Tool*. Utilizam o *Houdini* para gerar um conjunto de pontos dispersos que depois alimentam à *Procedural Fooliage Tool* para correr uma pequena simulação do crescimento das plantas. O motivo para usarem este sistema tem a ver com o facto dos autores se focarem na criação de um pipeline que ofereça controlo aos artistas por oposição a uma simulação realista do crescimento da vegetação. De forma a obterem maior controlo sobre a vegetação gerada, acabaram por fazer um conjunto de alterações à *Procedural Fooliage Tool*.

3

Abordagem

Este capítulo tem como objetivo principal apresentar a abordagem que vai ser adotada de modo a gerar a camada de vegetação de um mundo virtual, que seja uma aproximação da realidade, fazendo uso de um conjunto de dados reais e produtos que foram construídos usando técnicas de deteção remota. O desenvolvimento desta abordagem tem em conta a investigação que foi realizada durante o período de preparação da dissertação e exposta no capítulo anterior [2].

Assim, este capítulo começa por abordar as várias experiências que foram realizadas durante o período de preparação e as conclusões que podemos tirar das mesmas [3.1]. De seguida, é exposta a arquitetura do sistema que se pretende construir durante esta tese [3.2] em conjunto com uma análise da forma como os dados reais são integrados na solução proposta. O capítulo termina na secção [3.3] com uma breve análise do plano de desenvolvimento inicial, onde são explorados os pontos que foram cumpridos e os que foram deixados de parte.

3.1 Experiências realizadas

Durante o período de preparação de dissertação foram realizadas várias experiências, não só de forma a ganhar um maior entendimento das ferramentas que vão ser usadas durante o trabalho, mas também, para saber os dados que estão disponíveis e as técnicas de processamento que podem ser usadas. Com isto, os testes de maior interesse foram os realizados com as ferramentas mencionadas na secção [2.4], e os testes à volta dos dados apresentados na secção [2.3].

Principiando pelas experiências realizadas com as ferramentas da *UE4*, foram realizados testes com a *Procedural Fooliage Tool* e com a ferramenta *Grass*. Como já foi mencionado anteriormente a *Procedural Fooliage Tool* usa técnicas de simulação de modo a gerar a camada de vegetação, fornecendo ao utilizador um conjunto de parâmetros, que podem ser manipulados de forma a guiar o algoritmo de geração.

Esta ferramenta tem um conjunto de características que não a torna adequada ao contexto deste projeto. Primeiro esta assume primariamente uma abordagem *bottom-up*

no que toca a geração de conteúdo, isto quer dizer que os dados que esta recebe são maioritariamente atributos de plantas individuais. Por contraste os dados que conseguimos obter a partir dos mapas de cobertura do solo, como a COS e o CLC, e de camadas de alta resolução, são dados mais descritivos da forma como ambiente se encontra organizado.

Contudo, a *Procedural Foliage Tool* consegue-se adaptar a uma metodologia de geração estruturada de uma forma mais *top-down* utilizando o sistema de *landscape* do *unreal*, contudo este sistema tinha um conjunto de problemas que limitavam a geração de conteúdo. O trabalho ao qual estou a dar continuação focou-se em tentar resolver os vários problemas desta ferramenta. No entanto, esta solução continua a ter limitações: tem dificuldade em gerar várias camadas de vegetação, não integra dados reais no processo de geração, e para além disso o processo de geração não está automatizado.

Outra limitação da *Procedural Fooliage Tool* tem a ver com o facto desta ferramenta ainda estar em estado de desenvolvimento, ou seja, a versão que que se encontra disponível para o público não é o produto final que está a ser desenvolvido pela equipa da *Epic*. Consequentemente a documentação da mesma ainda é muito limitada.

Também foram realizados testes com a ferramenta *Grass*. Como já foi mencionado, esta ferramenta está internamente ligada ao sistema de geração de paisagens do *Unreal*, é usada maioritariamente por artistas, e oferecem pouco suporte de relativamente a alterações do seu código.

Assim, esta ferramenta apresenta dois problemas. Em primeiro lugar, não pode ser usada fora do contexto onde se encontra inserida, o sistema de *Landscape* que, como já foi mencionado, tem suporte limitado no que toca à realização de alterações do código fonte, no entanto tem imensas opções que permitem a sua utilização no desenvolvimento de materiais do *unreal*. Em segundo lugar, e mais notável, esta ferramenta não suporta a verificação de colisões entre plantas durante a geração, o que faz com que várias plantas possam ser geradas umas em cima das outras.

No contexto deste trabalho, o segundo problema faz com que esta ferramenta seja pouco viável para guiar a geração da vegetação completa de uma paisagem, uma vez que é preciso garantir que não há colisões entre as plantas colocadas no ambiente, a não ser para a representação de vegetação rasteira.

Ainda durante o período de preparação, foi realizada uma análise dos dados que foram mencionados na secção [2.3], ou seja os dados da COS, do CLC e das camadas de alta resolução. Esta análise foi conduzida com o objetivo de adquirir um maior entendimento dos dados que vão ser usados ao longo do desenvolvimento, e da forma como estes podem ser integrados no processo de geração. Começando pelos dados da COS e do CORINE, estes mapas de cobertura do solo encontram-se sob o formato de imagens vetoriais, mais precisamente *shapefiles* podendo ser rasterizadas na dimensão que se pretenda.

Por contraste, as camadas de alta resolução, têm uma resolução fixa de píxeis com áreas de vinte (20) metros quadrados. No entanto, estas imagens também podem ser ampliadas usando técnicas de *upsampling*.

De forma a ganhar um melhor entendimento dos dados que podemos obter a partir

da camada de cobertura florestal, foi realizado um teste com o objetivo de gerar um mapa de densidade de vegetação em tons de cinzento, indexando vários valores do mapa de cobertura segundo os valores percentuais apresentados na tabela de cores. Este teste permitiu verificar que os dados de cobertura podem ser usados diretamente para gerar mapa de densidade de vegetação. Os resultados encontram-se visíveis na seguinte figura 3.1.

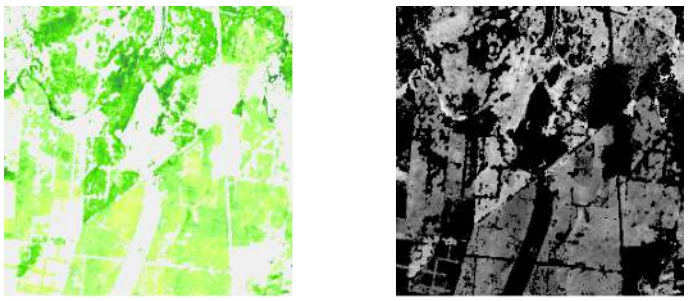


Figura 3.1: Esta imagem mostra o input que o processo de geração recebe (a), e o mapa de densidade que é gerado através da indexação dos valores da tabela de cores (b).

Um dos problemas encontrados na fase de preparação corresponde ao facto de que a COS possui um total de oitenta e três (83) classes, muitas das quais não têm relevância no contexto deste trabalho. Assim, foi realizada uma seleção das classes de maior importância da COS. Este processo começou por fazer a exclusão de todas as classes da carta que não representassem áreas com vegetação. Dada esta filtragem, sobram dezanove (19) classes que são usadas para classificar zonas florestais, sete (7) dedicadas a classificar florestas para uso agrícola, e doze (12) correspondem a florestas naturais. Estas dezanove (19) classes continuam a representar detalhes excessivos para os dados que pretendemos obter. Assim, de modo a reduzir ainda mais o número de classes foi feito um agrupamento das mesmas.

Como já foi referido, a COS apresenta uma estrutura hierárquica, isto quer dizer que é possível fazer uma redução destas classes, de acordo com a classificação onde se encontram inseridas na hierarquia.

A forma mais imediata seria agrupar as classes da vegetação em grupos para representar florestas resinosas e folhosas, no entanto, esta solução tem limitações dentro do contexto que queremos representar a vegetação, que é em situações de combate a incêndios. Isto acontece, uma vez que o agrupamento mencionado não faz a distinção entre plantas mais resistentes a incêndios, de plantas menos resistentes. Por exemplo, a grande maioria das árvores de tipo folhoso apresentam uma grande resistência à proliferação de incêndios, com a exceção do eucalipto que é uma planta de alto risco de incêndio.

Assim, uma forma mais informada de agrupar a vegetação é de acordo com o nível de resistência apresentado pela planta, podendo ter plantas resistentes, plantas com uma resistência média e plantas de alto risco.

Estas experiências permitiram ganhar um maior entendimento das ferramentas que foram apresentadas ao longo do segundo capítulo.

3.2 Arquitetura da abordagem

No seguimento da secção anterior é importante fazer uma análise da arquitetura da abordagem proposta a qual tem como principal propósito servir de guia na construção do sistema de geração (Figura 3.2).

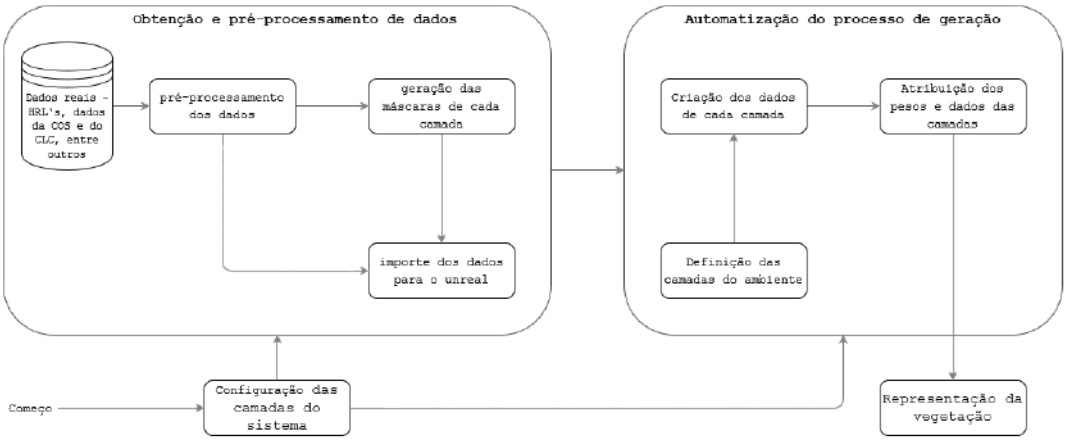


Figura 3.2: Esquema que mostra a arquitetura que vai guiar o desenvolvimento.

Esta arquitetura encontra-se organizada em duas fases distintas: a primeira corresponde à obtenção e o pré-processamento dos dados, e a segunda corresponde à automatização do processo de geração. A aquisição e o tratamento dos dados visa automatizar o processo de obtenção e processamento dos dados, construindo mapas de cobertura florestal para cada camada que queremos representar no ambiente. Estes dados são posteriormente importados para o *unreal* onde serão fornecidos à ferramenta de geração de vegetação.

Começando pela aquisição e pré-processamento dos dados, nesta fase o sistema tem como principal objetivo gerar uma mapa de densidade de vegetação por cada camada que queremos representar no ambiente. É importante notar que cada camada é composta por uma ou mais classes da COS ou da CLC, e está associada ou a um tipo de vegetação rasteira, ou a árvores de maior dimensão, ou uma combinação destes dois tipos de vegetação. Cada camada também está associada a um material do solo.

Esta forma de agrupar as classes permite gerar representações distintas da que podemos observar na COS e na CLC, onde as classes estão organizadas de acordo com o tipo de folhagem que apresentam, o que faz com que plantas como o eucalipto sejam colocadas em conjunto com carvalhos, apesar da primeira ter um nível de risco muito maior do que a segunda.

Assim, o sistema de agrupamento apresentado não só tem a capacidade de gerar a representação que foi apresentada na secção anterior [3.1], como também pode ser usado para gerar a representação nativa das cartas de ocupação do solo.

A camada de vegetação mais importante para representar de forma precisa é a que contém as árvores de grande dimensão, porque representa a maior carga de combustível e é também a que permite fazer um melhor reconhecimento do local. O mapa de vegetação desta camada, pode ser obtido a partir de uma das camadas de alta resolução que foram mencionadas na secção 2.3.2, mais precisamente o nível de cobertura florestal. Como já foi mencionado anteriormente, este tema de alta resolução fornece informação percentual do nível de cobertura florestal de uma determinada área. Também já foi visto, que é possível gerar mapas de densidade de vegetação a partir destes dados, colocando a imagem em tons de cinzento, através das tabelas de cor que são fornecidas nos metadados. Estas imagens podem ser posteriormente ampliadas de modo a obter a resolução desejada.

Dentro do contexto deste trabalho, a representação dos níveis de vegetação restantes não tem de ser tão precisa quanto a vegetação mais alta, considerando que este tipo de plantas não são um fator determinante no combate a incêndios. Portanto os mapas de densidade destas camadas podem ser gerados a partir de uma compilação de mapas com parâmetros abióticos.

Alternativamente, pode nem ser preciso gerar mapas de densidade para as zonas de mato e vegetação rasteira, sendo somente necessário garantir que não há colisões entre as plantas, e que estas são geradas de acordo com os dados observados nas cartas de ocupação do solo.

Esta forma de agrupar as classes da COS também garante que cada camada é constituída apenas pelas camadas que queremos representar no ambiente, garantindo que os mapas de densidade gerados têm em conta as zonas onde não pode ser gerada vegetação, como por exemplo estradas e rios, visto que as classes que representam estes não têm de ser incluídas nas camadas de vegetação. Por uma questão de simplicidade todas as máscaras *raster* terão a mesma resolução.

A segunda fase do sistema, a automatização do processo de geração pode ser decomposta em três fases. A primeira é o importe dos mapas de cobertura gerados na fase anterior, a segunda é a geração de todos os dados que são necessários para realizar a geração de vegetação usando as ferramentas anteriormente mencionadas [3.1] e, por fim, a terceira fase do processo passa por atribuir todos os dados necessários para realizar a geração.

A primeira fase da automatização passa por fazer importe de todos os dados que foram gerados na primeira fase do processo, ou seja os mapas de densidade de todas as camadas consideradas. Este passo garante que a criação de cada mapa não tem de ocorrer necessariamente dentro do *unreal*, o que facilita o processo de compilação das imagens que vão ser usadas, porque permite que sejam usados programas externos. Há várias formas de automatizar o processo de de importe dos dados, pode ser feito usando *blueprints*, c++ ou a interface de *python* do *unreal*.

A segunda fase do processo de automatização passa por gerar todos os dados que têm ser usados para o processo geração de vegetação, ou seja os dados necessários do sistema de *landscape*, todos os dados que informam a geração de vegetação usando a *Procedural Foliage tool* e todos os dados da ferramenta *Grass*. Estes dados têm de ser criados durante esta fase porque têm de ser guiados de acordo com as camadas que queremos representar no ambiente.

Por fim, a terceira fase do processo de automatização passa por atribuir todos os dados necessários para realizar a geração de vegetação usando a ferramenta *Grass* e a *Procedural Foliage Tool*, durante esta fase também é preciso atribuir os pesos de cada camada que foram importados no primeiro passo.

Como já foi mencionado anteriormente nesta secção cada camada gerada pelo sistema corresponde a um conjunto de classes das cartas de classificação do solo. Esta forma de representação do dados é expressiva o suficiente para conseguir assumir a representação que foi mencionada na secção anterior [3.1], e estender-se para outras representações como por exemplo a nativa da COS que agrupa classes com base na espécie de plantas que podemos encontrar no ambiente.

Assim, cada camada representa um conjunto de classes e está associada a um material do terreno e a um tipo vegetação que pode ser, ou vegetação rasteira, ou árvores de pequena e grande dimensão, ou uma combinação de vegetação rasteira e árvores, ou ainda uma zona sem vegetação. Esta forma de atribuir um tipo de vegetação por camada permite que duas camadas distintas partilhem a mesma classe, o que que quer dizer que conseguimos fazer uma representação de florestas mistas.

É importante notar que cada camada da vegetação vai estar mapeada segundo uma camada do material da *landscape*, que como já foi mencionado são posteriormente usadas para guiar a vegetação do ambiente.

Para representar cada modelo de vegetação do ambiente vão ser usadas malhas estáticas. Cada malha representa um tipo de vegetação, podendo haver malhas para a vegetação rasteira e para as árvores de média e grande porte, o sistema vai ter uma conjunto de tipos de vegetação já atribuídos, mas o sistema deve ser configurável o suficiente de modo a que novos utilizadores possam adicionar novos tipos de vegetação.

3.3 Plano de desenvolvimento

Esta secção tem como principal objetivo apresentar a proposta inicial do planeamento de tarefas em conjunto com as tarefas que foram realizadas ao longo da dissertação de mestrado. O plano inicial foi criado com base na abordagem que foi inicialmente proposta, esta não refelete a abordagem que foi tomada durante o desenvolvimento. Consequente muitos dos pontos do desenvolvimento inicial foram alterados e atualizados de acordo com as necessidades da nova abordagem.

O plano inicial passava pelo desenvolvimento do sistema se estender até à primeira semana de agosto com o resto do tempo dedicado ao processo de avaliação, a pequenas

correções e a escrita do documento dissertação. Dado as condições adversas do ano passado o desenvolvimento acabou por se estender até meados de outubro e processo de avaliação e escrita do documento estendeu-se para novembro.

Assim as tarefas que foram planeadas e realizadas após o estabelecimento do plano de tarefas inicial foram as seguintes

- **Período de preparação:** Esta fase da elaboração teve como principal objetivo fazer correções ao trabalho elaborado durante a fase de planeamento, e a organizar a plataforma de desenvolvimento que vai ser usada ao longo do processo de desenvolvimento;
- **Automatização do processo de obtenção dos dados:** Este período de tempo foi dedicado a automatizar o processo de obtenção dos dados necessários e a fazer uma avaliação da qualidade dos mesmos;
- **Implementação da fase de pré-processamento:** Esta fase da elaboração teve como principal objetivo desenvolver a componente do pré-processamento da solução proposta, o desenvolvimento desta componente está dependente da automatização do processo obtenção dos dados;
- **Implementação do processo de automatização do lado do *unreal*:** Este período do desenvolvimento foi dedicado a investigar e desenvolver o processo de automatização do sistema de geração do lado do *unreal*, começando com uma investigação do funcionamento dos sistemas do motor de jogos, seguido da implementação de uma solução;
- **Correção de erros no sistema desenvolvido:** Esta fase da elaboração foi dedicada ao processo de deteção e correção de erros do sistema desenvolvido. Com um principal foco em elaborar uma implementação completa do projeto;
- **Processo de avaliação:** Esta fase de desenvolvimento focou-se em realizar um conjunto de tarefas de avaliação do sistema recolhendo dados sobre o desempenho do projeto segundo um conjunto de métricas;
- **Escrita da dissertação:** Esta foi a última fase da tese e correspondeu ao processo de escrita da dissertação.

4

Implementação e Desenvolvimento

Este capítulo explora a implementação do sistema, fazendo uma análise detalhada de todos os passos necessários para gerar a vegetação de um cenário na *ue4*, usando as ferramentas que o motor fornece em conjunto com outros sistemas externos. Esta implementação guia-se pelo sistema que foi proposto no capítulo anterior [3], focando-se na automatização do processo de geração de vegetação com a integração de dados reais no processo de geração.

Assim, o capítulo começa com a análise geral do algoritmo desenvolvido [4.1] onde são expostas não só as várias componentes do algoritmo, como também as técnicas e ferramentas que são usadas em cada parte deste processo. Na secção seguinte vai ser feita uma análise da primeira fase do algoritmo, a obtenção e o processamento de dados [4.2.3]. De seguida é feita uma breve análise dos constituintes da interface gráfica que é usada para facilitar a interação com o sistema [4.3]. No ponto 4.4.3 é feita uma análise da segunda fase do processo de automatização, ou seja o processo de automatização da criação de vegetação do lado do *unreal*. Por fim, nos últimos pontos: 4.6, 4.8, 4.9 e 4.10, são abordadas, respetivamente, uma análise da criação do *plugin* do sistema, uma análise detalhada do material gerado pelo sistema, uma análise do sistema de gravação e uma exploração das várias opções que podem ser escolhidas por camada.

4.1 Visão global do sistema

Como já foi mencionado anteriormente, este algoritmo foi desenvolvido com o motor de jogos *ue4* em mente com o objetivo não só integrar dados reais no processo de geração de vegetação de um cenário do *unreal*, mas também automatizar e consolidar o processo de geração de vegetação. Antes de avançar para a explicação é importante fazer uma análise completa do processo de geração para compreender os vários passos do sistema desenvolvido.

Para começar é preciso obter os dados que vão ser usados para guiar a geração da vegetação, estes correspondem a mapas de densidade que ditam a distribuição de cada camada do ambiente. Para obter os dados de cada camada é preciso fazer uma combinação

dos dados das cartas de ocupação do solo com os dados de cobertura florestal, caso seja necessário representar a vegetação de um determinado local. Para obter cada um destes dados podem ser usadas API's de serviços remotos, ou podemos usar dados localmente, com imagens *raster* ou imagens vetoriais.

De seguida é importante fazer uma visão geral do processo de geração de vegetação no *unreal* usando as ferramentas já mencionadas [2.4.2, 2.4.3]. Assim, assumindo que o sistema já tem os dados que vão ser usados para gerar a vegetação, ou seja, os mapas de densidade de todas as camadas que queremos representar no ambiente, o primeiro passo passa por informar a *engine* das camadas que vamos querer representar no ambiente, usando o material da *landscape*. Como já foi mencionado anteriormente, há uma relação de um para um no que toca os mapas de densidade criados e as camadas criadas no material do *unreal*.

Assim que as camadas estiverem definidas é preciso garantir que os dados de cada camada estão a ser gerados corretamente, estes correspondem às *LayerInfo* de cada camada, as malhas estáticas da vegetação que se quer representar no ambiente e o *Procedural Foliage Spawner* que é responsável por controlar a forma como a vegetação é gerada no ambiente.

Logo que todos os dados sejam gerados, o passo seguinte passa pela importação dos pesos das camadas que vão ser representadas. No editor do *unreal* este processo passa por selecionar a camada que queremos importar, carregar na opção de importação e escolher a textura que se quer usar para importar os pesos das camadas. É importante notar que quando os pesos da camada não tem a mesma resolução que a malha da *landscape* a textura é cortada ou estendida até ficar com a mesma resolução que a malha.

Assim, podemos verificar que o processo de geração pode ser estruturado segundo uma espécie de *pipeline* onde cada fase do processo está dependente da que vem anteriormente, por exemplo, o pré-processamento dos dados não pode ser feito antes de se obterem os dados das camadas que queremos representar, e a mesma lógica aplica-se às restantes fases do processo.

Esta forma de organizar o processo de geração pode ser divida em duas fases (Figura 4.1) : a primeira corresponde ao processo de obtenção e processamento dos dados, e a segunda fase corresponde à automatização do processo de geração.

Foi decidido logo desde o início que uma grande parte sistema ia ser desenvolvido com a API de python para o *unreal* [41]. Esta decisão foi tomada por um conjunto de motivos. Primeiro porque a interface de *python* do *unreal* fornece ao utilizador acesso a quase todos os elementos do editor do motor de jogos, permitindo criar novos objetos do *unreal*, e mudar valores do editor. Em segundo lugar a integração de *python* no processo de geração permite que sejam usadas várias bibliotecas externas que facilitam a obtenção e o processamento de dados. Para além do já mencionado, *scripts* de *python* podem ser chamados a partir do editor do motor de jogos ou através de uma *blueprint*, o que facilita a integração de vários sistemas que não estejam acessíveis dentro da API de *python*.

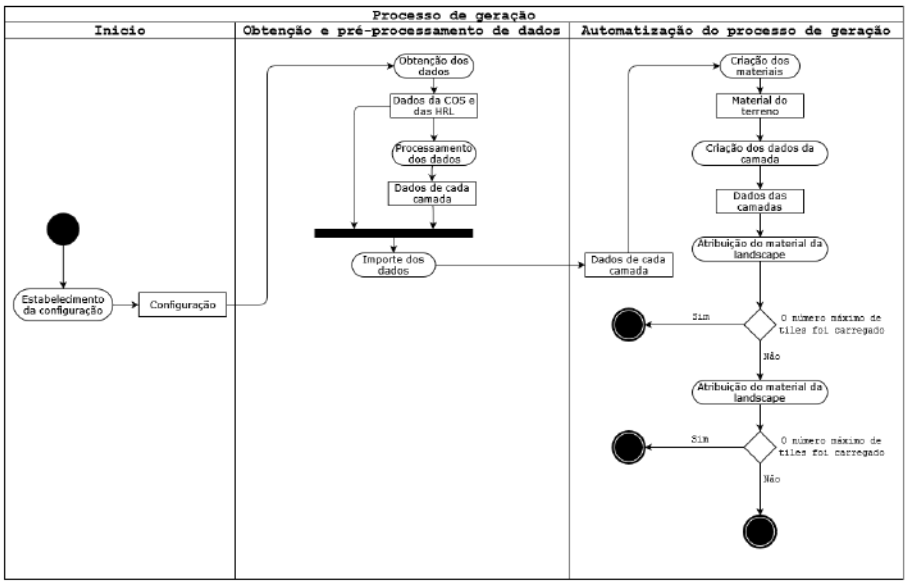


Figura 4.1: Diagrama de actividades do sistema atual.

Dada a estrutura do algoritmo onde cada fase está dependente do que vem anteriormente, o processo de desenvolvimento começou por se focar inicialmente no desenvolvimento dos sistemas de recolha e processamento de dados e só depois é que houve um foco na automatização do processo de geração de vegetação.

4.2 Obtenção e processamento dos dados

Na secção anterior verificámos que o processo de geração podia ser dividido em duas fases, sendo a primeira a obtenção e o processamento de dados. Esta fase do sistema tem como principal objetivo gerar os mapas de densidade para todas as camadas que se querem representar no ambiente.

Antes de avançar para os detalhes da implementação é importante relembrar que esta componente do sistema tem de ter a capacidade de agrupar várias classes da carta de ocupação do solo num conjunto de camadas de representação do ambiente, e usar estas camadas em conjunto com os dados da camada de cobertura florestal de modo a gerar mapas de densidade para todas as camadas que representação zonas com vegetação no ambiente. Para além de ter a capacidade de gerar mapas de densidade para zonas com vegetação, também era do nosso interesse que o sistema conseguisse gerar máscaras binárias para representar zonas sem vegetação, que sigam a distribuição do mundo real.

Esta componente do sistema funciona de forma praticamente independente do *unreal*, e é composta por vários *scripts* e classes. As classes mais importantes para a obtenção de dados são a classe *DataCollector*, que é responsável por interagir com os serviços de obtenção de dados e a classe *ShapeFileProcessor*, que é responsável pelo processamento de

dados locais, mais precisamente imagens vetoriais. Relativamente ao processamento de imagens a classe mais importante é o *ImageProcessor* (Figura 4.2).

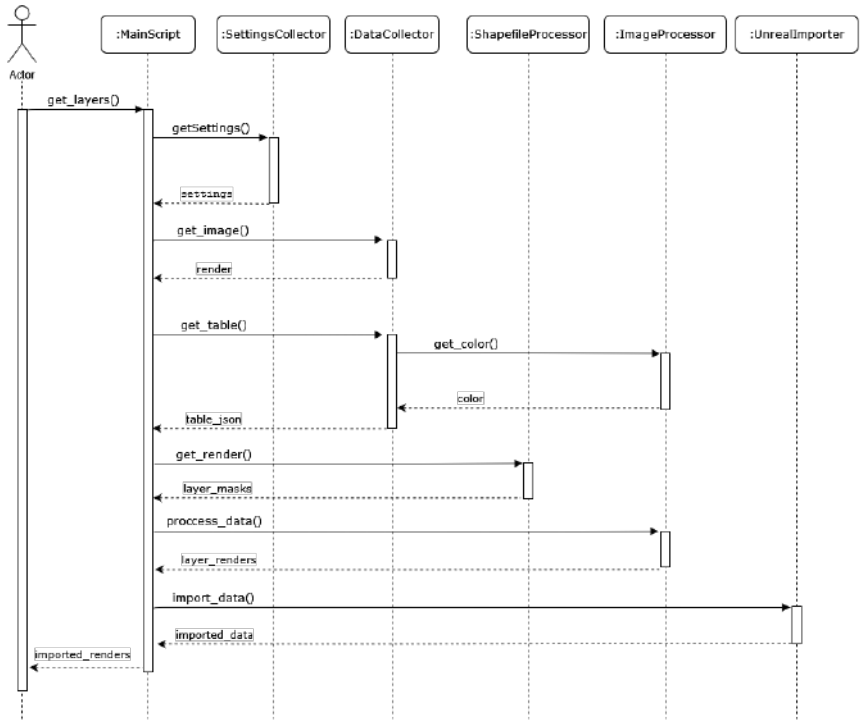


Figura 4.2: Diagrama de sequência do processo de obtenção e processamento dos dados.

4.2.1 Obtenção dos dados usados na geração

O primeiro passo da primeira fase de desenvolvimento corresponde à obtenção dos dados que vão ser usados na geração dos mapas de cada camada. Na iteração atual o programa permite que o utilizador apenas defina os limites do ambiente que queremos representar e o sistema de coordenadas que foi usado para os definir e a partir daí o sistema obtém todos os dados necessários.

Antes de avançar para a implementação, é importante relembrar os dados que estamos a tentar obter através desta componente do sistema, que são:

- Os dados das cartas de ocupação do solo, a COS e o CLC;
- Os dados das camadas de alta resolução, mais precisamente os da camada de cobertura florestal;
- A legenda de cada imagem obtida.

Como já foi mencionado anteriormente, estes dados podem ser obtidos remotamente usando a API rest de serviços de obtenção de dados, ou podem ser usados localmente; cada uma destas formas de obtenção dos dados tem as suas vantagens e desvantagens.

No que toca à utilização de serviços remotos a maior vantagem é que a aplicação não tem de ter os dados guardados localmente, o que reduz imensamente o peso do sistema de desenvolvimento. Para além disso, também pode resultar numa melhor performance computacional, por exemplo, no caso do processamento de *shapefile*'s, se houver um serviço que faça automaticamente o recorte do polígono este não tem de ocorrer localmente, podendo assim usar sistemas com maior capacidade computacional. Por contraste ao estarmos a usar um sistema remoto, estamos dependentes da disponibilidade destes serviços para conseguir realizar a geração de um determinado local.

Por outro lado, utilizar os dados localmente também tem as suas limitações, nomeadamente, o recorte das zonas que queremos representar tem de ser feito localmente, o que pode ser um processo pesado e demorado. Para além disto utilizar os dados localmente tem a desvantagem adicional de que o sistema fica mais pesado. No entanto, usar os dados localmente garante que o sistema não fica dependente da disponibilidade de serviços externos.

Para o sistema foi decidido, inicialmente, que se fosse possível íamos optar pela utilização de serviços remotos, isto porque um dos objetivos do sistema é que este fosse desenvolvido como sendo um *plugin* do *unreal*, de modo a que pudesse ser transferido entre vários projetos. Dada esta necessidade o ideal era que o projeto não fosse demasiado pesado e não tivesse dados desnecessários. Para além disto, as API's disponíveis fornecem um conjunto de serviços que facilitam o processo de processamento dos dados.

Começando pelas camadas de alta resolução, estes dados são obtidos usando os serviços *rest* da *ArcGIS*. Estes serviços foram desenvolvidos pela *Copernicus* e têm uma operação em comum que permite exportar imagens com base num conjunto de valores de *input*. Os mais relevantes para este projeto são os seguintes (Listagem 4.1):

- Os limites da zona que se quer representar;
- A referência espacial dos limites definidos;
- A resolução da imagem que estamos a tentar obter;
- O formato dos dados que queremos obter;
- O tipo de interpolação que os dados levam ao serem ajustados para a resolução e projeção desejada, esta pode ser NN (Nearest Neighbor), interpolação bilinear ou convolução cúbica.

Listagem 4.1: Pedido feito à API da ArcGIS.

```
1 {
2     'bbox': 'x_min,y_min,x_max,y_max',
3     'crs': 'EPSG:XYZW' # por default 'EPSG:3857'
4     'size': 'width,height',
5     'format': 'image_format', # por defualt 'png'
6     'interpolation': 'interpolation', # por default 'RSP_NearestNeighbor'
```

```
7       'pixelType': 'pixel_type', # por default 'U32'
8       'f': 'image' # 'pjson' para as tabelas
9   }
```

A classe responsável pela obtenção dos dados, já anteriormente mencionada, tem um método que faz a recolha das imagens das camadas chamado *get_image*. Este começa por criar o pedido dos dados que queremos obter, o pedido é criado de acordo com os parâmetros anteriormente descritos e pode ser usado com todos os serviços da *Copernicus*, os quais correspondem às camadas de alta resolução e aos dados da CLC. Na iteração atual só estamos a usar os dados de cobertura florestal (Figura 4.3), mas o sistema tem a capacidade de obter outros dados. Depois de gerado o pedido é enviado usando a biblioteca do *python*, *urllib3*.

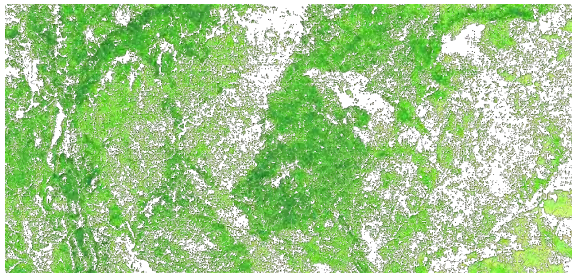


Figura 4.3: Exemplo de uma das possíveis renderizações geradas.

Este método consegue obter as imagens de todas as camadas de alta resolução, incluindo a camada de cobertura florestal, que é a que nos interessa e os dados do CLC, no entanto a COS não tem um serviço na *ArcGIS* e por isso os seus dados não podem ser obtidos usando este método. Assim, durante a fase de desenvolvimento foram realizados um conjunto de testes com uma API da COS, no entanto como iremos discutir na fase do processamento dos dados, os dados fornecidos pela API da COS tinham imenso ruído o que dificultava imenso o processo de isolamento e a compilação das camadas do ambiente.

Assim os dados da COS têm de ser processados localmente, para tal usamos o *shapefile* da COS que é processado usando uma das classes já mencionadas, o *ShapefileProcessor*. Esta tem um método que, dado um conjunto de limites, uma especificação das camadas que querem ser representadas e a referência espacial que está a ser usada vai gerar um conjunto de renderizações para todas as camadas que queremos representar no ambiente.

É importante notar que a representação das camadas é feita através de um dicionário, onde o nome de cada camada está associado às classes da COS que a compõem (Listagem 4.2).

Listagem 4.2: Estrutura do dicionário usado para codificar o mapeamento das tabelas.

```
1   {
2       'layer_1' : ['class_1', 'class_2'],
3       'layer_2' : ['class_3'],
```

```
4     'layer_3' : ['class_4', 'class_3'] # Duas camadas distintas podem usar a
          mesma classe
5 }
```

Para realizar esta tarefa foi usada uma biblioteca do *python* chamada *GeoPandas* que é maioritariamente usada para processamento de dados georreferenciados. Com isto, o processo de recorte começa com a definição um polígono de recorte definido segundo os limites da área que estamos a tentar representar.

De seguida, para cada camada do ambiente é feita uma filtragem das classes da COS que esta usa, e usando o *GeoPandas* é gerada uma renderização da área em que os dois polígonos se intersectam, o previamente mencionado e a *shapefile* da COS (Figura 4.4).

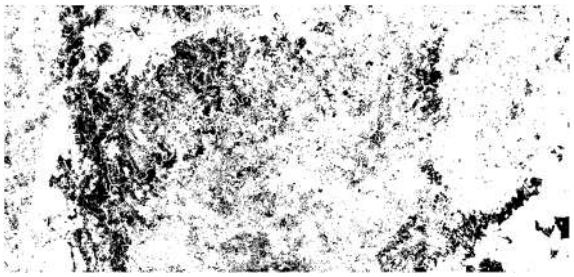


Figura 4.4: Exemplo de uma das possíveis renderizações geradas.

Ainda dentro deste tópico é importante mencionar o processo de obtenção das legendas das camadas de alta resolução. Os serviços rest da *ArcGIS* que estão a ser usados para obter as imagens destas camadas também têm uma operação para o método, este, semelhante ao que foi descrito anteriormente, começa com a construção do pedido, o qual recebe apenas como parâmetro o formato desejado para os dados da tabela. De seguida, o pedido é enviado usando a *urllib3*.

Listagem 4.3: Dados da tabela obtida sobre o formato de json.

```
1  {
2   "layers": [
3    {
4     "layerId": 0,
5     "layerName": "GioLandPublic/HRL_TreeCoverDensity_2018",
6     "layerType": "Raster Layer",
7     "minScale": 0,
8     "maxScale": 0,
9     "legendType": "Unique Values",
10    "legend": [
11     {
12      "label": "0: All non-tree and non-forest areas",
13      "url": "c6bf262107fc8adc332494d2208f0d21",
14      "imageData": "iVBORw0KGgoA (...)", # imagem codificada em base64
15      "contentType": "image/png",
16      "height": 20,
17      "width": 20,
18      "values": [
```

```
19          "0: All non-tree and non-forest areas"
20        ]
21      },
22      # (...)
23  }
```

Para terminar é importante mencionar que apesar de o sistema ter a capacidade de usar tanto dados da COS como dados da CLC, na implementação final optámos por apenas usar a COS por três motivos. Em primeiro lugar os dados da COS têm uma resolução espacial mais elevada que os da CLC, e consequentemente são mais detalhados e precisos. Em segundo lugar a COS tem mais classes disponíveis que a CLC, permitindo representar ambientes mais variados. Por fim, durante a fase de preparação da tese uma grande parte dos testes realizados foram feitos usando dados da COS.

4.2.2 Pré-processamento dos dados

O objetivo da fase de pré-processamento é pegar nos dados obtidos no passo anterior e gerar os mapas de densidade das camadas que queremos representar no ambiente. Para começar, é importante relembrar os dados que temos neste ponto da geração, que são:

- **Dados da camada de cobertura florestal**: Estes dados são usados para informar a o sistema da distribuição geral da vegetação no ambiente;
- **Máscaras das camadas que queremos representar no ambiente**: Estes dados são obtidos a partir da COS e servem para delimitar as zonas que queremos representar no ambiente;
- **A legenda da camada de cobertura florestal**: Estes dados vão ser usados no processamento das imagens.

Com isto, é importante começar pelo processamento dos dados da cobertura florestal, ou seja, a renderização da zona que se quer representar no ambiente, a qual vem sobre o formato de um imagem colorida com cem (100) classificações possíveis onde cada pixel indica a percentagem de copas que podemos encontrar num determinado ponto do ambiente. Assim, o primeiro passo é converter esta imagem para uma em tons de cinzento, de modo a ter um mapa de densidade global da vegetação completa da zona que estamos a tentar representar.

O primeiro passo para gerar o mapa de densidade é obter a rampa de cores que está a codificar a imagem, podendo ser obtida usando um dos dados já mencionados, a legenda da camada de cobertura florestal. Como foi visto anteriormente, estes dados vêm sobre o formato de um ficheiro *JSON*, onde cada classe da legenda tem uma cor associada, armazenada sobre o formato de uma imagem codificada em *base64*.

De modo a obter a cor exata de cada classe da legenda é preciso fazer uma descodificação da imagem e uma amostragem da cor dominante de cada imagem descodificada. Para

obter a cor dominante é feita uma contagem de todas as cores da imagem e é selecionada a que tem mais ocorrências. Estas cores são posteriormente compiladas numa lista, onde a cor de cada classe é indexada por ordem crescente de acordo com a percentagem de vegetação que representam.

Como uma forma de melhorar o processo de obtenção de dados futuros, o sistema faz uma pequena otimização onde a rampa de cores é guardada sobre o formato de uma imagem (Figura 4.5) que pode ser carregada no futuro, em vez de ter de se repetir o processo de geração.

Figura 4.5: Rampa de cores gerada para evitar que haja necessidade de repetir o processo de geração.

Assim, assumindo que já temos a rampa de cores e a renderização da camada de cobertura florestal, o próximo passo é gerar o mapa de densidade em tons de cinzento que cobre a paisagem que queremos representar. De modo a gerar este mapa é executado um processo onde a imagem é mapeada de acordo com o índice da rampa ao qual se aproxima mais, dividindo pelo tamanho da rampa, gerando assim um mapa em tons de cinzento (Listagem 4.4 e Figura 4.6).

Listagem 4.4: Método responsável pelo processamento dos dados da vegetação.

```
1 def unmap_nearest(img, rgb):
2     d = np.sum(np.abs(img[np.newaxis, ...] - rgb[:, np.newaxis, np.newaxis,
          :]), axis=-1)
3     i = np.argmin(d, axis=0)
4     return i / (rgb.shape[0] - 1)
```

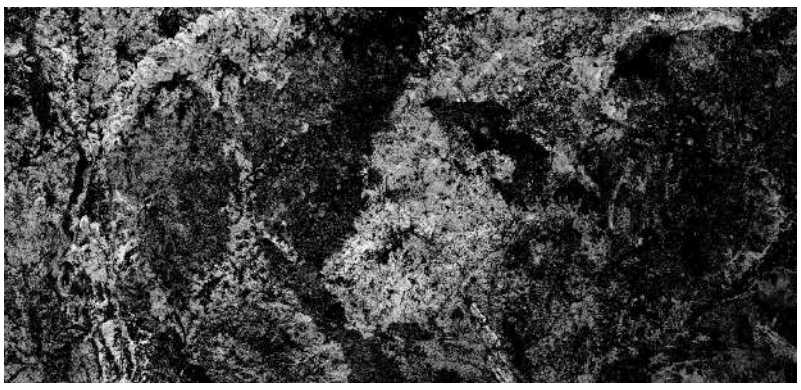


Figura 4.6: Imagem gerada depois de ser aplicada a transformação da rampa de cores.

Após o processamento dos dados de cobertura florestal, o sistema tem de gerar os mapas de densidade para todas as camadas que queremos representar no ambiente. É importante relembrar que na secção anterior (4.2.1) vimos que a partir do *shapefile* das COS conseguíamos obter as máscaras de cada camada que queremos representar no ambiente,

as quais podem ser usadas de forma a obter os mapas de densidade em tons de cinza de cada camada do ambiente. É importante notar que dada a forma como *GeoPandas* faz a renderização de imagens, as máscaras geradas têm de sofrer uma inversão dos valores, porque as zonas que queremos representar estão a ser representadas por pixeis pretos enquanto que as restantes são pixeis brancos.

Assim, depois de se obter o inverso de todas as máscaras, para cada camada que queremos representar, usando a representação mencionada na secção anterior, faz-se uma verificação se esta tem ou não vegetação. Caso tenha é realizada uma multiplicação pixel a pixel com o mapa de densidade gerado anteriormente, caso contrário é só considerada a máscara binária sem qualquer processamento. No final deste processo terminamos com um mapa de densidade para as camadas que queremos representar (Figura 4.7).

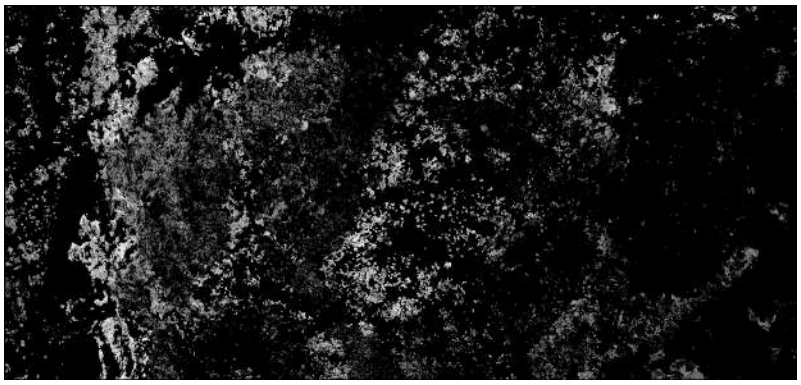


Figura 4.7: Mapa de densidade final de uma camada.

É importante mencionar que o sistema atualmente permite que duas camadas distintas usem a mesma classe da COS, isto causa um problema no que toca o importe dos pesos da vegetação, porque, como iremos ver mais à frente, ao passar os dados para o *unreal* faz com que as zonas da paisagem, onde as duas camadas se intersectam, tenham o dobro da vegetação. A solução passa por adicionar um novo passo à geração, onde são criadas um conjunto de máscaras que ao serem multiplicadas pelos mapas de densidade das camadas com vegetação, reduzem o peso das classes que ocorrem mais que uma vez no ambiente.

A geração destas máscaras começa por percorrer todas as camadas que estão a ser usadas no ambiente e verificar quais são as classes que estas têm em comum, as quais são colocadas numa lista. De seguida, é usada a classe de processamento de *shapefiles* descrita na secção anterior para gerar um conjunto de máscaras individuais para todas as classes que se repetem. As renderizações geradas vão ter as classes que se repetem representadas por pixeis pretos, com valor de zero (0), e os restantes pixeis a branco, com um valor a um (1). De seguida é somada a cada máscara a fração do número de vezes que a classe dela repete no ambiente, ou seja um (1) sobre o número de vezes que a classe repete, os valores são ajustados de forma a garantir que não há pixeis com um valor superior a um (1) (Figura 4.8). Como já foi mencionado, estas máscaras são depois multiplicadas pelos mapas de densidade de todas as camadas com vegetação.

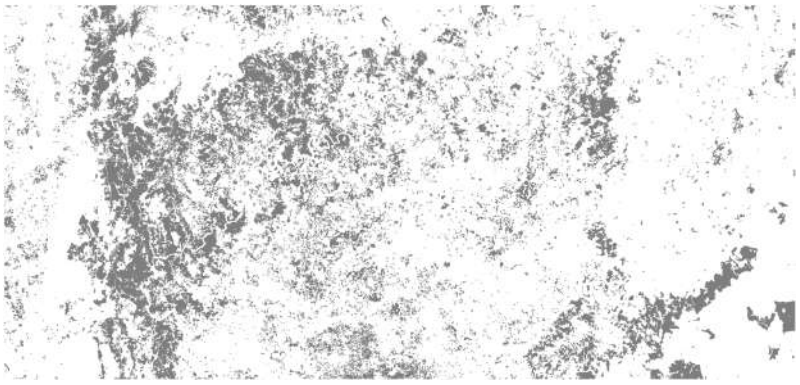


Figura 4.8: Máscara usada para evitar que haja uma sobreposição de cores.

Ainda dentro do tópico do processamento dos dados é importante mencionar que apesar de estar apenas a usar os dados da COS localmente, o sistema tem um conjunto de métodos que suportam o processamento de dados da CLC, e do serviço remoto da COS. Como foi mencionado no capítulo anterior, optámos por utilizar a COS por oposição ao CLC, dito isto, o processamento de dados obtidos a partir dos serviços remotos passa por tentar gerar as máscaras binárias descritas na secção anterior.

O processo de obtenção destas máscaras começa com um processamento da legenda da CLC. Semelhante à camada de cobertura florestal, a cor de cada camada vem sobre o formato de uma imagem codificada sobre o formato *base64*, e de modo a obter a cor específica repete-se o processo de amostragem descrito anteriormente. No entanto, as cores obtidas em vez de serem guardadas numa imagem são guardadas num dicionário, que depois é guardado localmente.

Assumindo que já obtemos a renderização da CLC usando o sistema de obtenção de dados, o processo de geração das máscaras é feito isolando da renderização da carta de ocupação do solo, as cores de cada classe que compõem as camadas que queremos representar no ambiente. Este processo gera um conjunto de máscaras semelhantes às que obtemos na secção anterior, uma por cada camada que queremos usar.

Este processo só pode ser feito para os dados obtidos usando o serviço remoto do CLC, e não é possível estender para o serviço da COS, porque não há nenhuma operação para obter os dados da legenda da COS, havendo só operações para consultar as classes que se encontram em cada pixel da camada. No entanto, consultar a classe de cada pixel da imagem é um processo demorado e não é viável. Uma possível solução passaria por identificar todas as cores únicas da renderização e a partir daí consultar os pixeis de cada classe. Apesar de ser viável, esta solução não funcionava porque as imagens obtidas através do serviço têm imenso ruído e geravam imensas cores únicas, mesmo usando um sistema de *thresholding*. Ainda se tentou arranjar uma forma de reduzir o ruído dos dados usando filtros de mediana, e técnicas de reamostragem de modo a reduzir o número de cores da imagem. No entanto estas técnicas acabaram por ser inúteis tendo em conta que os resultados eram bastante inferiores aos obtidos a partir dos dados locais.

Para terminar é importante relembrar os dados que temos no final da fase de processamento, os quais correspondem aos mapas de densidade de todas as camadas que queremos representar no ambiente.

4.2.3 Automatização da importação dos dados

Assim que os mapas de densidade de todas as camadas estejam gerados é preciso passá-los para o *unreal*. Este processo pode ser feito utilizando a API de *python* para o *unreal*, no entanto, de modo a colocar o motor de jogo a funcionar, com os sistemas já desenvolvidos, é preciso informa-lo do caminho para as bibliotecas externas de *python* que estão a ser usadas.

Como iremos ver, mais à frente, na secção 4.6, o processo de atribuição destes caminhos pode ser ligeiramente automatizado, mas para o interesse deste capítulo eles têm de ser definidos no menu do *plugin* do *python*. (Figura 4.9)

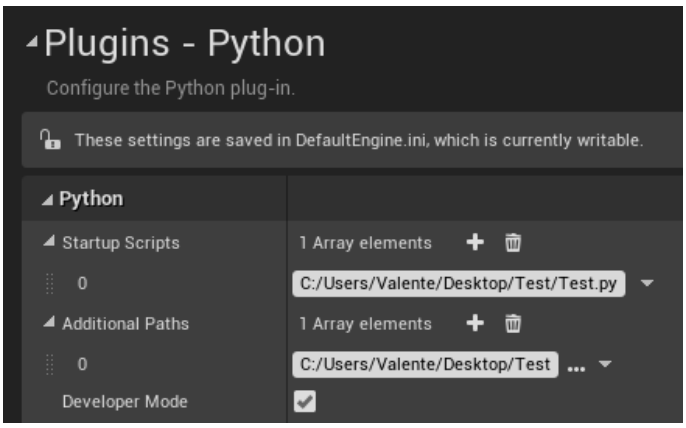


Figura 4.9: Menu de dependências do *plugin* de *python*.

É importante notar que a versão do *python* usada pelo *unreal*, depende da versão do motor de jogos que está a ser usada. Há formas de mudar o interpretador que está ser usado pelo *ue4*, no entanto, isto implica compilar a *engine* de raiz, que para além de ser um processo complicado faz com que o sistema fique dependente da versão do motor de jogos que tinha localmente, eliminando qualquer possibilidade de criar um *plugin*. Assim, simplesmente foi fixada uma versão do unreal que ia ser usada durante o desenvolvimento, esta é a 4.26.2, que está a usar a versão do python 3.7.7.

Ter uma versão do python pré-estabelecida facilitou imenso o processo de garantir que as bibliotecas que estavam a ser usadas, fora do motor de jogos, também funcionavam. No entanto, algumas bibliotecas foram mais difíceis de colocar a funcionar dentro do *unreal*, mais precisamente o *GeoPandas*, isto porque esta biblioteca tem imensas dependências externas que funcionam todas com versões distintas.

No momento em que o sistema está a funcionar dentro do *unreal* o próximo passo passa por importar todas as camadas previamente mencionadas para dentro do motor de jogos. Para cumprir esta tarefa, o sistema tem uma biblioteca de funções que ajuda o

processamento de dados e importe do lado do *unreal*, chamada *UnrealImporter*. Esta possui um método chamado *import_texture* que recebe como parâmetros de input o caminho do ficheiro que queremos importar e o caminho dentro da *ue4* para onde queremos enviar a textura. O método usa uma classe da biblioteca do *unreal* para realizar o importe dos dados e configurar os parâmetros iniciais da textura. Dentro deste tópico é importante mencionar que as imagens são importadas como sendo texturas em tons de cinzento, sem nenhuma forma de interpolação interna, porque como iremos ver mais à frente não é necessário que estas sejam interpoladas.

4.3 Menu de interação com o utilizador

Um ponto importante do desenvolvimento passa por desenvolver uma interface de interação com o sistema de geração, considerando que este menu permite testar várias configurações de uma forma rápida. Assim, o sistema tem de ter uma forma para conseguir comunicar com o interpretador do *python*, e enviar e receber dados do interpretador.

De modo a executar *scritps* de *python* a partir do sistema de *blueprints* é necessário usar um nó específico que executa comandos de *python* no editor (Figura 4.10). Este nó pode ser usado para executar programas de *python* de forma dinâmica, dependendo da configuração fornecida.

Figura 4.10: *Blueprint* usada para chamar scripts de *python* a partir do editor.

Sabendo isto, é importante relembrar quais são os principais objetivos desta interface:

- Primeiro, o menu de interação tem de conseguir informar o sistema de geração das camadas que estão a ser geradas, dos dados que estão a ser usados e dos limites que estão a ser representados no ambiente;
- Segundo, é importante que o sistema tenha uma parametrização predefinida para utilizadores que não queiram criar situações de representação específicas.

Para começar, para esta componente do projeto, foi usado o sistema de *editor widget*'s do *unreal*, permitindo a criação de menus de utilização dentro do editor da *ue4* usando o sistema de *widgets*, que facilita a criação de interfaces usando um menu de criação onde a lógica do sistema é controlada através de *blueprint*'s.

A versão final do sistema tem a capacidade de gerar uma listagem das camadas que queremos representar, e associar a cada camada as classes da COS que a representam. São

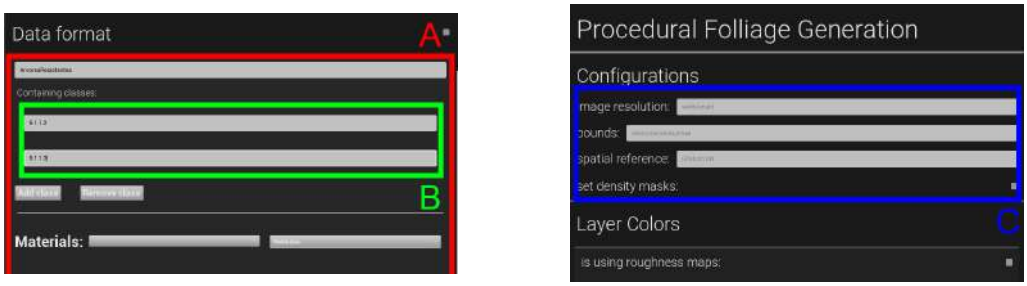


Figura 4.11: *Blueprint* usada para chamar scripts de *python* a partir do editor.

utilizadas duas *listview*'s dentro da *widget*, uma para as camadas queremos representar (Figura 4.11 - A) e uma dentro de cada camada para as classes as classes de cada camada (Figura 4.11 - B). À medida que as listas são geradas, os dados de cada camada vão sendo guardados num *DataAsset* [39], chamado *PDAImportData*, que os guarda num dicionário onde as chaves são o nome das camadas que queremos representar no ambiente, enquanto que os valores correspondem às listas das classes que constituem cada camada. Iremos ver mais à frente o processo de seleção da vegetação e dos materiais de cada camada.

Estes dados têm de ser passados do *unreal* para o *python* usando o nó anteriormente mencionado, no entanto, esta função recebe como input o *script* de *python* que queremos executar sobre o formato de uma *string*, e nem todos os valores das *blueprint*'s têm um método *ToString* que os converta para um valor aceite pelo *python*.

Deste modo, foi desenvolvido um conjunto de funções em *blueprint*'s responsáveis por converter dados em formatos que possam ser lidos pelo interpretador de *python*. As funções mais relevantes para a conversão dos dados das camadas são as seguintes:

- Uma função responsável pela geração de dicionários do *python*, esta recebe duas listas de *strings*, uma para os nomes de cada classe e outra para os valores associados à primeira lista;
- Uma função responsável pela tradução de listas de *blueprint*'s para o *unreal*, esta função recebe uma lista de valores e converte-os para uma lista do *python*.

Assim, estas funções são usadas para converter os dados do *PDAImportData* para o python.

Para estabelecer os limites da representação, e a referência espacial que os define, a interface tem um conjunto de caixas de diálogo (Figura 4.11 - C). Os dados são passados diretamente para os *scripts* do lado do *python*, visto que estes dados estão definidos como *strings*.

Para terminar, é também importante mencionar que a geração tem uma configuração pré-definida onde os dados são estabelecidos usando um ficheiro *.ini*, o qual detém uma configuração para uma compilação de camadas, e para um conjunto de limites. Caso não seja fornecido nenhum dado da interface gráfica, o sistema usa os dados deste ficheiro de configuração (Listagem 4.5). Também é importante notar que este ficheiro de configuração

é utilizado para armazenar valores específicos da geração, de modo a evitar que tenham de ser calculados várias vezes, como por exemplo a resolução da *landscape* em *tiles*.

Listagem 4.5: Configuração pré-definida da geração.

```
[DOWNLOAD]
bound_x_min = -946914.4277606456
bound_y_min = 4821995.293980075
bound_x_max = -811315.1395826909
bound_y_max = 4886202.397739647

img_size_x = 1280
img_size_y = 606

[FILE_NAMES]
table_name = test_table_1.txt
image_path = test_image_1.png
image_save_path = test_image_2.jpg

temp_folder = C:\\TempX

[LAND_COVER]
nb_of_layers = 1

layer_0_name = arvores_resistentes
nb_of_merged_layers_0 = 2
layer_id_00 = 5.1.1.4
layer_id_01 = 5.1.1.3

[MATERIAL]
material_name = default

[SYSTEM_PATHS]
nb_of_paths = 1
system_path_1 = C:\\Users\\olive\\Anaconda3\\envs\\TESE2020_21\\Lib\\site-
    packages
```

4.4 Automatização do processo de geração

A próxima fase do algoritmo tem a ver com a automatização do processo de geração, automatizando o processo de geração das ferramentas do *unreal* previamente mencionadas, de forma a gerar uma representação da vegetação que se guie pelos dados que foram processados anteriormente.

Antes de avançar, é importante relembrar os dados disponíveis neste passo do sistema, que são os mapas de densidade de cada camada que queremos representar no ambiente, e a representação da forma como as camadas estão organizadas.

Assim, é preciso usar os dados obtidos durante a primeira fase da geração para informar o sistema de geração utilizando o sistema de *landscape*'s da *ue4*, onde é possível definir várias camadas de representação, incluindo várias camadas de vegetação. Depois de definidas, é preciso fornecer ao sistema os pesos dos mapas de densidade de cada camada que estamos a representar no ambiente. Com estes dados definidos, o próximo passo é chamar as operações responsáveis pela geração da vegetação do ambiente.

Com isto, nesta secção vamos ver todos os processos da geração que foram automatizados de modo a gerar a vegetação guiada de um determinado local. Vamos começar por falar sobre a forma como as camadas do ambiente são definidas usando o material da *landscape*, de seguida, vamos explicar o processo de criação e atribuição dos dados de cada camada, e para terminar vamos fazer uma análise do processo de atribuição dos pesos de cada camada que queremos representar no ambiente.

4.4.1 Automatizar a criação de materiais

O primeiro passo da automatização da geração passa por automatizar o processo de definição das camadas do ambiente, este processo é feito usando o sistema de *landscape* do *unreal*. A definição das camadas, que queremos representar, é feita usando o material do terreno da paisagem. Assim, de forma a conseguirmos informar os sistemas do *unreal* das camadas que estão a ser representadas é preciso mudar ou modificar o material da *landscape*.

Para começar, é importante distinguir o processo de modificação do material existente do processo geração de um novo material, o primeiro é um processo mais simples onde não é preciso mudar os materiais da *landscape*, por contraste o segundo é um processo mais complexo onde temos de gerar um novo material e de seguida atribuí-lo a todas as camadas da *landscape*.

Na iteração atual do sistema começamos por assumir que o terreno não tem nenhum material já atribuído, assim o sistema cria sempre um material novo e faz a sua atribuição ao terreno que está a ser usado na geração.

Com isto, o desenvolvimento do processo de geração foi dividido em duas fases, a primeira passou por planear o material gerado pelo sistema e a segunda correspondeu à automatização do processo de criação do material. Apesar da primeira corresponder a uma fase de planeamento provou-se ser extremamente útil para fazer uma análise das várias opções de geração que podiam ser usadas na *landscape*.

Dado isto, os materiais do *unreal* funcionam usando um sistema de programação visual baseado em nós, onde cada nó é chamado expressão. Cada expressão recebe um conjunto de input's e produz um output, funcionando de forma semelhante ao sistema de *scripting* do *unreal*, as *blueprint*'s. O material, no final, tem uma expressão especial chamada *Main Material Node*, que é responsável por determinar a cor final do material, e outras configurações do material (Figura 4.12).

Para definir as camadas do ambiente é preciso usar um conjunto de expressões que

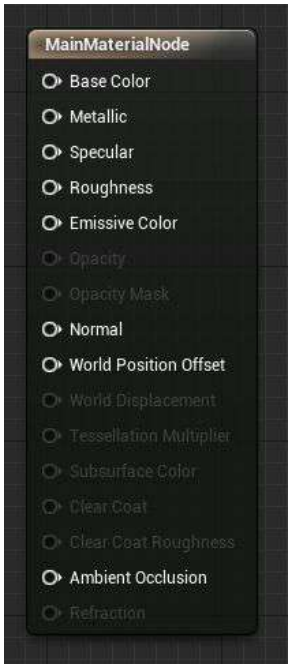


Figura 4.12: *Main Material Node*.

se encontram dentro do módulo da *landscape* do *unreal*, as quais permitem que sejam definidas as várias camadas do ambiente e a forma como elas interagem visualmente entre si.

Para definir as camadas temos um conjunto de expressões que recebem como *input* o nome das camadas que queremos representar. Algumas das expressões mais relevantes são as seguintes:

- A *LayerBlend*, esta expressão permite representar várias camadas de uma só vez e definir a forma como elas interagem visualmente entre si;
- A *LayerSample*, permite fazer uma amostragem dos pesos do terreno da paisagem usando valores entre zero (0) e um (1).

Ambas as expressões são usadas para representar as camadas da paisagem, e informam o sistema de *landscape*'s das camadas que podemos encontrar no ambiente.

Algumas expressões têm a capacidade de ser parametrizadas para que possam ser criadas instâncias distintas com valores diferentes, por exemplo ter dois materiais idênticos mas com cores e texturas diferentes. No entanto, nenhuma das expressões previamente mencionadas com a capacidade de definir as camadas do terreno têm a capacidade de ser parametrizadas, de tal maneira que possam ser geradas instâncias do material com camadas diferentes. Por este motivo, é sempre necessário modificar ou criar um novo material para *landscape*, em vez de ter um material já gerado do qual podemos gerar instâncias com parâmetros distintos.

Como iremos ver mais à frente, os materiais ao serem gerados de forma automática também permite que haja um maior controlo por parte do utilizador, permitindo gerar

materiais mais customizáveis que se adaptem melhor à representação que o utilizador quer gerar, e ao mesmo tempo permite que sejam feitas otimizações do material de forma dinâmica, à medida que é gerado.

No projeto a que estamos a dar continuação, o terreno do cenário principal já tinha um material associado, assim, o primeiro passo do desenvolvimento desta componente foi criar um sistema que conseguisse automatizar o processo da geração de materiais equivalentes ao do projeto, e que ao mesmo tempo fosse suficientemente generalizado de modo a poder gerar outros materiais.

Para cumprir esta tarefa foi desenvolvido um sistema em *python* usando a API do *unreal*. Esta é composta por uma classe responsável por criar e processar novos materiais da *ue4* e uma biblioteca de funções que dado um material geram um conjunto de combinações de expressões que representam parte do material gerado.

A classe principal chama-se *MaterialCreator* e foi construída por cima do sistema de criação de materiais da API de *python* sendo responsável pela geração e edição do material. Esta classe tem métodos que não só permitem adicionar e remover expressões do material, como também permitem conectar várias expressões, e a modificação dos valores de cada nó. Esta classe também tem a capacidade de gerar instâncias de materiais e modificar os seus valores.

A biblioteca de geração tem um conjunto de funções que podem ser usadas na geração do material, elas recebem com *input* um objeto *MaterialCreator* e a partir daí geram um conjunto de expressões que representam uma certa parte do material. Assim, de modo a gerar um material ele tem de ser decomposto nas várias componentes que o constituem e para cada componente é escrita uma função responsável por gerar essa parte do sistema.

Para o material da paisagem do projeto, ao qual estamos a dar continuação é possível identificar quatro componentes principais:

- **A separação das várias de camadas que estão a ser representadas no ambiente**(Figura 4.13): Esta componente define as várias camadas que estão a ser usadas no processo de geração. As camadas que são geradas são definidas através de uma lista que contém as camadas que queremos representar;

- **A configuração do camada de vegetação rasteira**(Figura 4.32): Esta componente serve para definir as zonas onde podemos encontrar vegetação rasteira no mapa, ela está dependente do número de camadas que queremos representar no ambiente, e consequentemente tem de ser gerada de forma dinâmica;

- **A definição das coordenadas escaladas**(Figura 4.15): Esta componente usa duas expressões das *landscapeCoords* para definir os *UV*'s da *landacape* escalados. A função que define esta componente foi estabelecida para que seja o programador a definir a escala que quer e este valor é usado como as coordenadas de UV do terreno na generalidade, que são depois usadas para renderizar a ortofoto do terreno;

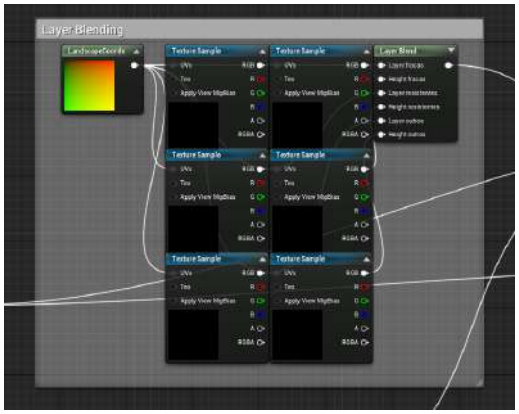


Figura 4.13: Mistura das camadas.

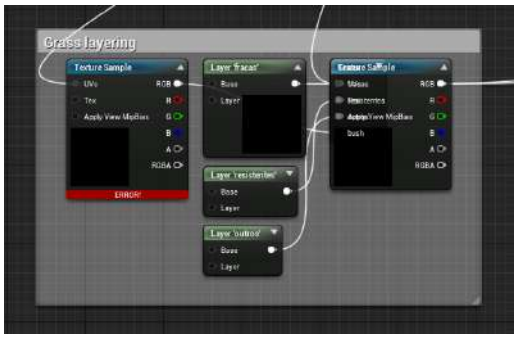


Figura 4.14: Configuração da relva.

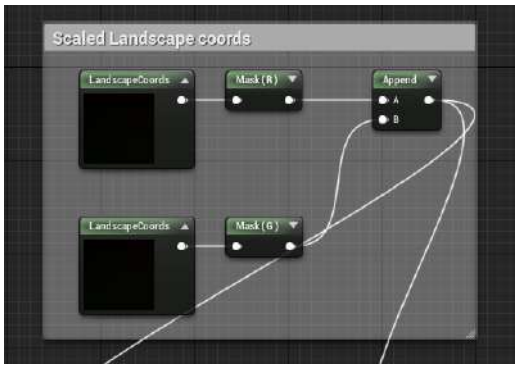


Figura 4.15: Coordenadas escaladas.

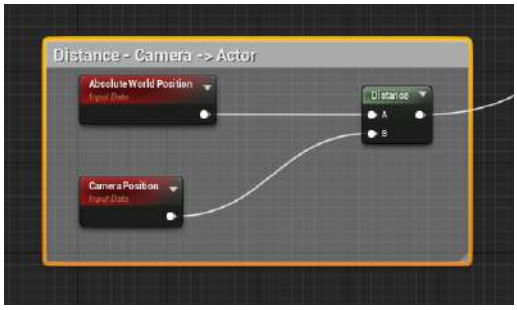


Figura 4.16: Distância ator câmara.

- **Distância entre a câmara e o ator da *landscape***(Figura 4.16): Esta componente cria um conjunto de expressões que definem a distância entre a câmara e o ator da *landscape*, que é posteriormente usada para fazer quebrar o efeito de repetição do material fazendo uma mistura entre o terreno *tiled* e a ortofoto da zona representada.

Assim, foi desenvolvido uma função para cada uma destas componentes do material que constituem a geração do material. Um problema com este material é que este não estava a usar todas as capacidades do sistema de materiais do *unreal* e não conseguia estender-se a uma representação de várias camadas, assim, dado a simplicidade deste material ele está a ser utilizado atualmente pelo sistema, apesar de este o suportar. Assim na secção 4.8 vamos ver o material alternativo que foi gerado para o projeto.

4.4.2 Automatizar a criação dos dados das camadas

Sabendo que existe uma forma de gerar o material da *landscape* e de definir as camadas que queremos representar, a segunda parte do processo de automatização passa por gerar todos os dados que precisamos para gerar a vegetação de um determinado ambiente. De modo a sabermos os dados que têm de ser gerados é preciso relembrar o processo de geração.

Para gerar a vegetação de um determinado local é preciso gerar os dados das camadas que queremos representar, isto passa por gerar as *LayerInfoObject*'s de todas as camadas, as malhas estáticas da vegetação que queremos representar no ambiente e o *ProceduralFoliageSpawner* responsável por ditar os parâmetros de geração da vegetação.

Começando pela geração dos *LayerInfoObject*'s, estes objetos ditam a forma como as camadas interagem entre si caso haja uma interseção entre camadas. Há duas formas de estabelecer a forma como as camadas interagem entre si. Na primeira, as camadas são definidas de uma forma subtrativa, ou seja, adicionar um peso a um ponto do terreno, removendo os restantes que lá se encontram. A segunda forma é aditiva, ou seja, quando um valor é adicionado a um ponto do terreno, não é retirado das restantes camadas(Figura 4.17).

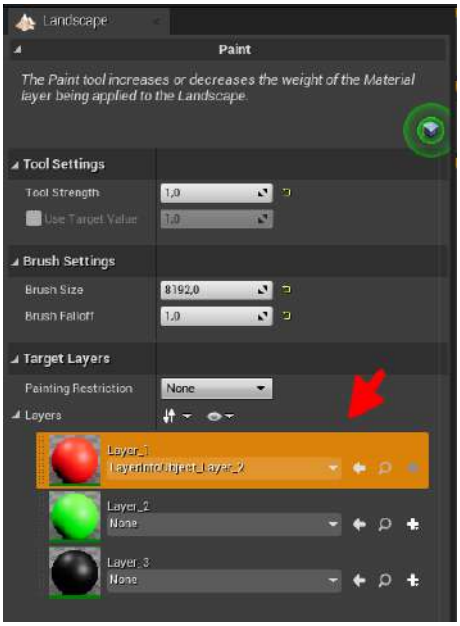


Figura 4.17: Editor do *unreal* com o menu de atribuição dos LayerInfoObjet.

Assim, antes da implementação do sistema de geração foi preciso decidir qual era o processo de interação que queríamos usar neste projeto. Já foi mencionado anteriormente que a maneira como estamos a configurar as camadas permite que duas camadas distintas partilhem uma classe, isto quer dizer que temos casos onde as camadas podem-se sobrepor. Na subsecção 4.2.2 já foi apresentada uma solução para evitar que zonas onde camadas se intersentem fiquem com valores duplicados. A implementação desta assume que os pesos do terreno usam uma abordagem aditiva, ou seja, quando um valor é adicionado a um ponto do terreno este não retira esse valor das outras camadas. Assim o sistema usa LayerInfo's configuradas de forma aditiva.

Não existe nenhuma função nem na API de *python*, nem nas *blueprint*'s nativas da *ue4* que permita a criação de novas *LayerInfoObject*'s. Assim foi desenvolvida uma classe em *C++* chamada *LandscapeHelper* que tem um conjunto de funções que ajudam o processo de geração, uma delas é responsável por criar e atribuir as *LayerInfoObject*'s. Cada um

destes métodos está acessível nas blueprint's e consequentemente também está acessível na API do *python*.

De modo a que estes dados sejam gerados corretamente eles têm de ser gerados de acordo com os padrões de geração do *unreal*. Deste modo, os dados têm de ser gerados para uma pasta específica do projeto, que se encontra situada dentro da pasta onde podemos encontrar o nível para o qual a vegetação está a ser gerada. Caso esta pasta não se encontre já criada, então tem de ser gerada automaticamente. Caso ela já exista, os seus conteúdos têm de ser apagados, dado que os dados presentes continuam a fornecer informação da composição do terreno ao sistema de geração, mesmo não estando atribuidos, e faz com que haja a possibilidade de haver erros de geração.

De seguida todos os *LayerInfoObject*'s gerados têm de ser gravados automaticamente, porque caso contrário não podem ser atribuídos ao terreno de geração. Assim é usado o módulo *EditorScriptingUtilities* para guardar os dados no final do processo de geração em C++.

Um problema com esta solução é que apesar dos dados serem gerados e atribuídos corretamente, eles ficam apenas em cache, o que quer dizer que quando o motor de jogos é reiniciado os dados deixam de estar atribuídos.

No que toca a geração das malhas estáticas e do *spawner* é feito usando a API do *python*, através do sistema de *factory*'s da *ue4*, que permite a criação de objetos chamados *factory*'s que podem ser subsequentemente usados para gerar dados do editor. Esta é usada por uma das classes do python já mencionadas, a *UnrealImporter*, de forma a gerar os dados necessários, ou seja uma *StaticFoliageMesh* por cada camada com vegetação e um *ProceduralFoliageSpawner* para os dados de vegetação.

Estes dados são gerados para uma pasta pré-definida na raiz do projeto, no entanto, este parâmetro é configurável, e pode ser modificado pelos utilizadores caso eles queiram usar outra pasta para guardar os dados da vegetação.

Não fornecendo nenhuma informação adicional da interface os dados são gerados com os valores pré-definidos, como iremos ver mais à frente estes valores podem ser alterados de acordo com o tipo de vegetação que estamos a tentar representar.

4.4.3 Atribuição e o importe dos pesos das camadas

Agora que todos os dados necessários, para realizar a geração de vegetação, estão gerados o passo seguinte corresponde à atribuição dos pesos do terreno e destes mesmos dados. Assim, é importante relembrar os dados que temos nesta fase do processo de geração, os quais correspondem aos pesos das camadas que estamos a tentar representar no ambiente, os dados da vegetação, o material do terreno que estamos a tentar gerar e as *LayerInfoObject*'s das camadas que queremos representar.

Antes de avançar, é importante mencionar as dependências da atribuição destes valores, isto é, antes de se poder atribuir as *LayerInfo* de cada camada é preciso atribuir o material do terreno da paisagem, dado que este dado é que informa o processo de geração

das camadas que podemos encontrar no ambiente. Por contraste os dados da geração de vegetação, ou seja as malhas estáticas e os *Spawner*'s podem ser atribuídos independentemente dos dados do terreno. Os pesos de cada camada só podem ser atribuídos depois dos *LayerInfoObjet*'s estarem todos atribuídos.

Assim o primeiro passo é atribuir o material da *landscape* usando o *python*, onde é selecionado o ator da *landscape* e usando uma função chamada *set_editor_property* é feita uma mudança do material do terreno. Como iremos ver mais à frente se o terreno for formado por mais de uma tile é preciso fazer um processo ligeiramente diferente. É importante relembrar que este sistema assume a existência de um terreno no ambiente.

De seguida, o próximo passo é atribuir os dados das *LayerInfo* de cada camada. Este processo já foi mencionado brevemente na subsecção anterior, e é feito usando a classe de ajuda desenvolvida em C++ que é responsável por gerar estes objetos. A atribuição destes dados é feita usando os dados do ator da *landscape*, mas tem o problema adicional de que os dados não ficam permanentemente atribuídos e quando o motor de jogos é reiniciado os valores deixam de estar atribuídos.

Assim que os dados de todas as camadas estejam atribuídos o passo seguinte é realizar a importação dos pesos de cada camada. No editor do *unreal* o processo de importação dos pesos pode ser feito selecionando a camada que queremos importar e carregando na opção de importação dos dados, isto vai abrir um menu de seleção a partir do qual podemos selecionar a textura que queremos usar para os dados do terreno. Este processo tem um conjunto de problemas:

- Primeiro, este método não está acessível a partir de nenhuma API do *unreal*, isto impede que ele possa ser usado diretamente;
- Segundo, este método não permite que sejam importados dados de vegetação que tenham um formato diferente do terreno para onde queremos importar estes dados, fazendo com que haja a possibilidade de haver falhas na geração;
- Terceiro, o processo de importação não sofre nenhum pós processamento impedindo a possibilidade de fazer algo interpolação dos dados.

Com isto, o processo de atribuição dos pesos de cada camada do terreno é ligeiramente diferente do que se encontra dentro do motor de jogos e foi descrito anteriormente. Este sistema foi desenvolvido em *python* e recebe uma textura como *input* que é importada para o terreno. Foi utilizado um material e uma textura de renderização, um *RenderTarget* no *unreal*, o processo passa por renderizar o material na textura, esta é gerada dinamicamente com a resolução do terreno do ambiente que queremos representar e que depois é importada diretamente para a *landscape*, usando uma função da API de *python* da *ue4*.

A razão pela qual a textura de renderização tem de ser gerada com o mesmo tamanho que o terreno do cenário é porque usando o sistema de *landscape*'s do *unreal* os pesos das camadas estão associados à malha do terreno.

O material inicialmente usado, consiste num processo de amostragem das texturas que contêm os dados dos pesos das camadas do ambiente. O processo de amostragem é feito usando uma expressão chamada *TextureSampler* que recebe como input os UV's da textura que estamos a tentar importar e a própria textura, os UV's são obtidos usando a coordenadas da textura de renderização. O valor da textura está parametrizado, o que quer dizer que se pode usar o mesmo material base para criar uma instância cujos valores são modificados de acordo com a textura que estamos a tentar importar.

Uma consequência interessante deste processo é que nos dá a possibilidade de se poder fazer um pós-processamento dos dados antes de serem importados para a *landscape*, e como iremos ver mais à frente este processo é usado para facilitar e melhorar o processo de importação dos dados. Outra consequência é que a resolução dos dados, que estão a ser importados, não depende da resolução do terreno (Figura 4.18).

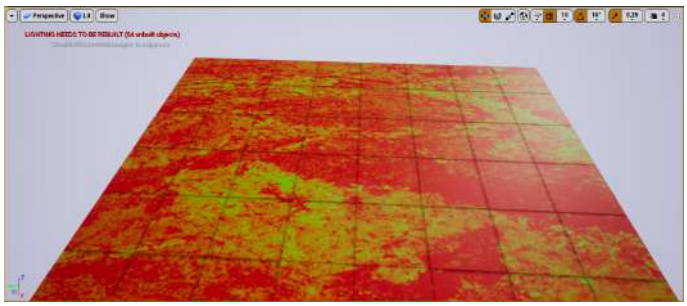


Figura 4.18: Exemplo do resultado da automatização do processo de importação dos dados de vegetação.

A atribuição dos dados de vegetação é feita em dois passos, o primeiro é a atribuição dos dados do *spawner*, ou seja de todas as malhas estáticas das camadas do terreno, e o segundo é a atribuição dos valores dos atores da vegetação no nível.

Assim que os dados da geração estejam todos atribuídos, o sistema está pronto para gerar a vegetação (Figura 4.19), acedendo ao menu do ator de geração do cenário e carregando no botão que diz *resimulate*. Também houve um esforço para tentar automatizar esta parte, no entanto este processo não podia ser feito sem alterar o código fonte da *ue4*, o que dificulta imenso o processo de geração do *plugin* do sistema.

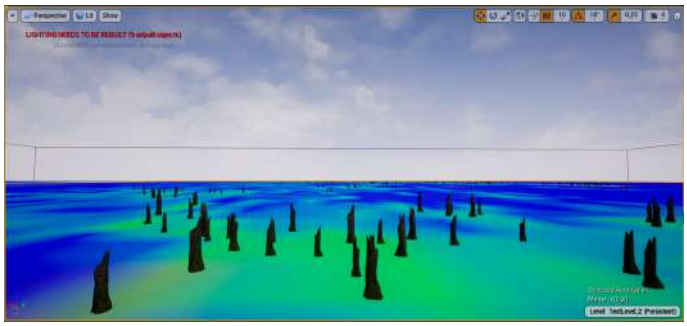


Figura 4.19: Exemplo de um dos primeiros terrenos gerados com vegetação.

4.5 Importação *tile* a *tile*

O processo discutido até agora, só aborda o processo de geração para cenários compostos por apenas um terreno, esta assunção não reflete o sistema mais comum que faz uso do sistema de *world composition* do *unreal*, o qual permite que o mundo seja dividido em *tiles* de modo a melhorar o desempenho em tempo real do projeto. Os projetos aos quais estamos a dar continuidade também usam esta técnica para melhorar o seu desempenho geral, assim é crucial garantir que o sistema funciona com terrenos divididos por várias *tiles* (Figura 4.20).

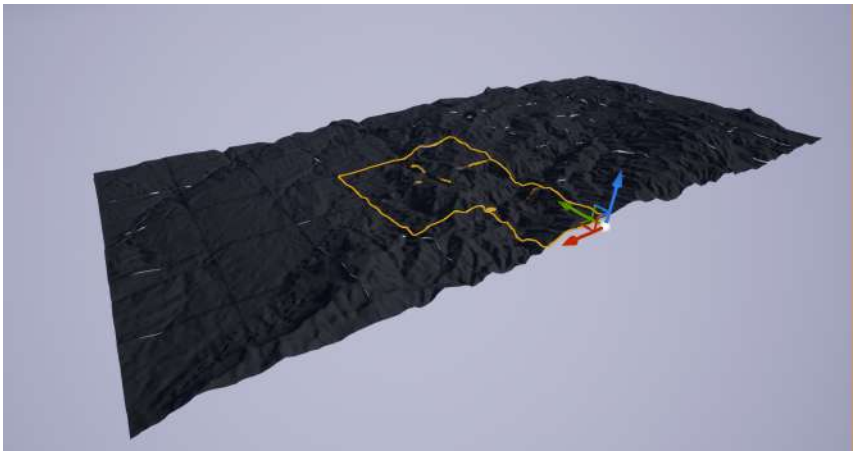


Figura 4.20: Exemplo de um dos terrenos usados para testar a geração de vegetação que está divido em *tiles*.

Antes de avançar, é importante ter uma visão geral do algoritmo atual e identificar os pontos que tiveram de ser modificados de modo a garantir que o sistema funcione adequadamente com mundos divididos em várias *tiles*.

Como foi descrito inicialmente, o sistema pode ser separado em duas fases, a primeira corresponde ao processo de obtenção e processamento dos dados, o qual não precisa de sofrer nenhuma alteração de modo a funcionar corretamente com terrenos divididos em várias *tiles*. Por contraste, a segunda fase do sistema ocorre toda dentro do *unreal*, ou seja, algumas das suas componentes vão ter de ser modificadas de forma a garantir que funcionam corretamente com a nova composição do mundo.

Na segunda fase do processo, o sistema começa por gerar o material e os dados de cada camada que queremos representar no ambiente. Este passo da geração não precisou de sofrer alterações de modo a funcionar corretamente com o sistema de *World Composition* da *ue4*, por contraste a importação dos dados teve de sofrer várias alterações de modo a ficar compatível com o novo sistema.

Para começar, é importante relembrar o processo que estava a ser usado para importar os dados para o ambiente, que começa por atribuir o material do terreno do ambiente, de seguida faz a atribuição dos dados de cada camada e por fim é feito a importação dos pesos da *landscape*.

De modo a adaptar este processo ao sistema de *World Composition* foi necessário arranjar um forma de carregar as *tiles* no editor, felizmente existe um método no *unreal* que nos permite fazer isto. As *tiles* ao serem carregadas instanciam um ator *LandscapeStreaming-Proxy* no cenário, que tem um campo que permite que se faça a atribuição do material da *tile* carregada no editor.

De modo a que o material do terreno seja modificado permanentemente também é preciso continuar a atribuir o mesmo material ao Ator do terreno, como já estava a ser feito anteriormente. Para assegurar que os dados ficam corretamente guardados é preciso guardá-los depois do material ser alterado, isto é feito em C++ usando um Módulo de processamento da *landscape* chamado *EditorScriptingUtilities*.

A implementação inicial passou por carregar todas as *tiles* do terreno ao mesmo tempo, e repetir o processo implementado anteriormente, com a modificação adicional onde a textura criada em vez de ter em conta a resolução do terreno composto apenas por uma *tile*, tem em conta a resolução do terreno de acordo com todas as *tiles* carregadas. Um problema com esta implementação é que como as *tiles* têm de ser carregadas todas ao mesmo tempo, pode fazer com que o sistema tenha problemas de desempenho e resulte mesmo numa falta de memória impedindo a geração de vegetação.

Assim, é preciso implementar uma solução alternativa onde cada *tile* é processada de cada vez, independentemente das outras. Anteriormente foi mencionado que os dados das camadas, mais precisamente as *LayerInfoObject*'s, não eram guardados de forma permanente quando o motor de jogos era reiniciado, ou quando um novo nível é carregado, este problema também se aplica ao carregamento de *tiles* no editor, ou seja sempre que as *tiles* são carregadas os dados das camadas não eram carregados.

De modo a resolver este problema foi necessário modificar a forma como estava a carregar as *tiles*. Este processo foi modificado para que quando as componentes do terreno são carregadas, os dados das camadas são carregados automaticamente, e caso não estejam gerados também são logo gerados.

Outro erro que resulta da mudança da forma como estamos a carregar as *tiles* tem a ver com a importação dos pesos da *landscape*, isto porque quando a função de importação é invocada são importados todos os valores da renderização que foi gerada, ou seja para cada *tile* estão a ser importados os pesos do terreno completo.

De modo a resolver este erro é importante relembrar o método de importação usado na geração, o qual é feito usando um material e uma textura de renderização, assim o material pode ser modificado de modo a que seja apenas selecionada parte da textura que queremos representar. Este processo foi modificado de modo a receber o índice da *tile* que está a ser processada e o número total de *tiles*, e com estes valores as UV's da textura são recortadas de acordo com a zona que estamos a tentar representar, ou seja para *tile* zero (0), zero (0) são apresentados os dados do canto inferior esquerdo da textura e para a *tile* um (1), zero (0) são apresentados os valores à direita da *tile* anterior.

$$\frac{UV + POS}{RES} = TARGET\_UV$$

Se considerarmos que a variável UV corresponde ao mapeamento das UV's da textura com valores de zero (0) a um (1), a variável POS Corresponde à posição da *tile* carregada, que começa a zero (0) e termina em RES que corresponde à resolução do terreno em número de *tiles*, a fórmula anterior descreve o processo de cálculo das UV's finais, as $TARGET\_UV$.

É importante mencionar que o processo de importação assume que o terreno foi gerado corretamente, que não contém artefactos e que as *tiles* do terreno seguem o padrão pré-definido para nomear cada *tile* do ambiente. Assim se houver *tiles* não retangulares, ou rodadas há uma possibilidade de o processo de importação não ser realizado corretamente. Por norma, isto nunca acontece se o utilizador estiver a usar as configurações normais do motor de jogos.

Assim com estas modificações, o sistema consegue fazer a importação e a atribuição dos dados *tile* a *tile*, carregando uma *tile* do ambiente, fazendo a atribuição do material da *LandscapeProxy*, atribuindo os dados das camadas, importando os pesos da *tile* carregada e por fim descarregando a *tile* e gravando as alterações feitas, este processo repete-se para todas as *tiles* do ambiente. No final deste processo o resultado é idêntico ao que encontrávamos no sistema anterior mas para um terreno composto usando o sistema de *World Composition*.

4.6 *Plugin* do sistema

Um dos pontos mais importantes deste sistema é que o processo de passagem de projeto para projeto seja facilitado, podendo ser feito usando um *plguin* do *unreal*. Estes facilitam o isolamento e a passagem de sistemas ou dados entre projetos.

Para criar a pasta do *plugin* é preciso ir ao menus dos *plugins* e selecionar a opção para criar um novo *plugin* e escolher o tipo de *plugin* que queremos gerar. Existem vários tipos de *plugins*, no entanto, para este projeto o que nos interessa é regular [40]. Este cria uma nova pasta no projeto que é onde temos de colocar todo o conteúdo do sistema que estamos a tentar exportar.

De modo a gerar um *plugin* no *unreal*, um dos primeiros passos é identificar todas as dependências externas que existem no sistema que se quer exportado, isto é dependências de bibliotecas externas e de outros *plugins* que estejam a ser usados no sistema, e no caso do *python* também é necessário exportar as bibliotecas que estão a ser usadas no projeto.

As dependências das bibliotecas de python no unreal têm de ser estabelecidas manualmente, ou seja, no menu de configuração do python o utilizador tem de indicar onde é que as bibliotecas necessárias se encontram localizadas. Este fator dificulta a criação do plugin porque não é possível simplesmente isolar estas bibliotecas.

Para resolver este problema temos duas soluções. A primeira solução passa por passar todas as bibliotecas do projeto, para dentro do projeto, isto resolve o problema mas faz como que o plugin fique muito pesado, o que dificultaria o processo de passagem. A segunda solução passa por exportar o ambiente de *python* que está a ser usado no sistema e arranjar uma forma de automatizar o processo de importe.

Inicialmente optámos por tentar usar a primeira solução, no entanto colocar esta solução a funcionar corretamente provou ser um processo muito mais difícil do que o que tinha sido planeado, e o importe das bibliotecas tinha de ser feito em vários passos, o que derrotou o propósito de ter um *plugin*. Assim, optou-se por usar a segunda opção, onde em vez de o projeto ter as bibliotecas diretamente, tem uma versão comprimida do ambiente de *python* que está a ser usado, e uma nova entrada no ficheiro de configuração com os caminhos absolutos para o sítio onde o ambiente está instalado, e usando estes caminhos os *path*'s do ambiente são logo definidos automaticamente no início da geração.

4.7 Algoritmo de *up-scaling*

Um problema detetado ao longo do processo de desenvolvimento tem a ver com a importação dos pesos do terreno. Ao serem importados geravam um efeito de mosaico que divide as áreas que queremos representar com saltos de valores em vez de uma transição suave entre os vários níveis que queremos representar (Figura 4.21).

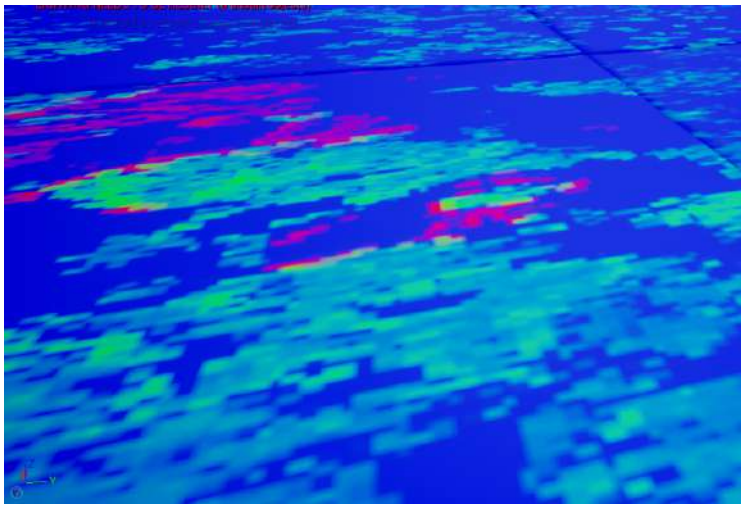


Figura 4.21: Demonstração do efeito de mosaico causado por importar os dados sem serem interpolados.

De modo a resolver este problema foram testados vários algoritmos de *up-sampling*. Os primeiros a serem testados foram os algoritmos internos do *unreal*, um algoritmo de Interpolação bilinear e o de interpolação trilinear. Ambos tiveram resultados bons, no entanto há várias alternativas que não foram testadas.

Também foram implementados vários algoritmos de *up-sampling* de modo a verificar quais eram os que geravam melhores resultados. Os algoritmos foram implementados

diretamente no método de importação dos pesos da *landscape*, mais precisamente no material que gera a renderização que é importada.

Implementar esta componente no material usado na importação tem duas vantagens: Primeiro como a interpolação é feita usando UV's da textura de renderização, o *output* produzido pode ser infinitamente ampliado ou reduzido, onde a resolução da renderização que queremos obter determina a qualidade do output gerado; A segunda vantagem é que como o processo de ampliação está a ocorrer num material implica que ele está a ser executado no *GPU*, sendo melhor em termos de desempenho que uma solução tradicional, mesmo que não seja executado em tempo real.

Em vez de se estar a implementar os algoritmos de *up-sampling* usando expressões dos materiais do *unreal*, o que iria ser uma tarefa bastante complexa, foi usado um nó especial que permite a execução de código *HLSL*, sobre o formato de funções. Assim, todos os algoritmos usados foram desenvolvidos em *HLSL*, o que facilitou a sua implementação e testagem.

Foram testados dois algoritmos, o primeiro foi um algoritmo de interpolação linear, que produziu resultados idênticos aos que foram verificados usando a implementação do *unreal*, e um algoritmo de interpolação bicúbica, o qual aparente gerar resultados mais corretos que os anteriormente mencionados.

Há várias formas de implementar o algoritmo de interpolação bicúbica, a solução implementada obtém a cor de cada valor de cada pixel da imagem usando um polinómio de múltiplas variáveis, onde os valores são interpolados dado uma função g que realiza o somatório do produto entre os coeficientes mostrados mais abaixo (Figura 4.22) e a posição relativa entre dois pixeis (x, y) elevada ao índice que está a ser processado.

$$g(x,y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

Assim, sabendo que p corresponde à matriz da janela de pixeis que está a ser processada, temos os seguintes valores para os coeficientes (Figura 4.22).

A implementação deste algoritmo resultou numa melhoria do processo de import dos dados e consequentemente uma melhoria dos resultados obtidos com o processo de geração de vegetação. Estes resultados são visíveis nas figuras abaixo apresentadas onde podemos comparar os vários algoritmos disponíveis (Figura 4.23).

É importante notar que na figura 4.23 os dados usados continuam a ter algumas falhas e saltos dramáticos para as zonas sem vegetação, este problema não pode ser facilmente resolvido uma fez que se trata de um erro diretamente ligado ao sistema que nos fornece os dados, o sistema não pode gerar para além das limitações dos dados usados.

4.8 Material da *landscape*

Foi anteriormente mencionado o algoritmo de geração de materiais que está a ser utilizado para gerar o material da *landscape*. Na subsecção 4.4.1 foi mencionado que o primeiro

$$
\begin{aligned}
a_{00} &= p_{11} \\
a_{01} &= -\tfrac{1}{2}p_{10} + \tfrac{1}{2}p_{12} \\
a_{02} &= p_{10} - \tfrac{5}{2}p_{11} + 2p_{12} - \tfrac{1}{2}p_{13} \\
a_{03} &= -\tfrac{1}{2}p_{10} + \tfrac{3}{2}p_{11} - \tfrac{3}{2}p_{12} + \tfrac{1}{2}p_{13} \\
a_{10} &= -\tfrac{1}{2}p_{01} + \tfrac{1}{2}p_{21} \\
a_{11} &= \tfrac{1}{4}p_{00} - \tfrac{1}{4}p_{02} - \tfrac{1}{4}p_{20} + \tfrac{1}{4}p_{22} \\
a_{12} &= -\tfrac{1}{2}p_{00} + \tfrac{5}{4}p_{01} - p_{02} + \tfrac{1}{4}p_{03} + \tfrac{1}{2}p_{20} - \tfrac{5}{4}p_{21} + p_{22} - \tfrac{1}{4}p_{23} \\
a_{13} &= \tfrac{1}{4}p_{00} - \tfrac{3}{4}p_{01} + \tfrac{3}{4}p_{02} - \tfrac{1}{4}p_{03} - \tfrac{1}{4}p_{20} + \tfrac{3}{4}p_{21} - \tfrac{3}{4}p_{22} + \tfrac{1}{4}p_{23} \\
a_{20} &= p_{01} - \tfrac{5}{2}p_{11} + 2p_{21} - \tfrac{1}{2}p_{31} \\
a_{21} &= -\tfrac{1}{2}p_{00} + \tfrac{1}{2}p_{02} + \tfrac{5}{4}p_{10} - \tfrac{5}{4}p_{12} - p_{20} + p_{22} + \tfrac{1}{4}p_{30} - \tfrac{1}{4}p_{32} \\
a_{22} &= p_{00} - \tfrac{5}{2}p_{01} + 2p_{02} - \tfrac{1}{2}p_{03} - \tfrac{5}{2}p_{10} + \tfrac{25}{4}p_{11} - 5p_{12} + \tfrac{5}{4}p_{13} + 2p_{20} - 5p_{21} + 4p_{22} - p_{23} - \tfrac{1}{2}p_{30} + \tfrac{5}{4}p_{31} - p_{32} + \tfrac{1}{4}p_{33} \\
a_{23} &= -\tfrac{1}{2}p_{00} + \tfrac{3}{2}p_{01} - \tfrac{3}{2}p_{02} + \tfrac{1}{2}p_{03} + \tfrac{5}{4}p_{10} - \tfrac{15}{4}p_{11} + \tfrac{15}{4}p_{12} - \tfrac{5}{4}p_{13} - p_{20} + 3p_{21} - 3p_{22} + p_{23} + \tfrac{1}{4}p_{30} - \tfrac{3}{4}p_{31} + \tfrac{3}{4}p_{32} - \tfrac{1}{4}p_{33} \\
a_{30} &= -\tfrac{1}{2}p_{01} + \tfrac{3}{2}p_{11} - \tfrac{3}{2}p_{21} + \tfrac{1}{2}p_{31} \\
a_{31} &= \tfrac{1}{4}p_{00} - \tfrac{1}{4}p_{02} - \tfrac{3}{4}p_{10} + \tfrac{3}{4}p_{12} + \tfrac{3}{4}p_{20} - \tfrac{3}{4}p_{22} - \tfrac{1}{4}p_{30} + \tfrac{1}{4}p_{32} \\
a_{32} &= -\tfrac{3}{2}p_{00} + \tfrac{5}{4}p_{01} - p_{02} + \tfrac{1}{4}p_{03} + \tfrac{3}{2}p_{10} - \tfrac{15}{4}p_{11} + 3p_{12} - \tfrac{3}{4}p_{13} - \tfrac{3}{2}p_{20} + \tfrac{15}{4}p_{21} - 3p_{22} + \tfrac{3}{4}p_{23} + \tfrac{1}{2}p_{30} - \tfrac{5}{4}p_{31} + p_{32} - \tfrac{1}{4}p_{33} \\
a_{33} &= \tfrac{1}{4}p_{10} - \tfrac{3}{4}p_{01} + \tfrac{3}{4}p_{02} - \tfrac{1}{4}p_{03} - \tfrac{3}{4}p_{10} + \tfrac{9}{4}p_{11} - \tfrac{9}{4}p_{12} + \tfrac{3}{4}p_{13} + \tfrac{3}{4}p_{20} - \tfrac{9}{4}p_{21} + \tfrac{9}{4}p_{22} - \tfrac{3}{4}p_{23} - \tfrac{1}{4}p_{30} + \tfrac{3}{4}p_{31} - \tfrac{3}{4}p_{32} + \tfrac{1}{4}p_{33}
\end{aligned}
$$

Figura 4.22: Coeficientes usados, imagem de [6].

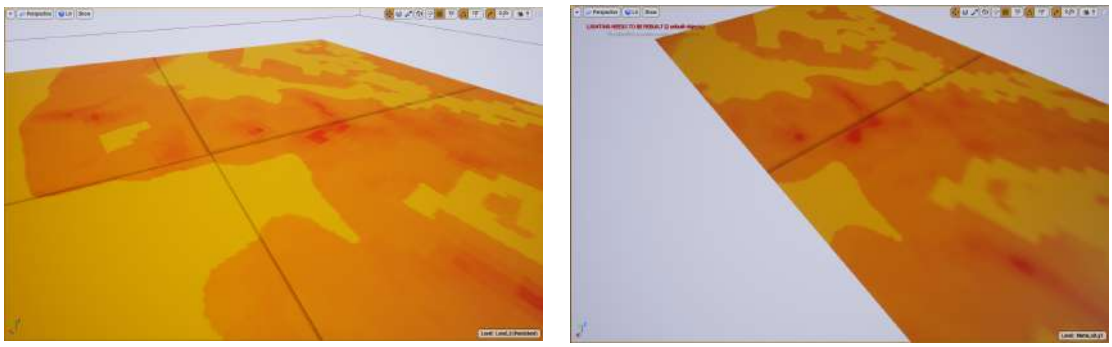


Figura 4.23: Exemplo de um terreno importado sem os valores interpolados (à esquerda) e uma imagem com os valores corretmente interpolados à direita.

material a ser gerado para testar o processo de geração não era o mais adequado para o sistema que estamos a tentar desenvolver, assim foi desenvolvido um outro material usando a mesma biblioteca do *python*.

A função atual de criação de materiais do projeto tem a capacidade de suportar várias opções de geração e gerar materiais de qualidade variável, onde partes do material podem ser desativadas de modo a regular o desempenho e a qualidade do material gerado.

Assim o material planeado tem um conjunto de opções que podem ser ativadas ou desativadas de acordo com o tipo de representação que estamos a tentar gerar, as quais são:

- A quebra da repetição de *tiles* do material (Figura 4.24);
- E opções de qualidade do material e a mudança do valor especular do material. (Figura 4.27)

A maneira mais comum para criar o material de uma *landscape* é fazer *tiling* das texturas ao longo das *UV*'s da *landscape*, no entanto esta técnica faz com que se note a

Figura 4.24: Demonstração do efeito de repetição do *tiling* das texturas.

repetição das texturas que estão a ser usadas para fazer tiling do terreno. Assim, este material usa um conjunto de técnicas para quebrar o efeito de repetição gerado pelo *tiling*, primeiro de acordo com a distância do jogador ao terreno, o material faz a interpolação entre duas texturas, amostradas com tamanhos distintos. Isto faz com que em pontos mais distantes não se note tanto a repetição, dado que são texturas maiores. (Figura 4.25)

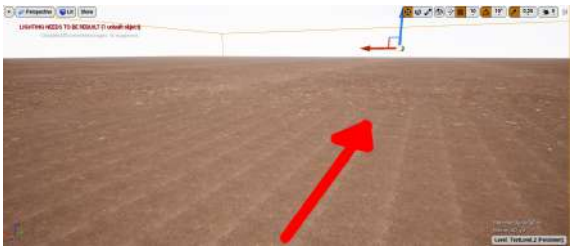


Figura 4.25: Demonstração do efeito de mistura das texturas com base na distância.

Outras técnica usada para quebrar a repetição gerada pelo *tiling* passam por "pintar"zonas da *landscape* com vários tons de cinzento, amostrados de uma textura de variação, esta técnica chama-se Variações de macros, e gera um conjunto de manchas mais escuras no ambiente. Por fim, o material também permite que quando estão a ser estabelecidos os sub-materiais do terreno seja feita uma combinação de duas cores distintas. Esta mistura é feita com base numa textura de ruído de *perlin*, com um valor ajustável (Figura 4.26).

Para além das opções para quebrar a repetição do terreno, o material também permite a utilização opcional de mapas de Normais, rugosidade e AO (*Ambient Occlusion* ou oclusão do ambiente), onde cada um destes mapas podem ser ativados ou desativados individualmente. Ainda dentro deste tópico, o utilizador também tem a capacidade

Figura 4.26: Capacidades do sistema de geração, zonas mais escuras do terreno (esquerda), mistura de texturas do terreno usando ruído de perlin (direita).

de selecionar se quer modificar o valor especular do material de acordo com o tom do ambiente.

Na interface gráfica todas as opções anteriormente descritas têm um campo booleano que permite que o utilizador defina o tipo de representação que quer gerar.(Figura 4.27)

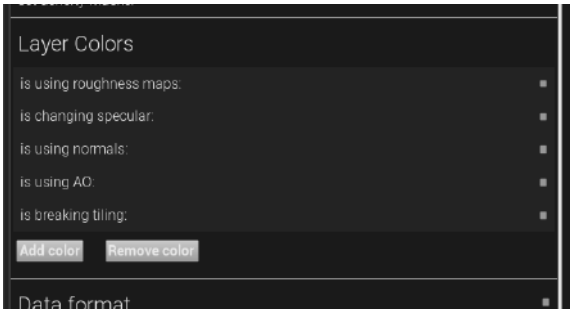


Figura 4.27: Opções de qualidade do material.

Outro ponto importante deste sistema de geração de materiais é que as cores, os sub-materiais visíveis no terreno, são definidos de forma independente das camadas do ambiente, isto faz com que cada camada esteja associada a um material específico, e que várias camadas podem usar o mesmo material. Cada sub-material corresponde a um conjunto de expressões, havendo no menu um nó para fazer a amostragem do seu albedo, e possivelmente outros para configurações de qualidade definidas como mapas de normais, entre outros.

Assim a interface de utilização tem um campo que permite adicionar novos sub-materiais ao material principal e um campo no menu de definição das camadas que permite estabelecer os materiais que representam essa camada.(Figura 4.28). Estes dados são depois guardados num *DataAsset* chamado *PDAColorAsset*.

Figura 4.28: Menu dos sub-materiais da *landscape*.

Depois da configuração do material ser definida no editor estes dados têm de ser passados para o *python*, isto é feito usando as *blueprints* já mencionadas anteriormente que são responsáveis pela conversão de dados das *blueprints* para a formatação de *python*. O objetivo deste passo é gerar uma lista das camadas de vegetação e um dicionário que contém informação sobre o material que está a ser gerado.

A lista é composta pelos nomes das camadas que queremos representar no ambiente, e tem sempre uma entrada com o nome "outros"que representa zonas sem vegetação. o dicionário contém campos para os elementos da geração que podem ser desativados, como

por exemplo os mapas de normais e os mapas da oclusão de ambiente. Para além disto o dicionário também contém uma entrada com uma lista de todas as "cores"que estão representadas no ambiente e um mapeamento entre as cores e a lista de vegetação que podemos encontrar no ambiente, na Listagem 4.6 podemos ver um exemplo da estrutura do dicionário.

Listagem 4.6: Exemplo da formatação do dicionário usado no processamento.

```
{
    'is_breaking_tiling': True,
    'is_changing_specular': True,
    'is_using_normals': True,
    'is_using_ambient_occlusion': True,
    'is_using_roughness': True,
    'visible_layers':
        [
            'Grass', 'DriedGrass', 'ForestGround'
        ],
    'layers':
        {
            'layer_1': [1, 2],
            'layer_2': [0],
            'layer_3': [0]
        }
}
```

Estes dados são enviados para o *python* e este com base na configuração fornecida gera o material desejado. O método de geração é idêntico ao que foi descrito na subsecção 4.4.1, onde as componentes principais do material são as seguintes:

- **A definição dos sub-materiais**: esta é a principal componente deste novo material e é responsável por definir o tipo de materiais que podemos encontrar no terreno, a qual lê os dados das cores que queremos representar e com base nas opções de qualidade que estão definidas gera as expressões necessárias;

- **A definição das opções de quebra da repetição**: esta componente encontra-se internamente divida em três (3) subcomponentes, uma para a definição das expressões responsáveis pela variação de macros, outra para expressões da combinação de materiais do terreno e por fim uma subcomponente para as expressões que definem a variação das texturas com base na distância;

- **A configuração das camadas visíveis**: esta componente gera as expressões responsáveis por definir as camadas visíveis do ambiente, definindo um *LayerBlendNode* que recebe como input as cores resultantes da definição dos sub-materiais, para saber o mapeamento entre cada camada e cada cor é usado o dicionário anteriormente mencionado;

- **A mudança do campo especular do ambiente**: esta componente define todas as expressões que são usadas para modificar o campo especular do material.

Assim, o novo material definido tem a capacidade de gerar materiais de qualidades distintas, configuráveis pelo utilizador, onde cada camada é definida de forma independente associada a um sub-material ou cor do ambiente, permitindo uma maior variedade comparado com o que tínhamos com o material anterior.

4.9 Sistema de gravação do progresso da geração

Um dos erros detetados durante o desenvolvimento tem a ver com o facto de que o *unreal* está constantemente a ir abaixo em certos momentos da geração, devido a falta de memória, mesmo depois de se fazerem otimizações. De modo a poder evitar isso foi desenvolvida uma forma de gravar o progresso da geração de modo a que possa interromper o processo, reiniciar o motor de jogos e retomá-lo do ponto onde foi parado.

Antes de avançar, é importante mencionar que este erro só surge no contexto da geração em terrenos de uma dimensão muito grande onde têm de ser carregadas várias centenas de *tiles*. Suspeita-se que a razão disto acontecer tem a ver com o facto do descarregamento da *tiles* do terreno ser feito de uma forma *lazy*, onde a função responsável por descarregá-las do editor não as remova de imediato, causando um erro de falta de memória no motor de jogo se muitas *tiles* forem carregadas num curto intervalo de tempo.

Para poder gerar a vegetação de zonas muito grandes é preciso dividir o processo nas várias etapas onde o progresso do sistema pode ser guardado. Como já foi mencionado, este erro ocorre enquanto as *tiles* estão a ser carregadas e descarregadas. Isto ocorre em duas instâncias do processo, a primeira é quando está a ser realizada a atribuição do material de cada *tile* do terreno, e a segunda é quando as *tiles* são carregadas para fazer importe dos pesos das camadas que queremos representar.

Dado isto, o sistema de gravação usa uma variável para determinar o progresso da geração, que guarda usando o sistema de gravação interno do *unreal*, esta variável é um valor inteiro que é atualizado sempre que uma nova *tile* é carregada. De modo a evitar que haja falhas inesperadas na geração o sistema interrompe o processo ao fim de um número pré-configurado de *tiles* ter sido carregado, este valor pode ser configurado pelo utilizador e vai variar de acordo com a capacidade do computador que está a correr o sistema.

Durante o processo de geração o sistema também guarda os valores dos *DataAssets* que estão a ser usados para guardar a configuração atual da geração. No início do processo de geração é feita uma verificação para confirmar se há dados guardados, caso haja o processo de geração retoma o processo a partir do último ponto guardado.

Apesar de funcionar corretamente este sistema tem um pequeno problema, sempre que queremos retomar o processo de geração depois dele ser interrompido é preciso guardar todos os dados gerados pelo sistema, isto pode ser feito simplesmente pressionando

o botão *save all assets* no editor do unreal. Para além deste problema, não há uma forma de verificar se os dados das tiles foram descarregados corretamente, ou se ainda se encontram carregados, assim a melhor forma de garantir de que a engine não falha por falta de memória é reiniciá-la sempre que o processo de geração seja interrompido.

Com a implementação desta componente o sistema fica com a capacidade de gerar vegetação para terrenos de grandes dimensões, mesmo com as limitações já mencionadas (Figura 4.29).

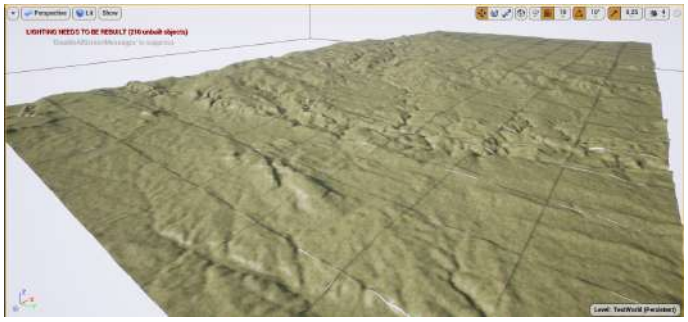


Figura 4.29: Um terreno com duzentas (200) *tiles* no meio do processo de geração, com os dados já parcialmente importados.

4.10 Opções de seleção por camada

Um dos pontos mais importantes para o desenvolvimento deste projeto é garantir que há uma forma de configurar o sistema de geração com o tipo de vegetação que estamos a tentar representar, isto implica garantir que que há várias formas de representar vários tipos de vegetação e várias espécies distintas.

Como já foi mencionado anteriormente os dados de vegetação, tanto a vegetação rasteira como árvores de grande e média porte têm de ser definidos por cada camada, assim o menu de seleção das camadas tem de ter um conjunto de opções que refletem o tipo de vegetação que queremos representar no ambiente.

Antes de mais é importante relembrar os dados que compõem cada camada do ambiente, que são:

- O tipo de vegetação que podemos encontrar no ambiente tanto rasteira como árvores de pequena e grande dimensão;
- Os sub-materiais ou cores do terreno que definem a camada que estamos a tentar representar.

Na secção 4.8 vimos brevemente o processo de seleção dos materiais da camada, assim nesta secção vamos fazer uma análise das restantes definições de cada camada, começando pelos dados de vegetação mais alta.

Estes dados têm uma opção dedicada no menu de definição das camadas do ambiente (Figura 4.30 - A), estas opções estão pré-configuradas no editor usando um *DataAsset*

chamado *PDATreeAsset*, este o qual pode ser modificado pelos utilizadores de modo a que eles possam adicionar outros tipos de vegetação.

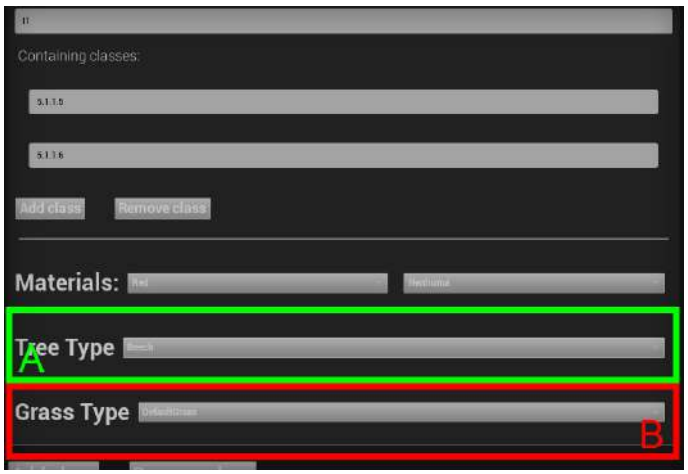


Figura 4.30: Menu de seleção das opções de vegetação das camadas.

Cada planta tem um conjunto de valores que caracterizam a instanciação das malhas no ambiente, estes parâmetros são os seguintes:

- O nome da planta;
- O caminho para a malha estática que representa a planta;
- Um valor booleano que indica se a planta consegue crescer na sombra;
- Um intervalo de valores que indica a variância de altura da planta;
- O raio de colisão da planta;
- A variância de idade que a planta pode apresentar;
- A distância média de dispersão das sementes;
- O raio de sombra da planta;
- E o nível de prioridade da planta no ambiente.

Quando o processo de geração das malhas estáticas de vegetação é chamado, ele recebe também um dicionário com os dados de vegetação que queremos gerar, incluindo os parâmetros anteriormente mencionados e o caminho para o *asset* da malha que representa a vegetação no ambiente. Estes dados são usados para estabelecer os valores de configuração de cada malha estática usada no ambiente, havendo um mapeamento direto entre o valor passado e o parâmetro de configuração da malha estática.

Para a vegetação rasteira foi usado o menu já desenvolvido de processamento de classes da COS, onde foi adicionado um novo campo que permite aos utilizadores indicarem o tipo de vegetação que querem encontrar numa determinada camada do terreno (Figura

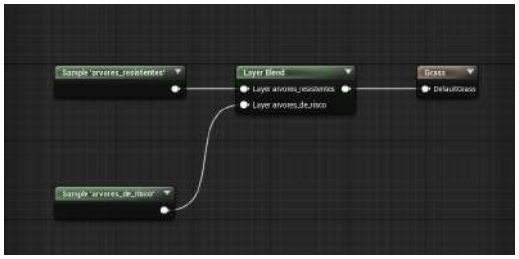


Figura 4.31: Nós da vegetação rasteira.

Figura 4.32: Vegetação rasteira.

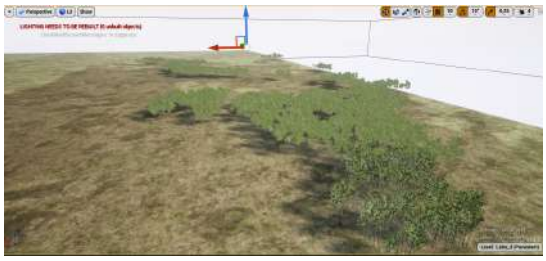


Figura 4.33: Exemplos de vegetação gerada usando a ferramenta desenvolvido.

4.30 - B), semelhante ao que foi feito para os materiais da camada e o tipo de vegetação mais alta.

Este campo corresponde a um menu de seleção onde o utilizador pode selecionar o tipo de vegetação rasteira que quer encontrar numa determinada área. Os elementos da lista são obtidos a partir de um *DataAsset* já mencionado chamado *PDAColorAsset* que contêm a referência de um objeto *GrassType* que é definido pelo utilizador, usando os valores já pré-definidos. A razão pela qual não são gerados vários objetos *GrassType*'s durante o processo de geração é porque a ferramenta *grass* só permite que haja uma instância de cada asset individual, por contraste a *Procedural Foliage Tool* não tem esta restrição, ou seja os seu dados podem ser gerados automáticamente.

Durante a fase de processamento é gerada uma nova componente no material do terreno que corresponde a um conjunto de expressões que guiam o tipo de vegetação rasteira que podemos encontrar na *landscape* gerada (Figuras 4.31 e).

Com estas funções o sistema tem a capacidade de representar um conjunto de vegetação variada onde é completamente controlada pelo utilizador que está a gerar a vegetação os resultados são visíveis nas seguintes figuras (Figuras 4.33 e 4.34). Para além das figuras mostradas neste capitulo é importante mencionar que há mais exemplos do processo de geração na secção do apêndice do documento (Apêndice A e as figuras A.1, A.2, A.3, A.4, A.5, A.6, A.6, A.7 e A.8).

Figura 4.34: Exemplo de vegetação gerada usando a ferramenta desenvolvido.

5

Avaliação

No início do projeto foram estabelecidos os objetivos do sistema desenvolvido, que passavam por desenvolver um sistema capaz de gerar a vegetação de um mundo virtual cuja distribuição espelhasse a do mundo real. Para cumprir este objetivo foi desenvolvido um sistema que funciona por cima das ferramentas já existentes do *unreal* e faz a integração de dados reais no processo de geração.

No entanto, não há forma de saber se a experiência foi ou não um sucesso sem fazer uma avaliação meticulosa dos resultados obtidos com o processo de geração. Deste modo vamos abordar o tema da avaliação do sistema, onde fazemos uma análise das várias métricas segundo as quais o sistema pode ser avaliado, com um foco em particular nas técnicas de avaliação que vão ser usadas.

O enfoque deste capítulo consiste em fazer uma análise geral do processo de avaliação teórico e nos vários processos de avaliação que vão ser usados para avaliar se o sistema cumpre os objetivos definidos, gerar uma representação da camada de vegetação que espelhe a distribuição real da vegetação, com um foco particular na extensibilidade e capacidade de representação do sistema.

Deste modo, este capítulo vai focar-se no estabelecimento das seguintes métricas de avaliação do sistema:

- O desempenho do sistema [5.1.2] avalia o desempenho computacional do sistema desenvolvido segundo um conjunto de métricas, incluindo o nível de automatização do sistema;

- A precisão da representação gerada [5.2.2] corresponde ao quão perto da representação real a distribuição gerada espelha a real, não no sentido de ter precisão ao nível das árvores individuais do ambiente, mas sim em relação à distribuição da representação gerada;

- Para terminar vamos fazer uma análise generalista dos resultados gerados, onde as representações obtidas são comparadas com os dados que foram usados para as gerar e com as ortofotos das zonas que estamos a tentar representar [5.3.2].

5.1 Desempenho do sistema

Um dos pontos mais importantes para avaliar este sistema é garantir que há vantagens significativas em usar o sistema de geração desenvolvido por oposição a gerar a vegetação manualmente, usando as ferramentas internas do *unreal*. Há várias vantagens na utilização de um sistema de geração automatizado. Nesta secção vamos comentar as métricas de desempenho do sistema que foram retiradas do sistema, comparando-as com o processo de geração manual.

Antes de avançar é importante mencionar quais são as métricas específicas que foram recolhidas para avaliar esta componente do sistema. Assim, foram recolhidos valores do tempo de execução segundo um conjunto de configurações distintas, de modo a poderem ser comparados com o tempo do processo de geração manual.

5.1.1 Procedimento experimental

Começando pelos testes de medição do tempo de geração, para a realização dos mesmos teve de ser desenvolvida uma plataforma que permitisse efetuar a recolha de várias medições temporais do processo de geração, a qual tem a capacidade de avaliar o tempo que uma função em particular demora a ser executada.

Desta forma é preciso recolher o tempo do processo de geração e analisar a sua variância segundo um conjunto de parâmetros de *input* distintos. Por si só, saber o tempo médio do processo de geração sem saber quais foram os valores inseridos como *input*, não seria uma métrica correta considerando que o tempo de geração está dependente da zona que está a ser gerada e dos dados que estão a ser usados para a processar.

Assim é preciso fazer uma avaliação do sistema segundo um conjunto de variantes que podem influenciar o tempo de geração:

- **Os hectares do terreno gerado**: Uma das métricas mais indicativas do desempenho do sistema é o tempo de geração por hectare de terreno gerado. De forma a obter o tempo de geração foram realizados vários testes onde a resolução do terreno gerado em hectares é modificado enquanto os restantes permanecem fixos;

- **O número de *tiles* no terreno**: Um dos parâmetros que mais influência o tempo de geração é o número de *tiles* do terreno, porque tal como foi visto no capítulo anterior o processo de carregamento de *tiles* é bastante pesado, ao ponto de poder fazer com que o motor de jogos falhe devido a falta de memória. Assim para avaliar esta métrica o processo de geração foi executado em vários terrenos com dimensões distintas, onde entre cada teste os restantes valores de configuração foram fixados;

- **A resolução das texturas processadas**: A resolução das texturas processadas afeta primariamente a primeira fase do processo de geração, nomeadamente o processo de obtenção e processamento dos dados, assim para testar esta componente foram

fixados todos os parâmetros de geração com exceção da resolução das imagens processadas.

As variantes, descritas anteriormente, foram testadas várias vezes e no final do teste de cada variante foram guardados os valores médios do tempo de execução.

De modo a haver uma base de comparação para estes valores é preciso saber qual é o tempo médio de geração para um utilizador que não está a utilizar o sistema desenvolvido; estes dados têm de ser recolhidos segundo um conjunto de restrições.

Em primeiro lugar, assume-se que as máscaras das camadas da vegetação já foram obtidas, atendendo ao facto de que o processo de obtenção dos dados ia ser demasiado difícil para ser feito de forma puramente manual.

Em segundo lugar, o material gerado manualmente não tem uma fidelidade tão grande como o procedimental, porque o sistema tem uma capacidade de geração de materiais bastante grandes. Assim foi escolhido um material de menor fidelidade para ser recriado, sendo um benefício para o processo manual e dá uma vantagem ao utilizador.

5.1.2 Resultados

Começando pelo processo de geração manual, é importante referir que para testar esta componente foram realizados quatro (4) testes onde gerámos a vegetação de um terreno segundo as restrições descritas anteriormente.Foram usados dois ambientes onde um terreno era dividido em cinquenta e cinco (55) *tiles* e o outro terreno era dividido em seis (6) *tiles*. Os resultados são visíveis na tabela em baixo, sendo que os tempos da tabela estão em segundos (Tabela 5.1).

teste / tamanho	55 tiles	6 tiles
Amostra 1	1441,2 (s)	1261,2 (s)
Amostra 2	1565,4 (s)	1230,3 (s)
Média	1503,3 (s)	1245,75 (s)

Tabela 5.1: Tempos de geração para o processo manual, valores em segundos.

Olhando para estes valores a primeira observação que podemos fazer é que a variação de tempo do processo de geração manual não varia de forma significativa segundo o número de *tiles* do terreno, onde o processo de geração manual com cinquenta e cinco (55) *tiles* dura à volta de vinte e cinco (25) minutos enquanto que para um terreno de seis *tiles* o processo dura uma média de vinte e um (21) minutos. Apesar de parecer uma diferença significativa é preciso ter em conta que o terreno de cinquenta e cinco (55) *tiles* tem muitos mais dados para processar e a diferença de tempo é facilmente justificada porque é perdido mais tempo no carregamento de dados, como iremos ver mais à frente esta diferença também se verifica nos dados do processo automático.

Começando pela variação do tempo do processo de geração segundo o tamanho da área representada, o sistema foi testado em dois cenários de geração um com uma área de cento e trinta e dois (132) hectares e outro com quinhentos e vinte e oito (528) hectares.

Nestes testes as configurações de geração foram mantidas e foram usados os mesmos números de *tiles*, um terreno de cinquenta e cinco (55) *tiles*, onde a única variante neste teste foi a área representada.

É importante notar que para o teste manual não foi experimentado usar várias representações com áreas diferentes porque apenas afeta o processo de obtenção dos dados usados, que como já foi mencionado esta componente não foi avaliada no processo manual, ou seja a mudança da área representada não ia afetar o tempo de geração.

Assim para a testagem do tempo de geração com base na área representada foram realizados três (3) testes por cada área representada, os dados da tabela em baixo correspondem aos tempos recolhidos em segundos (Tabela 5.2).

teste / área	132 ha	528 ha
Amostra 1	97,845 (s)	94,396 (s)
Amostra 2	103,41 (s)	95,373 (s)
Amostra 3	95,055 (s)	101,92 (s)
Média	98,77 (s)	97,23 (s)

Tabela 5.2: Tabela com os tempos de geração segundo várias áreas, valores em segundos.

Em primeiro lugar é importante realçar que não há uma diferença de valores significativa entre o processo de geração com cento e trinta e dois hectares (132) e o de quinhentos e vinte e oito (528) hectares, isto quer dizer que aumentar a área representada não afeta o processo de geração, o que faz sentido tendo em conta que o aumento da área representada só afeta o processo de obtenção dos dados, partindo do princípio que as resoluções dos dados obtidos não varia de acordo com a área processada.

Com esta tabela também podemos concluir que o processo de geração desenvolvido é significativamente mais rápido que o processo de geração manual, onde para um terreno de cinquenta e cinco (55) *tiles*, o processo automático é em média doze (12) vezes mais rápido que o processo de geração manual com o benefício adicional de conseguir gerar materiais mais detalhados.

De seguida, o próximo passo da avaliação foi repetir o processo anteriormente descrito, com a resolução dos dados a escalar de forma proporcional à área do terreno gerado, sendo possível obter uma métrica do tempo de geração por hectare de terreno representado. O processo desta avaliação consistiu em repetir o anteriormente descrito mas sem fixar a resolução dos dados, onde a imagem produzida tinha uma resolução de um metro por pixel. De modo a obter uma amostra de dados mais extensos também foram testadas duas novas representações, uma de duzentos e trinta (230) hectares e outra com trezentos e trinta (330) hectares. Assim foram obtidos os seguintes resultados (Tabela 5.3).

Observando os dados desta tabela podemos verificar que aumentar a resolução dos dados de forma proporcional ao tamanho da zona representada tem um impacto significativo no tempo do processo de geração, isto faz sentido dado que a resolução dos dados usados no processo de geração afeta o desempenho de todas as fases da geração, desde a obtenção dos dados ao processo de importe das texturas geradas para o *unreal*.

teste / área	132 ha	230 ha	330 ha	528 ha
Amostra 1	98,891 (s)	106,24 (s)	124,22 (s)	243,08 (s)
Amostra 2	97,596 (s)	112,21 (s)	130,89 (s)	228,82 (s)
Amostra 3	98,323 (s)	113,70 (s)	136,33 (s)	233,22 (s)
Média	98,27 (s)	110,71 (s)	130,48 (s)	235,04 (s)

Tabela 5.3: Tempos de geração segundo várias áreas, valores em segundos.

É importante mencionar que mesmo com o tempo adicional causado pela utilização de dados de maior dimensão, o processo continua a ser significativamente mais rápido do que a geração manual da vegetação.

Olhando para a figura 5.1 conseguimos verificar que o tempo da geração cresce de forma bastante acrescida de acordo com o tamanho da zona que está a ser gerada, caso os dados de geração estejam a ser escalados de acordo com a área da zona representada.

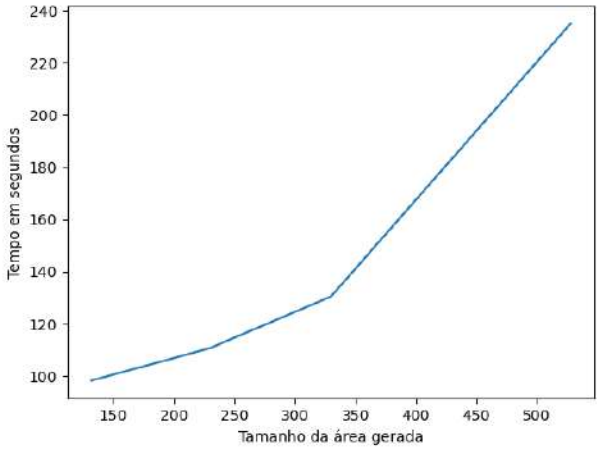


Figura 5.1: Gráfico dos tempos médios contra o tamanho da geração.

A razão pela qual isto acontece tem a ver com as várias tarefas de processamento de imagem executadas ao longo do processo de geração.

A primeira tarefa é a geração do mapa de densidade da vegetação total do ambiente percorrendo todos os pixéis da imagem e fazendo uma operação de *un-mapping* que envolve a utilização de uma tabela de cores de tamanho fixo.

A segunda tarefa é uma multiplicação pixel a pixel de todos os valores da imagem, a qual corresponde a um processo cuja duração depende diretamente do tamanho da imagem processada, crescendo de forma linear segundo este valor.

A terceira tarefa de processamento é o processo de *up-sampling* que é feito antes de os dados serem importados para o terreno, apesar de ser mais complexo este processo corre no *GPU* e a complexidade do algoritmo não justifica o crescimento acrescido que vemos no gráfico.

Se formos verificar o tempo que o sistema de geração passa em cada um dos processos, descritos anteriormente, podemos verificar que para a configuração de quinhentos

e vinte e oito hectares (528) ele passa a maior parte do tempo na primeira tarefa de processamento, em média à volta de cento e cinquenta e cinco segundos (155), ou seja quase sessenta e seis (66) por cento do tempo da geração é passado neste passo. Uma forma de acelerar este processo passava por em vez de usar a biblioteca do *numpy* normal usar uma variante que conseguisse correr num GPU como é o caso da *Numba*.

Por fim o último teste realizado foi medir a variância do tempo de geração de acordo com o número de *tiles* processadas. Para o processo de geração foi usado o sistema com o objetivo de gerar a vegetação no terreno de seis (6) *tiles* e num terreno de cinquenta e cinco (55) *tiles*.Por cada terreno foram realizados três testes e os resultados foram os visíveis na tabela em baixo.

teste / terreno	6 *tiles*	55 *tiles*
Amostra 1	82,946 (s)	107,21 (s)
Amostra 2	84,586 (s)	108,11 (s)
Amostra 3	81,165 (s)	113,03 (s)
Média	82,899 (s)	109,45 (s)

Tabela 5.4: Tempos de geração segundo várias áreas, valores em segundos.

Ao observarmos os valores desta tabela conseguimos verificar que o tempo de geração aumenta ligeiramente quando o número de *tiles* do terreno aumenta, o qual é justificado dado que o processo de carregamento das *tiles* é complexo e envolve o carregamento de um conjunto de dados da memória local para o cenário do terreno.

No entanto a testagem desta componente está limitada pelo facto de que o sistema a partir de um certo número de *tiles* necessita de ser parado para reiniciar o motor de jogos. O número de *tiles* possíveis de carregar varia de acordo com a capacidade do processamento do computador que está a ser usado no processo de geração, sendo que na máquina usada para recolher os dados o limite é à volta de setenta (70) *tiles*. No entanto é importante relembrar que mesmo com estes valores o processo de geração continua a ser mais rápido do que realizando a geração manualmente.

Para concluir é importante relembrar que em todos os casos apresentados o processo de geração automatizado é mais rápido que o processo de geração manual. Para além disso a geração manual implica que o utilizador tenha de estar ativamente a gerar a vegetação, enquanto no processo automático o utilizador não tem de fazer nada quando o processo está a decorrer.

5.2 Precisão da representação gerada

Como já foi referido o grande foco deste projeto consiste no desenvolvimento de um sistema de geração de vegetação que tivesse a capacidade de gerar representações que espelhem a realidade, não sendo necessário garantir que cada planta individual esteja a ser colocada corretamente no terreno, mas sim que a distribuição da vegetação gerada siga a distribuição real.

Dado que a representação é gerada usando dados reais obtidos através de camadas de alta resolução e mapas de cobertura do solo, que podem ser usados para avaliar a precisão dos dados gerados, onde através de um processo semelhante ao que é usado para obter os dados da camada de cobertura florestal, podemos gerar renderizações de mapas de densidade que podem ser usadas para avaliar se a representação gerada segue os dados fornecidos.

Assim, relembramos que os dados da camada de cobertura florestal são obtidos processando ortofotos, onde para cada área de trinta (30) metros quadrados da imagem é calculado a percentagem de pixéis que representam copas de árvores.

Este processo pode ser reproduzido dentro do *unreal* onde através da representação gerada o cenário pode ser dividido em blocos de trinta (30) metros quadrados, três mil (3000) unidades do *unreal*, onde dado o raio das copas representadas, pode ser calculada a área que estas copas ocupam no ambiente, e recriar a métrica anteriormente descrita.

Assim vamos começar por fazer uma pequena descrição do processo que foi usado para avaliar a representação gerada pelo sistema de geração, descrevendo os sistemas que foram desenvolvidos com o objetivo de facilitar o processo de geração[5.2.1]. De seguida, vamos fazer uma análise dos dados que foram obtidos com o processo descrito anteriormente.

5.2.1 Procedimento experimental

Antes de avançar com a descrição é importante relembrar o *output* que é produzido pela *ProceduralFoliageTool*, o qual corresponde a uma representação tridimensional da vegetação no cenário do *unreal*, onde as malhas da vegetação estão associadas aos atores da vegetação.

Como já foi mencionado a avaliação desta componente passa por gerar um mapa de densidade da representação gerada e depois compará-la com os dados que foram usados para a gerar. Assim o primeiro passo do processo de avaliação é gerar o mapa de densidade da representação gerada.

A primeira tentativa de implementaçãodeste processo de geração passou por dividir o terreno em várias secções onde para cada uma delas era gerada uma textura a preto e branco onde apenas renderizo a copa das árvores do ambiente. No entanto este processo é demasiado lento para testar várias representações.

A solução atual é feita em dois passos, primeiro começa por gerar um mapa de alta resolução da vegetação gerada, que contém todas as plantas que se pretendem representar no ambiente. Dada a criação desta imagem ela é posteriormente processada e comparada com o mapa que foi usado como *input* do sistema, isto resulta em dois valores, o erro quadrático médio e o índice de semelhança estrutural entre as duas imagens.

Começando pelo processo de geração do mapa de vegetação, este algoritmo foi implementado no *unreal* usando C++ e usa as posições da vegetação gerada para gerar a textura

desejada. O algoritmo recebe como *input* as posições da vegetação do ambiente, os limites da zona que está a ser representada e a resolução do mapa a ser gerado.

O algoritmo começa por projetar as posições da vegetação no plano XY, metendo o valor do Z (eixo vertical no unreal) a zero (0). De seguida é criada uma textura com a resolução definida pelo utilizador, onde os seus pixéis são iniciados a zero (0).

De seguida, a posição das plantas é transformada num pixel da textura que é pintado com o valor máximo. É importante notar que o sistema desenvolvido usa apenas o canal vermelho da imagem, simplificando e acelerando o processo de geração. Adicionalmente a posição tridimensional da planta no mundo é guardada numa matriz na mesma posição que foi usada para colorir a imagem, de forma a acelerar o próximo passo do processo.

Antes de avançar é importante mencionar que o processo de geração também recebe o raio das copas da vegetação, que se encontra separado em dois valores, um mais baixo corresponde ao círculo onde a copa apresenta a densidade máxima e outro valor que representa o raio máximo da copa, onde a densidade vai gradualmente diminuindo até às extremidades da copa, o efeito final é visível na Figura 5.2.

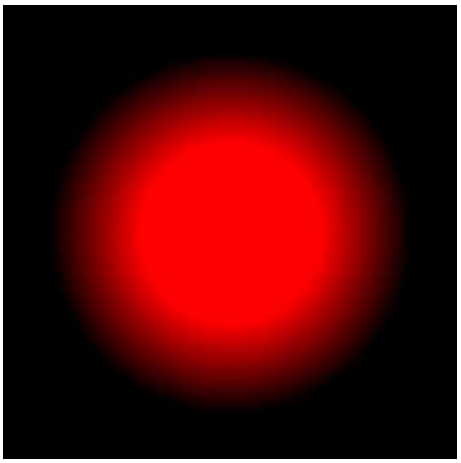


Figura 5.2: Exemplo da representação gerada para a copa de uma planta no ambiente.

Para colorir a copa das árvores é feito um processo de filtragem da imagem gerada, onde é dada uma janela de dimensões iguais ao maior raio considerado. Para cada pixel da imagem percorrem-se todos os pixéis da sua vizinhança usando a janela mencionada anteriormente, para cada pixel da vizinhança é calculada a distância entre a posição especulativa do pixel no cenário e o valor do ponto armazenado na matriz de posições, se a distancia for menor que o raio máximo da copa o pixel é colorido com a densidade adequada. No final é gerada uma figura semelhante à que podemos observar em baixo 5.3.

Este processo pode ser divido pelas várias *tiles* do ambiente o que facilita o processo de geração e permite aumentar significativamente a resolução do mapa de vegetação gerado. Assim, a textura resultante deste primeiro passo tem uma dimensão duas (2) a três (3) vezes maior do que a que foi usada para gerar o terreno.

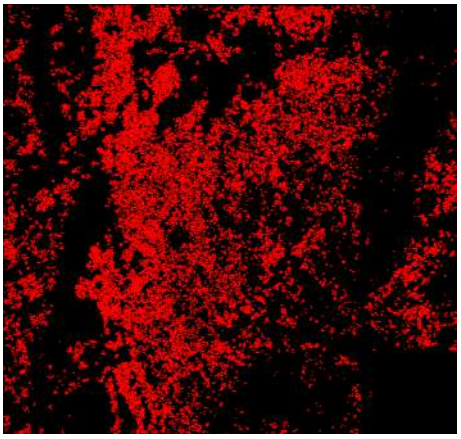


Figura 5.3: Representação gerada no inicio do processo de geração do mapa de densidade.

Figura 5.4: Mapa de densidade e *input*.

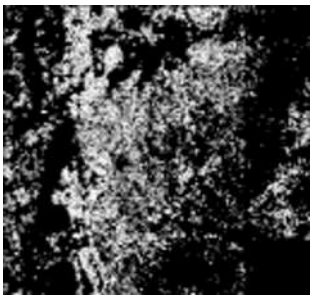


Figura 5.5: Mapa de densidade gerado.

O segundo passo do processo de geração do mapa de densidade de vegetação é feito usando *python*. O primeiro passo deste processo passa por fazer um processo de redução da resolução da imagem, usando um processo semelhante ao que é usado pelas camadas de alta resolução. O algoritmo recebe como input a área que vai segmentar o mapa gerado anteriormente num conjunto de subsecções de resolução igual à definida pela janela.

De seguida calcula-se a média do valor dos pixéis de cada subsecção, e a imagem de densidade final é construída usando estes valores. Esta imagem vai ter uma resolução mais baixa que a usada como *input*, assim é preciso fazer uma ampliação da mesma de modo a que esta fique com a mesma resolução que a textura com qual vai ser comparada. Assim é usado o algoritmo de ampliação anteriormente descrito de modo a que as imagens usem o mesmo tipo de amostragem. As figuras 5.5 e 5.4 mostram, respetivamente, a imagem de *input* e a imagem resultante.

Depois de se gerar a textura são calculados o erro quadrático médio das duas imagens e o índice de semelhança estrutural. O primeiro corresponde a uma medida de erro onde é calculada a distância quadrática entre cada par de pixéis das duas imagens, por contraste a segunda métrica é mais complexa dado que em vez de tentar calcular os erros visíveis entre as duas imagens, tenta modelar as diferenças estruturais da informação das duas imagens, sendo mais adequado para esta componente.

É importante mencionar que apesar de estar a testar o processo de geração em vários cenários usando várias configurações, não estão a ser usados os dados de elevação do

terreno visto que estes podiam alterar a representação resultante do processo de geração procedimental.

Acreditamos que este processo de avaliação é extenso o suficiente para identificar se os dados de densidade da vegetação estão a ser adequadamente interpretados pela *ProceduralFoliageTool* e se a representação resultante segue os valores que foram fornecidos como input. Na subsecção a seguir iremos ver os resultados deste processo de experimentação.

5.2.2 Resultados

Foram recolhidos vários dados usando o processo mencionado anteriormente, contudo de modo a fazer uma análise mais abrangente do processo de geração foram testados vários cenários de geração com parâmetros de geração diferentes. No entanto é importante realçar que para todos os cenários de geração testados, foi mantida a escala da zona que estava a ser gerada de modo a que representação gerada fosse o mais semelhante possível.

Como a escala dos resultados é mantida ao logo dos testes o fator que pode afetar mais a representação gerada é a área total representada. Assim foram conduzidos vários testes em várias zonas com áreas distintas, onde a representação da vegetação foi gerada em áreas cada vez maiores.

De modo a acelerar o processo de geração foi desenvolvido um sistema que dado um limite de maiores dimensões vai selecionar um limite aleatório de menor dimensão, com as dimensões estabelecidas pelo utilizador.

Antes de avançar é importante mencionar brevemente os dados que vão ser analisados, os quais correspondem ao erro quadrático médio e ao índice de semelhança estrutural. O primeiro é um valor entre zero (0) e um (1), onde o zero (0) indica que as imagens são perfeitamente iguais e quanto mais alto for o valor mais diferentes são as imagens, este valor é obtido medindo a distância de todos os pixéis das imagens.

No entanto, a distância dos pixéisda imagem, por si só, não é um bom indicador da semelhança de duas imagens, caso haja ou uma transformação dos valores da imagem ou uma alteração de valores causado por causa de diferenças na densidade da representação gerada, podem-se traduzir em erros maiores.

Assim, o índice de semelhança estrutural corresponde a um valor entre menos um (-1) e um (1), onde um (1) indica que as duas imagens são iguais, e resolve em parte as limitações que podemos encontrar usando apenas o erro quadrático médio.

Para os testes também foi considerada apenas uma representação com várias camadas de vegetação, onde todas contribuíram para o processo de geração do mapa de densidade de *output*, sem contar com a vegetação rasteira.

É importante notar que as zonas testadas podem variar imenso em termos de densidade de vegetação, ou seja o número de plantas por metro quadrado. Este fator afeta a representação gerada, assim de modo a evitar este problema foram selecionados vários cenários aleatoriamente com várias densidades médias. Em baixo podemos observar

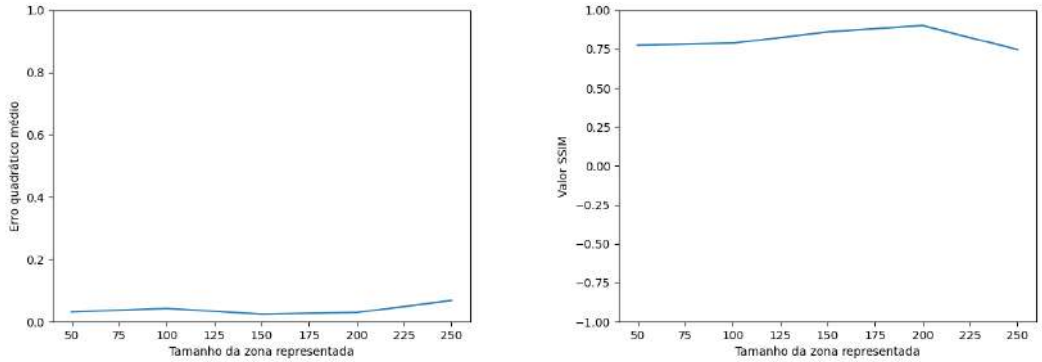


Figura 5.6: Gráfico com erro quadrático médio à esquerda e gráfico índice de semelhança estrutural à direita.

a tabela com os resultados do processo de geração (Tabela 5.5) em conjunto com uma representação gráfica dos mesmos dados (Figura 5.6).

Dimensões	*MSE*	*SSIM*
50 ha	0.0317	0.7737
100 ha	0.0434	0.7870
150 ha	**0.0241**	0.8599
200 ha	0.0299	**0.9004**
250 ha	0.0688	0.7458
Média	0.0396	0.8134

Tabela 5.5: Valores de erro médio recolhidos em vários cenários de geração.

Para gerar os dados desta tabela foram testados cinco (5) cenários de geração diferentes, todos com dimensões distintas, 50ha, 100ha, 150ha, 200ha e 250ha. Para cada área foram geradas seis (6) representações distintas onde tentámos obter a maior variedade de densidades distintas. Para gerar os dados da tabela, foi calculada a média dos valores recolhidos em cada área. A primeira coluna corresponde à média dos erros quadráticos médios recolhidos para a área analisada, enquanto a segunda é a média dos índices de semelhança estrutural da área analisada.

Em apêndice encontra-se um conjunto de algumas imagens que foram usadas como *input* no processo de experimentação para 100ha (Figura A.9), 150h (Figura A.11), 200ha (Figura A.13) e o exemplo apresentado anteriormente corresponde a uma área de 250ha (Figura 5.4), e os mapas de densidade resultantes (Figuras A.10, A.12, A.14 e 5.5, respetivamente).

Como podemos observar não só nas tabelas anteriores (Tabela 5.5), mas também nos gráficos (Figura 5.6), tanto os valores do erro quadrático médio como o índice de semelhança estrutural não variam de forma significativa dado o tamanho da área representada, isto indica que o que os resultados são consistentes, independente do tamanho da área que estamos a tentar representar, onde os melhores resultados obtidos foi um erro quadrático médio de 0.0241 e um indice de semelhança estrutural de 0.9004.

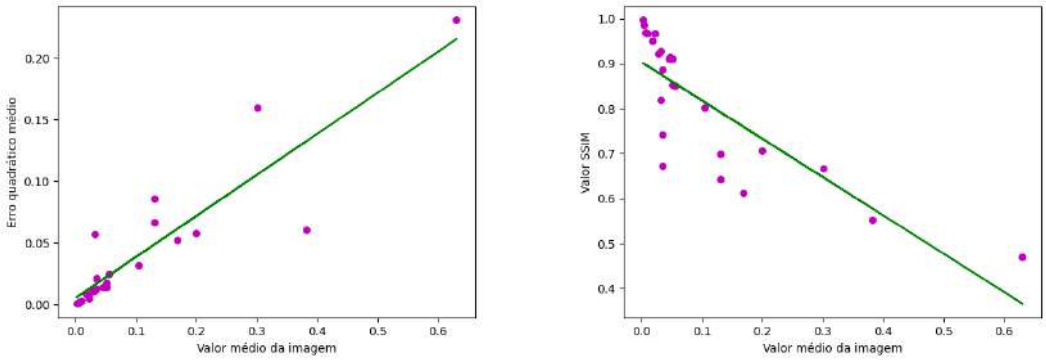


Figura 5.7: Gráfico com erro quadrático médio à esquerda e gráfico índice de semelhança estrutural à direita.

No entanto, é preciso relembrar que estes dados não têm em conta a densidade total da zona observada, dado que a única variante desta avaliação é o tamanho da área representada. Assim, é preciso fazer uma análise dos resultados do processo de geração em vários ambientes com densidades distintas.

Para avaliar este fator são geradas várias representações aleatórias, com valores médios de densidade distintos, para este teste a dimensão da representação não interessa, consequentemente este valor permanece fixo durante a realização deste teste.

Para obter a densidade média da vegetação gerada foi feita a média de todos os pixéis da imagem de *input*, assim temos um valor entre zero (0) e um (1), onde zero (0) é uma zona sem vegetação e um (1) é uma zona completamente coberta de vegetação. Os gráficos em baixo apresentam os resultados obtidos (Figura 5.7).

Observando os gráficos anteriores conseguimos concluir que há uma correlação entre a precisão das representações geradas e a densidade total da vegetação gerada, onde quanto maior é a densidade da representação piores são os resultados obtidos. De modo a facilitar a visualização deste efeito foi efetuada uma redução linear sobre os valores observados.

Este dado faz sentido se considerarmos que quanto maior for a área representada há uma maior possibilidade de haver erros no processo de geração, para além disto, a *ProceduralFoliageTool* tem maior facilidade a preencher zonas com vegetação e fazer a distinção de zonas com ou sem vegetação, do que ajustar a densidade da vegetação gerada. Permite-nos concluir que os resultados obtidos tendem a ter piores valores quando estamos a lidar com zonas com maior densidade de vegetação.

Outro fator que pode contribuir para o aumento da percentagem de erros dada a precisão da zona gerada tem a ver com o processo de geração da textura que é usada para comparar, este processo está sempre associado à geração de algum ruído, que tem tendência a aumentar à medida que o número de plantas representadas no ambiente aumenta.

No entanto é importante notar que o índice de semelhança estrutural tem maior facilidade em ignorar o ruído das imagens, e o padrão de haver piores resultados à medida

que a densidade das plantas aumenta verifica-se tanto no índice de semelhança estrutural como no erro quadrático médio.

Dito isto, é essencial mencionar que apesar de haver uma tendência de piores resultados em zonas de grande densidade da vegetação, estes continuam a não ser maus resultados, referindo que o pior que obtive durante o processo de testagem tinha um índice de semelhança estrutural que rodava à volta de 0.5 e um erro quadrático médio de 0.4, estes valores dado a escala dos valores, menos um a um para o primeiro e zero a um no segundo, não são maus valores maus.

Podemos assim concluir que mesmo tendo em conta as ocasiões onde os resultados obtidos são piores eles continuam a seguir a distribuição fornecida e a manterem-se fiéis às representações que estamos a tentar representar. Dito isto, é importante relembrarmos que apesar de todo este processo de testagem apenas abordar o problema de uma forma analítica, sem ter em conta a visualidade da representação gerada, assim é importante fazer uma análise visual dos resultados obtidos.

5.3 Capacidade de representação da solução

É importante também referir que os resultados apresentados anteriormente, como já foi mencionado, representam uma apreciação objetiva dos dados que foram gerados e pode não ser representativa da qualidade dos mesmos, isto é, a representação gerada mesmo que siga os dados fornecidos pode não ser representativa da zona que se estava a tentar representar, ou vice-versa.

Assim, também é do interesse desta tese haver uma avaliação dos dados gerados onde é feita uma comparação da representação gerada com as ortofotos que estávamos a tentar representar.

Esta comparação entre as representações também pode fornecer uma análise de uma outra métrica, a sua capacidade de representação, ou seja, a quantidade de representações distintas que podem ser geradas segundo limites e camadas já definidas, onde é feita uma análise da capacidade de representação do sistema.

5.3.1 Procedimento experimental

De modo a facilitar o processo de análise e obtenção dos dados foram estabelecidos um conjunto de sistemas que facilitam o processo de obtenção e processamento dos dados que vão ser analisados.

Primeiramente foi desenvolvido um sistema que faz uso dos sistemas do unreal para facilitar o processo de criação de capturas de ecrã que sejam semelhantes às ortofotos com as quais queremos comparar, facilitando assim o processo de comparação das representações geradas com as ortofotos.

De modo a que fosse possível fazer a captura de uma zona inteira foi preciso alterar os modelos que estavam a ser usados para renderizar a vegetação no ambiente, optando

por modelos alternativos que não têm níveis de detalhe. No entanto, a utilização destes modelos dificultou o processo de renderização tendo de ser separado em várias *tiles* onde cada *tile* individual é usada para gerar uma renderização.

Também é importante mencionar que para todas as representações geradas foram usados os mesmos materiais e os mesmos modelos, usando a configuração padrão do sistema de geração. Para além, disto as renderizações foram geradas com os materiais em *unlit*, de modo a facilitar a distinção entre as plantas do ambiente e de modo a melhorar o desempenho do sistema, facilitando o processo de geração das renderizações.

Para além do já mencionado, também foi necessário desenvolver um *script* para obter os dados ortográficos que vão servir de base para a análise dos resultados, este foi implementado em *python* usando as bibliotecas anteriormente mencionadas. O *script* desenvolvido tem um método que dado os limites de uma área obtém a imagem ortográfica e o *DEM* da zona definida.

Como iremos ver na subsecção seguinte os dados vão ser apresentados sobre o formato de uma tabela de modo a facilitar a comparação dos vários resultados. É importante referir que para cada zona apresentada foram testadas várias configurações distintas, e é quase sempre representado todo o tipo de vegetação presente na área escolhida.

5.3.2 Discussão

Na tabela em baixo (Tabela 5.6) conseguimos um conjunto de dados que vão ser analisados nesta discussão, os quais são:

- O DEM da zona representada;
- Os dados de *input* que foram usados para gerar a representação da vegetação;
- Uma renderização da representação gerada;
- E uma ortofoto da zona que estamos a tentar representar.

Todas as imagens presentes na tabela em baixo também podem ser visualizadas no Apêndice A.

Olhando para os dados da tabela temos de analisá-los tendo em conta o objetivo inicial da tese, gerar uma representação da vegetação que não se focasse no realismo mas sim em seguir os padrões de densidade da vegetação real. Que conseguisse suportar a geração de várias camadas de vegetação, dando principal ênfase à camada com a vegetação de maior dimensão. O processo de geração também tem de ter a capacidade de conseguir gerar representações com vários tipos de vegetação, facilitando a visualização de vários tipos de vegetação no ambiente.

Antes de avançar é importante mencionar que os resultados que obtivemos durante o processo de testagem estão dependentes da qualidade dos dados que foram usados, nomeadamente a camada de cobertura florestal e a carta de uso e ocupação do solo. Mesmo

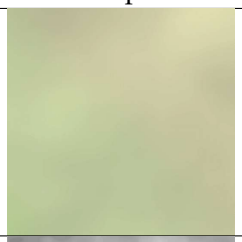
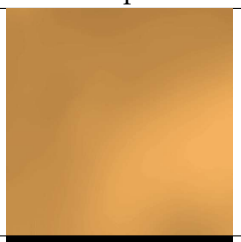
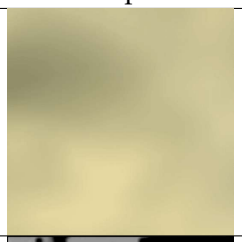
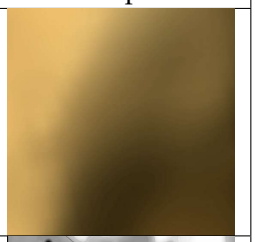
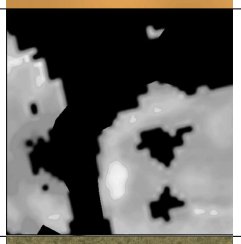
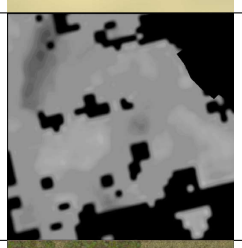
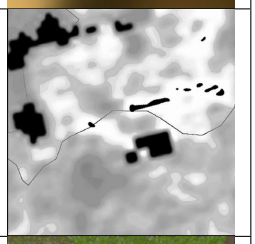
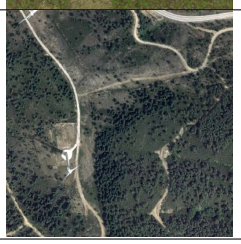
Dados	Exemplo 1	Exemplo 2	Exemplo 3	Exemplo 4
DEM				
Input				
Render				
Orto-foto				

Tabela 5.6: Representações geradas.

que estes dados sejam ampliados e combinados continuam a fornecer uma quantidade limitada de informação que está dependente da sua resolução.

Começando pelo exemplo da primeira coluna da tabela, comparando a representação gerada (Figura A.23) com a ortofoto da zona que estamos a representar(Figura A.27), conseguimos observar que a vegetação resultante segue a distribuição visível na ortofoto e as zonas com vegetação estão corretamente delimitadas na representação gerada.

No entanto, conseguimos imediatamente observar um problema com as estradas que se encontram visíveis na ortofoto, apesar da vegetação à volta das estrada seguir os padrões observados na ortofoto, as estradas em si não foram tidas em conta no processo de geração.

Isto ocorre porque tanto a COS como as camadas de alta resolução têm uma resolução demasiado baixa para conseguirem capturar a falha causada pela estrada. A COS tem uma distância mínima entre linhas de seis (6) metros enquanto a camada de cobertura florestal tem uma resolução de trinta (30) metros quadrados por pixel.

Recordo que o processo de geração é puramente inclusivo o que quer dizer que o utilizador apenas define as classes que vão ser usadas. Na configuração usada as camadas

de vegetação não incluem as classes correspondentes a estradas, assim se a estrada for capturada pela COS ela não vai ser incluída no mapa de densidade de input, no entanto, neste caso, a carta de uso e ocupação do solo não capturou a estrada visível na ortofoto. Assim podemos observar que a representação está puramente dependente do mapa de densidade que foi usado como input (Figura A.19) o qual também não tem em conta a estrada.

Este problema podia ser resolvido se houvesse uma integração de dados adicionais ou a geração de estruturas permanentes, o segundo já foi desenvolvido numa outra tese deste projeto, no entanto, dado que este teste se focava na geração da vegetação não havia necessidade de integrar este sistema dentro do processo de geração.

Por contraste, nos restantes exemplos conseguimos ver um dos benefícios da integração de dados da COS no processo de geração, na segunda e na terceira representação a contar da esquerda conseguimos observar como a utilização da COS facilitou o recorte da zona com vegetação permitindo que a representação gerada fosse mais próxima do que é observado na ortofoto.

Por exemplo, no segundo exemplo (Figura A.24) há um recorte muito mais suave e preciso à volta das estradas e da ventoinha eólica. Se observarmos o mapa de densidade de input (Figura A.20) conseguimos concluir que esta situação ocorre graças ao input da COS, que ajuda a cortar o mapa de densidade da camada de alta resolução evitando o efeito de mosaico já anteriormente mencionado.

Um dos problemas associados com a baixa resolução dos dados é que dado o processo que é usado para obter os dados de cobertura florestal há uma tendência a haver saltos de densidade muito bruscos sem qualquer forma de interpolação, ou seja na maioria dos testes realizados não há uma transição suave entre zonas de alta e baixa densidade.

Como já foi mencionado anteriormente foi implementado um algoritmo de interpolação bicúbica de modo a tentar evitar este problema, no entanto, em situações como o segundo exemplo, onde no lado direito há uma zona de alta densidade de vegetação diretamente ao lado de uma zona de baixa densidade, a zona de baixa densidade acaba por ser removida a favor da zona de maior densidade, isto verifica-se no mapa de densidade da zona representada (Figura A.20).

Algo que é importante notar é que no último exemplo apresentado, a representação gerada (Figura A.26) tem dois tipos de vegetação, a zona menos densa corresponde a uma floresta de pinheiros enquanto a zona mais densa é uma floresta de carvalhos. Aqui podemos observar uma das vantagens do sistema de geração atual em comparação com a ortofoto sendo que é mais fácil de distinguir os limites de cada floresta na representação gerada do que na ortofoto, facilitando desta forma o processo de identificação de zonas de risco e o tipo de vegetação.

No entanto, com este exemplo podemos observar outro problema associado com os dados da camada de cobertura florestal. Como os dados da camada de cobertura florestal são baseados na área que a copa de cada planta ocupa no ambiente e não com base no número de plantas no ambiente, pode haver situações onde uma zona com densidade alta

tem apenas árvores de grande dimensão com copas grandes, em vez de várias plantas, sendo que a camada de cobertura florestal por si só não faz a distinção destes dois casos.

Assim, por vezes ocorrem situações como a que podemos observar na última representação apresentada, onde apesar da vegetação estar a seguir a densidade definida pelos mapas de densidade de *input*, o tamanho da vegetação apresentada não segue a que é visível na ortofoto (Figura A.30).

Uma forma de resolver este problema poderia ser a integração de dados da idade da vegetação de um determinado local, e usar a idade da vegetação como uma forma de aproximar o tamanho da planta representada. É importante referir que o sistema atual tem a capacidade definir vários tipos de vegetação com dimensões diferentes, no entanto, a *ProceduralFoliageTool* não suporta um método para definir automaticamente o tamanho de cada planta no ambiente.Sendo assim, implementar esta função implicaria mudar o funcionamento da ferramenta atual.

No entanto, tenho de relembrar que esta tese não procurava gerar uma representação foto realista, mas sim uma representação precisa, e tendo isto em conta devo referir que a distinção entre vegetação com várias dimensões dentro da própria espécie, não está no âmbito desta tese, podendo vir a ser abordada em trabalho futuro.

Para terminar, é importante relembrar que o objetivo não era ter uma representação realista como uma precisão ao nível de cada planta individual, mas sim ter uma representação precisa que se baseasse em dados reais e que facilitasse o processo de identificação da vegetação no ambiente. Finalizando, acredito que o sistema atual, apesar de ter as suas limitações consegue ser competente o suficiente para cumprir estes pontos.

6

Conclusão e trabalho futuro

Para concluir é importante relembrar o grande foco desta dissertação, desenvolver um sistema que tenha a capacidade de gerar a camada de vegetação de um cenário, recorrendo a dados gerados a partir de ferramentas de deteção remota. O sistema não se foca em criar uma representação foto-realista mas sim uma cuja distribuição da vegetação no ambiente espelhe a real, não sendo necessário ter a precisão ao nível de plantas individuais, mas sim da distribuição completa. Para além disto convém que o sistema seja modelado e que tenha a capacidade de gerar uma grande variedade de representações distintas.

Apesar das limitações apresentadas pelo sistema desenvolvido a grande maioria das condições foram cumpridas com um nível de sucesso variável, principalmente no que toca a geração de representações de uma dimensão elevada, não só por causa das limitações do sistema desenvolvido como também por causa das limitações da plataforma onde foi desenvolvido, o *unrealengine* 4. Para além destas limitações a qualidade dos outputs gerados está sempre dependente dos dados usados na geração, não podendo nunca gerar uma representação melhor do que a ditada pelos dados.

Contudo considero que há partes do projeto que poderiam ser melhoradas, sendo que uma das melhorias imediatas passaria por atualizar o sistema de modo a que funcionasse com o sistema *landmass* do *unrealengine* 5. Este sistema não detém as limitações do sistema de *landscape* que está a ser usado no projeto atualmente. Outro ponto interessante a realçar seria a integração de um algoritmo de up-sampling mais sofisticado que fizesse uso de dados adicionais, ou a utilização de dados adicionais para determinar a composição do material do terreno.

Por fim, uma futura melhoria no projeto poderia passar por desenvolver um algoritmo de geração de vegetação que funcionasse de forma independente dos sistemas internos do *unreal*, dado que iria reduzir de uma forma imensa o número de dependências do sistema atual.

Bibliografia

[1] U. F. A. F. Academy. *Virtual reality training may save firefighter lives*. 2020. URL: https://www.usfa.fema.gov/training/coffee_break/070820.html (ver p. 1).

[2] M. Alsweis e O. Deussen. "Modeling and visualization of symmetric and asymmetric plant competition". Em: *Eurographics*. 2005, pp. 83–88 (ver p. 9).

[3] C. Andújar et al. "Inexpensive reconstruction and rendering of realistic roadside landscapes". Em: *Computer Graphics Forum*. Vol. 33. 6. Wiley Online Library. 2014, pp. 101–117 (ver pp. 8, 12, 14, 15).

[4] B. Benes e E. D. Espinosa. "Modeling virtual ecosystems with the proactive guidance of agents". Em: *Proceedings 11th IEEE International Workshop on Program Comprehension*. IEEE. 2003, pp. 126–131 (ver p. 9).

[5] G. A. Bradbury et al. "Guided ecological simulation for artistic editing of plant distributions in natural scenes". Em: *Journal of Computer Graphics Techniques Vol* 4.4 (2015) (ver pp. 8, 11).

[6] P. Breeuwsma. *Cubic interpolation*. 2016. URL: https://www.paulinternet.nl/?page=bicubic (ver p. 61).

[7] M. Caetano e F. Marcelino. *Especificações Técnicas da Carta de Uso e Ocupação do Solo (COS) de Portugal Continental para 2018*. Direção-Geral do Território, 2019. Especificações técnicas da Carta de Uso e Ocupação do Solo (COS) de Portugal Continental para 2018. Relatório Técnico. Direção-Geral do Território. 2019. URL: https://www.dgterritorio.gov.pt/sites/default/files/documentos-publicos/2019-12-26-11-47-32-0__ET-COS-2018_v1.pdf (ver pp. 17, 18).

[8] E. Carrier. *Procedural World Generation of Far Cry 5*. Nov. de 2018. URL: https://youtu.be/JBp8zvLVsgg (ver pp. 9, 11).

[9] CDC. *Virtual Reality Laboratory Training*. 2021. URL: https://www.cdc.gov/labtraining/vr.html (ver p. 1).

[10] E. Ch'ng. "Realistic placement of plants for virtual environments". Em: *IEEE computer graphics and applications* 31.4 (2010), pp. 66–77 (ver pp. 9, 10).

[11] E. Ch'Ng. "Model resolution in complex systems simulation: Agent preferences, behavior, dynamics and n-tiered networks". Em: *Simulation* 89.5 (2013), pp. 635–659 (ver pp. 9, 10).

[12] K. Compton. *Practical Procedural Generation for Everyone*. In this 2017 GDC session, Tracery developer Kate Compton explains the many surprisingly simple algorithms of procedural content generation. Mai. de 2017. URL: https://youtu.be/WumyfLEa6bU (ver pp. 5, 6).

[13] Copernicus e L. M. Service. *CORINE Land Cover and usage*. 2018. URL: https://land.copernicus.eu/pan-european/corine-land-cover (ver p. 19).

[14] G. Cordonnier et al. "Authoring landscapes by combining ecosystem and terrain erosion simulation". Em: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–12 (ver pp. 9, 10).

[15] O. Deussen et al. "Realistic modeling and rendering of plant ecosystems". Em: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, pp. 275–286 (ver pp. 9, 13).

[16] A. Emilien et al. "Worldbrush: Interactive example-based synthesis of procedural virtual worlds". Em: *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–11 (ver p. 11).

[17] J. Gain et al. "EcoBrush: Interactive control of visually consistent large-scale ecosystems". Em: *Computer Graphics Forum*. Vol. 36. 2. Wiley Online Library. 2017, pp. 63–73 (ver p. 13).

[18] S. Gillian. *Making Things Up: The Power and Peril of PCG*. Mar. de 2015. URL: https://youtu.be/B11R1HZsmGE (ver p. 6).

[19] C. Insights. *The $120B Gaming Industry Is Being Built On The Backs Of These Two Engines*. 2019. URL: https://www.cbinsights.com/research/game-engines-growth-expert-intelligence/ (ver p. 3).

[20] M. Jaap. *GPU-Based Run-Time Procedural Placement in Horizon: Zero Dawn*. Dez. de 2017. URL: https://youtu.be/ToCozpl1sYY (ver pp. 11, 12).

[21] K. Karantzalos e D. Argialas. "Towards automatic olive tree extraction from satellite imagery". Em: *Geo-Imagery Bridging Continents. XXth ISPRS Congress*. Citeseer. 2004, pp. 12–23 (ver p. 14).

[22] M. Konyk e M. Trottier. *Technical Artist Bootcamp: Procedural Islands of Dauntless*. In this 2018 GDC talk, Phoenix Labs' Mykola Konyk and Michael Trottier present the procedural island pipeline they created in order to build the shattered islands of Dauntless. Jul. de 2018. URL: https://youtu.be/krxmNpqKxtI (ver p. 25).

[23] T. Langanke. *Copernicus Land Monitoring Service – High Resolution Layer Forest*. https://land.copernicus.eu/user-corner/technical-library/hrl-forest (ver pp. 19, 20).

[24] T. Langanke. *Copernicus Land Monitoring Service – High Resolution Layer Grassland*. https://land.copernicus.eu/user-corner/technical-library/hrl-grassland-technical-document-prod-2015 (ver pp. 19, 20).

[25] T. Langanke. *Copernicus Land Monitoring Service – High Resolution Layer Imperviousness*. https://land.copernicus.eu/user-corner/technical-library/hrl-imperviousness-technical-document-prod-2015 (ver p. 19).

[26] T. Langanke. *Copernicus Land Monitoring Service – High Resolution Layer Water and Wetness*. https://land.copernicus.eu/user-corner/technical-library/hrl-water-wetness-technical-document-prod-2015 (ver p. 19).

[27] H. E. Lowood. *Virtual reality. Encyclopedia Britannica*. 2020. URL: https://www.britannica.com/technology/virtual-reality (ver p. 1).

[28] M. Makowski et al. "Synthetic silviculture: multi-scale modeling of plant ecosystems". Em: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–14 (ver p. 8).

[29] B. T. do Nascimento, F. P. Franzin e C. T. Pozzer. "Gpu-based real-time procedural distribution of vegetation on large-scale virtual terrains". Em: *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE. 2018, pp. 157–15709 (ver pp. 11, 13, 14, 17).

[30] T. Niese et al. "Procedural Urban Forestry". Em: *arXiv preprint arXiv:2008.05567* (2020) (ver pp. 14–16).

[31] P. Prusinkiewicz et al. "Visual models of plant development". Em: *Handbook of formal languages*. Springer, 1997, pp. 535–597 (ver p. 13).

[32] H. Samet. "The quadtree and related hierarchical data structures". Em: *ACM Computing Surveys (CSUR)* 16.2 (1984), pp. 187–260 (ver p. 13).

[33] V. R. Society. *Virtual Reality and Education*. 2020. URL: https://www.vrs.org.uk/virtual-reality-education/ (ver p. 1).

[34] V. R. Society. *Virtual Reality in Entertainment*. 2020. URL: https://www.vrs.org.uk/virtual-reality-applications/entertainment.html (ver p. 1).

[35] H. Team. *Houdini engine for game dev*. https://www.sidefx.com/media/uploads/products/engine/hengine_games.pdf (ver p. 24).

[36] M. Toftedahl e H. Engström. "A taxonomy of game engines and the tools that drive the industry". Em: *DiGRA 2019, The 12th Digital Games Research Association Conference, Kyoto, Japan, August, 6-10, 2019*. Digital Games Research Association (DiGRA). 2019 (ver pp. 3, 21).

[37] Unreal e E. G. team. *Procedural Foliage Tool documentation*. 2014. URL: https://docs.unrealengine.com/en-US/BuildingWorlds/OpenWorldTools/ProceduralFoliage/QuickStart/index.html (ver pp. 22, 23).

[38] Unreal e E. G. team. *Procedural Foliage Tool documentation*. 2014. URL: https://docs.unrealengine.com/en-US/BuildingWorlds/OpenWorldTools/Grass/QuickStart/index.html (ver p. 24).

[39] Unreal e E. G. team. *Unreal DataAsset documentation*. 2014. URL: https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/AssetManagement/ (ver p. 46).

[40] Unreal e E. G. team. *Unreal plugin documentation*. 2014. URL: https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/Plugins/ (ver p. 58).

[41] Unreal e E. G. team. *Unreal python plugin documentation*. 2014. URL: https://docs.unrealengine.com/4.27/en-US/PythonAPI/ (ver p. 34).

[42] *Visualizador do território da DGT*. http://mapas.dgterritorio.gov.pt/ (ver p. 19).

[43] D. wiki. *A Brief History of Computer Graphics*. 2019. URL: https://deseng.ryerson.ca/dokuwiki/mec222:brief_history_of_computer_graphics (ver p. 1).

[44] A. Willmott. "Fast object distribution." Em: *SIGGRAPH sketches*. 2007, p. 80 (ver p. 6).

[45] C. Xiao et al. "Individual tree detection from multi-view satellite images". Em: *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE. 2018, pp. 3967–3970 (ver p. 15).

[46] X. X. Zhu et al. "Deep learning in remote sensing: A comprehensive review and list of resources". Em: *IEEE Geoscience and Remote Sensing Magazine* 5.4 (2017), pp. 31–35 (ver p. 3).

A

Apêndice 1

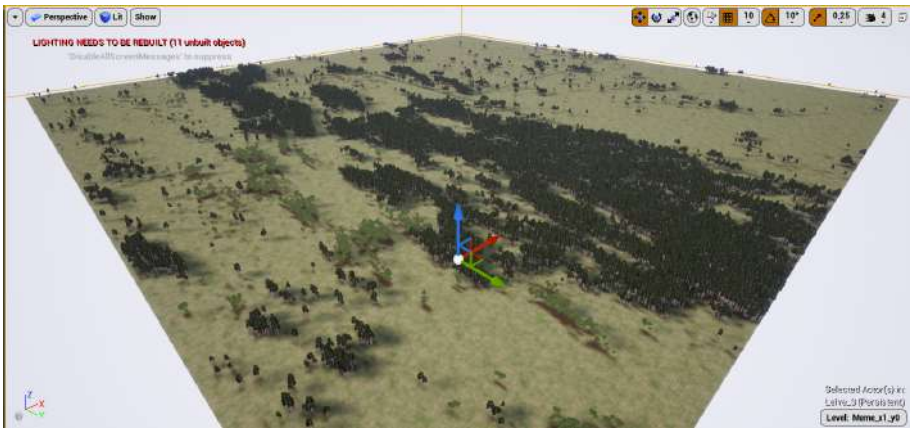


Figura A.1: Vegetação gerada usando a ferramenta desenvolvida, exemplo 1.

Figura A.2: Vegetação gerada usando a ferramenta desenvolvida, exemplo 2.

Figura A.3: Vegetação gerada usando a ferramenta desenvolvida, exemplo 3.

Figura A.4: Vegetação gerada usando a ferramenta desenvolvida, exemplo 4.

Figura A.5: Vegetação gerada usando a ferramenta desenvolvida, exemplo 5.

Figura A.6: Vegetação gerada usando a ferramenta desenvolvida, exemplo 6.

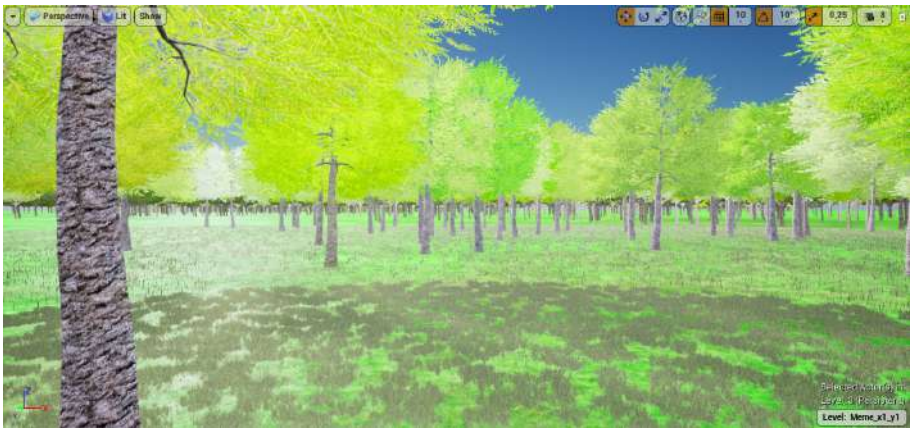


Figura A.7: Vegetação gerada usando a ferramenta desenvolvida, exemplo 7.

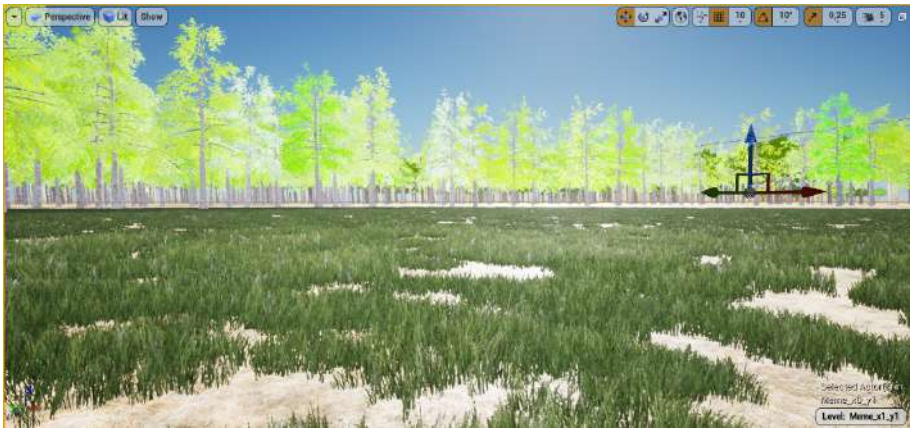


Figura A.8: Vegetação gerada usando a ferramenta desenvolvida, exemplo 8.

Figura A.9: Dados de *input* do processo de geração, exemplo 1.

Figura A.10: Dados gerados no final do processo de geração, exemplo 1.

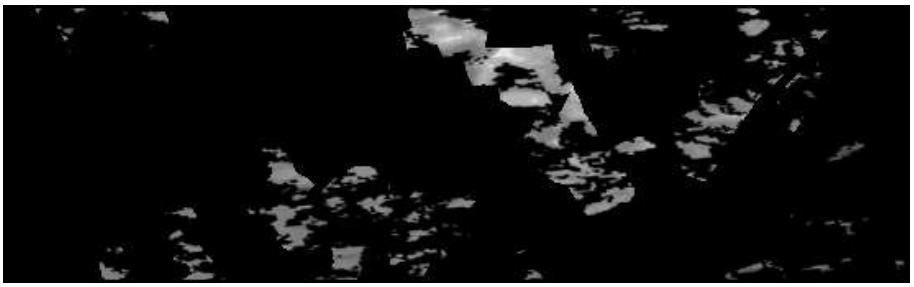


Figura A.11: Dados de *input* do processo de geração, exemplo 2.

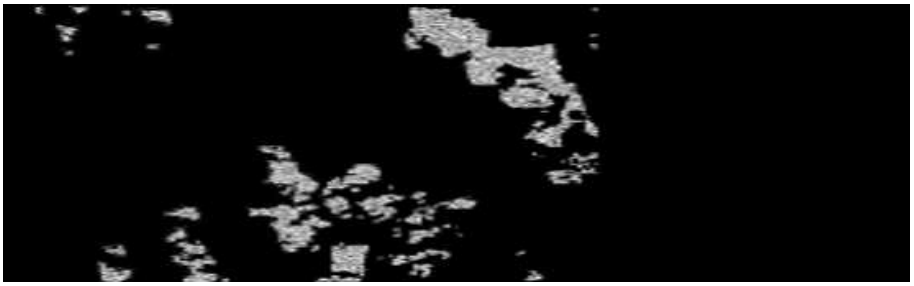


Figura A.12: Dados gerados no final do processo de geração, exemplo 2.

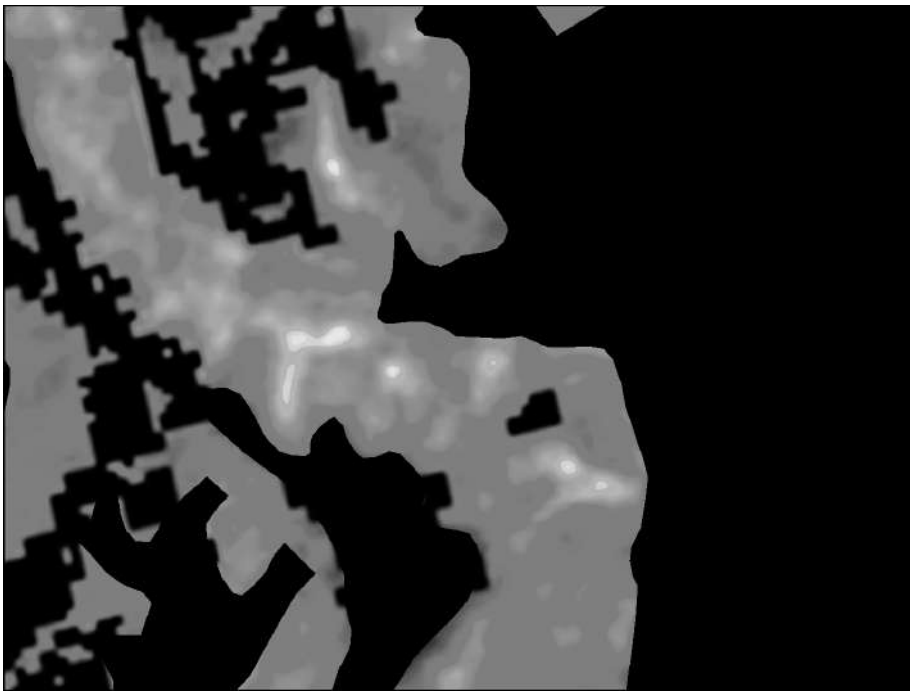


Figura A.13: Dados de *input* do processo de geração, exemplo 3.

Figura A.14: Dados gerados no final do processo de geração, exemplo 3.

Figura A.15: DEM do exemplo 1.

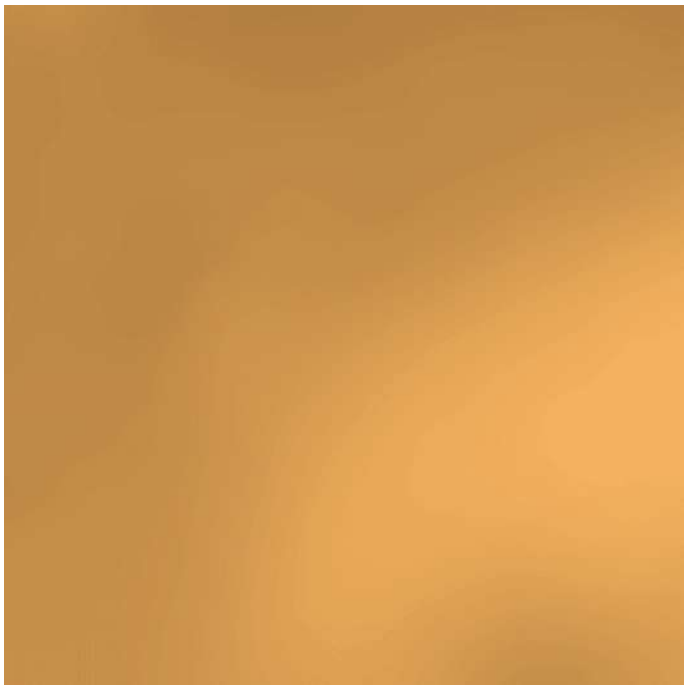


Figura A.16: DEM do exemplo 2.

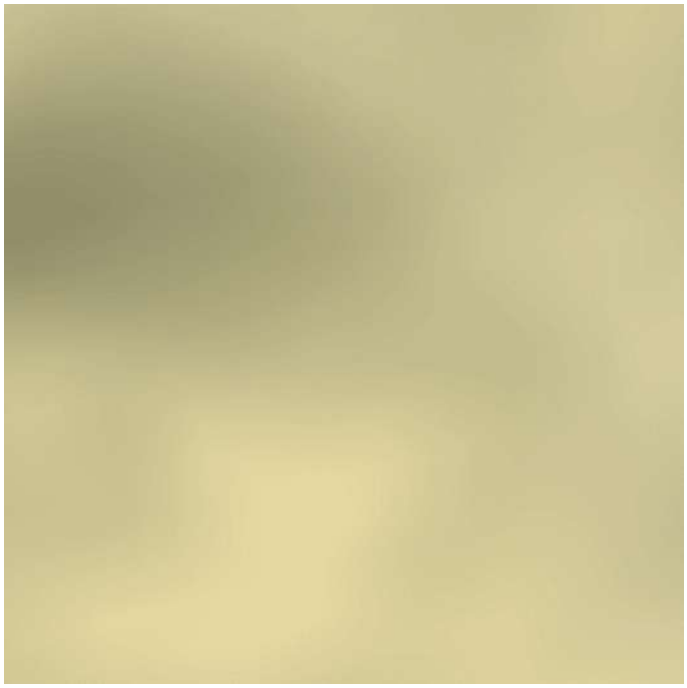


Figura A.17: DEM do exemplo 3.

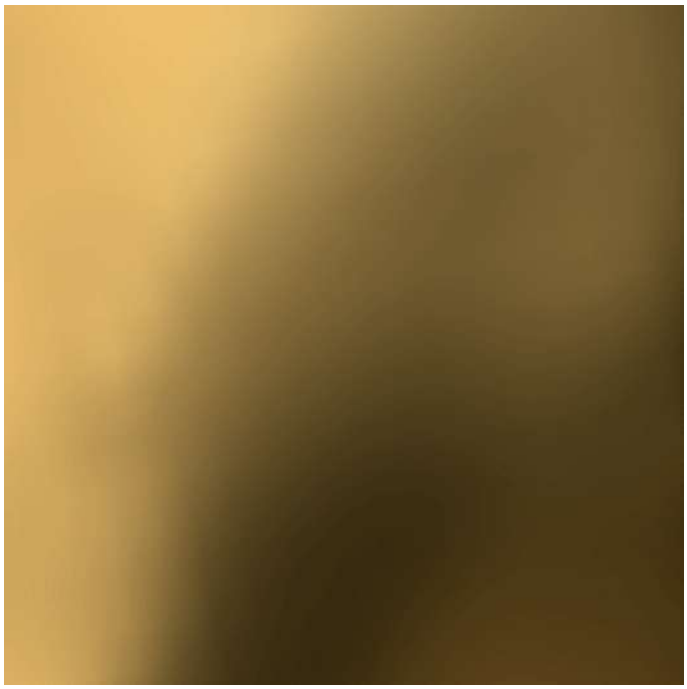


Figura A.18: DEM do exemplo 4.

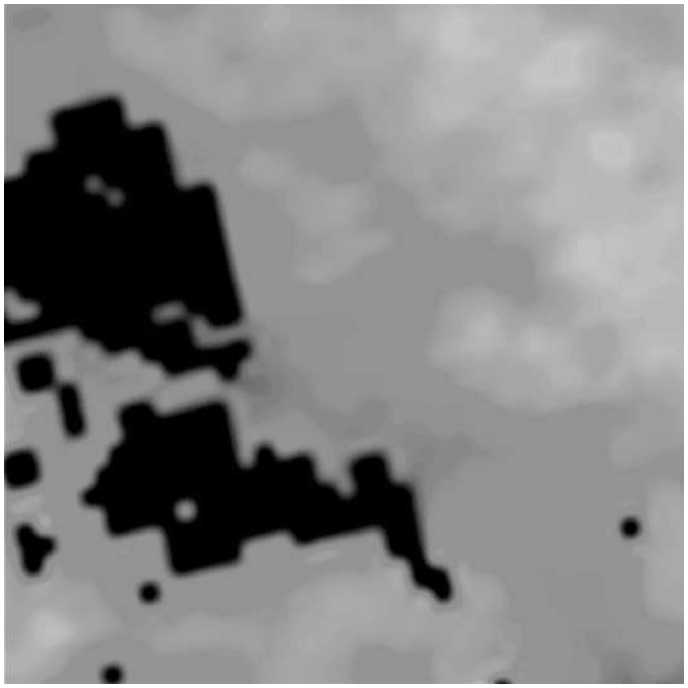


Figura A.19: Mapa de densidade do exemplo 1.

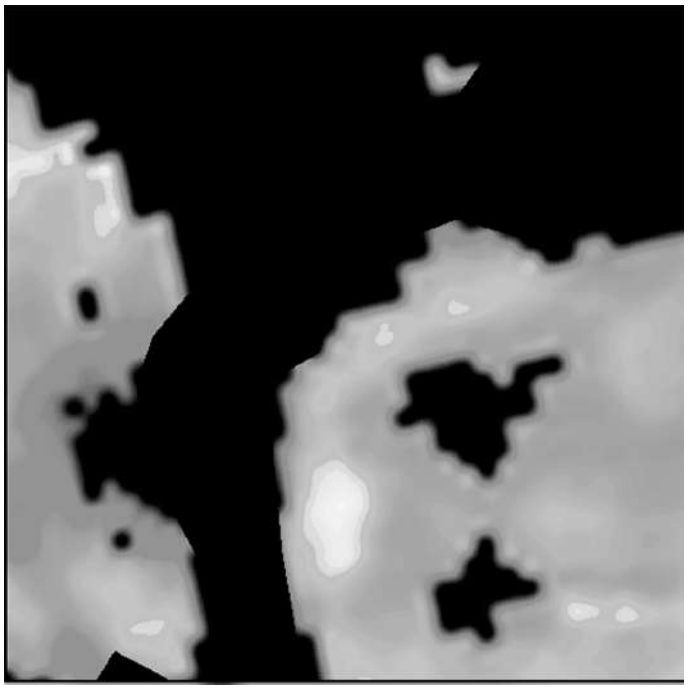


Figura A.20: Mapa de densidade do exemplo 2.

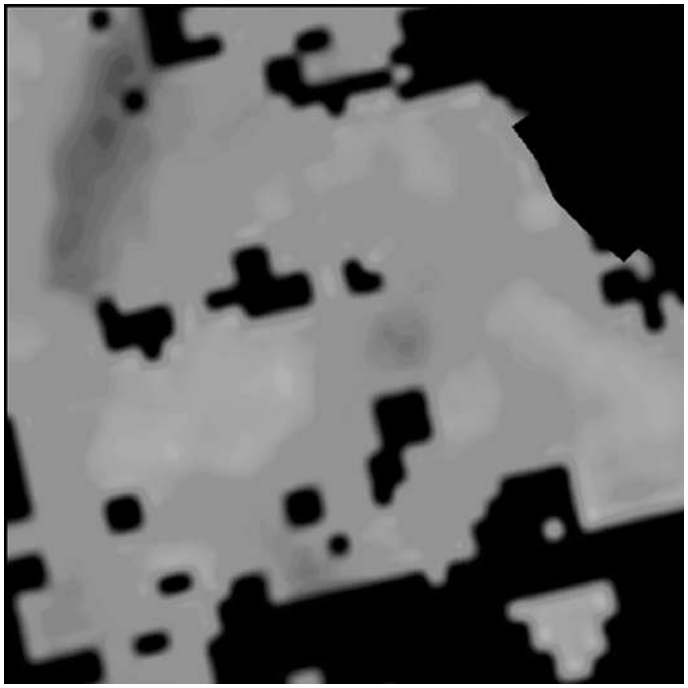


Figura A.21: Mapa de densidade do exemplo 3.

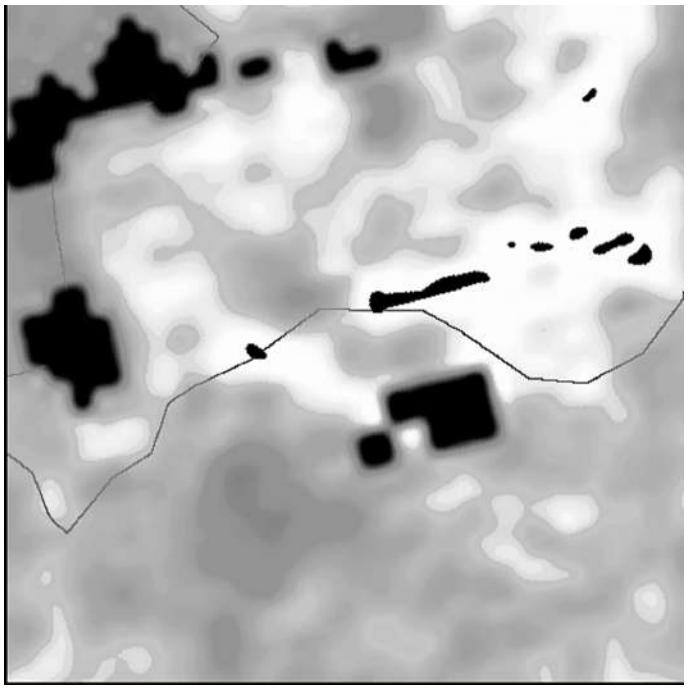


Figura A.22: Mapa de densidade do exemplo 4.

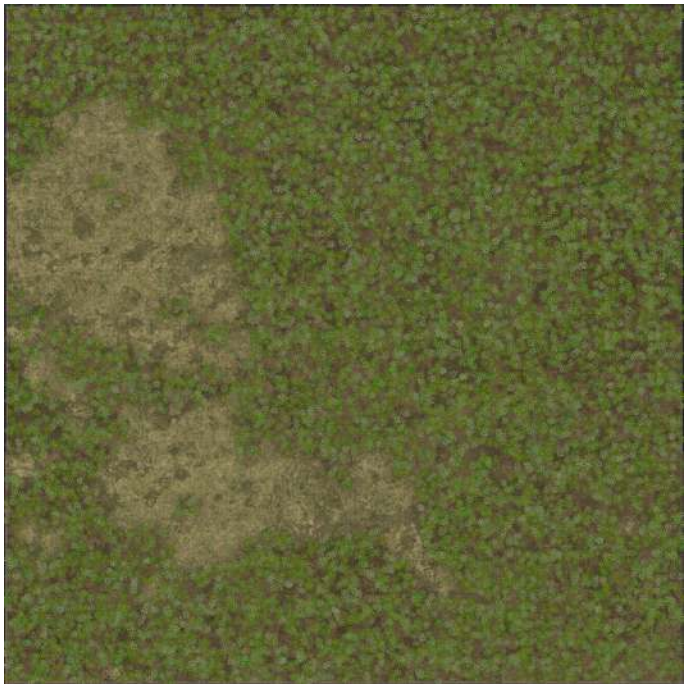


Figura A.23: Renderização do exemplo 1.

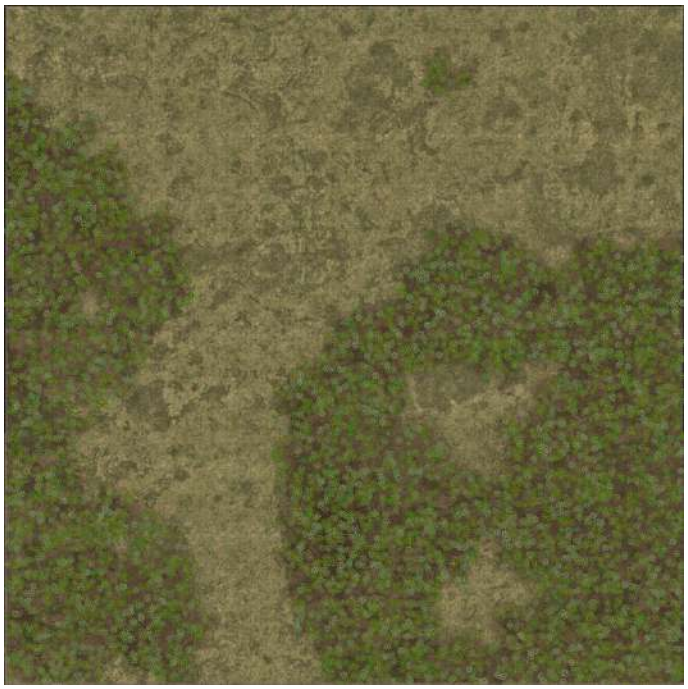


Figura A.24: Renderização do exemplo 2.

Figura A.25: Renderização do exemplo 3.

Figura A.26: Renderização do exemplo 4.

Figura A.27: Ortofoto do exemplo 1.

Figura A.28: Ortofoto do exemplo 2.

Figura A.29: Ortofoto do exemplo 3.

Figura A.30: Ortofoto do exemplo 4.