Tile-based Maps Optimization for OM Virtual Environments

Bernardo Costa
Nicolas Nery
Pedro Henrique Gonzalez
bernardo.costa@aluno.cefet-rj.br
nicolas.nery@aluno.cefet-rj.br
pegonzalez@eic.cefet-rj.br
Cefet/RJ

Windson Viana
windson@virtual.ufc.br
Universidade Federal do Ceará

Joel dos Santos
joel.santos@cefet-rj.br
GPMM - Cefet/RJ

Abstract

ENA is a mobile application for people with visual impairments who want to improve their Orientation and Mobility (OM) skills. It allows users to load personalized virtual maps of physical spaces, such as school buildings or mazes specially designed for OM training. The maps are created using a tile-based approach and could contain eight layers of tile matrices representing a floor, wall, or other objects. The map elements are enriched with 3D audio clues allowing users who are blind to navigate and accomplish OM tasks. In terms of loading large 3D environments, ENA had some inefficiencies. This performance issue is caused by the tool rendering method. It creates an object for every tile, even if there are contiguous areas of walls or floors made from the same material. We have developed two optimization algorithms integrated into ENA to address this issue. The first algorithm works on straight lines, while the second focuses on two-dimensional regions. These algorithms effectively reduce the number of objects created, resulting in a much faster and more efficient ENA tool.

Keywords: Orientation and Mobility, Virtual Environments, Map Optimization

1 Introdução

A Orientação e Mobilidade (OM), para além de habilidades fundamentais ao exercício das atividades do dia a dia, é uma importante área de estudo voltada à reabilitação e ganho de independência para pessoas com deficiência visual (PDVs) [7]. Visando facilitar o estudo de OM, trabalhos publicados na literatura apresentam ambientes virtuais enriquecidos com elementos auditivos.

Dentre tais ferramentas, destaca-se o *Espaço Navegável* (ENA) [5], uma aplicação interativa para dispositivos móveis capaz de carregar mapas personalizados. Tais mapas são criados por docentes de OM especificamente para cada aluno propondo uma tarefa de encontrar uma sequência de objetos dispostos no ambiente. Uma vez terminada a atividade, a ferramenta provê relatórios para o docente que visam indicar o desempenho do aluno de OM - por exemplo, o caminho percorrido, número de colisões, etc.

De maneira complementar ao ENA, Façanha et al. [5] apresentam um editor de mapas, o *Enhanced Environments for End Users Who Are Blind* (E3). Trata-se de uma ferramenta *web* de criação de mapas baseada em *tiles*. Nele, o usuário (e.g., um professor de OM) pode desenhar, em uma de diversas camadas, áreas contendo paredes ou pisos — além de diversos tipos de objetos, que podem representar obstáculos como móveis, elementos interativos, portas e janelas, ou objetivos a serem alcançados.

Um mapa criado com o editor E3 é exportado, seja para armazenamento ou distribuição para os alunos, num formato XML [5]. Nele, os elementos que compõem um mapa são divididos em oito camadas, cada uma definindo uma matriz de identificadores que indicam os tipos de piso, parede, etc. Na construção do ambiente a partir de um mapa, um objeto é criado para cada *tile* e cada camada. Esta abordagem de representação de um mapa e sua reconstrução, apesar de simples, tende a gerar representações muito grandes conforme o tamanho dos mapas aumenta. Isso resulta na geração de um ambiente 3D pouco eficiente, tornando impraticável a criação de ambientes virtuais para grandes ambientes (e.g., andares de uma escola, terminal de ônibus) que podem ser úteis para PDVs na aprendizagem sobre esses lugares.

Este trabalho, portanto, tem como objetivo criar uma representação mais eficiente para mapas gerados no Editor E3 e a serem carregados pela ferramenta ENA. Junto desse formato mais eficiente de representação, este trabalho também propõe o uso de algoritmos de otimização para reduzir o número de elementos necessários à representação completa dos mapas gerados pelo editor E3. Com isso, busca-se otimizar tanto a representação do mapa, mas também o número de objetos criados no ambiente 3D da ferramenta ENA.

In: III Concurso de Trabalhos de Iniciação Científica (CTIC 2023), Ribeirão Preto, Brasil. Anais Estendidos do Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia). Porto Alegre: Sociedade Brasileira de Computação, 2023.

ISSN 2596-1683

2 Mapas em E3 e ENA

Como apresentado na Introdução, mapas criados com o editor E3 são exportados em um formato XML proposto em [6]. Há oito camadas disponíveis em cada mapa: seis camadas de objetos (portas e janelas, móveis, eletrodomésticos, utensílios, elementos interativos e personagem), além de pisos e paredes. A Figura 1 ilustra essa representação. Cada camada é armazenada como uma matriz de *IDs* separados por vírgulas. Cada identificador representa o tipo de parede, piso ou outro elemento em um *tile*, de acordo com sua camada.

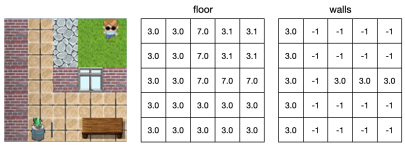

Figure 1. Matriz de representação dos *tiles* em uma camada.

Por ser um editor baseado em *tiles*, o E3 exporta mapas contendo uma matriz de células para cada camada disponível. Isso significa que, num mapa de tamanho $m \times n$, são necessários $8 \times m \times n$ *IDs* de elementos para armazenar todo o conteúdo do mapa. Cabe ressaltar que cada *tile* representa um espaço de $1m \times 1m$ no ambiente virtual.

A partir de um arquivo XML gerado pelo editor E3, a ferramenta ENA gera um ambiente 3D. Para isso, a ferramenta percorre a matriz de *IDs* de cada camada, criando um objeto para cada *ID* não vazio. Esse processo gera uma representação um para um entre E3 e ENA. A Figura 2 apresenta o ambiente criado a partir do mapa da Figura 1.

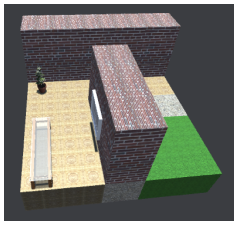

Figure 2. Ambiente criado a partir do mapa.

Como é possível observar na Figura 2, a escolha dessa representação acaba sendo pouco eficiente para a geração de grandes mapas no ambiente 3D da ferramenta ENA, dado que um objeto é criado para cada *tile*, mesmo em áreas contíguas de paredes ou pisos do mesmo material. Este problema é mais aparente quando são criados mapas grandes, como o apresentado na Figura 3.

3 Trabalhos Relacionados

Ao revisar a literatura, encontrou-se pesquisas relevantes que podem ser usadas para reduzir efetivamente o número de objetos. O estudo de Chaiken et al. [2] abordou o problema de

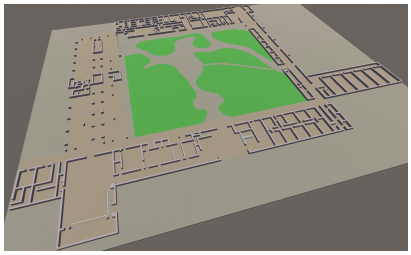

Figure 3. Mapa de ambiente grande.

cobrir regiões com retângulos, demonstrando a relação entre o número mínimo de retângulos necessários e a cardinalidade máxima de um antirretângulo. O método proposto por Cheng et al. [3] para compressão de imagem usando coberturas irredutíveis de retângulos máximos pode ser uma inspiração para otimizar o carregamento de ambientes 3D no ENA.

O estudo conduzido por Scheithauer et al. [9], que utiliza modelos de Programação Linear Inteira (PLI) para o problema de cobrir uma região poligonal com retângulos, também pode ser relevante para o aprimoramento do ENA. Em um contexto mais recente, Mansouri et al. [8] abordaram o problema de cobrir regiões com retângulos no contexto de inspeção aérea com Veículos Aéreos Não Tripulados (VANTs) equipados com sensores visuais remotos. Embora o contexto seja diferente do ENA, a abordagem de otimização utilizada pode oferecer insights valiosos para a melhoria da eficiência do aplicativo ao lidar com grandes ambientes 3D.

Em conclusão, os artigos revisados fornecem abordagens e técnicas que futuramente podem ser aplicadas para otimizar o carregamento de mapas virtuais no aplicativo ENA. A utilização futura de tais abordagens pode contribuir para reduzir o número de objetos criados e melhorar o desempenho geral do aplicativo, permitindo uma experiência mais eficiente e suave para os usuários com deficiência visual.

4 Otimização dos Mapas

A abordagem de otimização **desenvolvida pelos alunos de iniciação científica** busca gerar um formato intermediário mais apropriado à finalidade de gerar um mapa para um ambiente tridimensional, reduzindo tanto o número de objetos como a quantidade de informações armazenadas. Para isso, utiliza-se algoritmos de otimização para reduzir o número de objetos necessários para cobrir toda a área desenhada. Por terem finalidades diferentes, cada camada utiliza um algoritmo distinto. A seguir, são descritos os procedimentos utilizados para a otimização de camada.

4.1 Camada de Parede

Por ser composta majoritariamente de linhas retas, na camada de parede utiliza-se um algoritmo de busca em linha. Esse algoritmo encontra a maior área possível de células em

linha reta com o mesmo material, sendo que todas as paredes geradas têm largura 1.

O processo, para cada material de parede, inicia do canto superior esquerdo do mapa e percorre cada linha da matriz armazenando o comprimento de cada linha sequencial de paredes do mesmo tipo. Em seguida, processo semelhante é feito, mas percorrendo o mapa coluna a coluna.

Após os dois passos anteriores, obtém-se duas listas de linhas de paredes (horizontais e verticais) de um mesmo tipo. Estas listas são combinadas e ordenadas por comprimento, do maior para o menor.

O passo seguinte é percorrer a lista de paredes, buscando a maior parede que tenha pelo menos uma célula não coberta por outra parede. Encontrada, é criado um objeto de parede, com posição e comprimento correspondentes. Esse passo é repetido enquanto existirem células de parede descobertas. A Figura 4 apresenta o resultado obtido para um trecho do mapa da Figura 3.

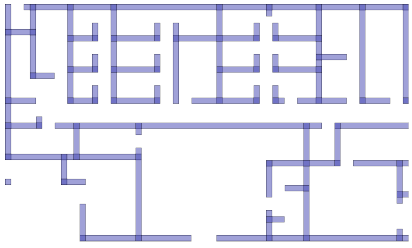

Figure 4. Paredes geradas após o algoritmo.

4.2 Camada de Piso

Aqui, busca-se cobrir a área de piso de um mesmo tipo com o menor número de retângulos. Para melhor resultado, é ideal que se inicie a busca do ponto mais distante em relação à borda do mapa, pois a escolha de um ponto arbitrário poderia resultar em um arranjo menos eficiente. Para isso, utiliza-se um algoritmo de detecção de bordas, fornecido pela biblioteca de manipulação de imagens *Pillow* [4], sucessivamente. Com isso, o mapa "encolhe" até que não haja mais bordas. Nesse momento, utiliza-se o ponto encontrado como ponto de início da geração de piso. A Figura 5 apresenta o resultado desse algoritmo para um trecho do mapa da Figura 3.

Em seguida, utiliza-se um algoritmo de busca bidimensional para encontrar a maior área possível de células contíguas com o mesmo tipo. Partindo do ponto com a maior distância em relação à borda, expande-se o retângulo para cima, direita, baixo e esquerda, verificando se as células adjacentes ao piso em cada direção são do mesmo tipo. Em caso afirmativo, expande-se o retângulo para a direção correspondente. Repete-se esse processo até que não seja possível expandir o retângulo. Então, cria-se um objeto de piso com a posição e dimensões do retângulo, removendo do mapa as células cobertas pelo objeto de piso inserido. Repete-se todo

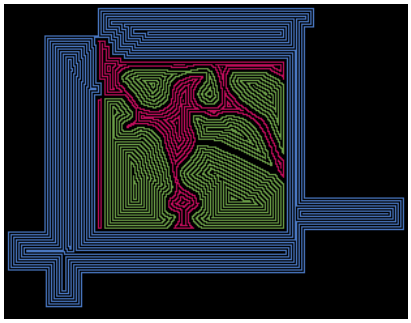

Figure 5. Resultado da detecção de bordas.

o processo com o mapa atualizado até que não haja mais células de piso restantes. A Figura 6 apresenta o resultado obtido para um trecho do mapa da Figura 3.

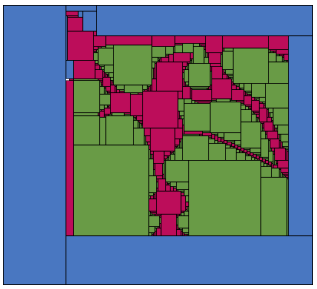

Figure 6. Piso gerado após o algoritmo.

4.3 Demais Camadas

Diferente das camadas de parede e piso, as seis camadas de objetos seguintes indicam a posição de objetos específicos, como portas, janelas, móveis, etc. A posição desses objetos é dada por *IDs* que os representam. Assim, as matrizes dessas camadas acabam sendo matrizes esparsas. O processo de otimização, portanto, cria um objeto JSON com o tipo e a posição que este ocupa no mapa.

4.4 Formato de dados

O mapa otimizado é exportado em um arquivo no formato JSON. Esse formato foi escolhido por ser um formato de dados baseado em texto, leve e amplamente utilizado, e fácil de ler e escrever para humanos e máquinas [1]. Além disso, o formato JSON tem a vantagem de reduzir a redundância de informações em relação ao formato XML, devido à ausência da estrutura de marcações aninhadas.

```
{
  "size": [<width>, <height>],
  "layers": [
    "walls": [
      {
        "type": <wallID>,
        "start": [<x>,<y>], "end": [<x>,<y>]
      }, ...
    ],
    "floors": [
```

```
      {
        "type": <floorID>,
        "start": [<x>,<y>], "end": [<x>,<y>]
      }, ...
    ],
    "door_and_windows": [
      {
        "type": <objID>, "pos": [<x>,<y>]
      }, ...
    ],
    "furniture": [<obj>, ...],
    "utensils": [<obj>, ...],
    "eletronics": [<obj>, ...],
    "goals": [<obj>, ...],
    "persons": [<obj>, ...]
  ]
}
```

Listing 1. Arquivo de representação do Mapa.

Conforme observado na Listagem 1, a representação do mapa é feita em um objeto JSON como atributos que indicam seu tamanho e camadas do mapa. Para as camadas de parede e piso são armazenados o tipo do material da parede, ou piso, e seu ponto de início e fim. Já para as demais camadas são armazenados o tipo do objeto e sua posição.

4.5 Ganhos com a otimização

Para exemplificar o uso da abordagem, foi feita a otimização do mapa do piso inferior do prédio do CEFET-RJ ilustrado na Figura 3. Seu tamanho é de 310 × 240, resultando em 74.400 células em cada camada, com um total de 3.482 objetos sendo construídos para paredes, e 44.392 para pisos. Após a otimização, o mapa possui 380 objetos de parede e 565 objetos de piso, uma redução de 89.09% e 98.73% no número de paredes e pisos, respectivamente. Isso representa uma redução geral de 98% no número de objetos necessários para a representação completa do mapa no aplicativo móvel.

Cabe destacar que, em pisos ou paredes com padrões complexos, a otimização alcançada não é tão expressiva, tendo em vista que muitos objetos de tamanho 1 são gerados. Na Figura 6, a área representada na cor verde é um exemplo desse fenômeno. Mesmo assim, foi possível reduzir o número de pisos deste material de 13.002 para 296 — uma redução de 97.9%. Já em áreas que se aproximam mais de um formato retangular, como o material representado na cor azul na Figura 6, a redução é ainda maior. Neste caso, passou-se de 25.442 objetos de piso para apenas 14 pisos — reduzindo em 99.9% o número de pisos deste material.

Em relação ao tamanho do arquivo, o arquivo gerado pelo editor E3 tem 1.83 MB. Já o arquivo gerado após a otimização tem 44.97 KB, uma redução de 97.56% no volume de dados a serem armazenados e transmitidos.

5 Considerações Finais

O treinamento de OM é importante para reabilitação e ganho de independência para PDVs. Trabalhos publicados na literatura apresentam ambientes virtuais enriquecidos com elementos de áudio como uma tecnologia assistiva para aprendizagem de OM. Uma dessas ferramentas é a solução ENA/E3, um ambiente virtual para OM que oferecem mapas multimodais criados por professores de OM no formato de *tiles*. Apesar de fornecer uma forma de empoderamento dos professores de OM, a estratégia de *tiles* demanda na renderização um número muito grande de objetos, que pode tornar ineficiente a aplicação móvel no caso de grandes mapas.

Este trabalho, portanto, apresentou uma forma de otimizar os mapas criados para construção de ambientes virtuais com a ferramenta ENA. Foram utilizados algoritmos de busca em linha, detecção de bordas e busca bidimensional da maior área possível. A aplicação destes algoritmos propicia a criação de mapas com uma menor quantidade de objetos, quando comparado com o formato original. Os testes iniciais com o mapa do CEFET-RJ mostraram reduções de mais de 89% em vários tipos de objetos contidos nos mapas.

Apesar dos bons resultados apresentados, a otimização de pisos fica prejudicada quando o autor do mapa deixa partes do piso em branco (abaixo de paredes, por exemplo). Um trabalho futuro no sentido de otimizar ainda mais os mapas construídos é completar pisos em branco com o mesmo tipo dos pisos ao redor, conseguindo assim preencher áreas maiores e consequentemente, usar menos objetos para a representação do piso.

Acknowledgments

Os autores agradecem à CAPES e ao CNPq por disponibilizarem os recursos necessários à viabilização deste trabalho.

References

[1] 2023. The JSON data interchange syntax. https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.

[2] Seth Chaiken, Daniel J Kleitman, Michael Saks, and James Shearer. 1981. Covering regions by rectangles. *SIAM Journal on Algebraic Discrete Methods* 2, 4 (1981), 394–410.

[3] Ying Cheng, SS Iyengara, and Rangasami L. Kashyap. 1988. A new method of image compression using irreducible covers of maximal rectangles. *IEEE transactions on software engineering* 14, 5 (1988), 651–658.

[4] Alex Clark. 2015. Pillow (PIL Fork) Documentation. https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[5] Agebson Rocha Façanha, Ticianne Darin, Windson Viana, and Jaime Sánchez. 2020. O&M indoor virtual environments for people who are blind: A systematic literature review. *ACM Transactions on Accessible Computing (TACCESS)* 13, 2 (2020), 1–42.

[6] Agebson Rocha Façanha. 2021. *Customização de ambientes virtuais de orientação e mobilidade para pessoas com deficiência visual.* Ph. D. Dissertation. Universidade Federal do Ceará.

[7] Orly Lahav. 2022. Virtual Reality Systems as an Orientation Aid for People Who Are Blind to Acquire New Spatial Information. *Sensors* 22, 4 (2022). https://doi.org/10.3390/s22041307

[8] Sina Sharif Mansouri, George Georgoulas, Thomas Gustafsson, and George Nikolakopoulos. 2017. On the covering of a polygonal region with fixed size rectangles with an application towards aerial inspection. In *2017 25th Mediterranean Conference on Control and Automation (MED).* IEEE, 1219–1224.

[9] G Scheithauer, Yu Stoyan, and T Romanova. 2009. Integer linear programming models for the problem of covering a polygonal region by rectangles. 2 (2009), 4–13.