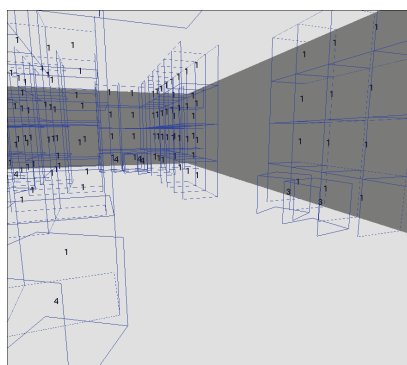


Bricking: Modelagem Tridimensional de Cenários de Jogos em Camadas

Raul Joaquim C. dos Santos
rauljcs@gmail.com

Selan Rodrigues dos Santos
selan@dimap.ufrn.br

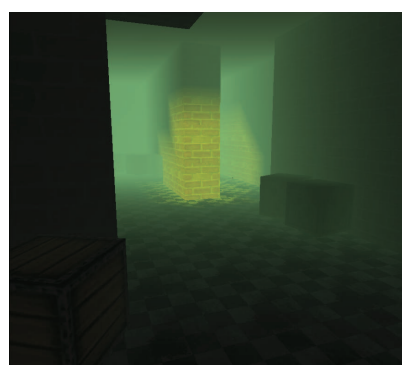
Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte
Campus Lagoa Nova, 59072-970, Natal/RN



(a)



(b)



(c)

Figura 1: Screenshot do jogo Pathfinder, desenvolvido com a modelagem por *bricking*. Em (a) temos a identificação dos *bricks* que compõem a cena. Em (b) temos a representação correspondente texturizada dos *bricks*. Finalmente em (c) é possível observar o efeito final, com iluminação e neblina.

Resumo

In this paper we introduce a practical and simple method for three-dimensional (3D) game level design, based on an extension of the tile modelling concept. The method, called bricking, preserves the main features of tiling, such as simplicity and low memory requirements. The main advantages of this proposal is a greater power of modelling since it is possible to define more than one layer of tiles, thus allowing complex structures to be subdivided and treated as individual units, called bricks. Differently from what happens in typical pseudo-3D games based on tile modelling, the players' movements are not restricted to a single plane — thereby allowing players to have a true 3D experience. To test our proposal, we present a first-person style game whose levels were modelled with bricking, and discuss the results from this experience.

Este artigo apresenta um método prático e simples de construir modelos de cenários de jogos para ambientes tridimensionais, baseado em uma extensão do conceito de modelagem através de *tiles*. O método proposto, denominado de *bricking*, mantém as principais características do método *tiling*, como simplicidade de manipulação e economia de memória, com o benefício de uma maior poder de representação e deslocamento não mais limitado a um único plano. Como forma de validação parcial da proposta, apresentamos um jogo modelado com a técnica *bricking* e analisamos esta experiência.

Keywords: Modelagem 3D, cenários de jogos, *tiles*, detecção de colisão.

1 Introdução

Atualmente existem várias metodologias para a criação de jogos, que dividem o processo de desenvolvimento em várias etapas. Uma das etapas da criação de um jogo é a de modelagem ou representação do cenário. Um dos objetivos da modelagem de cenários é dar suporte à história do jogo eletrônico e prender a atenção dos usuários, desta forma promovendo um maior engajamento. Além de prover a ambientação para o desenrolar do jogo, a modelagem de cenários pode auxiliar testes de colisão e determinação de visibilidade para otimiza-

ção do *pipeline* de renderização [Akenine-Möller et al. 2008, Cap. 14].

A construção de cenários pode ser uma tarefa complexa, mas existe uma diversidade de métodos que auxiliam este processo, diferindo em complexidade de implementação, leque de operações suportadas e demanda de processamento [Eberly 2006]. *Tiling* é um exemplo de técnica de representação de cenário, cujas maiores qualidades são simplicidade de representação, uso otimizado de memória e boa velocidade de renderização [LaMothe 2001]. Esta técnica, utilizada em jogos mais antigos, possibilita a criação de cenários dividindo a imagem da cena em uma grade regular — os elementos repetidos da grade são carregados apenas uma única vez e replicados nos demais locais.

Apesar de originalmente bidimensional (2D), a técnica *tiling* foi adaptada com sucesso para a tridimensionalidade (3D), como no caso do clássico título Wolfenstein 3D [id Software 2009b], dando origem ao conceito de representação 2.5D ou *pseudo-3D*. Neste caso, o mapeamento continua 2D, mas cada *tile* é representado por uma malha 3D ou poliedro, possivelmente texturizado.

Apesar de ser capaz de simular um ambiente tridimensional, o conceito de 2.5D possui uma limitação referente ao tratamento de colisão: cada *tile* do ambiente contém um único objeto, o qual é tratado de maneira atômica. Desta maneira, se um determinado *tile* é definido como “sólido” não é possível transpô-lo, mesmo que sua representação gráfica seja a de uma janela aberta ou um obstáculo baixo, como um caixa. Portanto, o problema da utilização de *tiles* para modelar cenários 3D reside no fato de que, se o *tile* for composto por várias partes, não é possível tratar cada uma delas independentemente.

Para resolver o problema acima citado, propomos uma extensão da modelagem por *tiles*, denominada *bricking*, que preserva as características originais de simplicidade de representação e uso reduzido de memória pra recursos gráficos. *Bricking* é organizado em N camadas de *tiles*, com suporte para a operação de tratamento de colisão e é capaz de oferecer uma representação verdadeiramente 3D, superando as limitações das representações 2.5D. A Figura 1 apresenta *screenshots* de um jogo desenvolvido com esta nova técnica.

2 Conceitos Relacionados

Nesta seção descrevemos em mais detalhes alguns conceitos relacionados à proposta ora apresentada, como a técnica de *tiling* e detecção de colisão.

Representação de cenário

O objetivo de *tiling* é a criação de cenários de forma simples e com economia de memória. Um cenário modelado por *tiles* é formado por um conjunto de pequenas imagens que juntas geram a imagem desejada da cena [Pa 2009]. Em linhas gerais, divide-se uma imagem correspondente ao cenário em partes menores, como pequenos azulejos; depois todas as figuras distintas que foram formadas são separadas. Em seguida, rotula-se cada figura diferente, posicionando-as de maneira a reconstruir a imagem original.

A economia de memória e processamento vem do fato de que não é mais necessário carregar de uma vez toda a imagem de fundo do cenário. Basta carregar apenas as imagens (menores) correspondentes aos *tiles* distintos entre si e replicá-los onde necessário. A Figura 2 exemplifica como acontece a divisão e a (re)utilização de *tiles*.

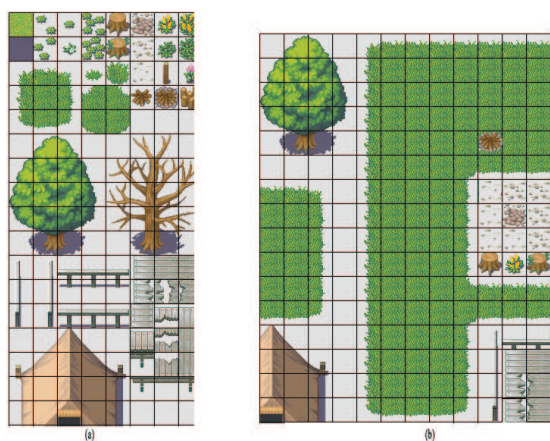


Figura 2: Em (a) temos a representação do mapa de *tiles* associado a um cenário. A imagem (b) exibe a criação de um cenário utilizando as imagens do mapa de *tiles*.

Tratamento de colisão

Com a utilização de uma grade regular para definir os *tiles*, é possível determinar a posição individual de cada *tile*. Na modelagem de um cenário de jogo é importante definir que determinados tipos de *tile* não podem ser penetrados por objetos que podem se movimentar dentro do ambiente. Portanto, a detecção da colisão ocorre quando a área ocupada por um objeto intersecta os limites da região do *tile* (ou conjunto de *tiles*) definido como ‘intransitável’ e que, portanto, não pode ser ocupada.

Esse mesmo mecanismo simples de detecção de colisão é usado em jogos 2.5D. A representação em 2.5D ou pseudo-3D combina uma representação 2D ou mapa de *tiles* com elementos renderizados em 3D. Exemplos de variações desta ideia são: *i*) personagens 2D atuando em frente a um cenário de fundo em 3D, como é o caso do *Street Fighter IV* [Capcom 2009]; *ii*) personagens poligonais 3D sobre uma plano de fundo bidimensional, como é o caso de *Ragnarok Online* [Gravity Co. and Myoungjin 2005]; ou *iii*) um jogo com representação completa em 3D (via *sprites* e polígonos texturizados), mas com deslocamento restrito a um único plano, como é o caso de títulos como *DOOM* da Id Software [id software 2009a] e *Duke Nukem 3D* da 3D Realms [Realms 2009].

Apesar do cenário ser renderizado em três dimensões, o controle de colisão realiza apenas a fase de descarte ou *broad phase* [Ericson 2005] entre a posição atual da câmera ou

personagem e o *tile* em contato. Se o *tile* em questão for ‘transitável’ o deslocamento prossegue normalmente; caso contrário é indicada a colisão, independentemente da representação 3D associada ao *tile*.

A estratégia de tratamento de colisão acima descrita pode gerar situações inconsistentes com a expectativa dos jogadores e a própria realidade representada no jogo eletrônico. Por exemplo, um jogador normalmente espera ser capaz de passar por cima de um obstáculo baixo como uma caixa ou passar pela abertura de uma grande janela, ao se dirigir a tais obstáculos. Contudo, se o *tile* que contém o obstáculo fosse definido como ‘intransitável’ esse comportamento não seria possível, uma vez que não há tratamento de colisão por refinamento (ou *narrow phase*) dentro do *tile*. Desta maneira, o jogador não conseguiria transpor o obstáculo, mesmo que visualmente isso parecesse ser possível.

A motivação para a técnica de representação de cenário proposta neste trabalho é justamente solucionar este problema e ainda conservar as vantagens da representação de cenários por *tiling*. Pode-se afirmar que os requisitos que nortearam o design da solução proposta foram os seguintes:

- Suportar a colisão a nível de descarte;
- Utilizar algoritmos de manipulação simples;
- Prover um método com economia de memória, uma vez que os componentes seria carregados uma única vez e reutilizados quando necessários;
- Oferecer uma estrutura de dados de simples representação e manipulação; e
- Proporcionar uma maior capacidade de representação de objetos do que o método tradicional de pseudo-3D, uma vez que não estaríamos mais restritos a um único plano de navegação e representação de objetos.

3 Modelagem Bricking

O modelo de representação de cenário por *bricking* tem como objetivo estender a técnica de representação por mapa de *tiles* de maneira a suportar mais de um plano de representação. A região de modelagem associada com o cenário é organizada em N camadas, de acordo com a necessidade de representatividade para elementos do cenário. As camadas seguem o conceito de *tiling* e são organizadas em grades regulares independentes umas das outras. Uma vez definidas, as camadas são dispostas em alturas diferentes formando uma pilha de camadas sobrepostas. Objetos podem ser inseridos em qualquer coordenada, (x,y) , em uma dada camada i , definindo desta maneira um *brick*, $B(x,y),i$.

A estrutura de dados básica usada para representação por *bricking* é uma matriz tridimensional. A primeira dimensão da matriz indica a quantidade de camadas; as outras duas dimensões indicam as coordenadas Cartesianas (x,y) de um *brick* dentro do plano definido por uma camada. Por exemplo, uma matriz $3 \times 4 \times 5$ terá três camadas, e cada uma delas formada por um plano que contém *bricks* distribuídos em uma matriz 4×5 , conforme ilustrado na Figura 3.

Assim, em uma matriz $M[c][i][j]$ o valor c corresponde à camada do cenário, os valores de i e j correspondem às coordenadas do objeto contido na camada c , em um ambiente tridimensional. Além da matriz de representação, é necessário definir uma lista L de conjuntos de coordenadas de *bricks* que correspondem a um mesmo identificador. O objetivo desta lista é otimizar o processo de renderização, pois ao invés de percorrer a matriz de maneira sequencial trocando de contexto cada vez que um *brick* diferente fosse acessado, renderiza-se todos os *bricks* que compartilham o mesmo contexto (i.e. textura, geometria, etc.), minimizando *overhead* associado com tal troca.

A técnica proporciona a criação de um modelo de cenário com os objetos da cena identificados por *bricks*. Tendo isto em vista, a modelagem do cenário deve ser feita determinando as dimensões de um *brick* de forma a conter o maior

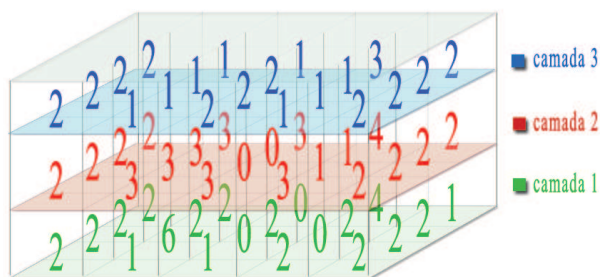


Figura 3: Representação esquemática de uma matriz correspondente a um cenário com 3 camadas, organizadas em uma grade de 4×5 bricks. Os números que aparecem dentro de cada brick são seus identificadores, ou seja, brick com o mesmo identificador significam objetos iguais.

objeto a ser representado no modelo, para que nenhum outro objeto do modelo ultrapasse as dimensões do brick. A operação de detecção de colisão da técnica é feita apenas em *broad phase*. Desta forma, o teste de colisão será feito diretamente com os bricks identificados como ‘intransitáveis’ na cena. A detecção de colisão em *narrow phase*, feita diretamente com o objeto contido no brick, ainda não é suportado pela técnica.

Economia de memória

Em relação ao custo de memória, a técnica mantém as características de *tiles*: os bricks que possuem a mesma identificação representam os mesmos objetos renderizados nas suas coordenadas correspondentes. Como os objetos são os mesmos, as texturas que forem aplicadas nestes objetos também serão as mesmas e, desta forma, só necessitarão ser carregadas uma única vez e aplicadas em todos os objetos de mesma identificação.

A modelagem de cenários por *bricking* poderá ser utilizada para subdividir partes maiores de um cenário. Por exemplo, paredes de um cenário podem ser divididas em vários ‘tijolos’ menores. Desta maneira, basta carregar apenas as imagens correspondentes às partes contidas nos bricks distintos que formam a parede e replicá-los onde for necessário.

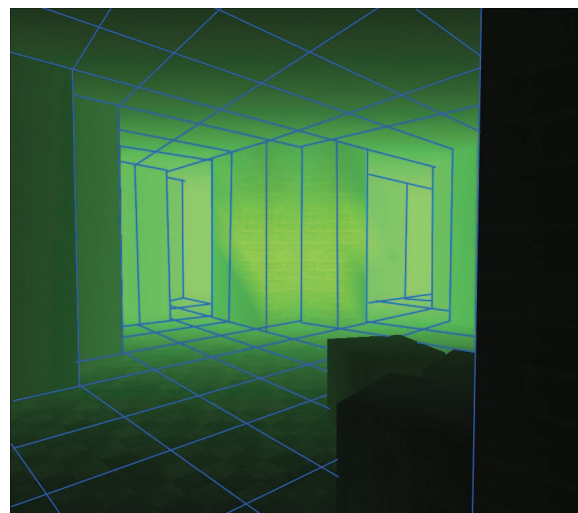
Poder de representação

Essa organização permite uma maior capacidade de representação. Por exemplo, a Figura 4-(a) apresenta uma imagem de um cenário modelado por *bricking*, porém utilizando apenas 1 camada — isso seria equivalente ao modelagem por *tiles*. Na imagem é possível notar que existe uma estrutura (em segundo plano, no centro da imagem) correspondente a uma coluna. Na Figura 4-(b), temos o mesmo cenário, desta vez organizado em 4 camadas de mesma altura. A antiga estrutura de coluna agora é representada com apenas 2 bricks, uma na camada superior e outro na inferior. Desta forma, se o personagem possuir uma altura de apenas 2 bricks, seria possível passar por “dentro” da coluna subindo no brick inferior. Note que esta ação não seria possível com a modelagem por *tiles*, pois a coluna ocuparia um único *tile* e seria definida como ‘intransitável’, mesmo que sua representação visual fosse a exibida na Figura 4-(b).

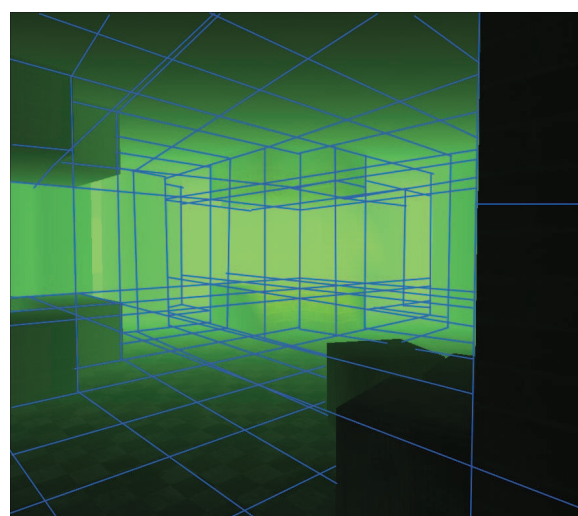
4 Resultados

Para testar o funcionamento da técnica de *bricking*, um jogo de navegação em primeira pessoa foi desenvolvido, denominado de Pathfinder [dos Santos 2009]. O jogo foi desenvolvido na linguagem C++, utilizando a biblioteca GLUT [Martz 2006].

A modelagem do cenário de jogo foi realizada com *bricking*,



(a)



(b)

Figura 4: Exemplo de um cenário modelado por *bricking*, sendo constituído de 4 camadas de mesma altura. Linhas foram sobrepostas à imagem para demonstrar a localização das camadas e suas subdivisões em bricks.

utilizando-se um editor de texto simples para a definição da matriz de componentes e seus recursos. A lista de conjunto de bricks idênticos também foi gerada manualmente. O cenário contém objetos como caixas, muros e paredes texturizados, colocados em diferentes posições no ambiente. Apresenta efeitos de neblina e de iluminação, de maneira a dar suporte ao enredo do jogo.

4.1 O Jogo

Em *Pathfinding* o jogador deve percorrer o cenário para coletar itens, em uma corrida contra o tempo. O cenário é relativamente complexo, contendo vários corredores, aberturas, obstáculos, colunas e entradas na forma de portas, janelas, e túneis — esta última modalidade requer que o usuário se abaixe e rasteje para atravessar.

Além das vias de acesso, o cenário possui diversas caixas, espalhadas e empilhadas, todas modeladas como um único brick. Especificamente, foram utilizadas 4 camadas de mesma altura para modelar o cenário.

4.2 Tratamento de colisão

O tratamento de colisão foi implementado realizando um teste para verificar se o jogador está tentando se mover para

uma região que foi determinada como ‘intransitável’. Em caso afirmativo, o jogador é impedido de entrar nesta região. Caso contrário, o jogador poderá transitar livremente sobre esta área.

O protagonista possui largura inferior a um *brick* e altura inferior a 2 *bricks*. Portanto é possível passar por qualquer área livre igual ou maior do que 1×2 *bricks*, inclusive buracos nas paredes, plataformas ou janelas. A Figura 5 apresenta um *screenshot* do jogo.

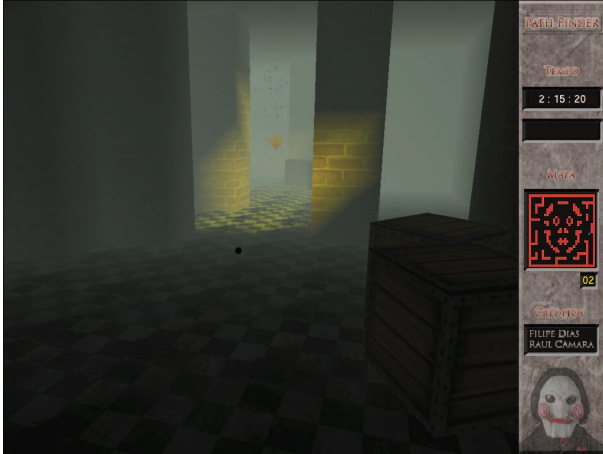


Figura 5: Screenshot do Pathfinder, exibindo o cenário modelado por *bricks*, e a interface com usuário (lado direito), que exibe o mapa do ambiente, a localização da chave e o tempo restante.

5 Conclusão e Trabalhos Futuros

Bricking é uma técnica de modelagem com o objetivo de prover uma forma simples de criar cenários de jogos visualmente atrativos e com baixo custo de memória. A forma de construir um modelo por *bricking* é simples, pois trata-se de uma extensão tridimensional do modelo de *tiles*, representada por uma matriz tridimensional. Além disso, jogos desenvolvidos com esta técnica serão capazes de realizar operações de detecção de colisão em alto nível (*broad phase*).

Em comparação com jogos desenvolvidos em 2.5D, especialmente os que são desenvolvidos em 2D para simular um ambiente tridimensional, nossa proposta traz benefícios em relação à detecção de colisão. O uso de *bricks* permite que a detecção de colisão seja realizada com objetos em diferentes alturas individualmente, possibilitando um comportamento mais compatível com uma tridimensionalidade real.

Contudo, a técnica ora proposta possui limitação de representação, determinada pelo tamanho dos *bricks*. Quanto maior o volume de um *brick*, menor será a precisão na detecção de colisão, caso o objeto associado ao *brick* não ocupe todo o volume.

Os próximos passos para o presente trabalho são:

- Adaptar a técnica de maneira a suportar a determinação de visibilidade. A meta é otimizar o *pipeline* de renderização, por exemplo, enviando apenas os *bricks* visíveis a partir da localização atual da câmera ou personagem.
- Definir um modelo descritivo de cena modelada por *bricking*, de maneira que seja possível desenvolver *plugins* para carregamento de cena em motores de jogos.
- Criar um editor de cenários interativo com suporte à *bricking*. Neste editor será possível definir o tamanho do modelo, escolher o número de camadas e *bricks* por camadas, definir os rótulos onde ficarão cada objeto da cena e gerar a lista representando a ordem de renderização dos objetos que compõem a cena.

Referências

- AKENINE-MÖLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering*, 3rd ed. A K Peters, Ltd., Wellesley, MA, USA.
- CAPCOM, 2009. Street Fighter IV. <http://www.capcom.com/streetfighter/>, acessado em julho 2009.
- DOS SANTOS, R. J. C., 2009. Pathfinder. <http://rauljcs.jimdo.com/arquivos-ufrn/computacao-grafica/pathfinder/>, acessado em julho 2009.
- EBERLY, D. H. 2006. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*, second ed. The Morgan Kaufmann Series in Interactive 3D Technology Game Architecture and Design. Morgan Kaufmann.
- ERICSON, C. 2005. *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3-D Technology. Morgan Kaufmann.
- GRAVITY CO., AND MYOUNGJIN, L., 2005. Ragnarok online. <http://www.ragnarokonline.com/>, acessado em julho 2009.
- ID SOFTWARE, 2009. DooM. <http://www.idsoftware.com>, acessado em maio 2009.
- ID SOFTWARE, 2009. Wolfenstein 3D. <http://www.idsoftware.com/games/wolfenstein/wolf3d/>, acessado em julho 2009.
- LAMOTHE, A. 2001. *Isometric Game Programming with DirectX 7.0*. Premier Press Game Development (Software). Muska & Lipman/Premier-Trade.
- MARTZ, P. 2006. *OpenGL(R) Distilled*, first ed. Addison-Wesley Professional.
- PA, T., 2009. Tile based games. <http://www.tonypa.pri.ee/tbw/>, acessado em julho 2009.
- REALMS, D., 2009. Duke Nukem 3D. <http://www.3drealms.com>, acessado em julho 2009.