2
Trabalhos Relacionados

2.1
Campos de Distância Tridimensionais

Um *campo de distância contínuo* é um campo escalar que especifica, para cada ponto, a distância até o ponto mais próximo de qualquer objeto no domínio. No caso 3-D, o campo de distância é uma função de $\mathbb{R}^3$ para $\mathbb{R}$, definida geralmente no espaço Euclidiano. Quando os objetos do domínio são manifolds fechados e orientáveis (*e.g.* a fronteira de sólidos), é possível definir um *campo de distância com sinal*, onde o sinal denota se um ponto está dentro ou fora do objeto. Tendo em conta a superfície ∂S de um sólido S, o campo de distância *sem sinal* de S é dado por:

$$d_{\partial S}(\mathbf{p}) = \inf_{\mathbf{x} \in \partial S} \|\mathbf{x} - \mathbf{p}\| \tag{2-1}$$

O campo de distância *com sinal* pode ser obtido com o auxílio de uma função de sinal:

$$d_S(\mathbf{p}) = sinal(\mathbf{p})\ d_{\partial S}(\mathbf{p}) \tag{2-2}$$

onde

$$sinal(\mathbf{p}) = \begin{cases} -1, & \text{se } \mathbf{p} \in S \\ 1, & \text{caso contrário.} \end{cases} \tag{2-3}$$

A convenção utilizada neste trabalho é que o campo de distância é negativo para pontos no interior do sólido, como mostrado na eq. (2-3).

Diversas propriedades podem ser derivadas a partir de um campo de distância com sinal. As superfícies dos objetos são definidas implicitamente como o conjunto de pontos onde o valor do campo de distância é zero. Para pontos na superfície, o gradiente ∇d do campo de distância é normal à superfície; e para pontos fora da superfície, o gradiente fornece a direção para o ponto mais próximo da superfície.

Uma propriedade importante de uma função de distância d é que [17]:

$$\|\nabla d\| = 1 \tag{2-4}$$

em quase todos os pontos, exceto aqueles que não têm um ponto mais próximo único (pontos do esqueleto). Nesses casos, o gradiente não é definido.

Uma *iso-superfície* do campo de distância é definida pelo conjunto de pontos que satisfazem $d(\mathbf{p}) = \tau$, onde a constante real τ é o iso-valor. Este tipo de iso-superfície equivale às *offset surfaces* de um sólido, onde a superfície natural do sólido é um caso particular ($\tau = 0$) de iso-superfície. Informações sobre a curvatura das iso-superfícies podem ser extraídas das derivadas de segunda ordem do campo de distância [14].

Funções de distância são contínuas, mas não diferenciáveis em todos os pontos. Diversos trabalhos [17] consideraram o problema de determinar onde um campo de distância é diferenciável. Os resultados mostram que, para uma superfície C^k ($k \geq 1$), o campo de distância com sinal também é C^k dentro de uma vizinhança da superfície. Se for possível tocar, por dentro e por fora, todos os pontos de uma superfície rolando-se uma bola de raio r por ambos os seus lados, então o campo de distância é diferenciável dentro de um raio r em volta da superfície (*local feature size* [7]).

Duas abordagens podem ser usadas para calcular campos de distância. A primeira envolve criar uma função que calcula, com exatidão, a distância até o ponto mais próximo de um objeto (*e.g.* uma malha de triângulos). A segunda consiste em utilizar uma *transformada de distância* (*distance transform*) para extrapolar valores de distância já conhecidos em alguns pontos (*e.g.* os pontos da superfície, cuja distância é zero). A primeira opção é mais interessante quando o custo da função exata é aceitável; no entanto, a segunda opção é usada freqüentemente quando o cálculo precisa ser rápido [29] e o erro da extrapolação não é um problema.

Para campos de distância com sinal, o cálculo do sinal pode ser um problema tão grande quanto o cálculo da distância. Muitos trabalhos já foram escritos, por exemplo, sobre o problema de determinar se um ponto está dentro ou fora de uma malha de triângulos fechada. Diversas abordagens foram propostas, como por exemplo, o lançamento de raios, a rasterização da malha, e cálculos (que nunca foram precisos) utilizando as normais dos triângulos. Apenas recentemente Bærentzen e Aanæs propuseram um método [1], baseado em pseudo-normais, que é rápido, robusto e preciso.

Computadores não podem representar campos de distância contínuos, mas apenas *estruturas discretas*. Um *campo de distância discreto* armazena amostras de distância em um número finito de pontos e, a partir dessas amostras, estima o valor do campo em qualquer ponto. A estratégia mais comum consiste em armazenar amostras de distância em uma grade de voxels, e reconstruir o campo por interpolação trilinear. Por ser muito simples, esta

técnica é freqüentemente usada nas aplicações baseadas em GPUs (*e.g.* [31]).

2.2 ADFs

Os campos de distância amostrados adaptativamente (ADFs) foram um projeto do *Mitsubishi Electric Research Laboratories* (MERL) que gerou diversos trabalhos [10, 24, 25, 8, 26, 9] pelos seus principais autores, Sarah Frisken e Ron Perry. Fora do MERL as ADFs receberam relativamente pouca atenção, exceto pelas contribuições [2] de Jacob Bærentzen, que desenvolveu durante seu PhD, paralelamente ao MERL, uma estrutura muito parecida com a ADF clássica de Frisken *et al.*

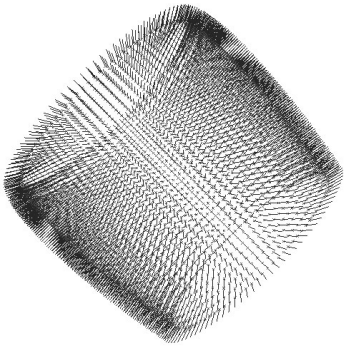
2.2.1 Trabalhos Precursores

Apesar de não ser citada em nenhum artigo sobre ADFs, uma das primeiras pesquisas relevantes sobre campos de distância adaptativos foi empreendida por Bajaj *et al.* [3], e publicada em 1995 — cinco anos antes do primeiro artigo sobre ADFs. Neste trabalho, Bajaj *et al.* propõem o uso de *octrees* para armazenar adaptativamente uma nuvem de amostras dispersas sobre um manifold. Além do campo de distância, outros campos escalares são amostrados e posteriormente reconstruídos, por interpolação triquadrática ou tricúbica. Esta técnica foi utilizada para a visualização do campo de pressão exercido sobre uma turbina (Figura 2.1), um tipo de aplicação bastante interessante para ADFs.

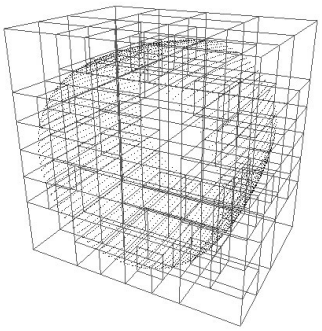
2.2.2 Definição e Classificação de ADFs

Segundo Frisken *et al.* [10], ADFs são compostas por *valores de distância amostrados adaptativamente e organizados em uma estrutura espacial, junto com um método para reconstruir o campo de distância a partir dessas amostras.* Esta definição genérica traz três características principais:

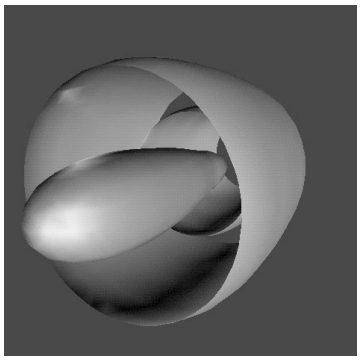
- Amostragem direcionada a detalhes. Freqüências de amostragem mais altas são utilizadas somente onde necessário para assegurar que o erro de reconstrução esteja abaixo de uma certa tolerância.
- O uso de uma estrutura espacial adaptativa (geralmente hierárquica) que permita localizar as amostras mais próximas ao redor de um ponto.
- Um método de reconstrução, adaptado à estrutura espacial, capaz de estimar o valor e o gradiente do campo de distância em um ponto.



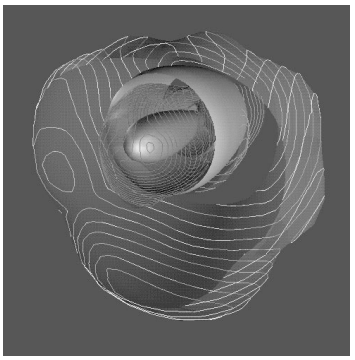
2.1(a): Pontos e normais do casco da turbina.



2.1(b): *Octree* gerada para o casco.



2.1(c): Visualização da turbina como uma superfície implícita.



2.1(d): Curvas de nível do campo de pressão junto à superfície da turbina.

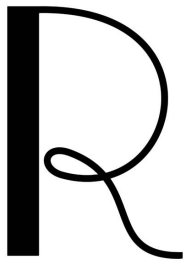
Figura 2.1: Turbina representada por uma estrutura semelhante às ADFs. Imagens extraídas dos resultados do trabalho [3].

A rigor, ADFs podem ser usadas como representações caixa-preta de funções de distância d-dimensionais arbitrárias, em qualquer espaço métrico.

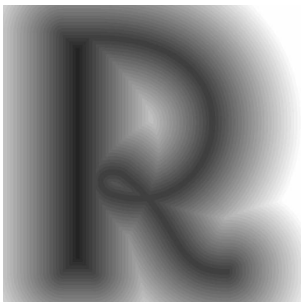
É possível imaginar diversas instanciações de ADFs utilizando uma variedade de funções de distância, estruturas espaciais e métodos de reconstrução. Algumas combinações de estrutura e método de reconstrução foram sugeridas por Frisken *et al.*:

- *Quadtree* e interpolação bilinear (ADF 2-D).
- *Octree* e interpolação trilinear (ADF 3-D).
- *Wavelets* e síntese de *wavelets* por *B-splines*.
- Malha de tetraedros com multi-resolução e interpolação baricêntrica.

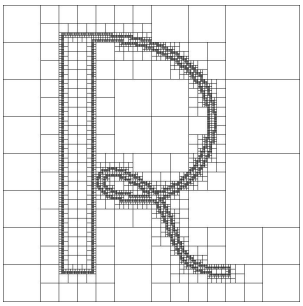
A instanciação *de facto* de ADF, utilizada em todos os trabalhos conhecidos, armazena amostras de distância nos vértices das células de uma *octree* e utiliza interpolação trilinear para reconstruir distâncias e estimar gradientes. A adaptação natural desse esquema em 2-D consiste de *quadtrees* e interpolação bilinear, como mostrado no exemplo da Figura 2.2.



2.2(a): Letra "R".



2.2(b): Campo de distância da letra "R".



2.2(c): *Quadtree* da ADF da letra "R".

Figura 2.2: Letra "R" representada por uma ADF bidimensional. Imagens extraídas do trabalho [10].

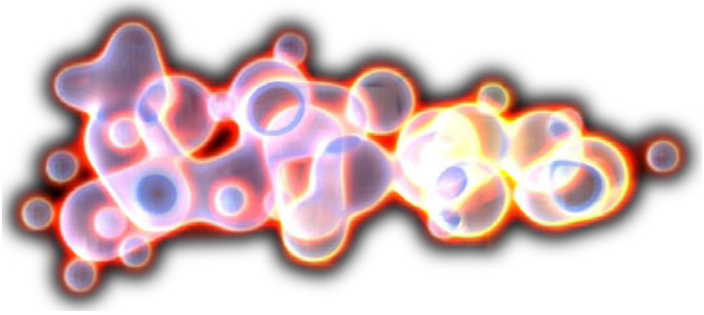
Com o objetivo de aumentar a eficiência das ADFs para aplicações que lidam apenas com a geometria dos objetos — e não precisam do campo de distância como um todo — Frisken *et al.* sugeriram [10] que somente as células que interceptam a superfície do objeto sejam armazenadas. A idéia é que o campo de distância seja bem amostrado próximo da superfície, e pouco amostrado no interior ou exterior dos objetos. Esta estratégia pode ser observada na Figura 2.2(c): a *quadtree* é bem subdividida na fronteira da forma 2-D, mas pouco subdividida em outras regiões, mesmo onde o campo de distância varia bastante. Figueiredo *et al.* sugeriram [6] chamar este tipo de ADF de "ADF de fronteira" (*boundary ADF*), para enfatizar o fato de que o campo de distância só tem boa precisão em pontos muito próximos da superfície representada. Já as ADFs que garantem a mesma precisão para todos os pontos são chamadas de "ADFs globais" (*global ADFs*).

Comparativamente, as ADFs globais têm *octrees* mais densas e ocupam mais memória, enquanto as ADFs de fronteira têm *octrees* mais esparsas, mas aplicabilidade limitada. As ADFs de fronteira são suficientes para aplicações que utilizam somente a superfície dos objetos, como por exemplo: a visualização de superfícies, detecção de colisão e operações CSG. Por outro lado, estas ADFs não são suficientes para aplicações que exijam um campo de distância preciso e completo, como por exemplo: metamorfoses, planejamento de caminhos, aplicação de hipertexturas e outros efeitos baseados no campo de distância (Figura 2.3(b)).

2.2.3 Implementação Clássica

As ADFs introduzidas por Frisken *et al.* [10] armazenam amostras de distância nos vértices das células de uma *octree* e utilizam interpolação trilinear para reconstruir distâncias e estimar gradientes. Nenhum trabalho

2.3(a): Escultura em um vaso, feita com cinzel arredondado.



2.3(b): Molécula de cocaína renderizada em uma neblina de névoa turbulenta. A névoa foi gerada usando uma função de cor baseada na distância até a superfície da molécula.

Figura 2.3: Imagens renderizadas de ADFs, retiradas do trabalho [10].

revela detalhes sobre a estrutura interna das ADFs. Com base nos trabalhos [10, 24, 2, 17], sabe-se apenas que as *octrees* são implementadas com ponteiros, e estima-se que cada célula da *octree* armazene oito amostras de distância, uma para cada vértice. Visto que um vértice é compartilhado por várias células, este esquema resulta no armazenamento redundante de amostras. A reconstrução do campo de distância e o cálculo de gradientes são sempre feitos no espaço local de uma célula, utilizando exatamente oito amostras.

Dada uma função de distância, uma ADF pode ser construída usando uma abordagem *top-down* ou *bottom-up* [10]. A abordagem *bottom-up* começa construindo uma *octree* completa, e então tenta descartar blocos de oito células irmãs, de baixo para cima, sempre que as células não forem necessárias para garantir a precisão de reconstrução desejada. A abordagem *top-down* começa com uma célula raiz e então subdivide a *octree* recursivamente, enquanto o erro de reconstrução estimado para as células-folha for maior do que o desejado. As *octrees* resultantes são *cheias* por construção, ou seja, todas as células têm sempre zero ou oito filhos.

A construção *top-down* utiliza memória de forma mais eficiente, mas requer que a função de distância seja amostrada repetidamente nos mesmos pontos (no caso de vértices compartilhados por diferentes células). Assim, se a função de distância for muito cara, a construção *top-down* será muito lenta. Por outro lado, a construção *bottom-up* evita amostragens redundantes, mas tem precisão limitada e requer muita memória. Uma *octree* completa de nível 10, por exemplo, ocupa mais de 4GB de memória. Tendo em vista estas limitações, Perry e Frisken [24] propuseram um método de construção híbrido, chamado de construção ladrilhada (*tiled*).

Na construção ladrilhada, a célula raiz é subdividida recursivamente até um nível L. As células de nível L que precisem de mais subdivisão são

adicionadas a uma lista. Então, as células da lista são subdivididas entre os níveis L e $2L$. As células de nível $2L$ que precisem de mais subdivisão são adicionadas a uma nova lista, e o algoritmo se repete. Este algoritmo funciona como uma construção *top-down*, mas sem que ocorra amostragem redundante dentro do mesmo ladrilho (*tile*). Um ladrilho é um pequeno volume, de tamanho $(2^L)^3$, usado para armazenar amostras de distância e evitar que elas sejam calculadas mais de uma vez. Segundo resultados reportados por Perry e Frisken [24], a construção feita com ladrilhos de 16^3 ($L = 4$) é aproximadamente 20 vezes mais rápida do que a construção *top-down* comum.

Para obter o máximo de precisão na reconstrução do campo de distância, é preciso sempre utilizar as amostras mais próximas ao redor de um ponto. Isto significa que a reconstrução deve ser feita nas células-folha da *octree*. No entanto, células de níveis mais altos também podem ser usadas quando um nível de detalhe mais rústico for desejável (Figura 2.4).

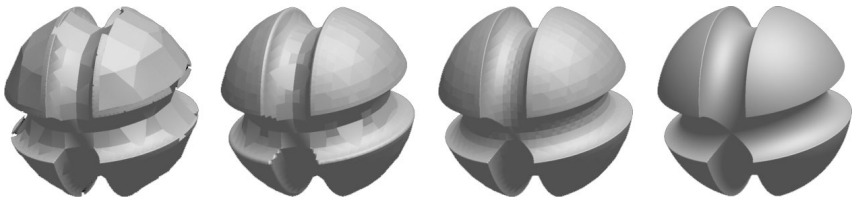


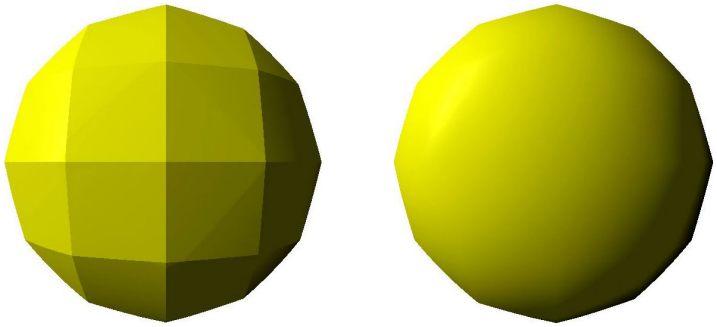
Figura 2.4: Quatro níveis de detalhe extraídos da mesma ADF. Imagens retiradas do trabalho [10].

As ADFs clássicas introduzem dois tipos de descontinuidades no campo de distância — ambos observáveis na Figura 2.4. Como em todo volume hierárquico, descontinuidades ocorrem nas faces e arestas da *octree* onde células-folha de diferentes níveis se encontram, causando pequenas rupturas na superfície dos objetos. Além disso, o uso de interpolação trilinear resulta em descontinuidades nas normais nas fronteiras das células. Estas descontinuidades fazem com que superfícies curvas fiquem facetadas, e geram artefatos de iluminação. Frisken *et al.* argumentam [10] que as rupturas causadas por descontinuidades geométricas são fáceis de contornar na maioria das aplicações, e que as descontinuidades nas normais não podem ser percebidas se as ADFs forem suficientemente subdivididas — mesmo que isto possa impor um custo de memória proibitivo.

2.2.4 Contribuições de Bærentzen

Bærentzen [2] propôs uma estrutura muito semelhante às ADFs, e trouxe duas contribuições importantes. Primeiro, com o objetivo de eliminar as amostras redundantes da estrutura da ADF, ele optou por desacoplar

o armazenamento de células do armazenamento de vértices. As células são organizadas em uma *octree* com ponteiros, como é usual, mas as amostras de distância são armazenadas em uma tabela de dispersão (*hash table*) 3-D, indexadas pela posição do vértice. Esta estratégia elimina completamente a repetição de amostras nas ADFs. No entanto, para recuperar os dados de uma célula, ao invés de oito acessos diretos à memória seriam necessárias oito consultas a uma tabela de dispersão. Neste caso, os custos extras do algoritmo de dispersão tornariam o método caro demais para aplicações interativas.



2.5(a): Normais calculadas pelo método padrão das ADFs.

2.5(b): Normais calculadas pelo método sugerido por Bærentzen.

Figura 2.5: ADF de uma esfera com apenas dois níveis de subdivisão.

Bærentzen também questionou o uso de normais de superfície calculadas como gradientes localizados em uma única célula. Os gradientes estão sujeitos a descontinuidades na fronteira das células (como pode ser observado na Figura 2.5(a)), e mesmo que as ADFs sejam muito subdivididas, é impossível evitar que as descontinuidades tornem-se visíveis caso a câmera se aproxime muito da superfície. Para obter uma normal suave em um dado ponto de uma superfície, Bærentzen propõe a seguinte solução:

1. Encontrar a menor célula da *octree* que contém o ponto onde a normal será calculada.

2. Calcular os gradientes do campo de distância nos oito vértices da célula. Isto é feito através de diferenças finitas, usando as seis amostras de distância mais próximas ao redor de cada vértice.

3. Estimar a normal suave através de interpolação trilinear dos gradientes calculados nos oito vértices da célula.

Apesar de conceitualmente simples, este método é caro, uma vez que, para calcular cada gradiente, são necessárias 48 buscas pelas amostras de distância mais próximas ao redor dos vértices da célula.

2.2.5 Técnicas de Renderização

Existem três abordagens básicas para a renderização de iso-superfícies de ADFs, como descrito em [24]:

I. Lançamento de raios (*ray-casting*);

II. Renderização baseada em pontos;

III. Tecelagem de malhas poligonais;

Para aplicações interativas, Perry e Frisken [24] argumentaram a favor das duas últimas abordagens, já que elas podem facilmente tirar proveito de placas gráficas na renderização. No entanto, com o advento das GPUs programáveis, tornou-se possível também acelerar o lançamento de raios através de placas gráficas — como demonstrado nesta dissertação.

O lançamento de raios é uma técnica cara, que pode ser usada para muitas aplicações e geralmente resulta na melhor qualidade gráfica. Nas ADFs, o lançamento de raios utiliza a *octree*: cada raio atravessa as células da *octree* até encontrar uma célula que contenha a iso-superfície sendo renderizada. Para calcular o ponto exato de interseção do raio com a superfície, é necessário encontrar as raízes de uma quádrica. Porém, na prática, o ponto de interseção pode ser aproximado linearmente, dadas as distâncias nos pontos em que o raio entra e sai da célula, sem qualquer diferença perceptível de qualidade. Por basear-se nas *octrees* das ADFs, esta técnica é mais eficiente para ADFs de fronteira, e menos eficiente para ADFs globais.

A renderização baseada em pontos, até poucos anos atrás, era a única solução viável para se renderizar ADFs complexas interativamente. Esta técnica utilizava a CPU para gerar e projetar pontos em uma iso-superfície, e a GPU para desenhar os pontos. Nesta técnica, inicialmente os pontos são distribuídos de forma aleatória dentro das células-folha que contenham a iso-superfície. Em seguida, utilizando o campo de distância e o gradiente, cada ponto é projetado na iso-superfície em um único passo. Segundo Perry e Frisken [24], este processo é muito rápido — podendo ser usado interativamente com mais de um milhão de pontos — mas oferece baixa qualidade de imagem.

A técnica de triangulação apresentada por Perry e Frisken [24] oferece um bom equilíbrio entre desempenho e qualidade gráfica. A triangulação é feita através de uma versão modificada do algoritmo *SurfaceNets*, com base na *octree* da ADF e no campo de distância. As malhas geradas são fechadas e orientadas, compostas por triângulos de alta qualidade (quase equiláteros).

O processo de tecelagem é rápido o suficiente para ser usado dinamicamente, gerando malhas detalhadas com base no *frustum* de visão da câmera.

Por utilizarem diretamente a *octree* das ADFs, as técnicas de renderização propostas por Perry e Frisken são impróprias para alguns tipos de aplicações. Por exemplo, elas não podem ser usadas para renderizar metamorfoses ou quaisquer transformações aplicadas dinamicamente ao campo de distância das ADFs. Uma das contribuições desta dissertação é justamente uma técnica de lançamento de raios de *propósito geral*, capaz de renderizar ADFs complexas em GPU utilizando apenas o campo de distância.

2.3 Lançamento de Raios em GPU

As GPUs modernas têm grande poder de processamento paralelo, sendo compostas por centenas de processadores especializados em contas com ponto flutuante. Algoritmos de lançamento de raios podem facilmente tirar proveito de arquiteturas paralelas, já que cada raio é independente e pode ser lançado por um processador diferente. Isto faz das GPUs uma das arquiteturas mais apropriadas para algoritmos de lançamento de raios simples.

Com o advento das GPUs programáveis, Toledo e Levy foram um dos primeiros a explorar a GPU para o lançamento de raios. No trabalho [32], eles mostram como esferas e quádricas podem ser usadas transparentemente como primitivas do *pipeline* gráfico tradicional, com base no lançamento de raios.

Trabalhos mais recentes sobre lançamento de raios em GPU incluem [21, 28] e [12]. Neste último, os autores representam um campo de distância amostrado regularmente através de uma grade hierárquica de dois níveis, e renderizam iso-superfícies na GPU utilizando técnicas de sombreamento avançadas, com sombreamento diferido (*deferred shading*).

Apesar de todas as vantagens mencionadas, as GPUs também sofrem sérias limitações que dificultam seu uso para algoritmos avançados. Uma das principais limitações é a ausência de uma *pilha de execução* intrínseca, o que dificulta — e com freqüência inviabiliza — a execução de algoritmos essencialmente recursivos, como traçado de raios (*ray-tracing*) e algoritmos que precisem percorrer hierarquias. Como estruturas espaciais hierárquicas são imprescindíveis para muitas aplicações, inclusive ADFs, diversos autores tentaram contornar as limitações da GPU e desenvolveram algoritmos e estruturas [27, 15] que podem ser atravessadas eficientemente sem necessitar de uma pilha. De forma semelhante, uma solução foi desenvolvida nesta dissertação para percorrer a *octree* das ADFs em GPU.