Relações Espaciais, Temporais e de Ordem para Detecção de Colisão Broad Phase Genérica e Escalável

Ygor Rebouças Serpa[1], **Maria Andréia Formico Rodrigues**[2] **(Orientadora)**

[1] Centro de Ciências Tecnológicas (CCT)
[2]Programa de Pós-Graduação em Informática Aplicada (PPGIA)
Universidade de Fortaleza (UNIFOR) – 60811-905 – Fortaleza-CE – Brasil

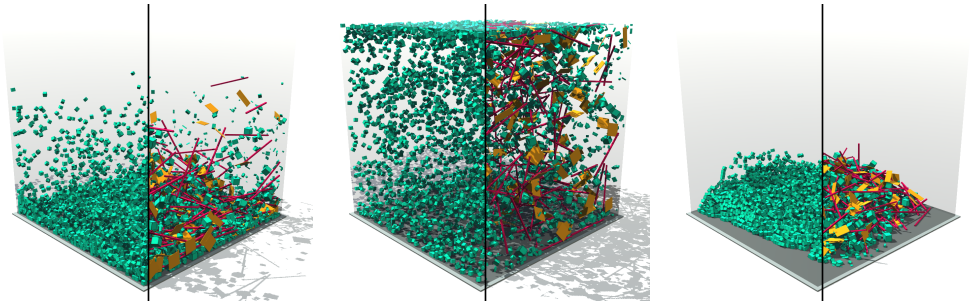
{ygor.reboucas,andreia.formico}@gmail.com

Abstract. *Physical simulations are fundamental to the computational representation of real-world situations. In this context, the collision detection among 3D objects is crucial, as it provides the illusion that they are solid, however, there is no solution yet simultaneously capable of acting efficiently at the general case and scaling competitively. This work presents and validates a novel optimized solution for the broad phase collision detection that fills this gap, which is based on an association between the KD-Tree and the Sweep-and-Prune and incremental techniques. The solution performed two to three times better than the other state-of-the-art solutions, being robust to varying behaviours, sizes and object distributions. Moreover, it was capable of efficiently processing all tested scenarios and scaling in scenes with up to one million objects.*

Resumo. *Simulações físicas são fundamentais para a representação computacional de situações do mundo real. Neste contexto, a detecção de colisões entre objetos 3D é crucial, uma vez que provê a ilusão de que estes são sólidos, porém, não há uma solução simultaneamente capaz de eficientemente atuar genericamente e escalar de maneira competitiva. Este trabalho apresenta e valida uma nova solução otimizada para a detecção de colisão broad phase que preenche esta lacuna, a qual é baseada em uma associação entre a KD-Tree e as técnicas Sweep-and-Prune e incremental. A solução obteve resultados duas a três vezes melhores que as demais soluções do estado-da-arte, sendo robusta a variações de comportamento, tamanho e distribuição dos objetos. Além disso, mostrou-se capaz de atuar eficientemente em todos os cenários testados e escalar em ambientes com até um milhão de objetos.*

1. Introdução

Simulações físicas são fundamentais para a representação computacional de situações do mundo real, tais como: geração de efeitos especiais, previsão climática, aplicações médicas, etc. Neste contexto, detectar colisões entre objetos é crucial, uma vez que esta garante a não-interseção entre os objetos e, portanto, confere a ilusão de que estes são sólidos [Ericson 2004]. É comum que esta etapa seja o maior gargalo da execução, pois é inerentemente quadrática (qualquer objeto pode potencialmente colidir com qualquer outro). Tradicionalmente, limita-se à detecção dos volumes envoltórios dos objetos (*broad phase*), refinando o resultado através do uso das geometrias reais (*narrow phase*). Desta forma, a *broad phase* é responsável pelo desempenho, buscando potenciais colisões; já



(a) Queda-Livre (b) Partículas (c) Gravidade

Figura 1: Exemplos de simulações por corpos rígidos.

a *narrow phase* é responsável pela precisão, refinando o resultado. Algoritmos de *broad phase* são notáveis pela sua eficácia. Todavia, as soluções existentes do estado-da-arte (em sua grande maioria) não escalam de forma competitiva ou são projetadas para cenários específicos, sendo inexistente até o momento uma solução capaz de lidar com números massivos de objetos e atuar no caso geral.

Este trabalho apresenta uma nova solução para a área de detecção de colisão *broad phase*, com foco na generalidade e na escalabilidade. Especificamente, consiste em uma solução híbrida que associa a *KD-Tree* [Glass 2005], a técnica *Sweep-and-Prune* [Tracy et al. 2009] e a técnica incremental [Liu et al. 2010], abordagens originalmente consideradas, respectivamente, insuficiente para cenas dinâmicas, não-escalável e limitada a cenas estáticas. Nos resultados obtidos, nossa solução mostrou-se bastante competitiva, sendo capaz de escalar para até um milhão de objetos de forma quasi-linear e de realizar a detecção de colisão *broad phase* na metade ou até mesmo no terço do tempo necessário pelas melhores soluções do estado-da-arte em CPU (mesmo em cenários originalmente representativos do melhor caso destas soluções). Finalmente, é importante destacar que este trabalho é fruto do trabalho exclusivo do aluno sob supervisão de sua orientadora, não sendo parte de um projeto maior.

2. Trabalhos Relacionados

Em alto nível, os trabalhos em detecção de colisão *broad phase* podem ser classificados pelo seu uso de relações espaciais, de ordem e temporais [Ericson 2004]. Relações espaciais são empregadas para dividir-e-conquistar o problema e, geralmente, consistem no uso de estruturas de particionamento espacial, como por exemplo: *Grid*s [Lo et al. 2013], *BVH*s [Coumans 2018], *KD-Tree*s, *Octree*s e *BSP-Tree*s [Luque et al. 2005]. Relações de ordem, por sua vez, são ordenações com a propriedade que objetos próximos na ordem possuem uma maior probabilidade de colisão [Baraff and Witkin 1992]. Por fim, relações temporais constituem otimizações, tais como, atualizar ao invés de reconstruir estruturas [Tracy et al. 2009, Coumans 2018], ignorar objetos estáticos [Liu et al. 2010] e extrapolar trajetórias para prever colisões [Coming and Staadt 2005].

Para que um algoritmo possa ser considerado aplicável a uma grande variedade de cenários, este deve ser indiferente à distribuição dos objetos e ao dinamismo da cena. Contudo, a maioria dos algoritmos possui piores-casos bastante comuns. Por exemplo, algoritmos baseados em *Grid*s ou *Octree*s são vulneráveis à distribuições não uniformes, enquanto que soluções baseadas em *BVH*s, *KD-Tree*s e *BSP*s são difíceis de atualizar

e construir. Métodos baseados em ordenação, como o *Sweep-and-Prune*, são bastante eficientes para poucos objetos, mas possuem complexidade algorítmica subquadrática [Tracy et al. 2009]. Já as relações temporais podem ser negativas quando há um grande número de objetos dinâmicos, consequentemente, soluções que fazem bastante uso destas relações tornam-se ineficientes em cenários dinâmicos.

Contrastando com o estado-da-arte, a solução aqui apresentada [Serpa 2017] é robusta às distribuições não-uniformes, devido ao uso da *KD-Tree*, sem comprometer a escalabilidade, já que é associada ao algoritmo SAP [Tracy et al. 2009]). Em relação ao dinamismo da cena, uma série de otimizações foram desenvolvidas, as quais são ativadas/desativadas aplicando-se heurísticas para obter o melhor desempenho possível. Assim, a solução implementada atua em várias classes de cenários similarmente a soluções específicas, com resultados equivalentes aos destas soluções. Esta versatilidade torna a solução adequada ao caso geral, porém, com o desempenho de uma solução específica.

3. O Novo Algoritmo Desenvolvido

Em linhas gerais, a solução desenvolvida consiste na sinergia de quatro elementos principais: o *Preparo dos Objetos*, a *Estrutura de Particionamento*, o *Algoritmo de Atualização* e o *Algoritmo de Busca por Pares Colidentes*, descritos nas próximas seções.

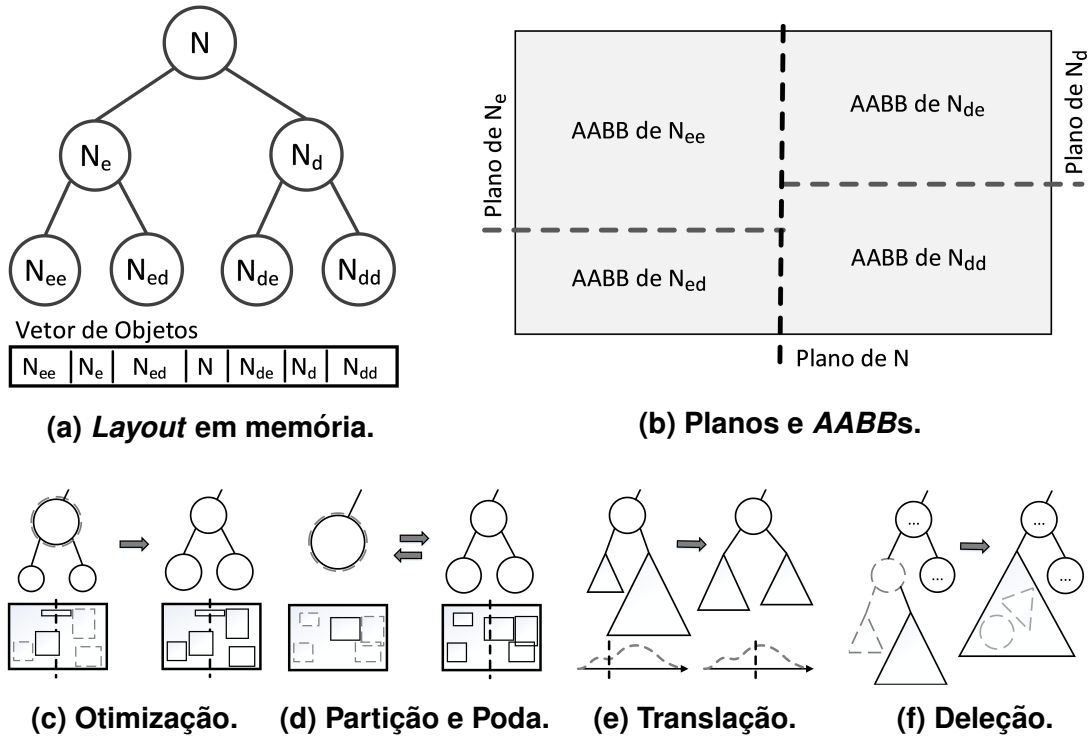
3.1. Preparo dos Objetos

Na inicialização do algoritmo, cada objeto é envolto na menor *Axis-Aligned Bounding Box* (*AABB*) [Ericson 2004] que o contém. Nas demais etapas, todas as operações com objetos são feitas com suas respectivas *AABB*s. A cada quadro, a *AABB* de cada objeto é atualizada para refletir a nova posição e orientação. Contudo, caso a nova *AABB* se ajuste dentro da *AABB* atual (acrescida de uma margem de 1%), a *AABB* não é atualizada e o objeto é considerado estático. Caso contrário, é atualizada e o objeto é julgado dinâmico. Esta classificação consiste em uma relação temporal, usada na detecção de colisão.

3.2. Estrutura de Particionamento

Para dividir-e-conquistar, emprega-se uma árvore k-dimensional estritamente binária, não balanceada e de altura variável, com objetos tanto nos nós internos, quanto nas folhas. Nesta estrutura, cada nó possui três elementos: uma *AABB*, um plano de corte e uma lista de objetos. A partir do nó raiz, a *AABB* correspondente consiste em uma *AABB* infinita, a qual é divida em duas partes pelo seu plano de corte. Subsequentemente, cada outro nó também subdivide sua respectiva *AABB*, até chegar aos nós folhas, os quais não subdividem o espaço e, portanto, não possuem plano de corte. Já para as listas de objetos, um vetor único é usado, seguindo um *layout* em-ordem. Esta representação permite que operações envolvendo subárvores inteiras sejam feitas de forma bastante eficiente (por exemplo, enumerar objetos, mesclar nós, etc.). A Figura 2 mostra em (a), a estrutura e o vetor de objetos e, em (b), os respectivos planos de corte e *AABB*s.

Para controlar a profundidade da árvore e maximizar seu uso, são aplicadas três regras: R_1 cada nó folha deve ter até T objetos; R_2 cada subárvore deve ter, no mínimo, T objetos; e R_3 cada objeto deve residir no nó mais profundo que consegue envolvê-lo. De forma controlável e granular, estas regras associadas representam a altura da árvore em função do número de objetos e garantem o aproveitamento máximo dos nós folha.



(a) *Layout* em memória. **(b) Planos e *AABB*s.**

(c) Otimização. **(d) Partição e Poda.** **(e) Translação.** **(f) Deleção.**

Figura 2: Estrutura e Operações definidas para a *KD-Tree*.

Adicionalmente, foram definidas e implementadas cinco operações: *Otimização*, *Partição*, *Translação*, *Poda* e *Deleção*, mostradas na Figura 2. Na operação de *Otimização*, aplica-se a regra R_3, transferindo objetos do nó para os seus nós filhos, sempre que possível. Já nas operações de *Partição* e *Poda*, são aplicadas as regras R_1 e R_2, respectivamente. Na *Partição*, cria-se um novo plano de corte a partir da média do eixo de maior variância dos objetos; já na *Poda*, mesclam-se todos os nós envolvidos. As operações de *Translação* e *Deleção* visam balancear a estrutura. Mais especificamente, na *Translação* move-se o plano de corte do nó para a nova média dos objetos e na *Deleção* remove-se o nó e sua subárvore menor, inserindo seus objetos na subárvore remanescente.

3.3. Algoritmo de Atualização

Em linhas gerais, esta etapa une as cinco operações anteriormente descritas em um fluxo eficiente e idempotente, com o objetivo de impor as três regras definidas e melhorar o particionamento, quando oportuno. Primeiramente, a árvore é percorrida das folhas à raiz em busca de objetos fora da *AABB* de seus nós, movendo-os para cima na estrutura. Ao fim desta operação, todos os objetos estarão em nós que os contenham por completo, porém, não necessariamente nos mais profundos. Em seguida, a árvore é percorrida da raiz às folhas, realizando um dos seguintes fluxos (dependendo do tipo do nó):

- Nó interno: primeiramente, executa-se a *Poda*, caso necessária, de acordo com a regra R_2. Caso contrário, faz-se a *Otimização* do nó e, em seguida, seu plano de corte é avaliado. Caso classificado como ineficiente, o nó sofre *Translação* e é novamente avaliado. Se ainda ineficiente, o nó sofre *Deleção*.
- Nó folha: nesta situação, faz-se apenas a *Partição*, caso necessária, conforme a regra R_1. Nesse caso, repete-se este fluxo para os nós recém criados.

Em execução, este algoritmo é conservativo em sua maior parte, buscando atender apenas às regras definidas. Contudo, quando necessárias, medidas mais drásticas são tomadas, como a *Translação* e *Deleção*. Em casos extremos, subárvores inteiras podem ser destruídas (via múltiplas deleções) e recriadas (via múltiplas partições). Já em cenários com pouco dinamismo, somente algumas manutenções são realizadas a cada quadro.

O plano de corte de um nó é avaliado comparando-se o custo estimado frente ao nível de balanceamento. O custo é calculado em termos do número de objetos existentes nele e em suas subárvores esquerda e direita, consistindo no número de pares formados, com exceção dos pares entre objetos de subárvores diferentes, de acordo com a Eq. 1:

$$C(n, n_e, n_d) = \binom{n}{2} + \binom{n_e}{2} + \binom{n_d}{2} + n(n_e + n_d) \tag{1}$$

Após calculado, este custo é normalizado para o intervalo $[0, 1]$ utilizando os custos mínimo ($C(0, \frac{n+n_e+n_d}{2}, \frac{n+n_e+n_d}{2})$) e máximo ($\binom{n+n_e,+n_d}{2}$). Em seguida, o nível de balanceamento do nó (a razão entre o número de objetos na subárvore menos populosa e o restante dos objetos) é calculado de acordo com a Eq. 2:

$$B(n, n_e, n_d) = \frac{min(n_e, n_d)}{n + max(n_e, n_d)} \tag{2}$$

Finalmente, caso $C > B$, o nó será avaliado negativamente. Analisando, a estimativa de custo do nó será sempre um limite superior do número real de pares gerados e sua precisão será proporcional à profundidade do nó. Tais características aumentam a frequência de avaliações negativas em nós próximos à raiz, mas não impedem que nós mais profundos sejam também otimizados. Esta propriedade é bastante interessante frente a outras técnicas, pois prioriza translações e deleções de nós que têm uma maior influência no particionamento, sem impedir que os demais nós também sejam otimizados.

3.4. Busca de Pares Colidentes

A priori, qualquer par de objetos tridimensionais existentes na cena é candidato a ser colidente. Todavia, para a maioria destes pares, é simples provar a inexistência de colisões entre os objetos do par, até que o conjunto de pares candidatos torne-se quase irredutível, restando como opção somente o teste força bruta. Neste trabalho, três técnicas distintas são aplicadas no descarte de pares de objetos, realizando o teste força bruta final com instruções *SIMD*.

Primeiramente, descartam-se pares candidatos, de acordo com o seguinte princípio: caso dois objetos estejam em ramos diferentes da árvore, é possível demonstrar que existe um plano de corte que os separa. A partir disso, um conjunto de pares candidatos é gerado para cada nó folha. Em seguida, para cada um destes conjuntos, aplica-se a técnica SAP, a qual consiste em duas tarefas: (1) ordenar os objetos pelos seus pontos de mínimo e máximo no eixo de maior variância; e (2) percorrer esta lista descartando colisões com o seguinte predicado: seja i e j os índices do ponto de mínimo e máximo de um objeto na lista ordenada, quaisquer outros objetos com ponto de máximo antes de i, ou de mínimo após j, não podem estar colidindo com este objeto. Desta forma, reduz-se ainda mais cada um dos sub-problemas, restando apenas uma pequena porcentagem do total de pares. Por fim, emprega-se a técnica incremental, cujo princípio consiste em buscar apenas novas colisões, ou seja, descartam-se pares candidatos que não possuem objetos

dinâmicos, já que estes são os únicos capazes de gerar novas colisões. Finalmente, os pares candidatos restantes são testados de modo força-bruta. Devido à técnica incremental, é necessário manter um *cache* de pares colidentes para armazenar colisões detectadas em quadros anteriores. Em linhas gerais, quando um objeto torna-se estático, suas colisões no quadro anterior são inseridas no *cache* e, quando torna-se dinâmico, suas colisões são removidas. Esta tarefa resulta em um custo e, para um número pequeno de objetos estáticos, torna-se prejudicial à execução. Neste contexto, uma ativação condicional é aplicada: caso haja mais de K objetos estáticos, a detecção incremental é ativada, caso contrário, é ignorada (esvaziando-se o *cache* e fazendo a detecção de forma completa, sem ignorar pares de objetos estáticos).

4. Metodologia de Teste

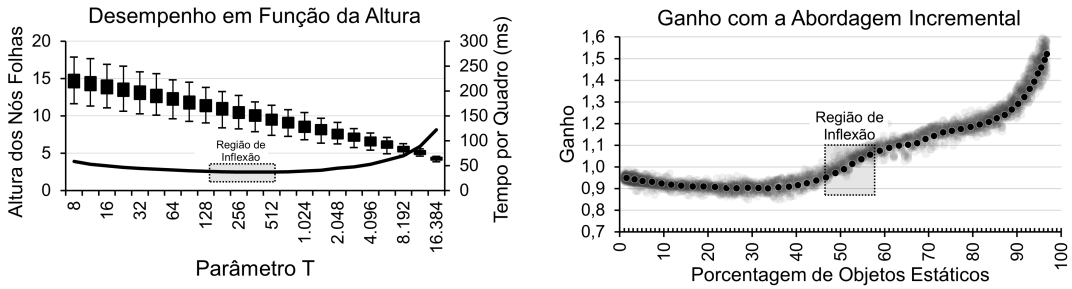
Para validar e comparar a solução desenvolvida com o estado-da-arte, três simulações foram criadas: *Queda Livre*, *Partículas* e *Gravidade*. Na primeira, objetos são instanciados e deixados a mercê da força gravitacional, eventualmente atingindo o solo e repousando. Já no cenário *Particulas*, cada objeto instanciado é sujeito a um impulso aleatório, aplicado de tempos em tempos. Finalmente, no cenário *Gravidade*, a direção da força gravitacional é continuamente alterada, evitando que haja repouso. Em resumo, o cenário *Queda Livre* é quase estático, enquanto que os cenários *Partículas* e *Gravidade* são totalmente dinâmicos, sendo os objetos distribuídos uniformemente em *Partículas* e não-uniformemente em *Gravidade*. Para analisar o impacto da geometria dos objetos no desempenho das simulações, cada uma delas foi gerada tanto com objetos de mesmo tamanho (*Cubos*), quanto com objetos de tamanhos variados (*Variados*). Estes cenários são mostrados na Figura 1. Animações ilustrativas de vários dos resultados obtidos podem ser visualizadas em `https://tinyurl.com/ybql9luy`.

Em geral, importantes aplicações do mundo real apresentam comportamento similar ao do cenário *Queda Livre*, tendo muitos objetos estáticos e apenas alguns poucos dinâmicos. Embora também relativamente comum em aplicações do mundo real, o cenário *Partículas* é melhor enquadrado como *benchmark*. Por fim, o cenário *Gravidade* foi especialmente criado pelos autores deste trabalho com a finalidade de modelar situações extremas, relevantes para testar os limites dos algoritmos.

Vale ressaltar que dois algoritmos do estado-da-arte têm importante destaque na indústria e literatura: *DBVT* da biblioteca *Bullet* [Coumans 2018] e *CGAL* [Kettner et al. 2018] da biblioteca de mesmo nome. O primeiro baseia-se unicamente em uma árvore *BVH* e possui duas especializações: *DBVT F*, voltada para cenas estáticas; e *DBVT D*, focada em cenas dinâmicas. Contrastando, o segundo apresenta uma abordagem híbrida (usa a técnica SAP e árvores de intervalo e de segmento), completamente direcionada a cenas dinâmicas. Cada teste foi executado cinco vezes, durante 1.000 quadros, totalizando 33 segundos de animação por teste. Em igualdade, todas as soluções, incluindo a dos autores deste trabalho, foram desenvolvidas em C++, baseadas em *AABB*s e executadas em um computador *Core i7 7700* com 8 GB de memória *RAM*.

5. Análise dos Resultados

Esta seção inicia-se com uma análise da influência dos parâmetros de configuração T e K na solução implementada. Em seguida, é realizado um estudo comparativo da mesma



(a) Parâmetro T **(b) Parâmetro K**

Figura 3: Em (a), o impacto de T na altura da árvore e no desempenho médio da solução nos 3 cenários (com 128 mil objetos). Em (b), o ganho/perda do uso da abordagem incremental em função do percentual de objetos estáticos (todos os quadros das execuções do cenário *Queda Livre*).

frente ao estado-da-arte. A Figura 3.a exibe uma nítida relação entre a altura da árvore e o desempenho médio nas três simulações executadas. Adicionalmente, mostra que árvores profundas ($T < 64$) ou rasas ($T > 2.048$) são significativamente menos eficientes que árvores médias ($T \simeq 256$). Observa-se que no caso de árvores profundas, atualizar e manter a estrutura torna-se muito custoso, enquanto que em árvores rasas, o tamanho dos subproblemas torna-se uma dificuldade para o SAP. Com base na curva de desempenho gerada e mostrada na Figura 3.a, foi escolhido $T = 256$, valor com melhor custo-benefício. Além disso, conforme o número de objetos estáticos aumenta, nota-se que a técnica incremental passa de negativa à vantajosa, como mostra a Figura 3.b. Para menos de 45% objetos estáticos, pode-se afirmar que é até 15% pior usar esta técnica, porém, para mais de 55% é coerente usá-la, havendo a possibilidade de obter ganhos de até 60%. Neste contexto, mostra-se estratégico o uso da ativação condicional, escolhendo-se $K = 55\%$.

Um estudo comparativo para até 32 mil objetos de ambos os tipos *Cubes* e *Variados* é mostrado na Figura 4. Nota-se que solução desenvolvida apresentou um desempenho superior ao das demais soluções em todos os testes realizados. No cenário *Queda Livre*, a solução é rivalizada pelo algoritmo *DBVT F*, o qual é otimizado para esta situação. Isto comprova que a solução desenvolvida é capaz de competir de igual para igual com uma solução otimizada para este tipo de cenário. De maneira similar, nos cenários *Gravidade* e *Partículas*, a solução é competitiva frente as duas técnicas otimizadas para o caso dinâmico (*DBVT D* e *CGAL*). Em particular, percebe-se nas três simulações testadas que todas soluções apresentam uma maior dificuldade no processamento do cenário *Gravidade*. Tal fato decorre de dois fatores principais: (1) O número de colisões é bastante elevado neste cenário; e (2) o tipo de movimentação gerada nesta simulação é bastante desafiadora, especialmente para soluções baseadas em árvores. Na Figura 4, comparando-se os resultados exibidos na coluna da esquerda com os da direita, nota-se um impacto no desempenho das soluções ao conduzir os testes com objetos de diferentes tamanhos (*Variados*). Entretanto, somente os algoritmos *DBVT F* e *DBVT D* apresentam resultados significativamente diferentes nestas simulações. A Figura 5 apresenta uma análise de escalabilidade para até um milhão de objetos do tipo *Cubes*. Apesar do número massivo de objetos, não há uma mudança significativa no comportamento dos algoritmos. Nesta análise, a solução desenvolvida se mantém como a melhor solução para as simulações, seguida das soluções *DBVT F* no cenário *Queda Livre* e da *DBVT D*, nos demais casos.

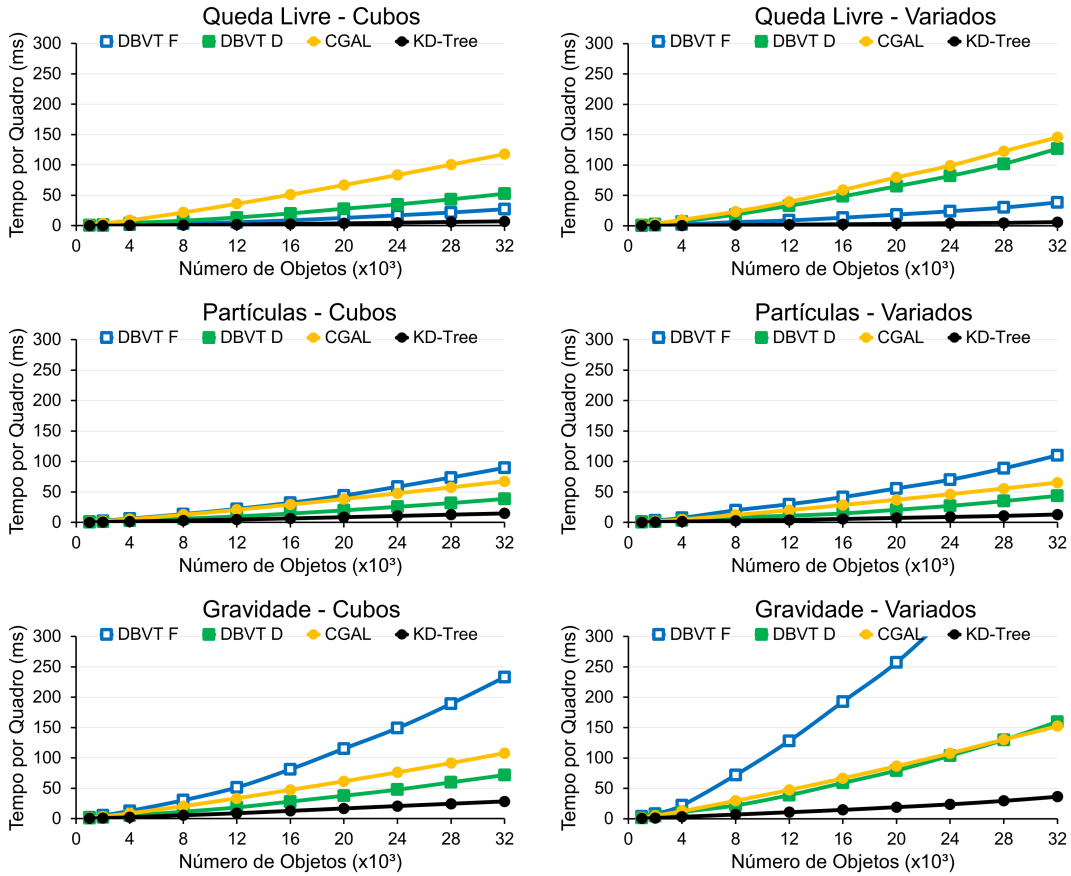


Figura 4: Resultados do DBVT F, DBVT D, CGAL e KD-Tree para simulações com até 32 mil objetos. À esquerda, com objetos de mesmo tamanho (*Cubos*) e, à direita, com objetos de diferentes tamanhos (*Variados*).

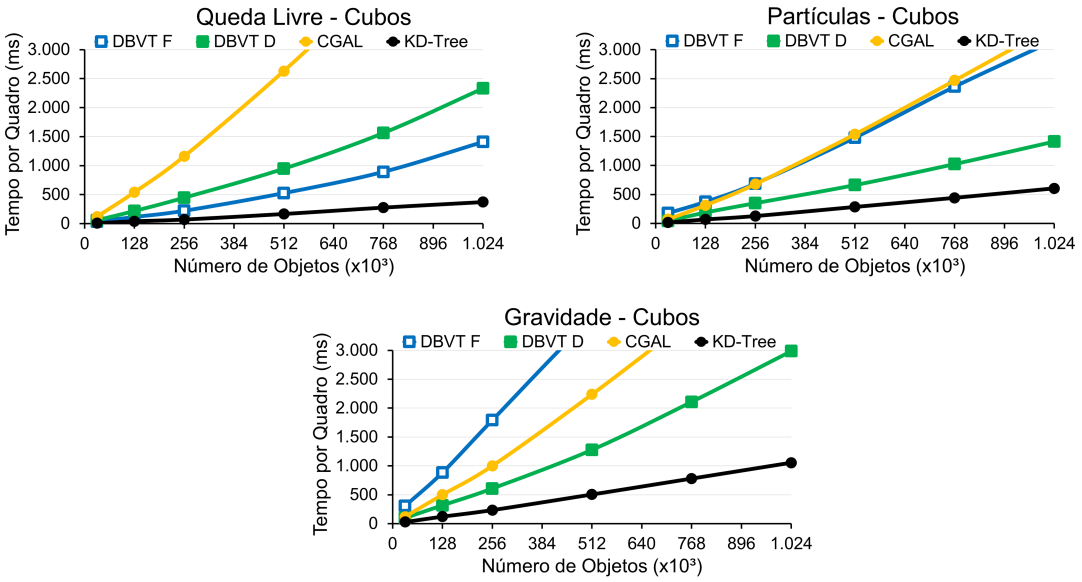


Figura 5: Análise de escalabilidade para até um milhão de objetos do tipo *Cubos*.

Nos dados mostrados nas Figuras 4 e 5, constatou-se um baixo valor para o *coeficiente de variação*, calculado como sendo o $\frac{desvio\ padrão}{média}$, sendo menor que $0,1$ em todos os casos, com exceção dos algoritmos *DBVT F* e do *KD-Tree* no cenário *Queda Livre*, os quais apresentaram valores em torno de $0,7$. Este aumento é decorrente das otimizações para cenários coerentes, que aceleram o algoritmo conforme os objetos repousam.

A partir das análises comparativas dos valores obtidos nos testes, pode-se argumentar que a solução desenvolvida é eficiente, escalável e aplicável a casos genéricos. Em particular, foi mostrado que a solução é capaz de lidar com cenas estáticas e dinâmicas, distribuições uniformes e não-uniformes, bem como com objetos de tamanhos idênticos e variados. Além disso, em todos estes diferentes cenários, o algoritmo desenvolvido apresentou o melhor desempenho frente todas as soluções testadas.

6. Conclusões e Trabalhos Futuros

Neste trabalho, uma nova solução para a detecção de colisão *broad phase*, baseada em uma associação entre a *KD-Tree* e as técnicas *Sweep-and-Prune* e incremental foi apresentada e validada. Os resultados obtidos mostram que a solução é capaz de atuar eficientemente em todas as simulações testadas e escalar para cenários massivos, com até um milhão de objetos. Comparativamente, a solução apresentou resultados duas a três vezes melhores que as demais soluções do estado-da-arte e mostrou-se robusta a variações de comportamento, tamanho e distribuição dos objetos. Além disso, pode ser aplicada com sucesso a outros domínios, por exemplo, detecção de proximidade, balanceamento de carga e organização de dados multidimensionais e dinâmicos.

Como trabalhos futuros, planeja-se a extensão da execução da solução desenvolvida para arquiteturas *multi* e *many-core*. É de interesse também explorar os seus elementos individuais em outros contextos, por exemplo, na *narrow phase* de malhas triangulares, em simulações de objetos deformáveis, no cálculo de visibilidade de objetos dinâmicos e na área de *ray tracing* em tempo real. Finalmente, espera-se prover acesso ao público, inserindo-a na biblioteca *Bullet*.

7. Publicações

Este trabalho de Iniciação Científica resultou em 6 publicações qualificadas, sendo 2 diretamente relacionadas ([Serpa and Rodrigues 2013] e [Serpa and Rodrigues 2014]) e 4 parcialmente relacionadas ([Oliveira et al. 2013], [Rodrigues et al. 2014], [Rodrigues et al. 2015] e [Serpa et al. 2016]), fundamentais para a aquisição de conhecimento técnico e científico ao longo do processo de desenvolvimento da pesquisa.

Agradecimentos

Ygor R. Serpa gostaria de agradecer à FUNCAP-CE (Processo No. 0074-00006.01.40/13) e ao CNPq (Processo No. 212628/2014-3), pelos apoios financeiros recebidos.

Referências

[Baraff and Witkin 1992] Baraff, D. and Witkin, A. (1992). Dynamic simulation of non-penetrating flexible bodies. In *Proc. of the* 19^{th} *ACM SIGGRAPH*, pages 303–308.

[Coming and Staadt 2005] Coming, D. S. and Staadt, O. G. (2005). Kinetic sweep and prune for collision detection. In *Proc. of the* 2^{nd} *VRIPHYS*.

[Coumans 2018] Coumans, E. (2018). Bullet physics: Dynamic bounding volume tree. `github.com/bulletphysics/bullet3`.

[Ericson 2004] Ericson, C. (2004). *Real-time Collision Detection*. CRC Press.

[Glass 2005] Glass, K. (2005). Analysis of broad-phase spatial partitioning optimizations in collision detection. Technical report, Rhodes University.

[Kettner et al. 2018] Kettner, L., Meyer, A., and Zomorodian, A. (2018). Intersecting sequences of dD iso-oriented boxes. In *CGAL User and Reference Manual*. CGAL Editorial Board.

[Liu et al. 2010] Liu, F., Harada, T., Lee, Y., and Kim, Y. J. (2010). Real-time collision culling of a million bodies on graphics processing units. *ACM TOG*, 29(6).

[Lo et al. 2013] Lo, S.-H., Lee, C.-R., Chung, I.-H., and Chung, Y.-C. (2013). Optimizing pairwise box intersection checking on GPUs for large-scale simulations. *ACM Transactions on Modeling and Computer Simulation*, 23(3).

[Luque et al. 2005] Luque, R. G., Comba, J. L. D., and Freitas, C. M. D. S. (2005). Broad-phase collision detection using semi-adjusting BSP-trees. In *Proc. of the ACM I3D*, pages 179–186.

[Oliveira et al. 2013] Oliveira, A. E., Serpa, Yvens R., Serpa, Ygor R., Abreu, A. P., Macedo, D. V., and Rodrigues, M. A. F. (2013). Visualização interativa 3D e simulação de dinâmica de acidentes de trânsito em forense utilizando a Blender Game Engine em um serious game. In *Anais do XII SBGames*, pages 39–47. SBC.

[Rodrigues et al. 2014] Rodrigues, M. A. F., Macedo, D. V., Serpa, Yvens R., Martins, C., Candolo, P. H. T., Gobet, T., Serpa, Ygor R., and Secundino, L. (2014). Combatendo a halitose: Um serious game multiplataforma em saúde bucal. In *Anais do XIII SBGames*, pages 210–219. SBC.

[Rodrigues et al. 2015] Rodrigues, M. A. F., Macedo, D. V., Serpa, Yvens R., and Serpa, Ygor R.(2015). Beyond fun: an interactive and educational 3D traffic rules game controlled by non-traditional devices. In *Proc. of the* 30^{th} *ACM SAC*, pages 239–246.

[Serpa 2017] Serpa, Ygor R. (2017). Algoritmos eficientes e escaláveis para detecção de colisão broad phase em CPU. Monografia de final de curso, Universidade de Fortaleza.

[Serpa and Rodrigues 2013] Serpa, Ygor R. and Rodrigues, M. A. F. (2013). Estudo comparativo entre algoritmos para detecção de colisão broadphase em CPU e GPU na Bullet. In *Proc. of the* 26^{th} *SIBGRAPI: Workshop of Undergraduate Works*. SBC.

[Serpa and Rodrigues 2014] Serpa, Ygor R. and Rodrigues, M. A. F. (2014). Parallelizing broad phase collision detection for animation in games: A performance comparison of CPU and GPU algorithms. In *Anais do XIII SBGames*, pages 770–779. IEEE.

[Serpa et al. 2016] Serpa, Yvens R., Serpa, Ygor R., and Rodrigues, M. A. F. (2016). A comparative study on a novel drawcall-wise visibility culling and space-partitioning data structures. In *Anais do XV SBGames*, pages 36–43. IEEE.

[Tracy et al. 2009] Tracy, D. J., Buss, S. R., and Woods, B. M. (2009). Efficient large-scale Sweep and Prune methods with AABB insertion and removal. In *Proc. of the IEEE VR*, pages 191–198.