INF 2063 – Tópicos em CG III

Visualização de Modelos Massivos

Prof. Alberto Raposo
Tecgraf / DI / PUC-Rio

Aula 07

Ray Tracing Interativo

Alberto Raposo - 2010

Referência principal da aula

- I. Wald, W. R. Mark, J. Gunther, et al. “State of the Art in Ray Tracing Animated Scenes”. Computer Graphics forum, 28 (6): 1691-1722, 2009.

Ray Tracing

- Mais poderoso para computar efeitos de iluminação mais sofisticados.
- Sempre foram usados offline por essa razão, e até bem pouco tempo atrás não atingiam taxas interativas.
- Com avanços de desempenho de CPU, disponibilidade de máquinas paralelas, avanços em algoritmos, etc
  - Ray Tracing agora atinge taxas interativas!

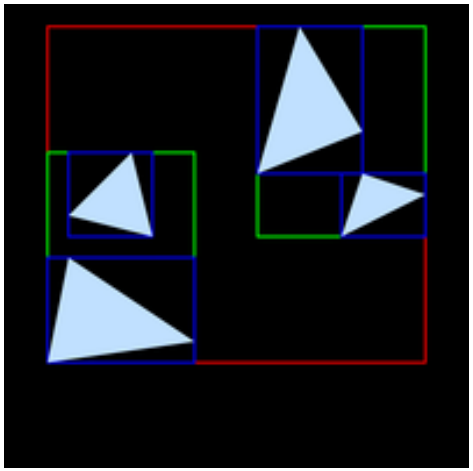
Ray Tracing Interativo (1)

- A chave para a velocidade do ray tracing está no uso de estruturas de dados “aceleradoras”, para diminuir o número de interseções raio-primitiva
  - Grids, BVH, Kd-trees, etc
- Até pouco tempo, a pesquisa se concentrava na eficiência dessas estruturas para traversal e operações de interseção com as primitivas
  - Tempo de construção das estruturas aceleradoras não importava para off-line rendering

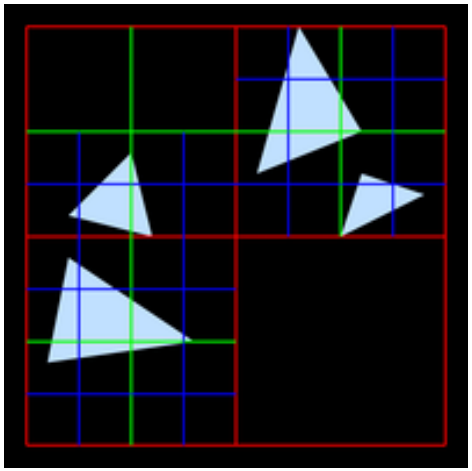
Ray Tracing Interativo (2)

- Com surgimento do ray tracing interativo, ele inicialmente apenas se aplicou a cenas estáticas, para fazer walkthroughs
- Ficou claro que tempo de construção das estruturas aceleradoras não pode ser ignorado, para a renderização de cenas dinâmicas
- Hoje, se busca balanço entre tempo de construção da estrutura vs eficiência na renderização/traversal

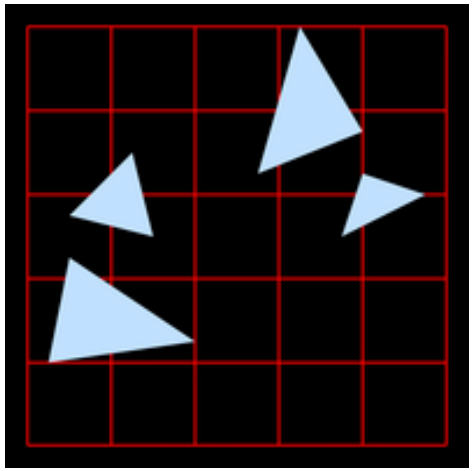
Estruturas aceleradoras



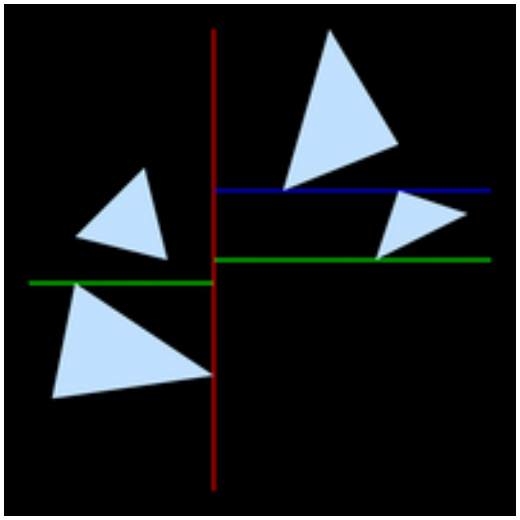
Bounding Volume Hierarchy



Octrees



Grids



kd-tree

Tipos de cena

- Estática
- Parcialmente estática (apenas algumas primitivas se movem)
- Dinâmica

Tipos de animação

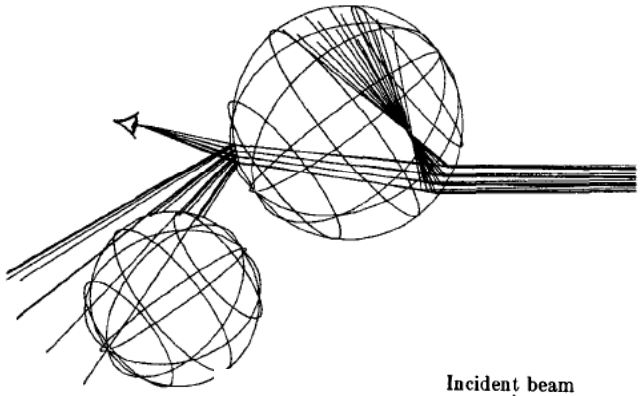
- Hierárquica
  - Cena dividida em grupos, e as primitivas de um grupo estão sujeitas ao mesmo tipo de transformação
- Movimento incoerente
  - Oposto da animação hierárquica: cada primitiva se move independente das demais
- Semi-hierárquica
  - Situação híbrida: movimento é principalmente hierárquico, mas há pequena quantidade de movimentos incoerentes em alguns objetos (ex., flock of birds).

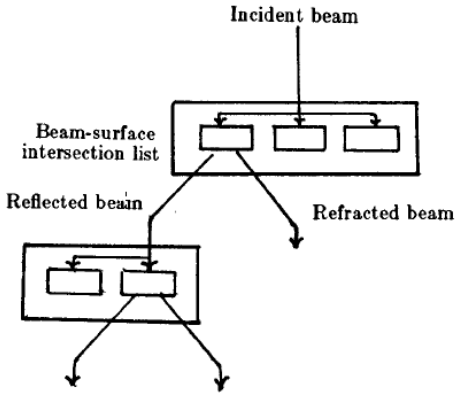
Coerência dos raios

- Os raios de um ray tracer típico apresentam grande coerência espacial
  - Mais forte para os eye rays, e menos forte (embora presente) em raios secundários
- Formas de explorar a coerência dos raios para melhorar desempenho do rendering
  - Beam tracing
  - Ray aggregation
  - Frustum techniques

Beam Tracing

- Raios não são representados explicitamente, mas os beams avaliam uma área exata da cena
- Em cenas compostas por primitivas médias ou grandes, beam tracing pode ser competitivo com frustum based ray-tracers
  - Mas com triângulos pequenos, desempenho piora, por causa dos inúmeros beam splits





Tecgraf PUC-RIO VR GROUP

Ray Aggregation

- Abordagem mais comum (em pacotes ou frustum)
- Raios são representados explicitamente, mas custo de algumas operações são amortizadas no “pacote” de raios
  - Acesso à memória
  - Chamadas de funções
  - Traversal computations
  - SIMD instructions

Ray Aggregation - Frustum

- Evolução do agregamento de raios por pacotes
  - Permitem pacotes maiores
  - Evitam passos no traversal e interseções com primitivas baseando-se em “bordas” conservativas do pacote de raios

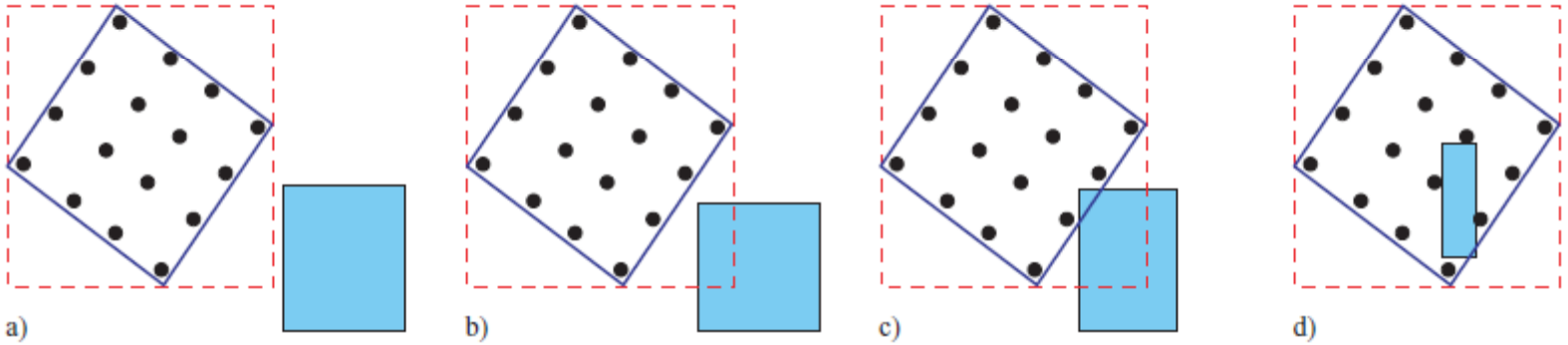


Fig. 3. Four ways a packet can miss a subtree of a BVH. Points denote rays. Blue lines are a perfect bounding frustum; red lines show a conservative bounding frustum as used by the interval arithmetic test; the blue box denotes the bounding volume. a) A wide miss where even the conservative frustum misses the bounding volume. b) The packet fully misses the bounding volume, but the (conservative) frustum test cannot detect it. c) Even the perfect frustum overlaps the node, but none of the rays does. d) The node is very small (as for highly-tesselated scenes) but falls in between the raster of rays. Only the first one is handled correctly with an interval arithmetic frustum technique. The second one is a deficiency of the conservative interval arithmetic test, while the third and fourth are general deficiencies of any frustum method. Our traversal scheme handles all four cases correctly.

I. Wald, S. Boulos, P. Shirley. “Ray tracing deformables scenes using dynamic bounding volume hierarchies. ACM Transactions on Graphics, 26 (1)Ç 1-18, 2007.

Estruturas aceleradoras e tradeoffs

- Subdivisão espacial vs hierarquia de objetos
- Alinhada com os eixos vs planos arbitrários
- Adaptativa vs não-adaptativa
- Tempo de construção vs qualidade da construção
- Necessidades da aplicação vs necessidades do rendering engine

Subdivisão espacial vs hierarquia de objetos (1)

- São duais por natureza
  - Subdivisão espacial representa cada ponto do espaço unicamente, mas cada primitiva pode estar em mais de uma célula
    - Grids, octrees, kd-trees
  - Hierarquia de objetos representa cada objeto apenas uma vez, mas cada ponto 3D pode ser "sobreposto" por várias células
    - BVHs e variantes

Subdivisão espacial vs hierarquia de objetos (2)

- Para traversal (encontrar o primeiro ponto de interseção ao longo do raio)
  - Mais simples com divisão espacial
    - Ponto no espaço é representado apenas 1 vez, permitindo "early exit" da estrutura tão logo essa interseção seja encontrada na ordem "frente para trás"
    - Porém divisão espacial pode fazer mesmo objeto ser visitado mais de uma vez, e também pode ter muitas células vazias
- Para atualizações da estrutura, quando objeto se move
  - Hierarquia de objetos é mais simples de atualizar
    - Objeto está em apenas um nó da hierarquia, e operações de atualização se tornam mais localizadas

Alinhamento com os eixos vs planos arbitrários

- Planos que particionam o espaço ou objetos são normalmente alinhados com os eixos x, y, ou z. Mas podem também ser arbirtrariamente orientados
  - Vantagens do alinhamento
    - Testes de interseção são mais fáceis e mais rápidos
    - Orientação dos planos ocupam menos espaço em memória
  - Vantagens do não-alinhamento
    - As geometrias ficam mais “encaixadas” nas subdivisões da estrutura

Adaptativas vs não-adaptativas

- Não-adaptativas
  - Localizações dos planos pré-determinadas; não olham a geometria da cena (ex., grids, octrees)
- Adaptativas
  - Consideram a geometria da cena para fazer a subdivisão do espaço (ex., kd-trees)
  - Melhores para lidar com pacotes de raios
  - Mais custosas para construir
    - Requerem cálculos para as subdivisões
    - Paralelização é mais difícil

Tempo de construção vs qualidade da construção

- Quão boa a estrutura é para minimizar o custo de rendering vs quanto tempo ela leva pra ser construída
- Exemplo
  - SAH (surface area heuristics) é a melhor maneira de decidir como organizar a geometria numa hierarquia, mas é cara para construir
  - Uso da mediana é mais barato, mas tem desempenho pior no traversal.

Necessidades da aplicação vs necessidades do rendering engine

- Como um sistema de RV ou um jogo vai usar um engine de ray-tracing?
- Exemplo
  - Uma “sopa de polígonos” ou um grafo de cena é mais adequado para a aplicação
  - Uma estrutura de aceleração é mais adequada para o engine de ray-tracing

Voltando às cenas dinâmicas

- Principal questão: como reconstruir ou atualizar a estrutura de aceleração a cada quadro ?
  - Deve-se buscar estruturas que melhorem a performance do traversal, desde que o seu custo de construção/atualização não ultrapasse o ganho de performance
  - De uma maneira geral:
    - Grids são mais baratos para construir, mas apresentam baixa performance de traversal
    - Kd-trees são mais eficientes para traversal, mas mais caras de construir
    - BVHs ficam mais ou menos no meio termo

Características da tarefa de rendering que afetam o trade-off entre construção e performance da estrutura de aceleração

- Tipo de movimento
  - Movimento gradual de objetos rígidos: favorece atualizações parciais da estrutura de aceleração
  - Criação e desaparecimento de geometrias na cena: favorece abordagens que reconstroem a estrutura de aceleração completamente
- Complexidade da geometria da cena
- Número total de raios
- Tipos de raios (ex., raios secundários são menos coerentes espacialmente)
- Estratégia de agregamento dos raios

Reconstrução vs Atualização vs Estático

- Reconstruir totalmente a estrutura a cada quadro
  - Mais geral: lida com qualquer tipo de movimento arbitrário
  - Pode sair cara, especialmente em cenas grandes
- Atualização da estrutura a cada quadro
  - Aproveita coerência dos quadros
  - É menos custoso que reconstruir a estrutura toda
  - Problema: qualidade da estrutura tende a degradar após várias atualizações
- Se há uma considerável parte da cena estática, essa parte pode ser armazenada em uma estrutura estática separada
- Há várias estratégias híbridas, misturando as acima

Abordagens baseadas em Kd-trees

Alberto Raposo - 2010

Kd-trees

- São as abordagens mais comuns para ray-tracing de cenas estáticas
- Têm dificuldades em cenas dinâmicas porque suas atualizações são custosas
  - Trabalhos têm sido realizados para superar essas limitações

Kd-trees e SAH (1)

- A SAH provê um método para estimar o custo de uma kd-tree para ray-tracing, baseada na distribuição dos raios pela cena.

Given AABB of current node and list of primitives *L*
  foreach possible split position
    $\boldsymbol{cost = C_{traversal} + Prob_{left} * Cost_{left} + Prob_{right} * Cost_{right}}$
    keep track of split plane which minimizes this cost

- Cálculo da probabilidade assume que os raios estão distribuídos homogeneamente ao longo do espaço
  - $Prob_{child}$ = ratio of child AABB's surface area to parent node's surface area
  - $Cost_{child}$ = number of primitives contained in child
  - $Prob_{child}$ * $Cost_{child}$ = *Expected* cost of entering child node

Kd-trees e SAH (2)

- Número de possições possíveis para split é infinito
  - quais considerar?
  - consider apenas splits nos extremos das primitivas no eixo de corte
- Complexidade: os altos custos de construção anulariam os ganhos no traversal?
  - naïve construction: $O(n^2 \log n)$
  - Como acelerar a contrução?
- Sort primitives along split axis
  - Em cada nó => $O(n \log^2 n)$
  - Uma vez apenas, na rai => $O(n \log n)$
  - I/O de memória é geralmente o fator limitante

Kd-trees e SAH (3)

- W. Hunt, G. Stoll, W. Mark. “Fast kd-tree construction with an adaptive error-bounded heuristic”. IEEE Symp. on Interactive Ray Tracing 2006, pp. 81-88.
  - Usa SIMD para contar as primitivas, single thread
  - Tempo de construção: 300K primitivas por segundo
- M. Shevtsov, A. Soupikov, A. Kapustin. “Fast and scalable kd-tree construction for interactively ray tracing dynamic scenes”. Eurographics 2007.
  - Escalabilidade linear com até 4 threads.

Kd-trees e SAH (4)

- Razor: sistema para construir kd-tree de alta qualidade (com SAH aproximado) em taxas interativas.
- P. Djeu, W. Hunt, R. Wang et al. “Razor: An Architecture for dynamic multiresolution ray tracing”. ACM Trans. on Graphics (2007).

Abordagens baseadas em BVHs

Alberto Raposo - 2010

BVHs vs Kd-trees

- Kd-trees têm sérias limitações para cenas dinâmicas
  - Mudanças pequenas invalidam toda a árvore
  - Tem que se reconstruir a kd-tree a cada frame, usando técnicas de construção rápida, ou no esquema lazy/on demand
- BVHs têm muito mais flexibilidade em termos de atualizações incrementais (já explorada para detecção de colisão)
  - Pode-se mudar apenas bounding volumes, e não a hierarquia toda
- Apesar dessas vantagens para cenas dinâmicas, durante bom tempo se acreditou que sua pior performance no traversal não compensaria para o uso em ray tracing interativo
  - Avanços recentes têm mudado essa visão

Construção de BVHs

- Em teoria, poderia ser qualquer árvore n-ária, e com qualquer volume envolvente. Na prática, BVHs são geralmente árvores binárias, com AABBs.
- O estado da arte em construir BVHs eficientes é usar SAH em construção top-down (similar às kd-trees).
  - Em geral são construídas mais rapidamente que as kd-trees, mas ainda não são tempo real.
- I. Wald. “On fast construction of SAH based bounding volume hierarchies”. IEEE Symp. on Interactive Ray Tracing 2007, pp. 33-40.

Fast BVH traversal (1)

- I. Wald, S. Boulos, P. Shirley. “Ray Tracing deformable scenes using dynamic bounding volume hierarchies”. ACM Transactions on Graphics, 26(1): 1-18. 2007.
- Combina 4 otimizações algoritmicas independentes (pacote de raios) em um novo algoritmo de traversal
  - Early hit test with speculative descent: se um raio do pacote colidir com a bounding box, desce na hierarquia, independente dos demais raios
  - Tracking the first active rays: raios do pacote, anteriores ao que “bateram” na bounding box não precisam mais ser testados na sequencia da hierarquia
  - Uso do frustum do pacote para testar interseção com nós
  - Processamento SIMD para todas essas operações

Fast BVH traversal (2)

Fig. 7. Fairy-forest test scene. An animated dragonfly (a) and a dancing fairy (b) placed into a typical game environment with background textures and animated foreground geometry (c). The resulting scene (d) consists of 180,000 animated triangles and is rendered with textures, shading, and shadows, at 2.2 fps at 1024 × 1024 resolution.

Resumo: BVHs vs Kd-trees

- BVHs foram negligenciadas por muito tempo, e voltaram à tona pela sua capacidade de “refitting”.
- Em hardware equivalente, parece que o desempenho de BVH no traversal ainda fica um pouco atrás das kd-trees
- Para cenas dinâmicas, houve esforço de pesquisa maior em kd-trees, mas a maior parte deles são aplicáveis a BVH: potencial para BVH superar kd-tree?
- Em outros aspectos, BVHs e kd-trees empatam
  - Consumo de memória
  - Adequação a pacotes e frustum de raios
  - Cenas complexas...

SKD-Trees

- Variação de BVH: ao invés de usar caixas completas, cada nó armazena 2 planos paralelos que particionam o nó em 2 metades
  - Resultado é parecido com kd-tree, mas se comporta como BVH
  - Comparação direta com BVH ainda não foi completamente investigada
- Vantagens
  - Podem usar os bons algoritmos de traversal de kd-tree
  - São fáceis de construir e atualizar como as BVHs
- Desvantagens
  - Bounding volumes não ficam tão “encaixados” nas geometrias
  - Produzem-se nós vazios

Abordagens baseadas em grids

Alberto Raposo - 2010

Grids

- Diferentemente de BVHs e kd-trees, grid não é estrutura adaptativa: é divisão espacial uniforme
- Principais vantagens
  - Custo de construção mínimo, podendo ser feito em uma única passada, e em paralelo
  - Pode ser reconstruído a cada frame, sem precisar considerar nada sobre os movimentos na cena
- Mesmo assim, até pouco tempo não receberam a devida atenção dos pesquisadores

Coherent Grid Traversal (1)

- Implementação de pacote de raios não é tão simples quanto nas estruturas baseadas em árvore.

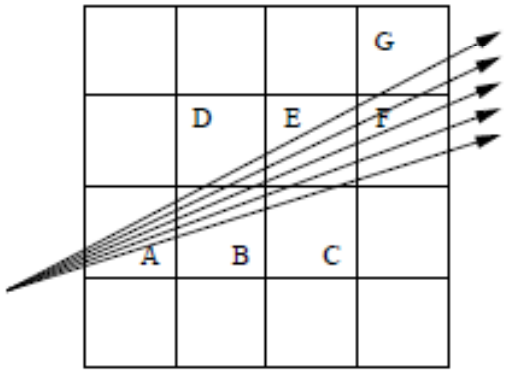


Figure 9: *Five coherent rays traversing a grid. The rays are initially together in cells A and B, but then diverge at B where they disagree on whether to first traverse C or D in the next step. Even though they have diverged, they still visit common cells (E and F) afterwards.*

Tecgraf PUC-RIO VR GROUP

Coherent Grid Traversal (2)

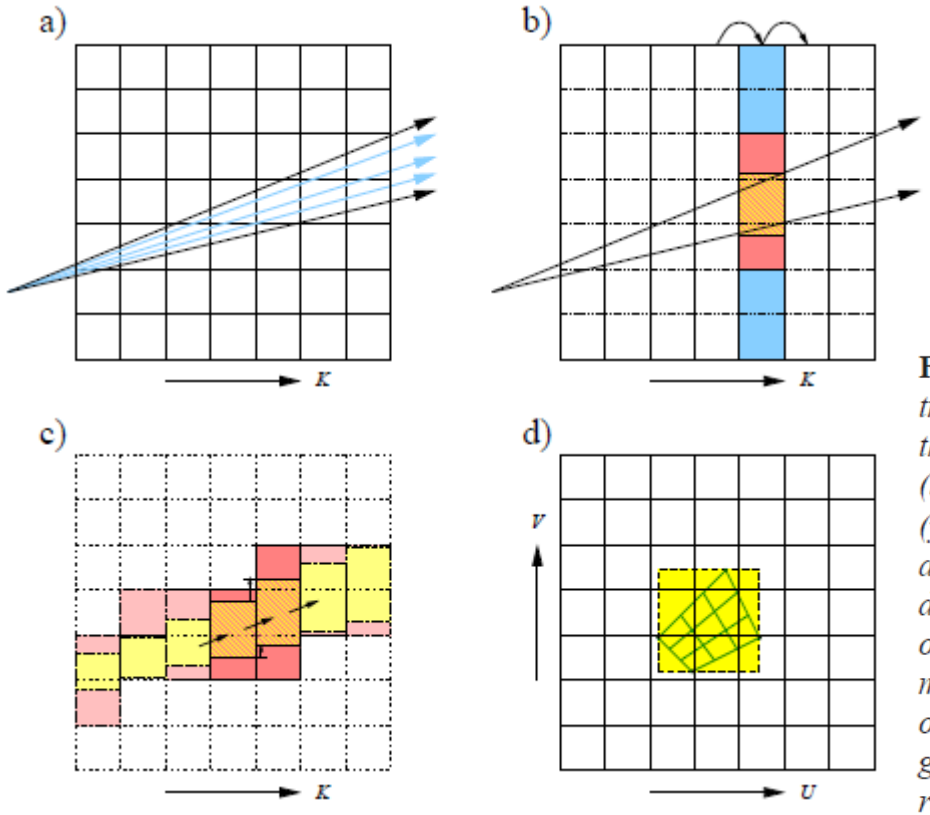


Figure 10: *Given a set of coherent rays, the coherent grid traversal first computes the packet's bounding frustum (a) that is then traversed through the grid one slice at a time (b). For each slice (blue), the frustum's overlap with the slice (yellow) is incrementally computed, which determines the actual cells (red) overlapped by the frustum. (c) Independent of packet size, each frustum traversal step requires only one four-float SIMD addition to incrementally compute the min and max coordinates of the frustum slice overlap, plus one SIMD float-to-int truncation to compute the overlapped grid cells. (d) Viewed down the major traversal axis, each ray packet (green) will have corner rays that define the frustum boundaries (dashed). At each slice, this frustum covers all of the cells covered by the rays.*

Problema

- Grid não se encaixa tão corretamente na geometria

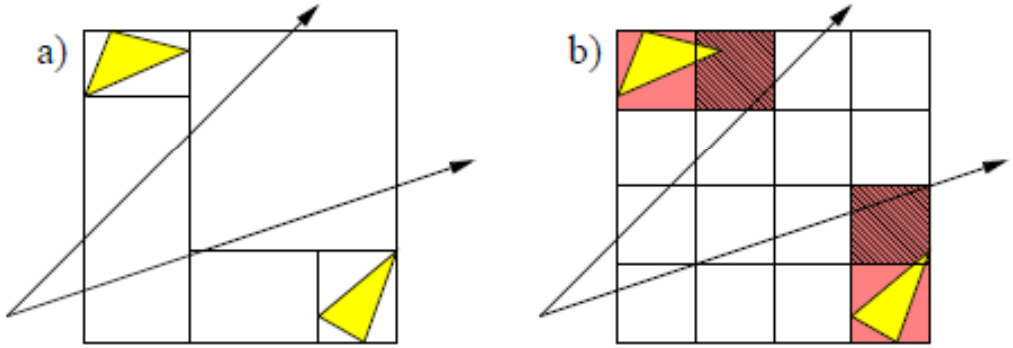


Figure 11: *Since a grid (b) does not adapt as well to the scene geometry as a kd-tree (a), a grid will often intersect triangles (red) that a kd-tree would have avoided. These triangles however usually lie far outside the view frustum, and can be inexpensively discarded by inverse frustum culling during frustum-triangle intersection.*

Solução

- SIMD frustum culling
  - Para traçado em pacote, se os 4 raios envolventes do pacote “perdem” o triângulo na mesma aresta do triângulo, então todos os raios do pacote não interceptam o triângulo

Outro problema

- Triângulos (especialmente os maiores) aparecem em várias células, e raios (e mais ainda, pacotes de raios) podem visitá-lo repetidas vezes

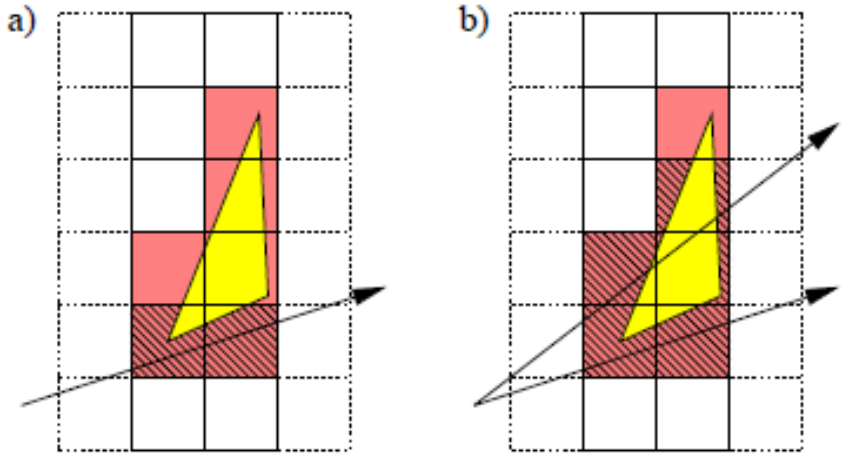


Figure 12: *While one ray (a) can re-visit a triangle in multiple cells only along one dimension, a frustum (b) visits the same triangle much more often (even worse in 3D). These redundant intersection tests would be costly, but can easily be avoided by mailboxing.*

Tecgraf PUC-RIO VR GROUP

Solução

- Mailboxing
  - Cada pacote assume um ID único, e o triângulo recebe um “tag” com essa ID quando é testado contra aquele pacote. Se o pacote reencontrar o mesmo triângulo em outra célula, ele verá o ID e pode pular o teste de interseção

Performance com grids

- Frustum culling e mailboxing reduzem bastante o número de testes de intereseção redundantes, remediando um pouco as deficiências de traversal em grids uniformes
  - Pode-se chegar a taxas próximas de sistemas com BVH e kd-trees, ainda com a facilidade de reconstrução dos grids

Conclusão

- Não há respostas claras para as perguntas sobre ray tracing interativo
- Há uma variedade de abordagens, difíceis de serem comparadas, pois usam diferentes hardware, cenas de teste, etc
- Há tantos fatores influenciando os prós e contras das abordagens, que uma “melhor” solução vai depender do problema abordado