INF 2063 – Visualização de Modelos Massivos

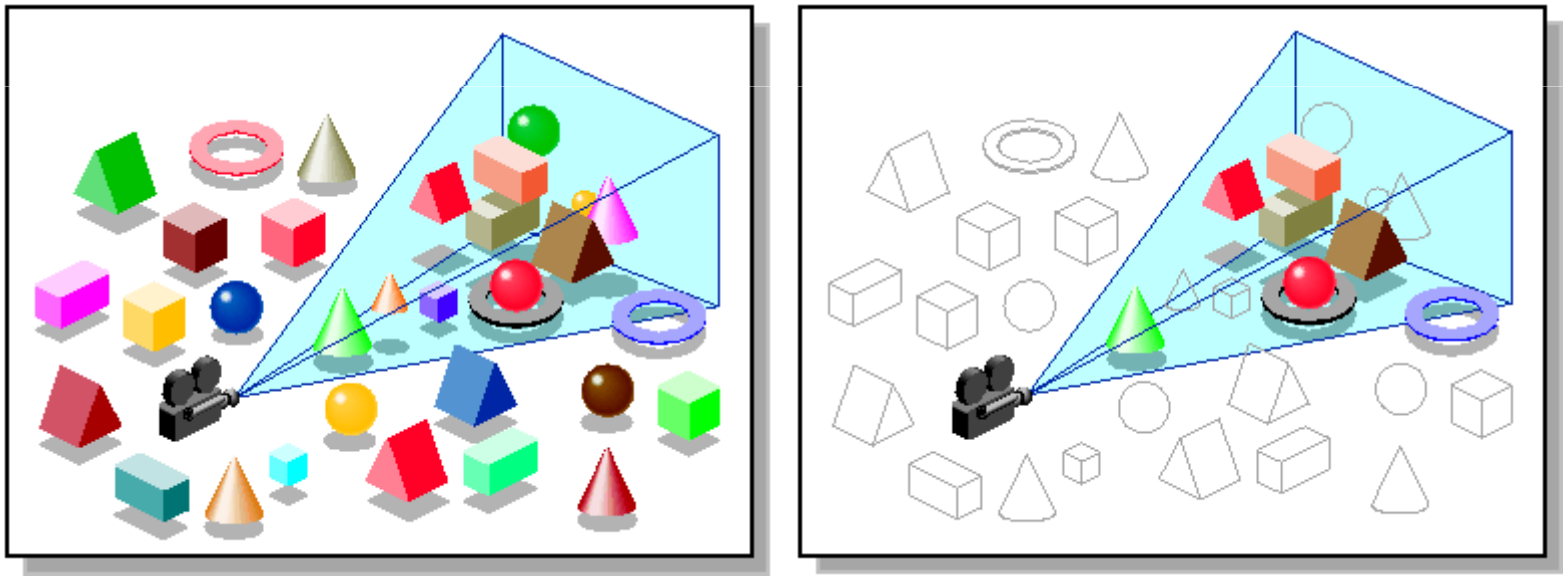
Trabalho de Pesquisa

Descarte por Oclusão
Algoritmo CHC++

Vitor Barata R. B. Barroso
vbarata@tecgraf.puc-rio.br

Descarte por Visibilidade

- Principais tipos de descarte
  - Contra o volume de visão: primitivas fora do campo visual
  - Por oclusão: primitivas escondidas atrás de outras primitivas



Descarte por visibilidade contra volume de visão e por oclusão
(fonte: OpenGL optimizer programmer's guide: an open API for large-model visualization)

Descarte por Oclusão

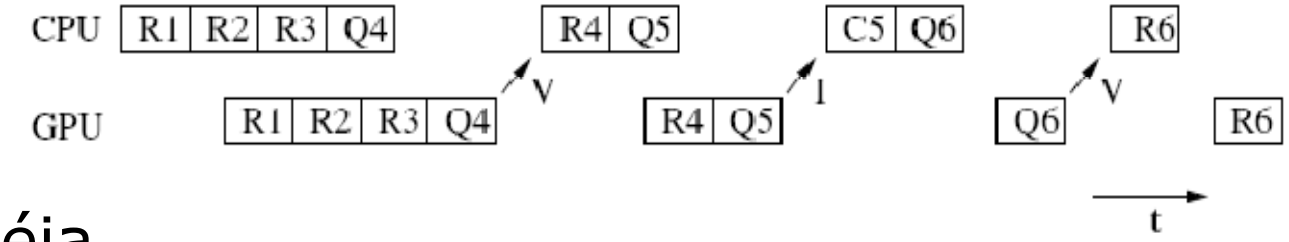
- Processamento
  - *Online* (por ponto): O PVS é determinado dinamicamente, em tempo de execução, para o ponto onde se encontra o observador no momento.
  - *Offline* (por região): Em pré-processamento, determina-se o PVS para vários pontos-de-vista espalhados pela cena. Em tempo de execução, escolhe-se e utiliza-se o PVS do ponto mais próximo.
- Abordagem
  - Exata: Determinam-se exatamente as primitivas visíveis ao observador.
  - Conservativa: Primitivas oclusas podem ser incluídas no PVS para economizar cálculos.
  - Agressiva: Primitivas visíveis podem ser excluídas do PVS.
- Espaço
  - Do objeto: o PVS é determinado a partir de testes envolvendo a geometria dos objetos da cena em seu espaço original. (Fast-V)
  - Da imagem: consideram-se os objetos após a rasterização, ou seja, uma amostragem de suas distâncias até o observador, como a fornecida pelo próprio z-buffer. (CHC++)

Teste de Oclusão em Hardware

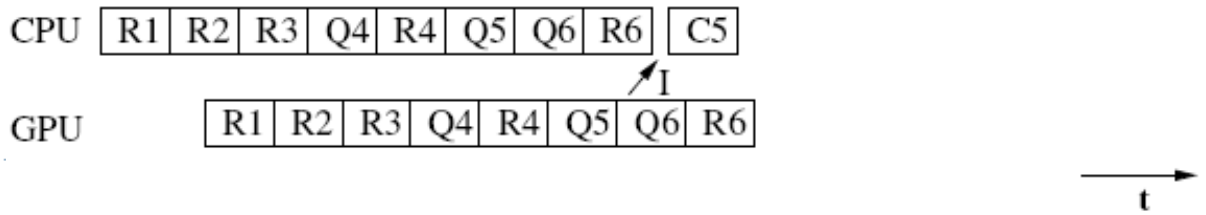
- Permite perguntar à placa gráfica quantos pixels seriam rasterizados numa operação de desenho qualquer
  - Considera teste de z com z-buffer atual
- Operação assíncrona
  - Chamada retorna imediatamente
  - Resultado demora algum tempo para ficar disponível
  - Pode-se consultar rapidamente se o resultado já está pronto
- Problema: ociosidade da CPU e da GPU
  - Se o resultado for requisitado cedo demais, CPU espera até que esteja disponível
  - Falta de novos dados para a GPU deixa-a subutilizada

Descarte Coerente Hierárquico (CHC)

- Ociosidade do CPU e da GPU



- Idéia
  - Explorar a coerência temporal da classificação de visibilidade
    - Visível tende a continuar visível e vice-versa
  - Reorganizar as chamadas
    - Intercalar testes de oclusão com outras operações
    - Enquanto espera, CPU continua processando e enviando dados à GPU



Algoritmo CHC++

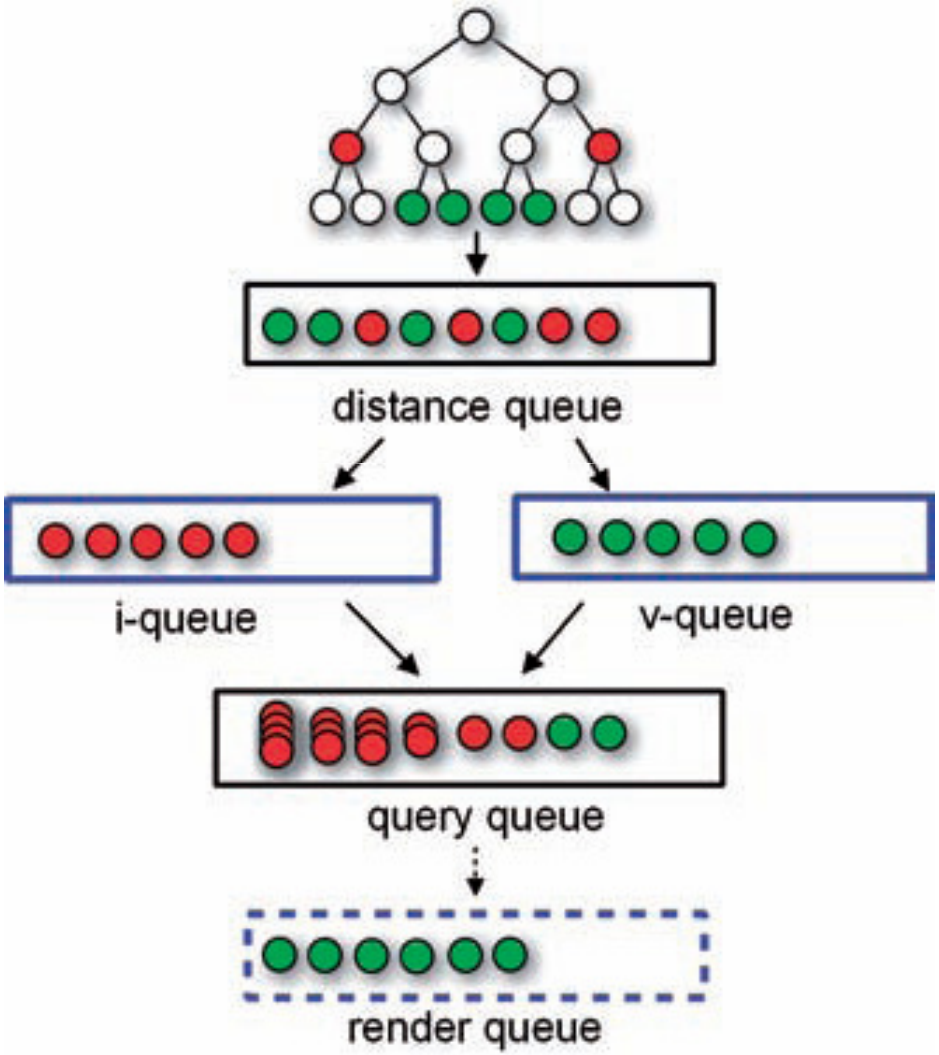
- Testes de visibilidade em lote
  - Minimizar mudanças de estado da renderização
  - Grupos (lotes) de testes são requisitados conjuntamente

- Lotes de nós previamente invisíveis
  - Em vez de requisitar imediatamente, colocar na "fila-i"
  - Requisitar quando o número de nós na fila chegar a um valor *b*

- Lotes de folhas previamente visíveis
  - Em vez de requisitar imediatamente, colocar o teste na fila-v
  - Requisitar quando for preciso esperar por outras operações

- Lotes de nós a renderizar
  - Em vez de desenhar imediatamente, colocar numa fila de render
  - Renderizar toda a fila logo antes de qualquer teste de oclusão

Algoritmo CHC++



Algoritmo CHC++

- Multitestes
  - Agrupar nós previamente invisíveis que têm a mesma probabilidade de permanecerem assim no quadro atual
  - Fazer apenas um teste desenhando todos os nós
    - Se invisível, economiza vários testes
    - Se visível, testa um por um
- CHC: objetos visíveis tendem a continuar visíveis
  - Refazer seus testes de oclusão apenas a cada n_{av} quadros
  - Da primeira vez, escolher n_{av} aleatório entre 0 e um limite
- Volumes envolventes mais justos
  - Uso de AABBs simples
  - Para nós internos, volume é a união dos volumes dos filhos!

Implementação do Projeto

- Ciclo do algoritmo:
  - Resultados já disponíveis
    - Pode acrescentar à fila de prioridades e ao lote-R
  - Nós já visitados e com vez na fila de prioridades
    - Descartar por *frustum*
    - Pode acrescentar aos lotes I, V ou R
  - Processamento do lote-I (inteiro)
    - Quando atingir tamanho determinado
    - Também por necessidade (fila de prioridades ou lote-V vazio)
    - Gera queries e acrescenta nós ao percorrimento
  - Processamento do lote-V (nó a nó)
    - Gastar o tempo necessário para evitar esperar
    - Gera queries
  - Espera por novos resultados

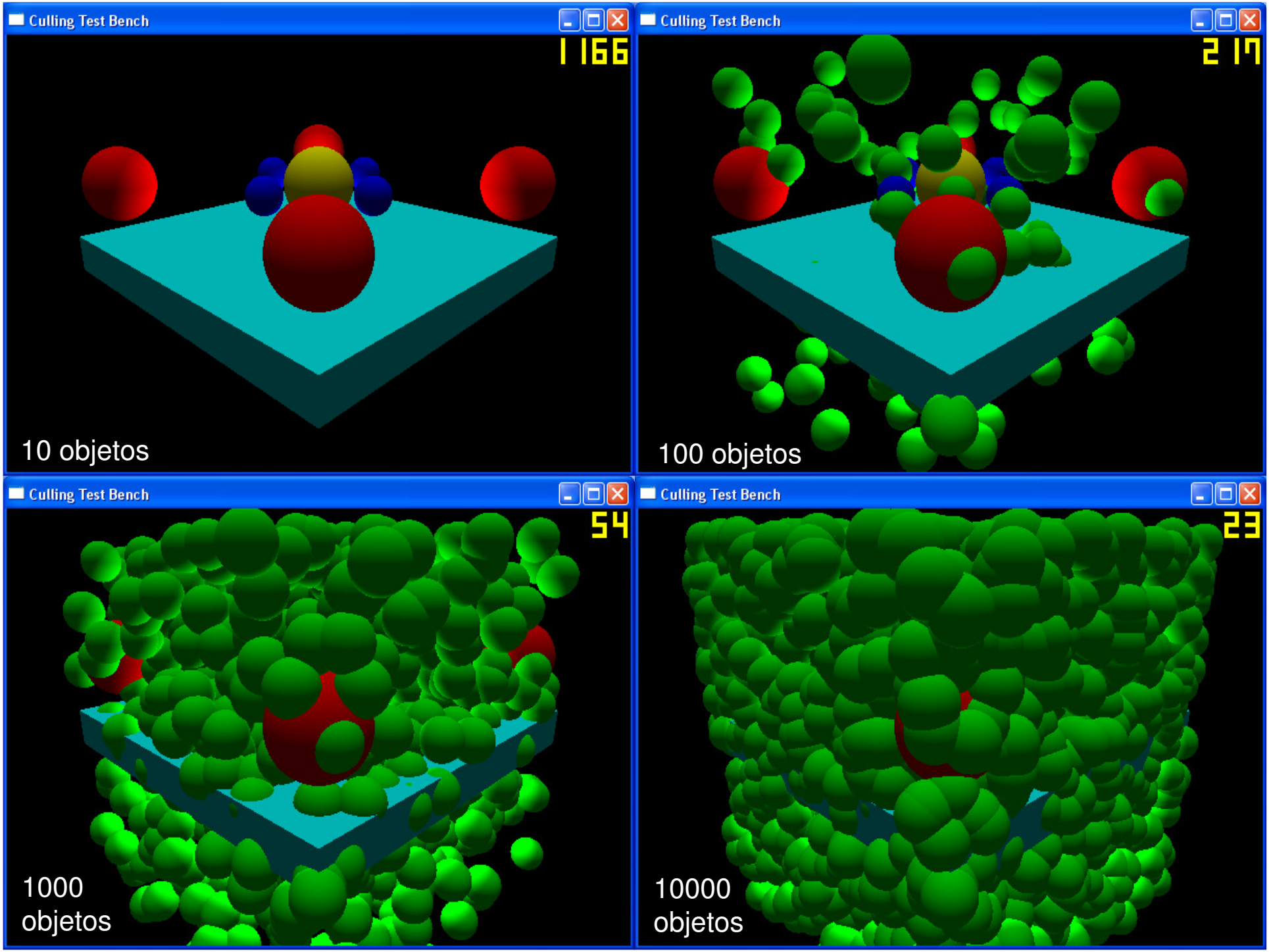
Implementação do Projeto

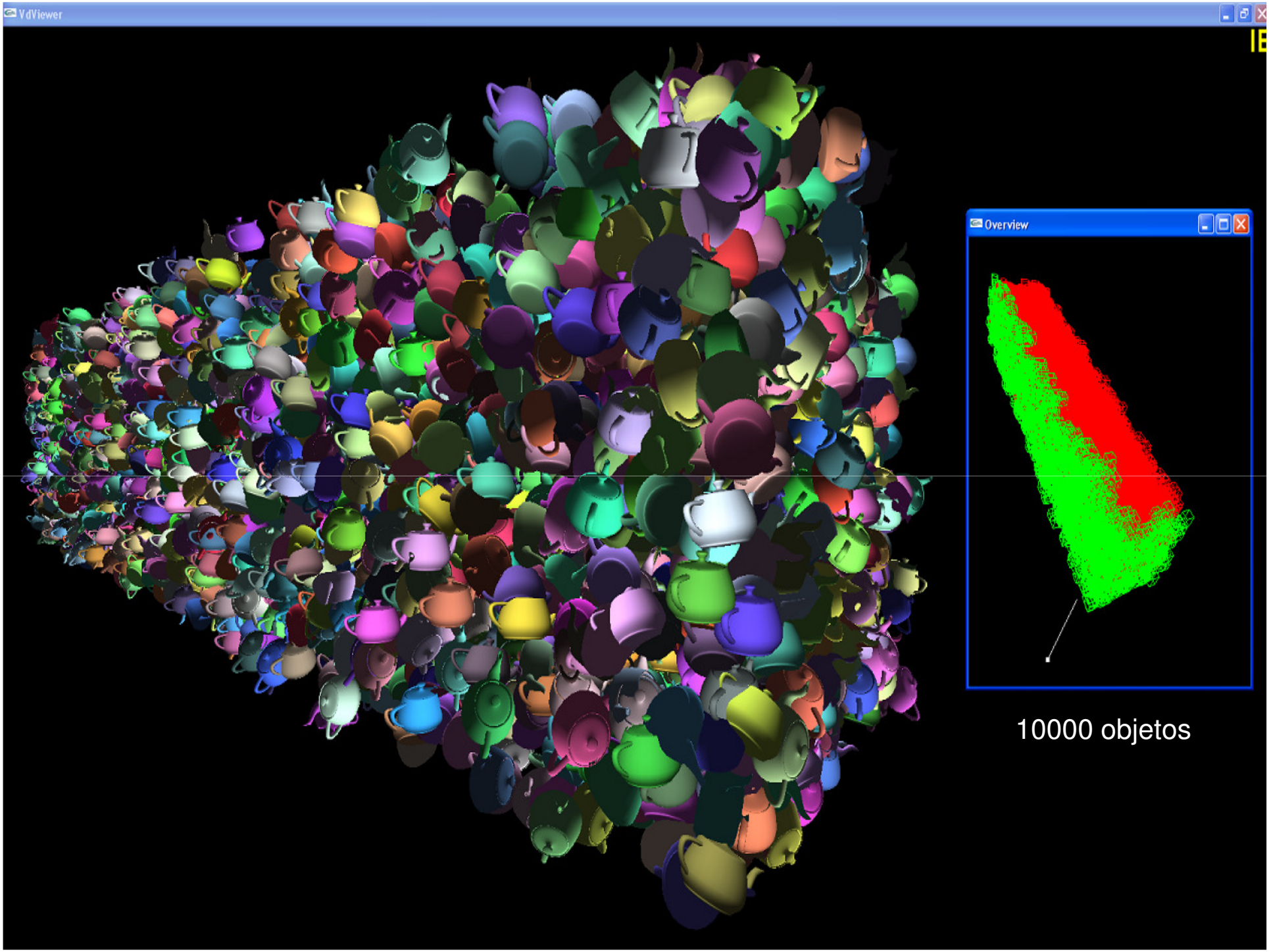
- Problemas não explicados nos artigos
  - Volumes e objetos que atravessam o plano near
  - Brigas de *pull-up* e *pull-down* de visibilidade
- Gerência de objetos *query* do OpenGL
  - *Pool* de objetos
- Multitestes
  - Uso de duas filas
  - Armazenar em cada consulta
    - 1 – ID OpenGL e número de nós do teste
    - 2 – Todos os nós (de todos os testes)
- Integração
  - Grafo de cena preparado apenas para percorrimento em profundidade
  - Custo adicional de manter transformações dos nós internos
    - Opção: “achatar” a parte estática da cena

Resultados

Cena de Teste	VFC	CHC	CHC++
100 objetos médios	1140 Fps	1390 Fps	1750 Fps
1000 objetos médios	162 Fps	395 Fps	510 Fps
1000 objetos pequenos	185 Fps	123 Fps	220 Fps
10000 objetos pequenos	19 Fps	79 Fps	151 Fps

Grafo de Cena	VFC	CHC++
10 objetos	1290 Fps	1175 Fps
100 objetos	209 Fps	239 Fps
1000 objetos	22 Fps	54 Fps
10000 objetos	4 Fps	23 Fps


Culling Test Bench
1166
10 objetos
Culling Test Bench
217
100 objetos
Culling Test Bench
54
1000
objetos
Culling Test Bench
23
10000
objetos



10000 objetos

Conclusão

- Descarte por visibilidade é uma técnica fundamental para a visualização de modelos, desde de médio porte até massivos
- Descarte por oclusão complementa, mas **não** descarta, descartes por volume de visão e portais
- CHC é eficaz em cenas com alto nível de oclusão
- CHC++ é sempre mais eficiente
  - Dificilmente mais lento do que desligado
  - Mais de 100% de ganho de FPS em cenas muita oclusão
  - Melhor integrável a renderizadores eficientes

Referências

[1] P. Santos, **Técnicas de Otimização para Visualização de Modelos Massivos** – Relatório Final de Projeto, PUC-Rio, 2006

[2] J Bittner,M Wimmer, H. Piringer, W. Purgathofer, **Coherent Hierarchical Culling: Hardware Occlusion Made Useful** – Proceedings of Eurographics 2004

[3] O. Mattausch, J. Bittner, M. Wimmer, **CHC++: Coherent Hierarchical Culling Revisited** – Proceedings of Eurographics 2008