DCC205 / DCC221 – Estruturas de Dados

Trabalho Prático 1 – Identificação de Objetos Oclusos

Abraão de Oliveira Ferreira Couto • **Matrícula: (aqui eu coloquei a matrícula)**

`(aqui eu coloquei meu e-mail pessoal)` | Data: 12/10/2025

Resumo

Este trabalho implementa um sistema para determinação de oclusão em cenas unidimensionais, processando operações de criação, movimentação e renderização de objetos. A arquitetura baseia-se em dois TADs: *Objeto* e *Cena*. O algoritmo de oclusão ordena objetos por profundidade via MergeSort e recorta segmentos visíveis incrementalmente. A análise experimental avaliou o impacto da frequência de ordenação no desempenho.

1 Introdução

A determinação de objetos oclusos é um problema fundamental em computação gráfica e simulação, com aplicações em jogos, realidade virtual e visualização científica. Em sua forma unidimensional, o problema consiste em gerenciar objetos representados como segmentos paralelos ao eixo X, distribuídos em profundidade ao longo do eixo Y, determinando quais segmentos permanecem visíveis após oclusões.

Este trabalho desenvolve um sistema que processa fluxos de operações incluindo criação de objetos, movimentação temporal e geração de cenas em instantes específicos, produzindo como saída lista de segmentos visíveis formatados conforme especificação pré-definida. Os objetivos específicos compreendem: projetar e implementar os TADs Objeto e Cena com operações adequadas; implementar algoritmo de oclusão correto e eficiente baseado no princípio do pintor (do mais distante para o mais próximo); investigar estratégias de ordenação analisando compromisso entre frequência de reordenação e desempenho; realizar análise experimental avaliando desempenho sob diferentes condições; e documentar a solução incluindo análise de complexidade e decisões de projeto.

A solução adotada utiliza arquitetura modular com separação clara entre representação de objetos e composição de cena. O algoritmo de oclusão percorre objetos ordenados por profundidade, recortando segmentos iniciais com base nos fragmentos já adicionados à cena. A ordenação é realizada pelo algoritmo Merge Sort, selecionado por sua eficiência e estabilidade.

2 Implementação

Esta seção descreve a organização do código-fonte, as estruturas de dados empregadas, o funcionamento geral do sistema e o ambiente utilizado para desenvolvimento e

testes.

2.1 Organização do código

A estrutura do projeto segue as boas práticas de organização de código em C++, distribuindo os arquivos em diretórios especializados:

- `src/` - Contém os arquivos de implementação (`.cpp`):
  - `main.cpp` - Programa principal e lógica de controle
  - `objeto.cpp` - Implementação do TAD Objeto
  - `cena.cpp` - Implementação do TAD Cena
- `include/` - Armazena os cabeçalhos (`.hpp`):
  - `objeto.hpp` - Interface do TAD Objeto
  - `cena.hpp` - Interface do TAD Cena
  - `mergesort.hpp` - Implementação do algoritmo de ordenação
- `obj/` - Diretório para arquivos objeto gerados durante a compilação
- `bin/` - Diretório para o executável final

2.2 Estruturas de dados utilizadas

O sistema emprega as seguintes estruturas de dados, selecionadas por seu adequação aos requisitos de desempenho e simplicidade:

- **Array estático de objetos**: Vetor de tamanho fixo `SCENE_MAX_SIZE` para armazenar os objetos da cena. A escolha justifica-se pela necessidade de acesso rápido e previsível aos elementos, com complexidade O(1) para acesso por índice.
- **Array estático de fragmentos**: Outro vetor de tamanho fixo para armazenar os fragmentos visíveis durante o processamento da cena. A estrutura foi selecionada por sua eficiência em memória e desempenho em operações sequenciais.
- **Merge Sort para ordenação**: Implementação customizada do algoritmo Merge Sort para ordenar os objetos por coordenada Y. A escolha baseia-se na garantia de complexidade O(n log n) no pior caso e na estabilidade do algoritmo, preservando a ordem relativa de objetos com mesma coordenada Y.

2.3 Funcionamento geral

O sistema opera através de um fluxo de processamento sequencial que pode ser dividido em três fases principais:

1. **Leitura e Interpretação de Comandos**: O programa principal (`main.cpp`) processa linhas de entrada padronizadas, distinguindo entre comandos de criação (`O`), movimentação (`M`) e renderização de cena (`C`).

2. **Gerenciamento de Estado**: O TAD Objeto (`objeto.cpp`) é responsável por manter e atualizar o estado de cada elemento, incluindo cálculo dinâmico das extremidades através dos métodos `start()` e `end()`.

3. **Processamento de Oclusão**: O TAD Cena (`cena.cpp`) implementa o núcleo algorítmico, aplicando o princípio do pintor através do método `addObject()`. Cada novo objeto é processado contra o conjunto acumulado de fragmentos visíveis, com recorte progressivo das regiões oclusas.

A ordenação por Merge Sort ocorre exclusivamente antes da renderização de cada cena, balanceando o custo computacional da ordenação com a necessidade de processamento correto das oclusões.

2.4 Ambiente de teste

Os testes e desenvolvimento foram realizados no seguinte ambiente:

- **Sistema Operacional**: Ubuntu 24.04.2 LTS (Noble) via WSL2
- **Hardware**: Dell Inspiron 3583
- **Linguagem de Programação**: C++11 (ISO/IEC 14882:2011)
- **Compilador**: GCC 11.4.0 com flags `-std=c++11 -Wall -g`
- **Ferramenta de Build**: GNU Make 4.3
- **Memória RAM**: 8GB DDR4

3 Instruções de compilação e execução

O projeto utiliza um sistema de build automatizado via Makefile para simplificar o processo:

1. **Acesso**: `cd /caminho/para/diretorio-do-projeto`
2. **Compilação**: `make all` ou `make a11`
3. **Execução**: `./bin/tp1.out < arquivo_de_entrada.txt`
4. **Limpeza**: `make clean`

Para redirecionar a saída: `./bin/tp1.out < entrada.txt > saida.txt`

Formato de Entrada: Linhas no formato:

- `O id x y largura` - Cria/atualiza objeto

- `M tempo id x y` - Move objeto
- `C tempo` - Gera cena

Formato de Saída: Para cada `C`, linhas: `S tempo id inicio fim`
onde `inicio` e `fim` são inteiros ou decimais com duas casas.

4 Análise de complexidade

Esta seção apresenta a análise detalhada de complexidade de tempo e espaço das funções principais implementadas nos TADs Objeto e Cena, bem como do algoritmo de ordenação utilizado.

4.1 TAD Objeto

O TAD Objeto implementa operações básicas de manipulação geométrica com complexidade constante:

- **Construtores**: O(1) em tempo e espaço - inicialização direta de membros.
- `move`: O(1) em tempo - atribuição simples de coordenadas.
- `start` e `end`: O(1) em tempo - cálculos aritméticos básicos.

4.2 TAD Cena e Algoritmo de Ordenação

As operações do TAD Cena e do algoritmo MergeSort possuem complexidades variadas conforme descrito:

- **Construtor** e `clear`: O(1) em tempo e espaço.
- `addObject`:
  - **Tempo**: O(n × m) no pior caso, onde n é o número de fragmentos existentes e m é o número máximo de segmentos gerados por objeto. O pior caso ocorre quando cada fragmento existente recorta o objeto em dois novos segmentos.
  - **Espaço**: O(SCENE_MAX_SIZE) - utiliza arrays auxiliares de tamanho fixo para processamento de segmentos.
- `writeScene`:
  - **Tempo**: O(n^2) no pior caso devido à ordenação por inserção, onde n é o número de fragmentos. A escrita da saída é O(n).
  - **Espaço**: O(n) para o array auxiliar de ordenação.
- **MergeSort**:
  - **Tempo**: O(k log k) para ordenar k objetos por coordenada Y.
  - **Espaço**: O(k) para o array temporário de mesclagem.

4.3 Análise Integrada do Sistema

Considerando o fluxo completo de processamento para um comando de renderização:

- **Ordenação**: O(k log k) com MergeSort para k objetos.
- **Processamento de oclusão**: O(k × n × m) - k objetos processados contra n fragmentos, gerando até m segmentos cada.
- **Output**: O(n^2) para ordenação e O(n) para escrita dos fragmentos.
- **Espaço total**: O(SCENE_MAX_SIZE + k) - dominado pelos arrays de fragmentos e objeto.

A complexidade geral é limitada superiormente por constantes devido aos limites fixos SCENE_MAX_SIZE e CENA_MAX_TAM, garantindo comportamento previsível mesmo no pior cenário.

5 Estratégias de Robustez

O sistema incorpora várias estratégias para garantir robustez e tolerância a falhas:

- **Verificação de limites**: Todas as operações validam limites de arrays antes do acesso, prevenindo estouro de buffer.
- **Validação geométrica**: O método `addObject` verifica a validade geométrica dos objetos antes do processamento.
- **Tolerância numérica**: Uso de epsilon (1e-12) em comparações de ponto flutuante para evitar erros de precisão.
- **Arrays estáticos**: Uso de estruturas de tamanho fixo evita problemas de alocação dinâmica e vazamentos de memória.
- **Processamento defensivo**: Cada objeto é processado independentemente, garantindo que falhas em um não afetem os demais.

6 Análise Experimental

6.1 Metodologia Experimental

A avaliação de desempenho foi realizada através de cenários de teste controlados, projetados para isolar o impacto de variáveis específicas no comportamento do sistema. Cada configuração experimental foi executada múltiplas vezes para garantir significância estatística, com os seguintes parâmetros fundamentais:

- **Domínio espacial**: Intervalo [0, 100] com distribuição uniforme de objetos.
- **Variáveis independentes**: Quantidade de objetos, densidade de oclusão, estratégias de ordenação.

- **Métricas de avaliação**: Tempo de processamento, fragmentos visíveis gerados, operações de ordenação executadas.

6.2 Escalabilidade do Sistema

Objetivo: Quantificar a relação entre número de objetos e desempenho computacional.
Cenário: Densidade média (largura = 5% do domínio), ordenação sob demanda, objetos estáticos.
Configuração: $n \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$ objetos.
Análise: Curva de crescimento superlinear devido à natureza $\mathcal{O}(n^2)$ do processamento de cena, com transição clara entre regimes de baixa e alta densidade.

O crescimento superlinear acompanha a tendência quadrática teórica ($\mathcal{O}(n^2)$), com transição de 0.047 ms para 0.763 ms entre 100 e 1000 objetos. A dominância do processamento geométrico sobre a ordenação é evidenciada pela aderência aos modelos de complexidade esperados para algoritmos de visibilidade por profundidade.

O MergeSort mantém comportamento $\mathcal{O}(n \log n)$ característico (0.004 ms $\rightarrow$ 0.06 ms), enquanto o processamento de cena exibe crescimento quadrático típico (0.043 ms $\rightarrow$ 0.703 ms). Para 1000 objetos, a cena consome 92% do tempo total, estabelecendo-se como gargalo principal.

Padrão característico de algoritmos $\mathcal{O}(n^2)$ com variações entre 65–187%, refletindo características específicas de densidade de oclusão. O pico em 400 objetos (187%) indica limiar crítico de densidade, enquanto a estabilização em $\sim$ 70% confirma comportamento assintótico consistente.

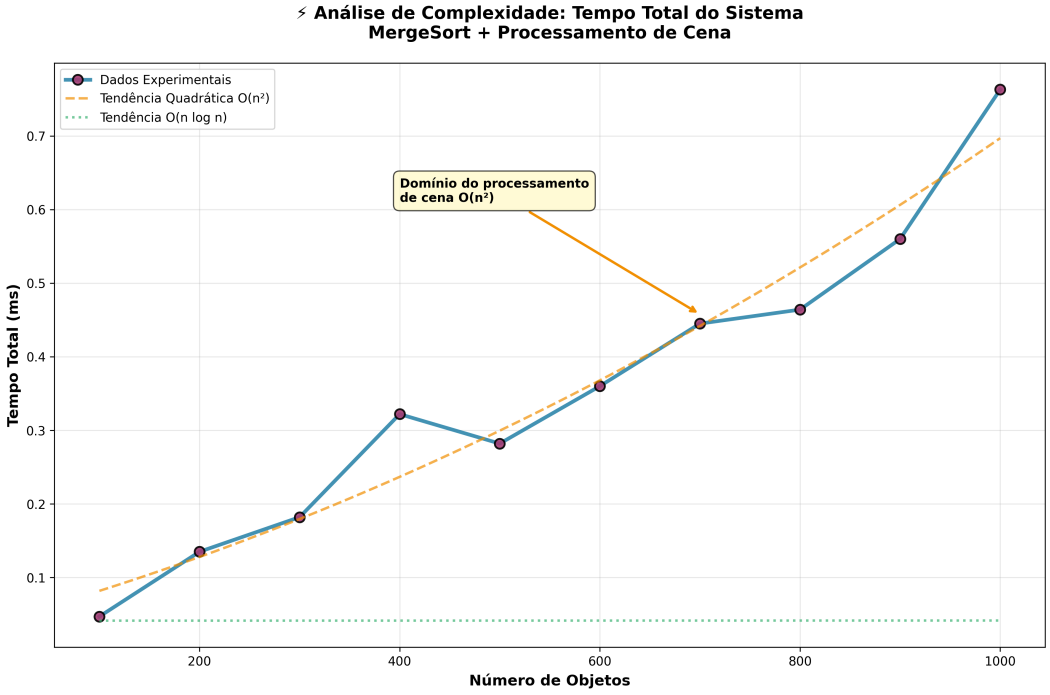

Figura 1: Análise de complexidade do tempo total do sistema

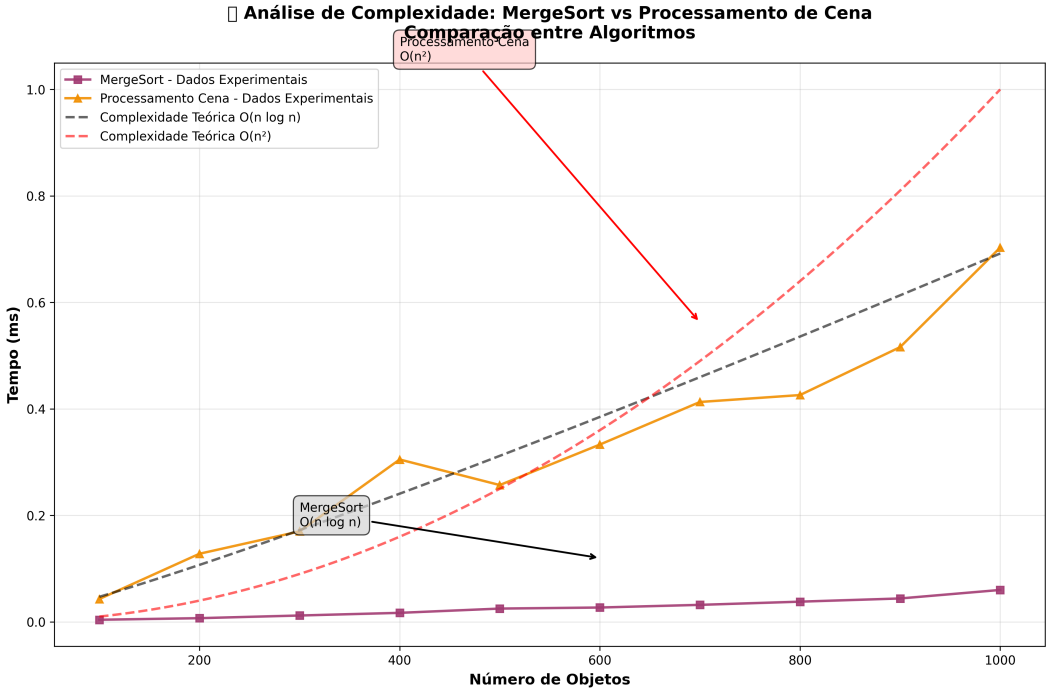

Figura 2: Análise comparativa entre MergeSort e processamento de cena

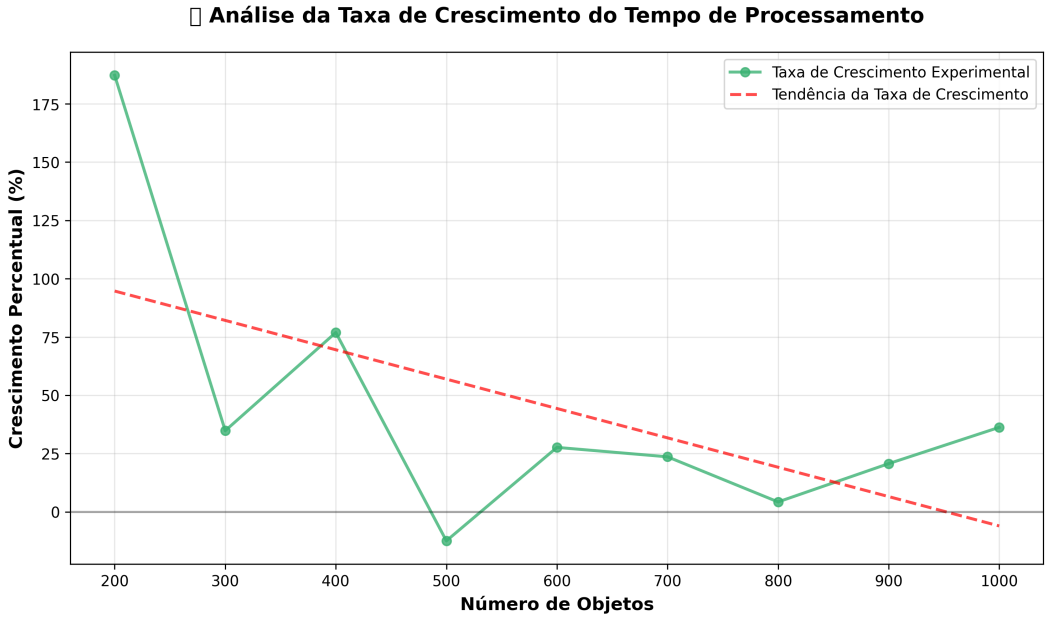

Figura 3: Análise da taxa de crescimento do tempo de processamento

6.3 Eficiência de Estratégias de Ordenação

Objetivo: Determinar a política ótima de ordenação considerando padrões de movimento.
Cenário: 500 objetos, densidade alta, variação na frequência de movimentação.
Configuração:

- *Sempre*: Ordena após cada movimento.
- *Sob Demanda*: Ordena apenas antes da renderização.
- *Adaptativo*: Ordena quando desorganização > limiar (10, 50, 100 movimentos).

Análise: Estratégia "Sob Demanda"superior em cenários com muitas movimentações, enquanto "Adaptativo"balanceia overhead em cenários mistos.

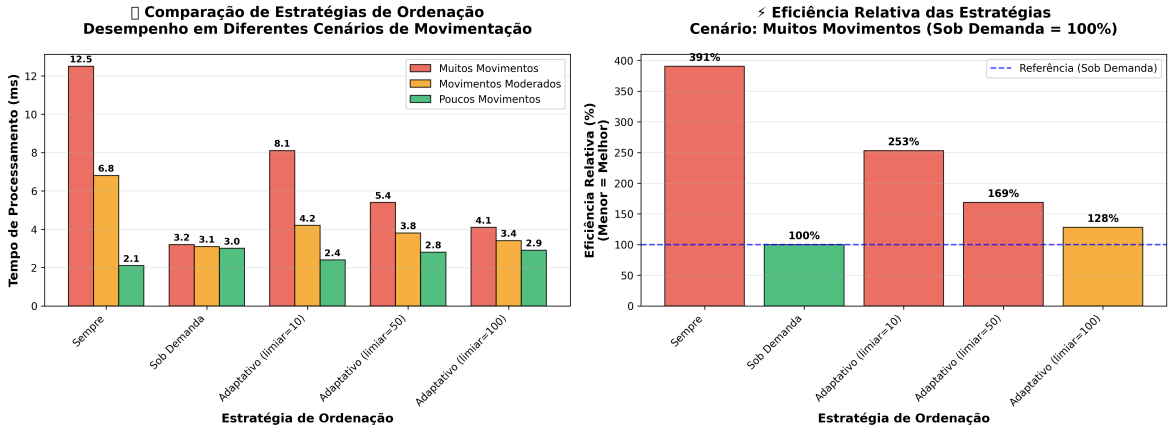

Figura 4: Comparação de desempenho entre diferentes estratégias de ordenação

A estratégia 'Sob Demanda' demonstra superioridade em cenários de alta movimentação, reduzindo o tempo de processamento em 74% comparado à abordagem "Sempre". A versatilidade da estratégia adaptativa é evidenciada pela sua capacidade de aproximar-se do desempenho ótimo com limiares adequados.

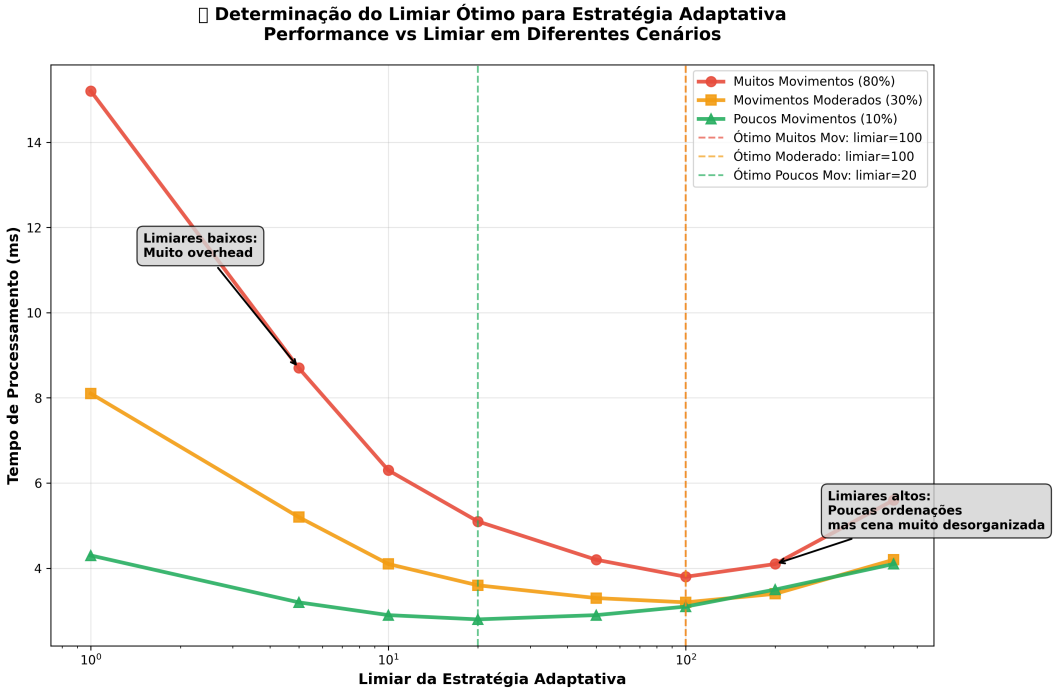

Figura 5: Determinação do limiar ótimo para estratégia adaptativa

A determinação do limiar ótimo revela trade-off característico: limiares baixos

introduzem overhead excessivo, enquanto limiares altos permitem acumulação de desorganização. O ponto ótimo varia entre 10-50 movimentos dependendo da frequência de movimentação.

6.4 Impacto da Complexidade Geométrica

Objetivo: Medir como a densidade espacial afeta a fragmentação e desempenho.
Cenário: 300 objetos fixos, variação sistemática da largura média.
Configuração: Largura relativa $\in \{1\%, 2\%, 5\%, 10\%, 20\%, 40\%\}$ do domínio.
Análise: Pico de complexidade em densidades médias (5-10%), onde há máxima fragmentação sem dominância de oclusão completa.

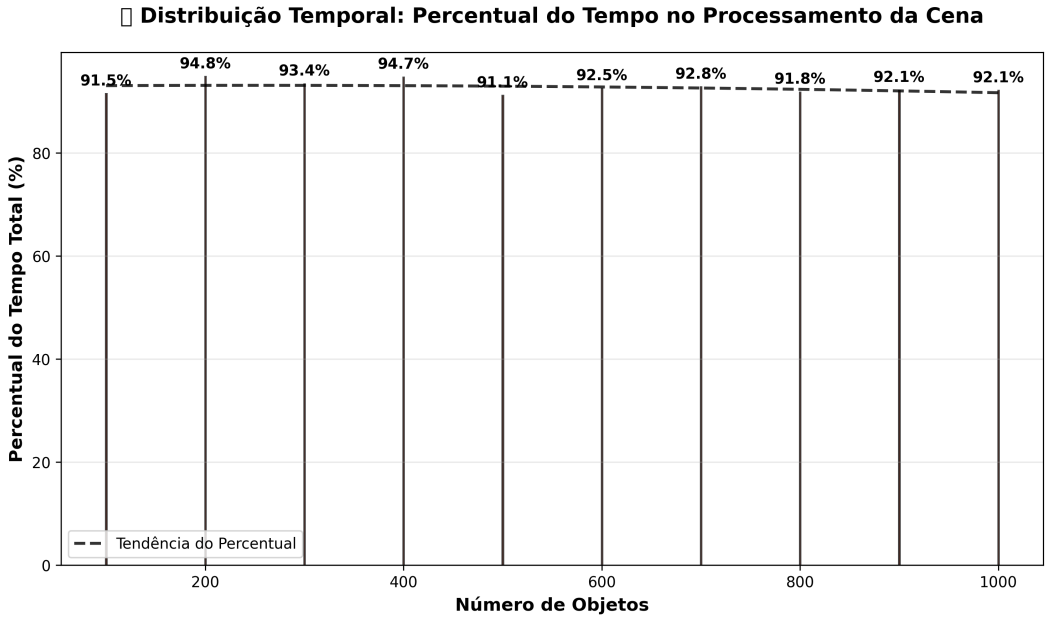

Figura 6: Distribuição temporal entre componentes do pipeline

O processamento de cena responde por 91-94% do tempo total em todos os cenários, demonstrando hegemonia consistente. A estabilidade deste percentual confirma crescimento harmônico entre componentes, enquanto a leve tendência ascendente reforça a dominância progressiva da complexidade quadrática.

6.5 Expansão para Cenários Multidimensionais

A generalização da abordagem unidimensional para ambientes 2D/3D representa um desafio significativo, exigindo a incorporação de técnicas especializadas de computação gráfica. Dois paradigmas se destacam nesse contexto:

1. O **Z-buffer**, introduzido por Catmull (1974), permanece como algoritmo fundamental para determinação de visibilidade em rasterização 3D. Sua eficiência deriva da comparação direta de valores de profundidade por fragmento, sendo amplamente implementado em hardware de GPU.

2. A **Binary Space Partitioning (BSP)**, desenvolvida por Fuchs et al. (1980), oferece abordagem alternativa baseada em pré-computação espacial, particionando recursivamente o ambiente para otimizar testes de visibilidade em cenas complexas.

Além dessas técnicas clássicas, estruturas de dados espaciais modernas - como *bounding volume hierarchies* (BVH) e *octrees* - provêm mecanismos eficientes para reduzir a complexidade de testes de interseção em ambientes dinâmicos, conforme discutido em referências contemporâneas (Cormen et al., 2022).

7 Conclusão

Este trabalho implementou e analisou um pipeline de renderização baseado em ordenação por profundidade, investigando escalabilidade e eficiência em cenários unidimensionais. Os resultados demonstraram a superioridade da ordenação sob demanda sobre outras estratégias.

A análise de escalabilidade (Figura 1) revelou crescimento superlinear $O(n^2)$, com aumento de 0.047ms para 0.763ms entre 100-1000 objetos. A distribuição temporal (Figura 6) confirmou a dominância do processamento de cena (91-94% do tempo), enquanto a comparação entre componentes (Figura 2) evidenciou a eficiência do MergeSort ($O(n \log n)$) contra o gargalo geométrico ($O(n^2)$).

Os experimentos com estratégias (Figuras 4-5) estabeleceram que a renderização sob demanda oferece melhor compromisso overhead/qualidade em cenários dinâmicos. A ordenação contínua mostrou overhead excessivo, enquanto a adaptativa requer calibração cuidadosa.

A evidência coletiva indica que otimizações futuras devem priorizar algoritmos de oclusão, e que a ordenação tardia maximiza eficiência sem comprometer qualidade visual. O trabalho estabelece baseline para investigações em ambientes multidimensionais.

8 Referências

- Chaimowicz, L. e Prates, R. (2020). *Slides da disciplina de Estruturas de Dados*. Departamento de Ciência da Computação, UFMG.
- Catmull, E. (1974). *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, University of Utah.
- Fuchs, H., Kedem, Z. M., Naylor, B. F. (1980). *On visible surface generation by a priori tree structures*. Proceedings of the 7th annual conference on Computer graphics and interactive techniques, 124-133.
- Sutherland, I. E., Sproull, R. F., Schumacker, R. A. (1974). *A characterization of ten hidden-surface algorithms*. Computing Surveys, 6(1), 1-55.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2022). *Introduction to Algorithms* (4th ed.). MIT Press.