Arquitetura de Computadores e Otimização de Recursos em Jogos Clássicos: Uma Análise do Pipeline Gráfico de Wolfenstein 3D e sua Influência em DOOM

Computer Architecture and Resource Optimization in Classic Games: An Analysis of Wolfenstein 3D's Graphics Pipeline and its Influence on DOOM

Cauã Cola[1], Davi Ribeiro[2], Giseli Lourenço[1], Guilherme Fagundes[3], Matheus Miranda[4]

[1]Universidade Vila Velha (UVV)
Vila Velha – ES – Brasil

Abstract. Introduction*:The development of digital games in the 1990s represented a period of intense innovation driven by severe hardware limitations. In a context where processors had low computing power and memory was extremely limited, efficient solutions became not only desirable but essential.*

In this scenario, games like Wolfenstein 3D established new paradigms by introducing innovative rendering and resource management techniques. Such approaches demonstrate a strong integration between software and computer architecture, directly exploiting hardware features to maximize performance.

This paper aims to analyze the techniques used in the development of Wolfenstein 3D, focusing on the programming environment, the representation of graphic assets, and the data processing pipeline. Furthermore, it seeks to relate these solutions to fundamental concepts of computer architecture and discuss their influence on subsequent games, such as DOOM.

In this way, it intends to demonstrate how technological limitations can drive significant advances, contributing to the evolution of real-time computer graphics.

Keywords *Computer Architecture, Computer Graphics, Ray Casting, Wolfenstein 3D, Game Engines.*

Resumo. Introdução: *O desenvolvimento de jogos digitais na década de 1990 representou um período de intensa inovação impulsionada por severas limitações de hardware. Em um contexto onde processadores possuíam baixo poder computacional e a memória era extremamente restrita, soluções eficientes tornaram-se não apenas desejáveis, mas essenciais.*

Nesse cenário, jogos como Wolfenstein 3D estabeleceram novos paradigmas ao introduzir técnicas inovadoras de renderização e gerenciamento de recursos. Tais abordagens evidenciam uma forte integração entre software e arquitetura de computadores, explorando diretamente características do hardware para maximizar desempenho.

Este trabalho tem como objetivo analisar as técnicas utilizadas no desenvolvimento de Wolfenstein 3D, com foco no ambiente de programação, na representação de assets gráficos e no pipeline de processamento de dados. Além disso, busca-se relacionar essas soluções com conceitos fundamentais da

arquitetura de computadores e discutir sua influência em jogos subsequentes, como DOOM.
Dessa forma, pretende-se demonstrar como limitações tecnológicas podem impulsionar avanços significativos, contribuindo para a evolução da computação gráfica em tempo real.
Palavras-Chave *Arquitetura de Computadores, Computação Gráfica, Ray Casting, Wolfenstein 3D, Motores de Jogos.*

1. Técnicas Utilizadas em Wolfenstein 3D

1.1. Programação e Ambiente de Desenvolvimento

O desenvolvimento de Wolfenstein 3D ocorreu em um contexto de limitações computacionais significativas, utilizando computadores equipados com processadores 386-DX de 33MHz e aproximadamente 4 MiB de memória RAM. Nesse cenário, a eficiência do código constituía um requisito essencial para garantir a viabilidade da execução em tempo real.

A linguagem utilizada foi predominantemente C, com o suporte do ambiente de desenvolvimento Borland C++ 3.1, que integrava ferramentas de compilação, execução e depuração. Essa integração contribuiu para um fluxo de desenvolvimento mais consistente, reduzindo a necessidade de ferramentas externas e permitindo maior controle sobre o processo de construção do software.

A divisão de responsabilidades entre os desenvolvedores também desempenhou papel relevante. John Carmack foi responsável pelo desenvolvimento do código principal de execução, enquanto John Romero implementou ferramentas fundamentais, como o editor de mapas TED5 e utilitários de empacotamento de recursos. Outros subsistemas, incluindo gerenciamento de entrada, memória e áudio, foram desenvolvidos por Jason Blochowiak.

Uma solução técnica adotada para contornar as limitações de visualização durante o processo de depuração consistiu na utilização de dois monitores simultaneamente. Essa estratégia explorava características da arquitetura x86, que permitia a coexistência de diferentes padrões de vídeo. Enquanto a placa VGA operava em modo gráfico (Modo 13h), utilizando o endereço de memória 0xA0000 para o framebuffer, um segundo monitor baseado em MDA (Monochrome Display Adapter) operava em modo texto, acessando o endereço 0xB8000.

Essa separação possibilitava a execução do jogo em modo gráfico sem interferências, ao mesmo tempo em que ferramentas como o Turbo Debugger 386 exibiam informações de depuração, como registradores e variáveis, no monitor secundário. Tal abordagem evidencia um uso direto dos recursos da arquitetura de computadores, permitindo maior eficiência no processo de desenvolvimento e análise do sistema.

1.2. Criação e Representação de Assets Gráficos

Os recursos gráficos de Wolfenstein 3D foram inteiramente produzidos manualmente, utilizando o software Deluxe Paint. Todos os elementos visuais eram armazenados no formato ILBM (InterLeaved BitMap), um padrão proprietário amplamente utilizado em sistemas gráficos da época.

Devido às limitações do hardware VGA, a representação de cores não era realizada em formato RGB de 24 bits, mas por meio de uma paleta indexada de 256 cores, implementada como uma Color Look-Up Table (CLUT). Nesse modelo, cada pixel armazenava apenas um índice que referenciava uma entrada na tabela de cores, reduzindo significativamente o consumo de memória, porém impondo restrições ao processo criativo.

Dessa forma, a definição da paleta de cores constituía uma etapa fundamental no desenvolvimento dos assets, exigindo planejamento prévio para garantir consistência visual em todos os elementos do jogo. Essa abordagem evidencia um compromisso direto entre qualidade gráfica e eficiência no uso de recursos computacionais.

Além disso, a utilização de um valor específico como transparência (0xFF) permitia que o mecanismo de renderização ignorasse determinados pixels durante o processo de desenho, viabilizando a composição de sprites e a sobreposição de elementos na cena sem a necessidade de operações adicionais de mascaramento.

Outro aspecto relevante refere-se à resolução de 320x200 pixels, diretamente associada ao Modo 13h do VGA. Essa resolução não correspondia a uma proporção de tela quadrada, resultando em pixels não uniformes (non-square pixels). Dessa forma, os assets precisavam ser desenhados considerando essa distorção, de modo que a imagem final fosse exibida corretamente após a conversão pelo hardware.

Esse conjunto de restrições e soluções evidencia a forte dependência entre software e hardware, característica marcante do desenvolvimento de jogos na época, onde decisões de implementação estavam diretamente condicionadas à arquitetura do sistema.

1.3. Pipeline de Assets e Otimização

O fluxo de processamento dos assets gráficos foi estruturado com o objetivo de maximizar eficiência no acesso aos dados e flexibilidade na organização dos recursos. Os assets eram divididos em duas categorias principais: elementos de interface bidimensional, armazenados em arquivos como VGAGRAPH, e elementos utilizados na fase de renderização tridimensional, como paredes e sprites, organizados no arquivo VSWAP.

Após a etapa de criação, os arquivos gráficos eram processados por uma ferramenta denominada IGRAB, responsável por empacotar os dados em estruturas binárias otimizadas e gerar automaticamente identificadores únicos para cada recurso. Esses identificadores eram posteriormente utilizados no código-fonte como referências indiretas aos assets.

Essa abordagem introduz um nível de abstração baseado em indireção, no qual os recursos não são acessados diretamente por seus endereços físicos na memória, mas por meio de índices que apontam para tabelas de offsets. Essas tabelas armazenam a posição exata de cada asset dentro dos arquivos binários, permitindo que o sistema localize os dados dinamicamente em tempo de execução.

Do ponto de vista da arquitetura de computadores, o uso de offsets representa uma estratégia eficiente de organização de memória, reduzindo a necessidade de realocação de dados e permitindo maior controle sobre o layout dos arquivos. Além disso, essa técnica possibilita que os assets sejam reorganizados, compactados ou atualizados sem a necessidade de alterações no código-fonte, desde que os índices sejam mantidos

consistentes.

Esse modelo de gerenciamento de recursos apresenta similaridades com sistemas modernos de asset management e pipelines de jogos, evidenciando que conceitos fundamentais de engenharia de software e organização de memória já eram aplicados, mesmo em ambientes com recursos computacionais extremamente limitados.

1.4. Fluxo de Trabalho e Camada de Indireção

A organização dos ativos gráficos no motor de *Wolfenstein 3D* era segmentada em duas categorias lógicas, com o objetivo de otimizar o acesso aos dados durante diferentes estados de execução do jogo. A primeira categoria compreendia os elementos de interface e menus (2D), distribuídos nos arquivos `VGAGRAPH`, `VGAHEAD` e `VGADICT`. A segunda categoria englobava os elementos da fase de ação (3D), como texturas de paredes e *sprites*, armazenados no arquivo de troca `VSWAP`.

O processamento desses ativos era centralizado na ferramenta **IGRAB**, responsável por consolidar os arquivos originais no formato ILBM em arquivos binários de recursos. Simultaneamente, a ferramenta gerava um arquivo de cabeçalho (*header*) em linguagem C contendo identificadores exclusivos (*IDs*) para cada recurso.

No código-fonte do motor, a referência aos elementos gráficos não era realizada por nomes de arquivos ou endereços fixos, mas por meio de tipos enumerados (*enums*). Essa arquitetura estabelecia uma camada de indireção baseada em múltiplos níveis: o identificador (*enum*) atuava como índice para a tabela `HEAD`, que, por sua vez, fornecia o *offset* necessário para localizar o dado correspondente no arquivo de dados.

Essa abordagem introduz um mecanismo eficiente de acesso indireto à memória, permitindo que os recursos sejam localizados dinamicamente em tempo de execução, sem dependência de endereços fixos.

Do ponto de vista de engenharia de software e arquitetura de computadores, essa solução apresenta benefícios significativos:

- **Manutenibilidade:** Permite a reorganização, inserção ou atualização de ativos sem necessidade de modificar ou recompilar o código-fonte principal, desde que a consistência dos índices seja preservada.
- **Eficiência no Gerenciamento de Memória:** Possibilita o carregamento seletivo de recursos, permitindo que apenas os dados necessários sejam mantidos em memória, respeitando as limitações da arquitetura da época.
- **Desacoplamento entre Código e Dados:** Reduz o acoplamento estrutural, facilitando a manutenção e evolução do sistema.

2. Figuras

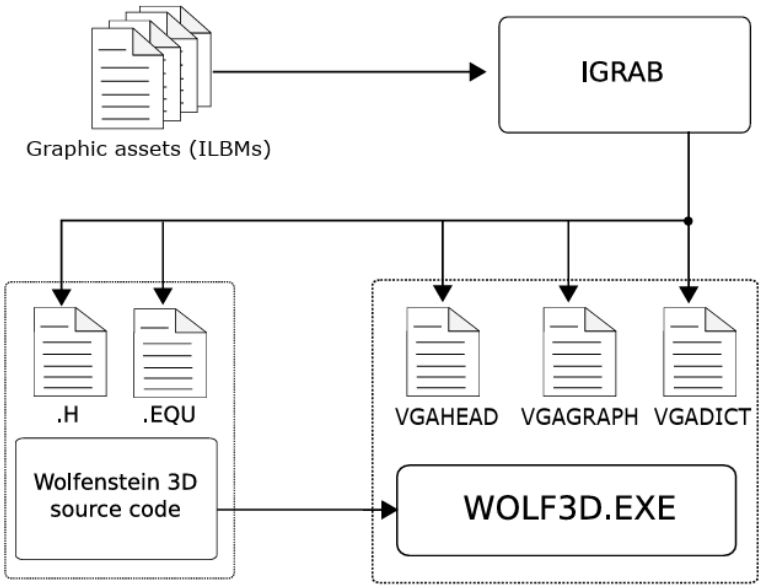

Figura 1. Pipeline de processamento de assets em Wolfenstein 3D, evidenciando o empacotamento de arquivos ILBM pela ferramenta IGRAB e a geração de estruturas de dados utilizadas pelo motor do jogo.

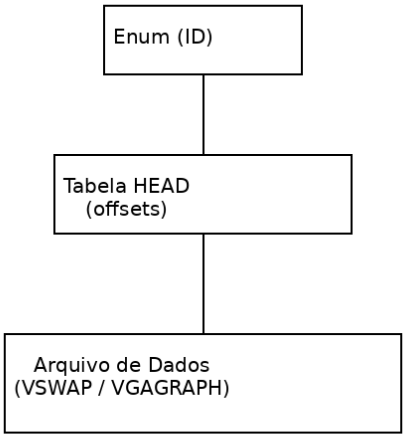

Figura 2. Modelo de acesso indireto a assets em Wolfenstein 3D, no qual identificadores (enums) indexam tabelas de offsets para localizar dados em arquivos binários.

Resumo. *Como ilustrado na Figura 2, o uso de indireção permite desacoplar o código da organização física dos dados, possibilitando maior flexibilidade na gestão dos recursos. Esse mecanismo é fundamental para permitir que os*

assets sejam reorganizados ou atualizados sem a necessidade de modificações no código-fonte, desde que os índices permaneçam consistentes.

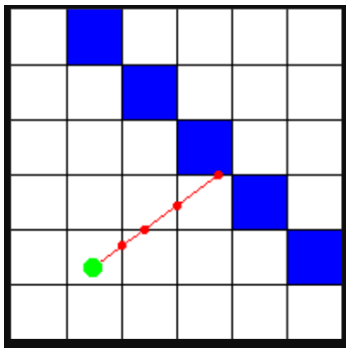

Figura 3. Ilustração do algoritmo de ray casting, no qual raios são projetados em um mapa bidimensional para determinar a distância até obstáculos.

Referências

[1] ABRASH, M. (1997). *Graphics Programming Black Book*. Coriolis Group.

[2] CARMACK, J. (1992). *Wolfenstein 3D Source Code*. id Software.

[3] CARMACK, J. (1993). *DOOM Source Code*. id Software.

[4] FOLEY, J. D. et al. (1996). *Computer Graphics: Principles and Practice*. Addison-Wesley.

[5] SANGLARD, F. (2014). *Game Engine Black Book: Wolfenstein 3D*.