Wolfenstein 3D: Análise Técnica do Motor, Ferramentas e Pipeline Gráfico

Erlon Felipe Castiglioni Tristão Pinheiro
Gabriel Araújo Pieckhardt Romão
Guilherme Gasperazzo Zamprogno
Lucas Pizzol
Pablo Barcellos Soares

Universidade de Vila Velha

Abril de 2026

Resumo

Este trabalho analisa o desenvolvimento técnico de Wolfenstein 3D, destacando o uso da linguagem C (Borland C++ 3.1) e de ferramentas proprietárias. Detalha-se a divisão de tarefas entre a equipe, as táticas de debugging em múltiplos monitores e o processo de criação de assets em Deluxe Paint. O estudo demonstra como a id Software superou as limitações de hardware da época, como a restrição da paleta VGA de 256 cores e a gestão de memória, criando um fluxo de trabalho altamente otimizado que marcou a história da engenharia de jogos.

Sumário

1 Introdução

No início dos anos 1990, o desenvolvimento de jogos lidava com severas restrições de processamento e memória. Nesse cenário, **Wolfenstein 3D** destacou-se ao introduzir técnicas pioneiras de programação e design gráfico. O objetivo deste trabalho é descrever as soluções técnicas arquitetadas pela id Software para viabilizar o projeto. O texto aborda o ambiente de programação e a divisão do código, os métodos de depuração em baixo nível e, por fim, as estratégias visuais adotadas para integrar gráficos complexos dentro das limitações da arquitetura VGA.

2 Programando

O desenvolvimento do jogo foi feito na linguagem C, porém o compilador utilizado foi o Borland C++ 3.1. Esta foi a principal ferramenta de desenvolvimento, sendo uma IDE completa que integrava editor **(BC.EXE)**, compiler **(BCC.EXE)** e linker **(TLINK.EXE)** em uma única aplicação. A interface oferecia múltiplas janelas para programar e syntax highlight, facilitando grande parte do desenvolvimento.

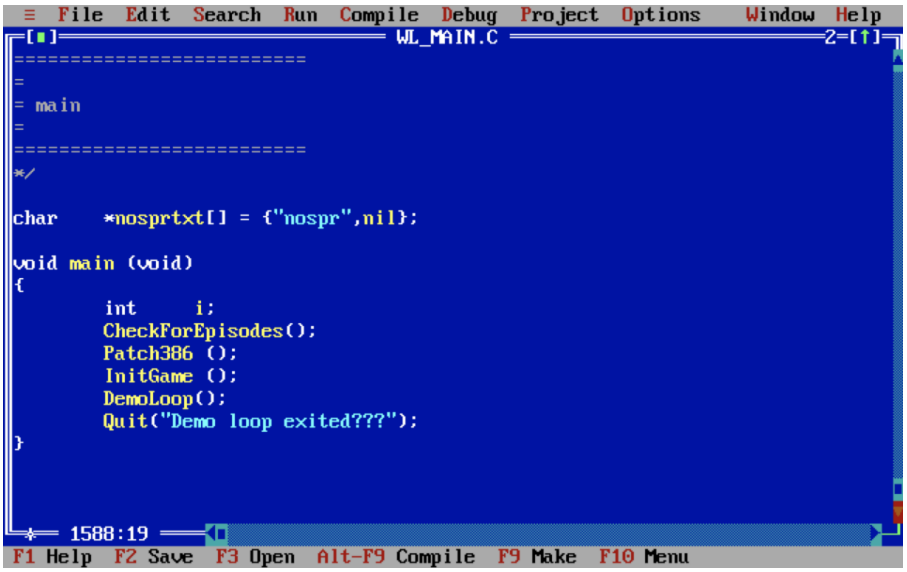

Figura 1: Interface do Borland C++ 3.1

O compilador rodava em VGA Mode 3, que era uma espécie de modo da placa de vídeo semelhante ao terminal, apresentando uma tela sem gráficos e com apenas textos para mostrar as informações.

Borland C++ 3.1

Modo texto VGA: sem gráficos, apenas caracteres, com resolução de 80x25.

2.1 Divisão do trabalho

John Carmack ficou responsável pelo parte do **'runtime code'**, que seria o código principal do jogo, referente a instruções que executam enquanto o programa está rodando.

Runtime Code

Código principal do jogo, referente a instruções que executam enquanto o programa está rodando.

John Romero programou grande parte dos componentes e ferramentas usadas, que foram TED5 Map editor, IGRAB-ED graphic assets packers, MUSE sound packer. Detalhando as informações sobre cada uma:

- **TED5 (Tile Editor 5)**: editor de mapas do jogo, onde se criava os níveis, defin
- **IGRAB**: empacotador de assets gráficos, junta as imagens em arquivos do jogo e gera cabeçalhos '.h' para o código acessar
- **MUSE**: ferramenta de áudio, onde se processava músicas e efeitos e preparava para rodar em placas de som da época.

Jason Blochowiak escreveu código dos sistemas secundários do jogo, onde se gerenciava as informações importantes. Estes foram:

- **Input Manager**: gerenciador de entradas, responsável por ler os comandos do teclado e do mouse e repassar para o jogo
- **Sound Manager**: gerenciador de som, responsável por tocar as músicas e os efeitos sonoros do jogo
- **Page Manager**: gerenciador de paginação, responsável por organizar as informações na memória RAM
- **User Manager**: gerenciador de usuário, responsável por salvar as informações do jogador

2.2 Debugando

Para compensar pelo pequeno monitor CRT, alguns dos desenvolvedores utilizavam de 2 monitores, sendo 1 monitor VGA rodando o jogo normalmente enquanto um segundo monitor MDA (monocromático) rodava o debugger (Turbo Debugger 386 - depurador de baixo nível (nível de hardware/máquina) incluso no pacote profissional do Borland C++). Isto só foi possível porque nos modos 13h e 3h do VGA não era utilizado o mesmo ponteiro inicial na memória RAM, permitindo conectar 2 placas de vídeo num mesmo PC.

Exemplo

- **VGA: 0xA0000**
- **MDA: 0xB8000**

Também utilizaram do modo de resolução 80x50, sacrificando um pouco a legibilidade para expandir o espaço vertical utilizável na tela.

3 Assets Gráficos

3.1 Criação e Ferramentas

Todo o universo visual de Wolfenstein 3D foi concebido pelas mãos de Adrian Carmack, que contou com a colaboração pontual de Kevin Cloud em algumas texturas e no design do livro de dicas do jogo. Em uma época onde ferramentas de digitalização (scanners) não eram utilizadas pela equipe, o processo era puramente artesanal: cada elemento gráfico foi desenhado à mão utilizando apenas um mouse. Para realizar esse trabalho, o artista utilizou o software **Deluxe Paint**, da Electronic Arts, e os arquivos eram salvos no formato proprietário **ILBM** (InterLeaved BitMap) antes de serem processados para o motor do jogo.

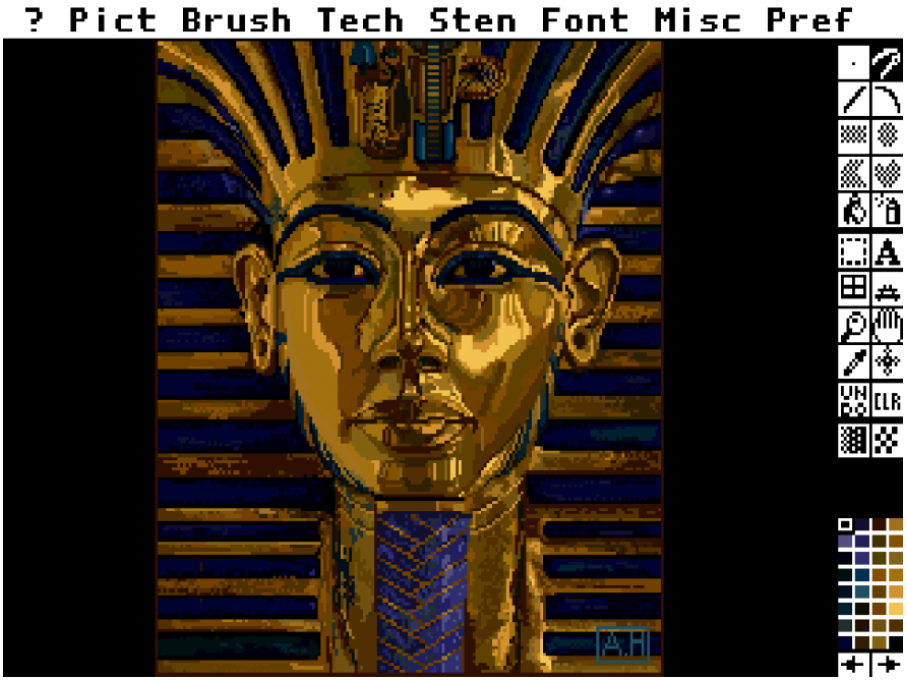

Figura 2: Interface do Deluxe Paint

3.2 A Restrição da Paleta VGA

A criação visual era limitada pelas restrições do hardware VGA da época, que não trabalhava com cores RGB de 24 bits diretamente, mas sim com uma paleta indexada de 256 cores. Isso exigia que Adrian Carmack tomasse a decisão estratégica de quais cores fariam parte dessa tabela antes mesmo de começar a desenhar.

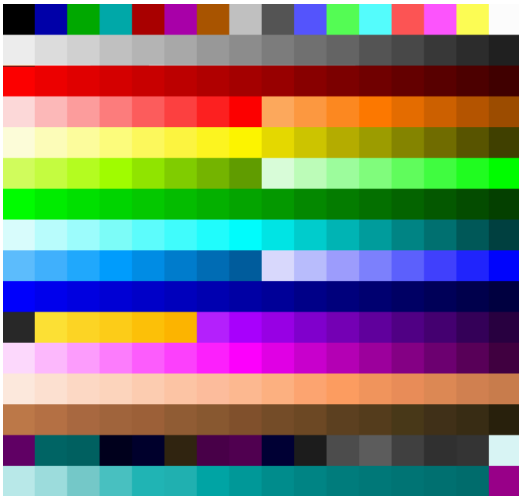

Figura 3: Paleta de Cores do Wolfenstein 3D

Enquanto outros jogos da época trocavam de paleta dependendo da fase, a id Software optou por utilizar uma paleta única e fixa para o jogo inteiro, simplificando a renderização. Dentro dessa paleta, o índice 0xFF (que visualmente parece um rosa choque) era reservado como uma cor especial, interpretada pelo motor como transparente e ignorada durante a renderização para permitir o recorte de objetos e personagens.

3.3 Resolução e Proporção

Os gráficos foram criados na resolução padrão de 320x200 pixels, mas o artista precisava lidar com uma particularidade dos monitores CRT: o hardware VGA ”esticava” o framebuffer verticalmente ao exibi-lo na tela. Assim, Adrian desenhava os objetos já prevendo essa distorção para que eles não parecessem deformados para o jogador. Para o motor gerenciar tudo isso, os arquivos foram divididos em duas grandes categorias: os itens de menu 2D, armazenados nos arquivos VGAGRAPH, VGAHEAD e VGADICT, e os itens da fase 3D, que compreendem as paredes e os sprites (objetos e inimigos), centralizados no arquivo de arquivo VSWAP. Essa estrutura permitia que o jogo buscasse rapidamente o que precisava dependendo da fase de processamento, fosse ela a navegação por menus ou a ação em tempo real.

4 Fluxo de Trabalho de Assets

4.1 Pipeline de Criação

O sistema de assets de Wolfenstein 3D foi projetado para otimizar o uso de memória e facilitar o acesso aos gráficos dentro do motor do jogo. Após a criação das imagens no formato ILBM (InterLeaved BitMap), uma ferramenta interna chamada **IGRAB** era responsável por agrupar todos os arquivos gráficos em um único pacote. Durante esse processo, o IGRAB também gerava automaticamente um arquivo de cabeçalho em C **(.h)**, contendo uma lista de identificadores (IDs) para cada asset. Esses IDs eram utilizados diretamente pelo código do jogo para referenciar imagens, eliminando a necessidade de acessar arquivos individuais. Os dados eram organizados em três arquivos principais:

- **VGAHEAD**: continha os índices dos assets, mapeando cada ID para uma posição específica dentro do arquivo de dados.
- **VGAGRAPH**: armazenava os dados gráficos compactados.
- **VGADICT**: continha a estrutura de compressão baseada em árvore.

Esse sistema criava uma camada de indireção eficiente, permitindo que os assets fossem reorganizados sem necessidade de alterar o código-fonte.

4.2 Uso no Motor do jogo e Funcionamento

Dentro do motor do jogo, os assets eram acessados por meio de um enum, que funcionava como um índice para localizar cada gráfico dentro do pacote de dados.Nesse caso, o jogo não acessa diretamente um arquivo de imagem, mas sim o identificador PAUSEDPIC, que aponta para uma posição na tabela de assets (HEAD), que por sua vez indica onde o dado está armazenado no arquivo (DATA). Antes de serem exibidos, os gráficos eram descompactados utilizando um algoritmo baseado em árvore de Huffman, armazenado no arquivo VGADICT. Esse sistema trazia grande flexibilidade, mas também causou problemas históricos: quando o código-fonte foi liberado, os arquivos .h não correspondiam corretamente aos assets das versões originais do jogo, resultando em erros visuais e gráficos incorretos.

5 Conclusão

O sucesso de Wolfenstein 3D derivou da criação de um ecossistema de ferramentas próprias que contornaram as barreiras físicas dos computadores da época. Soluções como o uso de uma paleta de cores fixa, debugging simultâneo em monitores distintos e um compilador interno de assets (IGRAB) garantiram a fluidez do jogo em máquinas limitadas. A criatividade da equipe na gestão de memória e renderização não apenas viabilizou o projeto, mas também estabeleceu os pilares técnicos para a indústria de jogos modernos.

6 Referências Bibliográficas

SANGLARD, Fabien. **Game Engine Black Book Wolfenstein 3D**: v2.2. [S. l.: s. n.], 2022.