Resumo de Estudo

Como Wolfenstein 3D renderizava seus gráficos

Arthur Schwambach, Chrstiano Rangel, Eduardo Rodrigues,
Elissa Bruneli e Rhayssa dos Santos

6 de abril de 2026

Resumo

Este material apresenta um resumo estruturado sobre o funcionamento gráfico de *Wolfenstein 3D*, com foco na forma como o jogo organizava menus em 2D, entrava na fase de ação e construía a ilusão de um ambiente tridimensional por meio de *raycasting*. O objetivo é explicar, de maneira conceitual, como o motor gráfico aproveitava recursos específicos da placa VGA, como bancos de memória, máscaras, múltiplas páginas de vídeo e *latches*. Também são abordados o ciclo de frames, o problema do tempo variável, as cinco etapas principais de renderização 3D, o HUD e as otimizações usadas para alcançar bom desempenho em computadores limitados do início dos anos 1990.

Sumário

1 Visão geral da matéria **3**

2 Renderizador 2D dos menus **3**
2.1 Introdução ao menu . . . 3
2.2 Uso dos bancos de memória da VGA . . . 3
2.3 Imagens, textos e gerenciamento interno . . . 3

3 Entrada na fase de ação e raycasting **4**
3.1 Início do renderizador 3D . . . 4
3.2 Funcionamento do raycasting . . . 4
3.3 Eficiência da técnica . . . 4
3.4 Tratamento das portas . . . 4

4 Ciclo de frames e tempo variável **5**
4.1 Loop principal do jogo . . . 5
4.2 Problema do tempo variável . . . 5
4.3 Demos e comparação com Doom . . . 5

5 As cinco etapas de renderização 3D **6**
5.1 Limpeza do framebuffer 6
5.2 Desenho das paredes 6
5.3 Desenho dos sprites 6
5.4 Desenho da arma do jogador 6
5.5 Troca dos buffers 6

6 HUD, otimizações da VGA e limpeza da tela **7**
6.1 Preparação do HUD 7
6.2 Elementos dinâmicos do HUD 7
6.3 Imagens entrelaçadas e ganho de velocidade 7
6.4 Limpeza da área 3D 7

7 Síntese final da matéria **7**

1 Visão geral da matéria

Este conteúdo mostra como *Wolfenstein 3D* organizava sua parte gráfica em um contexto de hardware limitado. O jogo precisava desenhar menus, cenários, inimigos, objetos, arma do jogador e elementos de interface sem depender de recursos modernos de aceleração gráfica.

A matéria pode ser dividida em dois grandes blocos. O primeiro trata do renderizador 2D, usado principalmente nos menus e telas de configuração. O segundo trata do renderizador 3D, responsável pela fase de ação, onde o jogador se movimenta por corredores e salas com aparência tridimensional.

A grande ideia do estudo é que o jogo não era impressionante apenas pelo resultado visual. Ele também era um exemplo de engenharia extremamente otimizada, pois explorava detalhes da VGA e da memória de vídeo para reduzir o número de operações necessárias em cada quadro.

2 Renderizador 2D dos menus

2.1 Introdução ao menu

Antes da ação começar, o jogador passava por uma tela de menu. Essa parte pode ser entendida como a fase de menu, ou renderizador 2D. Nela, o jogador podia configurar o jogo e escolher opções como *New Game*, *Sound*, *Control*, *Load Game* e outras funções.

Apesar de parecer uma parte simples, o menu também exigia cuidado técnico. A tela usada tinha resolução de 320 por 200 pixels, totalizando 64 mil pixels. Preencher toda essa área com uma cor, como o fundo vermelho do menu, poderia exigir muitas escritas na memória se o processo fosse feito pixel por pixel.

2.2 Uso dos bancos de memória da VGA

Para acelerar o preenchimento da tela, o jogo aproveitava os quatro bancos de memória da VGA. Em vez de escrever em cada pixel individualmente, o motor configurava uma máscara que permitia escrever em todos os bancos ao mesmo tempo.

Com esse recurso, uma única operação podia alterar até quatro pixels de uma vez. Usando registradores de 16 bits, o processo ficava ainda mais rápido, permitindo escrever oito pixels em uma única operação.

Na prática, uma tela que teoricamente exigiria 64 mil escritas podia ser preenchida com aproximadamente 8 mil operações. Esse detalhe mostra como os programadores precisavam conhecer profundamente o hardware para conseguir desempenho aceitável.

2.3 Imagens, textos e gerenciamento interno

Além do preenchimento de fundo, o menu também usava imagens e textos carregados do disco para a memória. Para isso, o jogo contava com sistemas internos como o *User Manager* e o *Cache Manager*.

As imagens do menu eram chamadas de *pic*. Já durante a fase 3D, os elementos móveis, como inimigos e objetos, eram chamados de *sprites*. Essa distinção ajudava a organizar os diferentes tipos de conteúdo gráfico usados pelo jogo.

3 Entrada na fase de ação e raycasting

3.1 Início do renderizador 3D

Depois que o jogador terminava de configurar o jogo no menu, começava a fase de ação. É nessa etapa que entrava o renderizador 3D, responsável por criar a aparência de corredores, paredes, portas e objetos no cenário.

O motor gráfico de *Wolfenstein 3D* usava uma técnica chamada *raycasting*. Essa técnica criava a ilusão de um ambiente tridimensional a partir de cálculos relativamente simples feitos sobre um mapa 2D.

3.2 Funcionamento do raycasting

A ideia principal do *raycasting* era lançar um raio para cada coluna vertical da tela. Como a tela possuía 320 colunas de pixels, o jogo lançava cerca de 320 raios, um para cada coluna.

Cada raio saía do ponto de vista do jogador e seguia pelo mapa até encontrar uma parede. Quando isso acontecia, o jogo calculava a distância entre o jogador e a parede atingida.

A altura da coluna desenhada na tela dependia dessa distância. Quanto mais perto a parede estivesse, maior ela aparecia. Quanto mais distante, menor ela era desenhada. Assim, a profundidade era produzida por uma relação simples entre distância e altura projetada.

3.3 Eficiência da técnica

Esse método era eficiente para a época porque não exigia a renderização de polígonos 3D complexos. O jogo calculava interseções entre raios e paredes em um mapa bidimensional e transformava esses resultados em uma imagem com aparência tridimensional.

Dessa forma, *Wolfenstein 3D* conseguia apresentar uma sensação de profundidade usando uma solução muito mais leve do que um motor 3D completo baseado em polígonos.

3.4 Tratamento das portas

Um ponto importante é que as portas não eram tratadas como *sprites*. Elas faziam parte do mundo sólido do jogo. O *raycaster* conseguia identificar quando um raio atingia uma porta, verificar se ela estava aberta ou fechada e decidir se o raio deveria parar ali ou continuar atravessando.

Isso permitia que as portas se integrassem ao funcionamento do mapa e da renderização das paredes, em vez de serem apenas objetos desenhados por cima da cena.

4 Ciclo de frames e tempo variável

4.1 Loop principal do jogo

O funcionamento básico de um jogo pode ser entendido como um ciclo contínuo. O motor lê o tempo atual, calcula quanto tempo passou desde o último quadro, atualiza o mundo e depois renderiza a cena.

Em termos conceituais, o processo segue uma sequência como esta:

- calcular o tempo atual;
- atualizar o estado do mundo;
- desenhar a tela;
- repetir o processo no próximo frame.

Esse modelo era comum nos jogos do início dos anos 1990, mas trazia um problema importante: o tempo de cada frame podia variar.

4.2 Problema do tempo variável

Um computador mais rápido renderizava os quadros em menos tempo. Um computador mais lento demorava mais para concluir o mesmo processo. Isso fazia com que a simulação do jogo não fosse totalmente determinística.

Em outras palavras, a mesma sequência de ações poderia gerar resultados diferentes dependendo da máquina. Esse problema também afetava a reprodução de demos gravadas, pois uma pequena diferença no tempo de execução podia deixar os comandos do jogador fora de sincronia.

4.3 Demos e comparação com Doom

Para resolver parcialmente esse problema nas demos, o motor de *Wolfenstein 3D* simulava os eventos em intervalos fixos. Porém, isso criava diferenças entre computadores: a reprodução podia ficar mais lenta em máquinas 286 e mais rápida em máquinas 486, já que a demo havia sido gravada em uma máquina 386DX.

O motor de *Doom*, lançado depois, lidou melhor com esse problema ao separar a atualização do mundo da renderização. A simulação passou a acontecer em intervalos fixos, enquanto o renderizador podia funcionar de maneira separada. Isso facilitava gravar e reproduzir partidas sem perder a sincronia.

5 As cinco etapas de renderização 3D

5.1 Limpeza do framebuffer

Uma cena 3D em *Wolfenstein 3D* era desenhada em cinco etapas principais. A primeira etapa era limpar o *framebuffer*, desenhando o teto e o chão com cores sólidas.

O teto ocupava a parte superior da área 3D, enquanto o chão ocupava a parte inferior. Essa limpeza preparava a tela para receber as paredes, objetos e demais elementos da cena.

5.2 Desenho das paredes

A segunda etapa era desenhar as paredes. Para isso, o jogo lançava raios a partir da posição do jogador. Cada raio encontrava a parede mais próxima e produzia uma coluna texturizada na tela.

A altura dessa coluna dependia da distância calculada pelo *raycasting*. Paredes próximas eram desenhadas mais altas, enquanto paredes distantes eram desenhadas menores.

5.3 Desenho dos sprites

A terceira etapa era desenhar os *sprites*, chamados no texto também de *things*. Esses elementos incluíam inimigos, barris, lâmpadas, corpos e outros objetos presentes no cenário.

Depois que as paredes eram desenhadas, os *sprites* eram posicionados na cena. Quando ficavam atrás de paredes ou portas, precisavam ser recortados para não aparecerem de forma incorreta.

5.4 Desenho da arma do jogador

A quarta etapa era desenhar a arma do jogador. Ela aparecia por cima da cena, geralmente no centro inferior da tela.

Em um motor moderno, a arma poderia ser desenhada com técnicas diferentes, como uso de *depth buffer*. Porém, naquela época, o acesso à memória era lento. Por isso, era mais vantajoso aceitar uma sobreposição controlada do que gastar mais processamento com uma solução mais complexa.

5.5 Troca dos buffers

A quinta e última etapa era trocar os buffers. O jogo usava um sistema de múltiplas páginas de vídeo. Depois de montar a imagem em um *framebuffer*, o controlador de vídeo era instruído a mostrar a nova página na próxima sincronização vertical.

Essas cinco etapas formavam o processo completo de criação de cada imagem vista pelo jogador durante a partida.

6 HUD, otimizações da VGA e limpeza da tela

6.1 Preparação do HUD

Antes de começar a renderizar os frames 3D, o motor preparava elementos fixos da interface. Entre eles estavam o fundo verde, a barra azul de status e os textos *LEVEL*, *SCORE*, *LIVES*, *HEALTH* e *AMMO*.

Esse HUD era desenhado apenas uma vez no começo da fase 3D, mas precisava existir nas três páginas de vídeo usadas pelo jogo. Assim, independentemente da página exibida, a interface continuava disponível.

6.2 Elementos dinâmicos do HUD

Algumas partes do HUD mudavam durante a partida, como vida, munição, pontuação, arma atual e o rosto do personagem. Para atualizar essas partes rapidamente, o jogo usava outro recurso da VGA: os *latches*.

Os *latches* eram circuitos que guardavam valores lidos da memória de vídeo. Embora fossem pensados para outro modo gráfico, os programadores descobriram que podiam reaproveitá-los no *Mode-Y*. Com isso, era possível copiar dados de uma região da VRAM para outra de forma mais rápida, transferindo quatro bytes de uma vez.

6.3 Imagens entrelaçadas e ganho de velocidade

O jogo também armazenava imagens de forma entrelaçada, separando os dados por banco de memória. Essa organização permitia carregar cada banco rapidamente por meio de operações de cópia.

Essa otimização ajudava principalmente na atualização dos elementos do HUD e gerava cerca de 30% de aumento geral de velocidade nessa parte.

6.4 Limpeza da área 3D

No início de cada frame, o motor também limpava a área 3D da tela com as cores do teto e do chão. Mais uma vez, ele usava a máscara dos bancos VGA para escrever em todos os bancos simultaneamente.

A área 3D tinha 304 por 152 pixels, mas, graças a instruções otimizadas como `REP STOSW`, o processo exigia muito menos instruções do que uma escrita pixel por pixel.

7 Síntese final da matéria

O estudo mostra que *Wolfenstein 3D* foi construído com forte dependência do conhecimento de hardware. Para alcançar bom desempenho, o motor explorava recursos específicos da VGA e organizava cuidadosamente cada etapa da renderização.

Na parte de menus e renderização 2D, os pontos mais importantes são:

- uso da resolução 320 por 200;

- preenchimento rápido da tela;
- aproveitamento dos quatro bancos de memória da VGA;
- uso de máscaras para escrever em vários bancos ao mesmo tempo;
- carregamento de imagens e textos com apoio de sistemas internos.

Na parte de renderização 3D, os conceitos principais são:

- uso de *raycasting*;
- lançamento de um raio por coluna da tela;
- cálculo da distância até paredes;
- relação entre distância e altura da coluna desenhada;
- tratamento especial de portas como parte do mundo sólido;
- ciclo de frames com atualização e desenho;
- problema do tempo variável e das demos;
- cinco etapas de renderização da cena;
- uso de múltiplas páginas de vídeo;
- otimizações com *latches*, *Mode-Y* e imagens entrelaçadas.

A principal conclusão é que *Wolfenstein 3D* não dependia apenas de uma boa ideia visual. O jogo combinava *raycasting*, manipulação direta da memória de vídeo e truques específicos da VGA para criar uma experiência 3D rápida em computadores limitados da época.