Universidade Vila Velha

Grupo 4 — Turma CC5Mb

Disciplina: Arquitetura e Organização de Computadores 2

Professor: Abrantes Araujo Silva Filho

Gabriel Henrique Garces de Deus

Teo Giambarra Roseiro

Arthur Prates Pessoti

Arthur Solar de Castro Gomes

Rafael Fassina

Resumo Didático da Seção 4.5

Startup

do *Wolfenstein 3D Game Engine Black Book*

Vila Velha

22 de abril de 2026

Resumo

Este documento apresenta um resumo didático da seção 4.5 do livro *Wolfenstein 3D Game Engine Black Book*, com foco no processo de inicialização do motor. O objetivo é explicar, de forma clara e direta, como o *Wolfenstein 3D* começa a rodar em um PC limitado da época e por que essa etapa era muito mais complexa do que simplesmente mostrar uma tela inicial. A seção mostra que o *startup* do jogo já precisava resolver problemas reais de compatibilidade, memória e vídeo antes mesmo da fase principal começar. Por isso, o capítulo ajuda a entender que a robustez do jogo dependia não apenas do renderizador 3D, mas também de uma sequência inicial muito bem planejada para diagnosticar o sistema, organizar a memória e contornar limitações da VGA.

Sumário

1 Introdução

A seção 4.5 trata da fase de inicialização do *Wolfenstein 3D*. Em um computador atual, abrir um jogo costuma parecer algo simples: o programa carrega arquivos, monta a interface e depois começa. No contexto do PC de 1991, porém, isso estava longe de ser trivial. Antes de entrar no menu ou mostrar qualquer ação em 3D, o motor precisava descobrir se a máquina era capaz de rodar o jogo, quanto de memória estava realmente disponível e de que forma poderia usar a placa de vídeo sem produzir artefatos visuais.

Essa parte do livro é importante porque mostra que o *startup* não era uma etapa secundária. Ele já concentrava várias decisões de engenharia. O jogo precisava exibir um diagnóstico útil para o usuário, reservar e liberar memória com cuidado e, logo em seguida, encontrar uma forma prática de contornar a falta de *double buffering* na VGA. Em outras palavras, o início do jogo já era um teste real da arquitetura apresentada na seção anterior.

Além disso, a seção deixa claro que muitos comportamentos que o jogador enxerga como "tela de abertura" ou "tela de aviso" existem por razões técnicas. A tela de *signon* não serve apenas para decorar a inicialização; ela também funciona como uma forma de autoavaliação do sistema. Da mesma maneira, a adaptação da VGA não é um detalhe isolado de hardware, mas uma escolha que influencia a forma como todo o resto do motor desenha na tela.

2 A inicialização como etapa crítica do motor

Logo no começo da seção, o livro reforça uma ideia central: quando o motor inicia, ele ainda está enfrentando de frente os problemas descritos no capítulo de hardware. Isso significa que o jogo não começa em um ambiente "limpo" ou previsível. Ele entra em execução dentro de um ecossistema heterogêneo, com diferentes drivers carregados, diferentes placas de som, diferentes quantidades de memória convencional e diferentes condições de vídeo.

Na prática, isso obriga o motor a ser defensivo. Em vez de assumir que tudo vai funcionar, ele precisa primeiro medir a realidade da máquina. Essa postura é muito importante do ponto de vista de engenharia, porque mostra uma preocupação com a execução no mundo real. O programa não podia depender apenas de um ambiente ideal. Ele precisava detectar limitações cedo, comunicar problemas ao usuário e só então seguir adiante.

O valor dessa etapa inicial está justamente nisso: ela transforma a abertura do jogo em uma combinação de diagnóstico, preparação e adaptação. O *startup* organiza o terreno para que as partes seguintes do motor possam funcionar de forma estável.

3 Signon

3.1 Função da tela inicial

A primeira grande parte da seção é a *signon screen*. O livro explica que essa tela existe principalmente para autodiagnóstico. Como havia muitos tipos de PCs no mercado, com diferentes drivers e diferentes placas de som, o jogo precisava verificar o que estava presente na máquina e se havia memória suficiente para continuar.

Por isso, a tela de *signon* não era só um enfeite visual. Ela mostrava dispositivos reconhecidos, como mouse, joystick e hardware de som, e ainda exibia um indicador importante de memória chamado *MAIN*. Esse ponto é central porque, em DOS, o programa só tinha acesso à memória convencional, dentro do limite clássico de 640 KiB. Tudo o que estivesse carregado no sistema consumia parte desse espaço, então o jogo precisava saber se sobrava RAM bastante para rodar.

3.2 A importância da memória convencional

O livro destaca que o *Wolf3D* precisava de pelo menos 320 KiB de memória convencional. Isso ajuda a entender o papel prático da tela inicial. Se o sistema não conseguisse carregar o executável em RAM, o usuário receberia a mensagem de falta de memória. A tela de *signon*, portanto, também servia como uma forma de reduzir dúvidas e facilitar o suporte, já que a equipe da id Software não tinha estrutura para resolver individualmente problemas de cada cliente.

Esse detalhe mostra algo importante sobre desenvolvimento na época: não bastava fazer o jogo funcionar na máquina do desenvolvedor. Era preciso antecipar erros comuns de instalação e de configuração do sistema operacional. O *startup* funcionava, assim, como uma barreira de segurança para impedir que o jogador seguisse adiante sem os recursos mínimos.

3.3 Por que a paleta e a imagem estavam dentro do executável

Outro ponto interessante é que, quando a tela de *signon* aparece, o único manager efetivamente carregado é o *Memory Manager*. Ainda não existe um sistema de arquivos pronto para o motor buscar imagens ou outros dados. Por causa disso, tanto a paleta quanto o bitmap dessa tela ficam compilados diretamente dentro do executável.

Na prática, isso significa que esses dados já entram na RAM quando o DOS carrega o programa. O motor só precisa configurar o modo de vídeo, enviar a paleta para a VGA, copiar a imagem da memória para a VRAM e desenhar os indicadores coloridos conforme os dispositivos detectados. É uma solução muito pragmática: como ainda não existe uma infraestrutura completa de carregamento, os recursos necessários para a primeira tela já vêm "embutidos" no próprio binário.

3.4 Reaproveitamento de memória

Depois que a tela é usada, o bitmap de *signon* deixa de ser necessário. O livro mostra que o motor então recupera essa memória para abrir espaço para a execução do jogo. Isso é importante porque a imagem ocupa 320×200 bytes, isto é, 64.000 bytes, uma quantia grande demais para ser ignorada em um ambiente tão limitado.

Esse comportamento resume bem a mentalidade do motor. Nada fica carregado sem necessidade. O *startup* usa a imagem, exibe o diagnóstico e depois devolve imediatamente essa área para o sistema de memória. Em um PC da época, economizar dezenas de kilobytes era uma decisão concreta de desempenho e de viabilidade.

3.5 Lição prática da etapa de signon

A parte do *signon* ensina que uma tela inicial pode cumprir várias funções ao mesmo tempo. Ela informa o usuário, verifica compatibilidade, ajuda no suporte técnico e ainda reaproveita recursos logo em seguida. Em vez de ser apenas uma abertura estética, ela participa diretamente da administração do sistema.

4 Resolvendo o problema da VGA

4.1 O limite do modo 13h

A segunda grande parte da seção trata do problema mais técnico do *startup*: a VGA. O capítulo de hardware já havia mostrado que os modos da VGA não ofereciam *double buffering* de forma direta, o que causava *tearing* e dificultava animações suaves. O modo mais atraente era o 13h, com resolução de 320×200 e 256 cores indexadas, mas ele oferecia apenas um framebuffer aparente.

O livro explica que isso acontecia por causa do mecanismo conhecido como *Chain-4*. Esse circuito distribuía automaticamente os acessos entre os quatro bancos de memória da VRAM. Do ponto de vista do programador, parecia existir um espaço contínuo e simples de acessar. O problema é que essa facilidade desperdiçava grande parte da VRAM disponível. Na prática, 75% da memória de vídeo ficava inutilizável nesse esquema.

4.2 Desligando o Chain-4

A solução adotada foi desabilitar o *Chain-4*. Essa ideia se relaciona ao que ficou conhecido como *Mode-X*, popularizado por Michael Abrash, mas o *Wolfenstein 3D* faz algo próprio: mantém a resolução em 320×200 em vez de subir para 320×240. Essa adaptação ficou conhecida como *Mode-Y*.

A escolha por 320×200 tem duas justificativas principais. A primeira é desempenho. Um framebuffer de 320×240 possui mais pixels, o que elevaria o custo de desenho por quadro justamente em um motor que já sofria para atingir uma taxa de quadros aceitável. A segunda é produção de assets. Como o Deluxe Paint trabalhava no modo 13h, com pixels

não quadrados, manter a resolução do jogo evitava um fluxo desconfortável em que os artistas produziriam imagens em um formato e o motor renderizaria em outro.

4.3 Como a VRAM passa a ser organizada

Ao desabilitar o *Chain-4*, o motor deixa de depender do mapeamento automático da VGA e passa a controlar explicitamente os bancos de memória. O ganho é enorme: os 256 KiB da VRAM ficam totalmente acessíveis. O livro mostra que essa memória então é repartida em quatro partes: três framebuffers de 64.000 bytes e uma área restante para assets gráficos.

Essa reorganização é a base da solução do jogo para a exibição na tela. Em vez de contar com um único framebuffer, o motor passa a ter várias páginas de vídeo à disposição. Isso reduz muito o risco de artefatos, porque o desenho pode acontecer fora da página atualmente visível.

4.4 O novo custo: escolher manualmente o banco

Mas a solução não vem de graça. Ao remover o *Chain-4*, o programador passa a ser responsável por escolher em qual banco da VGA cada escrita será feita. O livro mostra que isso pode ser feito alterando o registrador apropriado, mas cada troca de banco exige operações de I/O lentas. Se o motor desenhar da forma mais intuitiva possível, percorrendo a tela horizontalmente, a quantidade de trocas se torna alta demais e o desempenho despenca.

Esse detalhe é muito importante porque revela um padrão recorrente na engenharia do jogo: resolver um grande problema pode introduzir outros menores. Aqui, ao conquistar acesso total à VRAM, o motor ganha flexibilidade, mas assume também um custo extra de controle manual.

4.5 Por que tudo passa a ser desenhado verticalmente

A resposta para esse novo problema foi mudar a ordem de desenho. Em vez de rasterizar primeiro por linhas horizontais, o motor passa a privilegiar colunas verticais. Assim, ele reduz drasticamente a quantidade de trocas de banco, porque mantém a escrita por mais tempo dentro do mesmo plano da VGA.

O livro mostra que essa mudança praticamente dobra a velocidade em um exemplo simples de limpeza de tela. Mais importante do que o número em si é a consequência estrutural: isso não afeta apenas uma função isolada. A decisão se espalha por todo o motor. Paredes, sprites e até menus passam a seguir essa lógica. Ou seja, uma limitação do hardware força uma reorganização profunda na forma como o jogo desenha e até como os assets são armazenados em memória.

Esse é um dos pontos mais ricos da seção. A VGA não impõe apenas uma restrição local; ela molda o estilo inteiro de renderização do motor. A forma de escrever na VRAM acaba

influenciando a arquitetura dos dados gráficos.

5 Page flipping, triple buffering e correção visual

5.1 A ideia de usar várias páginas

Com a VRAM reorganizada, o motor passa a trabalhar com três páginas principais. O livro descreve que o jogo desenha em uma página, depois em outra, depois em outra, e só então retorna à primeira. Essa estratégia evita que o motor fique preso esperando sincronização vertical o tempo todo e garante que sempre exista uma página válida para ser mostrada enquanto outra está sendo preparada.

Na prática, isso é uma solução elegante para o problema do *tearing*. O jogo deixa de desenhar diretamente sobre a imagem visível naquele instante e passa a alternar o framebuffer mostrado pelo controlador de vídeo.

5.2 O problema de atomicidade

Só que surge um novo obstáculo: mudar a página visível depende de atualizar o endereço de início de leitura do CRT Controller. Esse endereço é um valor de 16 bits, mas o hardware só permite escrever 8 bits por vez. Isso cria um risco sério. Se a troca ocorrer em um momento ruim, o controlador pode enxergar temporariamente uma combinação inválida entre o byte alto e o byte baixo, produzindo uma imagem distorcida, com desalinhamento e mistura de partes de páginas diferentes.

Esse é um problema clássico de atomicidade. O valor lógico é único, mas a atualização física acontece em duas etapas separadas. Em um sistema sensível ao tempo como esse, essa pequena janela intermediária já basta para gerar erro visual.

5.3 A solução do padding

A forma como o *Wolfenstein 3D* resolve isso é uma das partes mais elegantes da seção. Em vez de posicionar os framebuffers um imediatamente após o outro, o motor adiciona um pequeno espaço de preenchimento entre eles. Por isso, cada página usa uma altura lógica de 208 linhas, e não exatamente 200.

À primeira vista, isso parece estranho, mas o objetivo é muito preciso: fazer com que o endereço inicial de cada página difira apenas no byte alto. Assim, ao alternar entre páginas, basta modificar esse byte mais significativo. Como o byte baixo permanece igual, a troca se torna efetivamente atômica do ponto de vista prático.

O resultado é que as páginas ficam alinhadas em endereços como 0x0000, 0x4100 e 0x8200. Com isso, o motor consegue alternar o framebuffer exibido sem correr o risco de o CRT Controller capturar um endereço “meio atualizado”.

5.4 Valor da solução

Essa parte da seção ensina muito sobre programação próxima do hardware. O problema não foi resolvido com uma abstração sofisticada nem com mais poder computacional. Ele foi resolvido com organização inteligente do espaço de memória. Em vez de tentar forçar uma escrita atômica impossível, o motor reorganiza os dados para que uma escrita parcial já seja suficiente.

É uma solução extremamente prática e criativa. Mostra também como, em sistemas limitados, layout de memória não é apenas um detalhe de implementação, mas parte essencial do comportamento correto do programa.

6 Profound Carnage

6.1 A tela de aviso

Depois da *signon screen*, o jogo mostra a famosa tela de classificação "PC Profound Carnage-13". O livro explica que, em 1991, ainda não existia uma entidade oficial como a ESRB para classificar jogos eletrônicos. Mesmo assim, o *Wolfenstein 3D* exibe um aviso, claramente inspirado no selo cinematográfico PG-13.

Essa parte é curta em termos técnicos, mas tem valor contextual. Ela mostra o tom irreverente da id Software e ajuda a marcar a identidade do jogo logo no início. A abertura não serve apenas para preparar hardware e memória; ela também estabelece personalidade.

6.2 Sentido dentro da sequência de abertura

Dentro da lógica do *startup*, essa tela funciona como uma transição entre o diagnóstico técnico e a apresentação do jogo ao jogador. Depois de verificar se a máquina consegue rodar o programa e de resolver os primeiros problemas de vídeo, o motor pode então seguir para uma abertura mais reconhecível do ponto de vista da experiência do usuário.

Mesmo não sendo a parte mais complexa da seção, o *Profound Carnage* ajuda a fechar a sequência de inicialização mostrando que o *startup* também possui um papel de comunicação e de identidade visual.

7 O que a seção 4.5 ensina na prática

A principal lição da seção 4.5 é que inicializar um jogo em 1991 significava resolver problemas reais antes mesmo da primeira fase ou do primeiro menu. O *startup* do *Wolfenstein 3D* já precisava diagnosticar o sistema, administrar memória com cuidado e reconfigurar o vídeo em um nível bastante próximo do hardware.

A tela de *signon* mostra a importância de pensar na execução concreta do programa, comunicando ao usuário o que foi detectado e evitando continuar quando faltavam recursos

mínimos. A adaptação da VGA mostra como uma limitação séria pode ser vencida com manipulação direta de registradores e reorganização da VRAM. Já o truque do *padding* entre páginas mostra uma solução de engenharia especialmente elegante, porque resolve um problema de atomicidade apenas mudando o layout da memória.

Em conjunto, esses elementos deixam claro que o *startup* não era um bloco secundário do motor. Ele já concentrava diagnósticos, otimizações e decisões estruturais que influenciavam o restante do jogo.

8 Conclusão

A seção 4.5 mostra que o início do *Wolfenstein 3D* era muito mais do que uma sequência de telas introdutórias. O processo de *startup* era uma etapa técnica decisiva, em que o motor verificava a máquina, preparava os recursos básicos e enfrentava diretamente as limitações da VGA.

A tela de *signon* exemplifica bem essa lógica, porque une autodiagnóstico, comunicação com o usuário e reaproveitamento imediato de memória. Já a transformação do modo 13h em *Mode-Y* mostra como o motor foi capaz de contornar uma deficiência importante do hardware e, ao mesmo tempo, estabelecer o padrão de desenho vertical que marcaria o restante da renderização. Por fim, a solução da troca atômica entre páginas mostra o nível de cuidado com que a equipe tratava até detalhes aparentemente pequenos da exibição na tela.

No fim, essa parte do livro reforça uma ideia importante: em sistemas limitados, a qualidade de um jogo não depende só da parte mais visível, como o gráfico 3D ou a ação. Ela depende também de como o software começa, se organiza e prepara o terreno para tudo o que vem depois.