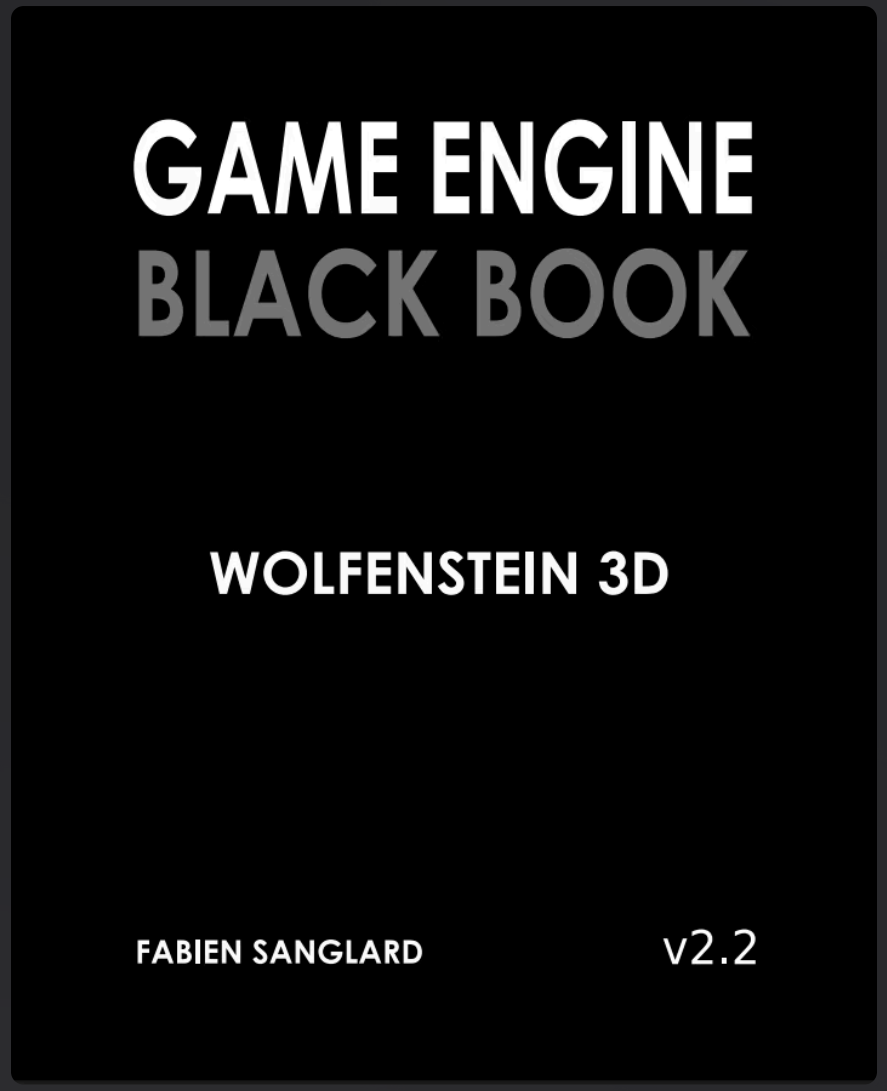
Game Engine Black Book: Wolfenstein 3D

Capítulos 2.2 e 2.3 — RAM e Vídeo

feito por Felipe Stein, Kaiky Viglioni, Marco Antonio, Nícolas Medeiros e Sérgio Alexandre



RAM (Random Access Memory)

Os processadores **Intel x86** surgiram em 1976 com os modelos **8080 e 8086**, que utilizavam registradores de 16 bits e barramento de 20 bits. Essa configuração permitia endereçar **1 MiB de RAM**, uma quantidade massiva para a época (anos 70), quando computadores populares como o Apple II operavam com apenas 64 KiB.

Em 1986, a evolução para os modelos **286 e 386** introduziu o **modo protegido**:

- ## Capacidade

  Barramento de 24 bits (16 MiB de RAM) e registradores de 32 bits (no 386).

- ## Recursos

  Uso de **MMU (***Memory Management Unit***)** para proteção de memória.

- ## Retrocompatibilidade

  Criação do **modo real**, que simulava o comportamento dos chips antigos (16 bits / 1 MiB) para rodar softwares legados.

Apesar do potencial do modo protegido, a transição foi freada pelo **MS-DOS**. Como todos os PCs precisam inicializar em modo real por compatibilidade, o sistema da Microsoft acabou mantendo os programadores presos às limitações técnicas da década anterior durante boa parte dos anos 90.

Limitações do DOS

A Microsoft priorizava a compatibilidade total com softwares antigos, o que fez o **MS-DOS 5.0** continuar operando exclusivamente em **modo real**. Isso gerava um gargalo tecnológico: mesmo em computadores modernos da década de 90, os programadores eram forçados a ignorar os avanços do hardware e programar como se estivessem em 1976.

As restrições impostas por essa escolha incluíam:

Registradores

Uso limitado a 16 bits (como AX, BX, IP, CS, DS).

Memória

Teto de 1 MiB de RAM endereçável.

Hardware subutilizado

CPUs potentes rodando apenas como versões aceleradas do antigo Intel 8086.

Curiosidade

Em 1991, **Linus Torvalds** iniciou um projeto pessoal na Universidade de Helsinki. Diferente do DOS, seu sistema utilizava o **modo protegido**, aproveitando os registradores de 32 bits e a MMU das CPUs modernas. Esse projeto deu origem ao **Linux**, tornando-se o principal concorrente técnico da Microsoft.

O Infame Modo Real: Limite de 1 MiB de RAM

Mesmo com CPUs de 32 bits, a programação era **segmentada em 16 bits.** Para acessar a memória, combinavam-se dois registradores de 16 bits (**Segmento e Offset**) para formar um endereço de 20 bits. Isso limitava o alcance a apenas **1 MiB**, ignorando qualquer memória RAM adicional instalada no hardware.

Layout da Memória (O Mapa de 1 MiB):

00000h a 003FFh

Tabela de Vetores de Interrupção

00400h a 004FFh

Dados do BIOS

00500h a 005FFh

command.com + io.sys

00600h a 9FFFFh

Disponível para programas (cerca de 620 KiB no melhor caso)

A0000h a FFFFFh

UMA (Upper Memory Area) — reservada para ROM do BIOS, placa de vídeo e E/S da placa de som

Dos 1024 KiB originais, apenas 640 KiB (denominados Conventional Memory) estavam acessíveis a um programa. Os 384 KiB restantes eram reservados para a UMA, e cada driver instalado (.SYS e .COM) reduzia ainda mais os 640 KiB disponíveis

O Infame Modo Real: Endereçamento Segmentado de 16 Bits

Para contornar o limite de 16 bits dos registradores em um barramento de 20 bits, a Intel adotou o **endereçamento segmentado**, que gerou complexidades tanto na performance quanto na lógica de programação.

Ponteiros Near vs. Far

1 Near (16 bits):

Rápido e direto, mas limitado a apenas **64 KiB** (dentro do segmento atual).

2 Far (32 bits)

Lento, pois exige o cálculo do endereço real. O processador desloca o registrador de segmento 4 bits à esquerda e soma o deslocamento (**offset**) para gerar o endereço final de 20 bits.

Impacto no Desenvolvimento (C/C++)

1 Novas Palavras-chave

Surgiram `near` e `far` para gerenciar ponteiros, além de macros como `MK_FP` (para criar ponteiros far).

2 Limitação da Libc

A função padrão `malloc` só alocava até 64 KiB. Para blocos maiores, era obrigatório usar `farmalloc`.

O Infame Modo Real: Endereçamento Segmentado de 16 Bits

A maior falha lógica desse sistema é que **o mesmo endereço físico pode ter várias representações lógicas**.

- A memória é dividida em "parágrafos" de 16 bytes pelo segmento.
- Como o deslocamento pode chegar a 64 KiB, diferentes combinações de Segmento:Offset apontam para o mesmo local na RAM.

```
# include < stdio .h >
# include < dos .h >

int main ( int argc , char ** argv ) {
void far * a = MK_FP (0 x0000 , 0 x0120 ) ;
void far * b = MK_FP (0 x0010 , 0 x0020 ) ;
void far * c = MK_FP (0 x0012 , 0 x0000 ) ;

printf ("%d\n", a == b );
printf ("%d\n", a == c );
printf ("%d\n", b == c );

}
```

Memória Estendida

Mesmo com a evolução do hardware, o **modo real** bloqueava o acesso direto a qualquer memória acima de **1 MiB**. A memória adicional instalada nas máquinas (comum ser 2 MiB ou 4 MiB em 1992) era chamada de **Extended Memory**.

A Batalha dos Padrões (Drivers)

O acesso à memória estendida não era padronizado. Os usuários podiam carregar um dos drivers fornecidos com o DOS:

- **EMS (Expanded Memory)**
  Utilizava o driver `EMM386.EXE`
- **XMS (eXtended Memory)**
  Utilizava o driver `HIMEM.SYS`

O Conflito com o Usuário

O maior problema era a falta de automação. Muitos usuários compravam RAM cara, mas não instalavam os drivers. O resultado era o erro **"Not enough memory"**, mesmo em máquinas potentes, pois o sistema permanecia limitado aos 640 KiB convencionais.

A situação era tão comum que empresas como a **id Software** incluíam notas explicativas em seus manuais para ensinar os jogadores a configurar a memória, garantindo que jogos como *Wolfenstein 3D* ou *DOOM* pudessem rodar.

API XMS

O driver XMS funciona de maneira semelhante ao malloc/free da libc e é o mais intuitivo para programadores. Ele permite manipular dados na memória estendida não endereçável por meio de operações como alocar, liberar, realocar e mover. O aspecto central do XMS é que a memória precisa ser copiada entre a RAM estendida e a memória convencional.

API EMS

O driver EMS abre uma janela além da RAM endereçável. A ideia é baseada em mapeamento de memória. O driver permite manipular quatro unidades de 16 KiB chamadas “páginas”, por meio de uma área de 64 KiB denominada Page Frame. Sob demanda ao driver, uma página pode ser trocada para o page frame sem necessidade de cópia.

EMS vs. XMS

A abordagem de mapeamento do EMS era diversas vezes mais rápida do que a abordagem de cópia do XMS. Essa diferença de desempenho influenciou significativamente o gerenciamento de memória do Wolfenstein 3D.

Um sistema “impossível de amar”

Ao longo dos anos, diversas descrições marcantes foram cunhadas sobre essa arquitetura:

> **“The x86 is an architecture that is difficult to explain and impossible to love.”**
>
> — David Patterson & John Hennessy, Computer Organization and Design

> **"That sounds odd, but Intel built it, Microsoft wrote it, and DOS grew up around it."**
>
> — Eccles-Jordan Trigger - Codeproject.com.

> **“Software poison.”**
>
> — Steve Morris, Co-Arquiteto do Intel 8086

Vídeo

Os PCs eram conectados a monitores CRT: grandes, pesados, com diagonal pequena, baseados em tubo de raios catódicos e com superfície curva. A maioria tinha diagonal de 14 polegadas e proporção 4:3.

Histórico dos Adaptadores de Vídeo

O Monochrome Display Adapter (MDA) foi lançado em 1981 com o IBM PC 5150. Oferecia apenas duas cores, permitindo 80 colunas por 25 linhas de texto. Embora modesto, era padrão em todos os PCs. Muitos outros sistemas se seguiram ao longo dos anos, cada um preservando a retrocompatibilidade

Tabela 1: Histórico das interfaces de vídeo

Nome	Ano de Lançamento
MDA (Monochrome Display Adapter)	1981
CGA (Color Graphics Adapter)	1981
EGA (Enhanced Graphics Adapter)	1985
VGA (Video Graphics Array)	1987

Cada iteração adicionou novos recursos, e em 1991 o sistema gráfico predominante era o VGA. Todas as placas de vídeo instaladas em PCs precisavam seguir o padrão definido pela IBM. A universalidade desse sistema era uma faca de dois gumes: enquanto os desenvolvedores tinham apenas um sistema gráfico para programar, não havia como escapar de suas limitações.

Arquitetura VGA

O VGA pode ser resumido em três sistemas principais: entrada, armazenamento e saída

- **Controlador Gráfico e Controlador de Sequência**

  controlam o acesso à RAM do VGA (interface CPU–VRAM)

- **Framebuffer (VRAM)**

  composto por quatro bancos de memória de 64 KiB (em vez de um banco linear de 256 KiB)

- **Controlador CRT e DAC**

  responsável por converter o framebuffer indexado por paleta em RGB e depois em sinal analógico (interface VRAM–CRT)

A parte mais surpreendente da arquitetura é o framebuffer. Por que ter quatro bancos pequenos e fragmentados em vez de um único grande e linear? Parte da explicação está na retrocompatibilidade. O EGA, predecessor do VGA, possuía apenas 64 KiB de RAM, tornando fácil projetar um sistema retrocompatível que utilizasse apenas um banco de 64 KiB.

A razão mais relevante era a latência da RAM e a necessidade de largura de banda mínima. Um pixel era codificado em 4 bits. Cada nibble era traduzido para uma cor RGB 6-6-6 pelo DAC, o que dividia a largura de banda por 4,5 — mas ainda exigia um byte a cada 108 ns. A latência de acesso à RAM era de 200 ns, insuficiente para atualizar a tela a 60 Hz. A solução foi ler dos quatro bancos em paralelo, resultando em uma latência amortizada de 200/4 = 50 ns.

A Loucura Planar do VGA

> **“Right off the bat, I’d like to make one thing perfectly clear: The VGA is hard — sometimes very hard — to program for good performance.”**
>
> — Michael Abrash, Graphics Programming Black Book

O primeiro problema desse design é sua falta de intuitividade. Não há um framebuffer linear e determinar qual byte corresponde a qual pixel na tela é difícil. Essa arquitetura é chamada de planar. No modo 13h, em que um pixel é codificado em um byte, escrever quatro pixels consecutivos em uma linha exige gravar um byte em cada banco — todos mapeados para o mesmo endereço de memória da UMA.

Para configurar esse emaranhado de planos e controladores, são necessários 50 registradores internos mal documentados.

Modos VGA

O BIOS podia ser chamado para configurar o VGA conforme a tabela a seguir:

Tabela 2: Modos VGA disponíveis

Modo	Tipo	Resolução	Cores	RAM (endereço)	Hz
0	texto	40×25	16 (mono)	B8000h	70
1	texto	40×25	16	B8000h	70
2	texto	80×25	16 (mono)	B8000h	70
3	texto	80×25	16	B8000h	70
4	CGA	320×200	4	B8000h	70
5	CGA	320×200	4 (mono)	B8000h	70
6	CGA	640×200	2	B8000h	70
7	MDA	9×14	3 (mono)	B0000h	70
0Dh	EGA	320×200	16	A0000h	70
0Eh	EGA	640×200	16	A0000h	70
0Fh	EGA	640×350	3	A0000h	70
10h	EGA	640×350	16	A0000h	70
11h	VGA	640×480	2	A0000h	70
12h	VGA	640×480	16	A0000h	60
13h	VGA	320×200	256	A0000h	70

Programação VGA: Modo 12h

O modo 12h oferece resolução de 640×480 a 60 Hz com 16 cores. Cada pixel é codificado em 4 bits (um nibble) distribuídos pelos quatro bancos. Para escrever a cor do primeiro pixel, o desenvolvedor precisa escrever o primeiro bit do nibble no plano 0, o segundo no plano 1, o terceiro no plano 2 e o quarto no plano 3. O Controlador CRT então lê 4 bytes de uma vez (um de cada plano), resultando em 8 pixels na tela.

As principais desvantagens desse modo são:

- **Sem duplo buffer: 640 × 480/2 = 0x25800 bytes, mais da metade dos 256 KiB disponíveis de VRAM**
- **Alta resolução implica mais pixels, mais cálculos e mais operações de desenho**
- **Dezesseis cores limitavam severamente os artistas**

Programação VGA: Modo 13h

O modo 13h é mais atraente, pois oferece resolução de 320×200 com 256 cores a 70 Hz. Ele também simula um buffer linear por meio de um chip especial chamado Chain-4, que utiliza os 2 bits menos significativos do endereço de RAM para programar automaticamente a máscara e rotear a operação para o banco de VRAM apropriado.

Para configurar o VGA no modo 13h usando o BIOS, são necessárias apenas duas instruções:

```
mov ax , 0 x13 ; /*AH =0 ( mudar modo de video ), AL =13 h ( modo )*/
2 int 0 x10 ; /*Interrompe o de vídeo da BIOS*/
```

Exemplo de código para limpar a tela em preto:

```
char far * VGA = ( byte far *) 0 xA0000000L ;

void ClearScreen ( void ) {
asm mov ax , 0 x13
asm int 0 x10
for (int i = 0; i < 320 * 200; i ++)
VGA [i ] = 0 x00 ;
}
```

As principais desvantagens do modo 13h são:

- Com o Chain-4 todo o espaço de endereços de RAM é utilizado e não há como fazer duplo buffer
- A proporção 320×200 (1,6) não corresponde à proporção do monitor (1,333), distorcendo o framebuffer ao ser transferido para o CRT

A Importância do Double Buffering

O duplo buffer (double buffering) é fundamental para animações fluidas. Com apenas um buffer, o software precisa operar exatamente na frequência do CRT (70 Hz). Caso contrário, ocorre o fenômeno conhecido como tearing (rasgamento de imagem).

Considere uma animação de um círculo se movendo da esquerda para a direita: a CPU escreve o framebuffer enquanto o feixe de elétrons do CRT já o está varrendo para a tela. Se a CPU for mais rápida que a frequência do CRT, ela pode reescrever o framebuffer antes que a varredura seja concluída, exibindo na tela uma composição de dois quadros diferentes — o resultado visual é como se dois quadros fossem rasgados e remendados, daí o nome tearing.

Com dois buffers, a CPU pode escrever no segundo framebuffer sem interferir no que está sendo varrido para a tela. Resultado: sem tearing.

FIM

Referências bibliográficas:

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