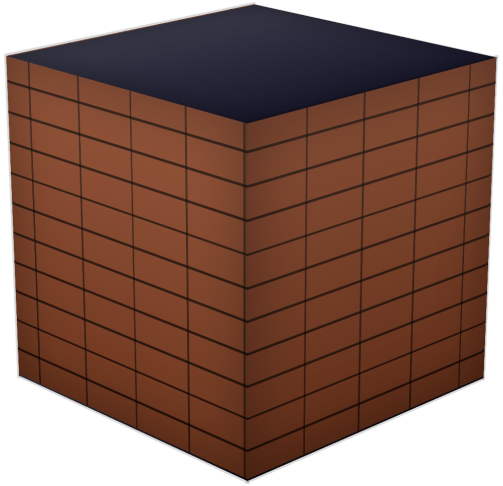
BLACK BOOK 4.7 · RAYCASTING DO WOLFENSTEIN 3D



# Renderizador 3D do Wolfenstein 3D

Como a magia acontecia em um 386

ESTUDO DO CÓDIGO · ID SOFTWARE · 1992

4.7.6 · DISTÂNCIA DIRETA D VS DISTÂNCIA PROJETADA Z

# Correção do Efeito Fisheye

- Raios medidos com a distância **direta (d)** causam distorção nas bordas
- Solução: usar a distância **projetada (z)** no eixo do jogador
- Linhas retas das paredes ficam **geometricamente corretas**
- Comparação visual nos dois próximos slides

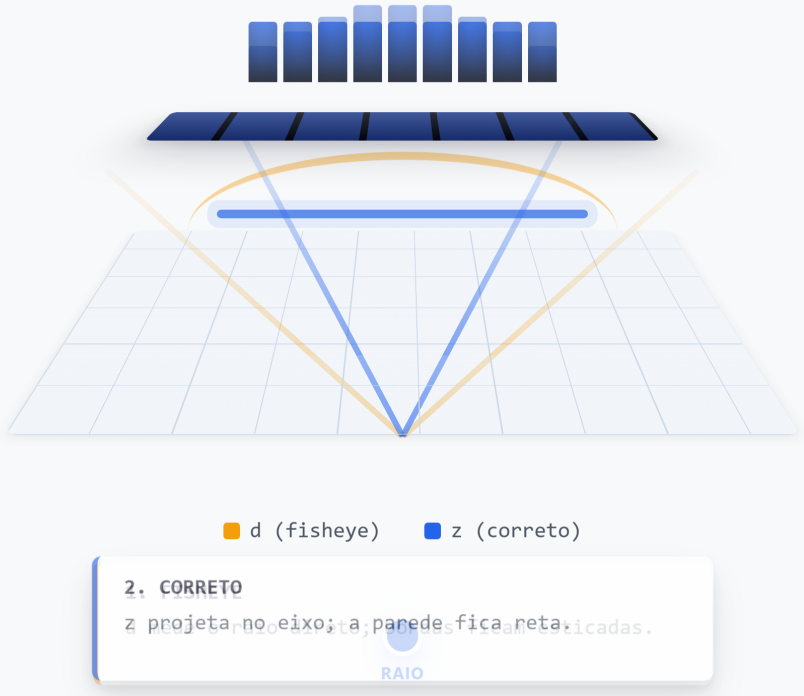
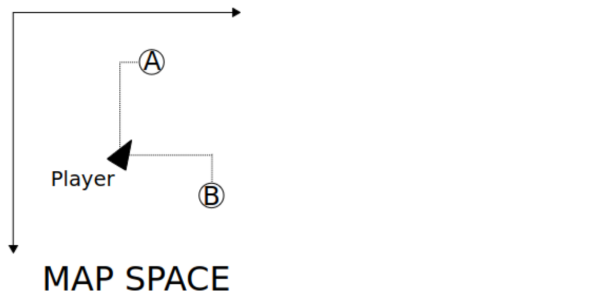
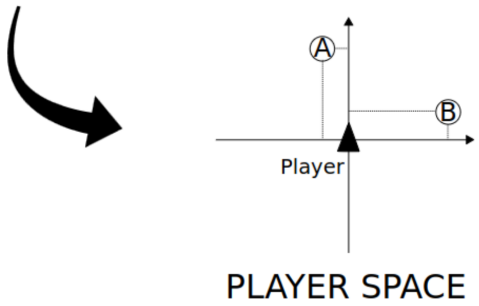


FIGURA 4.53 · TRANSFORMAR COORDENADAS PARA O EIXO DO JOGADOR

# Map Space → Player Space




O raycaster primeiro muda o referencial: o mundo deixa de ser medido pela origem do mapa e passa a ser medido a partir do jogador.

MAP SPACE
**origem fixa do mundo**

PLAYER SPACE
**jogador vira o centro**

COORDENADA Z
**distância projetada no olhar**

RESULTADO
**paredes retas sem fisheye**

1. TRANSLADAR
**hit - jogador**

→

2. ROTACIONAR
**alinhar com visão**

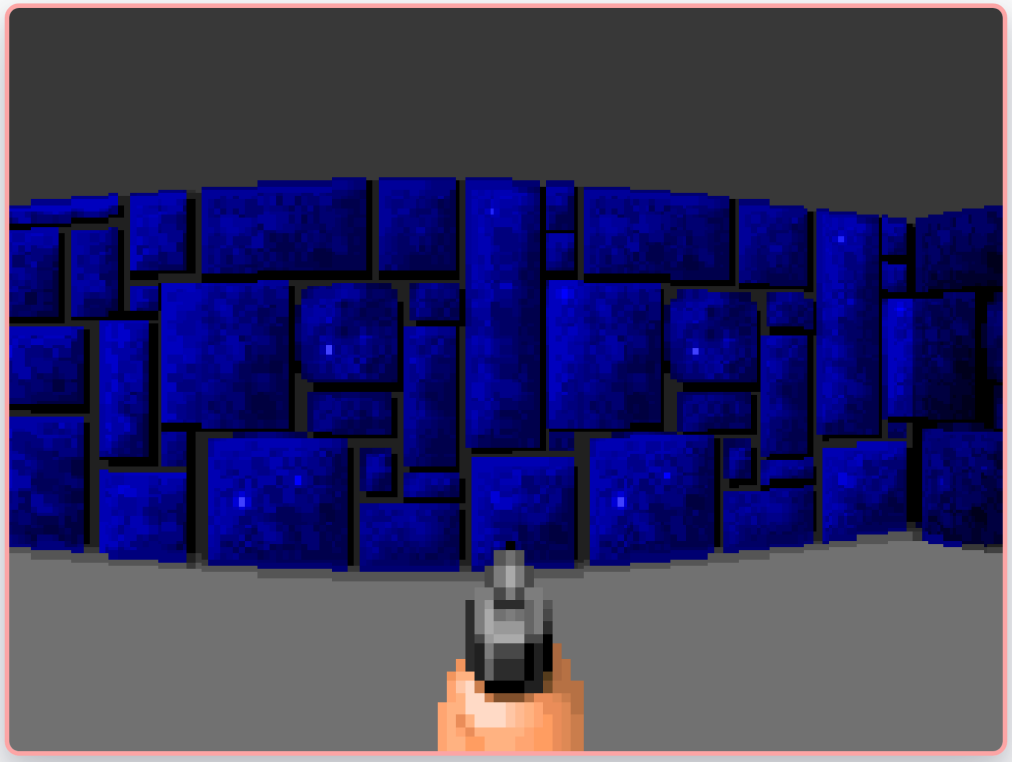
→

3. PROJETAR
**usar z**

COMPARATIVO · PAREDE LONGA REVELA DISTORÇÃO NAS BORDAS

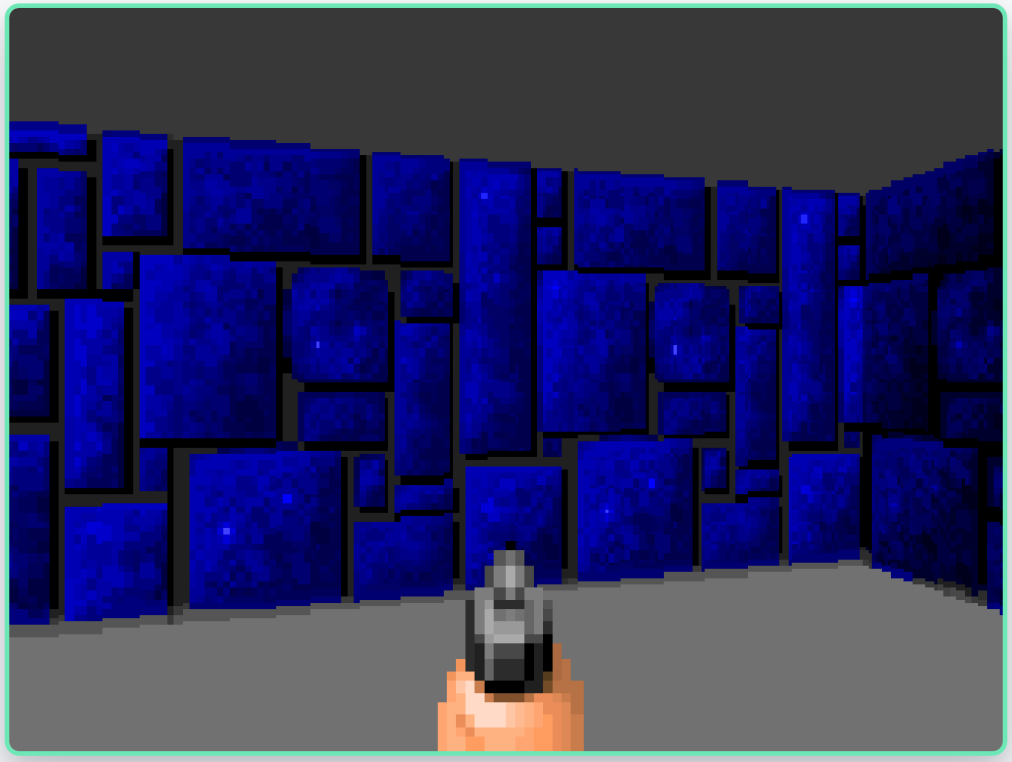
# Antes & Depois — Parede Longa

✗ COM FISHEYE · DISTÂNCIA D



A parede curva visivelmente nas bordas — tijolos esticam onde deveriam estar retos

✓ CORRIGIDO · DISTÂNCIA Z



Linhas retas, perspectiva geométrica correta — a sala parece sólida

COMPARATIVO · PORTA CENTRAL EXPÕE ERRO DE PERSPECTIVA

# Antes & Depois — Porta ao Centro

✗ COM FISHEYE · DISTÂNCIA D

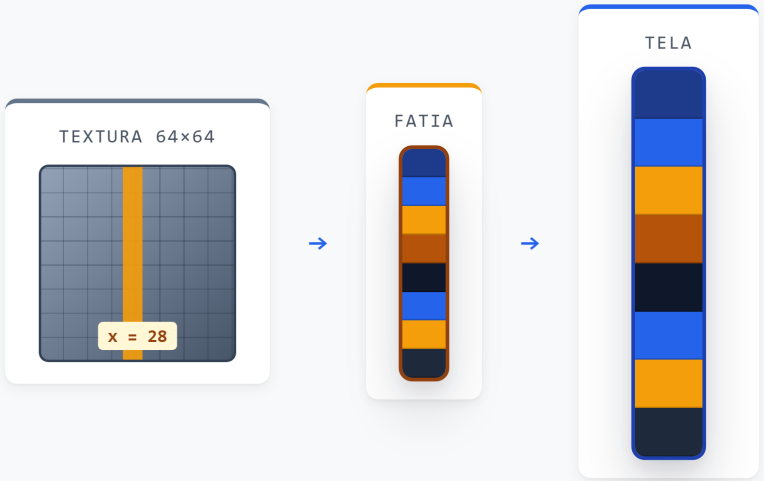
A porta e as paredes laterais aparecem deformadas, esticadas para fora

✓ CORRIGIDO · DISTÂNCIA Z

Porta retangular, paredes paralelas — leitura espacial natural

4.7.7 · CADA COLUNA DA PAREDE VIRA TEXEL ESCALADO

# Desenhando Paredes em um 386




- Altura da coluna calculada → desenhar coluna de **texels escalada**
- CPU 386 é limitada: escalar 64 texels é **caro**
- Wolfenstein supera os engines da época com **2 otimizações**
- Segredo: **Compiled Scalers** + **Deferred Column Rendering**

> o raycaster escolhe uma **fatia vertical** da textura e a estica até a altura da parede na tela

4.7.7.1 · TROCAR CÁLCULO POR SCALERS PRÉ-COMPILADOS

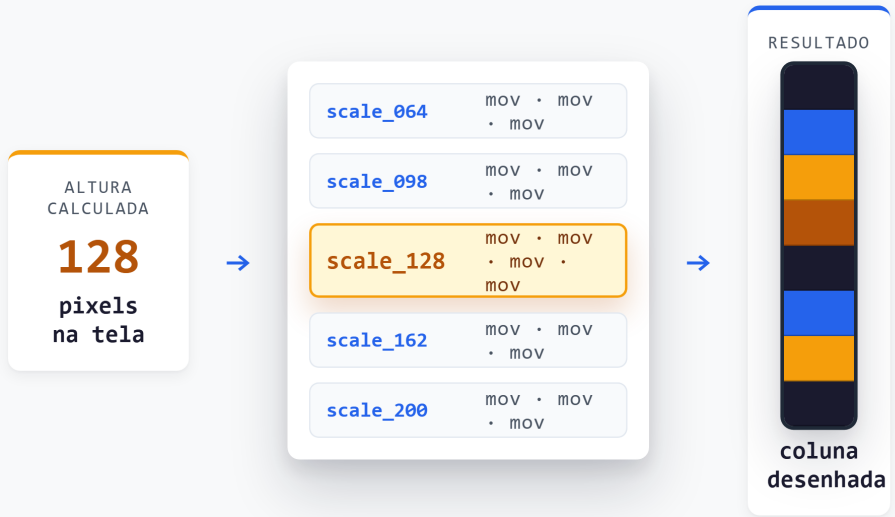
# Compiled Scalers

- Abordagem genérica usa **loop + acumulador** → muitas instruções
- Solução: gerar **funções hard-coded** para cada altura no startup
- 256 funções pré-compiladas **eliminam o loop** completamente
- Custo: **83.160 bytes** de RAM para 136 scalers
- Benefício: de dezenas de instruções para **3-7 por pixel**

GENÉRICO
**~30 instr/pixel**

COMPILED
**3–7 instr/pixel**

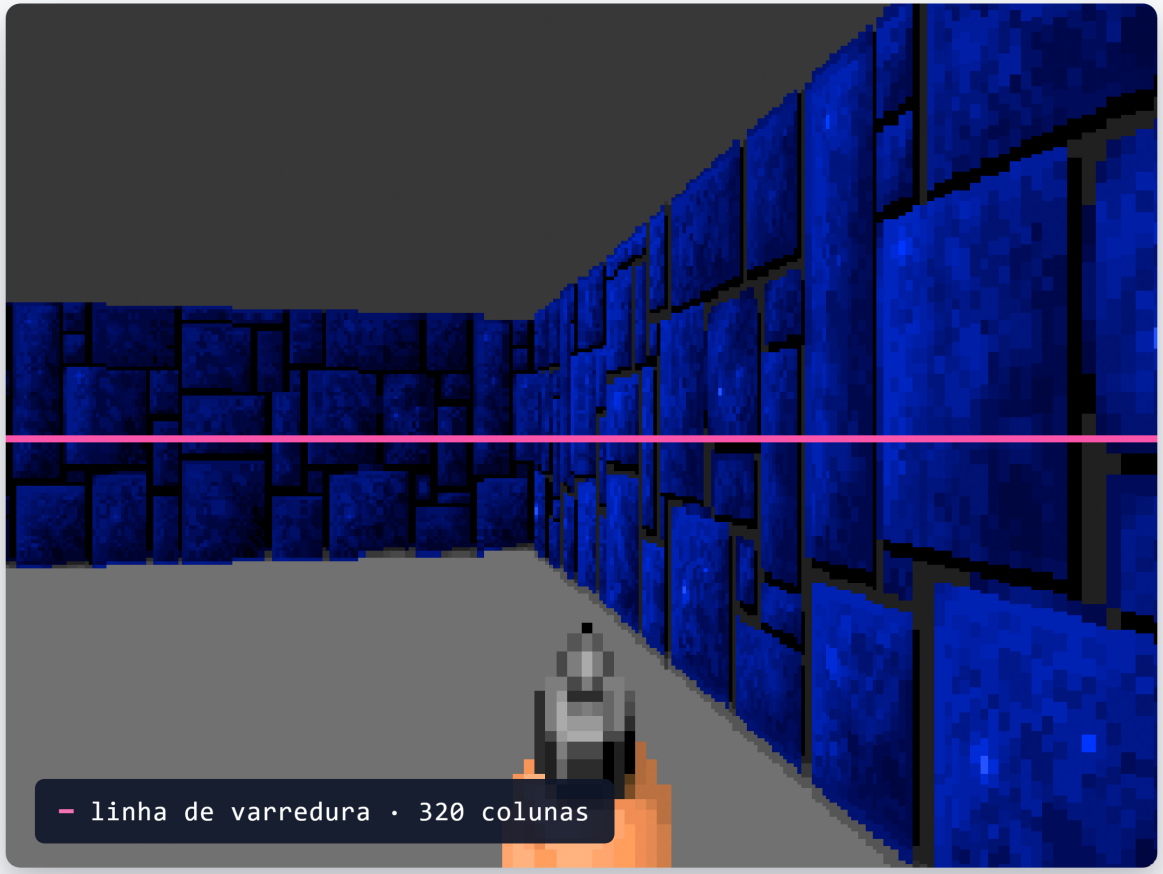
EXEMPLO PRÁTICO




`scale_128` já contém a sequência de cópias. Sem loop interno.

4.7.7.1 · UM SCALER PRONTO PARA CADA ALTURA DE COLUNA

# Cada Coluna, um Scaler Pronto




- Cada **coluna vertical** da tela é um raio lançado
- O raio devolve uma **altura em pixels** para a parede
- O engine seleciona o **scaler compilado** daquela altura
- 320 chamadas por frame, **sem loop interno**

**scale_h(src, dst)** roda 320× por frame — uma chamada por coluna

4.7.7.1 · LOOP GENÉRICO SUBSTITUÍDO POR BYTES X86 EMITIDOS

# Do Loop Genérico ao x86 Compilado

ANTES · GENÉRICO

```
// aceita qualquer altura — caro
void scaleTextureToHeight(int h, ...) {
  fixed_t cur = 0;
  fixed_t step = FixedDiv(64, h);
  while (h > 0) {
    if (not_clipped(d))
      dst[d] = src[cur >> 8];
    cur += step;
    h--; d++;
  }
}
```

▸ loop + jmp + acumulador a cada pixel

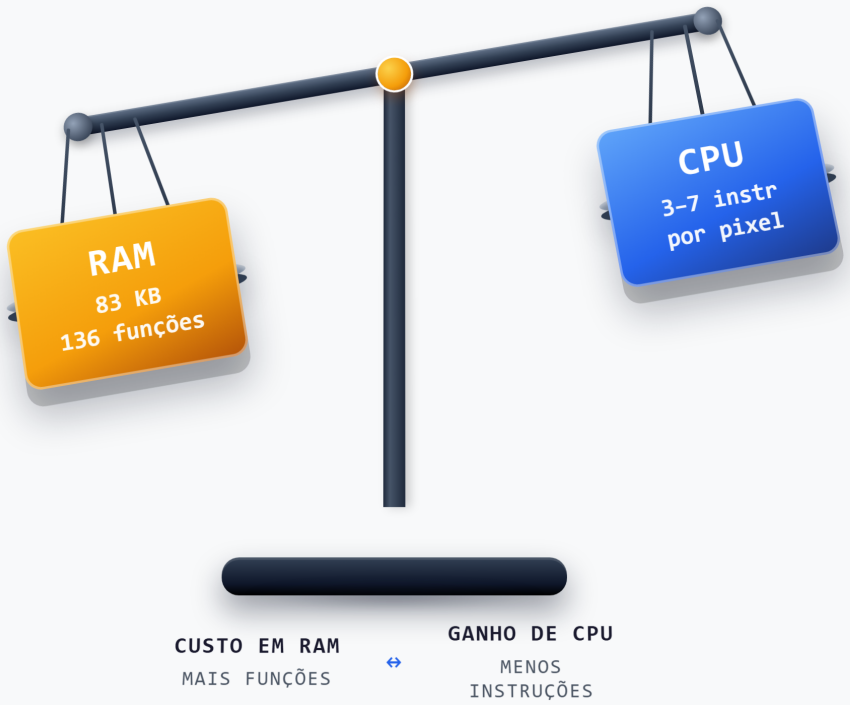
DEPOIS · HARD-CODED

```
// uma função por altura
void scaleTextureTo2(...) {
  dst[0] = src[16];
  dst[1] = src[48];
}
void scaleTextureTo4(...) {
  dst[0] = src[0];
  dst[1] = src[16];
  dst[2] = src[32];
  dst[3] = src[63];
}
```

▸ **BuildCompScale** emite os bytes x86 no startup

4.7.7.1 · GASTAR RAM PARA ECONOMIZAR INSTRUÇÕES POR PIXEL

# RAM vs CPU — O Tradeoff Clássico




- Versão original: 255 scalers = **178.479 bytes** → caro demais
- Otimização: acima de 76px, **pular alturas** (78, 82, 86…)
- Resultado: **136 scalers = 83.160 bytes** — custo aceitável
- Artefato visual **mínimo** ao usar o scaler errado
- Re-geração necessária ao redimensionar o canvas 3D

4.7.7.2 · ADIAR COLUNAS PARA DESENHAR EM LOTES

# Deferred Column Drawing

- Em vez de desenhar cada coluna assim que sai do raycaster...
- O engine **bufferiza** colunas "similares" e dispara em batch
- "Similar" = mesmo **texel horizontal** & mesma altura projetada
- Até **8 colunas** escritas em até **3 operações** no VGA
- Combina com a estrutura do hardware: **4 planos VGA** em paralelo

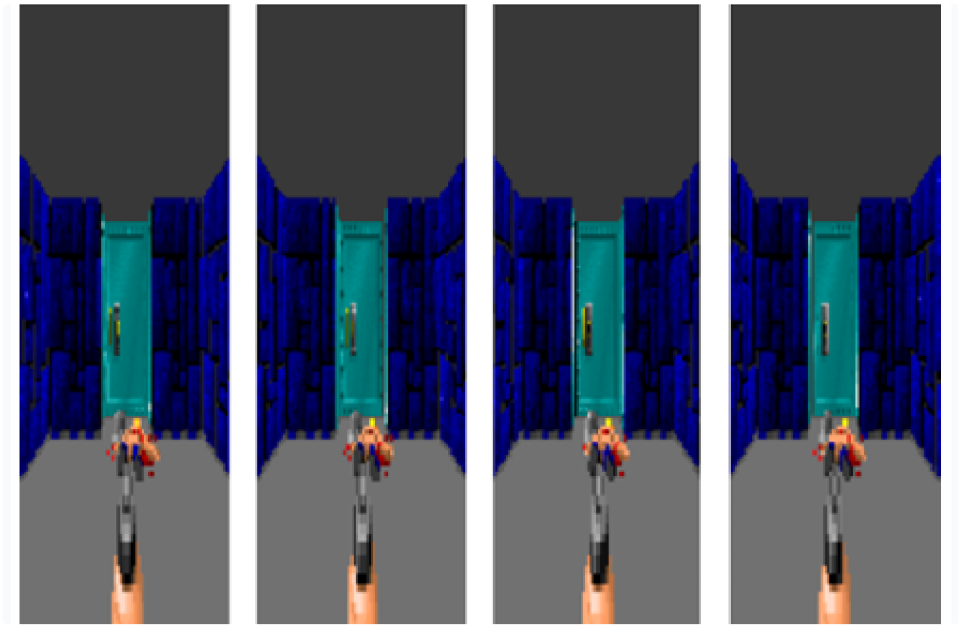
**Ideia central:** trocar muitas escritas pequenas por **poucas escritas largas**, alinhadas aos planos do VGA.


**Deferred**: guarda colunas similares agora, escreve o lote depois.

4.7.7.2 · O RENDERER PENSA EM PLANOS, O JOGADOR VÊ UMA TELA CONTÍNUA

# Quatro Planos, Um Frame

### COMO O RENDERER ORGANIZA



Cada faixa representa um plano VGA: colunas 0,4,8… em um bank; 1,5,9… em outro.

### COMO O JOGADOR RECEBE

No scanout, os quatro planos são intercalados e aparecem como uma imagem normal.

**A vantagem:** o engine agrupa colunas para escrever nos planos VGA, mas o resultado final continua sendo um frame único.

4.7.7.2 · AGRUPAR COLUNAS ANTES DE CHAMAR O RENDERER

# Bufferizar Antes de Desenhar

RAYCASTER INGÊNUO

```
// 1 raio = 1 escrita imediata
for (int x = 0; x < 320; x++) {
  castRay();
  height = CalculateWallHeight();
  drawColumn(x, height);   // ← VGA write
}
```

▸ **320 escritas** no VGA por frame

WOLFENSTEIN · COM BUFFER

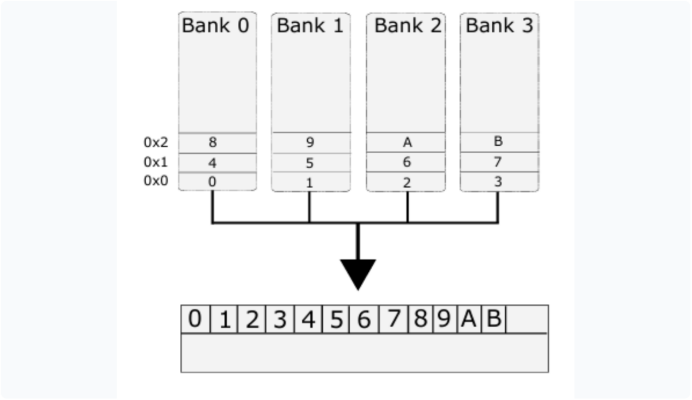
```
// agrupa colunas similares
for (int x = 0; x < 320; x++) {
  castRay();
  if (raySimilarToOnesInBuffer) {
    AddColumnToBuffer();   // adia
    continue;
  } else {
    DrawBuffer();          // ← 1 batch
    height = CalcWallHeight();
    AddColumnToBuffer();
  }
}
```

▸ **~½ das escritas** — colunas vizinhas viram **uma**

4.7.7.2 · MÁSCARAS VGA CONTROLAM QUAIS PIXELS SÃO ESCRITOS

# VGA Plane Masking

**COLUNAS INTERCALADAS NA MEMÓRIA**




Bank 0 escreve 0,4,8…; Bank 1 escreve 1,5,9…

- Tela em **4 banks VGA**: cols 0,4,8 → Bank 0; 1,5,9 → Bank 1…
- Uma escrita pode atingir **até 4 colunas** de uma vez
- A **máscara de bits** seleciona quais banks recebem o byte
- Função **ScalePost** aplica a máscara, escreve, e roda o scaler compilado
- Cada pass = (set mask) → (call compiled scaler)

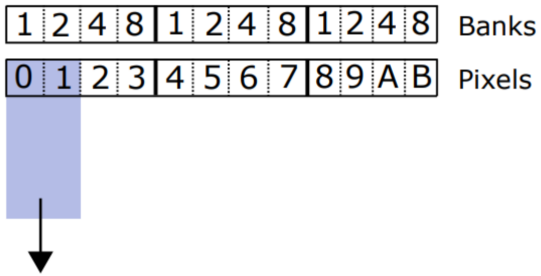
```
// ScalePost – VGA version (resumido)
asm mov al, BYTE PTR [mapmasks1-1+bx]  // pass 1
asm out dx, al                         // set VGA mask
asm call DWORD PTR [bp]                // compiled scaler

asm mov al, BYTE PTR [mapmasks2-1+bx]  // pass 2
asm or  al, al
asm jz  nomore                         // mask=0 ⇒ sai
asm out dx, al
asm call DWORD PTR [bp]
// pass 3 idem...
```

4.7.7.2 · MAPMASKS DECIDE 1, 2 OU 3 PASSES

# Quantos Passes? Depende do Alinhamento

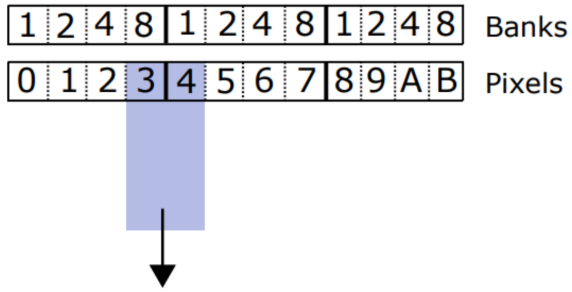
### CASO IDEAL · 1 PASS




pixels 0-1 · banks 0+1

mask = 3 · 1 escrita

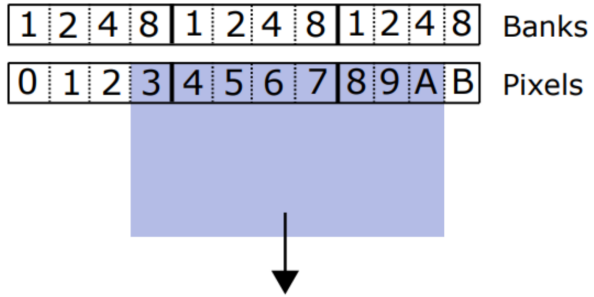
### DESALINHADO · 2 PASSES




pixels 3-4 · cruza fronteira

$mask_1$ = 8 · $mask_2$ = 1

### PIOR CASO · 3 PASSES




pixels 3-A · 8 colunas

$mask_1$=8 · $mask_2$=15 · $mask_3$=7

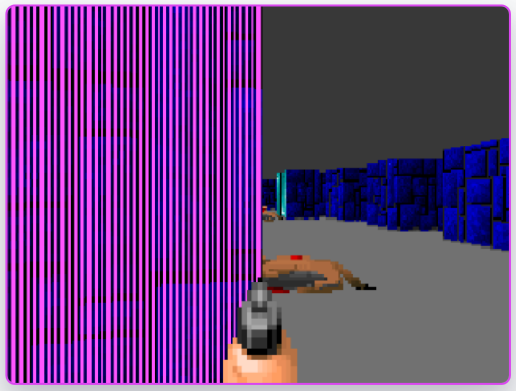
**IDEIA** o engine já sabe qual máscara usar para cada alinhamento — **sem calcular no meio do frame**

4.7.7.2 · ESCRITAS EM ROSA SÃO PULADAS PELO RENDERER

# Tudo em rosa é uma escrita evitada

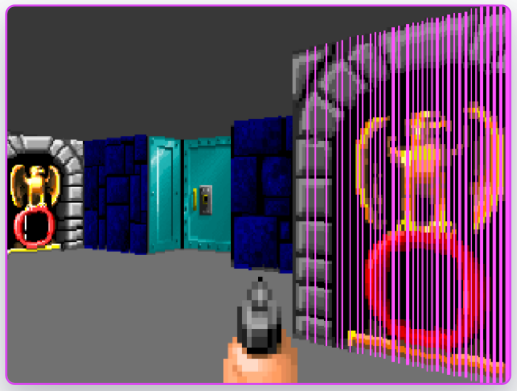
CENA 1 · ORIGINAL

CENA 1 · DEBUG ROSA



CENA 2 · ORIGINAL

CENA 2 · DEBUG ROSA



- Cada faixa rosa é uma coluna que o engine **NÃO** precisou escrever — veio de graça com a vizinha
- O ganho cresce quando o jogador está **perto da parede** (mais colunas idênticas adjacentes)

ESCRITAS EVITADAS
**~50%**

CUSTO EXTRA
**comparação por raio**

TRIVIA · DOOM '93
"low detail mode" duplica colunas (160→320) usando o mesmo truque do VGA para rodar em 386

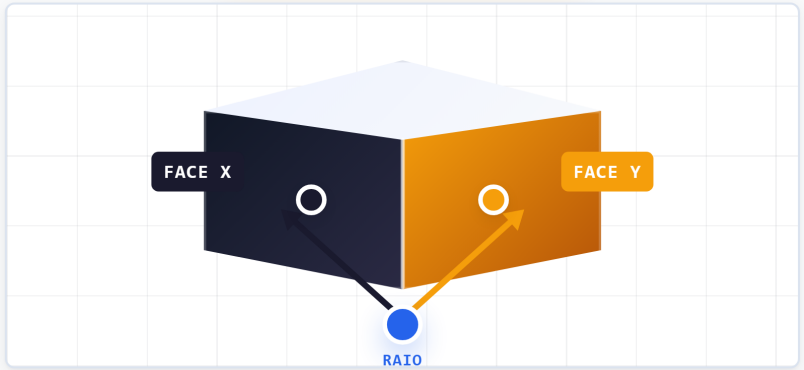
4.7.7.3 · LUZ SIMULADA ESCOLHENDO TEXTURA X OU Y

# Baked Light Texturing

- Sem GPU, sem shaders — iluminação em tempo real era **impossível** em 386
- Solução: pré-computar a luz direto no **asset**
- Cada textura existe **duas vezes**: versão **lit** e versão **unlit**
- O eixo de impacto do raio decide qual delas usar
- Custo em runtime: **zero** — apenas um if no raycaster

```
// dentro do raycaster
if (hit_axis == Y_AXIS)
  tex = textures[id].lit;
else
  tex = textures[id].unlit;
```

luz já vem pintada na textura




HIT_AXIS == X
**unlit**

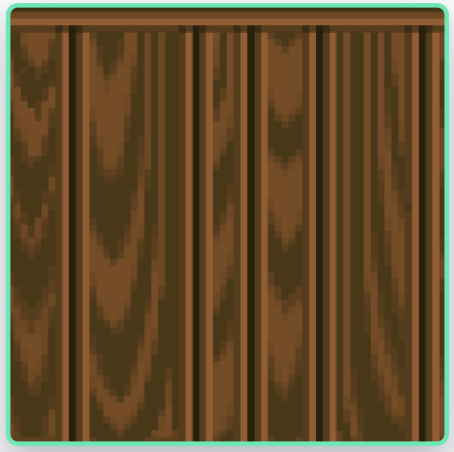
HIT_AXIS == Y
**lit**

O DDA já sabe o eixo atingido; o renderer só escolhe a versão da textura.

4.7.7.3 · FIGURA 4.55 — VERSÃO LIT E UNLIT POR TEXTURA

# Uma Textura, Duas Versões

▲ LIT · PAREDES EM Y



cores claras
"luz batendo de cima"

▼ UNLIT · PAREDES EM X

paleta escurecida
"face em sombra"

- Mesma arte, **duas paletas** — lit ~30% mais clara
- O raycaster sabe se acertou em **X ou Y** de graça (vem do DDA)
- Esse bit decide qual versão da textura amostrar
- Custo em RAM: **2× textures** · custo em CPU: **0**

**Sol implícito** — uma luz direcional em **+Y**, embutida na arte e nunca calculada

4.7.7.3 · FIGURA 4.56 — BAKED LIGHTING MUDA A LEITURA DA CENA

# Sem & Com Baked Lighting

✗ SEM BAKED · TUDO LIT

Paredes parecem **chapadas** — difícil distinguir cantos e profundidade

✓ COM BAKED · X ESCURO / Y CLARO

Cantos saltam à vista — a cena ganha **volume** e leitura espacial

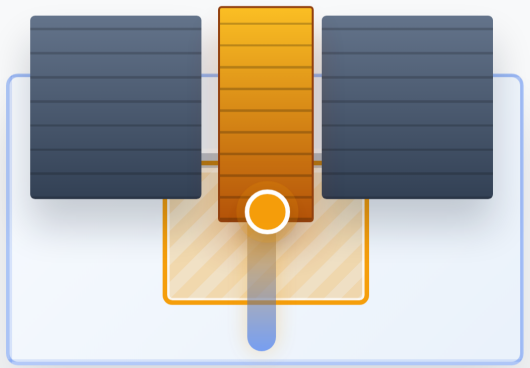
CUSTO EM RUNTIME
**0 ciclos**

CUSTO EM RAM
**2× texturas**

PRINCÍPIO
Trocar trabalho de runtime por trabalho de **artista**

4.7.7.4 · PORTA DEIXA O TILE PARCIALMENTE ATRAVESSÁVEL

# O Tile Deixa de Ser Binário



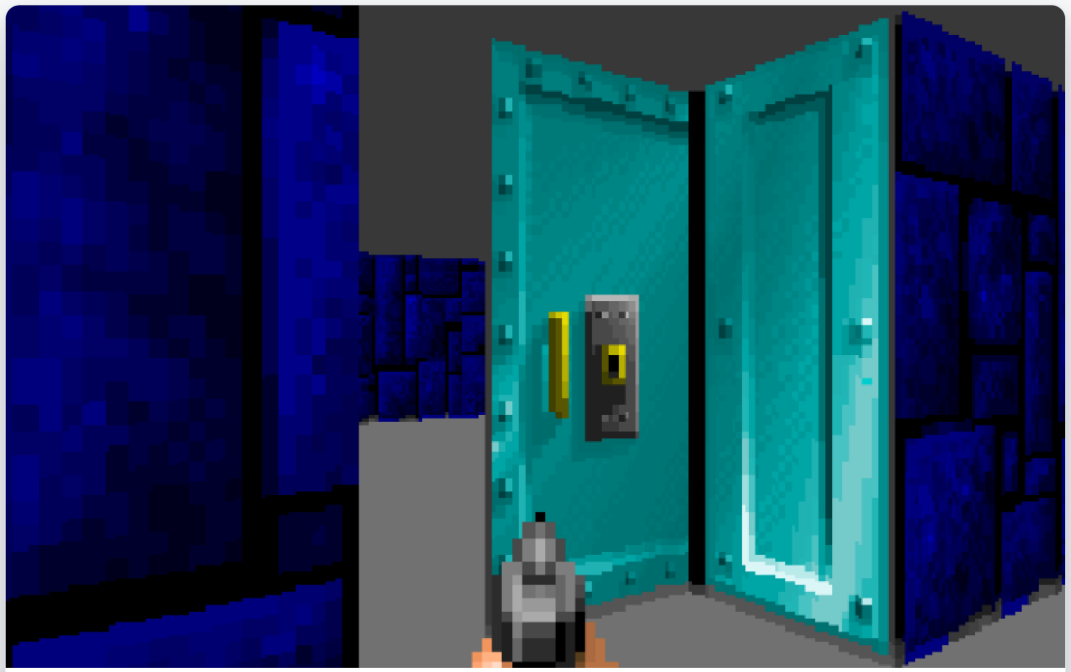
**A face desliza dentro do tile:** o raio testa onde ela está agora.

- Até aqui, cada passo do DDA tinha uma resposta simples: **vazio continua**, **parede para**
- A porta cria um terceiro caso: ela ocupa o tile, mas sua **face pode estar no meio do caminho**
- Uma porta fechada se comporta como parede; uma porta aberta se comporta como espaço
- No meio da animação, o raycaster precisa testar **a posição atual da face**, não só o tipo do tile

| VAZIO | PAREDE | PORTA |
|---|---|---|
| **o raio segue para o próximo tile** | **o raio para e vira coluna** | **o raio compara com a abertura** |

4.7.7.4 · FACE MÓVEL DA PORTA AJUSTA A COLISÃO DO RAIO

# A Porta Parece 3D, Mas É Um Caso do Raycaster



A porta desliza dentro do tile; o mundo continua sendo testado coluna por coluna.

GEOMETRIA

**não é uma malha 3D; é uma face móvel**

FECHADA

**o raio para como em uma parede comum**

ABERTA

**parte do tile vira passagem no mapa**

MEIO TERMO

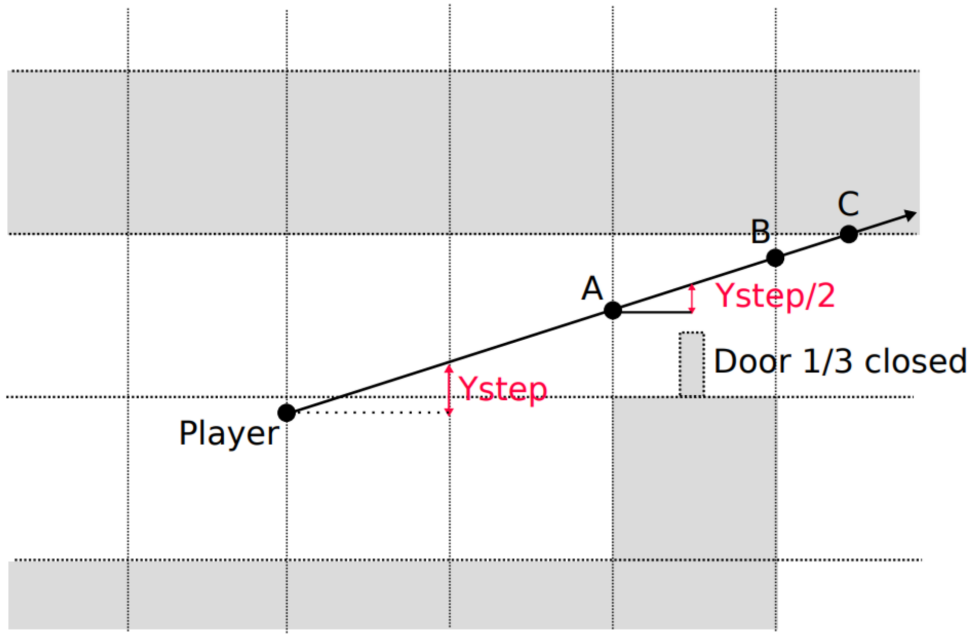
**cada raio decide se bate ou atravessa**

`tile == DOOR → consultar doorposition`

O tile informa o tipo; o array informa quanto da porta ainda está no caminho.

4.7.7.4 · FIGURA 4.57 — DOORPOSITION DECIDE COLISÃO

# O Teste Que Decide: Bate ou Atravessa?




O raio avança pelo grid; a colisão depende de onde está a borda móvel da porta.

$$A_X + \frac{ystep}{2} < doorposition[doorIndex]$$

Se passar, o raio atravessa o tile e continua o DDA.

AX

**posição acumulada do raio dentro do tile**

YSTEP / 2

**desloca o teste para a metade correta**

DOORPOSITION

**borda dianteira da porta deslizante**

INTERVALO

**0 fechada, 0xFFFF totalmente aberta**

4.7.7.4 · ESTADO DA PORTA VIRA INTERCEPT PARA DESENHAR A COLUNA

# A Porta É Estado + Um Intercept Ajustado

```
// max number of sliding doors
#define MAXDOORS 64

// leading edge of door
// 0 = closed, 0xFFFF = fully open
unsigned doorposition[MAXDOORS];
```

```
if (tile == DOOR) {
  edge = AX + ystep / 2;

  if (edge < doorposition[doorIndex])
    continue_ray();
  else
    hit_door();
}
```

$$Intercept_X = A_X + \frac{ystep}{2}$$

$$Intercept_Y = A_Y + \frac{TILE\_SIZE}{2}$$

Quando o raio bate, essas coordenadas localizam a face da porta para o scaler desenhar a coluna correta.

- O raycaster não cria geometria nova; ele entrega um ponto de colisão mais preciso
- O renderer usa esse ponto para escolher textura, coluna e altura projetada
- A ilusão vem de tratar uma animação 2D como uma parede dinâmica

4.7.7.5 · PUSH WALL DESLOCA A FACE ANTES DO TESTE DO RAIO

# Push Walls: A Mesma Ideia, Com Offset


**Push wall: a face "anda" dentro do tile**

`intercept' = intercept + pushOffset`

**pushOffset:** animação (0 → 1 tile) aplicada **antes** do teste do raio

**efeito:** o mesmo tile parece virar uma parede móvel no corredor

- Push walls também são resolvidas dentro do **raycaster**, não por um sistema físico separado
- Quando ativadas, elas continuam associadas ao tile original, mas sua face avança
- O truque é somar um **offset** nas coordenadas de intercept do raio
- Para o renderer, parece que a parede realmente saiu andando pelo corredor

```
if (tile == PUSHWALL && pushwall.active) {
  interceptX += pushOffsetX;
  interceptY += pushOffsetY;
  hit_pushwall();
}
```

| ENTRADA | AJUSTE | SAÍDA |
|---|---|---|
| tile especial | offset animado | coluna renderizada |