Raylib CPP

Uma biblioteca C++ wrapper moderna para Raylib, fornecendo uma interface orientada a objetos completa para desenvolvimento de jogos e aplicações gráficas 2D e 3D com alta performance.

Índice Completo

- Window Management
- Drawing & Rendering
- Textures & Images
- Text & Fonts
- Shapes 2D
- Input System
- Audio System
- Camera System
- 3D Drawing
- Models & Animations
- Collision Detection
- Math & Vectors
- Colors
- Timing & FPS
- File System
- Random & Noise
- Shaders & Materials
- VR & 3D Audio
- Performance & Optimization

Window Management

Window - Criação e Controle

```
Window window(800, 600, "Meu Jogo");
window.SetTargetFPS(60);
```

Quando usar: Primeira coisa a fazer, antes de qualquer operação gráfica.
Parâmetros:

- `width` (int): Largura em pixels (mínimo 1)
- `height` (int): Altura em pixels (mínimo 1)
- `title` (const char*): Título da janela

Funções de Controle:

```
// Loop principal
window.ShouldClose()                    // true quando ESC ou botão fechar
window.IsReady()                        // Verifica inicialização completa

// FPS e Timing
window.SetTargetFPS(60)                 // Define FPS alvo (0 = ilimitado)
GetFPS()                                // Retorna FPS atual
GetFrameTime()                          // Tempo do frame em segundos
GetTime()                               // Tempo desde inicialização

// Fullscreen
window.IsFullscreen()                   // Verifica estado fullscreen
window.ToggleFullscreen()               // Alterna fullscreen/windowed
window.SetFullscreen()                  // Força fullscreen
window.SetWindowed()                    // Força windowed

// Redimensionamento
window.SetSize(1920, 1080)              // Define novo tamanho
window.GetWidth()                       // Largura atual
window.GetHeight()                      // Altura atual
window.GetScreenWidth()                 // Largura da tela física
window.GetScreenHeight()                // Altura da tela física

// Posicionamento
window.SetPosition(100, 100)            // Posiciona na tela
GetWindowPosition()                     // Retorna Vector2 com posição

// Minimização
window.IsMinimized()                    // Verifica se minimizada
window.Minimize()                       // Minimiza janela
window.Maximize()                       // Maximiza janela
window.Restore()                       // Restaura tamanho normal

// Estado e Foco
window.IsFocused()                      // Verifica se tem foco
window.IsResized()                      // true se foi redimensionada no frame
window.IsHidden()                       // Verifica se está oculta
window.IsState(flag)                    // Verifica flag específica

// Título e Ícone
window.SetTitle("Novo Título")          // Muda título
SetWindowIcon(image)                    // Define ícone (Image)

// Monitor
GetCurrentMonitor()                     // ID do monitor atual
GetMonitorCount()                       // Quantidade de monitores
GetMonitorWidth(monitor)                // Largura do monitor
GetMonitorHeight(monitor)               // Altura do monitor
GetMonitorPhysicalWidth(monitor)        // Largura física em mm
GetMonitorPhysicalHeight(monitor)       // Altura física em mm
GetMonitorRefreshRate(monitor)          // Taxa de atualização (Hz)
GetMonitorName(monitor)                 // Nome do monitor
```

```
// Clipboard
GetClipboardText()                    // Lê texto da área de transferência
SetClipboardText("texto")             // Escreve na área de transferência
```

Flags de Configuração

```
SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);
InitWindow(800, 600, "Game");
```

IMPORTANTE: Configurar ANTES de criar a janela.

Flags Disponíveis:

Window Flags:

```
FLAG_WINDOW_RESIZABLE          // Permite redimensionar
FLAG_WINDOW_UNDECORATED        // Sem barra de título e bordas
FLAG_WINDOW_TRANSPARENT        // Fundo transparente
FLAG_WINDOW_HIDDEN             // Janela oculta
FLAG_WINDOW_MINIMIZED          // Inicia minimizada
FLAG_WINDOW_MAXIMIZED          // Inicia maximizada
FLAG_WINDOW_UNFOCUSED          // Inicia sem foco
FLAG_WINDOW_TOPMOST            // Sempre no topo
FLAG_WINDOW_HIGHDPI            // Suporte para telas HighDPI
FLAG_WINDOW_MOUSE_PASSTHROUGH  // Mouse "atravessa" janela
FLAG_WINDOW_ALWAYS_RUN         // Continua rodando em background
```

Rendering Flags:

```
FLAG_VSYNC_HINT               // Sincronização vertical
FLAG_FULLSCREEN_MODE          // Fullscreen na inicialização
FLAG_BORDERLESS_WINDOWED_MODE  // Fullscreen sem bordas
FLAG_MSAA_4X_HINT             // Anti-aliasing 4x
FLAG_INTERLACED_HINT          // Modo entrelaçado
```

Exemplos de Uso:

```
// Janela redimensionável com VSync
SetConfigFlags(FLAG_WINDOW_RESIZABLE | FLAG_VSYNC_HINT);

// Janela sem bordas, sempre no topo
SetConfigFlags(FLAG_WINDOW_UNDECORATED | FLAG_WINDOW_TOPMOST);

// Fullscreen com anti-aliasing
SetConfigFlags(FLAG_FULLSCREEN_MODE | FLAG_MSAA_4X_HINT);
```

Estados de Janela (Runtime)

```
SetWindowState(FLAG_WINDOW_RESIZABLE);  // Adiciona flag
ClearWindowState(FLAG_WINDOW_RESIZABLE); // Remove flag
```

Quando usar: Para mudar propriedades em tempo de execução.

Screenshot

```
TakeScreenshot("screenshot.png");       // Salva screenshot
```

Formato: Automaticamente PNG.
Quando usar: Sistema de fotos, debug, save states visuais.

Drawing & Rendering

Ciclo de Desenho Básico

```
while (!window.ShouldClose()) {
    // UPDATE
    // Sua lógica de jogo aqui

    // DRAW
    window.BeginDrawing();
        window.ClearBackground(RAYWHITE);

        // Todo seu código de renderização
        DrawText("FPS: " + std::to_string(GetFPS()), 10, 10, 20, GREEN);

    window.EndDrawing();
}
```

Estrutura obrigatória: Begin -> Clear -> Draw -> End

ClearBackground:

```
window.ClearBackground(RAYWHITE);
window.ClearBackground(Color{25, 25, 25, 255});
```

Quando usar: Sempre no início de cada frame para limpar buffer anterior.

Render Textures (Offscreen Rendering)

```
RenderTexture2D target = LoadRenderTexture(800, 600);

// Renderiza para texture
BeginTextureMode(target);
    ClearBackground(BLANK);
    DrawCircle(400, 300, 50, RED);
EndTextureMode();

// Usa a render texture
BeginDrawing();
    DrawTexture(target.texture, 0, 0, WHITE);

    // Ou com shader
    BeginShaderMode(blurShader);
        DrawTexture(target.texture, 0, 0, WHITE);
    EndShaderMode();
EndDrawing();

UnloadRenderTexture(target);
```

Quando usar:

- **Post-processing effects:** Blur, bloom, distorção
- **Minimaps:** Renderiza cena de cima
- **Espelhos/Portais:** Renderiza de outro ângulo
- **UI complexa:** Renderiza UI separada e combina
- **Sombras:** Shadow mapping
- **Partículas:** Additive blending

Parâmetros:

- `width, height`: Tamanho do buffer offscreen
- Consome VRAM proporcional ao tamanho

Acesso à textura:

```
target.texture        // Texture2D final
target.texture.width  // Largura
target.texture.height // Altura
target.id             // ID do framebuffer OpenGL
target.depth.id       // ID do depth buffer
```

Blend Modes

```
BeginBlendMode(BLEND_ALPHA);
    DrawTexture(sprite, x, y, WHITE);
EndBlendMode();
```

Modos disponíveis:

```
BLEND_ALPHA          // Alpha blending padrão (transparência)
BLEND_ADDITIVE       // Cores são somadas (brilho)
BLEND_MULTIPLIED     // Cores são multiplicadas (sombra)
BLEND_ADD_COLORS     // Soma sem alpha
BLEND_SUBTRACT_COLORS // Subtrai cores
BLEND_ALPHA_PREMUL   // Alpha pré-multiplicado
BLEND_CUSTOM         // Define manualmente
BLEND_CUSTOM_SEPARATE // Define RGB e Alpha separados
```

Quando usar cada modo:

- **ALPHA:** Sprites normais, UI, texto
- **ADDITIVE:** Fogo, lasers, luz, partículas brilhantes
- **MULTIPLIED:** Sombras, escurecimento, overlays
- **SUBTRACT:** Efeitos negativos, dano, veneno

Exemplo de partículas aditivas:

```
BeginBlendMode(BLEND_ADDITIVE);
    for (auto& particle : particles) {
        DrawTexture(glowTexture, particle.x, particle.y,
                    ColorAlpha(YELLOW, particle.alpha));
    }
EndBlendMode();
```

Scissor Mode (Clipping)

```
BeginScissorMode(100, 100, 400, 300);
    // Apenas esta região será desenhada
    DrawRectangle(0, 0, 800, 600, RED); // Só aparece dentro da área
EndScissorMode();
```

Quando usar:

- **Scroll containers:** Lista com scroll
- **Split screen:** Divide tela em regiões
- **Minimaps:** Limita área de desenho
- **UI clipping:** Text overflow, panels

Limitações:

- Sempre retangular (não pode rotacionar)
- Coordenadas de tela (não relativas à câmera)

- Não funciona com objetos 3D rotacionados

Exemplo de lista com scroll:

```
Rectangle listArea = {50, 50, 300, 400};
float scrollOffset = 0;

BeginScissorMode(listArea.x, listArea.y, listArea.width, listArea.height);
    for (int i = 0; i < items.size(); i++) {
        DrawText(items[i], listArea.x + 10,
                listArea.y + i * 30 - scrollOffset, 20, BLACK);
    }
EndScissorMode();
```

Drawing Layers & Z-Order

```
// Camada de fundo
DrawTexture(background, 0, 0, WHITE);

// Camada de jogo
for (auto& obj : gameObjects) {
    obj.Draw();
}

// Camada de UI (sempre por cima)
DrawRectangle(10, 10, 200, 100, Fade(BLACK, 0.7f));
DrawText("Score: 1000", 20, 20, 20, WHITE);
```

Ordem de desenho:

1. Tudo é desenhado na ordem das chamadas (Painter's Algorithm)
2. Último desenhado aparece na frente
3. Para 3D, use depth buffer automático

Para ordenação complexa:

```
std::vector<GameObject*> allObjects;
// Ordena por profundidade Y
std::sort(allObjects.begin(), allObjects.end(),
    [](GameObject* a, GameObject* b) {
        return a->position.y < b->position.y;
    });

// Desenha em ordem
for (auto* obj : allObjects) {
    obj->Draw();
}
```

Textures & Images

Texture2D - GPU Rendering

```
Texture2D texture = LoadTexture("sprite.png");
DrawTexture(texture, 100, 100, WHITE);
UnloadTexture(texture);
```

Funções de Desenho:

```
// Básico
DrawTexture(texture, x, y, tint);
DrawTextureV(texture, Vector2{x, y}, tint);

// Com retângulo de origem (sprite sheets)
DrawTextureRec(texture,
    Rectangle{srcX, srcY, srcWidth, srcHeight},  // Região da textura
    Vector2{destX, destY},                       // Posição destino
    tint);

// Controle completo (Pro)
DrawTexturePro(texture,
    Rectangle{srcX, srcY, srcWidth, srcHeight},  // Origem
    Rectangle{destX, destY, destWidth, destHeight}, // Destino (pode escalar)
    Vector2{originX, originY},                   // Ponto de rotação
    rotation,                                    // Graus
    tint);

// N-Patch (9-Slice para UI)
DrawTextureNPatch(texture, ninePatchInfo, destRec, origin, rotation, tint);
```

Exemplo de Sprite Sheet:

```
Texture2D spriteSheet = LoadTexture("characters.png");

// Cada frame é 32x32
Rectangle frameRec = {0, 0, 32, 32};
int currentFrame = 0;
float frameTime = 0;

// No update
frameTime += GetFrameTime();
if (frameTime >= 0.1f) {  // 10 FPS de animação
    currentFrame++;
    if (currentFrame > 7) currentFrame = 0;
    frameRec.x = currentFrame * 32;
    frameTime = 0;
```

```
}

// No draw
DrawTextureRec(spriteSheet, frameRec, position, WHITE);
```

Exemplo de Rotação e Escala:

```
Rectangle source = {0, 0, texture.width, texture.height};
Rectangle dest = {x, y, texture.width * 2, texture.height * 2}; // 2x maior
Vector2 origin = {texture.width, texture.height};  // Centro
float rotation = 45.0f;

DrawTexturePro(texture, source, dest, origin, rotation, WHITE);
```

N-Patch (9-Slice):

```
NPatchInfo ninePatch;
ninePatch.source = {0, 0, 64, 64};
ninePatch.left = 16;    // Borda esquerda
ninePatch.top = 16;     // Borda superior
ninePatch.right = 16;   // Borda direita
ninePatch.bottom = 16;  // Borda inferior
ninePatch.layout = NPATCH_NINE_PATCH;
// Desenha botão escalável sem distorcer bordas
Rectangle destRec = {100, 100, 200, 60};  // Qualquer tamanho
DrawTextureNPatch(buttonTexture, ninePatch, destRec, Vector2{0,0}, 0, WHITE);
```

Quando usar N-Patch:

- Botões de UI que precisam escalar
- Painéis e janelas
- Barras de progresso
- Qualquer UI que não pode distorcer bordas

Layouts N-Patch:

```
NPATCH_NINE_PATCH           // 9 regiões (padrão)
NPATCH_THREE_PATCH_VERTICAL // 3 regiões verticais
NPATCH_THREE_PATCH_HORIZONTAL // 3 regiões horizontais
```

Funções de Textura

```
// Carregamento
Texture2D texture = LoadTexture("file.png");
Texture2D texture = LoadTextureFromImage(image);
```

```
// Atualização dinâmica
UpdateTexture(texture, pixels);  // Atualiza toda textura
UpdateTextureRec(texture, rec, pixels);  // Atualiza região

// Geração procedural
Image img = GenImageColor(256, 256, RED);
Texture2D tex = LoadTextureFromImage(img);
UnloadImage(img);

// Propriedades
texture.id         // ID OpenGL da textura
texture.width      // Largura em pixels
texture.height     // Altura em pixels
texture.mipmaps    // Níveis de mipmap
texture.format     // Formato de pixel

// Filtros
SetTextureFilter(texture, TEXTURE_FILTER_POINT);
SetTextureFilter(texture, TEXTURE_FILTER_BILINEAR);
SetTextureFilter(texture, TEXTURE_FILTER_TRILINEAR);
SetTextureFilter(texture, TEXTURE_FILTER_ANISOTROPIC_4X);
SetTextureFilter(texture, TEXTURE_FILTER_ANISOTROPIC_8X);
SetTextureFilter(texture, TEXTURE_FILTER_ANISOTROPIC_16X);

// Wrap mode
SetTextureWrap(texture, TEXTURE_WRAP_REPEAT);
SetTextureWrap(texture, TEXTURE_WRAP_CLAMP);
SetTextureWrap(texture, TEXTURE_WRAP_MIRROR_REPEAT);
SetTextureWrap(texture, TEXTURE_WRAP_MIRROR_CLAMP);

// Mipmaps
GenTextureMipmaps(&texture);  // Gera mipmaps automaticamente

// Descarregar
UnloadTexture(texture);
```

Quando usar cada filtro:

- **POINT:** Pixel art, texturas que devem manter pixels nítidos
- **BILINEAR:** Padrão, bom balanço qualidade/performance
- **TRILINEAR:** Para texturas com mipmaps, transições suaves
- **ANISOTROPIC:** Texturas 3D vistas em ângulo, melhor qualidade

Exemplo de pixel art:

```
Texture2D pixelArt = LoadTexture("sprite_8x8.png");
SetTextureFilter(pixelArt, TEXTURE_FILTER_POINT);  // Mantém pixels nítidos
// Ao escalar, não ficará borrado
DrawTextureEx(pixelArt, pos, 0, 4.0f, WHITE);  // 4x maior, pixels definidos
```

Image - CPU Processing

```
Image img = LoadImage("photo.png");
// Processar...
Texture2D tex = LoadTextureFromImage(img);
UnloadImage(img);  // Libera memória RAM
```

Quando usar Image:

- Processar antes de carregar na GPU
- Gerar texturas proceduralmente
- Manipular pixels individualmente
- Aplicar filtros e efeitos
- Economizar VRAM (carregar só quando necessário)

Funções de Carregamento:

```
Image img = LoadImage("file.png");
Image img = LoadImageRaw("file.raw", width, height, format, headerSize);
Image img = LoadImageSvg("file.svg", width, height);
Image img = LoadImageAnim("file.gif", &frames);  // GIF animado
Image img = LoadImageFromMemory(".png", fileData, dataSize);
Image img = LoadImageFromTexture(texture);  // Copia de volta da GPU
Image img = LoadImageFromScreen();  // Screenshot como Image
```

Manipulação Básica:

```
// Dimensões
ImageResize(&img, newWidth, newHeight);           // Bicubic (qualidade)
ImageResizeNN(&img, newWidth, newHeight);         // Nearest Neighbor (rápido)
ImageResizeCanvas(&img, newWidth, newHeight,
                  offsetX, offsetY, fill);         // Adiciona bordas
ImageCrop(&img, Rectangle{x, y, width, height});  // Recorta

// Transformações
ImageFlipVertical(&img);
ImageFlipHorizontal(&img);
ImageRotateCW(&img);           // 90° horário
ImageRotateCCW(&img);          // 90° anti-horário
ImageMirrored(&img);           // Espelha horizontalmente

// Formato
ImageFormat(&img, newFormat);  // Converte formato de pixel
ImageAlphaClear(&img, color, threshold);
ImageAlphaMask(&img, alphaMask);
ImageAlphaPremultiply(&img);
ImageAlphaCrop(&img, threshold);
```

Efeitos de Cor:

```
ImageColorTint(&img, color);              // Multiplica cor
ImageColorInvert(&img);                   // Inverte cores
ImageColorGrayscale(&img);                // Converte para cinza
ImageColorContrast(&img, contrast);       // -100 a 100
ImageColorBrightness(&img, brightness);   // -255 a 255
ImageColorReplace(&img, color, replace);  // Substitui cor

// Exemplo: Criar variações de cor de um sprite
Image blueVersion = ImageCopy(originalImg);
ImageColorTint(&blueVersion, BLUE);
```

Desenho em Image:

```
ImageDraw(&dst, src, srcRec, dstRec, tint);
ImageDrawRectangle(&img, rec, color);
ImageDrawRectangleV(&img, position, size, color);
ImageDrawRectangleRec(&img, rec, color);
ImageDrawRectangleLines(&img, rec, thick, color);
ImageDrawCircle(&img, centerX, centerY, radius, color);
ImageDrawCircleV(&img, center, radius, color);
ImageDrawLine(&img, startX, startY, endX, endY, color);
ImageDrawLineV(&img, start, end, color);
ImageDrawText(&img, text, x, y, fontSize, color);
ImageDrawTextEx(&img, font, text, position, fontSize, spacing, tint);
```

Exemplo de geração de thumbnail:

```
Image fullImage = LoadImage("highres.png");
ImageResize(&fullImage, 128, 128);
ImageFormat(&fullImage, PIXELFORMAT_UNCOMPRESSED_R8G8B8);  // Remove alpha
ExportImage(fullImage, "thumbnail.png");
UnloadImage(fullImage);
```

Geração Procedural:

```
// Cores sólidas
Image img = GenImageColor(256, 256, RED);

// Gradientes
Image img = GenImageGradientLinear(width, height, direction, start, end);
Image img = GenImageGradientRadial(width, height, density, inner, outer);
Image img = GenImageGradientSquare(width, height, density, inner, outer);
```

```
// Checkerboard
Image img = GenImageChecked(width, height, checksX, checksY, col1, col2);

// Ruído
Image img = GenImageWhiteNoise(width, height, factor);
Image img = GenImagePerlinNoise(width, height, offsetX, offsetY, scale);

// Cellular
Image img = GenImageCellular(width, height, tileSize);

// Texto
Image img = ImageText(text, fontSize, color);
Image img = ImageTextEx(font, text, fontSize, spacing, tint);
```

Exemplo de textura procedural:

```
// Gera textura de ruído para terreno
Image noiseImg = GenImagePerlinNoise(512, 512, 0, 0, 4.0f);
ImageColorGrayscale(&noiseImg);
ImageColorContrast(&noiseImg, 30);

// Adiciona cor baseada na altura
Color* pixels = LoadImageColors(noiseImg);
for (int i = 0; i < noiseImg.width * noiseImg.height; i++) {
    int brightness = pixels[i].r;
    if (brightness < 80) pixels[i] = BLUE;        // Água
    else if (brightness < 120) pixels[i] = GREEN; // Grama
    else if (brightness < 180) pixels[i] = BROWN; // Terra
    else pixels[i] = WHITE;                       // Neve
}
UpdateImageColors(&noiseImg, pixels);
UnloadImageColors(pixels);

Texture2D terrainTex = LoadTextureFromImage(noiseImg);
UnloadImage(noiseImg);
```

Acesso a Pixels:

```
Color* pixels = LoadImageColors(img);     // Carrega array de cores
for (int i = 0; i < img.width * img.height; i++) {
    // Manipula pixels[i]
    pixels[i].r = 255;
}
UpdateTexture(texture, pixels);           // Atualiza textura
UnloadImageColors(pixels);                // Libera memória

// Pixel individual
Color color = GetImageColor(img, x, y);
ImageDrawPixel(&img, x, y, color);
```

```
// Array de pixels em formato raw
unsigned char* rawData = (unsigned char*)img.data;
```

Formatos de Pixel:

```
PIXELFORMAT_UNCOMPRESSED_GRAYSCALE        // 8 bpp (1 byte)
PIXELFORMAT_UNCOMPRESSED_GRAY_ALPHA       // 16 bpp (2 bytes)
PIXELFORMAT_UNCOMPRESSED_R5G6B5           // 16 bpp
PIXELFORMAT_UNCOMPRESSED_R8G8B8           // 24 bpp (3 bytes)
PIXELFORMAT_UNCOMPRESSED_R5G5B5A1         // 16 bpp
PIXELFORMAT_UNCOMPRESSED_R4G4B4A4         // 16 bpp
PIXELFORMAT_UNCOMPRESSED_R8G8B8A8         // 32 bpp (4 bytes) - PADRÃO
PIXELFORMAT_UNCOMPRESSED_R32              // 32 bpp float
PIXELFORMAT_UNCOMPRESSED_R32G32B32        // 96 bpp float
PIXELFORMAT_UNCOMPRESSED_R32G32B32A32     // 128 bpp float
PIXELFORMAT_COMPRESSED_DXT1_RGB           // Compressão S3TC
PIXELFORMAT_COMPRESSED_DXT1_RGBA
PIXELFORMAT_COMPRESSED_DXT3_RGBA
PIXELFORMAT_COMPRESSED_DXT5_RGBA
PIXELFORMAT_COMPRESSED_ETC1_RGB           // Compressão ETC
PIXELFORMAT_COMPRESSED_ETC2_RGB
PIXELFORMAT_COMPRESSED_ETC2_EAC_RGBA
PIXELFORMAT_COMPRESSED_PVRT_RGB           // PowerVR
PIXELFORMAT_COMPRESSED_PVRT_RGBA
PIXELFORMAT_COMPRESSED_ASTC_4x4_RGBA      // ASTC
PIXELFORMAT_COMPRESSED_ASTC_8x8_RGBA
```

Exportação:

```
ExportImage(img, "output.png");
ExportImageAsCode(img, "image.h");  // Como código C
```

Formatos de exportação: PNG, BMP, TGA, JPG, QOI, KTX, DDS

Text & Fonts

DrawText - Texto Básico

```
DrawText("Hello World!", 100, 100, 20, BLACK);
DrawText(TextFormat("Score: %d", score), 10, 10, 20, GREEN);
```

Parâmetros:

- `text`: String (const char*)

- `posX, posY`: Posição em pixels
- `fontSize`: Tamanho em pixels
- `color`: Cor do texto

Limitações da fonte padrão:

- Fonte bitmap embutida
- Qualidade degrada ao escalar
- Apenas caracteres ASCII básicos

Font Customizada

```
Font font = LoadFont("arial.ttf");
Font font = LoadFontEx("arial.ttf", 32, NULL, 0);  // Tamanho específico

DrawTextEx(font, "Custom Font", Vector2{100, 100}, 32, 2, BLACK);

UnloadFont(font);
```

LoadFontEx Parâmetros:

```
LoadFontEx(fileName, fontSize, codepoints, codepointCount);
```

- `fileName`: Caminho do arquivo
- `fontSize`: Tamanho de rasterização (maior = melhor qualidade)
- `codepoints`: Array de caracteres a carregar (NULL = ASCII padrão)
- `codepointCount`: Quantidade de caracteres (0 = todos ASCII)

Exemplo com caracteres específicos:

```
// Carrega apenas números e letras maiúsculas
int codepoints[] = {'0','1','2','3','4','5','6','7','8','9',
                    'A','B','C','D','E','F','G','H','I','J','K','L','M',
                    'N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};
Font font = LoadFontEx("font.ttf", 64, codepoints, 36);
```

Carregar range de Unicode:

```
Font font = LoadFontEx("chinese.ttf", 48, NULL, 0);
// Ou especificar range
int codepointsCount = 0;
int* codepoints = LoadCodepoints("你好世界", &codepointsCount);
Font fontChinese = LoadFontEx("chinese.ttf", 48, codepoints, codepointsCount);
UnloadCodepoints(codepoints);
```

Funções de Desenho de Texto

```
// Texto simples
DrawText(text, x, y, fontSize, color);

// Com font customizada
DrawTextEx(font, text, position, fontSize, spacing, color);

// Com Pro (mais controle)
DrawTextPro(font, text, position, origin, rotation, fontSize, spacing, color);

// Código específico (UTF-8)
DrawTextCodepoint(font, codepoint, position, fontSize, tint);
DrawTextCodepoints(font, codepoints, count, position, fontSize, spacing, tint);
```

Exemplo de texto rotacionado:

```
Vector2 textPos = {400, 300};
Vector2 origin = {MeasureTextEx(font, "ROTATED", 40, 2).x * 0.5f, 20};
DrawTextPro(font, "ROTATED", textPos, origin, 45, 40, 2, RED);
```

Medição de Texto

```
// Fonte padrão
int width = MeasureText("Hello", 20);

// Font customizada
Vector2 size = MeasureTextEx(font, "Hello World", 32, 2);
// size.x = largura, size.y = altura
```

Quando usar:

- Centralizar texto
- Text wrapping
- Verificar se texto cabe em área de caixas de texto dinâmicas

Exemplo de centralização:

```
const char* text = "GAME OVER";
Vector2 textSize = MeasureTextEx(font, text, 60, 3);
Vector2 position = {
    (GetScreenWidth() - textSize.x) / 2.0f,
    (GetScreenHeight() - textSize.y) / 2.0f
};
DrawTextEx(font, text, position, 60, 3, RED);
```

Exemplo de text wrapping:

```
void DrawTextBoxed(Font font, const char* text, Rectangle rec, float fontSize,
float spacing, bool wordWrap, Color tint) {
    int length = TextLength(text);
    float textOffsetY = 0;
    float textOffsetX = 0;

    float scaleFactor = fontSize / font.baseSize;

    for (int i = 0; i < length; i++) {
        int codepointByteCount = 0;
        int codepoint = GetCodepoint(&text[i], &codepointByteCount);
        int index = GetGlyphIndex(font, codepoint);

        float glyphWidth = (font.glyphs[index].advanceX == 0) ?
                           font.recs[index].width * scaleFactor :
                           font.glyphs[index].advanceX * scaleFactor;

        if (textOffsetX + glyphWidth > rec.width) {
            textOffsetX = 0;
            textOffsetY += fontSize + spacing;

            if (textOffsetY > rec.height) break;
        }

        DrawTextCodepoint(font, codepoint,
            Vector2{rec.x + textOffsetX, rec.y + textOffsetY},
            fontSize, tint);

        textOffsetX += glyphWidth + spacing;
        i += codepointByteCount - 1;
    }
}
```

Manipulação de Strings

```
// Cópia e concatenação
const char* text1 = "Hello";
const char* text2 = " World";
const char* result = TextJoin({text1, text2}, 2, " ");  // "Hello World"

// Substring
const char* sub = TextSubtext("Hello World", 0, 5);  // "Hello"

// Substituir
const char* replaced = TextReplace("Hello World", "World", "Raylib");

// Inserir
```

```
const char* inserted = TextInsert("Hello!", " World", 5);  // "Hello World!"

// Maiúsculas/Minúsculas
const char* upper = TextToUpper("hello");  // "HELLO"
const char* lower = TextToLower("HELLO");  // "hello"

// Conversões
int value = TextToInteger("123");
float value = TextToFloat("3.14");

// Formatação
const char* formatted = TextFormat("Player %d: %s", 1, "John");
```

Font Data

```
Font font = LoadFont("font.ttf");

// Propriedades
font.baseSize        // Tamanho base
font.glyphCount      // Quantidade de caracteres
font.glyphPadding    // Padding entre glyphs
font.texture         // Texture atlas com todos glyphs
font.recs            // Array de retângulos no atlas
font.glyphs          // Array de dados de glyph

// Obter informações de glyph específico
int index = GetGlyphIndex(font, 'A');
int glyphWidth = font.glyphs[index].advanceX;
```

Fonte Padrão

```
Font defaultFont = GetFontDefault();
DrawTextEx(defaultFont, "Text", pos, 20, 1, BLACK);
```

Quando usar fonte padrão:

- Debug rápido
- Protótipos
- Jogos de terminal/retro

Quando usar fonts customizadas:

- Identidade visual do jogo
- Suporte a múltiplos idiomas
- Melhor qualidade visual

SDF Fonts (Signed Distance Field)

```
// Carrega como SDF para escalar sem perder qualidade
Font font = LoadFontEx("font.ttf", 96, NULL, 0);
// Renderiza em qualquer tamanho
DrawTextEx(font, "Scalable!", pos, 200, 5, WHITE);
```

Vantagens SDF:

- Escala infinitamente sem perda
- Menor uso de memória
- Efeitos como outline, glow

Desvantagens:

- Mais pesado processamento
- Necessita shader especial para efeitos

Shapes 2D

Retângulos

```
// Filled
DrawRectangle(x, y, width, height, color);
DrawRectangleV(Vector2{x, y}, Vector2{width, height}, color);
DrawRectangleRec(Rectangle{x, y, width, height}, color);

// Lines (contorno)
DrawRectangleLines(x, y, width, height, color);
DrawRectangleLinesEx(Rectangle{x, y, w, h}, lineThick, color);

// Rounded corners
DrawRectangleRounded(Rectangle{x, y, w, h}, roundness, segments, color);
DrawRectangleRoundedLines(rec, roundness, segments, lineThick, color);

// Gradientes
DrawRectangleGradientV(x, y, w, h, color1, color2);  // Vertical
DrawRectangleGradientH(x, y, w, h, color1, color2);  // Horizontal
DrawRectangleGradientEx(rec, col1, col2, col3, col4); // 4 cantos diferentes
```

Parâmetros:

- `roundness`: 0.0 (quadrado) a 1.0 (totalmente arredondado)
- `segments`: Mais segmentos = cantos mais suaves (padrão 36)
- `lineThick`: Espessura da linha em pixels

Quando usar:

- **Filled:** Plataformas, obstáculos, UI backgrounds
- **Lines:** Borders, grids, debug

- **Rounded:** Botões modernos, panels de UI
- **Gradient:** Backgrounds elegantes, barras de vida

Exemplo de barra de vida:

```
// Background (vida perdida)
DrawRectangleRounded(Rectangle{20, 20, 200, 30}, 0.3f, 12, DARKGRAY);

// Foreground (vida atual)
float healthPercent = currentHealth / maxHealth;
DrawRectangleRounded(Rectangle{20, 20, 200 * healthPercent, 30}, 0.3f, 12, RED);

// Border
DrawRectangleRoundedLines(Rectangle{20, 20, 200, 30}, 0.3f, 12, 2, BLACK);
```

Círculos

```
// Filled
DrawCircle(centerX, centerY, radius, color);
DrawCircleV(Vector2{centerX, centerY}, radius, color);

// Lines (contorno)
DrawCircleLines(centerX, centerY, radius, color);

// Sector (fatia)
DrawCircleSector(center, radius, startAngle, endAngle, segments, color);
DrawCircleSectorLines(center, radius, startAngle, endAngle, segments, color);

// Ring (anel)
DrawRing(center, innerRadius, outerRadius, startAngle, endAngle, segments, color);
DrawRingLines(center, innerRadius, outerRadius, startAngle, endAngle, segments,
color);

// Gradiente
DrawCircleGradient(centerX, centerY, radius, innerColor, outerColor);
```

Parâmetros:

- `startAngle, endAngle`: Em graus (0° = direita, 90° = cima)
- `segments`: Qualidade do círculo (mais = mais suave)

Quando usar:

- **Circle:** Projéteis, moedas, botões circulares
- **Sector:** Indicadores de cooldown, gráfico de pizza
- **Ring:** Áreas de efeito, minimapas, loading rings
- **Gradient:** Efeitos de luz, aura

Exemplo de cooldown indicator:

```
float cooldownPercent = currentCooldown / maxCooldown;
float angle = 360.0f * cooldownPercent;

// Background
DrawCircle(400, 300, 50, GRAY);

// Cooldown restante
DrawCircleSector(Vector2{400, 300}, 50, -90, -90 + angle, 36, GREEN);

// Border
DrawCircleLines(400, 300, 50, BLACK);
```

Exemplo de área de efeito (AOE):

```
float pulseRadius = baseRadius + sin(GetTime() * 5) * 10;
DrawCircleGradient(posX, posY, pulseRadius,
                   Fade(YELLOW, 0.5f),
                   Fade(YELLOW, 0.0f));
```

Elipses

```
DrawEllipse(centerX, centerY, radiusH, radiusV, color);
DrawEllipseLines(centerX, centerY, radiusH, radiusV, color);
```

Quando usar:

- Sombras sob personagens
- Portais
- Formas orgânicas

Linhas

```
// Linha simples
DrawLine(startX, startY, endX, endY, color);
DrawLineV(Vector2 start, Vector2 end, color);

// Linha espessa
DrawLineEx(Vector2 start, Vector2 end, thickness, color);

// Bezier curves
DrawLineBezier(start, end, thickness, color);
DrawLineBezierQuad(start, end, control, thickness, color);
DrawLineBezierCubic(start, end, startControl, endControl, thickness, color);
```

```
// Linha contínua
DrawLineStrip(Vector2* points, numPoints, color);
```

Exemplo de trajetória:

```
std::vector<Vector2> trajectoryPoints;
// Calcular pontos...
DrawLineStrip(trajectoryPoints.data(), trajectoryPoints.size(), YELLOW);
```

Exemplo de laser com Bezier:

```
Vector2 start = {100, 100};
Vector2 end = {700, 400};
Vector2 control = Vector2{
    (start.x + end.x) / 2 + sin(GetTime() * 5) * 50,
    (start.y + end.y) / 2
};
DrawLineBezierQuad(start, end, control, 3, RED);
```

Polígonos

```
// Polígono regular
DrawPoly(Vector2 center, sides, radius, rotation, color);
DrawPolyLines(center, sides, radius, rotation, color);
DrawPolyLinesEx(center, sides, radius, rotation, lineThick, color);

// Polígono customizado
DrawTriangle(v1, v2, v3, color);
DrawTriangleLines(v1, v2, v3, color);

// Array de pontos
DrawTriangleFan(Vector2* points, numPoints, color);
DrawTriangleStrip(points, numPoints, color);
```

Exemplo de hexágono:

```
DrawPoly(Vector2{400, 300}, 6, 50, 0, BLUE);
DrawPolyLinesEx(Vector2{400, 300}, 6, 50, 0, 3, BLACK);
```

Exemplo de estrela:

```
void DrawStar(Vector2 center, int points, float outerRadius, float innerRadius,
Color color) {
```

```
    std::vector<Vector2> vertices;
    float angleStep = 360.0f / (points * 2);

    for (int i = 0; i < points * 2; i++) {
        float angle = i * angleStep * DEG2RAD;
        float radius = (i % 2 == 0) ? outerRadius : innerRadius;
        vertices.push_back({
            center.x + cos(angle) * radius,
            center.y + sin(angle) * radius
        });
    }

    DrawTriangleFan(vertices.data(), vertices.size(), color);
}

// Uso
DrawStar(Vector2{400, 300}, 5, 50, 20, YELLOW);
```

Pixel

```
DrawPixel(x, y, color);
DrawPixelV(Vector2{x, y}, color);
```

Quando usar:

- Particle systems simples
- Minimapas pixelados
- Efeitos de baixa resolução

Splines

```
DrawSplineLinear(Vector2* points, numPoints, thickness, color);
DrawSplineBasis(points, numPoints, thickness, color);
DrawSplineCatmullRom(points, numPoints, thickness, color);
DrawSplineBezierQuadratic(points, numPoints, thickness, color);
DrawSplineBezierCubic(points, numPoints, thickness, color);
```

Quando usar:

- Caminhos suaves para NPCs
- Trajetórias de projéteis
- Caminhos de câmera
- Animações de UI

Input System

Teclado

```
// Estados de tecla
IsKeyPressed(KEY_SPACE)      // true apenas no frame que pressionou
IsKeyDown(KEY_SPACE)         // true enquanto segura
IsKeyReleased(KEY_SPACE)     // true apenas no frame que soltou
IsKeyUp(KEY_SPACE)           // true quando não está pressionado

// Última tecla
int key = GetKeyPressed();   // Código da tecla (0 se nenhuma)
int chr = GetCharPressed();  // Caractere Unicode (para input de texto)

// Repeat rate
SetExitKey(KEY_NULL);        // Desabilita ESC para fechar
```

Todas as teclas disponíveis:

```
// Letras
KEY_A ... KEY_Z

// Números (teclado principal)
KEY_ZERO ... KEY_NINE

// Numpad
KEY_KP_0 ... KEY_KP_9
KEY_KP_DECIMAL, KEY_KP_DIVIDE, KEY_KP_MULTIPLY
KEY_KP_SUBTRACT, KEY_KP_ADD, KEY_KP_ENTER, KEY_KP_EQUAL

// Setas
KEY_RIGHT, KEY_LEFT, KEY_DOWN, KEY_UP

// Função
KEY_F1 ... KEY_F12

// Modificadores
KEY_LEFT_SHIFT, KEY_RIGHT_SHIFT
KEY_LEFT_CONTROL, KEY_RIGHT_CONTROL
KEY_LEFT_ALT, KEY_RIGHT_ALT
KEY_LEFT_SUPER, KEY_RIGHT_SUPER  // Windows/Command key

// Especiais
KEY_SPACE, KEY_ESCAPE, KEY_ENTER, KEY_TAB, KEY_BACKSPACE
KEY_INSERT, KEY_DELETE, KEY_HOME, KEY_END, KEY_PAGE_UP, KEY_PAGE_DOWN
KEY_CAPS_LOCK, KEY_SCROLL_LOCK, KEY_NUM_LOCK, KEY_PRINT_SCREEN, KEY_PAUSE

// Pontuação
KEY_APOSTROPHE, KEY_COMMA
```