Raylib C++ - Referência Avançada (Continuação)

CONFIGURAÇÃO DE JANELA (FLAGS)

```
SetConfigFlags(flags)                       // Chamar ANTES de InitWindow

// Flags disponíveis:
FLAG_VSYNC_HINT                             // Ativa V-Sync
FLAG_FULLSCREEN_MODE                        // Inicia em tela cheia
FLAG_WINDOW_RESIZABLE                       // Permite redimensionamento
FLAG_WINDOW_UNDECORATED                     // Remove barra de título
FLAG_WINDOW_HIDDEN                          // Janela oculta
FLAG_WINDOW_MINIMIZED                       // Inicia minimizada
FLAG_WINDOW_MAXIMIZED                       // Inicia maximizada
FLAG_WINDOW_ALWAYS_RUN                      // Continua rodando em background
FLAG_MSAA_4X_HINT                           // Anti-aliasing 4x

// Uso: SetConfigFlags(FLAG_VSYNC_HINT | FLAG_WINDOW_RESIZABLE);
```

CONTROLE DE JANELA AVANÇADO

```
IsWindowReady()                             // Verifica se janela foi criada
IsWindowFocused()                           // true se janela tem foco
IsWindowMinimized()                         // true se minimizada
IsWindowMaximized()                         // true se maximizada
IsWindowHidden()                            // true se oculta
SetWindowMinSize(width, height)             // Define tamanho mínimo
SetWindowMaxSize(width, height)             // Define tamanho máximo
SetWindowPosition(x, y)                     // Move janela na tela
SetWindowMonitor(monitor)                   // Move para monitor específico
SetWindowState(flags)                       // Define estado da janela
ClearWindowState(flags)                     // Remove estado
GetWindowPosition()                         // Posição da janela
GetMonitorCount()                           // Número de monitores
GetCurrentMonitor()                         // Monitor atual
GetMonitorWidth(monitor)                    // Largura do monitor
GetMonitorHeight(monitor)                   // Altura do monitor
```

CURSOR DO MOUSE

```
ShowCursor()                                // Mostra cursor
HideCursor()                                // Oculta cursor
IsCursorHidden()                            // Verifica se oculto
EnableCursor()                              // Habilita cursor
DisableCursor()                             // Desabilita cursor (FPS mode)
```

```
IsCursorOnScreen()                          // Verifica se cursor está na janela
SetMousePosition(x, y)                      // Define posição do cursor
SetMouseOffset(x, y)                        // Offset do cursor
SetMouseScale(x, y)                         // Escala do cursor
```

CLIPBOARD

```
SetClipboardText(text)                      // Copia texto
GetClipboardText()                          // Cola texto
```

IMAGEM (IMAGE) - Manipulação em CPU

```
LoadImage(fileName)                         // Carrega imagem na RAM (não GPU)
UnloadImage(image)
LoadImageFromTexture(texture)               // Copia textura GPU → RAM
ExportImage(image, fileName)                // Salva imagem
ImageCopy(image)                            // Cria cópia
ImageFromImage(image, rec)                  // Extrai região

// Manipulação (modifica Image original)
ImageResize(image, width, height)
ImageResizeNN(image, width, height)         // Nearest Neighbor
ImageCrop(image, crop)
ImageAlphaCrop(image, threshold)
ImageColorTint(image, color)
ImageColorInvert(image)
ImageColorGrayscale(image)
ImageColorContrast(image, contrast)
ImageColorBrightness(image, brightness)
ImageFlipVertical(image)
ImageFlipHorizontal(image)
ImageRotateCW(image)                        // 90° horário
ImageRotateCCW(image)                       // 90° anti-horário
ImageColorReplace(image, color, replace)
ImageDraw(dst, src, srcRec, dstRec, tint)   // Desenha imagem em outra

// Conversão Image ↔ Texture
LoadTextureFromImage(image)                 // RAM → GPU
UpdateTexture(texture, pixels)              // Atualiza textura com dados
```

TEXTO - GERENCIAMENTO AVANÇADO

```
TextCopy(dst, src)                          // Copia string
TextIsEqual(text1, text2)                   // Compara strings
TextLength(text)                            // Comprimento
TextFormat(text, ...)                       // Formata string (printf style)
```

```
TextSubtext(text, position, length)        // Substring
TextReplace(text, replace, by)             // Substitui texto
TextInsert(text, insert, position)         // Insere texto
TextJoin(textList, count, delimiter)       // Junta strings
TextSplit(text, delimiter, count)          // Divide string
TextAppend(text, append, position)         // Adiciona texto
TextFindIndex(text, find)                  // Encontra posição
TextToUpper(text)                          // Maiúsculas
TextToLower(text)                          // Minúsculas
TextToPascal(text)                         // PascalCase
TextToInteger(text)                        // String → int
```

SHADERS (Efeitos Gráficos)

```
LoadShader(vsFileName, fsFileName)         // Carrega vertex/fragment shader
UnloadShader(shader)
GetShaderLocation(shader, uniformName)     // Localização de uniform
SetShaderValue(shader, loc, value, type)   // Define valor uniform
SetShaderValueTexture(shader, loc, texture)
BeginShaderMode(shader) / EndShaderMode()  // Aplica shader

// Tipos de uniform: SHADER_UNIFORM_FLOAT, SHADER_UNIFORM_VEC2,
SHADER_UNIFORM_VEC3, etc.
```

RENDER BATCH - Performance

```
DrawRenderBatch(batch)                     // Renderiza batch manualmente
BeginBlendMode(mode) / EndBlendMode()      // Modo de mistura
BeginScissorMode(x, y, w, h) / EndScissorMode() // Área de corte

// Blend modes:
BLEND_ALPHA                                // Transparência padrão
BLEND_ADDITIVE                             // Adição (brilho)
BLEND_MULTIPLIED                           // Multiplicação (sombra)
BLEND_ADD_COLORS                           // Soma cores
BLEND_SUBTRACT_COLORS                      // Subtrai cores
```

COLISÃO 3D (Básico)

```
CheckCollisionBoxes(box1, box2)            // Caixas 3D (AABB)
CheckCollisionBoxSphere(box, center, radius)
CheckCollisionSpheres(c1, r1, c2, r2)
CheckCollisionRaySphere(ray, center, radius)
GetRayCollisionSphere(ray, center, radius)
GetRayCollisionBox(ray, box)
GetRayCollisionMesh(ray, mesh, transform)
```

COMPRESSÃO E ENCODING

```
CompressData(data, dataSize, compDataSize)
DecompressData(compData, compDataSize, dataSize)
EncodeDataBase64(data, dataSize, outputSize)
DecodeDataBase64(data, outputSize)
```

AUTOMAÇÃO E EVENTOS

```
StartAutomationEventRecording()            // Grava eventos de input
StopAutomationEventRecording()
PlayAutomationEvent(event)                 // Reproduz evento
SetAutomationEventList(list)
SetAutomationEventBaseFrame(frame)
```

GESTALT / UTILITÁRIOS MATEMÁTICOS

```
Clamp(value, min, max)                     // Limita valor
Lerp(start, end, amount)                   // Interpolação linear
Normalize(value, start, end)               // Normaliza [start,end] → [0,1]
Remap(value, inputStart, inputEnd, outputStart, outputEnd)

// Math extras
Vector2Rotate(v, angle)                    // Rotaciona vetor (radianos)
Vector2MoveTowards(v, target, maxDistance) // Move em direção ao alvo
Vector2Lerp(v1, v2, amount)                // Interpolação linear
Vector2Reflect(v, normal)                  // Reflexão em normal
Vector2Clamp(v, min, max)                  // Limita componentes
Vector2ClampValue(v, min, max)             // Limita magnitude
```

AUDIO AVANÇADO

```
LoadSoundFromWave(wave)                    // Carrega de Wave data
LoadWave(fileName)                         // Carrega arquivo como Wave
UnloadWave(wave)
ExportWave(wave, fileName)                 // Salva Wave
ExportWaveAsCode(wave, fileName)           // Exporta como código C

WaveCopy(wave)                             // Copia Wave
WaveCrop(wave, initSample, finalSample)
WaveFormat(wave, sampleRate, sampleSize, channels)

// Streaming de áudio customizado
```

```
LoadAudioStream(sampleRate, sampleSize, channels)
UpdateAudioStream(stream, data, frameCount)
UnloadAudioStream(stream)
IsAudioStreamProcessed(stream)
PlayAudioStream(stream)
PauseAudioStream(stream)
StopAudioStream(stream)
SetAudioStreamVolume(stream, volume)
SetAudioStreamPitch(stream, pitch)
```

FÍSICA - Chipmunk2D (se habilitado)

```
// Raylib não inclui física por padrão
// Use bibliotecas externas: Box2D, Chipmunk2D
// Ou implemente manualmente:

// Gravidade simples
velocity.y += gravity * GetFrameTime();
position.y += velocity.y * GetFrameTime();

// Colisão e resposta básica
if (CheckCollisionRecs(player, wall)) {
    // Recuar posição
    // Anular velocidade
}
```

GESTURES (Touch/Mobile)

```
SetGesturesEnabled(flags)               // Habilita gestos
IsGestureDetected(gesture)              // Detecta gesto
GetGestureDetected()                    // Último gesto
GetTouchPointCount()                    // Número de toques
GetTouchPosition(index)                 // Posição do toque
GetGestureDragVector()                  // Vetor de arrasto
GetGestureDragAngle()                   // Ângulo de arrasto
GetGesturePinchVector()                 // Vetor de pinça
GetGesturePinchAngle()                  // Ângulo de pinça

// Gestures:
GESTURE_TAP, GESTURE_DOUBLETAP
GESTURE_HOLD, GESTURE_DRAG
GESTURE_SWIPE_RIGHT, GESTURE_SWIPE_LEFT
GESTURE_SWIPE_UP, GESTURE_SWIPE_DOWN
GESTURE_PINCH_IN, GESTURE_PINCH_OUT
```

MEMORY MANAGEMENT

```
MemAlloc(size)                          // Aloca memória
MemRealloc(ptr, size)                   // Realoca
MemFree(ptr)                            // Libera

// Raylib gerencia automaticamente memória interna
// Use apenas para dados customizados
```

LOGGING E DEBUG

```
SetTraceLogLevel(logLevel)              // Define nível de log
TraceLog(logLevel, text, ...)           // Log customizado

// Níveis:
LOG_ALL, LOG_TRACE, LOG_DEBUG
LOG_INFO, LOG_WARNING, LOG_ERROR
LOG_FATAL, LOG_NONE

TakeScreenshot(fileName)                // Captura tela
```

NETWORK (rlgl - baixo nível)

```
// Raylib não inclui networking nativo
// Use bibliotecas externas:
// - enet (UDP confiável)
// - SDL_net
// - ASIO (C++)
// - WebSockets para web
```

PERFORMANCE - BOAS PRÁTICAS

```
// 1. Minimize Draw Calls
// Agrupe objetos com mesma textura
// Use spritesheets ao invés de texturas individuais

// 2. Evite Load/Unload no loop
// Carregue todos recursos no início
Texture2D tex = LoadTexture("sprite.png");  // ✓ Fora do loop
while (!WindowShouldClose()) {
    // LoadTexture aqui = ✗ ERRADO
}

// 3. Use RenderTexture para efeitos complexos
// Renderize uma vez, reutilize múltiplas vezes

// 4. Batch rendering automático
```

```
// DrawTexture com mesma textura = 1 draw call
// Alternância de texturas = múltiplos draw calls

// 5. Culling manual
if (IsInScreen(object)) {
    DrawTexture(object.texture, ...);  // Desenha apenas visíveis
}

// 6. Object pooling
// Reutilize objetos ao invés de criar/destruir
```

PADRÕES DE GAME LOOP

```
// Fixed timestep (física consistente)
const float FIXED_DT = 1.0f/60.0f;
float accumulator = 0.0f;

while (!WindowShouldClose()) {
    float frameTime = GetFrameTime();
    accumulator += frameTime;

    while (accumulator >= FIXED_DT) {
        UpdatePhysics(FIXED_DT);  // Física em passos fixos
        accumulator -= FIXED_DT;
    }

    UpdateGame(frameTime);  // Lógica com deltaTime real

    BeginDrawing();
        Render();
    EndDrawing();
}
```

STATE MACHINE (Gerenciamento de Estados)

```
enum GameState { MENU, PLAYING, PAUSED, GAME_OVER };
GameState currentState = MENU;

while (!WindowShouldClose()) {
    switch (currentState) {
        case MENU:
            UpdateMenu();
            if (startPressed) currentState = PLAYING;
            break;
        case PLAYING:
            UpdateGame();
            if (pausePressed) currentState = PAUSED;
            break;
        case PAUSED:
```

```
            if (resumePressed) currentState = PLAYING;
            break;
        case GAME_OVER:
            if (restartPressed) currentState = PLAYING;
            break;
    }

    BeginDrawing();
        DrawCurrentState();
    EndDrawing();
}
```

ANIMAÇÃO DE SPRITES

```
typedef struct {
    Texture2D texture;
    int frameWidth, frameHeight;
    int currentFrame, totalFrames;
    float frameTime, frameCounter;
} Animation;

void UpdateAnimation(Animation *anim, float dt) {
    anim->frameCounter += dt;
    if (anim->frameCounter >= anim->frameTime) {
        anim->currentFrame++;
        if (anim->currentFrame >= anim->totalFrames)
            anim->currentFrame = 0;
        anim->frameCounter = 0;
    }
}

void DrawAnimation(Animation anim, Vector2 pos) {
    Rectangle source = {
        anim.currentFrame * anim.frameWidth, 0,
        anim.frameWidth, anim.frameHeight
    };
    DrawTextureRec(anim.texture, source, pos, WHITE);
}
```

PARTICLE SYSTEM (Básico)

```
typedef struct {
    Vector2 position;
    Vector2 velocity;
    Color color;
    float lifetime;
    bool active;
} Particle;
```

```
Particle particles[MAX_PARTICLES];

void EmitParticle(Vector2 pos) {
    for (int i = 0; i < MAX_PARTICLES; i++) {
        if (!particles[i].active) {
            particles[i].position = pos;
            particles[i].velocity = {RandomFloat(-100,100),
RandomFloat(-100,100)};
            particles[i].
```