

DCC192

2025/1



# Desenvolvimento de Jogos Digitais

A2: Simple DirectMedia Layer (SDL)

Prof. Lucas N. Ferreira

# Plano de aula



- ▶ Configurando CLion + SDL
- ▶ Introdução a SDL
  - ▶ SDL vs. Engines
  - ▶ Subistemas SDL
  - ▶ Criando Janelas e Renderizadores
  - ▶ Loop Principal
  - ▶ Primitivas geométricas
  - ▶ Eventos de entrada

# SDL – Simple DirectMedia Layer



**Simple DirectMedia Layer (SDL)** é uma biblioteca de desenvolvimento de jogos projetada para fornecer acesso de baixo nível a áudio, teclado, mouse, joystick, gráficos, ...

- ▶ Multiplataforma:
  - ▶ Windows, Mac OS X, Linux, iOS, Android, ..
- ▶ Utilizada em muitos jogos e engines:
  - ▶ Valve's Source Engine
    - ▶ Half-Life 2, Portal, Counter-Strike, ...
  - ▶ Faster Than Light, Darf Fortress , VVVVVV, ...



# SDL Vs. Game Engines



SDL	Game Engines (Unity, Unreal, Godot)
Biblioteca de baixo nível	Frameworks completos de alto nível
Controle direto sobre o hardware	Abstrações prontas (física, renderização avançada)
Desempenho otimizado	Facilidade e rapidez de desenvolvimento
Curva de aprendizado média	Interface visual e ferramentas integradas
Menos recursos prontos	Ecossistema de assets e plugins

## Por que usar SDL?

- ▶ **Compreensão mais profunda:** entender os fundamentos de desenvolvimento de jogos
- ▶ **Flexibilidade:** criar estruturas personalizadas sem restrições de framework
- ▶ **Performance:** controlar o uso de recursos com mais precisão

# Subsistemas SDL2



A SDL2 é organizada em vários subsistemas, que oferecem funções específicas:

- ▶ `Video` – Desenho acelerado por hardware gráfico
- ▶ `Audio` – Reprodução e captura de sons
- ▶ `Timer` – Gerenciamento de tempo com alta precisão
- ▶ `Events` – Processar eventos de entrada
- ▶ `File I/O` – Manipulação de arquivos independentes de plataforma
- ▶ `Threading` – Gerenciamento de threads

**Extensões da SDL2:** `SDL_image`, `SDL_mixer`, `SDL_ttf`, `SDL_net`, `SDL_gfx`

# 1. Inicializando subsistemas SDL2



A primeira etapa de todo programa em SDL2 é inicializar os subsistemas desejados.

```
// A biblioteca SDL.h contém todos os subsistemas básicos da SDL.  
// Para incluir extensões, é necessário incluir cada extensão manualmente (veremos isso mais a frente)  
  
#include <SDL.h>  
  
int main() {  
    // A função SDL_Init(Uint32 flags) é responsável por inicializar os subsistemas desejados  
  
    SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER | SDL_INIT_JOYSTICK | ...);  
  
    // Note que para inicializar múltiplos sistemas você precisa combinar as flags usando  
    // o operador OR binário (|).  
}
```

Os subsistemas Events, File I/O e Threading são inicializados automaticamente!

## 2. Criando de Janelas



A segunda etapa é criar uma janela para mostrar os gráficos renderizados.

```
#include <SDL.h>

int main() {

    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window* window = SDL_CreateWindow(
        "Meu Jogo",           // título
        SDL_WINDOWPOS_CENTERED, // posição X
        SDL_WINDOWPOS_CENTERED, // posição Y
        800,                  // largura
        600,                  // altura
        SDL_WINDOW_SHOWN      // flags
    );

    // Flags comuns
    // SDL_WINDOW_FULLSCREEN - tela cheia
    // SDL_WINDOW_RESIZABLE - permite redimensionar
    // SDL_WINDOW_BORDERLESS - sem bordas
    // SDL_WINDOW_OPENGL - suporte a OpenGL
}
```



# 3. Inicializando Renderizador



A terceira etapa é criar um **buffer** para **renderizar** gráficos: primitivas, texturas, ...

```
#include <SDL.h>

int main() {

    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window* window = SDL_CreateWindow("Meu Jogo", 100, 100, 800, 600, SDL_WINDOW_SHOWN);

    // Criar renderizador
    SDL_Renderer* renderer = SDL_CreateRenderer(
        window,                      // janela associada
        -1,                          // índice do driver (-1 = primeiro compatível)
        SDL_RENDERER_ACCELERATED |   // usa aceleração de hardware
        SDL_RENDERER_PRESENTVSYNC    // sincroniza com taxa de atualização
    );

    // Configurar cor de desenho (RGBA)
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // preto

    // Limpar tela
    SDL_RenderClear(renderer);

    // Apresentar renderização
    SDL_RenderPresent(renderer);

}
```



## 4. Loop principal



A quarta etapa é o **loop principal** para manter a janela aberta enquanto o usuário não a fecha

```
#include <SDL.h>

int main() {

    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window* window = SDL_CreateWindow("Meu Jogo", 100, 100, 800, 600, SDL_WINDOW_SHOWN);
    SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);

    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);

    // Loop principal
    bool running = true;
    while (running) {

        // Processar todos os eventos disponíveis
        SDL_Event event;
        while (SDL_PollEvent(&event)) {
            if (event.type == SDL_QUIT) // Quando o usuário clicar no ícone (x) para fechar a janela
                running = false;      // interrompa o loop
        }
    }
}
```

## 5. Limpando SDL2



A quinta e última etapa é **destruir** os objetos e finalizar a SDL2

```
#include <SDL.h>

int main() {

    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window* window = SDL_CreateWindow("Meu Jogo", 100, 100, 800, 600, SDL_WINDOW_SHOWN);
    SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);

    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    SDL_RenderClear(renderer);
    SDL_RenderPresent(renderer);

    bool running = true;
    while (running) {
        SDL_Event event;
        while (SDL_PollEvent(&event))
            if (event.type == SDL_QUIT)
                running = false;
    }

    SDL_DestroyRenderer(renderer); // Destrói renderizador
    SDL_DestroyWindow(window);     // Destrói janela
    SDL_Quit();                   // Finaliza subsistemas inicializados
}
```

# Desenhando Primitivas Geométricas



Nos primeiros trabalhos práticos, iremos usar primitivas geométricas (ponto, linha, retângulo,...) para representar objetos do jogo visualmente

```
// Ponto
SDL_RenderDrawPoint(renderer, x, y);

// Linha
SDL_RenderDrawLine(renderer, x1, y1, x2, y2);

// Retângulo (contorno)
SDL_Rect rect = { x, y, width, height };
SDL_RenderDrawRect(renderer, &rect);

// Retângulo (preenchido)
SDL_RenderFillRect(renderer, &rect);

// Múltiplos pontos/linhas
SDL_RenderDrawPoints(renderer, pontos, numPontos);
SDL_RenderDrawLines(renderer, pontos, numPontos);
```

# Double Buffering



Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado

# Double Buffering



Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado

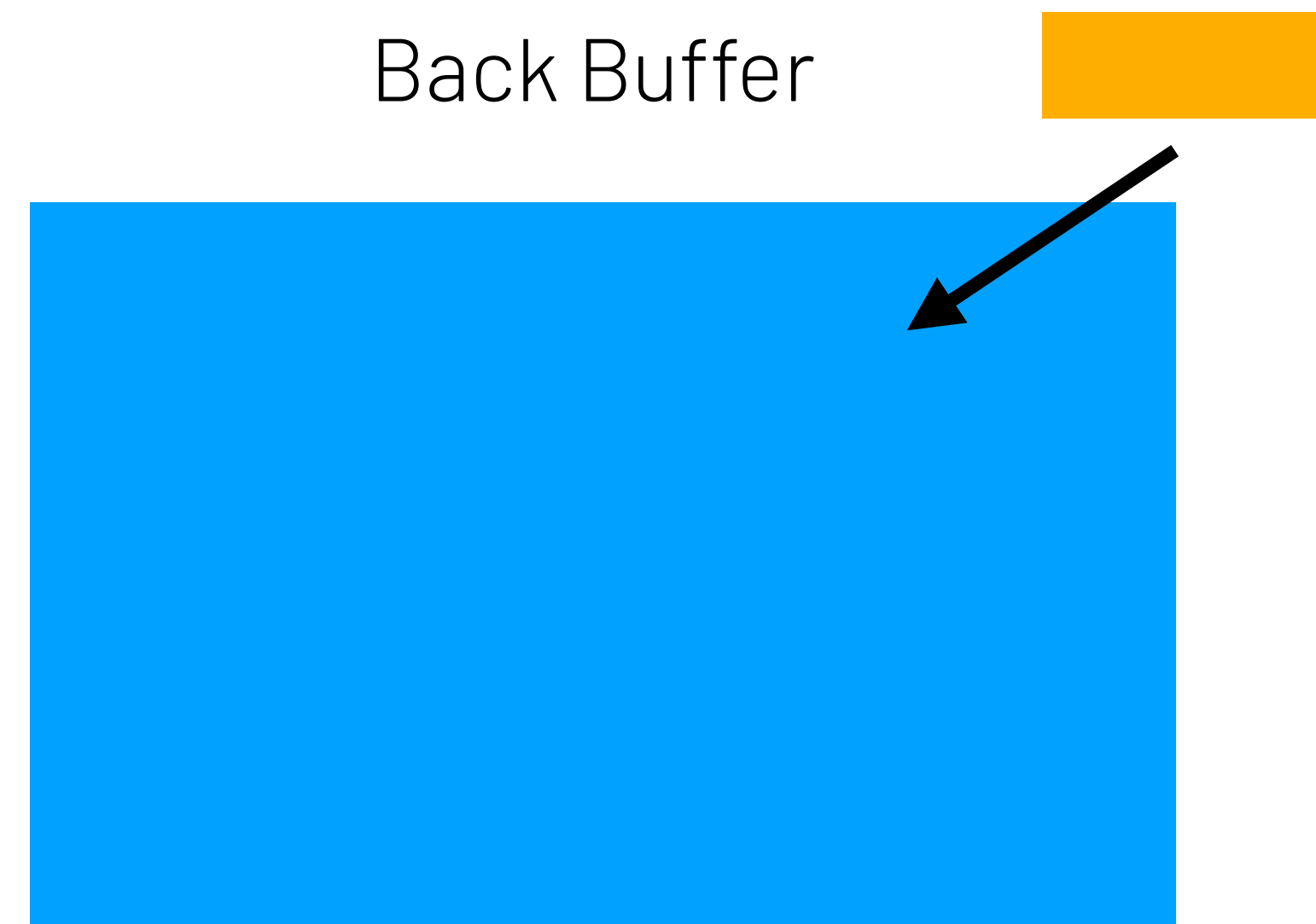
Back Buffer



# Double Buffering



Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado



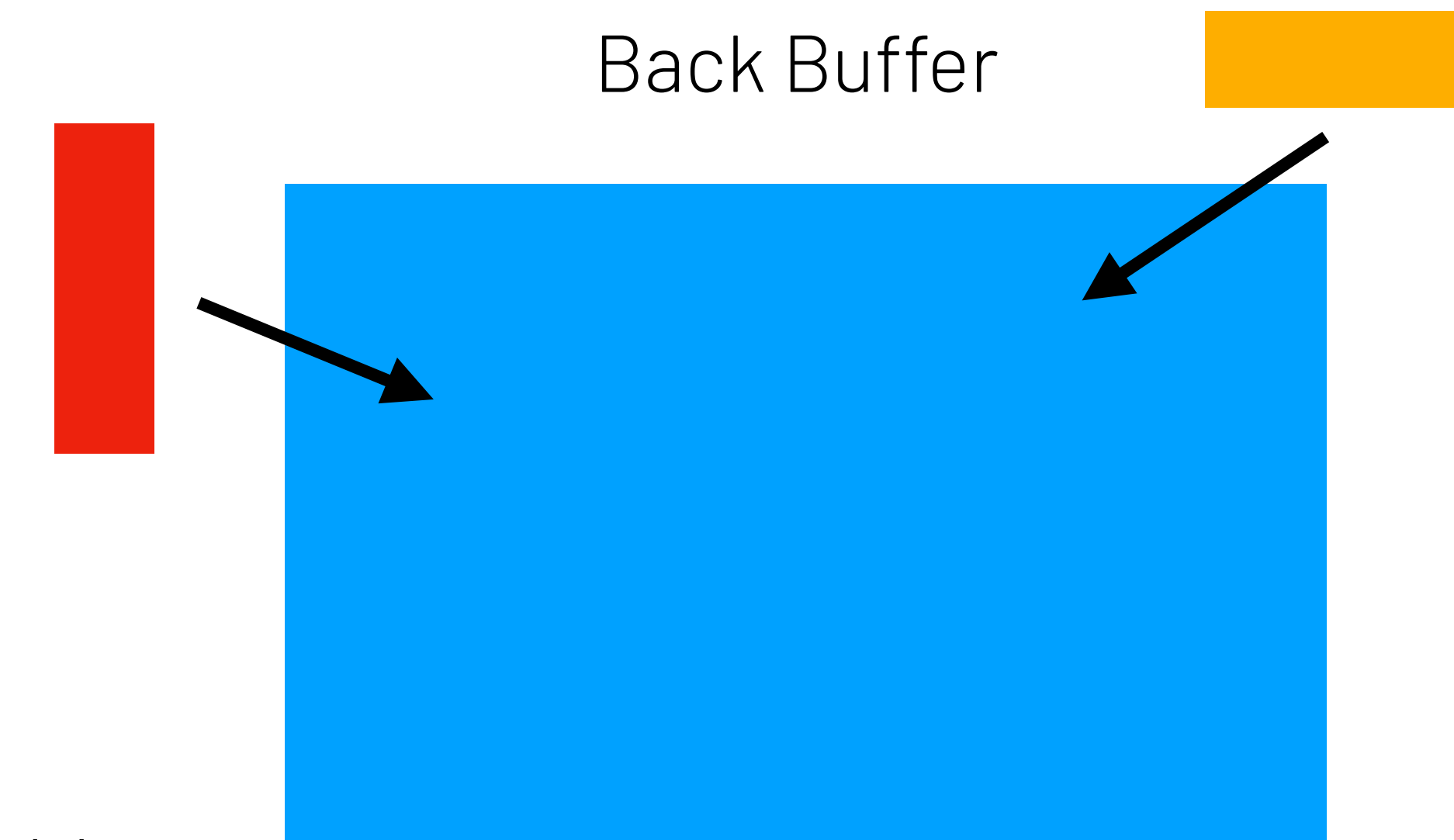
(1)

Operações Draw  
SDL\_RenderFillRect

# Double Buffering



Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado



(1)

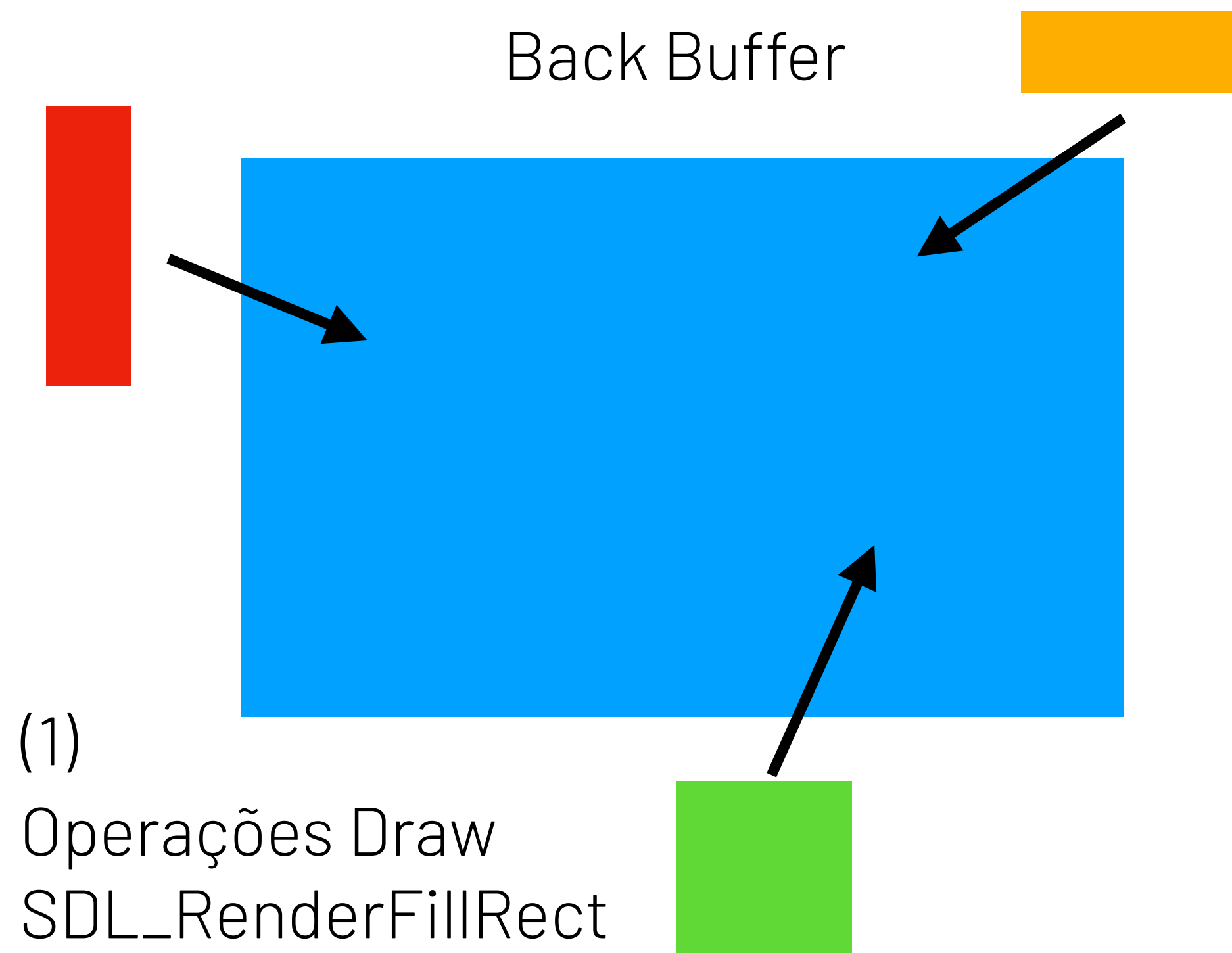
Operações Draw  
SDL\_RenderFillRect



# Double Buffering



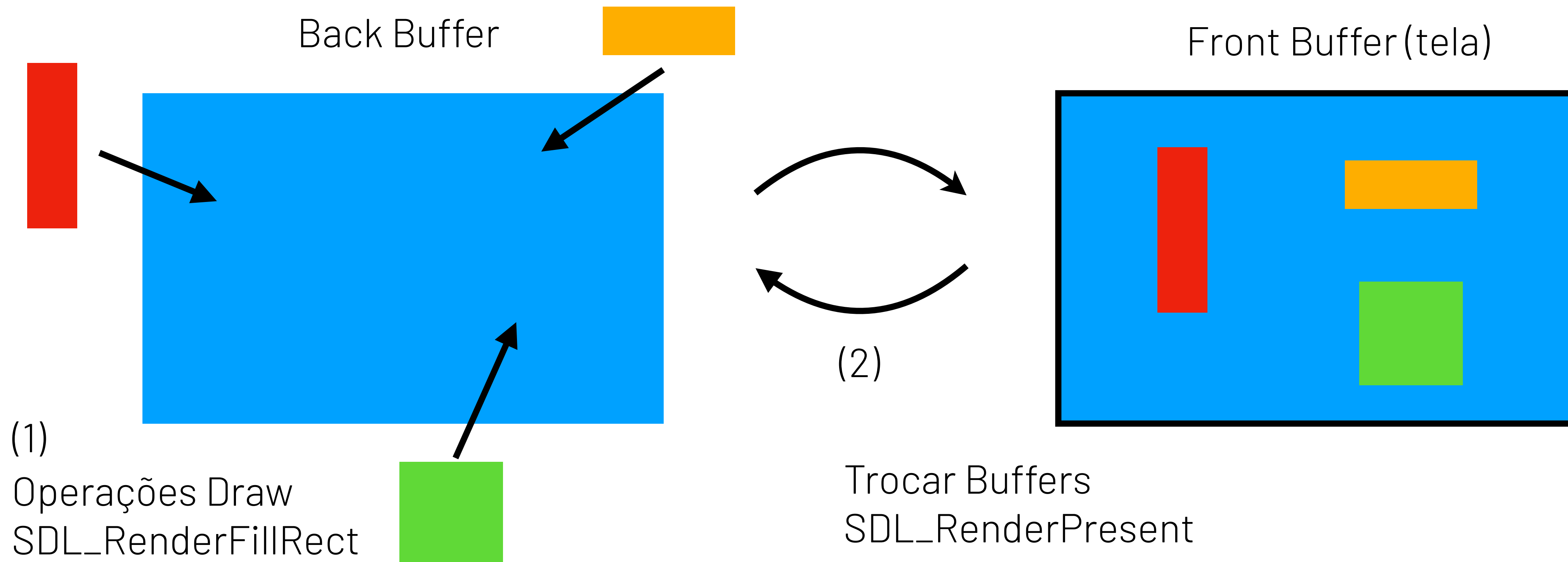
Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado



# Double Buffering



Jogos Digitais geralmente são renderizados usando **Double Buffering**, onde gráficos são desenhados em um back buffer, que é trocado com o front buffer quando o quadro inteiro foi desenhado



# Processando Eventos de Entrada



```
SDL_Event event;
while (SDL_PollEvent(&event)) {
    switch (event.type) {
        case SDL_KEYDOWN: // Tecla pressionada
            if (event.key.keysym.sym == SDLK_ESCAPE)
                running = 0;

        case SDL_KEYUP: // Tecla liberada
            break;

        case SDL_MOUSEMOTION: // Movimento do mouse
            int mouseX = event.motion.x;
            int mouseY = event.motion.y;
            break;

        case SDL_MOUSEBUTTONDOWN: // Botão do mouse pressionado break;
            if (event.button.button == SDL_BUTTON_LEFT)
                running = 0;
            break;
    }
}
```

# Estado do Teclado e Mouse



```
// Obter estado de todas as teclas
const Uint8* keyState = SDL_GetKeyboardState(NULL);

// Verificar teclas específicas
if (keyState[SDL_SCANCODE_RIGHT]) {
    // Seta direita pressionada
    playerX += 5; }

if (keyState[SDL_SCANCODE_LEFT]) {
    // Seta esquerda pressionada
    playerX -= 5;
}

// Obter estado do mouse
int mouseX, mouseY;
Uint32 mouseButtons = SDL_GetMouseState(&mouseX, &mouseY);

// Verificar botões do mouse
if (mouseButtons & SDL_BUTTON(SDL_BUTTON_LEFT)) {
    // Botão esquerdo pressionado
}
```

# Próxima aula



## A3: Game Loop

- ▶ Técnicas de controle de tempo em jogos
- ▶ FPS fixo vs. FPS dinâmico