bitEngine - Criando Janelas Multiplataformas

Canoi Gomes

23 de Abril de 2023

Conteúdo

1 Introdução **3**

2 Renderizando com OpenGL **5**

3 Considerações **8**

1 Introdução

Dessa vez decidir começar um projeto sobre uma parte que venho querendo aprender a um tempo, que seria como criar o contexto básico pra um jogo (janela, input, gráficos e áudios) utilizando somente bibliotecas do próprio sistema, em resumo, eu queria entender mais como bibliotecas como SDL2 e GLFW funcionam por baixo dos panos.

Nisso (como sempre faço na minha vida) decidi criar um projeto pra focar nos estudos dessa parada, bite, a ideia é:

- Conseguir criar uma janela pelo menos em desktop (Windows, Linux e Mac) e web (Emscripten)
- Criar um render básico utilizando OpenGL (OpenGLES2 com Emscripten), e carregar somente algumas funções específicas do modern OpenGL que vão ser necessárias (criar shaders, framebuffers, …).
- Tocar pelo menos 1 áudio (a ideia é fazer um mixer, mas vamo vê né).
- Filesystem básico (ler e escrever arquivos, listar diretórios, …).

A ideia é ter algo como:

```
#include <bite.h>
#if defined(__EMSCRIPTEN__)
    #include <emscripten.h>
#endif

void main_loop(void* arg) {
    be_Context* ctx = (be_Context*)arg;
    bite_poll_events(ctx);
    // render suff
    bite_swap(ctx);
```

```
}

int main(int argc, char** argv) {
    be_Config conf = bite_init_config("Hello Window", 640, 380);
    be_Context* ctx = bite_create(&conf);
#if defined(__EMSCRIPTEN__)
    emscripten_set_main_loop_arg(main_loop, ctx, 0, 1);
#else
    while(!bite_should_close(ctx)) main_loop(ctx);
#endif
    bite_destroy(ctx);
    return 0;
}
```

Onde por trás vai ser criado o contexto específico pra cada plataforma:

```
#if defined(_WIN32)
  #include <windows.h>
#elif defined(__EMSCRIPTEN__)
  #include <emscripten.h>
  #include <emscripten/html5.h>
#else
  #include <X11/Xlib.h>
  #include <GL/glx.h>
#endif

be_Context* bite_create(const be_Config* conf) {
#if defined(_WIN32)
  // Win32 Window and WGL context creation
#elif defined(__EMSCRIPTEN__)
  // Emscripten context creation
#else
  // Linux Window and GLX context creation
  // other systems .....
```

```
#endif
}
```

Se estiverem interessados em como funciona a criação da janela pra cada plataforma, esse artigo dá uma pincelada legal no assunto: `https://zserge.com/posts/fenster/`

2 Renderizando com OpenGL

Na parte de renderização vai OpenGL mesmo, que como eu disse, funciona bem pro meu escopo (Desktop e Web). Pra isso preciso carregar um contexto OpenGL que suporte extensões, já que as libs padrão de cada plataforma (GLX no Windows e WGL no Windows) só nos dão um contexto com uma versão antiga (versão 1.4 se não me engano), e cada plataforma tem sua maneira de carregar um contexto mais moderno. No Windows, por exemplo, é necessário criar uma "dummy window" com um contexto antigo, somente pra ser capaz de carregar a função responsável por criar o contexto mais novo, depois disso ela é simplesmente deletada, no Linux não é necessário (outras plataformas provavelmente tem suas especificidades também, mas não cheguei lá ainda).

Exemplo no Windows

Tutorial para Linux

Tendo o "contexto moderno" carregado, ainda é preciso carregar as funções que eu vou utilizar, e pra isso existe a função **GetProcAddress** de cada lib (**glXGetProcAddress** no Linux ou **wglGetProcAddress** no Windows).

```
typedef GLuint glCreateProgramProc(void);

static glCreateProgramProc* glCreateProgram = 0;

#if defined(_WIN32)
```

```
6       #define biteGetProcAddress wglGetProcAddress
7   #elif defined(__linux__)
8       #define biteGetProcAdress glXGetProcAddress
9   #else
10      #define biteGetProcAddress(x) ((void)(x))
11  #endif
12
13  int init_opengl_procs(void) {
14      glCreateProgram = (glCreateProgramProc*)biteGetProcAdress("
        glCreateProgram");
15      return 0;
16  }
```

Vale a pena dar uma olhada em outros loaders como o glad e o GLEW.

Tem uma lib minha que faz algo parecido com o que eu quero fazer aqui, tea, que é basicamente carregar somente o mínimo de funções necessárias e criar abstrações em cima delas.

Pra ter o básico pra suportar shaders, por exemplo, seriam necessárias:

- glCreateShader
- glShaderSource
- glCompileShader
- glGetShaderiv
- glGetShaderInfoLog
- glDeleteShader

- glCreateProgram

- glAttachShader
- glLinkProgram
- glGetProgramiv
- glGetProgramInfoLog
- glDeleteProgram
- glUseProgram

Sem expor isso pro usuário, mas sim abstraindo o processo em outras funções:

```
be_Shader* bite_create_shader(const char* vert_src, const char*
    frag_src) {
    be_Shader* shader = NULL;
    GLuint program;
    GLuint vert, frag;

    vert = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vert_src);
    // ....

    program = glCreateProgram();
    glAttachShader(program, vert);
    glAttachShader(program, frag);
    // ...

    shader->handle = program;
    glDeleteShader(vert);
    glDeleteShader(frag);

    return shader;
```

```
20 }
```

E é isto.

3 Considerações

Fora a renderização também vão ter outros pontos pra se lidar, como por exemplo:

- Eventos (janela movendo/redimensionando, tecla pressionada, …), que na real é bem tranquilo, o mais chatinho é lidar com a questão multiplataforma da parada mesmo, já que no caso de teclas pressionadas os KeyCodes são diferentes, por exemplo, então tu vai ter que criar os seus próprios e filtrar por plataforma pra retornar pro usuário o certo.
- Áudio, que sinceramente ainda é um mistério pra mim.

Eu pretendo (ou pelo menos espero conseguir) postar devlogs a medida que for aprendendo sobre os assuntos.

E outra coisa que to pensando em fazer é separar os backends em arquivos .c diferentes, queria muito ter um único **.h** e **.c** pra facilitar portabilidade, mas é horrível de mexer com tanto **#ifdef** .