

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 (OpenGL ES2 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:

```
1 #include <bite.h>
2 #if defined(__EMSCRIPTEN__)
3     #include <emscripten.h>
4 #endif
5
6 void main_loop(void* arg) {
7     be_Context* ctx = (be_Context*)arg;
8     bite_poll_events(ctx);
9     // render suff
10    bite_swap(ctx);
```

```

11 }
12
13 int main(int argc, char** argv) {
14     be_Config conf = bite_init_config("Hello Window", 640, 380);
15     be_Context* ctx = bite_create(&conf);
16     #if defined(__EMSCRIPTEN__)
17         emscripten_set_main_loop_arg(main_loop, ctx, 0, 1);
18     #else
19         while(!bite_should_close(ctx)) main_loop(ctx);
20     #endif
21     bite_destroy(ctx);
22     return 0;
23 }

```

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

```

1 #if defined(_WIN32)
2     #include <windows.h>
3 #elif defined(__EMSCRIPTEN__)
4     #include <emscripten.h>
5     #include <emscripten/html5.h>
6 #else
7     #include <X11/Xlib.h>
8     #include <GL/gl.h>
9 #endif
10
11 be_Context* bite_create(const be_Config* conf) {
12     #if defined(_WIN32)
13         // Win32 Window and WGL context creation
14     #elif defined(__EMSCRIPTEN__)
15         // Emscripten context creation
16     #else
17         // Linux Window and GLX context creation
18         // other systems .....

```

```
19 #endif
20 }
```

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 Linux 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).

```
1 typedef GLuint glCreateProgramProc(void);
2
3 static glCreateProgramProc* glCreateProgram = 0;
4
5 #if defined(_WIN32)
```

```

6     #define biteGetProcAddress wglGetProcAddress
7 #elif defined(__linux__)
8     #define biteGetProcAddress glXGetProcAddress
9 #else
10    #define biteGetProcAddress(x) ((void)(x))
11 #endif
12
13 int init_opengl_procs(void) {
14     glCreateProgram = (glCreateProgramProc*)biteGetProcAddress("
    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:

```
1 be_Shader* bite_create_shader(const char* vert_src, const char*
   frag_src) {
2     be_Shader* shader = NULL;
3     GLuint program;
4     GLuint vert, frag;
5
6     vert = glCreateShader(GL_VERTEX_SHADER);
7     glShaderSource(vert_src);
8     // ....
9
10    program = glCreateProgram();
11    glAttachShader(program, vert);
12    glAttachShader(program, frag);
13    // ...
14
15    shader->handle = program;
16    glDeleteShader(vert);
17    glDeleteShader(frag);
18
19    return shader;
```

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`.