Selene Engine

Canoi Gomes

19 de Outubro de 2023

Conteúdo

1 Introdução

Nas ultimas semanas estava focado em reestruturar um dos meus projetos que é basicamente uma game framework/engine que utilize Lua como linguagem de script, o projeto inicialmente nasceu como poti, mas há algum tempo estava tentando ressignificar outro dos meus projetos, a selene, e por achar que o nome casa mais com o projeto em questão, decidi fazer essa troca.

Inicialmente eu estava focado em fazer a framework base em C e operar ela utilizando Lua, então a ideia seria ter um renderizador básico em C, uma engine de áudio básica também, etc. Maaas, decidi seguir por um caminho diferente, e até explorar mais a ideia inicial do projeto que é a de ter um core simples e o projeto ser o mais modular possível via Lua, então ao invés de construir essas estruturas em C usando as libs (SDL2, OpenGL, …), achei melhor expor as funções das lib pra Lua e construir a framework lá.

Então por exemplo, uma função de desenhar um retângulo em C:

```
static int l_graphics_fill_rectangle(lua_State* L) {
    float *c = RENDER()->color;
    float x, y, w, h;
    x = luaL_checknumber(L, 1);
    y = luaL_checknumber(L, 2);
    w = luaL_checknumber(L, 3);
    h = luaL_checknumber(L, 4);
    set_texture(RENDER()->white_texture);
    float vertices[] = {
        x, y, c[0], c[1], c[2], c[3], 0.f, 0.f,
        x + w, y, c[0], c[1], c[2], c[3], 1.f, 0.f,
        x + w, y + h, c[0], c[1], c[2], c[3], 1.f, 1.f,

        x, y, c[0], c[1], c[2], c[3], 0.f, 0.f,
        x, y + h, c[0], c[1], c[2], c[3], 0.f, 1.f,
```

```
        x + w, y + h, c[0], c[1], c[2], c[3], 1.f, 1.f,

    };

    VertexFormat* v = (VertexFormat*)vertices;
    push_vertices(VERTEX(), 6, v);
}
```

Vira isso:

```
function graphics.fill_rectangle(x, y, width, height)
  set_image()
  set_draw_mode('triangles')

  local r,g,b,a = table.unpack(current.draw_color)
  local vertex_data = default.batch.data
  default.batch:push(x, y, r, g, b, a, 0.0, 0.0)
  default.batch:push(x+w, y, r, g, b, a, 0.0, 0.0)
  default.batch:push(x+width, y+height, r, g, b, a, 0.0, 0.0)

  default.batch:push(x, y, r, g, b, a, 0.0, 0.0)
  default.batch:push(x+width, y+height, r, g, b, a, 0.0, 0.0)
  default.batch:push(x, y+height, r, g, b, a, 0.0, 0.0)
end
```

Obviamente isso tem custos em relação a performance, mas dependendo da proposta do projeto funciona muito bem, fora que a ideia é a partir desse ponto construir uma engine em cima disso, então posso usar toda a modularidade que Lua me permite pra otimizar boa parte dessas chamadas de funções, e caso necessário é só refazer partes em C.

Perceba aqui a mudança na estrutura da biblioteca:

poti (old)	selene (C)	selene (Lua)
poti ├── audio │ └── AudioData ├── event ├── filesystem │ └── File ├── gamepad │ └── Gamepad ├── graphics │ └── Texture ├── joystick ├── keyboard ├── mouse └── window	selene (C) ├── Data ├── audio │ └── Decoder ├── font ├── fs │ └── File ├── gl │ ├── Buffer │ ├── Framebuffer │ ├── Program │ ├── Shader │ ├── Texture │ └── VertexArray ├── image .2 linmath .3 Mat4 └── sdl2 ├── AudioDeviceID ├── AudioStream ├── Event ├── Gamepad ├── GLContext ├── Joystick └── Window	Selene (Lua) ├── audio │ ├── Music │ └── Sound ├── filesystem ├── graphics │ ├── Batch │ ├── Canvas │ ├── Font │ ├── Image │ ├── Rect │ └── Shader ├── gamepad ├── joystick ├── keyboard └── mouse

Como expliquei acima, agora estou focando muito mais em expor as minhas

libs para Lua.

O executável ainda terá um pequeno script de boot embutido nele exatamente para dizer como nossa aplicação irá iniciar. A lógica por trás dele é de adicionar a pasta do executável no path do Lua, e então buscar por um módulo **main** .

```
local sdl = selene.sdl2
local function add_path(path)
    package.path = path .. '?.lua;' .. path .. '?/init.lua;' ..
    package.path
end
return function(args)
    add_path(sdl.GetBasepath())
    if selene.args[2] then add_path(selene.args[2]) end
    return require('main')
end
```

No módulo main você poderá fazer uso dos scripts da framework em Lua, que estarão na pasta **core/** que será distribuído junto com o executável.

A partir daí me aproveitar da modularidade do Lua pra criar um programa que possa ao mesmo tempo ter uma engine e na hora de distribuir ter só um runner. Então posteriormente distribuir também uma pasta **engine/** , **runner/** , **editor/** , e assim vai. Fazer uma API para plugins também.

Enfim, no final a decisão foi mais pra não me preocupar tanto com performance e me aproveitar mais da liberdade do Lua pra criar diferentes sistemas e um certo nível de modularidade, como mudar o renderizador em tempo real (que é uma das ideias, inclusive).