学习ejoy2d——sprite

gaccob

2014 年 3 月 14 日

1. sprite是什么

"sprite是ejoy2d中可以处理的基本图形对象, 每个 sprite 都是若干图元以树状组合起来的". 目前只有animation图元类型的sprite有孩子, 其他都是单个节点的存在.

2. sprite属性

结合github上的文档说明, 先看一下C kernel中sprite数据结构定义.

```
// file:  lib/spritepack.h
struct sprite_trans {
    struct matrix * mat;
    uint32_t color;
    uint32_t additive;
    int program;
};

// file:  lib/sprite.h
struct sprite {

    // 父节点, 与children节点一起, 维系了树状结构
    // lua接口: sprite.has_parent(只读), sprite.parent_name(只读)
    struct sprite * parent;

    // 图元类型
    uint16_t type;

    // 唯一id, 因为是数组, 所以不建议散的太开
    uint16_t id;

    // t.mat 渲染时的变换矩阵, 运行期, 默认单位矩阵
```

```
23      // t.color 渲染时的混合颜色, ARGB32, 默认为0xFFFFFFFF, 作用域是整个
            子树, 最常见的是做alpha半透明效果, 例如0x80FFFFFF就是50%的半透明
24      // t.additive 渲染时的叠加颜色, RGB24, 默认为0, 作用域是整个子树
25      // t.program 指定shader
26      // Lua API: sprite.matrix(读写), sprite.color(读写),
            sprite.additive(读写), sprite.program(只读)
27      struct sprite_trans t;
28
29      // 5种基本图元
30      union {
31          struct pack_animation *ani;
32          struct pack_picture *pic;
33          struct pack_polygon *poly;
34          struct pack_label *label;
35          struct pack_pannel *pannel;
36          struct matrix *mat;
37      } s;
38
39      // 只读anchor的特殊属性, 返回上一次这个anchor对象最终渲染的世界矩阵
40      // anchor.visible=false, 当不可显时, 引擎不计算 world matrix
41      // Lua API: sprite.wolrd_matrix(只读)
42      struct matrix mat;
43
44      // 总帧数 与 开始帧数
45      // Lua API: sprite.frame_count(只读)
46      int start_frame;
47      int total_frame;
48
49      // 当前帧号
50      // Lua API: sprite.frame(读写)
51      int frame;
52
53      // 如果设置为false, 则整个子树不显示
54      // Lua API: sprite.visible(读写)
55      bool visible;
56
57      // 对象是否截获 test 调用, 多用于 UI 控制
58      // Lua API: sprite.message(读写)
59      bool message;
60
61      // Lua API: sprite.name(只读)
62      const char *name;
63
64      union {
65          struct sprite * children[1];
66
67          // label的文字
68          // Lua API: sprite.text(读写)
69          const char * text;
```

```
70
71          // panel是否有scissor
72          // Lua API: sprite.scissor(只读)
73          int scissor;
74      } data;
75  };
76
77  // 上面注释中提到的Lua API基本都包含在了getter&setter中
78  // file:  lib/lsprite.c
79  static void
80  lgetter(lua_State *L) {
81      luaL_Reg l[] = {
82          {"frame", lgetframe},
83          {"frame_count", lgettotalframe },
84          {"visible", lgetvisible },
85          {"name", lgetname },
86          {"text", lgettext},
87          {"color", lgetcolor },
88          {"additive", lgetadditive },
89          {"message", lgetmessage },
90          {"matrix", lgetmat },
91          {"world_matrix", lgetwmat },
92          {"parent_name", lgetparentname },
93          {"has_parent", lhasparent },
94          {NULL, NULL},
95      };
96      luaL_newlib(L,l);
97  }
98
99  static void
100 lsetter(lua_State *L) {
101     luaL_Reg l[] = {
102         {"frame", lsetframe},
103         {"action", lsetaction},
104         {"visible", lsetvisible},
105         {"matrix" , lsetmat},
106         {"text", lsettext},
107         {"color", lsetcolor},
108         {"additive", lsetadditive },
109         {"message", lsetmessage },
110         {"program", lsetprogram },
111         {"scissor", lsetscissor },
112         {NULL, NULL},
113     };
114     luaL_newlib(L,l);
115 }
```

上层Lua中, 依据setter&getter接口, 设置sprite的metatable.

```
// file:  ejoy2d/sprite.lua
function sprite_meta.__index(spr, key)
    if method[key] then
        return method[key]
    end
    local getter = get[key]
    if getter then
        return getter(spr)
    end
    local child = fetch(spr, key)

    if child then
        return child
    else
        print("Unsupport get " ..  key)
        return nil
    end
end

function sprite_meta.__newindex(spr, key, v)
    local setter = set[key]
    if setter then
        setter(spr, v)
        return
    end
    assert(debug.getmetatable(v) == sprite_meta, "Need a
        sprite")
    method.mount(spr, key, v)
end
```

3. sprite.new

Lua接口sprite.new的实现如下:

```
-- file: ejoy2d/sprite.lua
function sprite.new(packname, name)

    -- 这里的资源是预先已经初始化过的
    local pack, id = pack.query(packname, name)

    -- 调用C接口实现new
    local cobj = c.new(pack,id)

    -- 设置meta
    if cobj then
        return debug.setmetatable(cobj, sprite_meta)
    end
```

```
end

-- file: ejoy2d/spritepack.lua
-- 从已经import的资源中查询spritepack数据, name可以是名字或者id
function spritepack.query( packname, name )
    local p = assert(pack_pool[packname], "Load package first")
    local id
    if type(name) == "number" then
        id = name
    else
        id = p.export[name]
    end
    if not id then
        error(string.format("'%s' is not exist in package %s", name, packname))
    end
    return p.cobj, id
end
```

相关的C代码如下:

```
// file: lib/lsprite.c
// 输入: userdata spritepack
//       number id
// 输出: userdata sprite
static int
lnew(lua_State *L) {
    struct sprite_pack * pack = (struct sprite_pack *)lua_touserdata(L, 1);
    if (pack == NULL) {
        return luaL_error(L, "Need a sprite pack");
    }
    int id = (int)luaL_checkinteger(L, 2);
    struct sprite * s = newsprite(L, pack, id);
    if (s) {
        return 1;
    }
    return 0;
}

// 图元类型的sprite new
static struct sprite *
newsprite(lua_State *L, struct sprite_pack *pack, int id) {
    // anchor类型单独new
    if (id == ANCHOR_ID) {
        return newanchor(L);
    }

```

```
27      // 计算sprite size, 从Lua VM中分配内存
28      int sz = sprite_size(pack, id);
29      if (sz == 0) {
30          return NULL;
31      }
32      struct sprite * s = (struct sprite *)lua_newuserdata(L,
            sz);
33
34      // 从spritepack初始化sprite数据
35      sprite_init(s, pack, id, sz);
36
37      // 递归载入孩子sprite(只有animation会触发)
38      int i;
39      for (i=0;;i++) {
40
41          // 引用的component
42          int childid = sprite_component(s, i);
43          if (childid < 0)
44              break;
45
46          if (i==0) {
47              lua_newtable(L);
48              lua_pushvalue(L,-1);
49              lua_setuservalue(L, -3);    // set uservalue for
                    sprite
50          }
51
52          // 初始化child
53          struct sprite *c = newsprite(L, pack, childid);
54
55          // 设置child名字
56          c->name = sprite_childname(s, i);
57
58          // 挂载child, 组织成树状结构
59          sprite_mount(s, i, c);
60
61          // 如果touchable, sprite->message设置为true
62          update_message(c, pack, id, i, s->frame);
63
64          // todo:
65          if (c) {
66              lua_rawseti(L, -2, i+1);
67          }
68      }
69      if (i>0) {
70          lua_pop(L,1);
71      }
72      return s;
73  }
```

```
74
75  // anchor类型的sprite, 没有spritepack数据
76  static struct sprite *
77  newanchor(lua_State *L) {
78      // 比常规sprite多了个matrix
79      // 此处代码加到sprite_size()更好看一些, 这里为了少一次函数调用?
80      int sz = sizeof(struct sprite) + sizeof(struct matrix);
81      struct sprite * s = (struct sprite *)lua_newuserdata(L,
            sz);
82
83      s->parent = NULL;
84      s->t.mat = NULL;
85      s->t.color = 0xffffffff;
86      s->t.additive = 0;
87      s->t.program = PROGRAM_DEFAULT;
88      s->message = false;
89
90      // 默认不可见
91      s->visible = false;
92
93      s->name = NULL;
94
95      // 所有的anchor类型都是一个默认id
96      s->id = ANCHOR_ID;
97
98      s->type = TYPE_ANCHOR;
99
100     // anchor类型用到的matrix, 不像其他图元类型sprite数据在pack包中,
101     // 所以这里要自己分配内存, 在sprite对象之后
102     s->s.mat = (struct matrix *)(s+1);
103     matrix_identity(s->s.mat);
104
105     return s;
106 }
107
108 // file: lib/sprite.c
109 // 从spritepack数据初始化sprite对象
110 void
111 sprite_init(struct sprite * s, struct sprite_pack * pack, int
        id, int sz) {
112     if (id < 0 || id >= pack->n)
113         return;
114     s->parent = NULL;
115     s->t.mat = NULL;
116     s->t.color = 0xffffffff;
117     s->t.additive = 0;
118     s->t.program = PROGRAM_DEFAULT;
119     s->message = false;
120     s->visible = true;
```

```
121     s->name = NULL;
122     s->id = id;
123     s->type = pack->type[id];
124 
125     // animation类型单独处理, 设置frame和action
126     if (s->type == TYPE_ANIMATION) {
127         struct pack_animation * ani = (struct pack_animation
                *)pack->data[id];
128         s->s.ani = ani;
129         s->frame = 0;
130 
131         // 默认只有一个action, 传入NULL
132         sprite_action(s, NULL);
133 
134         // 这里时保证分配的内存足够用
135         // animation包括孩子节点, 多出来n-1个child指针
136         // child默认都是NULL
137         int i;
138         int n = ani->component_number;
139         assert(sz >= sizeof(struct sprite) + (n - 1) * sizeof
                (struct sprite *));
140         for (i=0; i<n ;i++) {
141             s->data.children[i] = NULL;
142         }
143     }
144     // 其他类型
145     else {
146         // 因为是union, data是void*, 指定任意其他图元类型都一样
147         s->s.pic = (struct pack_picture *)pack->data[id];
148 
149         // 非animation无需frame
150         s->start_frame = 0;
151         s->total_frame = 0;
152         s->frame = 0;
153 
154         // API sprite.text设置, 默认NULL
155         s->data.text = NULL;
156 
157         // 减掉一个指针大小, 是因为无需计算animation child, 还是代码没修
                改?
158         assert(sz >= sizeof(struct sprite) - sizeof(struct
                sprite *));
159 
160         // pannel类型需要设置scissor
161         if (s->type == TYPE_PANNEL) {
162             struct pack_pannel * pp = (struct pack_pannel *)
                    pack->data[id];
163             s->data.scissor = pp->scissor;
164         }
```

```
165         }
166 }
```