

学习ejoy2d——shader

gaccob

2014 年 2 月 21 日

1. shader简单介绍

wiki上这么描述： shader(着色器)指一组供计算机图形资源在执行渲染任务时使用的指令。 shader是render的一部分，运行在GPU上，负责计算目标颜色。 OpenGL从1.5开始继承一种类C的着色语言，称为OpenGL Shader Language.

shader分两种，一种是顶点shader(OpenGL中是vertex shader)，目的是计算顶点位置，为后期像素渲染做准备；一种是像素shader(OpenGL中是fragment shader)，以像素为单位，计算光照和颜色。

2. ejoy2d.shader数据结构

```
1 // 分别是screen 和 texture 的坐标
2 struct vertex {
3     float vx;
4     float vy;
5     float tx;
6     float ty;
7     uint8_t rgba[4];
8 };
9
10 // 1个quad, 4个顶点
11 struct quad {
12     struct vertex p[4];
13 };
14
15 // 这个东东处理了所有渲染部分的工作
16 struct render_state {
17
18     // 当前的shader program
19     int current_program;
```

```
20 // ejoy2d支持最多6种 shader program, 这个会在lua中定义
21 struct program program[MAX_PROGRAM];
22
23 // texture id (OpenGL id)
24 int tex;
25
26 // 需要渲染的quad的数量, 在rs_commit()计算时需要用到
27 int object;
28
29 // 默认的blend方式 (这个下面代码有描述), 该值为0; 自定义blend方式时,
30 // 这个值=1
31 int blendchange;
32
33 // 顶点buffer
34 GLuint vertex_buffer;
35
36 // 索引buffer
37 GLuint index_buffer;
38
39 // 最多64个quad
40 struct quad vb[MAX_COMMBINE];
41 };
42
43 // 全局渲染状态机
44 static struct render_state *RS = NULL;
```

3. ejoy2d.shader初始化

```
1 // 初始化shader, 这个会在程序启动时调用
2 void
3 shader_init() {
4     assert(RS == NULL);
5     struct render_state * rs = (struct render_state *)
6         malloc(sizeof(*rs));
7     memset(rs, 0 , sizeof(*rs));
8     rs->current_program = -1;
9
10    // 设置颜色混合的模式
11    // ejoy2d还提供了shader_defaultblend()和shader_blend()接口来操作
12    // blend方式
13    rs->blendchange = 0;
14    glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
15
16    // 索引buffer
17    glGenBuffers(1, &rs->index_buffer);
```

```

17         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, rs->index_buffer);
18
19         GLubyte idxs[6 * MAX_COMMBINE];
20         int i;
21         for (i=0; i<MAX_COMMBINE; i++) {
22             idxs[i*6] = i*4;
23             idxs[i*6+1] = i*4+1;
24             idxs[i*6+2] = i*4+2;
25             idxs[i*6+3] = i*4;
26             idxs[i*6+4] = i*4+2;
27             idxs[i*6+5] = i*4+3;
28         }
29
30         // GL_STATIC_DRAW表示索引是固定的
31         // 上面的索引idxs, 实际上是将quad的4个顶点, 转为两个三角面, 节约了2
           个冗余顶点
32         glBufferData(GL_ELEMENT_ARRAY_BUFFER, 6*MAX_COMMBINE,
           idxs, GL_STATIC_DRAW);
33
34         // 顶点buffer, 这里的buffer会在程序运行中实时加载进来
35         glGenBuffers(1, &rs->vertex_buffer);
36         glBindBuffer(GL_ARRAY_BUFFER, rs->vertex_buffer);
37
38         glEnable(GL_BLEND);
39
40         RS = rs;
41     }

```

这里值得说一下的是OpenGL的颜色混合方式, 假设源颜色(Rs, Gs, Bs, As), 目标颜色为(Rd, Gd, Bd, Ad), OpenGL分别讲源颜色和目标颜色乘一个系数, 就得到了混合的结果. 这里的系数就由glBlendFunc()指定.

第一个参数GL_ONE表示使用1.0作为源颜色的系数, 第二个参数, GL_ONE_MINUS_SRC_ALPHA表示以1.0减去As的值作为目标颜色的系数.

具体的细节可以参考这一篇文章 [《颜色混合opengl》](#), 这里不再赘述.

4. ejoy2d.shader的加载

在前文讲过, render_state维护了一个预先加载的shader program数组. 可以从shader.lua中读到, shader程序有: sprite_fs, sprite_vs, text_fs, text_edge_fs, gray_fs 以及 color_fs, fs和vs组合后有5种shader.

```

1  — lua 中的shader name
2  local shader_name = {
3      NORMAL = 0,
4      TEXT = 1,

```

本着自由的精神, 本文档可以随意阅读, 修改, 发布; 如涉及相关引用的版权问题, 请联系gaccob@qq.com及时修改.

```
5         EDGE = 2,
6         GRAY = 3,
7         COLOR = 4,
8     }
9
10    — 在init时加载全部5种shader
11    function shader.init()
12        s.load(shader_name.NORMAL, PRECISION .. sprite_fs,
13              PRECISION .. sprite_vs)
14        s.load(shader_name.TEXT, PRECISION .. text_fs, PRECISION
15              .. sprite_vs)
16        s.load(shader_name.EDGE, PRECISION .. text_edge_fs,
17              PRECISION .. sprite_vs)
18        s.load(shader_name.GRAY, PRECISION .. gray_fs, PRECISION
19              .. sprite_vs)
20        s.load(shader_name.COLOR, PRECISION .. color_fs,
21              PRECISION .. sprite_vs)
22    end
```

```
1
2    // 编译shader代码
3    static GLuint
4    compile(const char * source, int type) {
5        .....
6    }
7
8    // 链接编译后的shader
9    static void
10    link(struct program *p) {
11        .....
12    }
13
14    // 如果shader中存在addi, 设置为对应的color值
15    static void
16    set_color(GLint addi, uint32_t color) {
17        if (addi == -1)
18            return;
19        if (color == 0) {
20            glUniform3f(addi, 0,0,0);
21        } else {
22            float c[3];
23            c[0] = (float)((color >> 16) & 0xff) / 255.0f;
24            c[1] = (float)((color >> 8) & 0xff) / 255.0f;
25            c[2] = (float)(color & 0xff) / 255.0f;
26            glUniform3f(addi, c[0],c[1],c[2]);
27        }
28    }
29
30    // 加载shader program
```

```
31 static void
32 program_init(struct program * p, const char *FS, const char
    *VS) {
33     // Create shader program.
34     p->prog = glCreateProgram();
35
36     // 编译FS, 像素shader
37     GLuint fs = compile(FS, GL_FRAGMENT_SHADER);
38     if (fs == 0) {
39         fault("Can't compile fragment shader");
40     } else {
41         glAttachShader(p->prog, fs);
42     }
43
44     // 编译VS, 顶点shader
45     GLuint vs = compile(VS, GL_VERTEX_SHADER);
46     if (vs == 0) {
47         fault("Can't compile vertex shader");
48     } else {
49         glAttachShader(p->prog, vs);
50     }
51
52     // 绑定顶点shader中的attribute 到这里的ATTRIB_*变量
53     // 这里的position, texcoord和color 是sprites_vs shader中的attribute
54     glBindAttribLocation(p->prog, ATTRIB_VERTEX, "position")
55     ;
56     glBindAttribLocation(p->prog, ATTRIB_TEXCOORD, "
        texcoord");
57     glBindAttribLocation(p->prog, ATTRIB_COLOR, "color");
58
59     // 链接
60     link(p);
61
62     // 获取像素shader中的uniform变量 additive, (一个偏移量, 默认是0)
63     p->additive = glGetUniformLocation(p->prog, "additive");
64     p->arg = 0;
65     set_color(p->additive, 0);
66
67     // 删除shader, link完之后, fs和vs就不用了.
68     // 跟平时c的编译其实很类似, link成lib或者bin之后, .o文件就不用了.
69     glDetachShader(p->prog, fs);
70     glDeleteShader(fs);
71     glDetachShader(p->prog, vs);
72     glDeleteShader(vs);
73
74     // 加载shader程序
75     void
76     shader_load(int prog, const char *fs, const char *vs) {
```

```
77     struct render_state *rs = RS;
78     assert(prog >=0 && prog < MAX_PROGRAM);
79     struct program * p = &rs->program[prog];
80     assert(p->prog == 0);
81     program_init(p, fs, vs);
82 }
83
84 // 卸载shader
85 // 其实我认为不仅是unload 还有release 为什么不写成两个函数呢?
86 void
87 shader_unload() {
88     if (RS == NULL) {
89         return;
90     }
91     int i;
92
93     // 卸载所有的shader程序
94     for (i=0; i<MAX_PROGRAM; i++) {
95         struct program * p = &RS->program[i];
96         if (p->prog) {
97             glDeleteProgram(p->prog);
98         }
99     }
100
101     // 删除顶点buffer和索引buffer
102     glDeleteBuffers(1, &RS->vertex_buffer);
103     glDeleteBuffers(1, &RS->index_buffer);
104
105     // 释放全局渲染状态机
106     free(RS);
107     RS = NULL;
108 }
```

5. ejoy2d.shader的渲染

ejoy2d中用了OpenGL的VAO来做渲染，具体的细节可以参考这一片文章《OpenGL.Vertex Array Object (VAO)》。

```
1 // 渲染的过程，这里对quad利用索引buffer做了一些优化(节省了冗余的vertex)
2 static void
3 rs_commit() {
4     if (RS->object == 0)
5         return;
6
7     // 顶点buffer，GL_DYNAMIC_DRAW说明这里每一帧可能会渲染多次
8     glBindBuffer(GL_ARRAY_BUFFER, RS->vertex_buffer);
```

```
9      glBufferData(GL_ARRAY_BUFFER, sizeof(struct quad) * RS->
      object, RS->vb, GL_DYNAMIC_DRAW);
10
11      // 指定ATTRIB_VERTEX -> shader中的position
12      // 2个float, 间隔一个struct vertex, offset=0, 对应vertex->vx,
      vertex->vy
13      glEnableVertexAttribArray(ATTRIB_VERTEX);
14      glVertexAttribPointer(ATTRIB_VERTEX, 2, GL_FLOAT,
      GL_FALSE, sizeof(struct vertex), BUFFER_OFFSET(0));
15
16      // 指定ATTRIB_TEXTCOORD -> shader中的texcoord
17      // 2个float, 间隔一个struct vertex, offset=8=2*sizeof(GL_FLOAT) 对应
      vertex->tx, vertex->ty
18      glEnableVertexAttribArray(ATTRIB_TEXTCOORD);
19      glVertexAttribPointer(ATTRIB_TEXTCOORD, 2, GL_FLOAT,
      GL_FALSE, sizeof(struct vertex), BUFFER_OFFSET(8));
20
21      // 指定ATTRIB_COLOR -> shader中的color
22      // 4个unsigned byte, 间隔一个struct vertex,
      offset=16=4*sizeof(GL_FLOAT) 对应vertex->rgba
23      glEnableVertexAttribArray(ATTRIB_COLOR);
24      glVertexAttribPointer(ATTRIB_COLOR, 4, GL_UNSIGNED_BYTE,
      GL_TRUE, sizeof(struct vertex), BUFFER_OFFSET(16));
25
26      // 使用顶点buffer绘制, count=6*object, 与索引buffer一致
27      glDrawElements(GL_TRIANGLES, 6 * RS->object,
      GL_UNSIGNED_BYTE, 0);
28      RS->object = 0;
29  }
30
31  // 渲染一个quad
32  void
33  shader_draw(const float vb[16], uint32_t color) {
34      struct quad *q = RS->vb + RS->object;
35      int i;
36      for (i=0; i<4; i++) {
37          q->p[i].vx = vb[i*4+0];
38          q->p[i].vy = vb[i*4+1];
39          q->p[i].tx = vb[i*4+2];
40          q->p[i].ty = vb[i*4+3];
41          q->p[i].rgba[0] = (color >> 16) & 0xff;
42          q->p[i].rgba[1] = (color >> 8) & 0xff;
43          q->p[i].rgba[2] = (color) & 0xff;
44          q->p[i].rgba[3] = (color >> 24) & 0xff;
45      }
46      if (++RS->object >= MAX_COMMBINE) {
47          rs_commit();
48      }
49  }
50
```

```
51 // 渲染一个polygon, 这里调用glDrawArrays(GL_TRIANGLE_FAN, ...)来实现绘制
52 // GL_TRIANGLE_FAN与GL_TRIANGLE_STRIP的区别, 可以参考这篇文章
53 void
54 shader_drawpolygon(int n, const float *vb, uint32_t color) {
55     rs_commit();
56     struct vertex p[n];
57     int i;
58     for (i=0; i<n; i++) {
59         p[i].vx = vb[i*4+0];
60         p[i].vy = vb[i*4+1];
61         p[i].tx = vb[i*4+2];
62         p[i].ty = vb[i*4+3];
63         p[i].rgba[0] = (color >> 16) & 0xff;
64         p[i].rgba[1] = (color >> 8) & 0xff;
65         p[i].rgba[2] = (color) & 0xff;
66         p[i].rgba[3] = (color >> 24) & 0xff;
67     }
68
69     glBindBuffer(GL_ARRAY_BUFFER, RS->vertex_buffer);
70     glBufferData(GL_ARRAY_BUFFER, sizeof(struct vertex) * n,
71                 (void*)p, GL_DYNAMIC_DRAW);
72
73     glEnableVertexAttribArray(ATTRIB_VERTEX);
74     glVertexAttribPointer(ATTRIB_VERTEX, 2, GL_FLOAT,
75                           GL_FALSE, sizeof(struct vertex), BUFFER_OFFSET(0));
76     glEnableVertexAttribArray(ATTRIB_TEXTCOORD);
77     glVertexAttribPointer(ATTRIB_TEXTCOORD, 2, GL_FLOAT,
78                           GL_FALSE, sizeof(struct vertex), BUFFER_OFFSET(8));
79     glEnableVertexAttribArray(ATTRIB_COLOR);
80     glVertexAttribPointer(ATTRIB_COLOR, 4, GL_UNSIGNED_BYTE,
81                           GL_TRUE, sizeof(struct vertex), BUFFER_OFFSET(16));
82     glDrawArrays(GL_TRIANGLE_FAN, 0, n);
83 }
```

6. ejoy2d.shader的lua接口

shader的lua接口代码, 都在lib/lshader.c中.

```
1 int
2 ejoy2d_shader(lua_State *L) {
3     luaL_Reg l[] = {
4         {"load", lload},
5         {"unload", lunload},
6         {"draw", ldraw},
7         {"blend", lblend},
8         {"version", lversion},
9     }
```



```
9         {NULL, NULL},
10     };
11     luaL_newlib(L, l);
12     return 1;
13 }
```

这里基本就是从lua读参数，然后调用C接口实现，源码很清晰，就不列举了。单独说一下shader.draw这个接口。

```
1 // 栈里的lua参数:
2 // int texture
3 // table float[16]: 先texture(tx,ty)列表 再screen(vx, vy)列表
4 // uint32_t color
5 // uint32_t additive
6 static int
7 ldraw(lua_State *L) {
8     int tex = (int) luaL_checkinteger(L, 1);
9     int texid = texture_glid(tex);
10    if (texid == 0) {
11        lua_pushboolean(L, 0);
12        return 1;
13    }
14    luaL_checktype(L, 2, LUA_TTABLE);
15    uint32_t color = 0xffffffff;
16
17    if (!lua_isnoneornil(L, 3)) {
18        color = (uint32_t) lua_tounsigned(L, 3);
19    }
20
21    // 设置program和additive
22    uint32_t additive = (uint32_t) luaL_optunsigned(L, 4, 0);
23    shader_program(PROGRAM_PICTURE, additive);
24
25    // 设置texture
26    shader_texture(texid);
27
28    // 每个vertex有vx,vy,tx,ty, 所以必然是4的倍数
29    int n = lua_rawlen(L, 2);
30    int point = n/4;
31    if (point * 4 != n) {
32        return luaL_error(L, "Invalid polygon");
33    }
34
35    float vb[n];
36    int i;
37    for (i=0; i<point; i++) {
38        // get (tx, ty) (vx, vy)
39        lua_rawgeti(L, 2, i*2+1);
40        lua_rawgeti(L, 2, i*2+2);
```

```

41         lua_rawgeti(L, 2, point*2+i*2+1);
42         lua_rawgeti(L, 2, point*2+i*2+2);
43         float tx = lua_tonumber(L, -4);
44         float ty = lua_tonumber(L, -3);
45         float vx = lua_tonumber(L, -2);
46         float vy = lua_tonumber(L, -1);
47         lua_pop(L, 4);
48
49         // screen 坐标系 normalize
50         screen_trans(&vx,&vy);
51
52         // texture 坐标系 normalize
53         texture_coord(tex, &tx, &ty);
54
55         // 这个是为了坐标系对齐, screen坐标系原点(0, 0)是屏幕中间, 现在
           调整到左下角
56         vb[i*4+0] = vx + 1.0f;
57         vb[i*4+1] = vy - 1.0f;
58         vb[i*4+2] = tx;
59         vb[i*4+3] = ty;
60     }
61
62     // quad
63     if (point == 4) {
64         shader_draw(vb, color);
65     }
66     // polygon
67     else {
68         shader_drawpolygon(point, vb, color);
69     }
70     return 0;
71 }

```

7. ejoy2d.shader的sample

可以参考examples/ex02.lua.

```

1
2     local shader = require "ejoy2d.shader"
3
4     — 这里是直接调用shader绘制的sample, 注意一下这里屏幕坐标的中心原点
5     function game.drawframe()
6         — use shader.draw to draw a polygon to screen (for
           debug use)
7         shader.draw(TEXTID, {
8             88, 0, 88, 45, 147, 45, 147, 0, — texture coord
9             -958, -580, -958, 860, 918, 860, 918, -580, —
           screen coord, 16x pixel, (0,0) is the center

```

本着自由的精神, 本文档可以随意阅读, 修改, 发布; 如涉及相关引用的版权问题, 请联系gaccob@qq.com及时修改.

```
10         }) of screen
11     end
```