

学习ejoy2d——ppm和texture

gaccob

2004 年 2 月 5 日

ppm贴图

- pbm, portable bitmap, 单色图(1 bit).
- pgm, portable gray map, 灰度图.
- ppm, portable pixel map, 真彩图.

pbm / pgm / ppm 图像的文件格式分为两部分： 文件头和数据部分。
一个典型ppm头的sample:

```
1  p6 # ppm格式
2  1024 1024 # 高, 宽
3  255 # 深度, 不一定是255
```

ppm的格式有p3和p6, p3表示用ascii码(文本)来表示数据, p6表示以字节码(二进制)来表示, 每一个像素按(r, g, b)的格式来存储.

```
1  P3
2  4 4
3  15
4  0 0 0 0 0 0 0 0 0 15 0 15
5  0 0 0 0 15 7 0 0 0 0 0 0
6  0 0 0 0 0 0 0 15 7 0 0 0
7  15 0 15 0 0 0 0 0 0 0 0 0
```

pgm与ppm类似, 格式有p2和p5, p2表示文本, p5表示二进制.

```
1  P2
2  18 7
3  15
4  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```

5      0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0
6      0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0
7      0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0
8      0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0
9      0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0
10     0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

对于pbm来说, 格式为p1, 但是文件头中没有最大颜色, 因为用0和1来表示就可以了.

```

1      P1
2      24 7
3      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4      0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
5      0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
6      0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
7      0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
8      0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
9      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

参考文章: <http://www.cppblog.com/windcsn/archive/2005/11/11/ppm.html>

ejoy2d中的ppm源码

ejoy2d中的ppm贴图的处理都在lib/ppm.h和lib/ppm.c中. ejoy2d中用pgm贴图来描述alpha通道, 用ppm贴图来描述rgb通道.

```

1  struct ppm {
2      // 指ppm贴图的格式, p1-p6分别对应1-6
3      int type;
4      // 图像的深度, 一般有255(8位)和15(4位)
5      int depth;
6      // 步长, alpha是1, rgb是3, rgba就是4
7      int step;
8      // 长 & 宽
9      int width;
10     int height;
11     // 图像数据
12     uint8_t *buffer;
13 };
14
15 // 载入ppm文件头
16 static int
17 ppm_header(FILE *f, struct ppm *ppm) {
18     .....
19     // 格式

```

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.

```
20     char c = 0;
21     sscanf(line, "P%c", &c);
22     .....
23     // 长 & 宽
24     sscanf(line, "%d %d", &(ppm->width), &(ppm->height));
25     .....
26     // 图像深度, 这里就是255或者15
27     sscanf(line, "%d", &(ppm->depth));
28     .....
29 }
30
31 // 载入ppm文件数据
32 // 这里会根据type(p1, p2, ...)的不同, 做对应的解析并载入
33 // skip是为了有一个初始offset(适用于rgb ppm载入alpha的情况)
34 static int
35 ppm_data(struct ppm *ppm, FILE *f, int id, int skip) {
36     .....
37     switch(id) {
38         case '3':      // RGB text
39             .....
40         case '2':      // ALPHA text
41             .....
42         case '6':      // RGB binary
43             .....
44         case '5':      // ALPHA binary
45             .....
46     }
47 }
48
49 // 载入ppm文件, 调用ppm_header()和ppm_data()完成.
50 // 如果是rgba, 需要从两个贴图文件一起载入(会做一致性校验)
51 static int
52 loadppm_from_file(FILE *rgb, FILE *alpha, struct ppm *ppm) {
53     .....
54 }
55
56 // 载入ppm文件的lua接口
57 // lua输入参数:
58 //   string ppm_name
59 // 输出lua结果:
60 //   string format(这里约定的格式有: RGBA8, RGB8, ALPHA8, RGBA4, RGB4, ALPHA4)
61 //   int width
62 //   int height
63 //   table buffer(ppm数据部分)
64 static int
65 loadppm(lua_State *L) {
66     .....
67 }
68
```

```
69 // 载入ppm文件到texture(纹理)的lua接口
70 // lua输入参数:
71 // string ppm_name
72 static int
73 loadtexture(lua_State *L) {
74     // 载入ppm文件, 调用loadppm_from_file()完成
75     .....
76
77     // 根据ppm的格式, 设置texture的格式
78     int type = 0;
79     if (ppm.depth == 255) {
80         if (ppm.step == 4) {
81             type = Texture2DPixelFormat_RGBA8888;
82         } else if (ppm.step == 3) {
83             type = Texture2DPixelFormat_RGB888;
84         } else {
85             type = Texture2DPixelFormat_A8;
86         }
87     }
88     // depth为15, 需要根据step的不同(即texture格式的不同)做buffer的转换
89     else {
90         if (ppm.step == 4) {
91             .....
92         } else if (ppm.step == 3) {
93             .....
94         } else {
95             .....
96         }
97     }
98
99     // 最后加载texture, 这一块具体可以参照下节
100     const char * err = texture_load(id, type, ppm.width, ppm.
        height, ppm.buffer);
101     free(ppm.buffer);
102     if (err) {
103         return luaL_error(L, "%s", err);
104     }
105     return 0;
106 }
107
108 // 根据format(上面描述的RGBA8等格式), 设置ppm数据: type, depth, step
109 static void
110 ppm_type(lua_State *L, const char * format, struct ppm *ppm) {
111     .....
112 }
113
114 // 从lua中读取数据, 保存rgb的ppm贴图(写文件), P6二进制格式.
115 static void
116 save_rgb(lua_State *L, int step, int depth) {
```

```
117     .....
118 }
119
120 // 从lua中读取数据, 保存alpha的pgm贴图(写文件), P5二进制格式.
121 static void
122 save_alpha(lua_State *L, int step, int depth, int offset) {
123     .....
124 }
125
126 // 保存ppm文件的lua接口, 调用save_rgb()和save_alpha()实现, 写文件.
127 // lua输入参数:
128 //   string save_filename(保存的文件名)
129 //   string format(同上)
130 //   int width
131 //   int height
132 //   table buffer(ppm数据部分)
133 static int
134 saveppm(lua_State *L) {
135     .....
136 }
137
138 // lua的导出接口
139 int
140 enjoy2d_ppm(lua_State *L) {
141     luaL_Reg l[] = {
142         { "texture", loadtexture },
143         { "load", loadppm },
144         { "save", saveppm },
145         { NULL, NULL },
146     };
147     luaL_newlib(L, l);
148     return 1;
149 }
```

texture纹理

这部分内容大部分来自<http://www.cnblogs.com/shengdoushi/archive/2011/01/13/1934181.html>

纹理实际上是一个二维数组, 其元素是一些颜色值, 每一元素称之为纹理像素(texel). OpenGL以一个整形id来作为句柄管理纹理对象. 通常, 一个纹理映射的步骤是:

1. 创建纹理对象, 获得一个句柄id.
2. 设定过滤, 定义了OpenGL显示图像的效果.

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.

3. 加载纹理数据，将图像数据赋值给纹理对象.
4. 绑定纹理对象.
5. 纹理映射，将绑定纹理的数据绘制到屏幕.

纹理有自己的一套坐标系 (Figure 1).

Figure 1: 纹理坐标系



纹理映射：是指将纹理绘制到屏幕的过程，这中间会有坐标系的转换 (Figure 2).

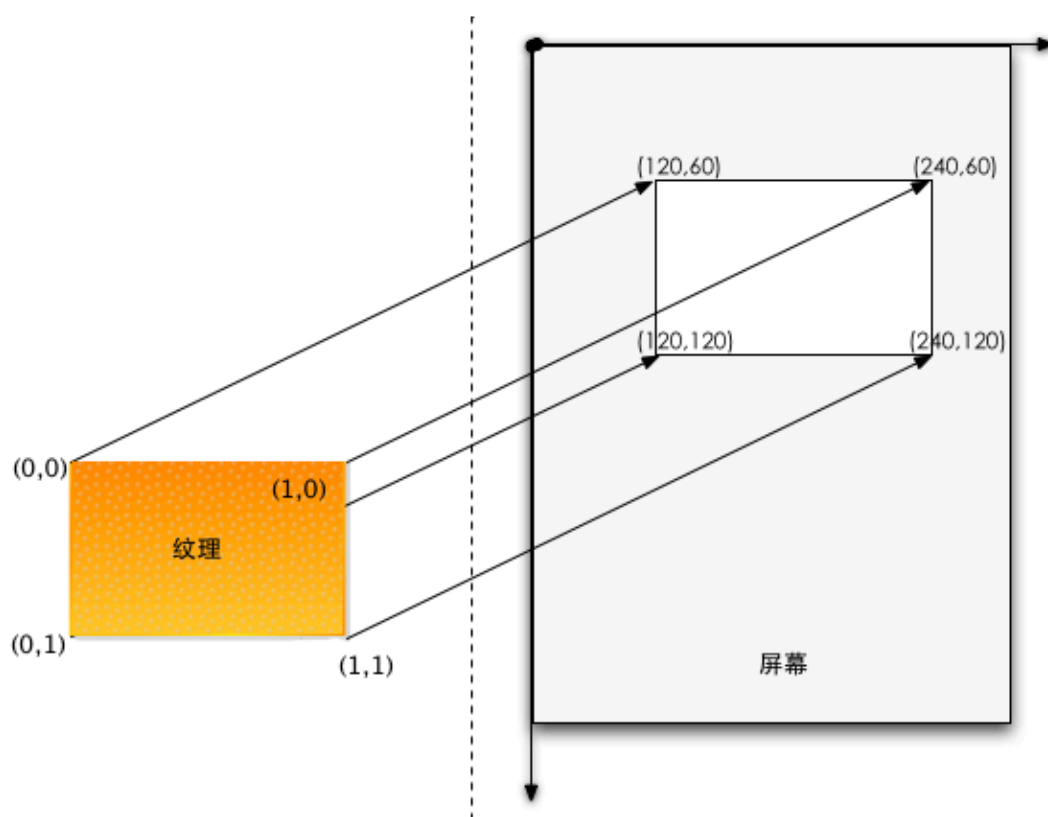
ejoy2d中的texture源码

ejoy2d中关于texture的代码大部分都在lib/texture.h和lib/texture.c中.

```
1 // 这是ejoy2d支持的纹理格式
2 #define Texture2DPixelFormat_RGBA8888 1
3 #define Texture2DPixelFormat_RGBA4444 2
4 #define Texture2DPixelFormat_PVRTC4 3
5 #define Texture2DPixelFormat_PVRTC2 4
6 #define Texture2DPixelFormat_RGB888 5
7 #define Texture2DPixelFormat_RGB565 6
8 #define Texture2DPixelFormat_A8 7
9
10 // 这是texture对象，这里的id就是OpenGL的句柄，invw和invh避免了做除法
11 struct texture {
12     int width;
```

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.

Figure 2: 纹理映射



```
13     int height;
14     float invw;
15     float invh;
16     GLuint id;
17 };
18
19 // 内存池来管理纹理对象, 默认最大128对象
20 struct texture_pool {
21     int count;
22     struct texture tex[MAX_TEXTURE];
23 };
24 static struct texture_pool POOL;
25
26 // 从数据(一般是贴图)加载纹理
27 const char *
28 texture_load(int id, int pixel_format, int pixel_width, int
29             pixel_height, void *data) {
30     .....
31
32     // OpenGL创建texture
33     glGenTextures(1, &tex->id);
34
35     // 指定当前的纹理单元
36     glActiveTexture(GL_TEXTURE0);
37
38     // 这里shader会调用glBindTexture()绑定纹理
39     shader_texture(tex->id);
40
41     // 缩小|放大过滤器, 线性滤波
42     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
43                     GL_LINEAR);
44     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
45                     GL_LINEAR);
46
47     // GL_CLAMP_TO_EDGE使得超出边缘的部分与边缘保持一致, 有个学名叫“箝位”
48     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
49                     GL_CLAMP_TO_EDGE);
50     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
51                     GL_CLAMP_TO_EDGE);
52
53     // 根据纹理格式来分别加载数据
54     switch(pixel_format) {
55         .....
56     }
57     return NULL;
58 }
59
60 // 卸载(删除)纹理对象
61 void
62 texture_unload(int id) {
```



```
57     if (id < 0 || id >= P00L.count)
58         return;
59     struct texture *tex = &P00L.tex[id];
60     if (tex->id == 0)
61         return;
62     // 从OpenGL中删除纹理对象
63     glDeleteTextures(1, &tex->id);
64     tex->id = 0;
65 }
66
67 // 纹理的坐标系转换
68 void
69 texture_coord(int id, float *x, float *y) {
70     if (id < 0 || id >= P00L.count) {
71         *x = *y = 0;
72         return;
73     }
74     struct texture *tex = &P00L.tex[id];
75     // 相当于 x/texture_x, y/texture_y, 这里的用法后面看sprite的时候再看
76     *x *= tex->invw;
77     *y *= tex->invh;
78 }
```