

## 原 SDL2源代码分析5：更新纹理（SDL\_UpdateTexture()）

2014年11月07日 01:02:07 阅读数：11649

=====

SDL源代码分析系列文章列表：

[SDL2源代码分析1：初始化（SDL\\_Init\(\)）](#)

[SDL2源代码分析2：窗口（SDL\\_Window）](#)

[SDL2源代码分析3：渲染器（SDL\\_Renderer）](#)

[SDL2源代码分析4：纹理（SDL\\_Texture）](#)

[SDL2源代码分析5：更新纹理（SDL\\_UpdateTexture\(\)）](#)

[SDL2源代码分析6：复制到渲染器（SDL\\_RenderCopy\(\)）](#)

[SDL2源代码分析7：显示（SDL\\_RenderPresent\(\)）](#)

[SDL2源代码分析8：视频显示总结](#)

=====

上一篇文章分析了SDL的创建纹理函数SDL\_CreateTexture()。这篇文章继续分析SDL的源代码。本文分析SDL更新纹理数据函数SDL\_UpdateTexture()。



SDL播放视频的代码流程如下所示。

**初始化:**

SDL\_Init(): 初始化SDL。

SDL\_CreateWindow(): 创建窗口（Window）。

SDL\_CreateRenderer(): 基于窗口创建渲染器（Render）。

SDL\_CreateTexture(): 创建纹理（Texture）。

**循环渲染数据:**

SDL\_UpdateTexture(): 设置纹理的数据。

SDL\_RenderCopy(): 纹理复制给渲染器。

SDL\_RenderPresent(): 显示。

上篇文章分析了该流程中的第4个函数SDL\_CreateTexture()。本文继续分析该流程中的第5个函数SDL\_UpdateTexture()。

## SDL\_UpdateTexture()

### 函数简介

SDL使用SDL\_UpdateTexture()设置纹理的像素数据。SDL\_UpdateTexture()的原型如下。

```
[cpp]
1. int SDLCALL SDL_UpdateTexture(SDL_Texture * texture,
2.                               const SDL_Rect * rect,
3.                               const void *pixels, int pitch);
```

参数的含义如下。

texture：目标纹理。

rect：更新像素的矩形区域。设置为NULL的时候更新整个区域。

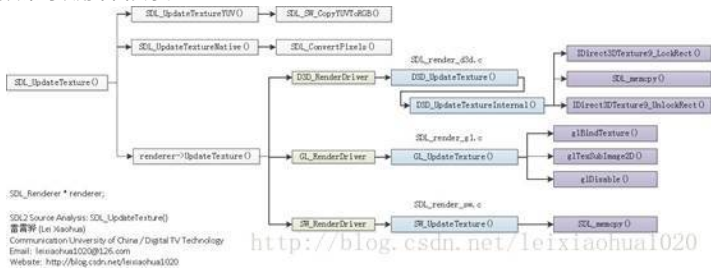
pixels：像素数据。

pitch：一行像素数据的字节数。

成功的话返回0，失败的话返回-1。

函数调用关系图

SDL\_UpdateTexture() 关键函数的调用关系可以用下图表示。



上面的图片不太清晰，更清晰的图片上传到了相册里面：

<http://my.csdn.net/leixiaohua1020/album/detail/1793769>

把相册里面的图片保存下来就可以得到清晰的图片。

源代码分析

SDL\_UpdateTexture()的源代码位于render\SDL\_render.c中。如下所示。

```
[cpp]
1.  int SDL_UpdateTexture(SDL_Texture * texture, const SDL_Rect * rect,
2.                      const void *pixels, int pitch)
3.  {
4.      SDL_Renderer *renderer;
5.      SDL_Rect full_rect;
6.
7.
8.      CHECK_TEXTURE_MAGIC(texture, -1);
9.
10.
11.     if (!pixels) {
12.         return SDL_InvalidParamError("pixels");
13.     }
14.     if (!pitch) {
15.         return SDL_InvalidParamError("pitch");
16.     }
17.
18.
19.     if (!rect) {
20.         full_rect.x = 0;
21.         full_rect.y = 0;
22.         full_rect.w = texture->w;
23.         full_rect.h = texture->h;
24.         rect = &full_rect;
25.     }
26.
27.
28.     if (texture->yuv) {
29.         return SDL_UpdateTextureYUV(texture, rect, pixels, pitch);
30.     } else if (texture->native) {
31.         return SDL_UpdateTextureNative(texture, rect, pixels, pitch);
32.     } else {
33.         renderer = texture->renderer;
34.         return renderer->UpdateTexture(renderer, texture, rect, pixels, pitch);
35.     }
36. }
```

从源代码中可以看出，SDL\_UpdateTexture()的大致流程如下。


1. 检查输入参数的合理性。例如像素格式是否支持，宽和高是否小于等于0等等。
2. 如果是一些特殊的格式，进行一定的处理：
  - a) 如果输入的像素数据是YUV格式的，则会调用SDL\_UpdateTextureYUV()进行处理。
  - b) 如果输入的像素数据的像素格式不是渲染器支持的格式，则会调用SDL\_UpdateTextureNative()进行处理。
3. 调用SDL\_Renderer的UpdateTexture()方法更新纹理。这一步是整个函数的核心。

下面我们详细看一下几种不同的渲染器的UpdateTexture ()的方法。

## 1.

### Direct3D

Direct3D 渲染器中对应UpdateTexture ()的函数是D3D\_UpdateTexture(), 它的源代码如下所示 (位于render\direct3d\SDL\_render\_d3d.c)。

```
[cpp]    
1. static int  
2. D3D_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,  
3.                  const SDL_Rect * rect, const void *pixels, int pitch)  
4. {  
5.     D3D_TextureData *data = (D3D_TextureData *) texture->driverdata;  
6.     SDL_bool full_texture = SDL_FALSE;  
7.  
8.  
9. #ifdef USE_DYNAMIC_TEXTURE  
10.    if (texture->access == SDL_TEXTUREACCESS_STREAMING &&  
11.        rect->x == 0 && rect->y == 0 &&  
12.        rect->w == texture->w && rect->h == texture->h) {  
13.        full_texture = SDL_TRUE;  
14.    }  
15. #endif  
16.  
17.  
18.    if (!data) {  
19.        SDL_SetError("Texture is not currently available");  
20.        return -1;  
21.    }  
22.  
23.  
24.    if (D3D_UpdateTextureInternal(data->texture, texture->format, full_texture, rect->x, rect->y, rect->w, rect->h, pixels, pitch) < 0) {  
25.        return -1;  
26.    }  
27.  
28.  
29.    if (data->yuv) {  
30.        /* Skip to the correct offset into the next texture */  
31.        pixels = (const void*)((const Uint8*)pixels + rect->h * pitch);  
32.  
33.  
34.        if (D3D_UpdateTextureInternal(texture->format == SDL_PIXELFORMAT_YV12 ? data->vtexture : data->utexture, texture->format, full_texture, rect->x / 2, rect->y / 2, rect->w / 2, rect->h / 2, pixels, pitch / 2) < 0) {  
35.            return -1;  
36.        }  
37.  
38.  
39.        /* Skip to the correct offset into the next texture */  
40.        pixels = (const void*)((const Uint8*)pixels + (rect->h * pitch)/4);  
41.        if (D3D_UpdateTextureInternal(texture->format == SDL_PIXELFORMAT_YV12 ? data->utexture : data->vtexture, texture->format, full_texture, rect->x / 2, rect->y / 2, rect->w / 2, rect->h / 2, pixels, pitch / 2) < 0) {  
42.            return -1;  
43.        }  
44.    }  
45.    return 0;  
46. }
```

从代码中可以看出, 该函数调用了D3D\_UpdateTextureInternal()函数。在这里需要注意, 如果输入像素格式是YUV, 就会使用3个纹理, 对于多出的那2个纹理会单独进行处理。调用的函数D3D\_UpdateTextureInternal()代码如下。

```

1. static int D3D_UpdateTextureInternal(IDirect3DTexture9 *texture, Uint32 format, SDL_bool full_texture, int x, int y, int w, int h, c
   const void *pixels, int pitch)
2. {
3.     RECT d3drect;
4.     D3DLOCKED_RECT locked;
5.     const Uint8 *src;
6.     Uint8 *dst;
7.     int row, length;
8.     HRESULT result;
9.
10.
11.     if (full_texture) {
12.         result = IDirect3DTexture9_LockRect(texture, 0, &locked, NULL, D3DLOCK_DISCARD);
13.     } else {
14.         d3drect.left = x;
15.         d3drect.right = x + w;
16.         d3drect.top = y;
17.         d3drect.bottom = y + h;
18.         result = IDirect3DTexture9_LockRect(texture, 0, &locked, &d3drect, 0);
19.     }
20.
21.
22.     if (FAILED(result)) {
23.         return D3D_SetError("LockRect()", result);
24.     }
25.
26.
27.     src = (const Uint8 *)pixels;
28.     dst = locked.pBits;
29.     length = w * SDL_BYTESPERPIXEL(format);
30.     if (length == pitch && length == locked.Pitch) {
31.         SDL_memcpy(dst, src, length*h);
32.     } else {
33.         if (length > pitch) {
34.             length = pitch;
35.         }
36.         if (length > locked.Pitch) {
37.             length = locked.Pitch;
38.         }
39.         for (row = 0; row < h; ++row) {
40.             SDL_memcpy(dst, src, length);
41.             src += pitch;
42.             dst += locked.Pitch;
43.         }
44.     }
45.     IDirect3DTexture9_UnlockRect(texture, 0);
46.
47.
48.     return 0;
49. }

```

从代码中可以看出，该函数首先调用IDirect3DTexture9\_LockRect()锁定纹理，然后使用SDL\_memcpy()将新的像素数据拷贝至纹理（SDL\_memcpy()实际上就是memcpy()），最后使用IDirect3DTexture9\_UnlockRect()解锁纹理。

## 2.

### OpenGL

OpenGL渲染器中对应UpdateTexture()的函数是GL\_UpdateTexture()，它的源代码如下所示（位于render/OpenGL/SDL\_render\_gl.c）。

```

1. static int GL_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,
2.                             const SDL_Rect * rect, const void *pixels, int pitch)
3. {
4.     GL_RenderData *renderdata = (GL_RenderData *) renderer->driverdata;
5.     GL_TextureData *data = (GL_TextureData *) texture->driverdata;
6.
7.
8.     GL_ActivateRenderer(renderer);
9.
10.
11.     renderdata->glEnable(data->type);
12.     renderdata->glBindTexture(data->type, data->texture);
13.     renderdata->glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
14.     renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH,
15.                               (pitch / SDL_BYTESPERPIXEL(texture->format)));
16.     renderdata->glTexSubImage2D(data->type, 0, rect->x, rect->y, rect->w,
17.                                rect->h, data->format, data->formattype,
18.                                pixels);
19.     if (data->yuv) {
20.         renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH, (pitch / 2));
21.
22.
23.         /* Skip to the correct offset into the next texture */
24.         pixels = (const void*)((const Uint8*)pixels + rect->h * pitch);
25.         if (texture->format == SDL_PIXELFORMAT_YV12) {
26.             renderdata->glBindTexture(data->type, data->vtexture);
27.         } else {
28.             renderdata->glBindTexture(data->type, data->utexture);
29.         }
30.         renderdata->glTexSubImage2D(data->type, 0, rect->x/2, rect->y/2,
31.                                    rect->w/2, rect->h/2,
32.                                    data->format, data->formattype, pixels);
33.
34.
35.         /* Skip to the correct offset into the next texture */
36.         pixels = (const void*)((const Uint8*)pixels + (rect->h * pitch)/4);
37.         if (texture->format == SDL_PIXELFORMAT_YV12) {
38.             renderdata->glBindTexture(data->type, data->utexture);
39.         } else {
40.             renderdata->glBindTexture(data->type, data->vtexture);
41.         }
42.         renderdata->glTexSubImage2D(data->type, 0, rect->x/2, rect->y/2,
43.                                    rect->w/2, rect->h/2,
44.                                    data->format, data->formattype, pixels);
45.     }
46.     renderdata->glDisable(data->type);
47.
48.
49.     return GL_CheckError("glTexSubImage2D()", renderer);
50. }

```

从代码中可以看出，该函数调用了OpenGL的API函数glBindTexture(), glTexSubImage2D()等更新了一个纹理。在这里有一点需要注意，如果输入像素格式是YUV，就会使用3个纹理，对于多出的那2个纹理会单独进行处理。

### 3.

#### Software

Software渲染器中对应UpdateTexture()的函数是SW\_UpdateTexture()，它的源代码如下所示（位于render\software\SDL\_render\_sw.c）。

```

1. static int SW_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,
2.                             const SDL_Rect * rect, const void *pixels, int pitch)
3. {
4.     SDL_Surface *surface = (SDL_Surface *) texture->driverdata;
5.     Uint8 *src, *dst;
6.     int row;
7.     size_t length;
8.
9.
10.     if(SDL_MUSTLOCK(surface))
11.         SDL_LockSurface(surface);
12.     src = (Uint8 *) pixels;
13.     dst = (Uint8 *) surface->pixels +
14.           rect->y * surface->pitch +
15.           rect->x * surface->format->BytesPerPixel;
16.     length = rect->w * surface->format->BytesPerPixel;
17.     for (row = 0; row < rect->h; ++row) {
18.         SDL_memcpy(dst, src, length);
19.         src += pitch;
20.         dst += surface->pitch;
21.     }
22.     if(SDL_MUSTLOCK(surface))
23.         SDL_UnlockSurface(surface);
24.     return 0;
25. }

```

该函数的源代码还没有详细分析。其中最关键的函数要数SDL\_memcpy()了，正是这个函数更新了纹理的像素数据。但是Software渲染器纹理修改的时候是否需要Lock()和Unlock()呢？这一点一直也没太搞清。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/40876089>

文章标签：[SDL](#) [OpenGL](#) [Direct3D](#) [纹理](#) [函数调用](#)

个人分类：[SDL](#)

所属专栏：[开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com