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
int SDLCALL SDL_UpdateTexture(SDL_Texture * texture,
                              const SDL_Rect * rect,
                                  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。

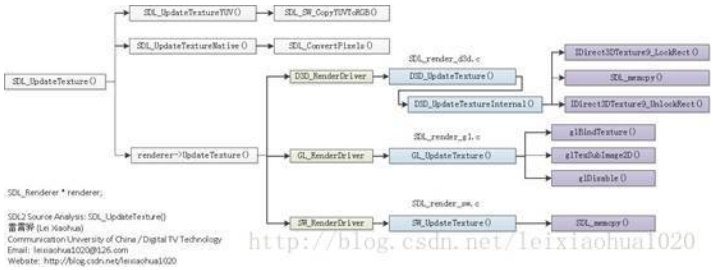
pixels：像素数据。

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
int SDL_UpdateTexture(SDL_Texture * texture, const SDL_Rect * rect,
                      const void *pixels, int pitch)
{
    SDL_Renderer *renderer;
    SDL_Rect full_rect;


    CHECK_TEXTURE_MAGIC(texture, -1);


    if (!pixels) {
        return SDL_InvalidParamError("pixels");
    }
    if (!pitch) {
        return SDL_InvalidParamError("pitch");
    }


    if (!rect) {
        full_rect.x = 0;
        full_rect.y = 0;
        full_rect.w = texture->w;
        full_rect.h = texture->h;
        rect = &full_rect;
    }


    if (texture->yuv) {
        return SDL_UpdateTextureYUV(texture, rect, pixels, pitch);
    } else if (texture->native) {
        return SDL_UpdateTextureNative(texture, rect, pixels, pitch);
    } else {
        renderer = texture->renderer;
        return renderer->UpdateTexture(renderer, texture, rect, pixels, pitch);
    }
}
```

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_Render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
static int
D3D_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,
                  const SDL_Rect * rect, const void *pixels, int pitch)
{
    D3D_TextureData *data = (D3D_TextureData *) texture->driverdata;
    SDL_bool full_texture = SDL_FALSE;


#ifdef USE_DYNAMIC_TEXTURE
    if (texture->access == SDL_TEXTUREACCESS_STREAMING &&
        rect->x == 0 && rect->y == 0 &&
        rect->w == texture->w && rect->h == texture->h) {
        full_texture = SDL_TRUE;
    }
#endif


    if (!data) {
        SDL_SetError("Texture is not currently available");
        return -1;
    }


    if (D3D_UpdateTextureInternal(data->texture, texture->format, full_texture, rect->x, rect->y, rect->w, rect->h, pixels, pitch) < 0) {
        return -1;
    }


    if (data->yuv) {
        /* Skip to the correct offset into the next texture */
        pixels = (const void*)((const Uint8*)pixels + rect->h * pitch);


        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) {
            return -1;
        }


        /* Skip to the correct offset into the next texture */
        pixels = (const void*)((const Uint8*)pixels + (rect->h * pitch)/4);
        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) {
            return -1;
        }
    }
    return 0;
}
```

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。

```cpp
static int D3D_UpdateTextureInternal(IDirect3DTexture9 *texture, Uint32 format, SDL_bool full_texture, int x, int y, int w, int h, const void *pixels, int pitch)
{
    RECT d3drect;
    D3DLOCKED_RECT locked;
    const Uint8 *src;
    Uint8 *dst;
    int row, length;
    HRESULT result;


    if (full_texture) {
        result = IDirect3DTexture9_LockRect(texture, 0, &locked, NULL, D3DLOCK_DISCARD);
    } else {
        d3drect.left = x;
        d3drect.right = x + w;
        d3drect.top = y;
        d3drect.bottom = y + h;
        result = IDirect3DTexture9_LockRect(texture, 0, &locked, &d3drect, 0);
    }


    if (FAILED(result)) {
        return D3D_SetError("LockRect()", result);
    }


    src = (const Uint8 *)pixels;
    dst = locked.pBits;
    length = w * SDL_BYTESPERPIXEL(format);
    if (length == pitch && length == locked.Pitch) {
        SDL_memcpy(dst, src, length*h);
    } else {
        if (length > pitch) {
            length = pitch;
        }
        if (length > locked.Pitch) {
            length = locked.Pitch;
        }
        for (row = 0; row < h; ++row) {
            SDL_memcpy(dst, src, length);
            src += pitch;
            dst += locked.Pitch;
        }
    }
    IDirect3DTexture9_UnlockRect(texture, 0);


    return 0;
}
```

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）。

```cpp
static int GL_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,
                const SDL_Rect * rect, const void *pixels, int pitch)
{
    GL_RenderData *renderdata = (GL_RenderData *) renderer->driverdata;
    GL_TextureData *data = (GL_TextureData *) texture->driverdata;


    GL_ActivateRenderer(renderer);


    renderdata->glEnable(data->type);
    renderdata->glBindTexture(data->type, data->texture);
    renderdata->glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH,
                              (pitch / SDL_BYTESPERPIXEL(texture->format)));
    renderdata->glTexSubImage2D(data->type, 0, rect->x, rect->y, rect->w,
                                rect->h, data->format, data->formattype,
                                pixels);
    if (data->yuv) {
        renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH, (pitch / 2));


        /* Skip to the correct offset into the next texture */
        pixels = (const void*)((const Uint8*)pixels + rect->h * pitch);
        if (texture->format == SDL_PIXELFORMAT_YV12) {
            renderdata->glBindTexture(data->type, data->vtexture);
        } else {
            renderdata->glBindTexture(data->type, data->utexture);
        }
        renderdata->glTexSubImage2D(data->type, 0, rect->x/2, rect->y/2,
                                    rect->w/2, rect->h/2,
                                    data->format, data->formattype, pixels);


        /* Skip to the correct offset into the next texture */
        pixels = (const void*)((const Uint8*)pixels + (rect->h * pitch)/4);
        if (texture->format == SDL_PIXELFORMAT_YV12) {
            renderdata->glBindTexture(data->type, data->utexture);
        } else {
            renderdata->glBindTexture(data->type, data->vtexture);
        }
        renderdata->glTexSubImage2D(data->type, 0, rect->x/2, rect->y/2,
                                    rect->w/2, rect->h/2,
                                    data->format, data->formattype, pixels);
    }
    renderdata->glDisable(data->type);


    return GL_CheckError("glTexSubImage2D()", renderer);
}
```

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）。

```cpp
static int SW_UpdateTexture(SDL_Renderer * renderer, SDL_Texture * texture,
                const SDL_Rect * rect, const void *pixels, int pitch)
{
    SDL_Surface *surface = (SDL_Surface *) texture->driverdata;
    Uint8 *src, *dst;
    int row;
    size_t length;


    if(SDL_MUSTLOCK(surface))
        SDL_LockSurface(surface);
    src = (Uint8 *) pixels;
    dst = (Uint8 *) surface->pixels +
                      rect->y * surface->pitch +
                      rect->x * surface->format->BytesPerPixel;
    length = rect->w * surface->format->BytesPerPixel;
    for (row = 0; row < rect->h; ++row) {
        SDL_memcpy(dst, src, length);
        src += pitch;
        dst += surface->pitch;
    }
    if(SDL_MUSTLOCK(surface))
        SDL_UnlockSurface(surface);
    return 0;
}
```

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。

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