原 SDL2源代码分析4：纹理（SDL_Texture）

==========

SDL源代码分析系列文章列表：

==========

上一篇文章分析了SDL中创建渲染器的函数SDL_CreateRenderer()。这篇文章继续分析SDL的源代码。本文分析SDL的纹理（SDL_Texture）。

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3个函数SDL_CreateRenderer()。本文继续分析该流程中的第4个函数SDL_CreateTexture()。

SDL_Texture

SDL_Texture结构定义了一个SDL中的纹理。如果直接使用SDL2编译好的SDK的话，是看不到SDL_Texture的内部结构的。有关它的定义在头文件中只有一行代码，如下所示。

[cpp]

```cpp
/**
 *  \brief An efficient driver-specific representation of pixel data
 */
struct SDL_Texture;
typedef struct SDL_Texture SDL_Texture;
```

在源代码工程中可以看到SDL_Texture的定义，位于render\SDL_sysrender.h文件中。它的定义如下。

```cpp
/* Define the SDL texture structure */
struct SDL_Texture
{
    const void *magic;
    Uint32 format;              /**< The pixel format of the texture */
    int access;                 /**< SDL_TextureAccess */
    int w;                      /**< The width of the texture */
    int h;                      /**< The height of the texture */
    int modMode;                /**< The texture modulation mode */
    SDL_BlendMode blendMode;    /**< The texture blend mode */
    Uint8 r, g, b, a;           /**< Texture modulation values */


    SDL_Renderer *renderer;


    /* Support for formats not supported directly by the renderer */
    SDL_Texture *native;
    SDL_SW_YUVTexture *yuv;
    void *pixels;
    int pitch;
    SDL_Rect locked_rect;


    void *driverdata;           /**< Driver specific texture representation */


    SDL_Texture *prev;
    SDL_Texture *next;
};
```

可以看出其中包含了一个“纹理”所具备的各种属性。下面来看看如何创建这个SDL_Texture。

SDL_CreateTexture()

函数简介

使用SDL_CreateTexture()基于渲染器创建一个纹理。SDL_CreateTexture()的原型如下。

```cpp
SDL_Texture * SDLCALL SDL_CreateTexture(SDL_Renderer * renderer,
                                        Uint32 format,
                                        int access, int w,
                                        int h);
```

参数的含义如下。

renderer：目标渲染器。

format
：纹理的格式。后面会详述。
access
：可以取以下值（定义位于SDL_TextureAccess中）

SDL_TEXTUREACCESS_STATIC
：变化极少
SDL_TEXTUREACCESS_STREAMING
：变化频繁
SDL_TEXTUREACCESS_TARGET
：暂时没有理解

w
：纹理的宽

h
：纹理的高

创建成功则返回纹理的ID，失败返回0。

函数调用关系图

SDL_ CreateTexture ()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/1793543

把相册里面的图片保存下来就可以得到清晰的图片了。

源代码分析

SDL_CreateTexture()的源代码位于render\SDL_render.c中。如下所示。

[cpp]

```cpp
SDL_Texture * SDL_CreateTexture(SDL_Renderer * renderer, Uint32 format, int access, int w, int h)
{
    SDL_Texture *texture;


    CHECK_RENDERER_MAGIC(renderer, NULL);


    if (!format) {
        format = renderer->info.texture_formats[0];
    }
    if (SDL_ISPIXELFORMAT_INDEXED(format)) {
        SDL_SetError("Palettized textures are not supported");
        return NULL;
    }
    if (w <= 0 || h <= 0) {
        SDL_SetError("Texture dimensions can't be 0");
        return NULL;
    }
    if ((renderer->info.max_texture_width && w > renderer->info.max_texture_width) ||
        (renderer->info.max_texture_height && h > renderer->info.max_texture_height)) {
        SDL_SetError("Texture dimensions are limited to %dx%d", renderer->info.max_texture_width, renderer->info.max_texture_height);
        return NULL;
    }
    texture = (SDL_Texture *) SDL_calloc(1, sizeof(*texture));
    if (!texture) {
        SDL_OutOfMemory();
        return NULL;
    }
    texture->magic = &texture_magic;
    texture->format = format;
    texture->access = access;
    texture->w = w;
    texture->h = h;
    texture->r = 255;
    texture->g = 255;
    texture->b = 255;
    texture->a = 255;
    texture->renderer = renderer;
    texture->next = renderer->textures;
    if (renderer->textures) {
        renderer->textures->prev = texture;
    }
    renderer->textures = texture;


    if (IsSupportedFormat(renderer, format)) {
        if (renderer->CreateTexture(renderer, texture) < 0) {
            SDL_DestroyTexture(texture);
            return 0;
        }
    } else {
        texture->native = SDL_CreateTexture(renderer,
                                GetClosestSupportedFormat(renderer, format),
                                access, w, h);
        if (!texture->native) {
            SDL_DestroyTexture(texture);
            return NULL;
        }


        /* Swap textures to have texture before texture->native in the list */
        texture->native->next = texture->next;
        if (texture->native->next) {
            texture->native->next->prev = texture->native;
        }
        texture->prev = texture->native->prev;
        if (texture->prev) {
            texture->prev->next = texture;
        }
        texture->native->prev = texture;
        texture->next = texture->native;
        renderer->textures = texture;


```

```
        if (SDL_ISPIXELFORMAT_FOURCC(texture->format)) {
            texture->yuv = SDL_SW_CreateYUVTexture(format, w, h);
            if (!texture->yuv) {
                SDL_DestroyTexture(texture);
                return NULL;
            }
        } else if (access == SDL_TEXTUREACCESS_STREAMING) {
            /* The pitch is 4 byte aligned */
            texture->pitch = (((w * SDL_BYTESPERPIXEL(format)) + 3) & ~3);
            texture->pixels = SDL_calloc(1, texture->pitch * h);
            if (!texture->pixels) {
                SDL_DestroyTexture(texture);
                return NULL;
            }
        }
    }
    return texture;
}
```

从源代码中可以看出，SDL_CreateTexture()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. **新建一个SDL_Texture。** 调用SDL_calloc()（实际上就是calloc()）为新建的SDL_Texture分配内存。
3. **调用SDL_Render的CreateTexture()方法创建纹理。** 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CreateTexture()的方法。

1. Direct3D

Direct3D 渲染器中对应CreateTexture()的函数是D3D_CreateTexture()，它的源代码如下所示（位于render\direct3d\SDL_render_d3d.c）。

```cpp
static int D3D_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
{
    D3D_RenderData *renderdata = (D3D_RenderData *) renderer->driverdata;
    D3D_TextureData *data;
    D3DPOOL pool;
    DWORD usage;
    HRESULT result;


    data = (D3D_TextureData *) SDL_calloc(1, sizeof(*data));
    if (!data) {
        return SDL_OutOfMemory();
    }
    data->scaleMode = GetScaleQuality();


    texture->driverdata = data;


#ifdef USE_DYNAMIC_TEXTURE
    if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
        pool = D3DPOOL_DEFAULT;
        usage = D3DUSAGE_DYNAMIC;
    } else
#endif
    if (texture->access == SDL_TEXTUREACCESS_TARGET) {
        /* D3DPOOL_MANAGED does not work with D3DUSAGE_RENDERTARGET */
        pool = D3DPOOL_DEFAULT;
        usage = D3DUSAGE_RENDERTARGET;
    } else {
        pool = D3DPOOL_MANAGED;
        usage = 0;
    }


    result =
        IDirect3DDevice9_CreateTexture(renderdata->device, texture->w,
                                       texture->h, 1, usage,
                                       PixelFormatToD3DFMT(texture->format),
                                       pool, &data->texture, NULL);
    if (FAILED(result)) {
        return D3D_SetError("CreateTexture()", result);
    }


    if (texture->format == SDL_PIXELFORMAT_YV12 ||
        texture->format == SDL_PIXELFORMAT_IYUV) {
        data->yuv = SDL_TRUE;


        result =
            IDirect3DDevice9_CreateTexture(renderdata->device, texture->w / 2,
                                           texture->h / 2, 1, usage,
                                           PixelFormatToD3DFMT(texture->format),
                                           pool, &data->utexture, NULL);
        if (FAILED(result)) {
            return D3D_SetError("CreateTexture()", result);
        }


        result =
            IDirect3DDevice9_CreateTexture(renderdata->device, texture->w / 2,
                                           texture->h / 2, 1, usage,
                                           PixelFormatToD3DFMT(texture->format),
                                           pool, &data->vtexture, NULL);
        if (FAILED(result)) {
            return D3D_SetError("CreateTexture()", result);
        }
    }


    return 0;
}
```

从代码中可以看出，该函数调用了Direct3D的API函数IDirect3DDevice9_CreateTexture()创建了一个纹理。

2. OpenGL

OpenGL渲染器中对应CreateTexture()的函数是GL_CreateTexture ()，它的源代码如下所示（位于render\opengl\SDL_render_gl.c）。

```cpp
static int GL_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
{
    GL_RenderData *renderdata = (GL_RenderData *) renderer->driverdata;
```

```
    GL_TextureData *data;
    GLint internalFormat;
    GLenum format, type;
    int texture_w, texture_h;
    GLenum scaleMode;


    GL_ActivateRenderer(renderer);


    if (!convert_format(renderdata, texture->format, &internalFormat,
                        &format, &type)) {
        return SDL_SetError("Texture format %s not supported by OpenGL",
                            SDL_GetPixelFormatName(texture->format));
    }


    data = (GL_TextureData *) SDL_calloc(1, sizeof(*data));
    if (!data) {
        return SDL_OutOfMemory();
    }


    if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
        size_t size;
        data->pitch = texture->w * SDL_BYTESPERPIXEL(texture->format);
        size = texture->h * data->pitch;
        if (texture->format == SDL_PIXELFORMAT_YV12 ||
            texture->format == SDL_PIXELFORMAT_IYUV) {
            /* Need to add size for the U and V planes */
            size += (2 * (texture->h * data->pitch) / 4);
        }
        data->pixels = SDL_calloc(1, size);
        if (!data->pixels) {
            SDL_free(data);
            return SDL_OutOfMemory();
        }
    }


    if (texture->access == SDL_TEXTUREACCESS_TARGET) {
        data->fbo = GL_GetFBO(renderdata, texture->w, texture->h);
    } else {
        data->fbo = NULL;
    }


    GL_CheckError("", renderer);
    renderdata->glGenTextures(1, &data->texture);
    if (GL_CheckError("glGenTexures()", renderer) < 0) {
        SDL_free(data);
        return -1;
    }
    texture->driverdata = data;


    if ((renderdata->GL_ARB_texture_rectangle_supported)
        /* && texture->access != SDL_TEXTUREACCESS_TARGET */){
        data->type = GL_TEXTURE_RECTANGLE_ARB;
        texture_w = texture->w;
        texture_h = texture->h;
        data->texw = (GLfloat) texture_w;
        data->texh = (GLfloat) texture_h;
    } else {
        data->type = GL_TEXTURE_2D;
        texture_w = power_of_2(texture->w);
        texture_h = power_of_2(texture->h);
        data->texw = (GLfloat) (texture->w) / texture_w;
        data->texh = (GLfloat) texture->h / texture_h;
    }


    data->format = format;
    data->formattype = type;
    scaleMode = GetScaleQuality();
    renderdata->glEnable(data->type);
    renderdata->glBindTexture(data->type, data->texture);
    renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER, scaleMode);
    renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER, scaleMode);
    /* According to the spec, CLAMP_TO_EDGE is the default for TEXTURE_RECTANGLE
       and setting it causes an INVALID_ENUM error in the latest NVidia drivers.
    */
    if (data->type != GL_TEXTURE_RECTANGLE_ARB) {
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
                                    GL_CLAMP_TO_EDGE);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
                                    GL_CLAMP_TO_EDGE);
    }
#ifdef __MACOSX__
#ifndef GL_TEXTURE_STORAGE_HINT_APPLE
#define GL_TEXTURE_STORAGE_HINT_APPLE       0x85BC
```

```
#endif
#ifndef STORAGE_CACHED_APPLE
#define STORAGE_CACHED_APPLE                0x85BE
#endif
#ifndef STORAGE_SHARED_APPLE
#define STORAGE_SHARED_APPLE                0x85BF
#endif
    if (texture->access == SDL_TEXTUREACCESS_STREAMING) {
        renderdata->glTexParameteri(data->type, GL_TEXTURE_STORAGE_HINT_APPLE,
                                    GL_STORAGE_SHARED_APPLE);
    } else {
        renderdata->glTexParameteri(data->type, GL_TEXTURE_STORAGE_HINT_APPLE,
                                    GL_STORAGE_CACHED_APPLE);
    }
    if (texture->access == SDL_TEXTUREACCESS_STREAMING
        && texture->format == SDL_PIXELFORMAT_ARGB8888
        && (texture->w % 8) == 0) {
        renderdata->glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_TRUE);
        renderdata->glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
        renderdata->glPixelStorei(GL_UNPACK_ROW_LENGTH,
                          (data->pitch / SDL_BYTESPERPIXEL(texture->format)));
        renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w,
                                 texture_h, 0, format, type, data->pixels);
        renderdata->glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_FALSE);
    }
    else
#endif
    {
        renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w,
                                 texture_h, 0, format, type, NULL);
    }
    renderdata->glDisable(data->type);
    if (GL_CheckError("glTexImage2D()", renderer) < 0) {
        return -1;
    }


    if (texture->format == SDL_PIXELFORMAT_YV12 ||
        texture->format == SDL_PIXELFORMAT_IYUV) {
        data->yuv = SDL_TRUE;


        renderdata->glGenTextures(1, &data->utexture);
        renderdata->glGenTextures(1, &data->vtexture);
        renderdata->glEnable(data->type);


        renderdata->glBindTexture(data->type, data->utexture);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER,
                                    scaleMode);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER,
                                    scaleMode);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
                                    GL_CLAMP_TO_EDGE);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
                                    GL_CLAMP_TO_EDGE);
        renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w/2,
                                 texture_h/2, 0, format, type, NULL);


        renderdata->glBindTexture(data->type, data->vtexture);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_MIN_FILTER,
                                    scaleMode);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_MAG_FILTER,
                                    scaleMode);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_S,
                                    GL_CLAMP_TO_EDGE);
        renderdata->glTexParameteri(data->type, GL_TEXTURE_WRAP_T,
                                    GL_CLAMP_TO_EDGE);
        renderdata->glTexImage2D(data->type, 0, internalFormat, texture_w/2,
                                 texture_h/2, 0, format, type, NULL);


        renderdata->glDisable(data->type);
    }


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

从代码中可以看出，该函数调用了OpenGL的API函数glGenTextures()，glBindTexture()创建了一个纹理。并且使用glTexParameteri()设置了有关的一些参数。在这里有一点需要注意，在OpenGL渲染器中，如果输入像素格式是YV12或者IYUV，就会使用3个纹理。

3. Software

Software渲染器中对应CreateTexture()的函数是SW_CreateTexture ()，它的源代码如下所示（位于render\software\SDL_render_sw.c）。

[cpp]

```cpp
static int SW_CreateTexture(SDL_Renderer * renderer, SDL_Texture * texture)
{
    int bpp;
    Uint32 Rmask, Gmask, Bmask, Amask;


    if (!SDL_PixelFormatEnumToMasks
        (texture->format, &bpp, &Rmask, &Gmask, &Bmask, &Amask)) {
        return SDL_SetError("Unknown texture format");
    }


    texture->driverdata =
        SDL_CreateRGBSurface(0, texture->w, texture->h, bpp, Rmask, Gmask,
                             Bmask, Amask);
    SDL_SetSurfaceColorMod(texture->driverdata, texture->r, texture->g,
                           texture->b);
    SDL_SetSurfaceAlphaMod(texture->driverdata, texture->a);
    SDL_SetSurfaceBlendMode(texture->driverdata, texture->blendMode);


    if (texture->access == SDL_TEXTUREACCESS_STATIC) {
        SDL_SetSurfaceRLE(texture->driverdata, 1);
    }


    if (!texture->driverdata) {
        return -1;
    }
    return 0;
}
```

该函数的源代码还没有详细分析。可以看出其中调用了SDL_CreateRGBSurface()创建了“Surface”。

文章标签： SDL Direct3D OpenGL 纹理 Texture

个人分类： SDL

所属专栏： 开源多媒体项目源代码分析