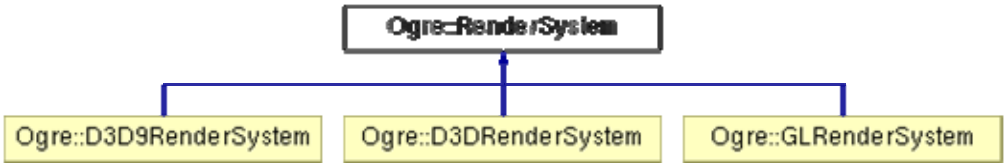
OGRE 分析之场景渲染

Mythma
http://www.cppblog.com/mythma
Email: mythma@163.com

SceneManager 负责场景的管理，而渲染则由 RenderSysem 统一管理。

一、RenderSystem

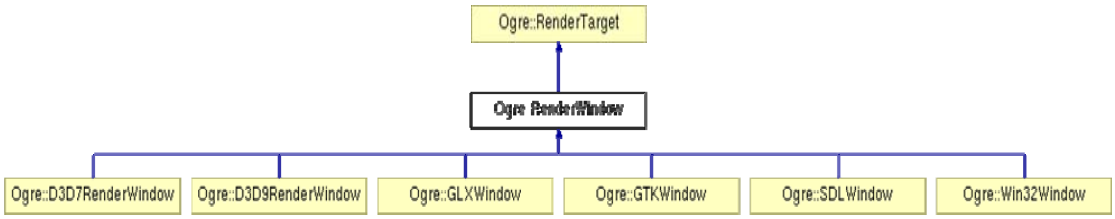
3D 图形的渲染，一般使用 Direct3D 或 OpenGL，OGRE 提供了对两者封装，并提供了统一的接口——RenderSystem。RenderSystem 本身是个虚基类，提供了与 3D API 无关的接口和操作。



除了负责渲染外，RenderSystem 还负责窗口 RenderWindow 的管理：

```
virtual RenderWindow* initialise(bool autoCreateWindow,
                                 const String& windowTitle = "OGRE Render Window");
virtual RenderWindow* createRenderWindow(const String &name, unsigned int width,
                                         unsigned int height,  bool fullScreen,
                                         const NameValuePairList *miscParams = 0) = 0;
virtual void destroyRenderWindow(const String& name);
```

RenderWindow 是属于 RenderTarget 的子类，并根据不同的平台实现不同的 xRenderWindow：



从 RenderSystem 的数据成员可以发现，各种 RenderTarget 都是由 RenderSystem 统一管理的：

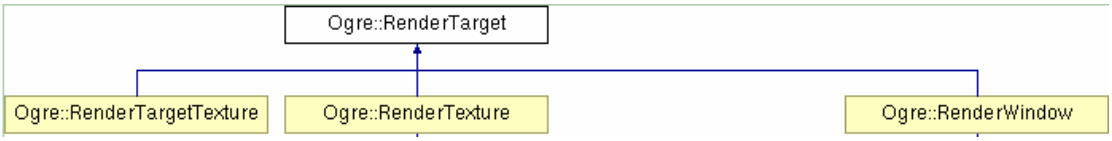
```
typedef std::map< String, RenderTarget * > RenderTargetMap;
typedef std::multimap<uchar, RenderTarget * > RenderTargetPriorityMap;

  RenderTargetMap mRenderTargets;
  RenderTargetPriorityMap mPrioritisedRenderTargets;
  RenderTarget * mActiveRenderTarget;
  TextureManager* mTextureManager;
```

其中，mTextureManager 是由具体的 RenderSystem 创建的。

二、RenderTarget

RenderTarget 用来接收渲染操作的结果，它可以是屏幕上的窗口、离屏面（如 texture）等：



FPS 信息的统计也是由 RenderTarget 完成的。在 RenderTarget 每次更新完成后，将会更新统计信息（封装于 FrameStats 中）。

除了负责统计帧的信息外，RenderTarget 还负责创建维护 Viewport（视口）：

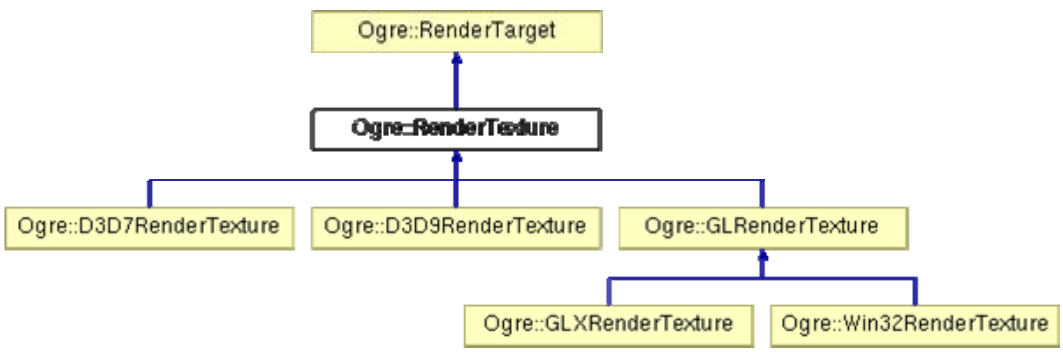
```
typedef std::map<int, Viewport*, std::less<int> > ViewportList;
    /// List of viewports, map on Z-order
ViewportList mViewportList;

    Viewport* RenderTarget::addViewport(Camera* cam, int ZOrder,
                        float left, float top ,float width , float height)
    {
        // Check no existing viewport with this Z-order
        ViewportList::iterator it = mViewportList.find(ZOrder);
        if (it != mViewportList.end())
        {   //.................
        }
        // Add viewport to list . Order based on Z-Order
        Viewport* vp = new Viewport(cam, this, left, top, width, height, ZOrder);
        mViewportList.insert(ViewportList::value_type(ZOrder, vp));
        return vp;
    }
```

由上面的代码可以看出，每个 Viewport 都对应一个 Camera 和一个 RenderTarget。当创建一个 Viewport 后，它会自动建立与 Camera 的联系。可以把 Camera 看作是图像的来源，而 RenderTarget 是图像渲染的目的地。

一个 Viewport 只能对应一个 Camera 和一个 RenderTarget，而一个 Camera 也只能对应一个 Viewport，但 RenderTarget 却可以拥有几个 Viewport。

顺便看一下 RenderTexture，也是不同的平台有不同的实现：



三、渲染过程

从 OGRE 的例子中我们可以发现，在初始化完成之后，OGRE 通过调用 startRendering 进行消息循环，然后调用 renderOneFrame，通过 RenderSystem 的 _updateAllRenderTargets 方法，更新所有的 RenderTarget。RenderTarget 通过 update 方法更新与之关联的 Viewport 并产生 FPS 统计信息。而 Viewport 则调用与之关联的 Camera 的 _renderScene 方法进行渲染，Camera 此时把“球”踢给 SceneManager。进入 SceneManager 的 renderScene 成员函数中后，在经过“漫长”的计算后，把需要渲染的场景送给 RenderSystem 去做真正的渲染（至于怎么渲染暂且放过），此时我们可以看到熟悉的_breginFrame 和_endFrame。

整个过程（顺序图更好一点，抽空再画吧）：

Root →RenderSystem → RenderTarget → Viewport → Camera → SceneManager →RenderSystem

以上这几部分几乎是 OGRE 最最基本的构件。