1、OSGEARTH 安装

OsgEarth 是一个跨平台的库，可以使用版本号为 2.6.2 或更新版本的 Cmake 对其进行编译。

1.1、获取源码

三个方法可以获取 osgearth 源码：

Option1：使用 GIT

osgEarth 源码托管于 GitHub，所以我们需要一个 Git 客户端。Windows 下推荐 TortoiseGit，SVN 小乌龟的 Git 版本。

评价：该有的功能基本都有了，还是不错的。

备注：

GitHub 是一个为那些使用 Git 版本控制系统的项目提供基于互联网的存取服务的 Git 存取站点。它是由 GItHub 公司（先前被称作用 Logical Awesome）的开发者 Chris Wanstrath, PJ Hyett, 和 Tom Preston-Werner 使用 Ruby on Rails 写成的。最新版本是 Version 2.0，*osgearth_2.0_T2011-02-24*。

Option2：下载 tarball

tarball 是 linux 下最方便的打包工具，是以 tar 这个指令来打包与压缩的档案。下载页面: http://github.com/gwaldron/osgearth/downloads

Option3:使用版本控制 SVN

在您的 SVN 客户端上键入：http://svn.github.com/gwaldron/osgearth.git

1.2、准备工作

osgEarth 需要第三方支持库才能编译。

所需的依赖服务如下：

- OpenSceneGraph 2.8 或更新的
- GDAL 1.6 或更新的，Geospatial Data Abstraction Layer 的缩写。是一个在 X/MIT 许可协议下的开源栅格空间数据转换库
- CURL- HTTP transfer library，是一个利用 URL 语法在命令行方式下工作的文件传输工具。

可选依赖如下（没有他们，osgEarth 可以运行，但将会失去一些功能）：

- GEOS3.2.0 或更高版本-C++拓展库，这个库是用来 osgEarthFeatures 模块执行各种几何处理操作，如缓冲和交集 。如果您计划显示矢量/功能 osgEarth 数据，就需要这个依赖。
- SQLite3.6 或更高版本-嵌入式关系数据库引擎。是一款轻型的数据库，遵守 ACID 的关联式数据库管理系统。
- LibZIP-读取，创建和修改 ZIP 的 C 库。OsgEarth 有一个实验性的压缩文件需要用到这个库。

1.3、编译 osgEarth

OsgEarth 使用 Cmake 生成系统，版本可以说 2.6.2 或更新的。

备注：

- 对于可选依赖（像 GEOS），您可以舍弃或者不选。
- 只要输入 OSG_DIR 值，在 Cmake 中生成，Cmake 将会自动找到其他 OSG 的文件夹。
- 保证“DYNAMIC_OSGEARTH”标识是“ON”的，建立 osgEarth 这种静态库还没有在所有平台上都能顺利完成的。
- 有时 Cmake 可能无法运行 osgversion，在大多数情况下，您可以放心的忽略这个问题。

1.4、测试

在命令行输入：osgviewer sample.earth 进行简单测试

Sample.earth

基本故障：

- 最普遍的问题是共享库的路径配置问题

路径应包括：

OSG 和 osgEarth 库

其他 osgEarth 所依赖的工具

OSG 的第三方支持（特别是 zlib 和 libpng）

- 如果提示丢失 CURL plugin:

  由于在 OSGCMake 配置管理时，CURL 是可选的。确定你启用并已经生成了 CURL。

- GDAL 驱动无法正常工作

  确定你的 PATH 包含了 GDAL 共享库

  设置环境变量 GDAL_DATA 指向包含了 GDAL's.csv 文件的文件夹

2、建立地图

（即：如何用以.earth 为后缀的 XML 文件定义自己的地图）

osgEarth 使用一个基于 XML 的文件格式，被称为 Earth File 的文件来说明数据是怎样加载进 OSG 的。当您创建一个 Earth File 时，对可用的功能想详细了解，请参阅“Earth File 元素索引”。

Earth File 的核心作用是指明以下 3 点：

- 你创建的地图类型（geocentric 或 projected）
- 可使用的图像、三面图 elevation、矢量和模型数据
- 你的数据缓存在哪里

这是 Earth File 文件库，包含大量 Earth files 并告诉你如何使用它们：https://github.com/gwaldron/osgearth/tree/master/tests/

2.1 地图文件元素索引

2.1.1.简单图像文件

这是一个很简单的例子，从 WMS 服务器读取数据，并渲染在一个圆形地球的三维模型上。

```
  <map name="MyMap" type="geocentric" version="2">
    <image name="bluemarble" driver="gdal">
        <url>/data/world.tif</url>
    </image>
</map>
```

这个文件建立了一个地图“MyMap”，geocentric 类型，GeoTIFF 图片源名称是“bluemarble”（GeoTiff 是包含地理信息的一种 Tiff 格式的文件）。驱动 driver 属性告诉 osgearth 哪个驱动去加载这些图片，所有子元素针对特定的驱动。

2.1.2.多重图像层

osgEarth 支持有多个图像源的地图。这允许你创建的地图时，在基础层上覆盖高分辨率的插图，当然了基础图层是一个较低分辨率的底图。要添加多个图像到 Earth File，只需添加多个“image”元素到你的 Earth file。

```
<map name="Transportation" type="geocentric" version="2">
    <!--Add a base map of the blue marble data-->
    <image name="bluemarble" driver="gdal">
```

```
        <url>c:/data/bluemarble.tif</url>
    </image>

    <!--Add a high resolution inset of Washington, DC-->
    <image name="dc" driver="gdal">
        <url>c:/data/dc_high_res.tif</url>
    </image>
</map>
```

上述地图使用本地数据源（使用 GDAL 插件）提供的两个图像。osgEarth 使用各种方法来渲染图像层，所以可以渲染多少图像层的限制取决于您的硬件。顺序是很重要的，定义多个图像源时，它们在该 earth 文件中指定的顺序是从在底部到顶部的。

2.2 高程数据

添加高程数据到地球的文件与添加图像非常相似。高程数据可以通过将**高程**元素加入到 XML 从而添加到地球文件中去。

```
<map name="Elevation" type="geocentric" version="2">
    <!--Add a base map of the blue marble data-->
    <image name="bluemarble" driver="gdal">
        <url>c:/data/bluemarble.tif</url>
    </image>

    <!--Add SRTM data-->
    <elevation name="srtm" driver="gdal">
        <url>c:/data/SRTM.tif</url>
    </elevation>
</map>
```

这个 earth file 有基础层 image：bluemarble，以及一个从本地加载 的 GEOTIFF 文件作为高程网格。Earth 文件可以添加任意的高程数据，他们将通过 osgEarth 结合在一起 。图像，顺序是很重要的高程数据以及的 。例如，如果你有一个基地海拔的数据源，低分辨率覆盖整个世界和一个高分辨率的科罗拉多州丹佛市的插图，您将指定的基础数据的第一， 高分辨率数据。

多数驱动在 osgEarth 支持阅读 heightfields 以及图像。但是，重要的是要注意，只有 16 位和 32 位数据源可以作为 heightfield 数据源使用。

3、.Earth 文件索引

Osgearth<**map**>

Osgearth 能够识别或者读取 2 种方式的地图：

- **Geocentric**： 如果地图类型属性是地心的 geocentric,圆形的 round, 球体的 globe 或者是地球 earth，那么这个地图就是以一个完整地球形状的。请注意只有以地心为坐标的墨卡托模型才能被呈现成一个地心坐标的球体。
- **Projected**：如果地图类型属性是 Projected 或者是平面（flat）的,那么这个地图就是一个平面投影的样子，这个类型的 map 在投影坐标系统中是非常有用的，比如 UTM。当然了墨卡托模型和整个地球也是可以用这种类型的 map 来表现的，比如 2D 制图应用程序。

属性：

name	可读的地图名称	可选
type	读取地图类型： *geocentric* for round-earth (默认), *projected* for flat-earth (默认: geocentric)	可选
version	制定地球文件格式的版本. 使用 osgEarth 2.x. 那版本号就是“2”	必须

子元素：

<options>	配置地图和所有的地形引擎的运行时行为。	optional
<image>	图像图层。随着 osgearth 的合成器 compositor 使用，你的电脑图像显示硬件将决定<image>层可以显示的最大数量 。	optional
<elevation>	高层图层. 不同大小和分辨率的高层图层可以堆叠。Osgearth 对于高层图层的数量没有仿真限制	optional
<model>	一个模型的数据源（例如，功能，数据，外部模型）	optional
<overlay>	地形褶状几何层	optional

例子：

```
<map name="earth" type="geocentric" version="2">
   <options>
      ...
   </options>
   <image ...>
   </image>
   <elevation...>
   </elevation>
</map>
```

<Compositor>使用方法

<compositor>元素（下图：选项：地形）

此元素控制的方法，osgEarth 使用复合图像层。

值：

Auto	这种模式是默认的。osgEarth 会自动选择一个基于图形硬件的构图方法。它将测试硬件的能力，以支持以下技术，在外观下面的顺序，第一个支持的技术和解决。
multitexture	将每个图像层自身的纹理图像单元，GPU 上的复合材料层。允许的图像层的最大数量是有限的纹理图像，通过你的 GPU 硬件支持的单位的数量。（注：现代 GPU 暴露更多的纹理图像时使用片段着色器，比他们在固定功能管线单位。）
texture_array	复合材料在 GPU 上的图像层，使用 Texture2DArray 构造。这项技术有效地为您提供了无限的图像层。它需要的图形硬件支持 GL_TEXTURE_2D_ARRAY_EXT 扩展和 GL_EXT_gpu_shader4 支持。
multipass	复合材料由多个渲染的图像，通过对场景图。这种技术对图像层，你可以有数量没有限制，但可能会影响性能，因为每个额外的层意味着另一个渲染传递的场景图。

示例：

```
<map>
   <options>
      <terrain>
         <compositor>multitexture</compositor>
      </terrain>
      ...
   </options>
   ...
</map>
```

4、Driver

4.1 图像/高程图层驱动器

这些驱动程序生成那些构成地形的图片或者高程数据。

AGGLite	将数据栅格化成若干图像片
ArcGIS	从 ESRI ArcGIS 地图服务系统或者网上 ArcGIS 服务读取图像数据
Composite	融合多层<image>层使之成为一个本地图像层
GDAL	支持大量格式读取地理参考的影像和高程数据包括 GeoTIFF
OSG	使用 OpenSceneGraph 图片 reader/writers 来读取那些非地理参考的图像比如 jpeg 格式的图片
TileCache	Reads image tiles from a MetaCarta TileCache cache folder
TileService	从 NASA WorldWind TileService 库读取图像片
TMS	从 OSGeo TileMapService 库读取图像片
WCS	从 OGC Web Coverage Service 读取图像片
WMS	从 OGC Web Mapping Service 读取图像片
VPB	融合 VirtualPlanetBuilder 地形

4.2 模型驱动 Model Drivers

模型驱动程序采集数据，并创建地形上可以放置的节点。这些可能是”褶皱”层，也可以是三维模型。

Feature Geometry	OSG 的几何渲染矢量数据
Feature Stencil	使用模板缓冲技术褶皱化地形矢量数据
Simple	加载外部模型，并放置在场景图

4.2.1.Feature Geometry model driver

建立矢量特征数据的 OSG 几何形状

如今，这个驱动程序可以简单的将矢量数据嵌合进几何图形中，同时会有一个可选择的高度偏移（这样你可以将几何图形放置于地形上），将来它将支持足迹挤压，纹理化和其他一些功能。

结构

<features>	描述要渲染的特点	必须
<style>	定义矢量数据的整体外观	可选
<class>	定义一个或多个要素类的功能分类	可选
<max_triangle_size>	指定一个三角形边的最大长度，角度。仅适用于 geocentric 型地图。 可控制几何图形的细分曲面，使大型的形状可以更好地符合的地球椭球的形状	可选 默认 ="5.0"
<feature_name>	每个功能的表达式解析。名称在 osg::Geometry 中设置	

例子

Feature_geom.earth

```
<map name="Feature Geometry Demo" type="geocentric">


    <image name="world" driver="gdal">
        <url>../data/world.tif</url>
    </image>


    <model name="states" driver="feature_geom">


        <!-- Configure the OGR feature driver to read the shapefile -->
        <features name="states" driver="ogr">
            <url>../data/usa.shp</url>
        </features>


        <!-- Sets the name property on each geometry to the value of the
"name" attribute -->
        <feature_name>[name]</feature_name>


        <!-- Appearance details -->
        <style type="text/css">
            states {
               stroke: #ffff00;
               stroke-width: 1.0;
```

```
            altitude-offset: 1000;
        }
    </style>
        </model>

  <lighting>false</lighting>
</map>
```

4.2.2.Feature Stencil model driver 特征模板模型驱动

使用模板缓冲技术褶皱化地形矢量数据,技术根据：Schneider和Klein提出的方法《Efficient and Accurate Rendering of Vector Data on Virtual Landscapes》

结构

此驱动程序支持以下子元素

<features>	描述要渲染的特点	必须
<style>	定义矢量数据的整体外观	可选
<class>	定义一个或多个要素类的功能分类	可选
<extrusion_distance>	设定模板卷在各个方向上的距离。如果你有简单的，跨度大的地区 (如整个美国的周边)，你可能需要增加这个数字，以避免看到 Z-缓冲干扰的发生	可选，默认=“300000”

<Features>元素

定义了一个矢量数据源，渲染矢量特征数据的模型层要求一个特征源

属性

name	可读的特征数据源名称。如果您使用全局样式设置所要渲染的特征数据的外观，那么这个名字，样式将要使用到	必选
driver	读取特征数据的驱动插件名称。	必选

结构

3 个子元素

<url>	数据源的位置（URL 或者路径名）	required
<layer>	要读取的数据层名称， 数据源支持多层时，才是必须选择的.	optional
<buffer>	适用于几何缓冲操作传入的几何形状（dialation or erosion of a shape），缓冲总是转换成多边形的数据	optional

例子

```
<map name="Feature Stencil Demo" type="geocentric">

    <image name="world" driver="gdal">
        <url>../data/world.tif</url>
    </image>

    <model name="countries" driver="feature_stencil">

        <!-- Configure the OGR feature driver to read the shapefile. -->
```

```
        <features name="world" driver="ogr">
            <url>../data/world.shp</url>
            <ogr_driver>ESRI Shapefile</ogr_driver>
            <buffer distance="0.02"/>
        </features>

        <geometry_type>line</geometry_type>

        <style>
            world {
                stroke: #ff7f00;
            }
        </style>

    </model>

</map>
```

<Style>元素

影响所要呈现的几何外形，比如颜色、透明度、纹理等等

Style 格式

Osgearth 支持 2 种方法定义样式：CSS 和 SLD

1、 Css 样式：

Css 样式是基于 OGC SLD 样式规格的。Eg,如下：定义线条颜色为黄色，2 个单位粗，50%的透明度

```
<style type="text/css">
    element {
        stroke: #FF0000;
        stroke-width: 2.0;
        stroke-opacity: 0.5;
    }
</style>
```

多边形需要用到"fill"这个 css 参数

```
<style type="text/css">
    element {
        fill: #FF0000;
        fill-opacity: 0.5;
    }
</style>
```

2、SLD/XML 样式

SLD / XML 是一个冗长的 XML 的样式特征数据的描述。这种支持是仍在开

发中。

造型方法：Styling Approaches

当您要为一个模型层做样式描述时，你可以

1）整个层用一种样式（一般方法）-> General styling

2）你可以将特征数据分解成很多“类”，然后单独描述-> Styling by feature class

相应举例：

General styling

只要包含一个<style>块，这个块是参考特征源 Example 的

```
<model ...>
    <features name="world" driver="ogr">
        <url>data/world.shp</url>
        <ogr_driver>ESRI Shapefile</ogr_driver>
    </features>

    <style type="text/css">
        world {
            fill: #FF0000;
            fill-opacity: 0.5;
        }
    </style>
</model>
```

注意 CSS 标签”world”必须跟你所要 styling 的特征源名称相匹配。为什么这样做呢？

因为你还可以包含外部的.css 文件，同时你还可以有其他的样式表。内

联.earth 文件只是一个捷径。

Styling by feature class

如果特征源支持它（请看 OGR driver）,你可以将特征分解成多个目录，对于不同子特征单独进行样式描述，这是一个对矢量数据符号化的很好应用。

下面例子法语国家黄色表示，非法语国家红色表示：

```
<model ...>
       <features name="world" driver="ogr">
           <url>data/world.shp</url>
           <ogr_driver>ESRI Shapefile</ogr_driver>
       </features>

       <class name="french-speaking">
           <query>
               <expr> french="true" </expr>
           </query>
           <style type="text/css">
               world {
                   fill: #FFFF00;
                   fill-opacity: 0.5;
               }
           </style>
       </class>

       <class name="non-french-speaking">
```

```
        <query>
            <expr> french="false" </expr>
        </query>
        <style type="text/css">
            world {
                fill: #FF0000;
                fill-opacity: 0.5;
            }
        </style>
    </class>
</model>
```

备注：<expr>标签是一个 SQL 表达式，对得到的特征数据源进行评估，在这个例子中，我们只是简单测试了一个特征属性的值并根据这个值对数据进行分类

键入回购样品 feature_stencil_polygon_draping.earth，看到它在行动。

<Class> 元素

定义一个特征类，该特征类将一个数据集分解为基于过滤器查询的目录，不同风格不同归类。

用法

使用一个特征类去获取一个子集的特征

```
<model ...>
    <features name="world" driver="ogr">
        ...
    </features>
```

```
        <!-- Only render countries that are members of NATO: -->
        <class name="class1">
            <query>
                <expr> nato="true" </expr>
            </query>
            <style type="text/css">
                 world { fill: #66ff66; }
            </style>
        </class>
    </model>
```

或者，您可以使用多种功能类，以不同的方式呈现不同的特性和功能。请参阅<STYLE>子元素

例子

```
<map name="Feature Stencil Demo" type="geocentric">

    <image name="world" driver="gdal">
        <url>../data/world.tif</url>
    </image>

    <model name="states" driver="feature_stencil">

        <!-- Configure the OGR feature driver to read the shapefile. -->

        <features name="world" driver="ogr">
            <url>../data/world.shp</url>
            <ogr_driver>ESRI Shapefile</ogr_driver>
        </features>

        <!-- Since some countries span large areas on the globe, we need
to
             use a larger-than-normal extrusion distance on the stencil
             volumes. (300000 is the default.) -->


        <extrusion_distance>400000</extrusion_distance>
```

```
        <style type="text/css">
            world {
               fill: #ffff80;
               fill-opacity: 0.4;
            }
        </style>

    </model>

</map>
```

开发人员备注

这个驱动程序使用 OpenGL 模板缓冲。如果你采用从代码使用该驱动，请保证为 OSG viewer 分配了有足够的模板位。

如果您正在使用 *osgviewer*，你是否得到模板位因平台而异。您可以像这样强制分配：

```
osgviewer --stencil file.earth
```

如果您采用从代码使用该驱动，请确保创建 viewer 之前，分配了足够的模板位：

```
    osg::DisplaySettings::instance()->setMinimumNumStencilBits( 8 );
    ...
    osgViewer::Viewer viewer(args);
```

4.2.3 Simple 模型驱动

目前对于已加载的模型没有任何转换，也 没有地形夹紧，这些功能可能会在即将到来的“点模型替代”（point model substitution）驱动中实现。如果你想

使用这个驱动器，可以暂时使用 OSG 的.trans/.rot/.scale

结构

<url>	要加载的外部模型所在位置	required

例子

```
<map name="Simple model demo">

    <image name="world" driver="gdal">
        <url>../data/world.tif</url>
    </image>
    <model name="model" driver="simple">
        <url>something.ive</url>
    </model>
</map>
```

4.3 特征驱动 Feature Drivers

特征驱动程序是用来读取原始的 GIS 矢量数据的。他们不直接创造可见的几何图形;显示矢量数据模型的驱动程序将使用功能的驱动程序读取矢量数据。

OGR	使用 GDAL/OGR 工具包从 Shapefiles 读取原始矢量功能数据，也支持其它大量的格式

OGR Fearth Driver

这个特征驱动使用 GDAL/OGR 矢量库来读取特征（矢量数据的属性）

用法：

```
<features name="world" driver="ogr">
    <url>../data/world.shp</url>
    <ogr_driver>ESRI Shapefile</ogr_driver>
</features>
```

结构

<url>	数据源位置	optional
<ogr_driver>	OGR 驱动程序代码中指定的OSG驱动 (默认="ESRI Shapefile")	optional
<geometry>	可以用 WKT 格式指定内联几何图形来代替 url	optional

请参考<features>了解对于所有特征驱动都适用的额外的配置元素

矢量数据格式

《OGR Vector Formats》列出的都是理论上支持的格式，一些格式是具有固定的’one-layer” 像 ESRI shapefile，因此不需要使用一个<layer>元素，其

它的要求从数据源 call out 一个<layer>层

Ogr_driver 指定使用哪一个

查询支持

OGR 驱动支持在 feature classes 中使用 SQL 查询，对于大多数 OGR 格式，你可以使用<OGR SQL>,如果底层的 OGR 驱动程序有其自身的原生查询的方言（如 Oracle 和 PostgreSQL 等）,SQL 将通过在该方言处理

可以看<query>获取更多信息

内联 WKT 几何

可以选择性的指定你的几何图形在.earth 文件中的位置，像这样：

```
<features name="data" driver="ogr">
  <geometry>
    POLYGON((-71.096 42.332, -71.096 42.386, -70.988 42.386, -70.988 42.332))
  </geometry>
</features>
```

内联 WKT 字符串必须符合这个规定的格式

http://en.wikipedia.org/wiki/Well-known_text#Geometric_objects

4.4 缓存驱动 Cache Drivers

缓存驱动器是用来为 osgearth 提供高速缓存实现的

tms	在硬盘上存储 TMS 层次结构文件的缓存切片 (默认)
sqlite3	硬盘上的一个 sqlite3 数据库的缓存切片

4.4.1 TMS

用法

```
<cache type="tms">
    <path>c:/osgearth_cache</path>
</cache>
```

结构

<path>	存储缓存切片的文件夹路径	**必选**

相关网站：http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification

4.4.2 sqlite3

Sqlite3 缓存驱动

用法

```
<cache type="sqlite3">
      <path>osgearth.cachedb</path>
 </cache>
```

结构

<path>	Sqlite3 数据库的安装路径	必选
<max_size>	数据库的近似最大值（单位 MB） 设置为 0：没有最大值（默认是：100）	可选
<async_writes>	Bool 值，是否支持数据库在后台线程上执行写操作（默认：true）	可选

Sqlite 官网：http://www.sqlite.org/

4.5 地形引擎驱动 Terrain Engine Drivers

提供一个渲染实现的地形皮肤（图片+高程数据）

osgterrain	基于 OSG 的 osgTerrain 库地形渲染。（默认）

5、处理数据源

如果你想在 osgearth 中查看地理数据，你可以使用 GDAL 驱动程序。你一旦选择用 GDAL，以下一些性能指南该遵守：

Reproject your data 重建数据（栅格空间数据转换）

如果你的数据没有必需的坐标系统，osgearth 将重建数据。

有无正确坐标系统的重要性？举个例子，在大地测量中查看一个 UTM 图像（epsg:4326），如果你的数据已经在正确的坐标系统中，osgearth 会运行得更快。你可以使用任何可以的工具去重建数据比如 GDAL，Global Mapper 或者 ArcGIS。

下面是使用 gdal_warp 重建大地测量中 UTM 图像的一个例子：

```
gdalwarp -t_srs epsg:4326 my_utm_image.tif my_gd_image.tif
```

5.1 Build tile imagery 建立平铺图像

典型的格式如 GeoTiff 以扫描线方式存储它们的像素数据，这种方式一般表现得很好，但是随着在 osgearth 中平铺方法（tiled approach）访问数据，你会发现使用一个平铺化的数据集将会更有效率，因为 osgearth 提取一个 tile 不需要从磁盘读取很大的数据。

使用 gdal_translate 建立一个瓦片化的 GeoTiff：

```
gdal_translate -of GTiff -co "TILED=YES" myfile.tif
myfile_tiled.tif
```

Build overviews 建立概述

添加概述（也可以叫金字塔或 rsets）将极大的增加 osgearth 数据源的表现力。你可以使用 gdaladdo 这个实用工具向一个数据集添加概述：

```
gdaladdo -r average myimage.tif 2 4 8 16
```

5.2 生成平铺化的数据集

当在 osgearth 中直接使用地理空间数据成为可能，将数据预先平铺化成一个平铺优化层次的数据将会更加有效。

Osgearth

一个创建好的平铺化的数据源的最好方法是使用 osgearth 内置缓存机制。Osgearth 创建的缓存是基于开放标准的，比如 TMS.一旦它们被缓存，就可以直接作为一个数据源。举个例子，我们要平铺化一个 NASA BlueMarble 图片的 GeoTiff 文件

首先，建立一个 bluemarble.earth 地图文件（使用 GDAL 插件）

```
<map name="bluemarble" type="geocentric" version="2">


    <!--Add a reference to the image -->
    <image name="bluemarble" driver="gdal">
        <url>c:/data/bluemarble.tif</url>
    </image>


    <options>
        <!--Tell osgEarth to cache the tiles in a TMS format-->
        <cache type="tms">
            <path>c:/osgearth_cache</path>
            <!--Tell osgEarth to cache the tiles to JPG to save
disk space-->
            <format>jpg</format>
        </cache>
    </options>
```

```
</map>
```

当你输入

```
osgviewer bluemarble.earth
```

你会看到 tile 开始缓存到 c:/osgearth_cache/bluemarble，我们可以使用 osgearth_seed program 让缓存进程自动化。

运行：

```
osgearth_seed --max-level 7 bluemarble.earth
```

一旦编译，你将有一个完全缓存的 TMS 数据源，在目录 c:/osgearth_cache/bluemarble。

现在，让我们删除原来需要的 GeoTiff 文件，直接访问 TMS 数据源，创建一个地图文件 bluemarble_tms.earth

```
<map name="bluemarble-tms" type="geocentric" version="2">
    <!--Add a reference to the TMS xml file -->
    <image name="bluemarble" driver="tms">
        <url>c:/osgearth_cache/bluemarble/tms.xml</url>
    </image>
</map>
```

直接运行 osgviewer 将直接命中缓存，您可以自由移动的 bluemarble 数据源到你想要的任何地方，甚至一个网络服务器！你可以通过复制 bluemarble 目录到你网络服务器上的一个公共文件夹实现调配 tiles.

之后，你应该改变你的 bluemarble_tms.earth 文件：

```
<map name="bluemarble-tms" type="geocentric" version="2">
    <!--Add a reference to the TMS xml file -->
    <image name="bluemarble" driver="tms">

<url>http://www.myserver.com/tiles/bluemarble/tms.xml</url>
    </image>
</map>
```

现在运行 osgviewer 将命中你服务器上的 tiles,为了完整，你也可以复制 bluemarble_tms.earth
到你的网络服务器然后使用时从网上下载它：

```
osgviewer http://www.myserver.com/maps/bluemarble_tms.earth
```

重要的是要注意，此过程不会只适用于您的源计算机上的地理空间数据。这是完全有可能使用此过程中，如 WMS 服务器，将数据缓存到本地缓存 TMS！

Maptiler

MapTiler 是一个小巧的 GUI tile 生成器，最初是作为 GDAL 脚本 gdal2tiles 开发出来的。可以生成被 osgearth 目录使用的 TMS tile 子集。
重要注意点：

MapTiler 输出文件 tilemapresource.xml 中的 BoundingBox 元素有 x 和 y 的反转，所以一旦 tile 生成完成，用记事本打开 tilemapresource.xml, 交换

minx/miny,maxx/maxy 的值然后保存。

然后您应该能够创建一个新的[TileSourcePluginTMS TMS 的图像源引用的 tilemapresource.xml 文件

TileCache

TileCache 是一个 Python WMS-C 实现，可以容易发布缓存 tile 服务器，一旦你根据官网的指南安装了 TileCache,你可以，你可以设置 osgearth 去访问作为一个 WMS 或者 TMS 服务器的 TileCahce，当你浏览时 TileCahe 将缓存这些 tiles。还有一个好处是，你可以使用 TileCahce 的“disk”缓存模式然后使用 osgearth 的 TileCache 驱动程序访问缓存文件目录。

6、Caching

Osgearth 可以缓存源码程序 tiles 以提高性能。

一张图片或者 heightfield 已经创建以后，一般访问本地缓存比重新建立这个程序或者从网上下载来的更快。

地图中所有图层共享一个缓存对象，有多种缓存类型可以用在 osgearth，但是所有缓存必须以如下的格式定义：

```
<cache type="cache_type">
   <property1>value</property2>
   <property2>value</property2>
   ...
 </cache>
```

参数 type：要建立的缓存类型。所有的属性及相应的值将被传递到缓存配置。

6.1 一般注意事项

- 所有的基于磁盘的缓存使用图像和 heightfield 的名称来创建文件夹用来缓存。请确保各个缓存路径使用图层名称是唯一的。
- OsgEarth 使用 OpenSceneGraph 现有的插件来读写数据。确保你指定的缓存格式对于读写有良好的支持。举个例子，OpenSceneGraph2.6.1 中的 tiff 插件对于写 16 位和 32 位的数据集存在一些问题，所以它对于缓存 hetghtfields 是不合适的。DDS 插件不支持 16 位和 32 位数据集写入磁盘。Tiff 插件推荐使用 OSG2.8 或更高版本的 OSG。
- osgEarth 完全关闭缓存的运行，只要设置环境变量 OSGEARTH_CACHE_ONLY 就可以了，或者使用属性<cache_only>
- 可以通过设置<cache_enabled>和<cache_format>来覆盖缓存格式和禁用某个层的缓存

Seeding the Cache

有时在某一特定领域，预先处理（pre-seed）你的缓存是很有用的。

osgEarth 提供一个实用的应用程序，osgearth_seed 去完成这项任务。Osgearth_seed 采用（take）一个 Earth file 并填充你的缓存

osgearth_seed 用法：

```
Command:

       osgearth_seed [options] filename

Options:
       --bounds minx miny maxx, maxy         The geospatial
extents to seed.
       --min-level level                     The minimum level
to seed down to.
       --max-level level                     The maximum level
to seed down to.
```

```
      -b                                    Shorthand for
--bounds.
      --cache-path                          Use a different
cache path than the one defined in the earth file
      --cache-type                          Override the cache
type if you override the cache path (tms or disk).
```

缓存类型

TMS（默认）

如果 cache 类型是 tms 那么 osgearth 缓存文件必须使用与“http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification”所规定的格式

Tms 样式属性：

Path-磁盘上缓存切片的路径

Format-缓存的格式（如 png,jpg,dds 等）。如果不指定格式将使用首选的图片或者 heightfield 源码，如果首选的格式也没有，那默认的格式是 png

Tms_type –如果 tms_type 的值是 google,那么 y tile index 将反转变为 0，0 是左上角的 tile

而不是左下角的 tile.

```
<map name="My Map" type="geocentric">


  <!--Specify a map level "tms" cache for all images and
heightfields-->
  <cache type="tms">
     <path>c:/osgearth_cache</path>
  </cache>


  <image name="bluemarble" driver="tms">
```

```
<url>http://labs.metacarta.com/wms-c/Basic.py/1.0.0/satellite/
</url>
  </image>


  <image name="inset" driver="gdal">
     <url>c:/data/inset.tiff</url>
  </image>


</map>
```

Sqlite3

设置缓存类型是 sqlite3，用 sqlite 数据库文件来存储你的缓存，全部的缓存仅仅是一个单独的文件，如果需要转移它将是非常容易的

属性：

Path-数据库文件路径

例子

```
<map type="globe">


  <cache type="sqlite3">
     <path>c:/osgearth_cache.db</path>
     <max_size>300</max_size> <!-- MB -->
  </cache>


  <image name="bluemarble" driver="tms">
```

```
<url>http://labs.metacarta.com/wms-c/Basic.py/1.0.0/satellite/
</url>
  </image>
```

Tile Cache

如果缓存类型是 tile cache，osgearth 缓存文件到磁盘的格式是与 Metacarta 的“Disk”风格缓存想兼容的格式。这意味着：osgearth 缓存的结果可以被那些客户端访问，这些客户端可以公开访问 tileCache 磁盘高速缓存如 OpenLayers 或者 osgearth.这是很有意义的，当你在浏览和缓存 tiles 时，你实际上是在创建一张可以被其他客户端访问地图。所以假如没有缓存类型指定，可以考虑 tileCache。

属性

Path-缓存 tiles 的磁盘路径

Format-缓存的格式（如 png,jpg,dds 等）。如果不指定格式将使用首选的图片或者 heightfield 源码，如果首选的格式也没有，那默认的格式是 png

例子

```
<map name="My Map" type="geocentric">
  <!--Specify a map level "tilecache" cache for all images and
heightfields-->
  <cache type="tilecache">
    <!--All tiles will be cached to c:/osgearth_cache-->
    <path>c:/osgearth_cache</path>
    <!--All tiles will be cached to png format.  If we did not
specify the format, tiles would be cached to the preferred format
of the tile source-->
```

```
    <format>png</format>
  </cache>

  <image name="TMS" driver="tms">

<url>http://labs.metacarta.com/wms-c/Basic.py/1.0.0/satellite/
</url>
  </image>

  <image name="inset" driver="gdal">
     <url>c:/data/inset.tiff</url>
   </image>

</map>
```

7、Osgearth 开发指南（版本 2.x）

Osgearth 的一个主要目标是易于应用集成，实际上有 2 种方式去整合 osgearth.a)你可以建立一个 earth file，然后在你的应用程序中加载它.b)在您使用 osgearth API 的时候以编程方式动态建立一个地图。

标题：

- 地图
- 公共库
- 着色合成
- 建立自己的驱动

7.1 地图

7.1.2 加载一张地图

绝大多数 osgearth 的功能是包含在 OpenSceneGraph 插件里的，所以从 earth file 加载一张地图是很容易的，只要一行代码就可以了：

```
osg::Node* globe = osgDB::readNodeFile("myglobe.earth");
```

你现在有一个可以被加载到你的场景图并可以显示的 OpenSceneGraph 节点了。这种加载地图的方法不是所有的应用程序必须做的。如果你的应用程序支持 VirtualPlanetBuilder 地形数据库，整合 osgearth 就可以做很少的工作，因为场景图形之间是极其相似的。两者都是基于 osgTerrain 和 PagedLOD 系统，都是以 osg::CoordinateSystemNode 修饰的。唯一的不同是一个 osgearth 数据库是动态生成的而不是预先生成的

7.1.3 编程方式创建地图

Osgearth 也提供一个 API 供您动态创建地图。如果您的应用程序允许在运行时从不同层中选择一个层显示，那么这种方式就显得很有用。

基本步骤：

- 建立一个地图对象
- 加入您认为合适的影像和高程地图
- 建立将渲染地图对象的地图节点
- 向场景加入你的地图节点

你可以在任何时候加入图层。但是，一旦向地图加入了一个图层，这张地图就有了选项的副本，你不能再改变它了。

```
#include <osgEarth/Map>
#include <osgEarth/MapNode>
#include <osgEarthDrivers/tms/TMSOptions>
#include <osgEarthDrivers/gdal/GDALOptions>

using namespace osgEarth;
using namespace osgEarth::Drivers;
...

// Create a Map and set it to Geocentric to display a globe
Map* map = new Map();

// Add an imagery layer (blue marble from a TMS source)
{
    TMSOptions tms;
    tms.url() =
"http://labs.metacarta.com/wms-c/Basic.py/1.0.0/satellite/";
    ImageLayer* layer = new ImageLayer( "NASA", tms );
    map->addImageLayer( layer );
}

// Add an elevationlayer (SRTM from a local GeoTiff file)
```

```
{
    GDALOptions gdal;
    gdal.url() = "c:/data/srtm.tif";
    ElevationLayer* layer = new ElevationLayer( "SRTM", gdal );
    map->addElevationLayer( layer );
}

// Create a MapNode to render this map:
MapNode* mapNode = new MapNode( map );
...
viewer->setSceneData( mapNode );
```

Working with a MapNode at runtime

不论这个地图是从 earth file 加载的还是运行时动态建立，地图在运行时是可以修正改良的。

如果地图是从 Earth 文件加载的，首先你必须获得地图节点的 reference.

Map::findMapNode 方法：

```
//Load the map
osg::Node* loadedModel = osgDB::readNodeFile("mymap.earth");

//Find the MapNode
```

```
osgEarth::MapNode* mapNode =
MapNode::findMapNode( loadedModel );
```

一旦获取了地图节点 MapNode 的 reference,就可以加入图像或高程数据。

```
// Add an OpenStreetMap image source

TMSOptions driverOpt;

driverOpt.url() = "http://tile.openstreetmap.org/";

driverOpt.tmsType() = "google";



ImageLayerOptions layerOpt( "OSM", driverOpt );

layerOpt.profile() = ProfileOptions( "global-mercator" );



ImageLayer* osmLayer = new ImageLayer( layerOpt );

mapNode->getMap()->addImageLayer( osmLayer );
```

你也可以删除或者重新排列各图层

```
// Remove a layer from the map.  All other layers are repositioned
accordingly

mapNode->getMap()->removeImageLayer( layer );



// Move a layer to position 1 in the image stack

mapNode->getMap()->moveImageLayer( layer, 1 );
```

Working with Layers

地图包含 ImageLayers 和 ElevationLayers.在运行时你可以修改一些属性。比

如，你可以切换一个图层的状态使它开或者关，还可以修改一个 ImageLayers 的透明度。

```
ImageLayer* layer;

...

layer->setOpacity( 0.5 );  // makes the layer partially
transparent
```

7.2 公共库

Osgearth 提供一个公共的库叫 osgEarthUtil ,它包含大量有用的类，在您的应用程序中可以使用。

Controls：一个简单，容易使用的 2D UI 工具包

EarthManipulator：一个完全可编程的 osgGA::CameraManipulator

EarthManipulator：一个可编程的面向地理地图可视化的 OSG 相机控制者，

它有许多功能：

- 自定义鼠标/键盘输入绑定，设置视角界限等等的 API 接口
- 保存和恢复通过纬度/经度或者其他空间坐标表示的观察点
- Smooth fly-to-Viewpoint function that's smart about round-earth maps
- 对象绑定，自动与任何 OSG 节点绑定你的基准点
- 灵敏度控制
- 轻松获取关于地图的空间参考信息
- ObjectPlacer-通过纬度/长度定位对象的实用工具
- 负责定位将 OSG 众节点放置于一个地形上这样一个共同任务：
- 将节点放置于纬度/经度坐标系上
- 在用户指定的 LOD 地形上进行可选性地形夹合操作
- 生成任何纬度/经度的布局矩阵

AutoClipPlaneHandler：调整基于地平线的剪裁平面

这是一个 OSG::GUIEventHandler，能自动调节一个 geocentric 地图上剪裁平面的远近

当你选择使用 geocentric 地图时，你会发现这样一个问题，当你太靠近地面时，出现近剪裁不堪入目的画面，这是一个地形引擎如何剔除 tiles 造成的问题。

AutoClipPlaneHandler：通过自动调节从你的相机到水平面的距离进而解决这个难题。

AutoClipPlaneHandler 的安装：

```
osgViewer::Viewer viewer;
...
viewer.addEventHandler( new AutoClipPlaneHandler() );
```

ObjectLocator：方便放置和移动实体的基类

7.3 Shader Composition

Osgearth 在它的一些渲染模式中使用 GLSL（OpenGL Shading Language）着色器。默认情况下，osgearth 将检查你的图形硬件能力，自动选择一种合适的模式。

由于 osgearth 依赖于着色器，同时作为开发者你也希望使用你自己的着色代码，osgearth 提供了一个着色组成框架（osgearth shader composition），当你把你自己的着色器融合进 osgearth 时,这个框架表现出了强大的灵活性。

下面我们将介绍一些融合自己着色器到 osgearth 的方法。当然前提是你已经理解了 osgearth 着色组成框架的基本原理。

Framework Basics

Osgearth 会安装默认的着色器，默认的着色器如下所示：

请注意以下功能类型：

- 内置功能：osgearth 默认安装的功能（可以覆盖）
- 用户功能：“inject”到着色器之前或之后的内置插件的功能

```
// VERTEX SHADER:


void main(void)
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    ...


    // "LOCATION_PRE_VERTEX" user functions are called here:
    pre_vertex_function_1(...);
    ...
```

```
    // the built-in functions are called next:
    osgearth_vert_setupTexturing();
    if ( lighting_enabled )
        osgearth_vert_setupLighting();

    // "LOCATION_POST_VERTEX" user functions are called last:
    post_vertex_function_1(...);
}



// FRAGMENT SHADER:

void main(void)
{
    vec4 color = vec4(1,1,1,1);
    ...

    // "LOCATION_PRE_FRAGMENT" user functions are called here:
    pre_fragment_function_1(color);
    ...

    // then the built-in osgEarth functions are called:
    osgearth_frag_applyTexturing(color);
    if (osgearth_lighting_enabled)
        osgearth_frag_applyLighting(color);
```

```
    ...

    // "LOCATION_POST_FRAGMENT" user functions are called last:
    post_fragment_function_1(color);
    ...

    gl_FragColor = color;
}
```

Virtual Program

Osgearth 包含一个叫 VirtualProgram 的 OSG 状态属性，表示 shader composition 的运行状态。由于 VirtualProgram 是一个 osg::StateAttribute，所以你可以附加在场景图形中的任何节点。你可以在 osgearth 中覆盖个人着色功能（In the way you can override individual shader functions in osgEarth）。下面的集成部分将演示如何使用 Virtual Program：

Integrating Custom Shaders 集成自定义着色器

使用着色组成 shader composition 有 2 种方法：

- Injecting user functions 用户函数
- Sampling image layers 采样图像层

Injecting user functions 用户函数

在上述着色器核心代码中，你会发现一些“user functions”比如pre_fragment_function_1()等，它们不存在于 osgearth 生成的默认着色器中，你作为开发者能够向内置着色器的不同位置添加”inject”相应的用户函数的代码。

举个例子：

用户自定义函数实现一个简单的“朦胧”效果（可在 osgearth_shadercomp.cpp 查看完整代码）

```
static char s_hazeVertShader[] =
    "varying vec3 v_pos; \n"
    "void setup_haze() \n"
    "{ \n"
    "    v_pos = vec3(gl_ModelViewMatrix * gl_Vertex); \n"
    "} \n";


static char s_hazeFragShader[] =
    "varying vec3 v_pos; \n"
    "void apply_haze(inout vec4 color) \n"
    "{ \n"
    "    float dist = clamp( length(v_pos)/10000000.0, 0, 0.75 );
\n"
    "    color = mix(color, vec4(0.5, 0.5, 0.5, 1.0), dist); \n"
    "} \n";


osg::StateAttribute*
```

```
createHaze()

{

    osgEarth::VirtualProgram* vp = new
osgEarth::VirtualProgram();



    vp->setFunction( "setup_haze", s_hazeVertShader,
osgEarth::ShaderComp::LOCATION_POST_VERTEX );

    vp->setFunction( "apply_haze", s_hazeFragShader,
osgEarth::ShaderComp::LOCATION_POST_FRAGMENT );



    return vp;

}



...

sceneGraph->getOrCreateStateSet()->setAttributeAndModes( creat
eHaze() );
```

函数"setup_haze"是核心顶点着色器调用内置顶点函数之后被调用

apply_haze 是核心碎片着色器调用内置碎片函数之后被调用

插入点有以下 4 个：

Location	Shader	Signature
ShaderComp::LOCATION_PRE_VERTEX	VERTEX	void functionName(void)
ShaderComp::LOCATION_POST_VERTEX	VERTEX	void functionName(void)
ShaderComp::LOCATION_PRE_FRAGMENT	FRAGMENT	void functionName(inout vec4 color)
ShaderComp::LOCATION_POST_FRAGMENT	FRAGMENT	void functionName(inout vec4 color)

正如你看到的，用户函数是向主着色器无缝的插入代码

使用自定义 ShaderFactory (着色器工厂)覆盖 osgearth 的内置功能。Customizing

the Shader Factory

如果你想替换 osgearth 内置着色函数(比如 osgearth_vert_setupLighting()等)，你可以安装一个自定义 ShaderFactory..ShaderFactory 存储在 osgearth 注册表，包含建立内置函数的所有方法。你可以这样安装你自己的 ShaderFactory:

```
#include <osgEarth/ShaderComposition>

...


class CustomShaderFactory : public osgEarth::ShaderFactory

{

   ... override desired methods here ...

};

...


osgEarth::Registry::instance()->setShaderFactory( new
CustomShaderFactory() );
```

上述示例代码是替换 osgearth 内置照明着色器的代码。注意这一点，替换内置纹理函数不一定好使，因为 osgearth 的图像层组成机制是覆盖这些方法，自己执行图层渲染。

Sampling image layers 采样图像层

如何从你的着色器访问地图的 ImageLayers？由于 osgearth 内部自己管理各图像层，纹理单元以及组成，这不是简单的调用 GLSL 的 texture2D()函数。下面告诉你如何去做。

使用接口 TextureCompositor 为你想查询的图层创建一个采样器方法，这样你可以从你的着色器访问这个方法。

举个简单的例子：

```
// assume "layer" is the image layer you want to sample, and "vp"
is a VirtualProgram state attribute:

osgEarth::ImageLayer* layer;

osgEarth::VirtualProgram* vp;



// first get a reference to the texture compositor.

osgEarth::TerrainEngine* engine = mapNode->getTerrainEngine();

osgEarth::TextureCompositor* comp =
engine->getTextureCompositor();



// next, request a sampling shader for the layer in question.

osg::Shader* sampler = comp->createSamplerFunction( layer,
"sampleMyLayer", osg::Shader::FRAGMENT );



// add it to your VirtualProgram:

vp->setShader( "sampleMyLayer", sampler );



...
```

之后在你的着色器代码中调用函数“SampleMyLayer“

```
// FRAGMENT SHADER
void sampleMyLayer(void);  // declaration
...
void someFunction()
{
    ...
    vec4 texel = sampleMyLayer();
    ...
}
```

采样器函数会自动进行与当前的纹理坐标匹配的正确采样

系统格式 System Uniforms

由于 OpenSceneGraph 系统格式是以”osg_”打头，osgearth 则以“osgearth_”为前缀

各种 uniforms：

- 布尔型 bool osgearth_LightingEnabled-GL 灯光是否可用
- Bool[]osgearth_ImageLayerEnabled-图像层“n”是否可用
- Float osgearth_CameraRange-相机位置与当前顶点的距离

8、Geospatial Data Sources 地理空间数据源

8.1 Imagery

1. USGS（美国地质勘探局无缝服务器）-美国拥有大量的影像和高程数据的服务器
2. JPL onEarth(喷气推进实验室)–JPL 提供的采集 WMS 数据的服务器
3. NASA 蓝色大理石-访问 NASA 蓝色大理石影像库
4. ICEDS-综合地球探测卫星委员会的欧洲数据服务器-包含大量图像层的 WMS 服务器，特别是关于欧洲和非洲的
5. NRL GIDB OpenGIS Web Services(海军实验室 GIDB 与 OpenGIS Web 服务器
6. Natural Earth 自然地球-一个公共领域的地图数据集，比例尺有 1:10000000,1:50000000 和 1：110000000;
7. Virtual Terrain Project 虚拟地形项目-全球图像库

8.2 高程数据 Elevation data

1. CGIAR SRTM90m 国际农业研究磋商小组的 SRTM90m-全球 90m 高程数据集
2. SRTM 30 Plus 地形-全球高程数据 包括海洋探测数据，google earth 也是使用这个高程数据
3. SRTM GLCF 全球土地覆盖基金的 SRTM 数据，这里有一个漂亮的 30 弧秒的数据
4. GEBCO (General Bathymetric Chart of theOceans)通用海底地形图

1.3 功能数据 Fearture data

OpenStreeMap-全世界的街道和土地利用的数据（矢量和栅格化矢量数据）

DIVA-GIS-免费下载全球矢量数据（DCW，VMAP 等）

Natural Earth-一个公共领域的地图数据集，比例尺有 1:10000000,1:50000000 和 1：110000000;