LocaSpace-DotNet 开发指南

V1.2.7

撰写单位	**苏州中科图新网络科技有限公司**
最后修订日期	**2017 年 4 月 11 日**

法律说明

目录

概述

LocaSpace 是一个专业的三维地理信息平台。它为公共事业单位、企业和科研机构提供功能强大、性能稳定和性价比高的三维地理信息解决方案。使用 LocaSpace 三维 GIS 平台，用户能够轻松地创建、浏览、分析和发布三维地理信息数据。LocaSpace 还提供方便的二次开发接口。

LocaSpace 与国外同类产品相比具有两大优势：成本和本地化。在保证同等功能和性能的前提下，LocaSpace 为您大幅度降低了软件采购成本，使您的三维地理信息解决方案更具竞争力。LocaSpace 的核心研发团队和技术服务团队都在中国大陆，产品和技术支持更为本地化，不存在语言差异、数据格式和图标不符合中国的标准等劣势。我们的团队能为您提供更好的技术服务。

1.1 开发环境

硬件环境：

CPU：不低于 i7

内存：不低于 8G

硬盘：10G 以上

显卡：推荐使用 Nvidia GTX 系列，不低于 GTX 960。不支持 Quadro 系列显卡。

软件环境：

Windows 操作系统：Windows 7/ Windows 8/ Windows 10。

开发环境：Visual Studio 2012。

目标框架：.NET Framework 4

1.2 主要功能

1.2.1 全空间三维可视化表达

LocaSpace 具备包括对地表、地下、天空、太空在内的全空间三维可视化能力，以数字地球方式对地球空间系统内的自然地物，人工设施、天气现象、人类活动进行一体化显示。

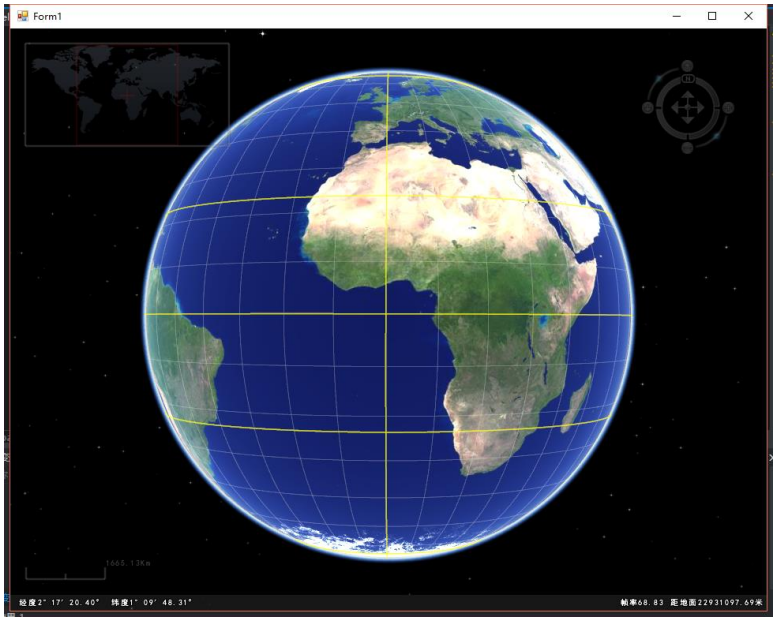


图 1-1 三维球展示

1.2.2 海量数据的三维浏览

实现对不同比例尺海量影像、地形、矢量、模型等数据的大范围、多尺度快速浏览，在 PC 机上实现大型三维场景的流畅的漫游操作

1.2.3 三维空间查询、分析和运算

LocaSpace 具备常规地形分析，如剖面分析、坡度分析/坡向分析、三维缓冲区分析、淹没分析、通视分析等，LocaSpace 还同时支持三维模型的选择、查询，以及空间几何对象关系的各类运算。

1.2.4 方便的二次开发

LocaSpace 基于组件式开发，所有功能以控件的方式封装在 dll 文件中，用户可以方便地进行二次开发，定制个性化界面和功能，甚至将 LocaSpace 嵌入各类信息系统中。

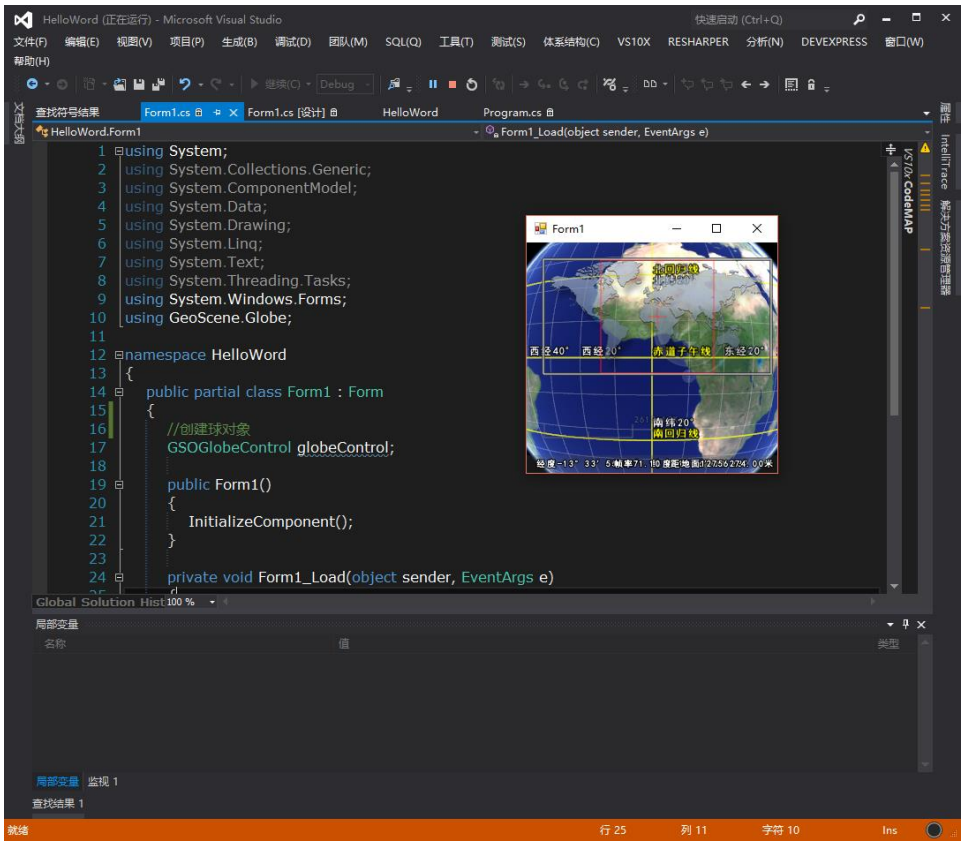


图 1-3 二次开发界面

1.3 技术参数

1. 支持海量地形、影像、模型、矢量、KML 的渲染，支持多图层叠加显示，支持矢量图层设置颜色、形状等风格设置，支持统计图、专题图等；

2. 矢量支持 Shapefile，GML，Autocad 的 dwg、dxf; MapInfo 的 mif；KML 等。

3. 栅格支持 ArcGIS Grid 数据，Erdas 的 img 数据，USGS 的 DEM 格式，支持 Jpeg、png、Tiff、GIF、bmp、raw、jp2k 等图片导入； 模型数据支持 3ds、obj、.x、dae 格式。

4. 支持 Oracle、SQL Server、MySQL 等数据库存储；

5. 支持天空、大气、雾、星空、火焰、雨、雪、风等离子效果、支持水、树木、路灯等虚拟小品构建；

6. 支持飞行、第一视角浏览、支持高精度出图、支持动画效果、支持 AVI 视频录制；

7. 支持空间查询和空间分析（量算、DEM 分析，如：坡度、坡向、填挖方、计算土方量等）、洪水淹没演示、通视分析、可视域分析、雷达分析、剖面分析、扩散分析、噪声分析、光照分析等。

8. 支持矢量批量拉伸建模（面拉伸成体、线拉伸成管道或者公路或者墙面，点拉伸成柱子或者杆）；

9. 支持规则几何体编辑（在场景中可以直接放置球、立方体、柱体等规则几何体）、支持模型库（利用模型库可以指定某个模型为某个点图层的符号样式，达到快速构建场景的目的）、填充库（在编辑的时候可以用填充库指点某个模型某个面的填充风格）；

10. 系统运行速度，在 GeForce6600 上渲染 10 万面帧速可达 60fps；

1.4 LocaSpace SDK 介绍

LocaSpace SDK 主要是面向开发用户的产品，是一套完整的三维 GIS 组件库。 利用 LocaSpace SDK 各个行业用户可以开发自己的行业相关的三维 GIS 应用。LocaSpace SDK 支持的开发语言有：C++，VB，C#，VB.NET，JavaScript。

使用 LocaSpace SDK 可以开发独立运行的程序，也可以嵌入到其他系统内使用。程序开发任意可以在自己的计算机上安装 LocaSpace SDK 开发工具包，该工具包包含三维 GIS 类库，开发帮助，示例代码等。

使用 LocaSpace SDK 可以利用的三维 GIS 功能包括：

加载高分辨率影像数据（DOM）

支持高精度地形数据（DEM）

支持真实地下三维场景

支持 KML，Shapefile，3ds，vrml 等数据格式

采用中间层接口设计，支持 OpenGL 和 D3D

场景中支持各类几何体，例如：点，线，面，三维实体等

三维场景中对象的空间位置的灵活控制

海量的文字标注支持，小比例尺场景下文字标注有智能避让功能，可以定制标注的字体，大小等属性

方便的测量功能，在三维场景中精确测量平面距离，表面距离，高度，投影面积，表面积等

纵剖面分析功能，根据数字高程数据分析纵剖面

三维模型的编辑功能，可以直接在三维场景中编辑模型，例如移动模型的空间位置，缩放模型的大小，旋转模型等

模型的渐进加载，Geometry framework 和纹理的分离加载效果，模型的半透明的渐进加载效果

灵活控制视点，设定飞行轨迹，支持 fly to, jump to, circle 等预定义飞行模式

三维控件支持多种事件，可方便添加对应的事件处理

三维场景参数控制，可以根据具体客户机器的性能，控制系统运行速度和效果，调整三维渲染的参数设置，例如：反锯齿，分形，重采样等

1.5 License 注册

LocaSpace SDK 由 LicenseManager.exe 负责许可的管理，在开发或者部署之前需要首先使用 LicenseManager.exe 来进行授权。

使用方法如下：

运行 LicenseManager.exe，把许可标志码发给我公司市场人员（4008675155）。我们会生成一个 license 文件给您。在这个界面上点击“浏览”按钮，找到 license 文件，点击“应用”按钮，即可完成许可的授权过程。

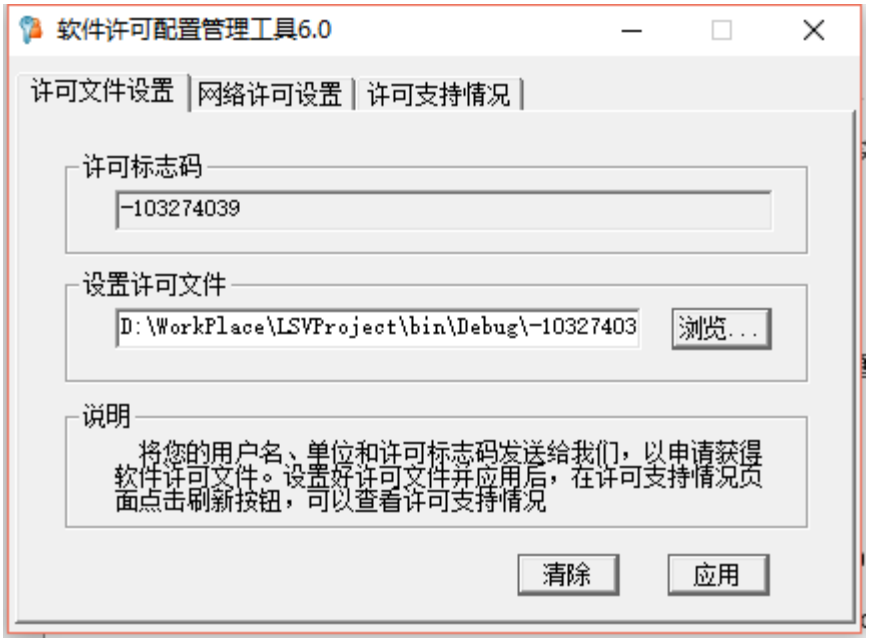
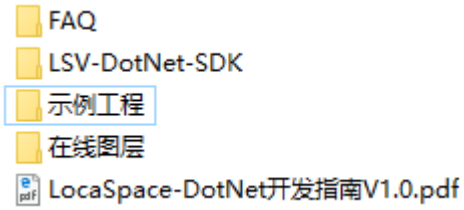


图 1-5-1 LicenseManager 软件界面

1.6 文档结构介绍



其中，

【FAQ】是常见问题集锦，里面记录了常见的问题以及解决办法。

【LSV-DotNet-SDK】是依赖库，里面含有.dll 格式的库文件以及资源文件。

【示例工程】是包含了本当里所有列出的示例工程和源码文件。

【在线图层】是提供了几种 lrc 格式的在线图源的示例。

【LocaSpace-DotNet 开发指南 V1.0.pdf】为本文档。

二、Hello World

*本节源代码收录在《1、HelloWorld》中

LSV 三维球的呈现，项目的开始。向地球 say hello~~

2.1 Visual Studio 2012 创建过程

1. 打开 Microsoft Visual Studio 2012 选择新建 Windows Form Application。

选择.NET FrameWork 4

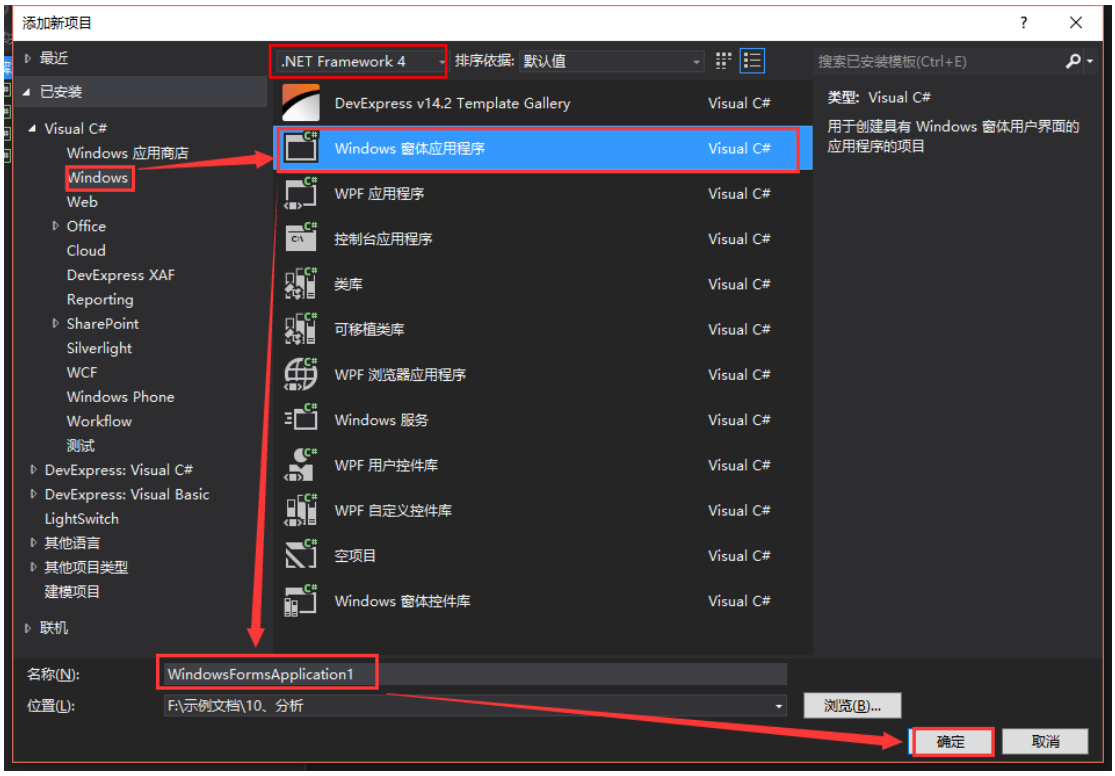


图 2-1-1

2. 把【LSV-DotNet-SDK】文件夹下所有文件拷贝到项目的【Debug】或者【Release】文件夹中。

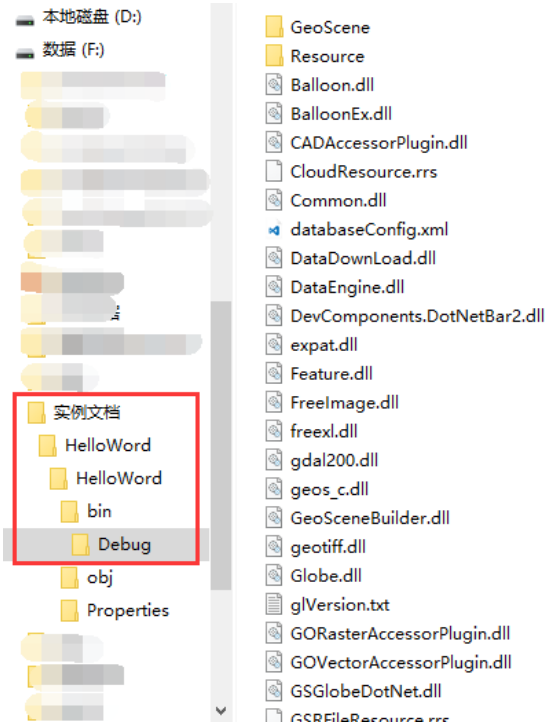


图 2-1-2

3. 在项目管理器的“ 引用”处右击，选择“添加引用”，然后浏览到 Debug 文件夹，选择 GSGlobeDotNet.dll 和 GSToolTipDotNet.dll。其中 GSGlobeDotNet.dll 是核心的 assembly，包含了核心的类库和三维控件。GSToolTipDotNet.dll 提供了和三维控件一起使用的气泡控件，可以方便的在三维场景中显示文本、图片、表格等信息。

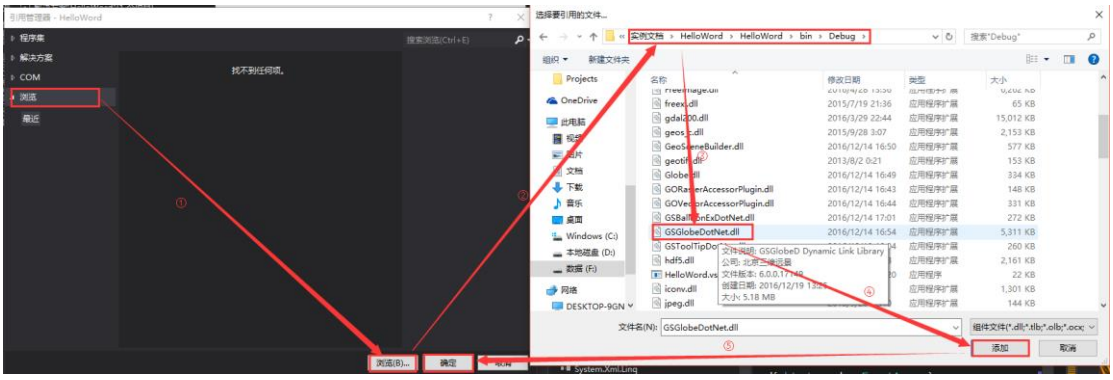


图 2-1-3

4. 现在，开发的基础环境已经搭建好了。现在可以使用 LocaSpace SDK 进行二次开发了。首先在 Form1.cs 中导入命名空间，加载三维球。

```
using GeoScene.Globe;
namespace HelloWord
{
    public partial class Form1 : Form
```

```
    {
        //创建球对象
        GSOGlobeControl globeControl1;
        public Form1()
        {
            InitializeComponent();
            //添加球
            globeControl1 = new GSOGlobeControl();
            this.Controls.Add(globeControl1);
            globeControl1.Dock = DockStyle.Fill;
        }
    }
}
```

点击【F5】运行，可以看到如下的界面：

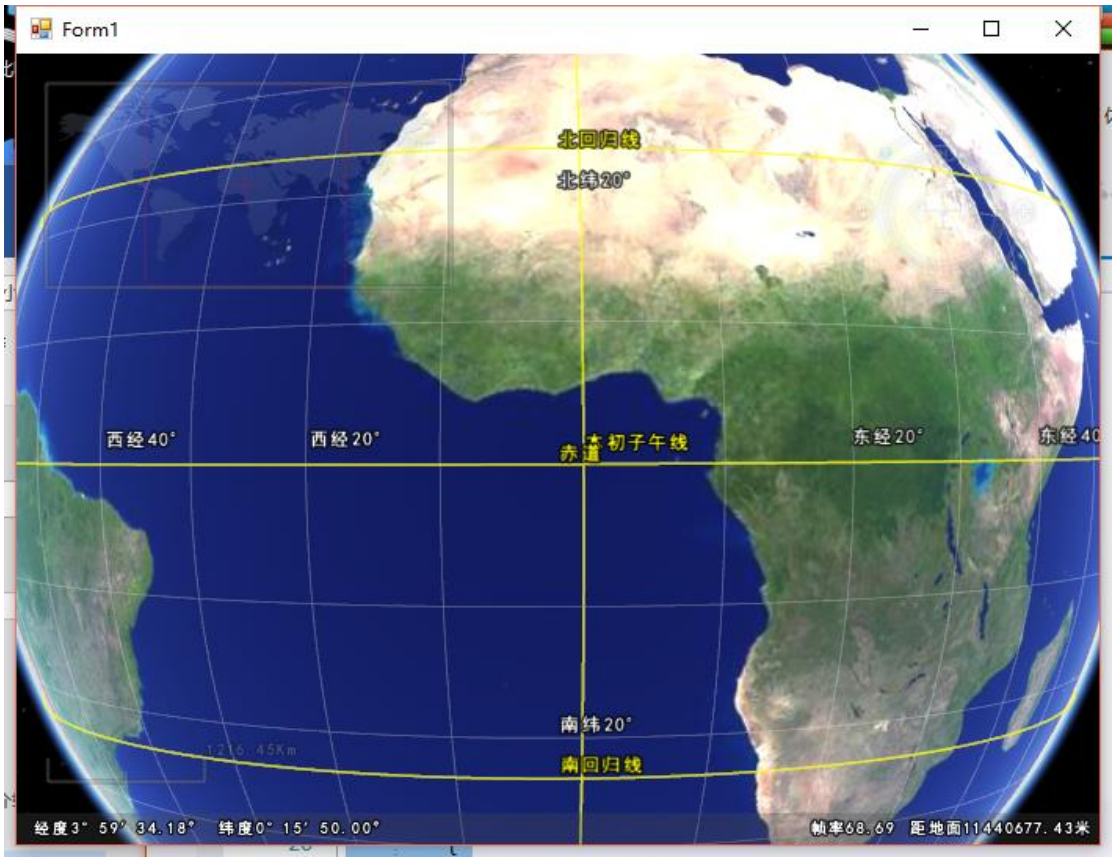


图 2-1-4

2.2 球的属性设置

创建完球，就可以对球的属性进行初始化

```
public Form1()
        {
            InitializeComponent();
            //添加球
            globeControl1 = new GSOGlobeControl();
            this.Controls.Add(globeControl1);
            globeControl1.Dock = DockStyle.Fill;
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            globeControl1.Globe.UserBackgroundColorValid = true;
            globeControl1.Globe.UserBackgroundColor = Color.White;
            globeControl1.Globe.LatLonGrid.Visible = false; //经纬网
            globeControl1.Globe.OverviewControl.Visible = false;//鹰眼
            globeControl1.Globe.ControlPanel.Visible = true;//控制面板
            globeControl1.Globe.ScalerControl.Visible = true; //比例尺
            globeControl1.Globe.StatusBar.Visible = true;//状态栏
            globeControl1.Globe.Atmosphere.Visible = true;    //大气层
            globeControl1.Globe.Atmosphere.ShaderUsing = true; //光影大气
            globeControl1.Globe.MarbleVisible = false;    //大气雾效/大理石表面
            globeControl1.Globe.UnderGroundFloor.Visible = false; //地下网格
            globeControl1.Globe.Antialiasing = true;          //反走样
            globeControl1.Globe.BothFaceRendered = false;
            globeControl1.Globe.FlyToPointSpeed = 200000;//飞行速度
            globeControl1.Globe.EditClampObject = false;//是否依附对象
            globeControl1.Globe.FeatureMouseOverEnable = true;  //允许鼠标浮动
            globeControl1.Globe.Object2DMouseOverEnable = false;//是否鼠标手势
提示
            globeControl1.Globe.IsReleaseMemOutOfView = false;//超出视野重新加
载
        }
```

运行后效果如图：

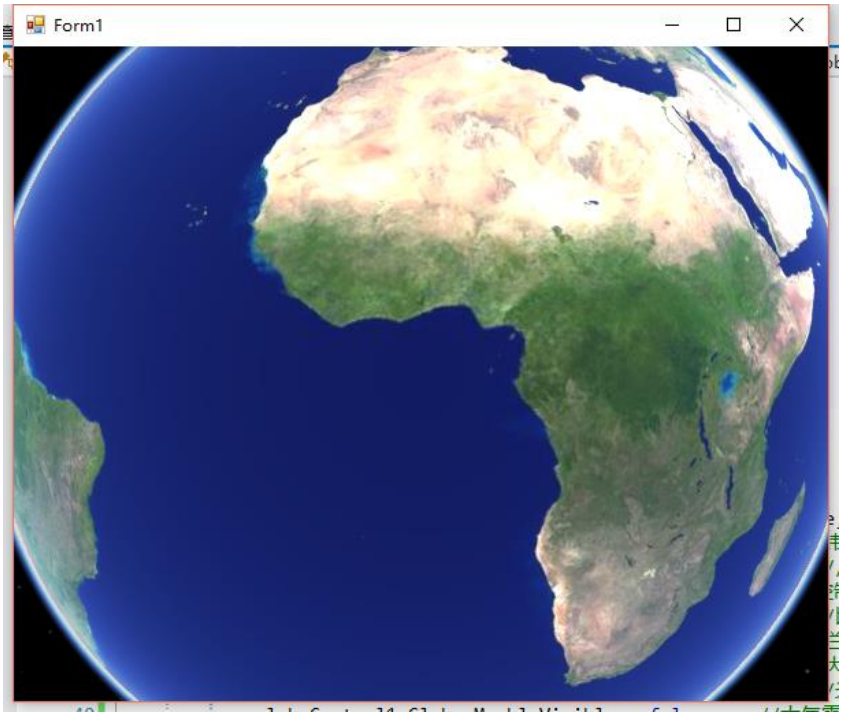


图 2-2-1

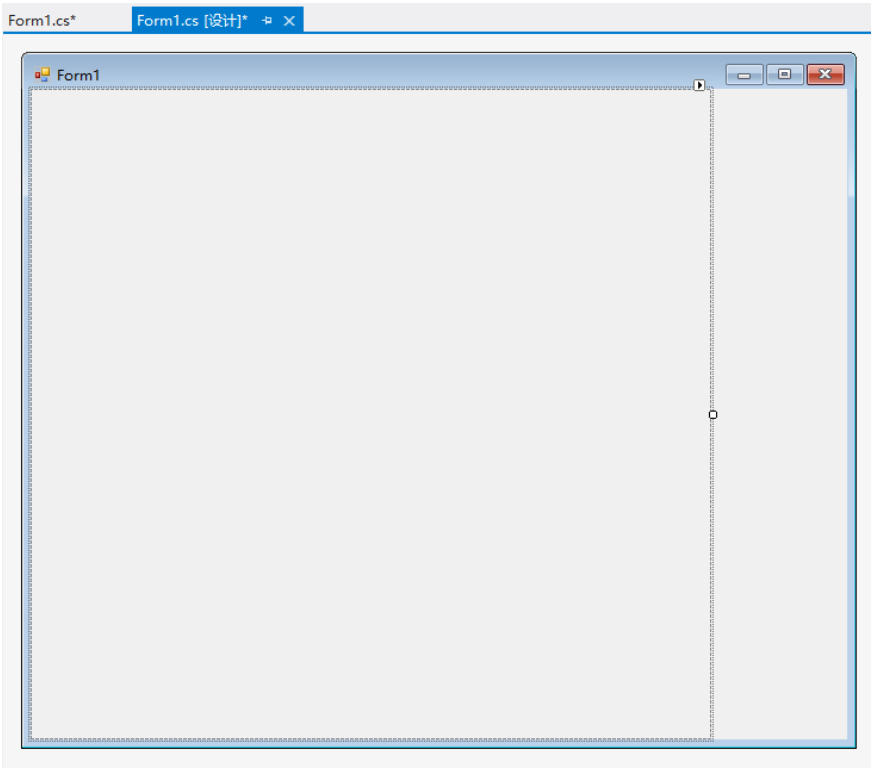
三、图层的添加和删除

*本节源代码收录在《2、图层的添加和删除》中

3.1 图层的添加删除

3.1.1 添加图层

1. 项目中我们先添加一个 panel，将上一章的球添加到 panel 中，留下一部分用来放按钮。




```
public partial class Form1 : Form
    {
        GSOGlobeControl globeControl1;
        public Form1()
        {
            InitializeComponent();
            globeControl1 = new GSOGlobeControl();
            panel1.Controls.Add(globeControl1);
            globeControl1.Dock = DockStyle.Fill;
        }
```

```
}
```

运行效果图如下：

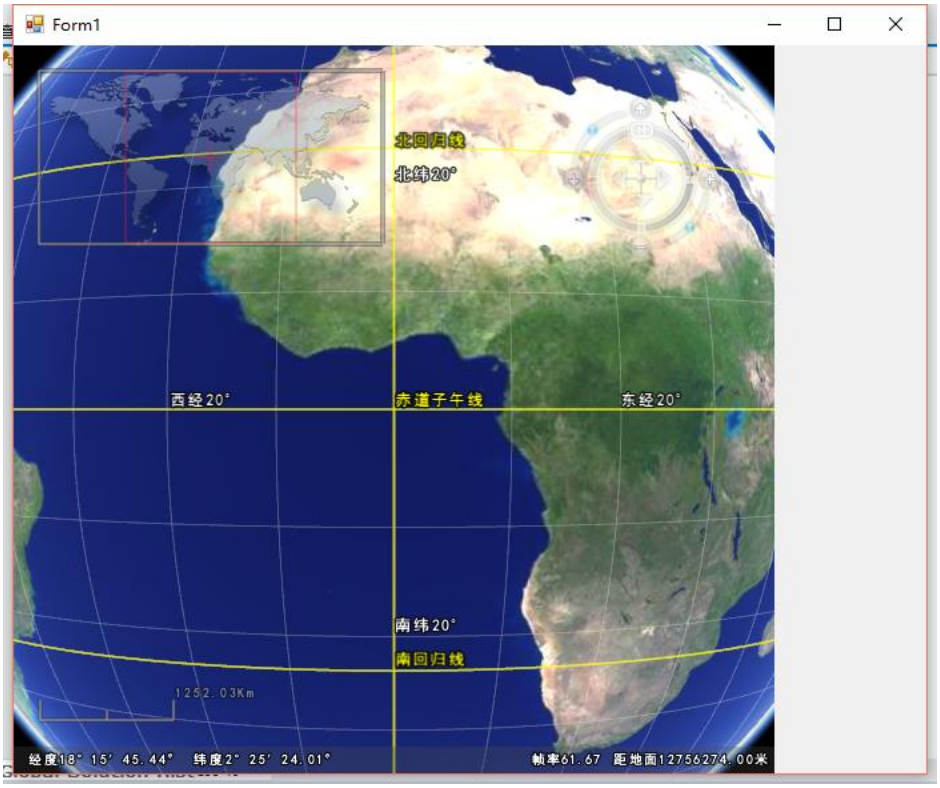


图 3-1-1

2. 添加一个按钮，进行添加图层

```
//添加图层文件
globeControl1.Globe.Layers.Add(
Application.StartupPath + "\\Resource\\gisdata\\tianditudata\\天地图地图.lrc");
globeControl1.Globe.Layers.Add(
Application.StartupPath + "\\Resource\\gisdata\\tianditudata\\天地图标注.lrc");
//刷新球
globeControl1.Refresh();
```

完了运行，可以看到添加图层后的效果。

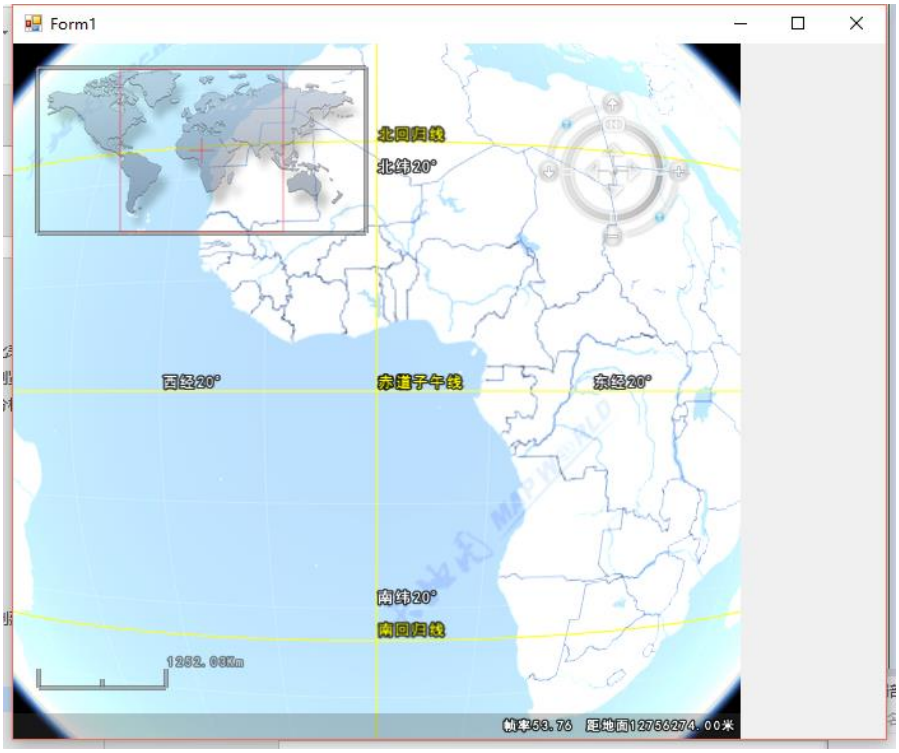


图 3-1-2

3.1.2 删除图层

1. 添加一个按钮，在其中点击删除图层：

```
//删除图层有三种方法
//1.根据图层删除
    //根据说明查找图层，说明为文件名称（不带后缀）
GSOLayer layer = globeControl1.Globe.Layers.GetLayerByCaption("天地图标注");
globeControl1.Globe.Layers.Remove(layer);
//2.根据图层目录删除
globeControl1.Globe.Layers.Remove(
Application.StartupPath + "\\Resource\\gisdata\\tianditudata\\天地图标注.lrc");
//3.根据索引删除
globeControl1.Globe.Layers.Remove(1);
//还有几个删除方法
//全部删除
globeControl1.Globe.Layers.RemoveAll();
//根据图层ID删除
globeControl1.Globe.Layers.RemoveLayerByID(1);
```

3.2 显示隐藏图层

1. 创建一个 Button，用来点击改变显示隐藏图层：

```
private void btnShowD_Click(object sender, EventArgs e)
{
    //根据说明查找图层，说明为文件名称（不带后缀）
    GSOLayer layer = globeControl1.Globe.Layers.GetLayerByCaption("天地图地图");
    layer.Visible = !layer.Visible;
}
```

3.3 图层顺序的调整

1. 我们在上面原有的基础上，再添加一个图层：

```
//添加图层文件
globeControl1.Globe.Layers.Add(Application.StartupPath + "\\Resource\\gisdata\\tianditudata\\天地图地图.lrc");
globeControl1.Globe.Layers.Add(Application.StartupPath + "\\Resource\\gisdata\\tianditudata\\天地图标注.lrc");
```

2. 创建一个按钮，代表移动图层：

```
//图层的移动是通过索引移动的
//索引越大表示越在商城，表示后绘制
//将索引为1的图层移动到下一层，结果为该图层的索引变为2，原来索引为2的图层变为1
globeControl1.Globe.Layers.MoveUp(1);
//将索引为2的图层移动到上一层，结果为该图层索引变为1，原来索引为1的变为2
globeControl1.Globe.Layers.MoveDown(2);
//将索引为1的图层移动到索引为8的位置
globeControl1.Globe.Layers.MoveTo(1,2);
```

运行后效果如下：

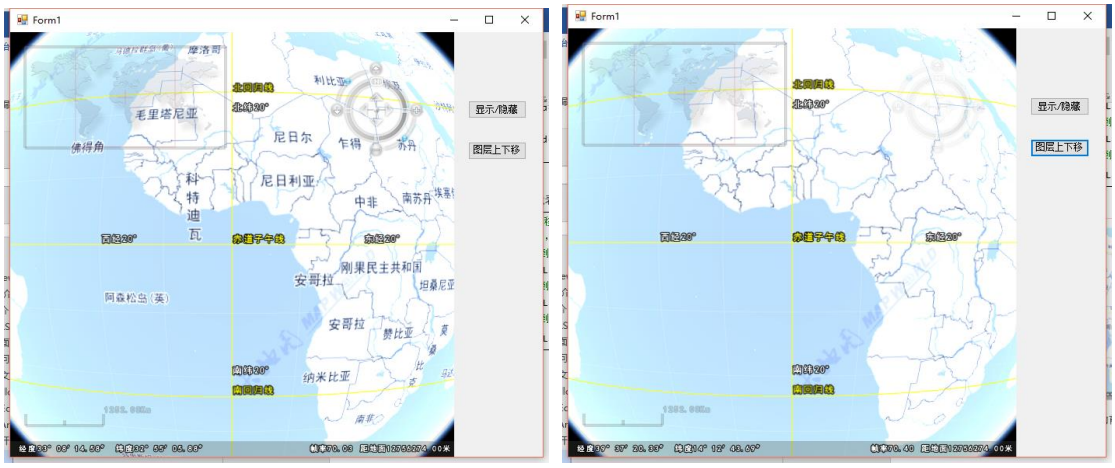


图 3-1-3 移动前后对比

3.4 地形的添加删除

地形管理同图层管理接口基本一致，GSOTerrains 和 GSOLayers 都有添加、获取、移除、调整顺序等操作，用法也基本一样。具体用法可参考图层管理的用法。

四、元素的添加和删除及属性样式设置

*本节源代码收录在《3、元素的添加和删除及属性样式设置》中

4.1 点的添加

创建点对象，并把点对象添加到场景中，需要进行以下步骤：

a. 创建点对象；

b. 创建点的显示风格 GSOMarkerStyle3D()，并把显示风格赋予点；

c. 创建几何要素，并把点对象赋予几何要素，

d. 然后把几何要素添加到图层中；

```
GSOGeoMarker point = new GSOGeoMarker();    //创建点对象
point.X = 120.417888231016;                          //设置点X的值，单位为度
point.Y = 31.3283140323302;                          //设置点Y的值，单位为度
point.Z = 100;                                 //设置点Z的值，单位为米
point.AltitudeMode = EnumAltitudeMode.RelativeToGround;
//设置点的高程模式，设置为相对地面。则点的为（X,Y）处地面高程上方100米（point.Z = 100）
point.Text = "中科图新";                        //设置点对象显示的文字
GSOMarkerStyle3D mstyle = new GSOMarkerStyle3D();//新建点样式
mstyle.IconPath = Application.StartupPath +
"\\Resource\\image\\DefaultIcon.png";     //设置图标路径
point.Style = mstyle;                          //把显示风格
GSOFeature feature = new GSOFeature();       //创建几何要素
feature.Geometry = point;                      //把点赋予集合要素
feature.Name = point.Text;                     //赋予名字
globeControl1.Globe.MemoryLayer.AddFeature(feature);//将要素添加到图层中
//刷新球
globeControl1.Refresh();
```

运行效果如下：

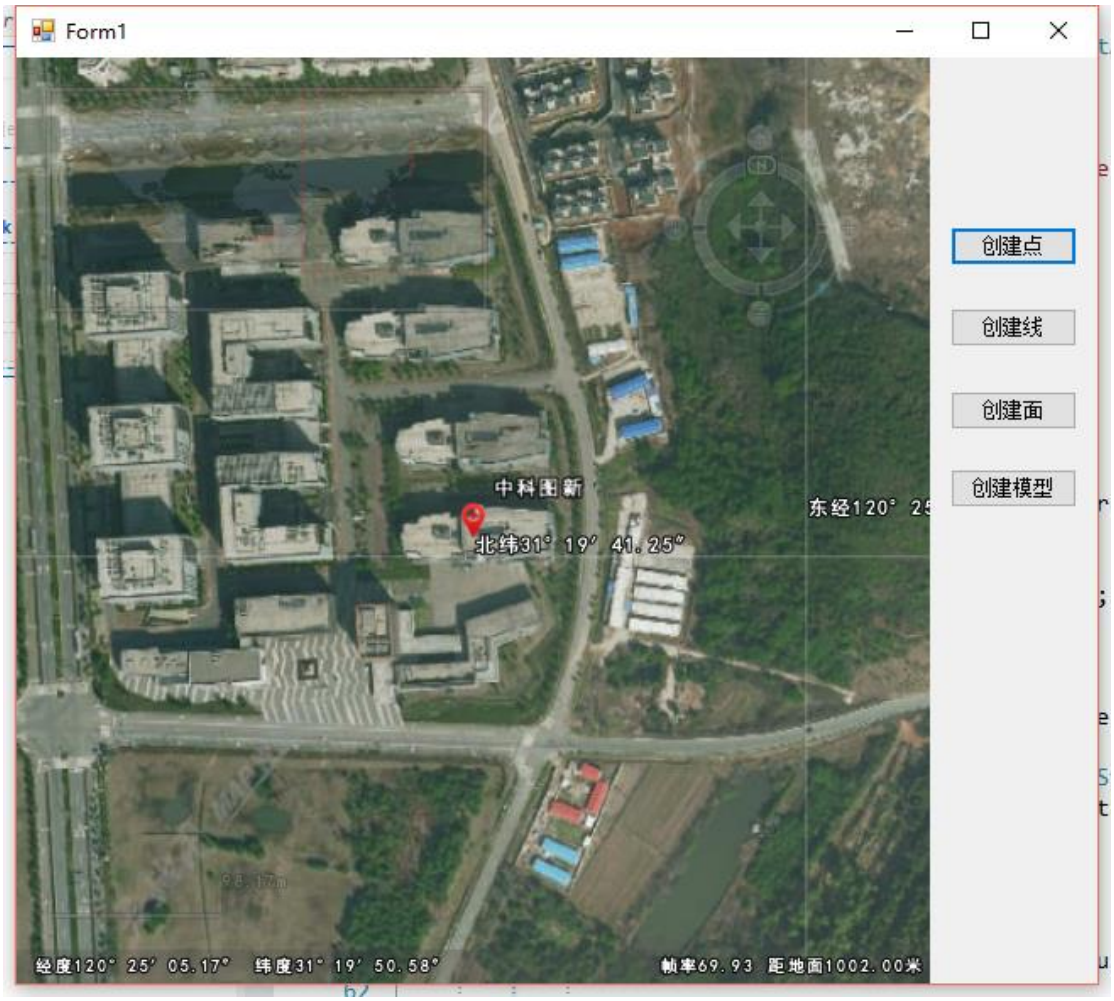


图 4-1-1

4.2 线的添加

创建线对象与创建点对象的步骤基本类似，不同点是增加了节点对象：

a. 创建线对象；

b. 创建节点对象，把节点添加到线上；

c. 创建线的风格，把风格添加到线上；

d. 创建几何要素，把线添加到几何要素上；

e. 把几何要素添加到图层中

```
GSOGeoPolyline3D line = new GSOGeoPolyline3D(); //创建线对象
GSOPoint3ds pnts = new GSOPoint3ds(); //创建节点对象

pnts.Add(new GSOPoint3d(116.6, 39.9, 1000));    //把各节点添加到节点对象上
```

```
pnts.Add(new GSOPoint3d(116.61, 39.91, 3000));
pnts.Add(new GSOPoint3d(116.62, 39.92, 2000));
pnts.Add(new GSOPoint3d(116.63, 39.90, 2500));
pnts.Add(new GSOPoint3d(116.64, 39.94, 4000));
line.AddPart(pnts); //把节点添加到线上

GSOSimpleLineStyle3D style = new GSOSimpleLineStyle3D(); //创建线的风格
//设置透明度及颜色，FromArgb()中的四个参数分别为alpha、red、green、blue，取值范围为0到255
style.LineColor = Color.FromArgb(150, 0, 255, 0);
style.LineWidth = 3;          //设置线的宽度为3
style.VertexVisible = true; //显示线的节点
line.Style = style;          //把风格添加到线上

//创建几何对象并设置属性
GSOFeature f = new GSOFeature();
f.Geometry = line;           //把线对象添加到几何对象上
f.Name = "线 01";          //设置几何对象的名称
f.SetFieldValue("description", "这是线的属性");       //设置几何对象的字段值
//把几何要素添加到内存图层中
globeControl1.Globe.MemoryLayer.AddFeature(f);
//刷新球
globeControl1.Refresh();
```

*globeControl 控件也提供了在三维球上直接绘制线的功能，具体参考《绘制模式》章节

运行结果如图：

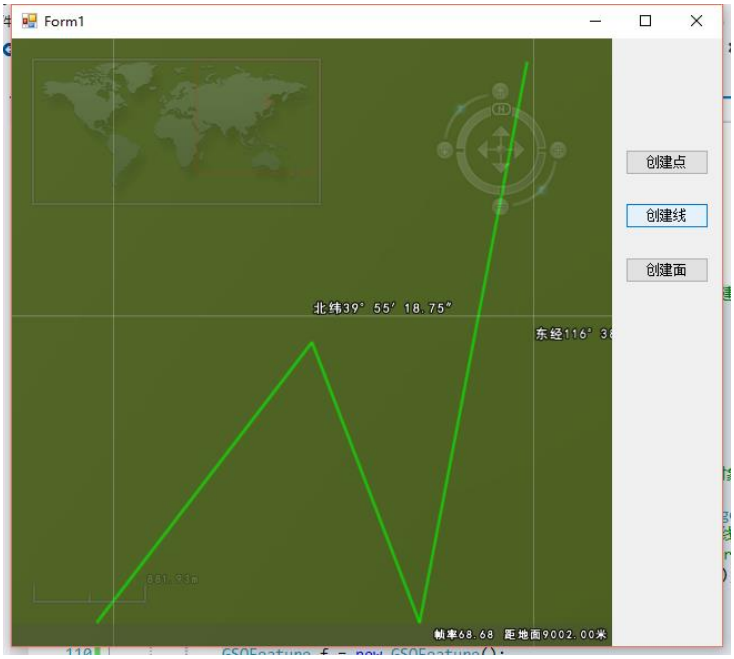


图 4-2-1

4.3 面的添加

创建面对象与创建线对象的步骤相似：

```
GSOGeoPolygon3D geoPolygon = new GSOGeoPolygon3D(); //创建多边形对象

//创建节点对象
GSOPoint3ds polygonPnts = new GSOPoint3ds();
polygonPnts.Add(new GSOPoint3d(116.7, 39.8, 0));
polygonPnts.Add(new GSOPoint3d(116.8, 39.9, 0));
polygonPnts.Add(new GSOPoint3d(116.8, 39.7, 0));
polygonPnts.Add(new GSOPoint3d(116.7, 39.7, 0));

geoPolygon.AddPart(polygonPnts);    //把节点添加到多边形对象上

GSOSimplePolygonStyle3D stylePolygon = new GSOSimplePolygonStyle3D();    //创建风格
stylePolygon.OutLineVisible = true;    //显示多边形的边缘线
//设置多边形的填充颜色，FromArgb()中的四个参数分别为alpha、red、green、blue，取值范围为0到255
stylePolygon.FillColor = Color.FromArgb(100, 255, 255, 0);
geoPolygon.Style = stylePolygon;  //把风格添加到多边形上

//创建几何对象并设置属性
GSOFeature f = new GSOFeature();
f.Geometry = geoPolygon;
f.Name = "多边形 01";
f.SetFieldValue("description", "a demo polygon");

globeControl1.Globe.MemoryLayer.AddFeature(f);  //把几何要素添加到内存图层中
//刷新球
globeControl1.Refresh();
```

*globeControl 控件也提供了在三维球上直接绘制面的功能，具体参考《绘制模式》章节

运行结果如图：

4.5 模型的添加

在场景中添加模型：

```
GSOGeoModel model = new GSOGeoModel(); //创建模型
GSOPoint3d pt = new GSOPoint3d(); //创建点
pt.X = 116.6;
pt.Y = 39.9;
pt.Z = 0;

GSOModelPointStyle3D style = new GSOModelPointStyle3D();  //创建模型的风格
model.Style = style;

//模型可以是3ds、obj、gse、gsez格式的三维模型
//模型所在路径，用户可根据实际情况进行设置
string filepath = Application.StartupPath + "\\Model\\dongbang.3DS";

//设置模型
model.FilePath = filepath;
model.Position = pt;

model.AltitudeMode = EnumAltitudeMode.ClampToGround; //把几何体放到地面上

GSOFeature f = new GSOFeature(); //创建几何要素
f.Geometry = model;
f.Name = "模型 01";
f.Description = "模型 01";  //设置feature description的值，这个值将在tooltip上显示

//把几何要素添加到内存图层中
GSOFeature newFeature = globeControl1.Globe.MemoryLayer.AddFeature(f);
globeControl1.Refresh();    //刷新场景
globeControl1.Globe.FlyToFeature(f);  //飞行到模型所在的位置
//刷新球
globeControl1.Refresh();
```

4.6 属性样式设置

每个要素都有自己对应的风格（style）可以设置。

*本示例为部分示例，详细示例请参考《LocaSpace 接口说明》

4.1.1 点风格

```
GSOMarkerStyle3D mstyle = new GSOMarkerStyle3D();//新建点样式
mstyle.IconPath =
Application.StartupPath + "\\Resource\\image\\DefaultIcon.png";//设置图标路径
mstyle.IconVisible = true;              //标注图标是否可见
mstyle.TextAvoidance = false;           //标注文本是否为空
mstyle.TextStyle = new GSOTextStyle();  //标注文本风格（详情请见“文本风格”）
mstyle.TextVisible = true;              //标注文本是否可见
```

4.1.2 线风格

```
GSOSimpleLineStyle3D style = new GSOSimpleLineStyle3D(); //创建线的风格
//设置透明度及颜色，FromArgb()中的四个参数分别为alpha、red、green、blue，取值范围为0到255
style.LineColor = Color.FromArgb(150, 0, 255, 0);
style.LineWidth = 3;          //设置线的宽度为3
style.VertexVisible = true; //显示线的节点
style.LineType = EnumLineType.Custom;//设置线的类型
```

4.1.3 面风格

```
GSOSimplePolygonStyle3D stylePolygon = new GSOSimplePolygonStyle3D();    //创建风格
stylePolygon.OutLineVisible = true;    //显示多边形的边缘线
//设置多边形的填充颜色，FromArgb()中的四个参数分别为alpha、red、green、blue，取值范围为0到255
stylePolygon.FillColor = Color.FromArgb(100, 255, 255, 0);
stylePolygon.Fill = true;                          //面内部是否填充
stylePolygon.OutLineVisible = true;                //是否显示轮廓线
stylePolygon.OutlineStyle = new GSOLineStyle3D();   //轮廓线样式（参考线风格）
stylePolygon.TextureParam = new GSOTextureParam();  //表面纹理样式
```

4.7 获取要素（Feature）

要素代表着三维球上面的元素，点、线、面和模型等等。

获取要素是操作球体上的要素必须掌握的能力，获取要素的方式有很多种：

1、全局变量获取

初始化获取代表着，当加入球体的时候，就用全局变量获取到这个要素：

```
GSOFeature feature; //创建一个全局变量

feature = globeControl1.Globe.MemoryLayer.AddFeature(f);  //添加的时候获取要素

MessageBox.Show("获取到：" + feature.Name);
```

2、根据属性获取

每个要素都有独一无二的属性，我们可以根据属性找出要素：

```
GSOFeature myFeature = null;
//根据CustomID获取①
myFeature = globeControl1.Globe.MemoryLayer.GetFeatureByCustomID(1);
//根据ID来获取要素
myFeature = globeControl1.Globe.MemoryLayer.GetFeatureByID(1);
//根据名称获取要素
myFeature = globeControl1.Globe.MemoryLayer.GetFeatureByName("多边形 01",
true)[0];
//根据描述获取要素
myFeature = globeControl1.Globe.MemoryLayer.GetFeatureByDescription("这是一个
多边形", true)[0];
```

①，CustomID 的生存周期只在创建起到关闭软件。它不会以任何形式保存。

3、根据点击要素获取

我们也可以根据我们点击三维球来获取点击到的要素：

```
//添加球点击事件
globeControl1.FeatureMouseClickEvent += (o, args) =>
{
```

```
//判断是否为模型
if (args.Feature != null)
{
    MessageBox.Show("获取到:" + args.Feature.Name);
}
```

4、根据选中的要素获取

三维球有很多的模式，像是浏览模式、选中模式等等，我们可以通过启用含有的选中模式来获取已经选中了的要素：

```
//改变球为选中模式
globeControl1.Globe.Action = EnumAction3D.SelectObject;
GSOFeature myFeature = null;
myFeature = globeControl1.Globe.SelectedObject;
MessageBox.Show("获取到:" + (myFeature == null?"null":myFeature.Name));
```

4.8 要素的删除

可以根据在图层中查找要素，根据 Name，ID，Description 等进行查找删除。

*注意，因为 CustomID 生存周期的问题，重加载之后便会失效，所以不宜用作查找条件。

```
//在球上查找后删除，可用Name,ID,Description等查找。
GSOFeature feature = globeControl1.Globe.MemoryLayer.GetFeatureByName("中科图新",false)[0];
feature.Delete();
globeControl1.Refresh();
```

也可以通过全局变量中保存的 GSOFeature 要素进行删除：

```
//根据全局变量删除
lineFeature.Delete();
globeControl1.Refresh();
```

或者将要素的数据源绑定在树形图节点或者其他控件上，通过控件读取要素进行删除：

```
//根据绑定数据删除
(btnRemovePolygon.Tag as GSOFeature).Delete();
globeControl1.Refresh();
```

也可以通过在地图上选定要素进行删除：

```
//通过选中进行删除
if (globeControl1.Globe.SelectedObject == null)
{
    MessageBox.Show("已经切换为选择模式，请选择删除的要素，再点击此按钮！");
    globeControl1.Globe.Action = EnumAction3D.SelectObject;
    return;
}

globeControl1.Globe.Action = EnumAction3D.ActionNull;
globeControl1.Globe.SelectedObject.Delete();
globeControl1.Refresh();
```

4.9 要素的显示隐藏

根据上一小节获取到了要素，我们就可以对要素进行二次改变：

```
feature.Visible = !feature.Visible;
//刷新球
globeControl1.Refresh();
```

4.10 气泡与标注

4.10.1 气泡提示（GSOBalloon）

LocaSpace 提供 GSOBalloon 功能，用户把鼠标放在场景中的几何对象(GSOFeature)上面稍作停留，该几何对象上就会出现 GSOBalloon，显示该几何对象 Name 和 Description 属性。目前，GSOBalloon 仅对点和模型起作用，对于线和多边形不起作用。GSOBalloon 的效果如下图所示：

“点 01”是该点的 name 属性，“首都机场”是该点的 Description 属性。

ToolTip 的显示与隐藏是通过场景和几何对象的一系列事件(Event)实现的，这些事件包括：

FeatureMouseOverEvent（该事件的触发需要设置控件的 FeatureMouseOverEnable 属性为 true）

FeatureMouseHoverEvent

FeatureMouseIntoEvent

MouseMove

FeatureMouseOutEvent 等。

关于事件的详细介绍，请看相关章节。下面的例子给出使用 FeatureMouseIntoEvent 和 FeatureMouseOutEvent 实现 ToolTip 的方法：

首先先声明一个全局变量：

```
GSOBalloon featureTooltip;
```

实例化 GSOBalloon 对象，并设置属性，以下代码放在 globeControl1 控件创建以后，因为 GSOBalloon 的构造函数需要 globeControl1 控件的句柄作为参数。

```
featureTooltip = new GSOBalloon(globeControl1.Handle);      //实例化GSOBalloon 对象
featureTooltip.SetSize(EnumSizeIndex.ROUNDED_CX, 5); // 设置边角的圆润度
featureTooltip.SetSize(EnumSizeIndex.ROUNDED_CY, 5); // 设置边角的圆润度
featureTooltip.SetSize(EnumSizeIndex.MARGIN_CX, 3);  // 设置空白边缘宽度
featureTooltip.SetSize(EnumSizeIndex.MARGIN_CY, 3);  // 设置空白边缘宽度
featureTooltip.SetSize(EnumSizeIndex.ANCHOR_HEIGHT, 30); // 设置GSOBalloon 锚的高度
featureTooltip.EscapeSequencesEnabled = true;
featureTooltip.HyperlinkEnabled = true;  // 设置是否可以点击GSOBalloon里面的超链接
featureTooltip.Opaque = 30;  // 透明度，取值范围是0～100，0为不透明，100为全透明
featureTooltip.MaxWidth = 300;  // 最大宽度
featureTooltip.SetBorder(Color.FromArgb(255, 255, 128, 64), 1, 1);  // 边框
featureTooltip.SetColorBkType(EnumBkColorType.SILVER);  //填充颜色
// 也可以下面方法设置
//featureTooltip.SetColorBk(Color.FromArgb(255, 255, 255, 255),
Color.FromArgb(255,240, 247, 250), Color.FromArgb(255,192, 192, 200));
featureTooltip.SetEffectBk(EnumBkEffect.VBUMP, 10); //渐变效果
featureTooltip.SetShadow(0, 0, 50, true, 0, 0); // 阴影
//featureTooltip.CloseButtonVisible = true;  //显示关闭按钮
```

在组窗体的 Form_load 函数中加入事件处理(event handler):

```
//鼠标进入显示气泡
globeControl1.FeatureMouseIntoEvent += (sender, e) =>
{
    if (!featureTooltip.IsVisible())
    {
        String str1 = "<table><tr><td valign=vcenter><center><h2> " +
e.Feature.Name + "</h2><br><hr color=blue></center></td></tr></table>";
        String str2 = e.Feature.Description;

        // 显示GSOBalloon
        featureTooltip.ShowBalloon((int)e.MousePos.X, (int)e.MousePos.Y, str1
+ str2);

    }
};

//鼠标移出时隐藏气泡
globeControl1.FeatureMouseOutEvent += (sender, e) =>
{
    if (featureTooltip.IsVisible())
    {
        globeControl1.SwapBuffer(); //为了避免闪屏
        featureTooltip.HideBalloon();
    }

};
```

GSOBalloon 的显示内容是通过 html 语言进行设置，目前支持文字和图片。显示标题文字，可以使用代码 <h2>some text</h2>。添加图片可使用以下代码：

```
<img src="C:/test.png" width=200 height=200>
<img src=" http://192....." width=200 height=200>
<img file="C:/test.png" width=200 height=200>
<img file=" http://www.foo.com/demo.jpg" width=200 height=200>
```

width=200 height=200 表示图片被拉伸到 200,200。如果不写这两个表示图片按原大小输出。路径也可以不带双引号。

注意鼠标浮过对象要显示 tooltip 和让对象高亮，必须启用下面这个开关才行

```
globeControl1.Globe.FeatureMouseOverEnable = true;
```

4.10.2 增强气泡信息提示（GSOBalloonEX）

增强性气泡可以直接显示 html 页面，可以支持各种 html 标签，支持在气泡中嵌入 flash 动画、视频等；具有更强的表现力。

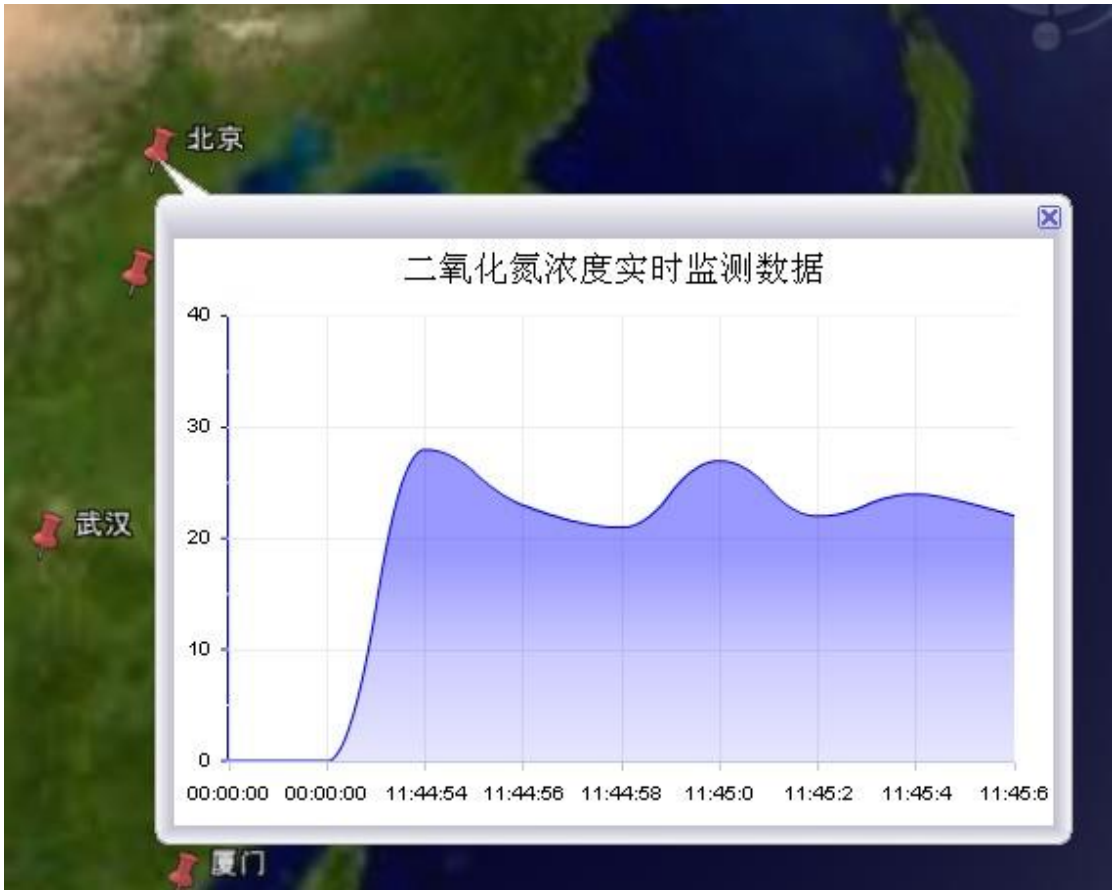


图 4-10-2-1 增强性气泡提示

气泡的显示与隐藏是通过场景和几何对象的一系列事件(Event)实现的，这些事件包括：

FeatureMouseOverEvent（该事件的触发需要设置控件的 FeatureMouseOverEnable 属性为 true）

FeatureMouseHoverEvent

FeatureMouseIntoEvent

MouseMove

FeatureMouseOutEvent 等。

关于事件的详细介绍，请看相关章节。

首先引用 SDK 下的 GSBalloonExDotNet.dll，再声明全局变量：

```
GSOBalloonEx balloonEx;
```

实例化 GSOBalloonEx 对象，并设置属性，以下代码放在 globeControl1 控件创建以后，因为 GSOBalloonEx 的构造函数需要 globeControl1 控件的句柄作为参数。

```
balloonEx = new GSOBalloonEx(globeControl1.Handle);
```

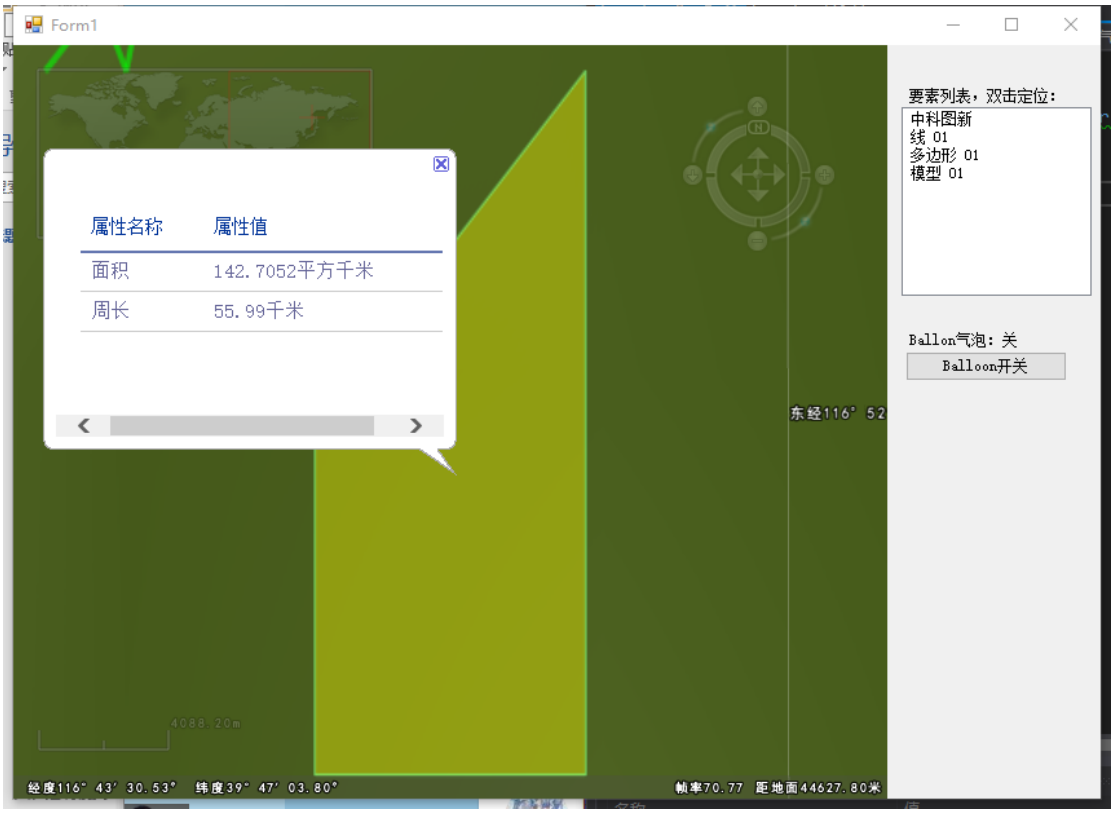
然后在相应事件中处理函数中显示或者隐藏气泡：

```
//点击要素显示
globeControl1.FeatureMouseClickEvent += (sender, e) =>
{
    GSOBalloonParam balloonParam =
balloonEx.ParseParam(e.Feature.Description);

balloonEx.ShowBalloonEx((int)e.MousePos.X,(int)e.MousePos.Y,balloonParam);
};

//移动视角时，隐藏气泡
globeControl1.CameraBeginMoveEvent += (sender, args) =>
{
    balloonEx.HideBalloon();
};
```

feature.Description 的属性可以按照以下格式填充：

```
<!--
<BALLOON>
<ROUNDED_CX>20</ROUNDED_CX>
<ROUNDED_CY>20</ROUNDED_CY>
<CONTENT_CX>450</CONTENT_CX>
<CONTENT_CY>350</CONTENT_CY>
<MARGIN_CY>24</MARGIN_CY>
<MARGIN_CX>24</MARGIN_CX>
<ANCHOR_MARGIN>8</ANCHOR_MARGIN>
<CONTENT_TYPE>link</CONTENT_TYPE>
</BALLOON>
-->
D:\work\DynamicChart.html
```

增强气泡还可以显示表格



增强性气泡还可以以 html 中的 table 标签来显示模型的属性信息。Html 框架文件如下：

```
<html>
<head>
<style>
#tab-list {
border-collapse:collapse;
font-size:15px;
margin:20px;
text-align:left;
width:280px;
}

#tab-list th {
border-bottom:2px solid #6678B1;
color:#003399;
font-size:14px;
font-weight:normal;
padding:10px 8px;
}
#tab-list td {
```

```
border-bottom:1px solid #CCCCCC;
color:#666699;
padding:6px 8px;
}
</style>
</head>
<body style="border:none">
<center>
<table id="tab-list">
<thead><tr><th>属性名称</th><th>属性值</th></tr></thead>
<tbody>$tablecontent</tbody></table>
</center>
</body>
</html>
```

其中<tbody></tbody>标签中的$tablecontent 即为模型的属性信息。模型的属性信息可以通过以下的方式从数据库中获取：

```
string sql = "select * from " + TableName + " where ModelID=" +
e.Feature.Name;
//构建sql语句，通过元素名称来查询模型的属性信息，语句中的TableName即为数据库中存储模型信息的表名。

string htmltable = ExecuteBallonText(sql);
//调用ExecuteBallonText方法，获取模型的属性信息并储存到htmltable中。
ExecuteBallonText方法的实现在下面会有介绍。
string txt = TableCSS;
//TableCSS即为上述的html框架文件中的代码。
txt = txt.Replace("$tablecontent", htmltable);
//将html框架中的$tablecontent替换成htmltable。
balloonEx.ShowBalloon((int)e.MousePos.X, (int)e.MousePos.Y,txt);
//显示气泡
```

```
public string ExecuteBallonText(string sql)
{
    try
    {
        if (connection.State != ConnectionState.Open)
            connection.Open();
        OleDbCommand aCommand = new OleDbCommand(sql, connection);
        OleDbDataReader reader = aCommand.ExecuteReader();
        DataTable dt = new DataTable();
        dt.Load(reader);
        ArrayList cols = new ArrayList();
```

```
        for (int i = 0; i < dt.Columns.Count; i++)
        {
            Type tp = dt.Columns[i].DataType;
            if (tp.FullName.Contains("Byte[]"))
            {
                cols.Add(dt.Columns[i]);
            }
        } //获取模型信息的各个属性值。
        for (int j = 0; j < cols.Count; j++)
        {
            dt.Columns.Remove((DataColumn)cols[j]);
        }
        if (dt.Rows.Count <= 0)
            return "";
        string result = "";
        for (int i = 1; i < dt.Columns.Count - 1; i++)
        {
            result += "<tr><td>" + dt.Columns[i].ColumnName + "</td><td>" +
dt.Rows[0][i].ToString() + "</td><tr>";
        }//根据模型信息的各个属性值来构建html框架中的tbody中的行、列代码，并储存到
result中。
        return result;  //ExecuteBallonText方法返回result。前面说到，调用
ExecuteBallonText方法时会用htmltable接收该方法的返回值result。
    }
    catch (Exception)
    {
        return "";
    }
}
```

当然也可以自己通过其他方法来建立表格等等。

4.10.3 要素标注（Label）

LocaSpace 提供给要素的标注功能，可以通过一个引线把属性信息和要素（点、线、面、模型）关联，如下图所示：

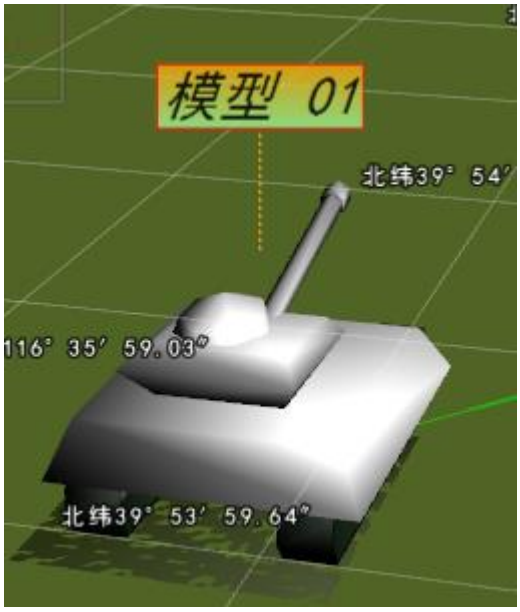


图 4-10-3-1 文字的 label

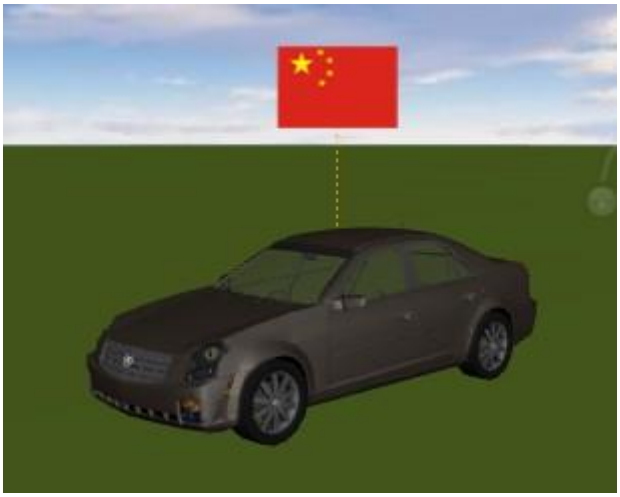


图 4-10-3-2 图片的 label

给对象的标注可以是文字，也可以是图片，还可以把文字放在图片之上。

```
GSOLabel label = new GSOLabel();
label.Text = feature.Name;  //设置显示的文本信息
label.Style = new GSOLabelStyle();
label.Style.Opaque = 0.8;  //设置标注的透明度，取值区间是0-1

//设置引线的类型，可以是实线、虚线等。
label.Style.TracktionLineType = EnumTracktionLineType.Dot;

//设置字体大小
label.Style.TextStyle.FontHeight = 20;

//设置字体类型
```

```
label.Style.TextStyle.FontName = "黑体";

//设置标注的位置，默认标注在要素的正上方，下面80，60的单位是像素
//label.Style.TractionLineEndPos = new GSOPoint2d(80, 60);
label.Style.TractionLineEndPos = new GSOPoint2d(0, 60);

//设置是否为斜体
label.Style.TextStyle.Italic = true;

//设置标注的边框颜色
label.Style.OutlineColor = Color.Red;

//设置标注的边框粗细
label.Style.OutlineWidth = 1;

//设置标注的引线粗细
label.Style.TracktionLineWidth = 1;

//设置标注矩形区的颜色渐变
label.Style.BackBeginColor = Color.Orange;
label.Style.BackEndColor = Color.PaleGreen;

//label.BKImage = @"D:\图片资源\国旗.jpg";

////该属性如果设置为空，标注只显示图片，如果不为空，那么在图片之上显示文字。
//label.Text = "";

feature.Label = label;

globeControl1.Globe.Refresh();
```

五、HUD 以及视角设置

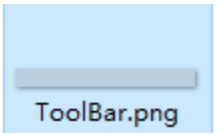
5.1 HUD

HUD 是在球界面上显示的一个界面，它集成在球上。使用快捷方便，这章节我们首先来制作一个 HUD。

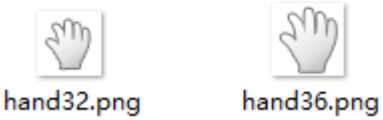
在开始制作之前，你需要了解以下知识

1、所有在 HUB 上的图标都应该是*.PNG 的，否则不能背景透明。

2、HUD 的背景大小取决于你的背景图片的大小，系统只能设置它所在的坐标。



3、对于 HUD 上的图标，准备两个大小，他们分别用于鼠标进入放大，和退出缩小。



上面的了解过后，我们开始编写 HUD，这章节可能较多，请仔细阅读：

编写 GSOHudButtonInfo 类

这个类是为了更方便快捷的将按键与事件更好的贴合在一起，并且方便的去定义 GSOHudButton。

这个类文件可以在示例文件中《相关类》文件夹中找到，此类是此方法的必须类。

创建全局变量

```
//建立字典，把按钮跟事件对应
Dictionary<string, GSOHudButtonInfo> listQuickToolBar;
//判断鼠标是否接触按钮
bool isMouseInHudControl = false;
```

创建好了之后，开始配置按钮

```
//配置按钮属性
GSOHudButtonInfo navigate = new GSOHudButtonInfo(
    "navigate",                    //按钮名称
    EnumAction3D.ActionNull,        //按钮事件
    "/Resource/image/hand32.png",    //按钮小图标
    "/Resource/image/hand36.png",    //按钮大图标
```

```
    -180, 30,                       //按钮左上角的坐标
"浏览对象");                    //按钮文字
    GSOHudButtonInfo select = new GSOHudButtonInfo(
    "select",
    EnumAction3D.SelectObject,
    "/Resource/image/select32.png",
    "/Resource/image/select36.png",
    -140, 30,
    "选中对象");
GSOHudButtonInfo line = new GSOHudButtonInfo(
    "line",
    EnumAction3D.DrawPolyline,
    "/Resource/image/Polyline32.png",
    "/Resource/image/Polyline36.png",
    -100, 30,
    "绘制线");
GSOHudButtonInfo polygon = new GSOHudButtonInfo(
    "polygon",
    EnumAction3D.DrawPolygon,
    "/Resource/image/Polygon32.png",
    "/Resource/image/Polygon36.png",
    -60, 30,
    "绘制面");
GSOHudButtonInfo showFrom = new GSOHudButtonInfo(
    "showFrom",
    EnumAction3D.ActionNull,
    "/Resource/image/Model32.png",
    "/Resource/image/Model36.png",
    180, 30,
    "显示窗口");
//Hud背景
GSOHudButtonInfo toolBar = new GSOHudButtonInfo(
    "toolbar",
    EnumAction3D.ActionNull,
    "/Resource/image/ToolBar.png",
    "/Resource/image/ToolBar.png",
    -195, 25,
    "");
//添加进HudControl中
globeControl1.Globe.AddHudControl(navigate);
globeControl1.Globe.AddHudControl(select);
globeControl1.Globe.AddHudControl(line);
globeControl1.Globe.AddHudControl(polygon);
```

```
globeControl1.Globe.AddHudControl(showFrom);
globeControl1.Globe.AddHudControl(toolBar);
//添加进字典中
listQuickToolBar = new Dictionary<string, GSOHudButtonInfo>();
listQuickToolBar.Add(navigate.Name, navigate);
listQuickToolBar.Add(select.Name, select);
listQuickToolBar.Add(line.Name, line);
listQuickToolBar.Add(polygon.Name, polygon);
listQuickToolBar.Add(showFrom.Name, showFrom);
listQuickToolBar.Add(toolBar.Name, toolBar);
```

到此，我们已经能够在界面上看到 Hud 了，但是还没有鼠标滑过和点击事件，接下来，我们来写点击事件：

```
//鼠标按下事件
globeControl1.HudControlMouseDownEvent += (sender, e) =>
{
    if (e.HudControl != null)   //如果按钮不为空
    {
        GSOHudButton button = e.HudControl as GSOHudButton;
        if (button == null)
        {
            return;
        }
        //在字典中找到按钮
        if (listQuickToolBar.ContainsKey(button.Name) == true)
        {
            GSOHudButtonInfo buttonInfo = listQuickToolBar[button.Name];
            if (buttonInfo != null)
            {
                //把字典中的动作赋给球
                globeControl1.Globe.Action = buttonInfo.action;
                //判断名称，显示窗口
                if (buttonInfo.Name == "showFrom")
                {
                    MessageBox.Show("显示窗口");
                }
            }
        }
    }
```

```
};
```

现在我们点击，可以实现功能了，但是鼠标滑过图标没有反应，还得添加鼠标滑过的事件。在此之前，我们需要先写一个恢复控件大小的方法：

```
/// <summary>
/// 当鼠标移出，缩小图标并隐藏文字，恢复原状
/// </summary>
private void setButtonTextAndImageEmpty()
{
    if (listQuickToolBar != null)
    {
        foreach (string key in listQuickToolBar.Keys)
        {
            GSOHudButtonInfo button = listQuickToolBar[key];
            if (button != null)
            {
                button.SetImage(Application.StartupPath + button.imagePath);
                button.Text = "";
            }
        }
    }
}
```

```
//鼠标移入事件
globeControl1.HudControlMouseIntoEvent += (sender, e) =>
{
    if (e.HudControl != null)
    {
        isMouseInHudControl = true;
        //先恢复默认
        setButtonTextAndImageEmpty();
        GSOHudButton button = e.HudControl as GSOHudButton;
        if (button == null)
        {
            return;
        }
        if (listQuickToolBar.ContainsKey(button.Name) == true)
        {
            GSOHudButtonInfo buttonInfo = listQuickToolBar[button.Name];
            if (buttonInfo == null)
            {
                return;
```

```
            }
            //改变鼠标滑过的按钮图标和文字
            button.SetImage(Application.StartupPath +
buttonInfo.bigImagePath);
            button.Text = " " + buttonInfo.toolTip;
        }
    }
};

//鼠标移出事件
globeControl1.HudControlMouseOutEvent += (sender, e) =>
{
    if (e.HudControl != null)
    {
        isMouseInHudControl = false;
        GSOHudButton button = e.HudControl as GSOHudButton;
        if (button == null)
        {
            return;
        }
        //恢复原样
        setButtonTextAndImageEmpty();
    }
};
```

这时运行一下，感受一下 Hud 吧。

上面这个是具体的示例，我们开始更加详细的来了解 HUD 吧！

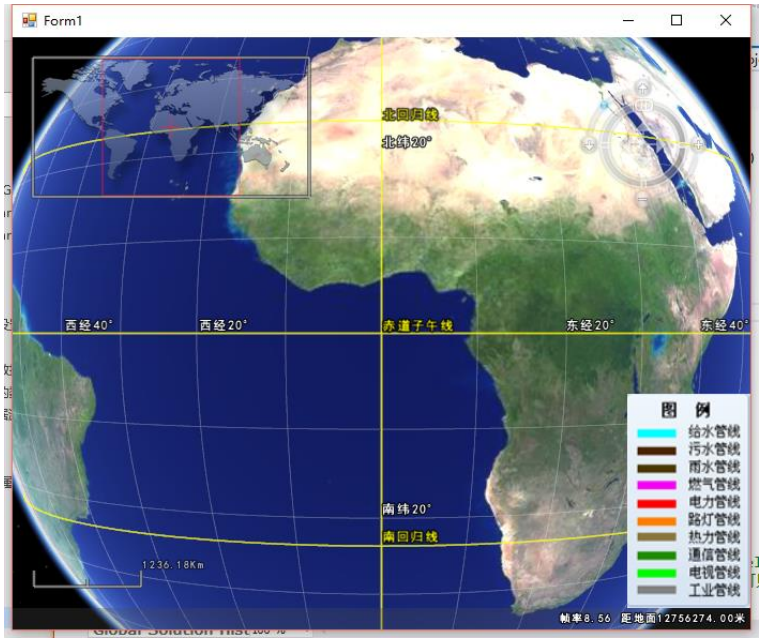
5.1.1 GSOHudButton

在三维场景中 GSOHudButton 控件，使用方法如下：

```
/// <summary>
///HUDButton示例
/// </summary>
private void GSOHUDBUTTON()
{
    GSOHudButton btn = new GSOHudButton();//创建GSOHudButton对象
    btn.SetImage(Application.StartupPath + "/Resource/image/t1.png");//设置图
片路径，图片格式可以是png，jpg或ico。
    btn.Align = EnumAlign.BottomRight;
```

```
    //设置图片对齐方式为右下角对齐，EnumAlign成员分别有
BottomCenter,BottomLeft,BottomRight,MiddleCenter,MiddleLeft,MiddleRight,TopCe
nter,TopLeft,TopRight。
    btn.SetOffset(0, 20);//设置图片相对于场景右下角的距离。
    btn.MinOpaque = 1;//通过MaxOpaque、MinOpaque这两个属性设置GSOHudButton对象的
透明度，其值为0-1。
    btn.MaxOpaque = 1;
    btn.Height = 200;//设置btn高度
    btn.Width = 115;//设置宽度
    btn.HeightFixed = true; //设置GSOHudButton对象的图片高度与该对象的高度相同
    btn.WidthFixed = true; //设置GSOHudButton对象的图片宽度与该对象的宽度相同
    globeControl1.Globe.AddHudControl(btn);//将GSOHudButton对象添加到场景中
}
```

实际效果如下：



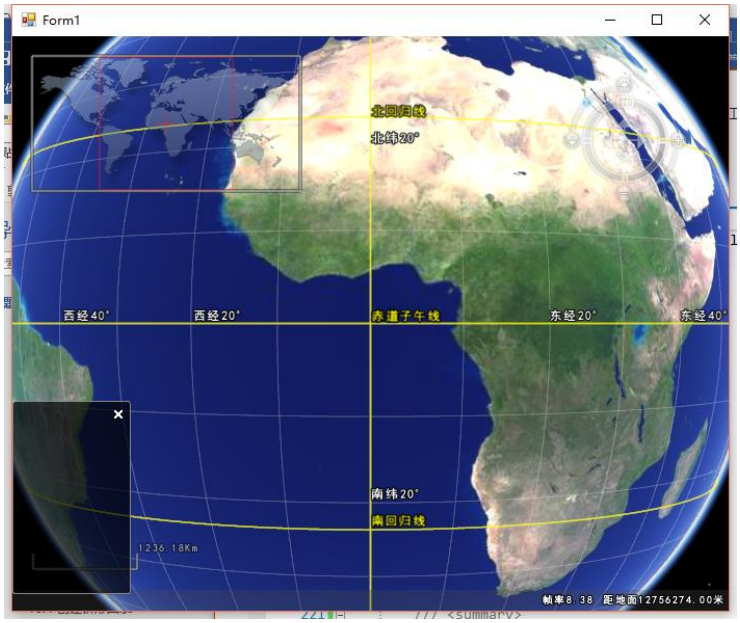
5.1.2 GSOHudPanel

在场景中添加 GSOHudPanel 控件，使用方法如下：

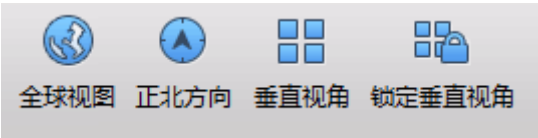
```
/// <summary>
/// GSOHUDPanel示例
/// </summary>
private void GSOHUDPANEL()
{
    GSOHudPanel panel = new GSOHudPanel();//创建GSOHudPanel对象
    panel.CloseButtonVisible = true;//设置panel的关闭按钮可见
```

```
    panel.FadeOut = true;//设置panel的淡出效果
    panel.MaxOpaque = 1;
    panel.MinOpaque = 1;
    panel.Height = 187;//设置panel的高度
    panel.Width = 115;//设置panel的宽度
    panel.Align = EnumAlign.BottomLeft;//设置panel对齐方式为右下角对齐
    panel.SetOffset(0, 15);//设置panel相对于场景右下角的距离
    globeControl1.Globe.AddHudControl(panel);//在场景中添加panel
}
```

实际效果如下：



5.2 视角设置



视角设置，是上面 4 个按键使用的基础，也是用户在三维球上进行“跳转”的关键，我们先从核心的讲起，之后对于视角的控制就是非常简单了：

首先，用户在软件中所看到的视角可以比作是用一个摄像机从一个视角拍摄下来的，而视角的移动就是摄像机的移动，这样我们就可以创建一个相机位置对象：

```
//新建一个摄像机状态对象
GSOCameraState camera = new GSOCameraState();
camera.Altitude = 20000;    //设置视点离地面的垂直距离
//设置视点的高度模式，支持三种模式，绝对高度，相对地面高度，贴地
```

```
camera.AltitudeMode = EnumAltitudeMode.ClampToGround;
//设置视点高度，真实高度，不可与Altitude、AltitudeMode公用
camera.Distance = 20000;
//视线的方向与正北的夹角，0度表示正北，90度表示正东，180度表示正南
camera.Heading = 0;
camera.Latitude = 500;  //设置视点的经度
camera.Longitude = 500; //设置视点的纬度
//视线与铅垂线的夹角，0度表示垂直向下看，90度表示沿水平方向看
camera.Tilt = 0;
```

视角设置完了之后，我们开始把当前视角移动到设置好的视角位置。这时候，我们有两种选择：

```
globeControl1.Globe.FlyToCameraState(camera);
```

```
globeControl1.Globe.JumpToCameraState(camera);
```

那么这两种的区别：

FlyToCameraState()是飞到目标视角，有一个飞的过程动画，过程平缓过渡。

JumpToCameraState()是跳到目标视角，没有飞的过程，直接显示目标视角。

这两种方法中 FlyToCameraState()更平滑顺畅，JumpToCameraState()更直接快速。至于选择用哪个，自己选择吧。

原理讲明白了，接下来我们来开始准备刚开始我们遗留下来的问题吧。

5.3.1 全球视角

全球视角，就是跳转到拉远距离的视点中：

```
GSOCameraState camestate = new GSOCameraState();
camestate.Longitude = 109.07286491474;
camestate.Latitude = 37.3716091749733;
camestate.Distance = 20732473.52;
globeControl1.Globe.JumpToCameraState(camestate);
```

5.3.2 正北方向

```
//设置视角为当前视角，不会改变经纬度
```

```
GSOCameraState camestate = globeControl1.Globe.CameraState;
camestate.Heading = 0.0;
globeControl1.Globe.JumpToCameraState(camestate);
```

5.3.3 垂直视角

```
//设置视角为当前视角，不会改变经纬度
GSOCameraState camestate = globeControl1.Globe.CameraState;
camestate.Tilt = 0.0;
globeControl1.Globe.JumpToCameraState(camestate);
```

5.3.4 锁定垂直视角

首先我们先建立一个识别是否锁定的全局变量

```
bool isLockVerticalCamera = false;
```

然后在窗口 Load()方法中加入事件

```
globeControl1.BeforeSceneRenderEvent += (sender, e) =>
{
    if (isLockVerticalCamera)
    {
        globeControl1.Globe.SetCenterTilt(0.0);
    }
};
```

随后在 button 上添加就可以了

```
//锁定垂直视角
private void button6_Click(object sender, EventArgs e)
{
    isLockVerticalCamera = !isLockVerticalCamera;
}
```

运行之后感受一下吧。

5.3.5 地上、地下、行走视角

5.3.5.1 地上视角

地上视角，就是我们普通球初始化之后的视角，可以随意拖动以及放大缩小，操作模型的视角：

```
/// <summary>
/// 地上视角
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button7_Click(object sender, EventArgs e)
{
    //如果之前是地下视角，则重新设置视角，如果不是，就只改变状态
    if (globeControl1.Globe.CameraMode == EnumCameraMode.UnderGround)
    {
        //视角变更为普通
        globeControl1.Globe.CameraMode = EnumCameraMode.Navigation;
        //把当前的摄像机状态复制
        GSOCameraState state = globeControl1.Globe.CameraState;
        //设定俯仰角度
        if (globeControl1.Globe.CameraState.Tilt < 95 &&
globeControl1.Globe.CameraState.Tilt > 85)
        {
            state.Tilt = 85;
        }
        else
        {
            state.Tilt = 180 - globeControl1.Globe.CameraState.Tilt;
        }
        //跳转
        globeControl1.Globe.JumpToCameraState(state);
    }
    else
    {
        globeControl1.Globe.CameraMode = EnumCameraMode.Navigation;
    }
}
```

5.3.5.2 行走视角

行走视角，是系统模拟人在球上行走的浏览视角。此视角不能用鼠标在球上操作，只能通过 WSAD 进行行走，以及←→进行视角的调整。

```
globeControl1.Globe.CameraMode = EnumCameraMode.Walk;
```

5.3.5.3 地下视角

地下视角，是为了方便观察地下的管道以及建筑等设计的视角：

```
/// <summary>
/// 地下视角
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button9_Click(object sender, EventArgs e)
{
    globeControl1.Globe.CameraMode = EnumCameraMode.UnderGround;
    GSOCameraState state = globeControl1.Globe.CameraState;
    if (globeControl1.Globe.CameraState.Tilt < 95 &&
globeControl1.Globe.CameraState.Tilt > 85)
    {
        state.Tilt = 95;
    }
    else
    {
        state.Tilt = 180 - globeControl1.Globe.CameraState.Tilt;
    }
    globeControl1.Globe.JumpToCameraState(state);
}
```

5.3 地图全屏

地图全屏，是为了更好的让用户演示地球，去除所有无用的界面。

在最开始，我们先定义三个全局变量用来保存信息。

```
//保存现在是否已经全屏的状态
bool isFullScreen = false;
//保存未全屏之前的窗口大小坐标信息
Rectangle rectangleOld = new Rectangle(0, 0, 0, 0);
//保存未全屏之前，窗口的显示信息
FormWindowState fws;
```

接下来，我们引用系统对于全屏窗口的应用方法。

```
/// <summary>
/// 设置全屏或取消全屏
/// </summary>
/// <param name="fullscreen">true:全屏 false:恢复 </param>
/// <param name="rectOld">设置的时候，此参数返回原始尺寸，恢复时用此参数设置恢复
</param>
/// <returns>设置结果 </returns>
public static bool SetFullScreen(bool fullscreen, ref Rectangle rectOld)
{
    if (fullscreen)
    {
        Rectangle rectFull = Screen.PrimaryScreen.Bounds;
        SystemParametersInfo(SPI_GETWORKAREA, 0, ref rectOld,
SPIF_UPDATEINIFILE);//get
        SystemParametersInfo(SPI_SETWORKAREA, 0, ref rectFull,
SPIF_UPDATEINIFILE);//set
    }
    else
    {
        SystemParametersInfo(SPI_SETWORKAREA, 0, ref rectOld, SPIF_UPDATEINIFILE);
    }
    return true;
}

#region 引用系统DLL

[DllImport("user32.dll", EntryPoint = "ShowWindow")]
private static extern int ShowWindow(int hWnd, int _value);
[DllImport("user32.dll", EntryPoint = "SystemParametersInfo")]
private static extern int SystemParametersInfo(
        int uAction,
        int uParam,
```

```
        ref Rectangle lpvParam,
        int fuWinIni
        );

public const int SPI_SETWORKAREA = 47;
public const int SPI_GETWORKAREA = 48;
public const int SW_HIDE = 0;
public const int SW_SHOW = 5;
public const int SPIF_UPDATEINIFILE = 0x0001;


#endregion
```

这时候任务栏等系统界面已经隐藏，但是软件界面还是没有变化，接下来我们配置全屏方法：

```
/// <summary>
/// 开始全屏
/// </summary>
private void FullScreen()
{
    SetFullScreen(!isFullScreen, ref rectangleOld);
    //是否在任务栏显示图标
    this.ShowInTaskbar = isFullScreen;
    //判断是否为全屏，是全屏就隐藏（这里需要把所有控件隐藏）
    panel1.Dock = isFullScreen == false ? DockStyle.Fill : DockStyle.Left;
    button1.Visible = isFullScreen;
    //临时挂起布局逻辑
    this.SuspendLayout();
    if (!isFullScreen)
    {
        fws = this.WindowState;
        this.WindowState = FormWindowState.Maximized;
        this.FormBorderStyle = FormBorderStyle.None;
    }
    else
    {
        this.FormBorderStyle = FormBorderStyle.FixedSingle;
        globeControl1.Globe.SetStereoEnable(false);
        this.WindowState = fws;
    }
    this.ResumeLayout(false);
    isFullScreen = !isFullScreen;
    this.Focus();
```

```
}
```

这时候我们只需要给 button1 添加 FullScreen()方法就可以了，但是你会发现没有控件来返回界面，这时候我们有两种方法，第一是用一个永远最前端的窗口来恢复，或者就注册键盘事件，下面我们的方法是注册一个键盘事件：

```
/// <summary>
/// 按键盘开始全屏
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.KeyCode)
    {
        case Keys.F5:
            FullScreen();
            break;
            //Esc仅用于退出全屏
        case Keys.Escape:
            if (isFullScreen)
            {
                FullScreen();
            }
            break;
        default:
            break;
    }
}
```

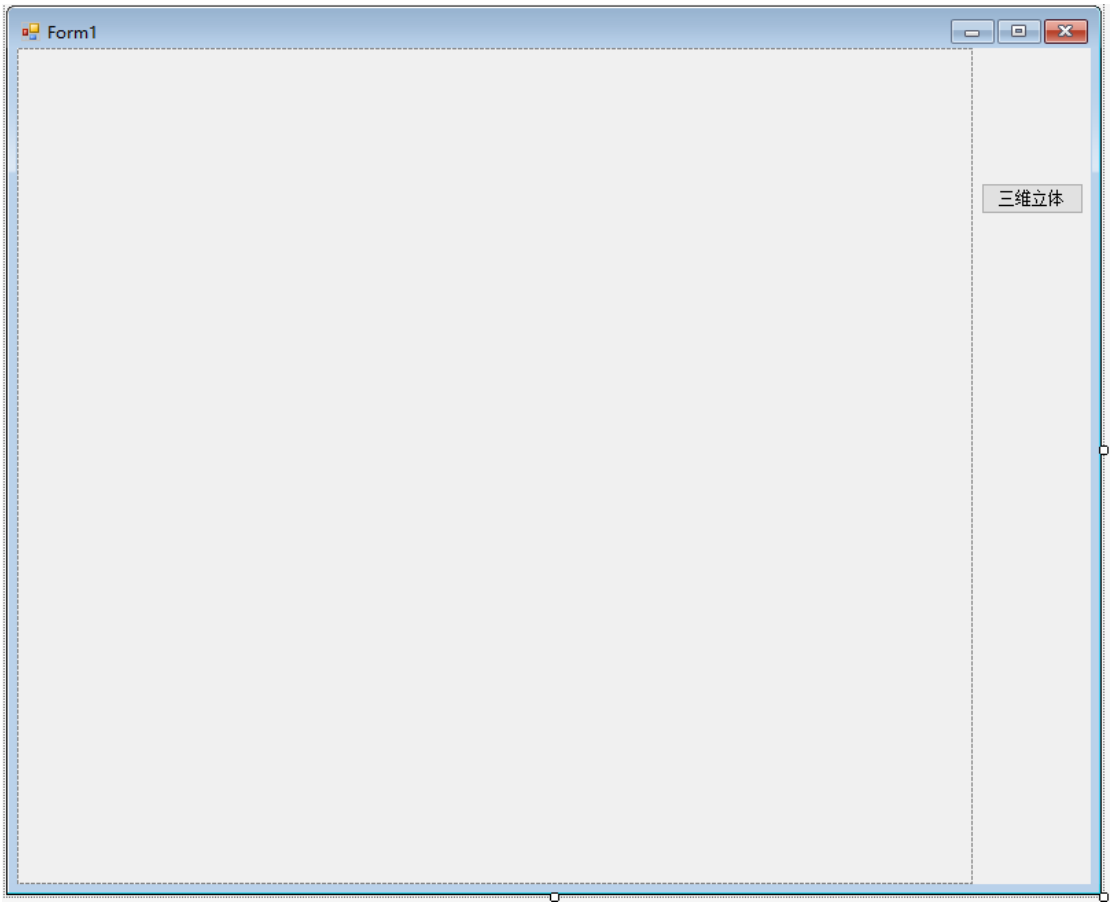
到这里，别忘了把窗口属性 KeyPreview 改为 true。

这时候运行程序就会看到全屏之后的三维球了。

5.4 三维立体

三维立体是为了迎合 VR 和 3D 电视而推出的一项功能，他可以让三维球变成两个视图，可以从而达到三维立体的效果。

首先，创建一个 panel，用来分开按键和三维球。具体如图：



然后代码方面我们要初始化 GlobeControl 控件。

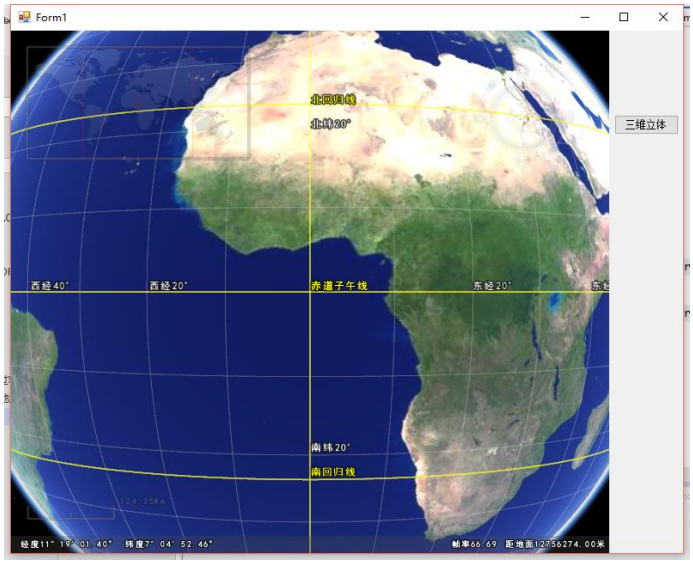
```
GSOGlobeControl globeControl1;
```

在窗体构建的时候，我们还得稍微改一下。

```
public Form1()
{
    InitializeComponent();

    //初始化两个控件
    globeControl1 = new GSOGlobeControl();
    splitContainer1.Panel1.Controls.Add(globeControl1);
```

```
    globeControl1.Dock = DockStyle.Fill;
}
```

我们现在可以运行一下，可以看到，跟我们之前创建的一样，还是一个球。



接下来我们还是得用到《地图全屏》里面的全屏事件，用来填充整个屏幕。

添加一个全局变量，用来记录是否开启 3D。

```
bool is3D = false;
```

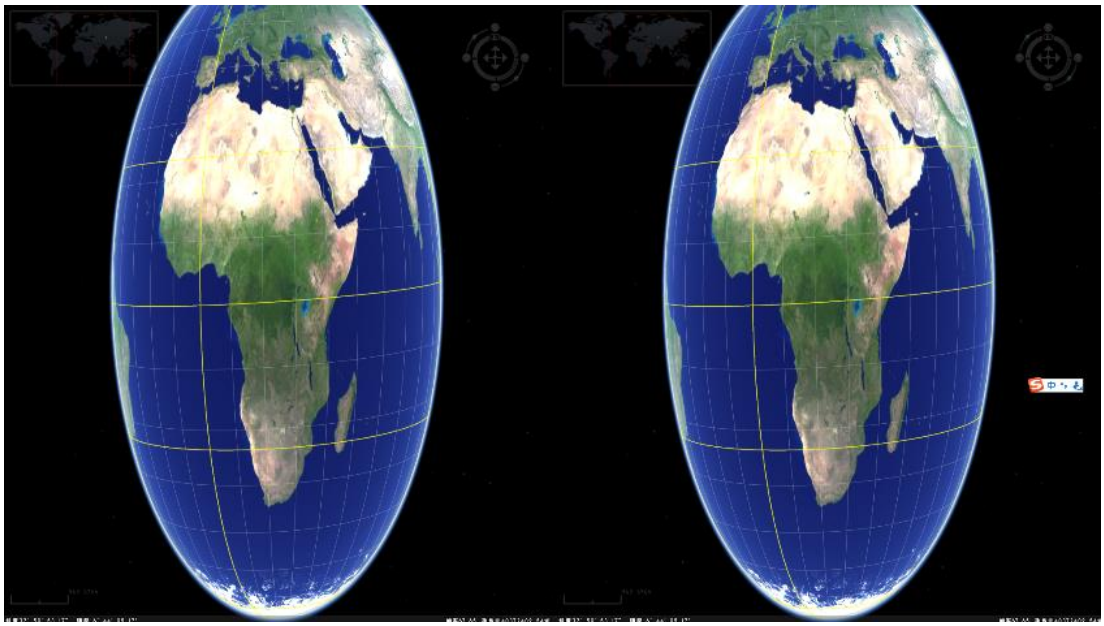
```
/// <summary>
/// 三维立体
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>

private void button1_Click(object sender, EventArgs e)
{
    is3D = !is3D;
    FullScreen();
    globeControl1.Globe.SetStereoEnable(is3D);
    globeControl1.Globe.SetStereoMode(EnumStereoMode.HORIZONTAL_SPLIT);
}
```

这时候运行一下，可以看到效果，但是退出不了？所以我们还得加上键盘注册事件来进行关闭效果。

```
/// <summary>
/// 键盘注册
/// </summary>
```

```
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    switch (e.KeyCode)
    {
        case Keys.F5:
            FullScreen();
            is3D = !is3D;
            break;
        //Esc仅用于退出全屏
        case Keys.Escape:
            if (isFullScreen)
            {
                FullScreen();
                is3D = !is3D;
            }
            break;
        default:
            break;
    }
}
```

现在我们就可以运行一下看看效果了！



5.5 飞行模式

5.5.1 沿线飞行

沿线飞行，是沿着画好的线飞行的一个方法，他有两种模式：

- FlyAlongWithLine

  飞行的时候可以用鼠标滚轮在自己调整视角。

```
/// <summary>
/// FlyAlongWithLine
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, EventArgs e)
{
    if (feature.Geometry.Type == EnumGeometryType.GeoPolyline3D)
    {
        GSOGeoPolyline3D line3d = feature.Geometry as GSOGeoPolyline3D;

        globeControl1.Globe.FlyAlongLineRotateSpeed = 2000;  //过弯速度
        globeControl1.Globe.FlyAlongLineSpeed = 500;       //飞行速度
        //globeControl1.Globe.FlyToPointSpeed = 5000;         //飞到点速度

        double dHeightAboveLine = 100;  //高于线
        double dHeading = 0;            //视角夹角
        double dTilt = 80;                  //水平夹角

globeControl1.Globe.FlyAlongWithLine(line3d,dHeightAboveLine,dHeading,dTilt);
    }
}
```

- FlyEyeAlongWithLine

  飞行的时候视角为垂直飞行，且不可调整视角。

```
/// <summary>
/// FlyEyeAlongWithLine
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{
    if (feature.Geometry.Type == EnumGeometryType.GeoPolyline3D)
```

```
    {
        GSOGeoPolyline3D line3d = feature.Geometry as GSOGeoPolyline3D;

        double dHeightAboveLine = 100;  //高于线
        double dHeading = 0;            //视角夹角
        double dTilt = 80;                  //水平夹角

globeControl1.Globe.FlyEyeAlongWithLine(line3d,dHeightAboveLine,0,true,0,fals
e);
    }
}
```

5.5.2 环绕飞行

- **绕中心点飞行**

绕着屏幕中心点进行飞行：

```
/// <summary>
/// 绕中心点飞行
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button6_Click(object sender, EventArgs e)
{
    globeControl1.Globe.FlyAroundCenter();
}
```

- 绕视角飞行

```
/// <summary>
/// 绕视角飞
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button7_Click(object sender, EventArgs e)
{
    globeControl1.Globe.FlyAroundEye();
}
```

- 绕要素飞行

```
/// <summary>
/// 绕要素飞行
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button8_Click(object sender, EventArgs e)
{
    globeControl1.Globe.FlyAroundFeature(feature, true, 100,
EnumFlyRepeatValueType.Degrees);
}
```

5.5.2 飞行控制

暂停飞行：

```
globeControl1.Globe.PauseFly();
```

继续飞行：

```
globeControl1.Globe.ContinueFly();
```

停止飞行：

```
globeControl1.Globe.StopFly();
```

5.6 界面文字

界面文字是在项目中常用的功能，他可以用作水印文字，也可以用作提示用户的鼠标跟随文字。这一节我们就添加这两种界面文字：

5.6.1 水印文字

首先，水印文字是停放在三维球界面的文字，它其实就是一个要素（GSOFeature），为了后期方便管理，我们一般都创建一个文字图层，保存为 lgd。

首先先初始化一个图层，这个图层路径为空也没有关系。

```
//文字图层地址
```

```
string ScreenTextLayerPath = Application.StartupPath +
"/Resource/layers/ScreenText.lgd";
//文字图层
GSOLayer layerScreenText = null;
```

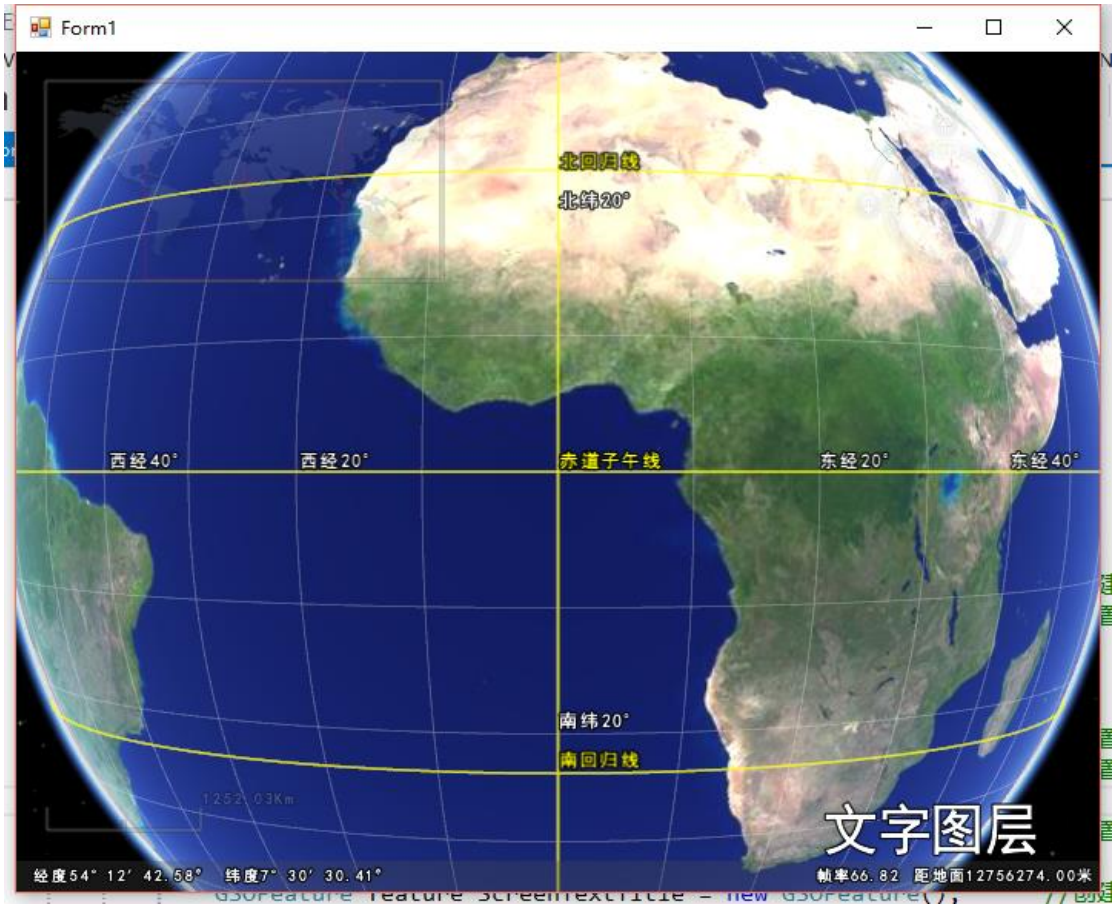
接着，我们在 Form1_Load 里面初始化一下图层文件：

```
//如果没有文件，就将内存图层复制一份
if (File.Exists(ScreenTextLayerPath) == false)
{
    globeControl1.Globe.MemoryLayer.SaveAs(ScreenTextLayerPath);
}
layerScreenText = globeControl1.Globe.Layers.Add(ScreenTextLayerPath);
```

之后，我们开始加载屏幕文字：

```
if (layerScreenText != null)
{
    //清除其中所有要素
    layerScreenText.RemoveAllFeature();
    //创建屏幕文字
    GSOGeoScreenText overlayTextTitle = new GSOGeoScreenText();
    GSOTextStyle textStyle = new GSOTextStyle();                //设置属性
    textStyle.ForeColor = Color.White;
    textStyle.FontSize = 36;
    overlayTextTitle.TextStyle = textStyle;
    overlayTextTitle.Align = EnumAlign.TopLeft;          //设置文字对齐方式
    overlayTextTitle.PosAlign = EnumAlign.BottomRight;    //设置文字位置
    overlayTextTitle.Name = "ScreenTextTitle";
    overlayTextTitle.SetOffset(180, 60);                  //设置文字偏移量
    overlayTextTitle.Text = text;                        //设置文字
    GSOFeature feature_ScreenTextTitle = new GSOFeature();      //创建要素
    feature_ScreenTextTitle.Geometry = overlayTextTitle;        //赋予要素
    layerScreenText.AddFeature(feature_ScreenTextTitle);
    layerScreenText.Save();
}
```

这样，你就会发现，在三维球的右下角出现了图层文字。



5.6.2 鼠标跟随文字

在鼠标跟随文字中，一般是用作功能说明，首先，我们初始化：

```
private GSOGeoScreenText overlayText = null;
private GSOFeature feature_ScreenText = null;
```

在跟之前，初始化一个文字图层之后，我们就可以向其中添加鼠标跟随文字：

```
//如果要素存在就删除
if (feature_ScreenText != null)
{
    feature_ScreenText.Delete();
}
if (text != "" && layerScreenText != null)
{
    overlayText = new GSOGeoScreenText();           //创建对象
    GSOTextStyle textStyle = new GSOTextStyle();    //创建样式
    textStyle.ForeColor = Color.White;              //字体颜色
    textStyle.FontSize = 9;                         //字体大小
```

```
    overlayText.TextStyle = textStyle;                //赋予字体样式
    overlayText.Align = EnumAlign.TopLeft;            //字体对其方式
    overlayText.Name = "ScreenText";                  //字体对象名称
    overlayText.SetOffset(20, 0);                     //字体偏移量
    overlayText.Text = text;                          //显示文字
    feature_ScreenText = new GSOFeature();            //创建要素
    feature_ScreenText.Geometry = overlayText;        //赋予要素
    layerScreenText.AddFeature(feature_ScreenText); //添加要素到图层
}
```

跟屏幕水印文字不同的是，我们需要添加一个鼠标移动事件，一边让文字跟踪鼠标：

```
void globeControl1_MouseMove(object sender, MouseEventArgs e)
{
    if (overlayText != null && overlayText.Text != "" &&
feature_ScreenText != null)
    {
        overlayText.SetOffset(e.X + 20, e.Y);
        feature_ScreenText.Geometry = overlayText;
        globeControl1.Refresh();
    }
}
```

这样的话，文字就会跟着鼠标移动，一起试一试吧~！

六、绘制模式

绘制是三维球中最常用的操作，在三维球中，用户可以用自己的鼠标在三维球中绘制点、线和面。

在我们的三维球中，系统底层已经为大家添加了绘制方法，用户只需要将鼠标的动作开启即可直接绘制：

6.1 绘制线

```
globeControl1.Globe.Action = EnumAction3D.DrawPolyline;
```

6.2 绘制面

```
globeControl1.Globe.Action = EnumAction3D.DrawPolygon;
```

6.3 绘制点

绘制点就比较特殊一点，首先改变鼠标的动作为 NormalHit：

```
globeControl1.Globe.Action = EnumAction3D.NormalHit;
```

然后需要添加鼠标放下和鼠标抬起的事件：

```
//鼠标按下事件，获取坐标
globeControl1.MouseDown += (sender, e) =>
{
    Location_MouseDown = e.Location;
};

//鼠标抬起事件
globeControl1.MouseUp += (sender, e) =>
{
    //鼠标抬起如果坐标不能对上，就认定为拖动球，不反应
    if (Location_MouseDown != e.Location)
        return;

    if (e.Button == MouseButtons.Left)
    {
        //如果按照上面示例按我们方法添加HUD，需要加上这个判断，判断是否点在HUD中
        //if (isMouseInHudControl == false)
        {
            if (globeControl1.Globe.Action == EnumAction3D.NormalHit)
            {
                GSOPoint3d point3d = new GSOPoint3d();

                GSOFeature newFeature = new GSOFeature();
                GSOGeoMarker p = new GSOGeoMarker();
                GSOMarkerStyle3D style = new GSOMarkerStyle3D();
                style.IconPath = Application.StartupPath +
"/Resource/image/DefaultIcon.png";
                p.Style = style;
                p.Z = point3d.Z <= 0 ? 0 : point3d.Z;
```

```
                if (point3d.Z <= 0.0)
                {
                    point3d = globeControl1.Globe.ScreenToScene(e.X, e.Y);
                }
                p.X = point3d.X;
                p.Y = point3d.Y;
                p.Z = point3d.Z;
                newFeature.Geometry = p;
                newFeature.Name = "我的地标";

                globeControl1.Globe.MemoryLayer.AddFeature(newFeature);
                globeControl1.Globe.Action = EnumAction3D.ActionNull;
            }
        }
    }
};
```

七、工程文件的打开和保存

工程文件也叫工作空间文件，其扩展名为.gws，它能够将当前场景保存下来，保存所有图层、地形和场景的属性设置。打开场景文件可恢复场景保存前的场景面貌。

7.1 打开工程文件

```
//获得文件地址
string path = Application.StartupPath + "/Resource/GWS/1.gws";
//先关闭当前工程
if (!CloseWorkSpace()) return;
//打开工程文件
globeControl1.Globe.OpenWorkSpace(path);
//刷新球
globeControl1.Refresh();
```

7.2 关闭工程文件并保存

```
/// <summary>
/// 关闭工程
/// </summary>
/// <returns>执行成功返回真否则返回假</returns>
private bool CloseWorkSpace()
{
    if (globeControl1.Globe.Layers.Count <= 0 &&
globeControl1.Globe.Terrains.Count <= 0) return true;
    var result = MessageBox.Show("是否保存数据?", "提示",
MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
    if (result != DialogResult.Yes) return false;
    SaveAllData();
    ClearWorkSpace();
    return true;
}

/// <summary>
/// 保存数据
/// </summary>
```

```
private void SaveAllData()
{
    //遍历图层，保存数据
    for (var i = 0; i < globeControl1.Globe.Layers.Count; i++)
    {
        var layer = globeControl1.Globe.Layers[i];
        if (layer != null)
        {
            //保存图层数据
            layer.Save();
        }
    }
}
```

另外，在我的地标下的要素都是属于“内存图层”，保存的时候直接保存到*.KML 中：

```
//保存内存图层
globeControl1.Globe.MemoryLayer.SaveAs(
    Application.StartupPath + "/MyPlace.kml"
    );
```

打开工程的时候也打开 Kml 就可以了。

八、连接服务器

8.1 介绍

LocaSpace 产品团队通过深入分析当前技术领域主流的数据发布服务平台和云服务平台，凭借团队自身强大的自主研发能力，深度结合 GIS 和服务发布等专业技术，整合服务发布功能和资源，通过多年的技术攻关，研发出了高效率、易上手、低成本、轻量级的数据发布和云服务平台—LSVServer。

多用户同时连接服务器，支持用户与服务器一对多，多对多，多对一的连接并发操作，多用户共享数据更简单。在 LSVServer 中发布的多种类型的数据，结合 Locaspace Viewer 或其他 Locaspace 产品，无论用户的数量以及用户的在线或者离线状态，均可访问和浏览 LSVServer 所发布的数据。

在 PC 端发布服务，不同终端不同系统均可共享服务，支持 Android、Windows、IOS、Linux 同时访问，真正做到多终端支持。

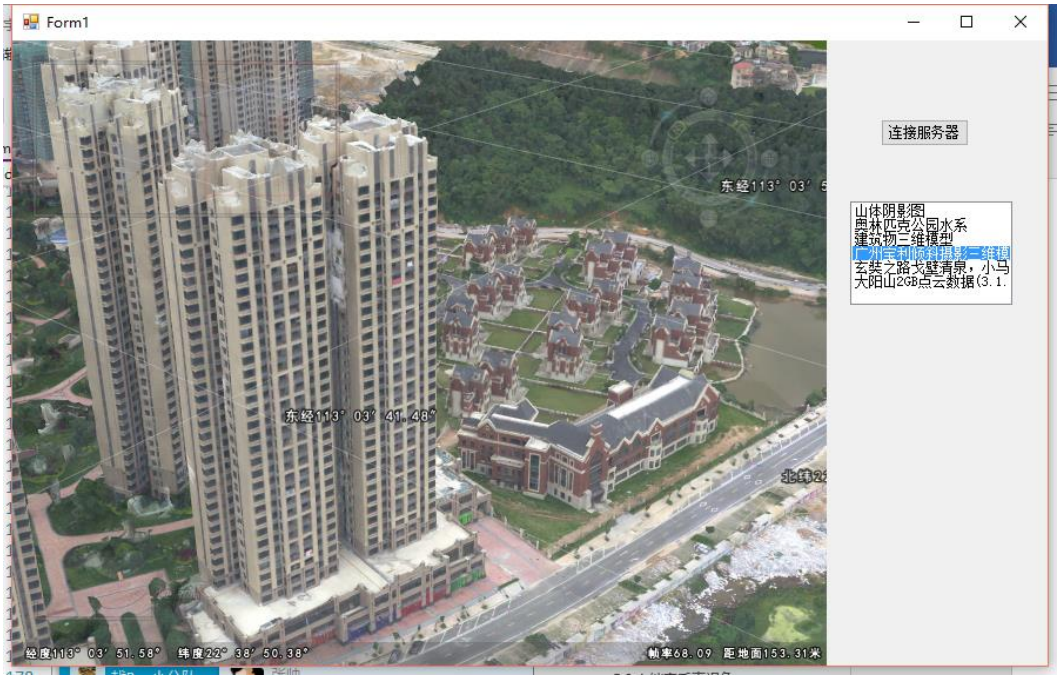
更多关于 LSVServer 功能、优势、报价的信息可拨打 400-867-5155 电话咨询。

8.2 连接服务器方法

我们本身的底层就带有服务器连接的方法：

```
//IP地址
string strIP = "data1.locaspace.cn";
//用户名
string strUser = "admin";
//密码
string strPsw = "admin";
//端口
int nPort = 1500;
//连接服务器
globeControl1.Globe.ConnectServer(strIP, nPort, strUser, strPsw);
```

连接之后就可以看到服务器上的数据了



连接服务器之后的图层是一个一个添加的，所以可以使用事件来遍历每个图层。

```
//添加图层之后
globeControl1.AfterLayerAddEvent += (sender, args) => { };
```

九、测量

测量是三维 GIS 软件中常用的功能，它只需要点点鼠标，就可以完成精确的测量工作。

9.1 三角测量

```
//更改鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureDistance;
//更改来测量动作
globeControl1.Globe.DistanceRuler.MeasureMode = EnumDistanceMeasureMode.HVSLineMeasure;
```

9.2 高度测量

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureHeight;
//变更测量模式
globeControl1.Globe.HeightRuler.SpaceMeasure = true;
```

9.3 测量地表距离

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureDistance;
//变更测量模式
globeControl1.Globe.DistanceRuler.MeasureMode=EnumDistanceMeasureMode.GroundPolylineMeasure;
```

9.4 测量空间距离

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureDistance;
//变更测量模式
globeControl1.Globe.DistanceRuler.MeasureMode=EnumDistanceMeasureMode.SpacePolylineMeasure;
```

9.5 测量投影距离

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureDistance;
//变更测量模式
globeControl1.Globe.DistanceRuler.MeasureMode = EnumDistanceMeasureMode.HoriLineMeasure;
```

9.6 测量地表面积

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureArea;
//变更测量模式
globeControl1.Globe.AreaRuler.MeasureMode = EnumAreaMeasureMode.GroundAreaMeasure;
```

9.7 测量空间面积

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureArea;
//变更测量模式
globeControl1.Globe.AreaRuler.MeasureMode = EnumAreaMeasureMode.SpaceAreaMeasure;
```

9.8 测量投影面积

```
//改变鼠标动作
globeControl1.Globe.Action = EnumAction3D.MeasureArea;
//变更测量模式
globeControl1.Globe.AreaRuler.MeasureMode = EnumAreaMeasureMode.SphereAreaMeasure;
```

9.9 清除测量

```
//清除测量
globeControl1.Globe.ClearMeasure();
//刷新球
globeControl1.Globe.Refresh();
```

十、事件机制

10.1 在线图层的添加

SDK 支持加载在线的图层，前面很多示例工程也给大家添加了几个在线图层。

我们的在线图层的地址保存在一个（*.lrc）文件里面，添加的时候只需要像添加本地图层一样添加即可。

```
string path = Application.StartupPath + "/Resource/gisdata/baidudata/百度标注.lrc";
globeControl1.Globe.Layers.Add(path);
```

如果我们打开*.lrc 文件的话，你会看到一个类似 KML 的格式：

```
<?xml version="1.0" encoding="GB18030"?><DataDefine>
<Version>0</Version>
<Name>baiduimg</Name>
<GeoGridType>4</GeoGridType>
<SampleSize>256</SampleSize>
<DataVersion></DataVersion>
<DataType>urlformat</DataType>
<LocalPath></LocalPath>
<NetPath>http://online0.map.bdimg.com/tile/?qt=tile&x=%s&y=%s&z=%d&styles=sl&v=017</NetPath>
<FileExt>png</FileExt>
<Range>
<West>-160</West>
<East>160</East>
<South>-85</South>
<North>85</North>
<LevelBegin>3</LevelBegin>
<LevelEnd>18</LevelEnd>
</Range>
</DataDefine>
```

在其中我们可以自己设定在线图层：

Name：图层名称

GeoGridType：类型

SampleSize：切分大小

DataVersion：数据版本

DataType：URL 格式

urlParam：URL 拆分顺序

NetPath：在线图层地址

FileExt：文件后缀名

Range：这其中的是对于图层偏移和等级的设定

West、East、South、North：是图层对于东南西北四个方向的范围大小

LevelBegin：地图起始级别

LevelEnd：地图终止级别

10.2 选中事件

选中事件位于第 4 章的获取要素中，示例请移步第四章。

```
//改变球为选中模式
globeControl1.Globe.Action = EnumAction3D.SelectObject;
GSOFeature myFeature = null;
myFeature = globeControl1.Globe.SelectedObject;
MessageBox.Show("获取到:" + (myFeature == null?"null":myFeature.Name));
```

10.3 键盘事件

三维球的键盘事件有两个：

```
//在用Delete键删除要素之后
globeControl1.AfterKeyDownDeleteFeatureEvent += (sender, args) => { };
//在用Delete键删除要素之后
globeControl1.BeforeKeyDownDeleteFeatureEvent += (sender, args) => { };
```

10.4 鼠标事件

```
//在要素上点击鼠标
globeControl1.FeatureMouseClickEvent += (sender, args) => { };
//鼠标在要素内保持静止一段时间
globeControl1.FeatureMouseHoverEvent += (sender, args) => { };
//鼠标进入要素内部
globeControl1.FeatureMouseIntoEvent += (sender, args) => { };
//鼠标离开要素内部
```

```
globeControl1.FeatureMouseOutEvent += (sender, args) => { };
//鼠标在要素上
globeControl1.FeatureMouseOverEvent += (sender, args) => { };

//鼠标在HUD双击
globeControl1.HudControlMouseDbClickEvent += (sender, args) => { };
//鼠标进入HUD
globeControl1.HudControlMouseIntoEvent += (sender, args) => { };
//鼠标移出HUD
globeControl1.HudControlMouseOutEvent += (sender, args) => { };
//鼠标移过HUD
globeControl1.HudControlMouseMoveEvent += (sender, args) => { };
//鼠标在HUD上抬起
globeControl1.HudControlMouseUpEvent += (sender, args) => { };
//鼠标在HUD上按下
globeControl1.HudControlMouseDownEvent += (sender, args) => { };
```

10.5 回调事件

```
//鼠标在HUD双击
globeControl1.HudControlMouseDbClickEvent += (sender, args) => { }
//鼠标进入HUD
globeControl1.HudControlMouseIntoEvent += (sender, args) => { };
//鼠标移出HUD
globeControl1.HudControlMouseOutEvent += (sender, args) => { };
//鼠标移过HUD
globeControl1.HudControlMouseMoveEvent += (sender, args) => { };
//鼠标在HUD上抬起
globeControl1.HudControlMouseUpEvent += (sender, args) => { };
//鼠标在HUD上按下
globeControl1.HudControlMouseDownEvent += (sender, args) => { };

//在线加载地形之后
globeControl1.AfterNetTerrainAddEvent += (sender, args) => { };
//在线加载图层之后
globeControl1.AfterNetLayerAddEvent += (sender, args) => { };
//在线加载工程之后
globeControl1.AfterNetSceneAddEvent += (sender, args) => { };
//添加图层之后
globeControl1.AfterLayerAddEvent += (sender, args) => { };
//图层移动之后
globeControl1.AfterLayerMoveEvent += (sender, args) => { };
```

```
//图层移除之后
globeControl1.AfterLayerRemoveEvent += (sender, args) => { };
//地形添加之后
globeControl1.AfterTerrainAddEvent += (sender, args) => { };
//地形移动之后
globeControl1.AfterTerrainMoveEvent += (sender, args) => { };
//地形移除之后
globeControl1.AfterTerrainRemoveEvent += (sender, args) => { };

//移除图层之前
globeControl1.BeforeLayerRemoveEvent += (sender, args) => { };
//移除地形之前
globeControl1.BeforeTerrainRemoveEvent += (sender, args) => { };
//读取工程之前
globeControl1.BeforeSceneRenderEvent += (sender, args) => { };
```

*还有很多更加细致的回调事件，这里就不一一说明。

十一、分析

此章节需要引入一个新的库，ZedGraph.dll，位于我们 SDK 之中，这个库用于图表等的显示。（如果有自己的图表库的话，可以使用自己的图表库）

首先我们先说一下 ZedGraph 类库的应用：

11.1 ZedGraph 类库

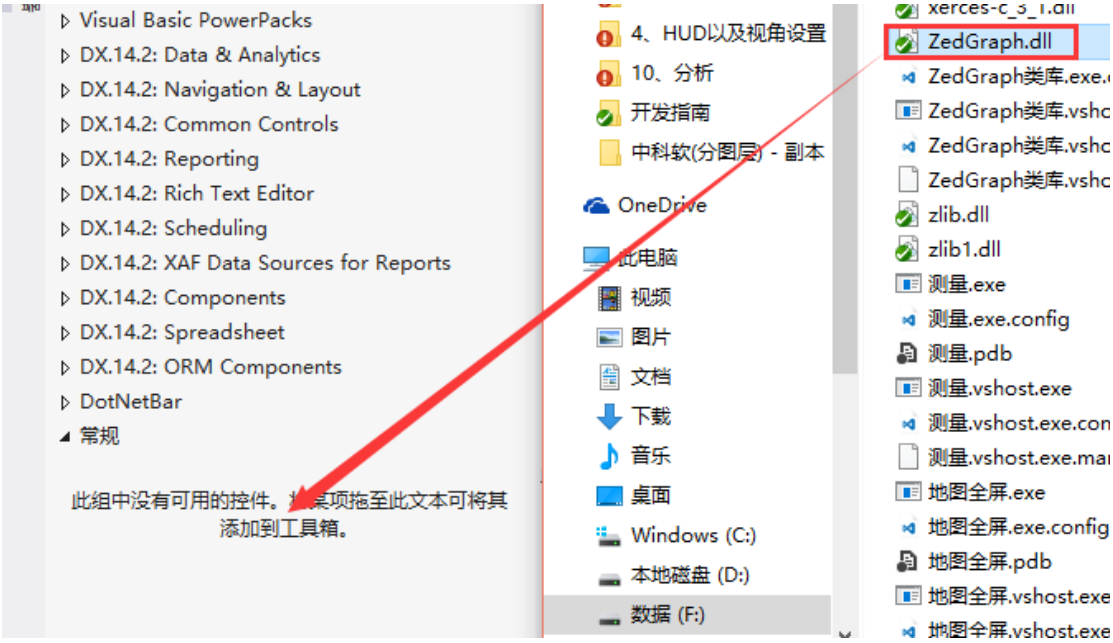
首先需要在程序中添加 ZedGrap.dll 引用，在代码中添加命名空间 using ZedGraph，在窗体中添加 ZedGraphControl 控件（假定该控件为 zedGraphControl1），然后就可以对该控件的属性进行修改，接下来会对使用 ZedGraph 类库如何创建线性、柱形和饼状图表进行分别阐述。

添加命名空间：

```
using ZedGraph;
```

如何添加 ZedGraphControl 控件：

将 ZedGraph.dll 拖拽到工具箱的空白处：



11.1.1 图形区域

11.1.1.1 创建图形区域

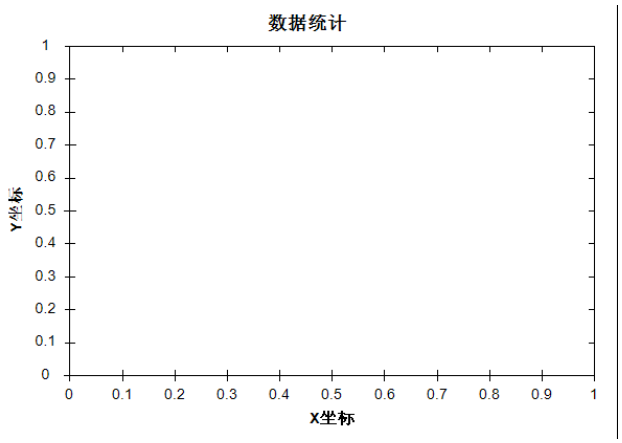
创建图形显示区域并对图表的标题及图表 X,Y 轴的标题进行设置，创建图表区域有两种方法，下面对于这两种方法进行分别描述：

方法一：

```
//在坐标(40,40)处创建一个新图形，大小为 600x400，
//图表区域标题为My Test Graph,X轴标题为My X Axis，Y轴标题为My Y Axis。
myPane = new GraphPane(
    new Rectangle(40, 40, 600, 400),
    "My Test Graph",
    "My X Axis",
    "My Y Axis");
//添加到zedGraphControl1中

zedGraphControl1.GraphPane = myPane;
```

方法二：

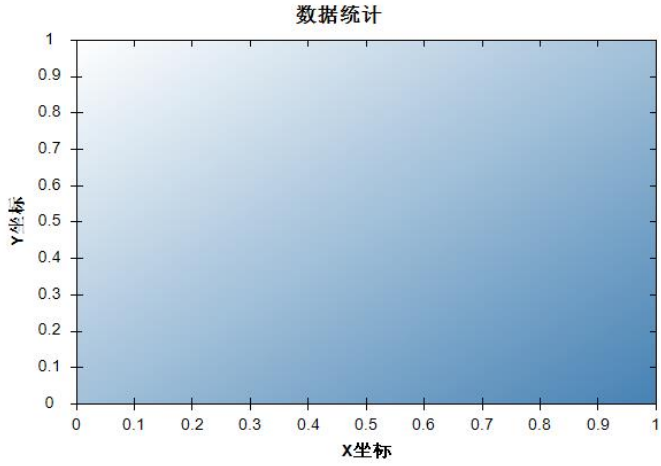
```
myPane = zedGraphControl1.GraphPane;          //获取zedGraphControl1的图表显示区域赋值给myPane。
myPane.Title.Text = "数据统计";               //图表区域标题
myPane.XAxis.Title.Text = "X坐标";            //X轴标题

myPane.YAxis.Title.Text = "Y 坐标";           //Y 轴标题
```



11.1.1.2 图形区域属性设置

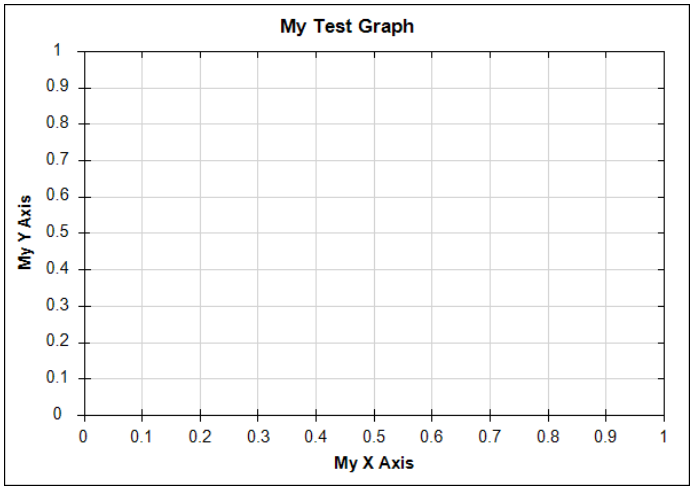
图表区域颜色渐变设置：

```
myPane.Chart.Fill = new Fill(Color.White, Color.SteelBlue, 45.0f);
//图表显示区域颜色，渐变颜色1White，渐变颜色2SteelBlue，45.0f为渐变颜色角度
```



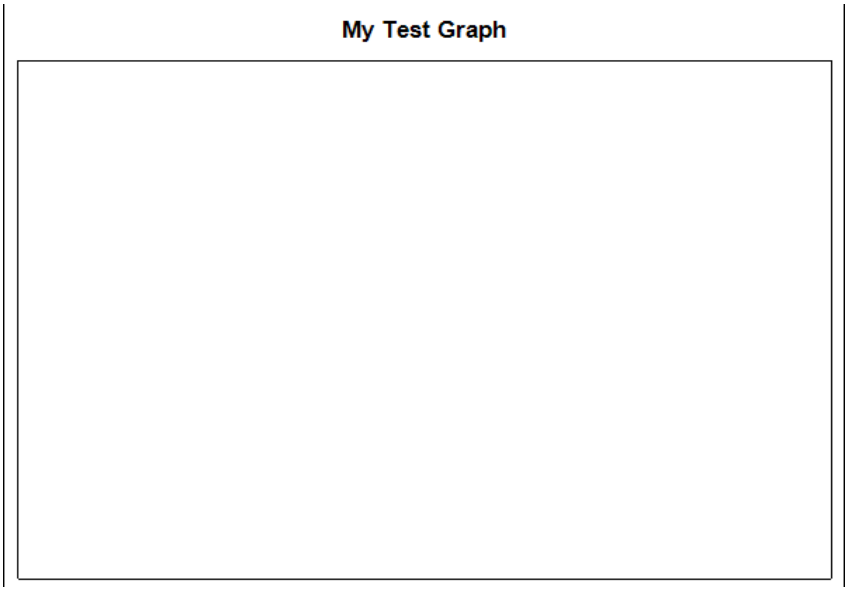
大跨度的 X,Y 轴表格虚线是否显示：

```
//大跨度的X轴表格虚线是否显示
myPane.XAxis.MajorGrid.IsVisible = !myPane.XAxis.MajorGrid.IsVisible;
//大跨度的Y轴表格虚线是否显示
myPane.YAxis.MajorGrid.IsVisible = !myPane.YAxis.MajorGrid.IsVisible;

myPane.XAxis.MajorGrid.Color = Color.LightGray;
myPane.YAxis.MajorGrid.Color = Color.LightGray;//表格虚线颜色设置

myPane.YAxis.MajorGrid.DashOff = 0;
myPane.XAxis.MajorGrid.DashOff = 0;            //表格线以实线显示
```



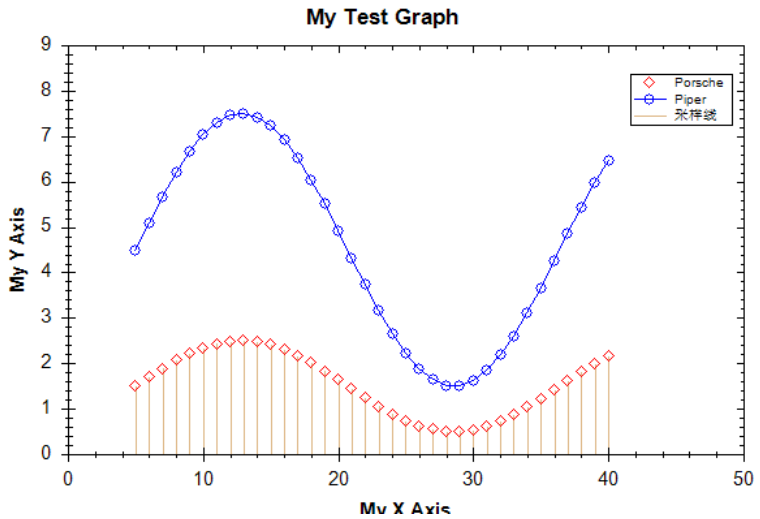
X,Y 轴及轴刻度：

```
//X,Y轴及轴刻度：
myPane.XAxis.IsVisible = false;            //X轴是否显示
myPane.YAxis.IsVisible = false;            // Y轴是否显示

myPane.XAxis.Scale.IsVisible = false;      //X轴刻度是否显示
myPane.YAxis.Scale.IsVisible = false;      //Y轴刻度是否显示

myPane.XAxis.Scale.Min = -1000;  //X轴最小刻度
myPane.XAxis.Scale.Max = 1000;  //X轴最大刻度

myPane.YAxis.Scale.Min = -1000;  //Y轴最小刻度
myPane.YAxis.Scale.Max = 1000;  //Y轴最大刻度

myPane.XAxis.Scale.BaseTic = 0;            //X轴第一个主刻度从哪里开始
myPane.YAxis.Scale.BaseTic = 0;            //Y 轴第一个主刻度从哪里开始
```



图表注释是否显示并对显示内容属性进行设置：

```
myPane.Legend.IsVisible = true;            //图表注释是否显示

myPane.Legend.Position = LegendPos.Float;
//LegendPos是一个枚举，共有13个枚举值：Top、Left、Right、Bottom、InsideTopLeft、InsideTopRight、
//InsideBotLeft、InsideBotRight、Float、TopCenter、BottomCenter、TopFlushLeft和BottomFlushLeft。

myPane.Legend.Location = new Location(0.95f, 0.15f, CoordType.PaneFraction, AlignH.Right, AlignV.Top);
//Location是指定Legend具体坐标的一个类，要注意的是，只有当LegendPos是Float时，Location才会起作用。

myPane.Legend.FontSpec.Size = 10f;
//FontSpec类就是一个字体类，里面是关于字体的一些相关设置

myPane.Legend.IsHStack = false;
//IsHStack 是一个 Legend 的属性，是设置 Legend 中文字和图形的显示方式是水平还是垂直。
```

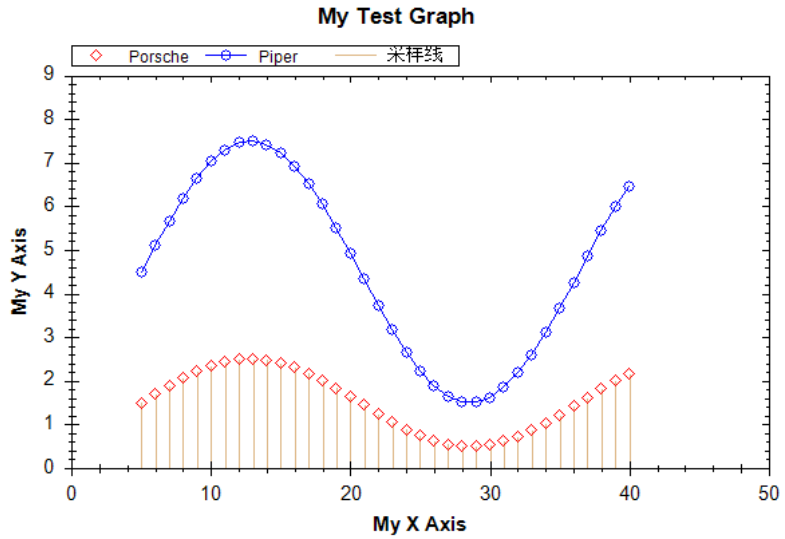


11.1.2 创建线性图表

使用 ZedGraph 类库创建线性图表，其显示数据可以是直接指定（X,Y）坐标或者二维数组或者连接数据库，使用 sql 语句查询数据库中的数据作为图表的数据源，下面对于这两种数据源分别作介绍。

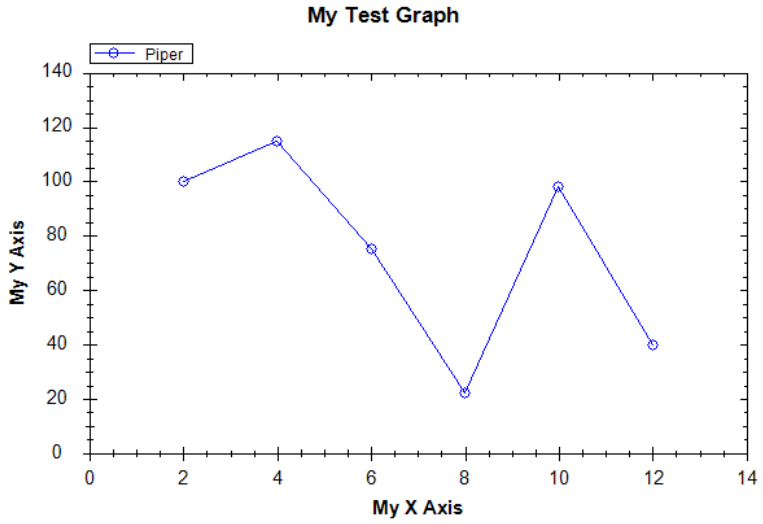
11.1.2.1 通过制定（X,Y）坐标作为图表的数据源

```
double x, y1, y2;        //定义变量存放点的X,Y坐标
PointPairList list1 = new PointPairList();
PointPairList list2 = new PointPairList();  // PointPair类是一个包含(X,Y)的坐标
类

for (int i = 0; i < 36; i++)
{
    x = (double)i + 5;                          //给数据的X坐标赋值
    y1 = 1.5 + Math.Sin((double)i * 0.2);       //给数据的Y坐标赋值
    y2 = 3.0 * (1.5 + Math.Sin((double)i * 0.2));
    list1.Add(x, y1);
    list2.Add(x, y2);   //在列表中存数每个点的X,Y坐标
}
LineItem myCurve = myPane.AddCurve("Porsche", list1, Color.Red,
SymbolType.Diamond);
//LineItem类是ZedGraph中的线条类. 根据数据集list1创建红色的菱形曲线, 标记
"Porsche", 这个函数的返回值是一个LineItem。
//可以通过myCurve这个变量来对它进行进一步的设定。其中SymbolType是个Enum, 它枚举了
12个可供使用的形状,
//包括Circle,Default,Diamond,HDash,None,Plus,
Square,Star,Triangle,UserDefined,VDash,XCross.

LineItem myCurve2 = myPane.AddCurve("Piper", list2, Color.Blue,
SymbolType.Circle);
// 根据数据集list2创建蓝色的圆形曲线, 标记"Piper"

myCurve.Line.IsVisible = false;      //设置曲线的线不可见, 则只显示各个散点

myPane.AddStick("采样线", list1, Color.BurlyWood);//对于数据集list1的每个点添加
投影线


myPane.AxisChange(this.CreateGraphics()); // 在数据变化时绘制图形
```



11.1.2.2 数组作为图表的数据源

```
double[] x = { 2, 4, 6, 8, 10, 12 };          //定义变量存储数据的X坐标
double[] y = { 100, 115, 75, 22, 98, 40 };    //定义变量存储数据的Y坐标
PointPairList list = new PointPairList();
for (int i = 0; i < x.Length; i++)
{
    list.Add(x[i], y[i]);    //将X,Y坐标保存到list中
}
LineItem myCurve = myPane.AddCurve("Piper", list, Color.Blue,
SymbolType.Circle);
// 根据数据集list创建蓝色的圆形曲线，标记"Piper"
myPane.AxisChange(this.CreateGraphics());  //刷新显示
```



11.1.2.3 DataTable 数据源

连接数据库，查询数据库中的数据作为图表的数据源：

*此示例没有数据库，仅供参考

在配置文件中添加连接字符串：

```
<connectionStrings>
    <add      name="SqlConn"      connectionString="data      source=Server;initial
catalog=DataBaseName;
        integrated security=true"/>
</connectionStrings>
```

在上述字符串中的 Server 为服务器名，DataBaseName 为数据库名。

通过 sql 语句查询数据方法：

```
private static readonly string path =
System.Configuration.ConfigurationManager.ConnectionStrings["SqlConn"].Connec
tionString;
//读取配置文件中的连接字符串SqlConn

public static DataSet Select(string sql)
{//执行sql语句，返回dataset
    SqlConnection connection = null;  //创建连接
    try
    {
        using (connection = new SqlConnection(path))
        {
            DataSet ds = new DataSet();
            connection.Open();     //打开连接
            SqlDataAdapter command = new SqlDataAdapter(sql, connection);
            command.Fill(ds, "ds");
            return ds;
        }
    }
    catch (System.Exception)
    {
        throw;
    }
    finally
    {
        connection.Dispose();  //释放资源
    }
```

```
}
```

将查询出的数据在图表中显示：

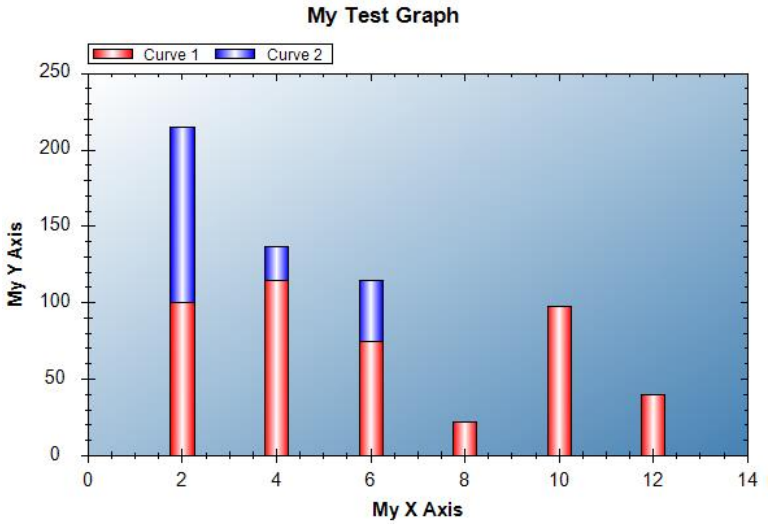
```
DataSet m_dataSet;                                    //定义dataset，接收查询出的数据
string sql = "select ColumnName from TableName";
//查询数据sql语句,其中ColumnName为查询的列名，TableName为查询表名。
m_dataSet = Select(sql);                    //Select为执行sql语句的方法，将sql语句传给该方法
DataTable table = m_dataSet.Tables[0];      //将查询的数据存放到datatable中
PointPairList list = new PointPairList();
for (int x = 0; x < table.Rows.Count; x++)
{
    double y = double.Parse(table.Rows[x][1].ToString());  //从查询的数据中获取Y坐标
    list.Add(x, y);   //将X,Y坐标存放到list中
}
LineItem myCurve = myPane.AddCurve("Piper", list, Color.Blue,
SymbolType.Circle);
// 根据数据集list创建蓝色的圆形曲线，标记"Piper"

myPane.AxisChange(this.CreateGraphics());  //刷新显示
```

11.1.3 创建柱形图表

使用 ZedGraph 类库创建柱形图表，BarItem 它的基类 ZedGraph.CurveItem 里包含了 Pane 上单个曲线图表的数据和方法。它实现了图表的属性设置，例如关键词(Key)，成员的名字、颜色、符号、尺寸和线条的风格等等。

```
double[] x = { 2, 4, 6, 8, 10, 12 };          //定义变量存储数据的X坐标
double[] y = { 100, 115, 75, 22, 98, 40 };    //定义变量存储数据的Y坐标

BarItem myBar = myPane.AddBar("Curve 1", x, y, Color.Red);
// BarItem它的基类ZedGraph.CurveItem里包含了Pane上单个曲线图表的数据和方法。
//它实现了图表的属性设置，myPane.AddBar方法中的第一个参数为柱形图的注释，第二三个参数分别是数据的X,Y坐标，
//最后一个参数为柱形图的颜色。

myBar.Bar.Fill = new Fill(Color.Red, Color.White, Color.Red);
//设置柱形图颜色，两边为Red，中间为White。

//myPane.BarSettings.Base = BarBase.Y;       //以Y轴为基轴
```

```
myPane.BarSettings.Base = BarBase.X;          //以X轴为基轴

myBar = myPane.AddBar("Curve 2", null, y, Color.Blue);
myBar.Bar.Fill = new Fill(Color.Blue, Color.White, Color.Blue);

myPane.BarSettings.Type = BarType.Stack;
//BarType是一个枚举，共有六项，
//分别为Cluster、ClusterHiLow、Overlay、SortedOverlay、Stack和PercentStack，
//Cluster和ClusterHiLow是让多个同一个基类Bar依次排开，Cluster还可以使用来自
IPointList的“Z”的值来定义每一个Bar的底部，
//Overlay和SortedOverlay就是柱形按坐标相互覆盖。
//不同之处在于Overlay是按照哪个先画哪个在前的原则(注意这里不是按后画把先画的柱形覆盖
的原则，而是正好相反按先画在前原则)。
//SortedOverlay是按位标的大小，按小的位标在前，大的位标在后的原则来绘图的，最后的两
个Stack和PercentStack就是按先前的位标依次累积上升。

myPane.Chart.Fill = new Fill(Color.White, Color.SteelBlue, 45.0f);
myPane.AxisChange(this.CreateGraphics());
```



11.1.4 创建饼形图表

使用 ZedGraph 类库创建饼形图表，饼形图也是继承 ZedGraph.CurveItem，与其它继承 CurveItem 不同的是，PieItem 有一个 value 的属性，可以方便的存取 PieItem 实例的值，从而可以很方便的动态改变一个饼形在整个饼形区域所占的比重。

```
PieItem segment1 = myPane.AddPieSlice(20, Color.Navy, Color.White, 45.0f, 0,
"North");
PieItem segment3 = myPane.AddPieSlice(30, Color.Purple, Color.White, 45f, 0,
"East");
```

```
//六个参数分别为：在整个饼形图中占的比重，渐变颜色1，渐变颜色2，渐变颜色角度，远离中心点的距离，饼形图的文字注释

PieItem segment4 = myPane.AddPieSlice(10.21, Color.LimeGreen, Color.White, 45f, 0, "West");
PieItem segment2 = myPane.AddPieSlice(40, Color.SandyBrown, Color.White, 45f, 0.2, "South");
PieItem segment6 = myPane.AddPieSlice(250, Color.Red, Color.White, 45f, 0, "Europe");
PieItem segment7 = myPane.AddPieSlice(50, Color.Blue, Color.White, 45f, 0.2, "Pac Rim");
PieItem segment8 = myPane.AddPieSlice(400, Color.Green, Color.White, 45f, 0, "South America");
PieItem segment9 = myPane.AddPieSlice(50, Color.Yellow, Color.White, 45f, 0.2, "Africa");
segment2.LabelDetail.FontSpec.FontColor = Color.Red;     //对其中一部分的 Label 提示信息进行设置
```

11.1.5 ZedGraphControl 属性设置

对于 ZedGraphControl 的一些属性设置：

```
zedGraphControl1.IsShowPointValues = true;
//当鼠标悬浮到图表中的某一点上时鼠标经过图表上的点时是否气泡显示该点所对应的值.

zedGraphControl1.IsShowCursorValues = true;//鼠标在图表上移动时是否显示鼠标所在点对应的坐标值。默认为false

zedGraphControl1.IsShowHScrollBar = true; //是否显示横向滚动条
//zedGraphControl1.IsShowVScrollBar = true; //是否显示纵向滚动条

zedGraphControl1.IsEnableZoom = true;//是否允许缩放
zedGraphControl1.IsEnableHZoom = true;//是否允许横向缩放
//zedGraphControl1.IsEnableVZoom = true;//是否允许纵向缩放

zedGraphControl1.IsShowContextMenu  =  true;//  是否显示右键菜单，如果指定了 ContextMenuStrip 会一直显示指定的 ContextMenu。
```

11.2 填挖方分析

填挖方分析即在场景中选择要分析的面和高度，然后分析改空间内，哪些地方需要挖，哪些地方需要填。

首先我们需要画出一个范围：

```
/// <summary>
/// 填挖方分析
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, System.EventArgs e)
{
    globeControl1.Globe.Action = GeoScene.Data.EnumAction3D.TrackPolygon;
    globeControl1.Globe.TrackPolygonTool.TrackMode =
EnumTrackMode.GroundTrack;
    trackPolygonType = "填挖方分析";
}
```

接下来在画完的时候调用一个事件：

```
//添加分析范围面画完事件
globeControl1.TrackPolygonEndEvent += new
TrackPolygonEndEventHandler(globeControl1_TrackPolygonEndEvent);
/// <summary>
/// 范围画完事件
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void globeControl1_TrackPolygonEndEvent(object sender,
TrackPolygonEndEventArgs e)
{
    if (e.Polygon != null)
    {
        if (trackPolygonType == "填挖方分析")
        {
            globeControl1.Globe.Action = EnumAction3D.ActionNull;
            FrmAnalysisDigFillOfTerrain frm = new
FrmAnalysisDigFillOfTerrain(globeControl1,e.Polygon);
            frm.Show(this);
        }
    }
```

```
}
```

这时会打开一个窗口，这个窗口只需要输入“基准面高程”便可分析，其余项为显示项：

FrmAnalysisDigFillOfTerrain 的构造事件

```
public FrmAnalysisDigFillOfTerrain(GSOGlobeControl _globeControl1,
GSOGeoPolygon3D _polygon3D)
{
    InitializeComponent();

    //从主窗口引入两个值，一个球，一个画的范围
    globeControl1 = _globeControl1;
    m_polygon3D = _polygon3D;
}
```

FrmAnalysisDigFillOfTerrain 窗口的 Load 事件

```
/// <summary>
/// 剖面分析窗口Load事件，用于设定一个默认高度
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

```
private void DigFillAnalysisDlg_Load(object sender, EventArgs e)
{
    //如果传过来的两个值都不为空
    if (globeControl1.Globe != null && m_polygon3D != null)
    {
        double dAlt = 0;//默认高度为0
        //交给系统处理，传入值为面，高度，返回为挖的高度，填的高度，挖的区域，填的区域，全部的面积，最高点，最低点，最后两项默认false，0
        globeControl1.Globe.Analysis3D.DigFillAnalyse(m_polygon3D, dAlt, out m_dDigVolume, out m_dFillVolume,
            out m_dDigArea, out m_dFillArea, out m_dTotalArea, out m_pntMaxAlt, out m_pntMinAlt, false, 0);
        textBoxDestAlt.Text = (m_pntMaxAlt.Z / 2.0 + m_pntMinAlt.Z / 2.0).ToString("0.0");
        buttonAnalyse_Click(null, null);
    }
}
```

分析按钮的单击事件

```
private void buttonAnalyse_Click(object sender, EventArgs e)
{
    if (globeControl1.Globe != null && m_polygon3D != null)
    {
        double dAlt = System.Convert.ToDouble(textBoxDestAlt.Text);
        //交给系统处理，传入值为面，高度，返回为挖的高度，填的高度，挖的区域，填的区域，全部的面积，最高点，最低点，最后两项默认false，0
        globeControl1.Globe.Analysis3D.DigFillAnalyse(m_polygon3D, dAlt, out m_dDigVolume, out m_dFillVolume,
            out m_dDigArea, out m_dFillArea, out m_dTotalArea, out m_pntMaxAlt, out m_pntMinAlt, false, 0);

        GSOFeature altFeature = null;//创建一个空要素
        GSOFeatures tempFeatures = globeControl1.Globe.MemoryLayer.GetFeatureByName("DigFillAltPolygon", true);//要素从内存图层里获取
        if (tempFeatures.Length > 0)         //判断是否获取到
        {
            altFeature = tempFeatures[0];   //获取到就直接赋值
        }
        //把传进来的范围面复制一个
        GSOGeoPolygon3D newPolygon = (GSOGeoPolygon3D)m_polygon3D.Clone();
        newPolygon.SetAltitude(dAlt);       //给这个面设置高度
```

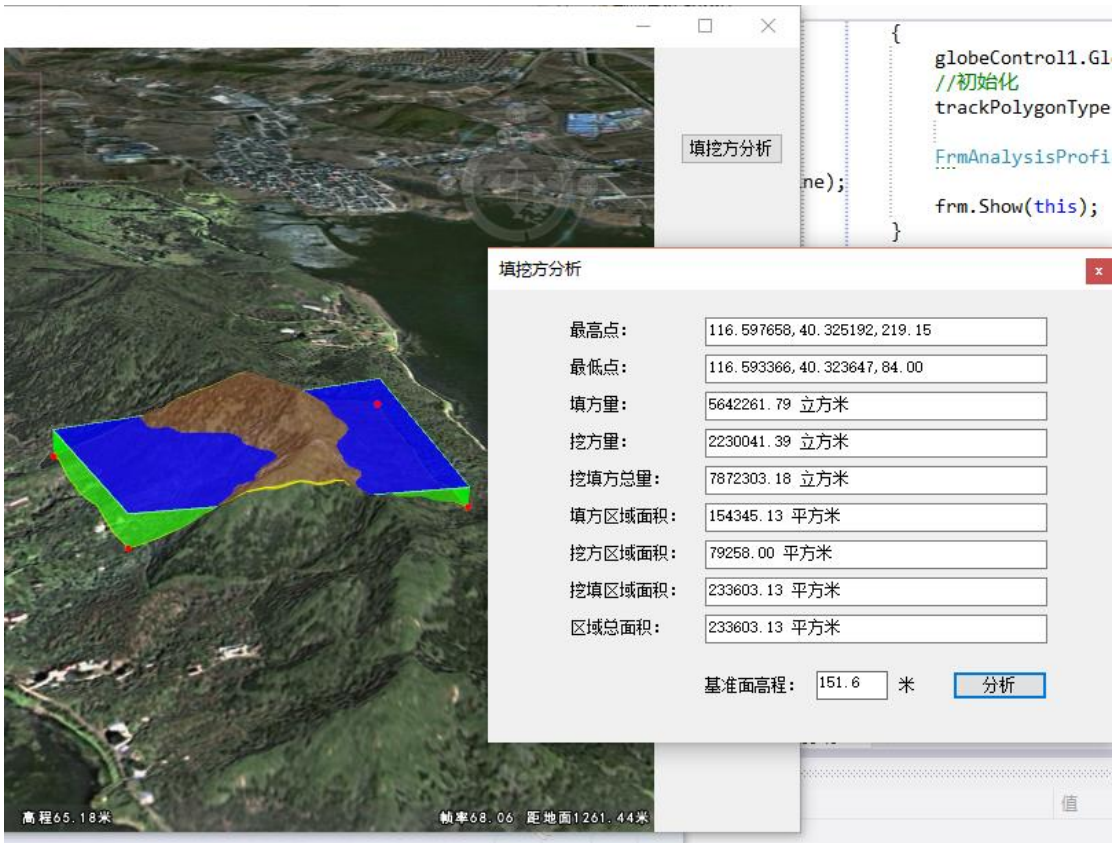
```
        newPolygon.AltitudeMode = EnumAltitudeMode.Absolute;//高度类型是相对地面
        GSOExtrudeStyle extrudeStyle = new GSOExtrudeStyle();//拉伸风格
        extrudeStyle.ExtrudeType = EnumExtrudeType.ExtrudeToValue;//拉伸方式
        extrudeStyle.ExtrudeValue = m_pntMinAlt.Z;//拉伸的值
        extrudeStyle.TailPartVisible = false;     //末端是否可见
        //多边形风格，立面风格
        GSOSimplePolygonStyle3D extrudePolygonStyle = new GSOSimplePolygonStyle3D();
        extrudePolygonStyle.FillColor = Color.FromArgb(150, 0, 255, 0);//多边形填充颜色
        extrudeStyle.BodyStyle = extrudePolygonStyle;      //把风格赋给拉伸风格
        //多边形风格，顶层风格
        GSOSimplePolygonStyle3D polygonStyle = new GSOSimplePolygonStyle3D();
        polygonStyle.FillColor = Color.FromArgb(200, 0, 0, 255);//设置颜色
        newPolygon.Style = polygonStyle;    //面的顶层风格
        newPolygon.ExtrudeStyle = extrudeStyle; //棉的立面风格
        //判断如果要素为空，或者被删除
        if (m_AltFeature == null || m_AltFeature.IsDeleted)
        {
            //新建一个要素，并且把刚才配置的要素赋值给它
            m_AltFeature = new GSOFeature();
            m_AltFeature.Name = "DigFillAltPolygon";
            m_AltFeature.Geometry = newPolygon;
            globeControl1.Globe.MemoryLayer.AddFeature(m_AltFeature);
        }
        else
        {
            //如果原来就有，就直接赋值
            m_AltFeature.Geometry = newPolygon;
        }
        //刷新球
        globeControl1.Globe.Refresh();
        //设置文字
        SetText();
    }
}
```

窗口关闭事件：

```
/// <summary>
/// 窗口关闭事件
/// </summary>
/// <param name="sender"></param>
```

```
/// <param name="e"></param>
private void DigFillAnalysisDlg_FormClosed(object sender, FormClosedEventArgs
e)
{
    if (m_AltFeature != null)
    {
        globeControl1.Globe.MemoryLayer.RemoveFeatureByID(m_AltFeature.ID);
        globeControl1.Globe.ClearLastTrackPolygon();
        m_AltFeature = null;
        globeControl1.Refresh();
    }
}
```

运行一下，便可以看到结果：



我们也可以根据画完的面进行填挖方分析：

```
string mouseClikeInfo = String.Empty;
Point mouseDown = new Point();
```

```
/// <summary>
/// 选择面
/// </summary>
```

```
/// <param name="sender"></param>
/// <param name="e"></param>
private void button3_Click(object sender, EventArgs e)
{
    globeControl1.Globe.Action = EnumAction3D.SelectObject;
    mouseClikeInfo = "填挖方分析";
}
```

```
globeControl1.MouseDown += (sender, e) =>
{
    mouseDown = e.Location;
};

globeControl1.MouseUp += (sender, e) =>
{
    if (mouseDown != e.Location) return;

    if (e.Button == MouseButtons.Left)
    {
        if (globeControl1.Globe.Action == EnumAction3D.SelectObject)
        {
            if (mouseClikeInfo == "填挖方分析")
            {
                GSOFeature feature = globeControl1.Globe.SelectedObject;
                if (feature == null ||
                    feature.Geometry == null ||
                    feature.Geometry.Type != EnumGeometryType.GeoPolygon3D)
                {
                    MessageBox.Show("请选择一个面");
                    return;
                }
                globeControl1.Globe.Action = EnumAction3D.ActionNull;
                GSOGeoPolygon3D polygon = feature.Geometry as GSOGeoPolygon3D;
                FrmAnalysisDigFillOfTerrain frm = new
FrmAnalysisDigFillOfTerrain(globeControl1, polygon);
                frm.Show(this);
            }
        }
    }
};
```

11.3 剖面分析

剖面分析即在场景中选择要分析的剖面，然后分析出该剖面的高程变化。

首先我们要呼出一个范围：

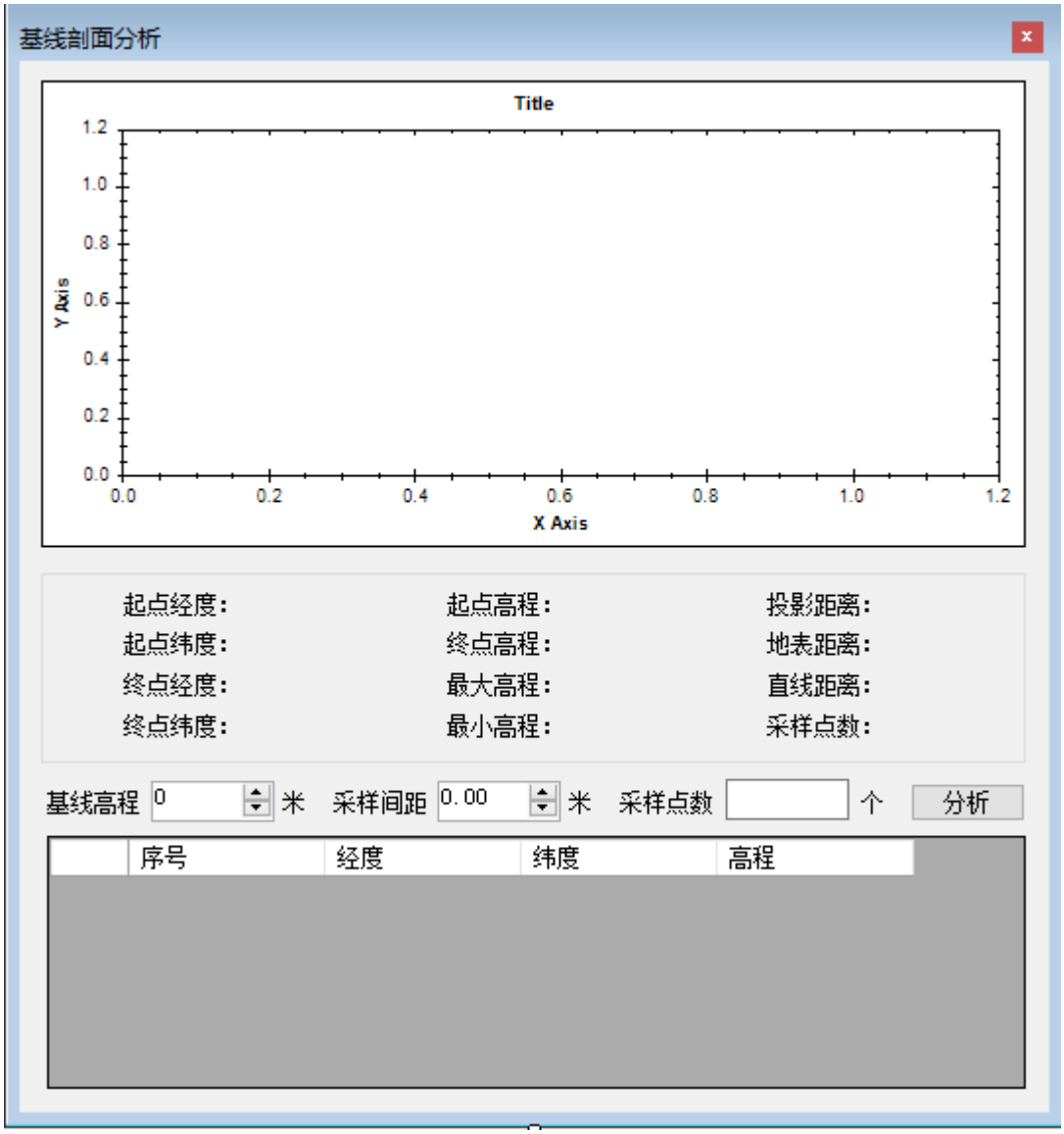
```
globeControl1.Globe.Action = EnumAction3D.TrackPolyline;
globeControl1.Globe.TrackPolylineTool.TrackMode = EnumTrackMode.GroundTrack;
globeControl1.Globe.TrackPolylineTool.VerticalLineVisible = true;
trackPolygonType = "剖面分析";
```

接下来调用一个画完事件

```
//添加分析分为线画完事件
globeControl1.TrackPolylineEndEvent +=
 new TrackPolylineEndEventHandler(globeControl1_TrackPolylineEndEvent);
```

```
/// <summary>
/// 分析范围面画完事件
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void globeControl1_TrackPolylineEndEvent(object sender,
TrackPolylineEndEventArgs e)
{
    if (e.Polyline != null)
    {
        if (trackPolygonType == "剖面分析")
        {
            globeControl1.Globe.Action = GeoScene.Data.EnumAction3D.ActionNull;
            //初始化
            trackPolygonType = "";

            FrmAnalysisProfileBaseLine frm = new
FrmAnalysisProfileBaseLine(globeControl1, e.Polyline);
            frm.Show(this);
        }
        else
        {
            trackPolygonType = "";
        }
    }
}
```

这时会打开一个窗口，这个窗口需要输入“基线高程”、“采样间距”和“采样点数”便可分析，其余项为显示项：



窗口构造阶段传入球和分析的线：

```
public FrmAnalysisProfileBaseLine(GSOGlobeControl _globeControl1,
GSOGeoPolyline3D _geopolyline)
{
    InitializeComponent();

    globeControl1 = _globeControl1;
    m_geopolyline = _geopolyline;
}
```

窗口 Load 事件：

```
private void BaseLineProfillAnalysis_Load(object sender, EventArgs e)
{
    GraphPane myPane = zedGraphControl1.GraphPane;
    myPane.Title.Text = "剖面分析";
    myPane.Title.FontSpec.Family = "黑体";
    myPane.Title.FontSpec.IsBold = false;
    myPane.Title.FontSpec.Size = 18.0f;

    myPane.XAxis.Title.Text = "长度";
    myPane.XAxis.Title.FontSpec.Family = "黑体";
    myPane.XAxis.Title.FontSpec.IsBold = false;
    myPane.XAxis.Title.FontSpec.Size = 18.0f;

    myPane.YAxis.Title.Text = "高程";
    myPane.YAxis.Title.FontSpec.Family = "黑体";
    myPane.YAxis.Title.FontSpec.IsBold = false;
    myPane.YAxis.Title.FontSpec.Size = 18.0f;

    myPane.Chart.Fill = new Fill(Color.White, Color.LightGray, 45.0f);

    double lineLength = m_geopolyline.GetSphereLength(6378137);
    textBoxPointCount.Text = "100";
    textBoxJianJu.Text = (lineLength / 99.0).ToString("0.0");
    buttonAnalyse_Click(null, null);
}
```

分析点击事件：

```
/// <summary>
/// 开始分析
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonAnalyse_Click(object sender, EventArgs e)
{
    //判断球和分析线不为空
    if (m_geopolyline != null && globeControl1.Globe != null)
    {
        //获取线的地表长度  输入：半径
        m_dSphereLength = m_geopolyline.GetSphereLength(6378137.0);

        //获取线的空间长度  输入：是否忽略Z值 ， 半径
        m_dSpaceLength = m_geopolyline.GetSpaceLength(false, 6378137.0);
        //获取地标距离  输入：分析线，false，0
```

```
        m_dGroundLength =
globeControl1.Globe.Analysis3D.GetGroundLength(m_geopolyline, false, 0);
        //采样间距
        float jianJu = (float)textBoxJianJu.Value;
        //采样数量
        int pointCount = (int)(m_dSphereLength / jianJu);
        GSOPoint3d pntMax, pntMin, pntStart, pntEnd;
        GSOPoint3ds pnt3ds;
        double dLineLength;

        //获取分析结果 输入：分析线，采样数量
        //输出：点集合，线长度，最大点，最小点，开始点，结束点
        globeControl1.Globe.Analysis3D.ProfileAnalyse(m_geopolyline,
pointCount, out pnt3ds, out dLineLength, out pntMax, out pntMin, out
pntStart, out pntEnd);

        //开始根据数据整理成到DataGridView中
        try
        {
            m_pnt3ds = pnt3ds;
            m_pntMax = pntMax;
            m_pntMin = pntMin;
            m_pntStart = pntStart;
            m_pntEnd = pntEnd;
            m_dXTotalLength = dLineLength;
            dataGridViewPoints.Rows.Clear();
            int index = 1;
            for (int i = 0; i < m_pnt3ds.Count; i++)
            {
                GSOPoint3d pt = m_pnt3ds[i];
                if (pt != null)
                {
                    int rowIndex = dataGridViewPoints.Rows.Add();
                    dataGridViewPoints.Rows[rowIndex].Cells[0].Value =
index.ToString();
                    dataGridViewPoints.Rows[rowIndex].Cells[1].Value =
pt.X.ToString("0.0000000");
                    dataGridViewPoints.Rows[rowIndex].Cells[2].Value =
pt.Y.ToString("0.0000000");
                    dataGridViewPoints.Rows[rowIndex].Cells[3].Value =
pt.Z.ToString("0.00");
                    index++;
                }
```

```
            }
            //设置文字
            SetLableText();
            //画出图
            DrawCurveGraph();
        }
        catch (Exception ext)
        {
            MessageBox.Show("数据异常，参考信息：" + ext.Message, "系统提示",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

绘图事件：

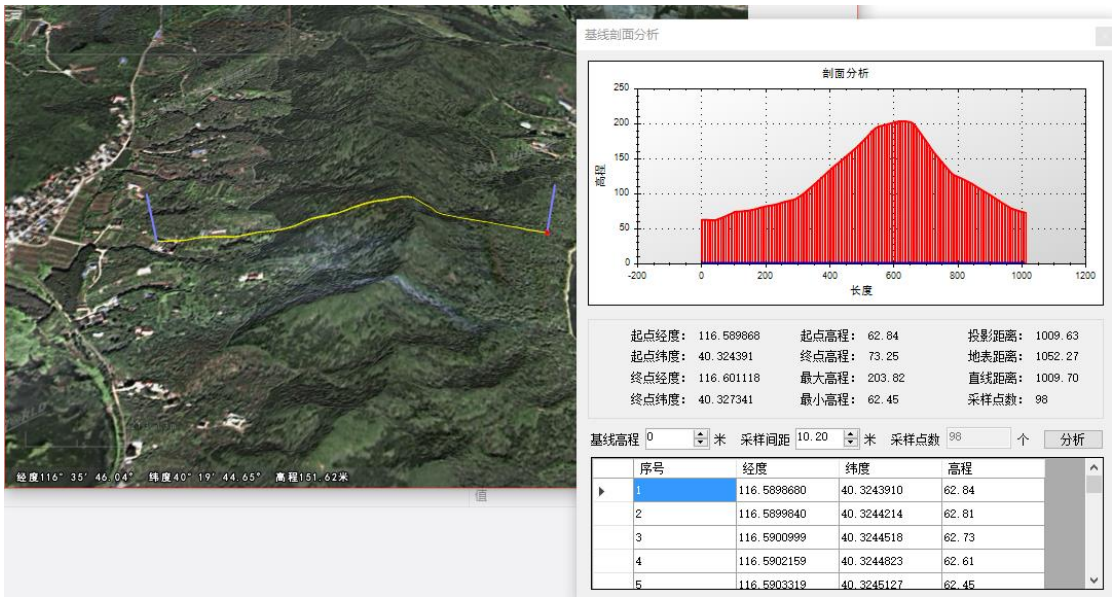
```
/// <summary>
/// 绘图事件
/// </summary>
private void DrawCurveGraph()
{
    GraphPane myPane = zedGraphControl1.GraphPane;
    myPane.CurveList.Clear();
    myPane.GraphObjList.Clear();
    zedGraphControl1.RestoreScale(zedGraphControl1.GraphPane);
    m_dBaseAlt = Convert.ToDouble(textBoxBaseAlt.Text);
    myPane.Legend.IsVisible = false;
    PointPairList listHLAbove = new PointPairList();
    PointPairList listAbove = new PointPairList();
    PointPairList listHLBelow = new PointPairList();
    PointPairList listBelow = new PointPairList();
    PointPairList listEqual = new PointPairList();
    PointPairList listBase = new PointPairList();

    ArrayList arryHLAbove = new ArrayList();
    ArrayList arryHLBelow = new ArrayList();
    ArrayList arryAbove = new ArrayList();
    ArrayList arryBelow = new ArrayList();

    int nPointCount = m_pnt3ds.Count;
    double dOneSegLen = m_dXTotalLength / (nPointCount - 1);

    int nSegmentType = -1; //0=m_dBaseAlt,1=above,2=below
    for (int i = 0; i < m_pnt3ds.Count; i++)
```

```
    {
        double x = i * dOneSegLen;
        double y = (int)(Math.Round(m_pnt3ds[i].Z * 100)) / 100.0; // 精确到厘
米就行了
        if (y > m_dBaseAlt)
        {
            // 如果当前段是基准线下面的段
            if (nSegmentType == 2)
            {
                arryHLBelow.Add(listHLBelow);
                arryBelow.Add(listBelow);
                listHLBelow = new PointPairList();
                listBelow = new PointPairList();
            }
            nSegmentType = 1;
            listAbove.Add(x, y);
            listHLAbove.Add(x, y, m_dBaseAlt);
        }
        else if (y < m_dBaseAlt)
        {
            if (nSegmentType == 1)
            {
                arryHLAbove.Add(listHLAbove);
                arryAbove.Add(listAbove);
                listHLAbove = new PointPairList();
                listAbove = new PointPairList();
            }
            nSegmentType = 2;
            listBelow.Add(x, y);
            listHLBelow.Add(x, m_dBaseAlt, y);
        }
        else
        {
            if (nSegmentType == 2)
            {
                arryHLBelow.Add(listHLBelow);
                arryBelow.Add(listBelow);
                listHLBelow = new PointPairList();
                listBelow = new PointPairList();
            }
            else if (nSegmentType == 1)
            {
                arryHLAbove.Add(listHLAbove);
```

```
                arryAbove.Add(listAbove);
                listHLAbove = new PointPairList();
                listAbove = new PointPairList();
            }
            nSegmentType = 0;
            listEqual.Add(x, y, m_dBaseAlt);
        }
    }
    if (nSegmentType == 2)
    {
        arryHLBelow.Add(listHLBelow);
        arryBelow.Add(listBelow);
    }
    else if (nSegmentType == 1)
    {
        arryHLAbove.Add(listHLAbove);
        arryAbove.Add(listAbove);
    }
    listBase.Add(0, m_dBaseAlt);
    listBase.Add(m_dXTotalLength, m_dBaseAlt);

    LineItem myCurveBase = myPane.AddCurve("基线剖面", listBase, Color.Blue,
SymbolType.None);
    myCurveBase.Line.IsAntiAlias = true;
    myCurveBase.Line.Width = 2;


    int k = 0;
    for (k = 0; k < arryHLAbove.Count; k++)
    {
        LineItem myCurveAbove = myPane.AddCurve("高于基线剖面",
(PointPairList)arryAbove[k], Color.Red, SymbolType.None);
        myCurveAbove.Line.IsAntiAlias = true;
        myCurveAbove.Line.Width = 2;
        myCurveAbove.Line.IsSmooth = true;

        HiLowBarItem hlAboveItem = myPane.AddHiLowBar("高于基线",
(PointPairList)arryHLAbove[k], Color.Red);
        hlAboveItem.Bar.Border.Color = Color.Red;
    }
    for (k = 0; k < arryHLBelow.Count; k++)
    {
```

```
        LineItem myCurveBelow = myPane.AddCurve("低于基线剖面",
(PointPairList)arryBelow[k], Color.Green, SymbolType.None);
        myCurveBelow.Line.IsAntiAlias = true;
        myCurveBelow.Line.Width = 2;
        myCurveBelow.Line.IsSmooth = true;

        HiLowBarItem hlBolowItem = myPane.AddHiLowBar("低于基线",
(PointPairList)arryHLBelow[k], Color.Green);
        hlBolowItem.Bar.Border.Color = Color.Green;
    }

    // Show the x axis grid
    myPane.XAxis.MajorGrid.IsVisible = true;
    myPane.XAxis.IsAxisSegmentVisible = true;
    if (m_bSetMinX)
    {
        myPane.XAxis.Scale.Min = 0;
    }
    if (m_bSetMinY)
    {
        myPane.YAxis.Scale.Min = m_pntMin.Z;
    }

    //myPane.XAxis.Scale.Min = 0;

    myPane.YAxis.MajorGrid.IsVisible = true;

    myPane.YAxis.MajorGrid.IsZeroLine = false;

    myPane.YAxis.Scale.Align = AlignP.Inside;

    myPane.Chart.Fill = new Fill(Color.White, Color.LightGray, 45.0f);
    zedGraphControl1.IsAutoScrollRange = true;

    // 是否显示点的X,Y坐标
    zedGraphControl1.IsShowPointValues = true;
    //鼠标悬浮到曲线上某一点时出发的事件
    zedGraphControl1.PointValueEvent += new
ZedGraphControl.PointValueHandler(MyPointValueHandler);

    // 添加ZedGraph缩放事件
    zedGraphControl1.ZoomEvent += new
ZedGraphControl.ZoomEventHandler(MyZoomEvent);
```

```
    zedGraphControl1.AxisChange();

    if (m_bXYSameScale)
    {
        graphPane_AxisChangeEvent();
    }

    zedGraphControl1.Invalidate();
}
```



我们也可以根据画完的线进行剖面分析：

```
string mouseClickInfo = String.Empty;
Point mouseDown = new Point();
```

```
/// <summary>
/// 选择线
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button3_Click(object sender, EventArgs e)
{
    globeControl1.Globe.Action = EnumAction3D.SelectObject;
    mouseClickInfo = "剖面分析";
}
```

```
globeControl1.MouseDown += (sender, e) =>
{
```

```
    mouseDown = e.Location;
};

globeControl1.MouseUp += (sender, e) =>
{
    if (mouseDown != e.Location) return;

    if (e.Button == MouseButtons.Left)
    {
        if (globeControl1.Globe.Action == EnumAction3D.SelectObject)
        {
            if (mouseClickInfo == "剖面分析")
            {
                GSOFeature feature = globeControl1.Globe.SelectedObject;
                if (feature == null ||
                    feature.Geometry == null ||
                    feature.Geometry.Type != EnumGeometryType.GeoPolyline3D)
                {
                    MessageBox.Show("请选择一条线");
                    return;
                }
                GSOGeoPolyline3D line = feature.Geometry as GSOGeoPolyline3D;
                FrmAnalysisProfileBaseLine frm = new
FrmAnalysisProfileBaseLine(globeControl1, line);
                frm.Show(this);
            }
        }
    }
};
```

11.4 通视分析

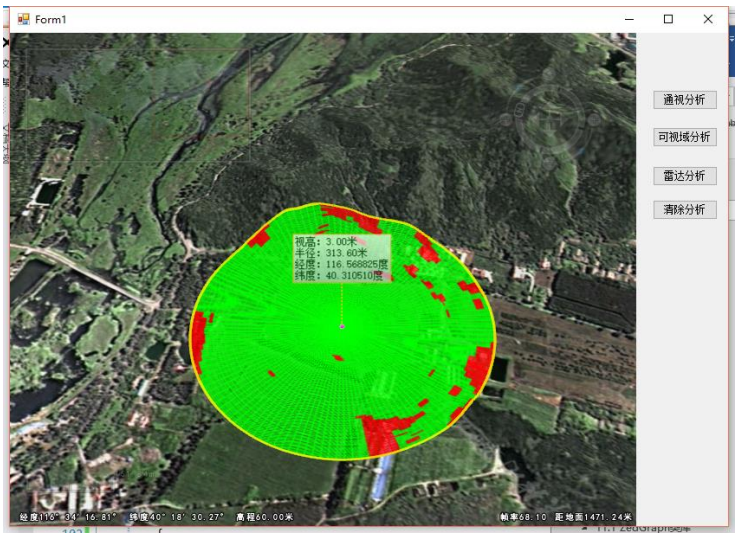
通视分析，即两点之间是否可视的分析。本分析底层有方法，只需要改变鼠标动作即可。

```
//通视分析
globeControl1.Globe.Action = EnumAction3D.VisibilityAnalysis;
```

11.5 可视域分析

可视域分析，即通过一个点和一个半径，查看视野可以查看地表的范围。本分析底层有方法，只需要改变鼠标动作即可。

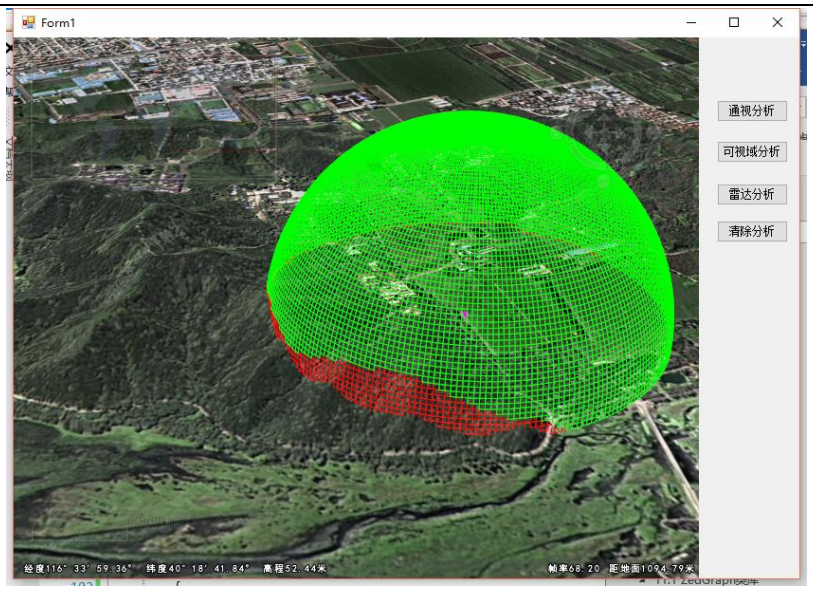
```
//可视域分析
globeControl1.Globe.Action = EnumAction3D.ViewshedAnalysis;
```



11.6 雷达分析

雷达分析，即即通过一个点和一个半径，查看视野可以查看空间的范围。本分析底层有方法，只需要改变鼠标动作即可。

```
//雷达分析
globeControl1.Globe.Action = EnumAction3D.ViewEnvelopeAnalysis;
```



11.7 淹没模拟

淹没模拟，即可以在地图上，根据地形模拟水位高度变化的淹没程度。

首先我们先改变一下鼠标动作：

```
string trackPolygonType = null;
```

```
/// <summary>
/// 淹没模拟
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, EventArgs e)
{
    globeControl1.Globe.Action = GeoScene.Data.EnumAction3D.TrackPolygon;
    globeControl1.Globe.TrackPolygonTool.TrackMode =
GeoScene.Globe.EnumTrackMode.GroundTrack;
    trackPolygonType = "淹没模拟";
}
```

范围绘制结束事件：

```
globeControl1.TrackPolygonEndEvent += (sender, e) =>
{
    if (trackPolygonType == "淹没模拟")
    {
        globeControl1.Globe.Action = EnumAction3D.ActionNull;
        FrmAnalysisFloodSubmerge frm = new
FrmAnalysisFloodSubmerge(globeControl1, e.Polygon);
        frm.Show(this);
    }
};
```

接下来我们绘制淹没模拟界面：

页面构建事件：

```
public FrmAnalysisFloodSubmerge(GSOGlobeControl _globeControl1,
GSOGeoPolygon3D _polygon3D)
{
```

```
    InitializeComponent();
    //默认20次每秒
    textBoxFrequency.Text = "20";

    globeControl1 = _globeControl1;
    m_polygon3D = _polygon3D;
}
```

页面 Load 事件，初始化：

```
private void FloodSubmergeAnalysisDlg_Load(object sender, EventArgs e)
{
    timerPlay.Stop();   //停止计时器
    Analysis();         //开始分析
    SetText();          //显示文字
    ShowWater();        //显示水面
    buttonSetPlayParam_Click(null, null);   //推荐参数
}
```

```
/// <summary>
/// 分析
/// </summary>
private void Analysis()
{
    if (globeControl1.Globe != null && m_polygon3D != null)
    {
        m_dBaseAlt = Convert.ToDouble(textBoxWaterAlt.Text);
        globeControl1.Globe.Analysis3D.NoSourceFloodAnalyse(m_polygon3D,
m_dBaseAlt,out m_dFloodArea, out m_dTotalArea,
            out m_pntMaxAlt, out m_pntMinAlt, false,0);
    }
}
```

```
/// <summary>
/// 设置文字
/// </summary>
private void SetText()
{
    textBoxPntMax.Text = m_pntMaxAlt.X.ToString("f6") + "," +
m_pntMaxAlt.Y.ToString("f6") + "," + m_pntMaxAlt.Z.ToString("f2");
    textBoxPntMin.Text = m_pntMinAlt.X.ToString("f6") + "," +
m_pntMinAlt.Y.ToString("f6") + "," + m_pntMinAlt.Z.ToString("f2");
    textBoxFloodArea.Text = m_dFloodArea.ToString("f2") + " 平方米";
    textBoxTotalArea.Text = m_dTotalArea.ToString("f2") + " 平方米";
}
```

```
/// <summary>
```

```
/// 显示水面
/// </summary>
private void ShowWater()
{
    if (m_WaterFeature==null || m_WaterFeature.IsDeleted)
    {
        //水面高度先设置为最低点高度
        m_dBaseAlt = m_pntMinAlt.Z;
        GSOGeoWater geoWater = m_polygon3D.ConvertToGeoWater(); //根据范围绘制
水面模型
        GSOExtrudeStyle extrudeStyle = new GSOExtrudeStyle();   //设置Style
        if (checkBoxExtrude.Checked)
        {
            extrudeStyle.ExtrudeType = EnumExtrudeType.ExtrudeToValue;
        }
        else
        {
            extrudeStyle.ExtrudeType = EnumExtrudeType.ExtrudeNone;
        }

        extrudeStyle.ExtrudeValue = m_pntMinAlt.Z;
        extrudeStyle.TailPartVisible = false;

        GSOSimplePolygonStyle3D polygonStyle = new GSOSimplePolygonStyle3D();
        polygonStyle.FillColor = Color.FromArgb(200, 0, 0, 255);

        extrudeStyle.BodyStyle = polygonStyle;

        geoWater.ExtrudeStyle = extrudeStyle;

        geoWater.AltitudeMode = EnumAltitudeMode.Absolute;  //高度类型
        geoWater.SetAltitude(m_dBaseAlt);   //设置高度
        geoWater.ReflectSky = false;        //是否反射天空
        geoWater.WaveWidth = 0.1;           //浪的高度
        geoWater.Play();                   //开始动画

        //添加水面要素
        m_WaterFeature = new GSOFeature();
        m_WaterFeature.Geometry = geoWater;
        globeControl1.Globe.MemoryLayer.AddFeature(m_WaterFeature);
        globeControl1.Globe.Refresh();

        trackBarAlt.Maximum = (int)m_pntMaxAlt.Z;
```

```
        trackBarAlt.Minimum = (int)m_pntMinAlt.Z;
        trackBarAlt.Value = trackBarAlt.Minimum;
        textBoxWaterAlt.Text = m_dBaseAlt.ToString("f2");
    }
    else
    {
        GSOGeoWater geoWater = (GSOGeoWater)m_WaterFeature.Geometry;
        geoWater.SetAltitude(m_dBaseAlt);
        globeControl1.Globe.Refresh();

        trackBarAlt.Maximum = (int)m_pntMaxAlt.Z;
        trackBarAlt.Minimum = (int)m_pntMinAlt.Z;
    }
}
```

```
/// <summary>
/// 推荐参数
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonSetPlayParam_Click(object sender, EventArgs e)
{
    //播放频率
    int nF = Convert.ToInt32(textBoxFrequency.Text);
    if (nF <= 0)
    {
        nF = 1;
    }
    //最大值为最高点
    numericUpDownAddPerTime.Maximum = (decimal)m_pntMaxAlt.Z;
    numericUpDownAddPerTime.Minimum = 0;
    double dAltDiff = (m_pntMaxAlt.Z - m_pntMinAlt.Z);  //绝对高度

    numericUpDownAddPerTime.Increment = (decimal)(0.1*dAltDiff / nF); // 用10
秒播放完
    numericUpDownAddPerTime.Value = numericUpDownAddPerTime.Increment;

    trackBarAlt.Maximum = (int)m_pntMaxAlt.Z;   //滑块最大为最高点
    trackBarAlt.Minimum = (int)m_pntMinAlt.Z;   //滑块最小为最低点
    trackBarAlt.Value = trackBarAlt.Minimum;    //默认为最小处

    int nTF = (int)numericUpDownAddPerTime.Increment;   //如果滑块每格的值小于
1，这换成1
    if (nTF < 1)
```

```
    {
        nTF = 1;
    }
    trackBarAlt.TickFrequency = nTF;

    m_dBaseAlt = m_pntMinAlt.Z;
    textBoxWaterAlt.Text = m_dBaseAlt.ToString("f2");
}
```

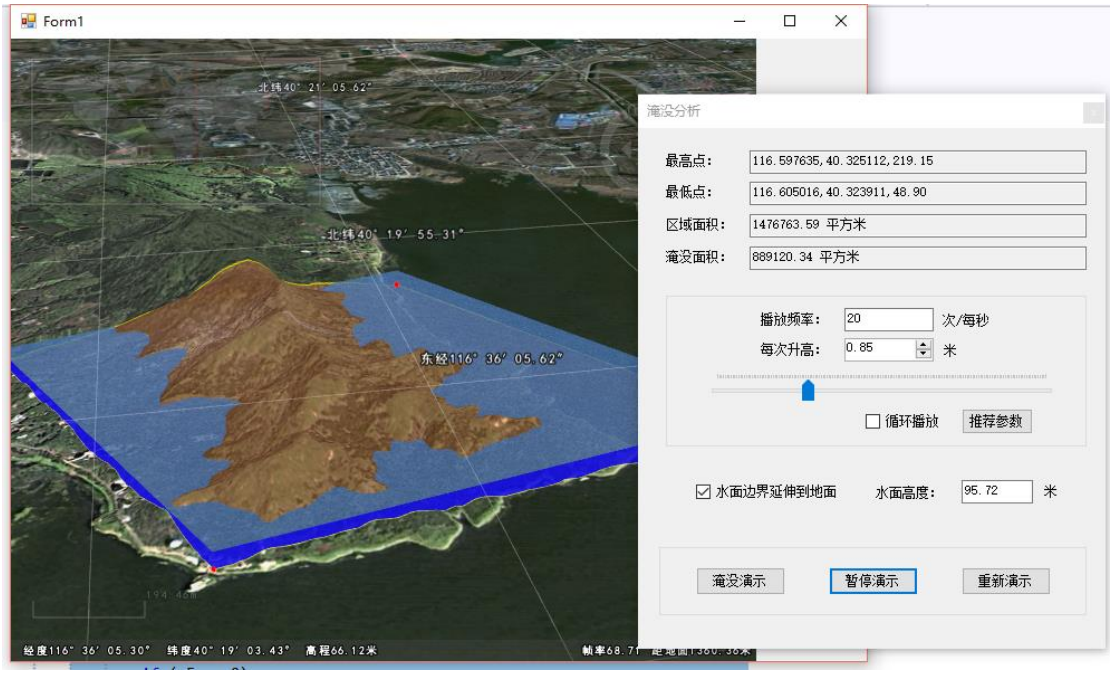
创建一个计时器，用来模拟不断上升的水面：

```
/// <summary>
/// 创建个计时器
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timerPlay_Tick(object sender, EventArgs e)
{
    if (m_WaterFeature!=null)
    {
        if (m_dBaseAlt>m_pntMaxAlt.Z)
        {
            if (checkBoxLoopPlay.Checked)
            {
                m_dBaseAlt = m_pntMinAlt.Z;
            }
            else
            {
                timerPlay.Stop();
            }
        }
        //每秒升高
        m_dBaseAlt += (double)numericUpDownAddPerTime.Value;

globeControl1.Globe.Analysis3D.FetchNoSourceFloodAnalyseResult(m_dBaseAlt,
out m_dFloodArea, out m_dTotalArea,
     out m_pntMaxAlt, out m_pntMinAlt);

        SetText();

        textBoxWaterAlt.Text = m_dBaseAlt.ToString("f2");
        trackBarAlt.Value = Math.Min((int)m_dBaseAlt, trackBarAlt.Maximum);
        GSOGeoWater geoWater = (GSOGeoWater)m_WaterFeature.Geometry;
        geoWater.SetAltitude(m_dBaseAlt);
    }
```

```
}
```

开始淹没模拟：

```
/// <summary>
/// 开始演示
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonPlay_Click(object sender, EventArgs e)
{
    if (m_WaterFeature != null)
    {
        //水面高度
        m_dBaseAlt = Convert.ToDouble(textBoxWaterAlt.Text);
        //判断不能超过最大最小值
        if (m_dBaseAlt > m_pntMaxAlt.Z)
        {
            m_dBaseAlt = m_pntMaxAlt.Z;
        }
        if (m_dBaseAlt < m_pntMinAlt.Z)
        {
            m_dBaseAlt = m_pntMinAlt.Z;
        }
        textBoxWaterAlt.Text = m_dBaseAlt.ToString("f2");
        trackBarAlt.Value = Math.Min((int)m_dBaseAlt, trackBarAlt.Maximum);
        GSOGeoWater geoWater = (GSOGeoWater)m_WaterFeature.Geometry;
        geoWater.SetAltitude(m_dBaseAlt);

        int nF = Convert.ToInt32(textBoxFrequency.Text);
        if (nF <= 0)
        {
            nF = 1;
        }
        timerPlay.Interval = (int)(1000.0 / nF);
        timerPlay.Start();
    }
}
```



我们也可以通过选择一个面来进行淹没模拟：

```
/// <summary>
/// 选择面
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{
    globeControl1.Globe.Action = EnumAction3D.SelectObject;
    trackPolygonType = "淹没模拟";
}
```

```
//记录鼠标落下的坐标
globeControl1.MouseDown += (sender, e) =>
{
    mouseDown = e.Location;
};

//判断鼠标抬起时，坐标位置不变
globeControl1.MouseUp += (sender, e) =>
{
    if (mouseDown == e.Location)
    {
        if (e.Button == MouseButtons.Left)
        {
            if (globeControl1.Globe.Action == EnumAction3D.SelectObject)
```

```
            {
                if (trackPolygonType == "淹没模拟")
                {
                    GSOFeature feature = globeControl1.Globe.SelectedObject;
                    if (feature == null||feature.Geometry == null
                        || feature.Geometry.Type !=
EnumGeometryType.GeoPolygon3D)
                    {
                        MessageBox.Show("请选择一个面");
                        return;
                    }
                    GSOGeoPolygon3D ePolygon = feature.Geometry as
GSOGeoPolygon3D;
                    globeControl1.Globe.Action = EnumAction3D.ActionNull;
                    FrmAnalysisFloodSubmerge frm = new
FrmAnalysisFloodSubmerge(globeControl1,ePolygon);
                    frm.Show(this);
                }
            }
        }
    }
};
```

11.8 缓冲区创建

缓冲区创建，是可以在原有要素的基础上，创建一个圆滑的缓冲区。

缓冲区首先先选中对象：

```
string mouseClickType = null;
```

```
/// <summary>
/// 创建缓冲区
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, System.EventArgs e)
{
    globeControl1.Globe.Action = EnumAction3D.SelectObject;
    mouseClickType = "缓冲区创建";
}
```

然后获得根据获得的要素进行缓冲区的创建：

```
//鼠标按下事件
globeControl1.MouseDown += (sender, e) =>
{
    MouseDown = e.Location;
};

//鼠标抬起事件
globeControl1.MouseUp += new MouseEventHandler(globeControl1_MouseUp);


private void globeControl1_MouseUp(object sender, MouseEventArgs e)
{
    if (MouseDown != e.Location) return;

    if (e.Button == MouseButtons.Left)
    {
        //开始判断是否是缓冲区分析
        if (globeControl1.Globe.Action == EnumAction3D.SelectObject)
        {
            if (mouseClickType == "缓冲区创建")
            {
                //获取选中要素
                GSOFeature feature = globeControl1.Globe.SelectedObject;
                if (feature == null || feature.Geometry == null)
                {
                    MessageBox.Show("请选择一个目标");
                    return;
                }
                if (feature.Geometry.Type != EnumGeometryType.GeoMarker
                        && feature.Geometry.Type !=
EnumGeometryType.GeoPolyline3D
                        && feature.Geometry.Type !=
EnumGeometryType.GeoPolygon3D)
                {
                    MessageBox.Show("请选择一个点、线、面对象！", "提示");
                    return;
                }
                globeControl1.Globe.Action = EnumAction3D.ActionNull;
                makeBuffer(feature);
            }
        }
    }
}
```

```
private void makeBuffer(GSOFeature feature)
{
    double radius = (double) numericUpDownRadius.Value; //缓冲区宽度
    double value = (double) numericUpDownFenDuan.Value; //圆角角度
    bool isRoundCorner = CBRoundCorner.Checked;         //拐角是否圆滑
    bool isRoundEnds = CBRoundEnds.Checked;             //两端是否圆滑

    GSOGeoPolygon3D buffer = null;  //创建缓冲面
    if (feature.Geometry.Type == EnumGeometryType.GeoMarker)//如果要素为点
    {
        GSOGeoMarker marker = feature.Geometry as GSOGeoMarker;
        buffer = marker.CreateBuffer(radius, value, false);
        //创建点的缓冲面（宽度，角度，false）
    }
    else if (feature.Geometry.Type == EnumGeometryType.GeoPolyline3D)
    {
        GSOGeoPolyline3D line = feature.Geometry as GSOGeoPolyline3D;
        buffer = line.CreateBuffer(radius, isRoundCorner, value, isRoundEnds,
false);
        //创建线的缓冲面（宽度，拐角圆滑，角度，两端圆滑，false）
    }
    else if (feature.Geometry.Type == EnumGeometryType.GeoPolygon3D)
    {
        GSOGeoPolygon3D polygon = feature.Geometry as GSOGeoPolygon3D;
        buffer = polygon.CreateBuffer(radius, isRoundCorner, value,
isRoundEnds, false);
        //创建面的缓冲面（宽度，拐角圆滑，角度，两端圆滑，false）
    }
    if (buffer != null)
    {
        //缓冲面颜色
        GSOSimplePolygonStyle3D style = new GSOSimplePolygonStyle3D();
        style.FillColor = Color.FromArgb(122, Color.Yellow);
        //缓冲线颜色
        GSOSimpleLineStyle3D outlineStyle = new GSOSimpleLineStyle3D();
        outlineStyle.LineColor = Color.Red;
        style.OutlineStyle = outlineStyle;
        buffer.Style = style;
        GSOFeature featureBuffer = new GSOFeature();
        featureBuffer.Geometry = buffer;
        featureBuffer.Name = feature.Name + "_" + radius + "米_Buffer";
        globeControl1.Globe.MemoryLayer.AddFeature(featureBuffer);
```

```
        //重新创建原有要素,使得要素在缓冲面上方，可选择
        if (feature != null && feature.Dataset.Type == EnumDatasetType.Memory
&& feature.Dataset.Caption == "" && feature.Dataset.Name == "")
        {
            GSOFeature featureSelectCopy = feature.Clone();
            globeControl1.Globe.MemoryLayer.AddFeature(featureSelectCopy);
            feature.Delete();
        }
        globeControl1.Refresh();
    }
}
```

11.9 3D 可视域分析

3D 可视域分析，是一种依附于模型的可视域分析。它可以多个联动，来分辨可视区域和不可视区域。

首先，我们需要创建几个公共对象：

```
//3D可视域分析
//第一个分析
GSOViewshed3DAnalysis curViewshed3DAnalysis;
//其他分析
GSOViewshed3DAnalysis curViewshed3DAnalysisOther;
//其他判断
Boolean bViewshed3DAnalysisCheck = false;
Boolean bViewshed3DAnalysising = false;
GSOPoint3d pntDistDirPoint;
GSOPoint3d pntViewerPos;
```

可视域分析按钮事件：

```
//如果第一个分析为空，则为第一个分析
if (curViewshed3DAnalysis == null)
{
    curViewshed3DAnalysis = new GSOViewshed3DAnalysis(globeControl1.Globe);
}
//否则为其他分析
else
{
    curViewshed3DAnalysisOther = new
GSOViewshed3DAnalysis(globeControl1.Globe);
}
bViewshed3DAnalysisCheck = true;
globeControl1.Refresh();
```

还得添加鼠标按下事件和鼠标移动事件：

```
/// <summary>
/// 可视域分析，当鼠标按下
/// </summary>
/// <param name="sender"></param>
/// <param name="mouseEventArgs"></param>
private void GlobeControl1OnMouseDown(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
```

```
        //如果进入可视域分析状态
        if (bViewshed3DAnalysisCheck)
        {
            GSOLayer resLayer = null;
            GSOPoint3d resIntersectPoint;
            //测试点击，返回点击图层和点位置信息
            globeControl1.Globe.HitTest(e.X, e.Y, out resLayer, out
resIntersectPoint, false, true, 0);

            //如果为第一次分析，并且为设置视点（第一次点击）
            if (curViewshed3DAnalysis != null && !bViewshed3DAnalysising &&
                curViewshed3DAnalysisOther == null)
            {
                //设置可视域分析视点
                curViewshed3DAnalysis.ViewerPosition = resIntersectPoint;
                pntViewerPos = resIntersectPoint;
                bViewshed3DAnalysising = true;
            }
            //如果为第二次分析，设置第二次分析视点（第一次点击）
            else if (curViewshed3DAnalysisOther != null
&& !bViewshed3DAnalysising)
            {
                curViewshed3DAnalysisOther.ViewerPosition = resIntersectPoint;
                pntViewerPos = resIntersectPoint;
                bViewshed3DAnalysising = true;
            }
            //如果不是第一次点击
            else
            {
                //如果其他分析为空，则为第一次分析，设置视线点（第二次点击）
                if (curViewshed3DAnalysisOther == null)
                {
                    curViewshed3DAnalysis.SetDistDirByPoint(resIntersectPoint);
                }
                //如果其他分析不为空，为第二次分析，设置视线点并且与第一次分析的关联
（第二次点击）
                if (curViewshed3DAnalysisOther != null)
                {

curViewshed3DAnalysisOther.SetDistDirByPoint(resIntersectPoint);
                    //curViewshed3DAnalysisOther.VisibleAreaColor =
Color.FromArgb(128, 255, 255, 0);
                    //curViewshed3DAnalysisOther.HiddenAreaColor =
```

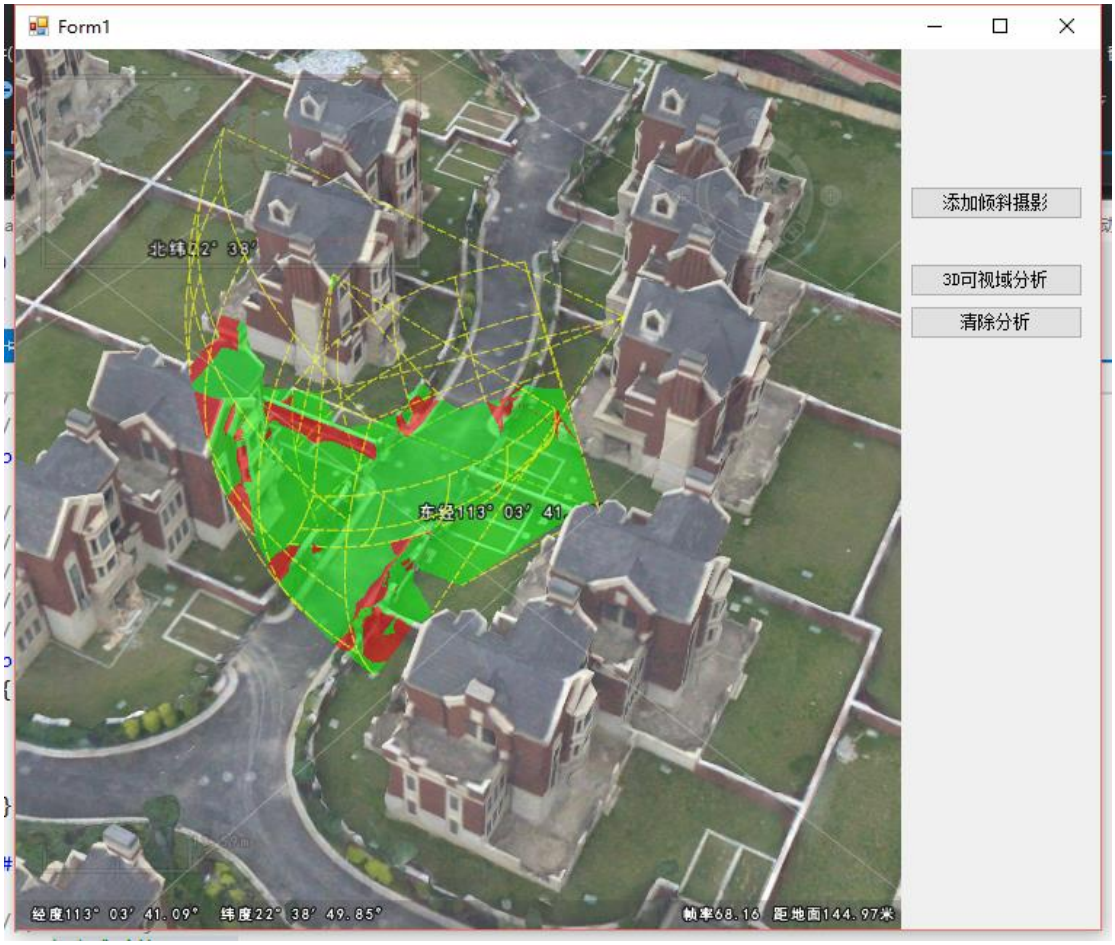
```
Color.FromArgb(128, 0, 0, 255);

                    //用第一个分析关联其他可视域分析
                    //如果有三个可视域分析，只能A关联B，A关联C。！！不能A关联B，B关联C！！

curViewshed3DAnalysis.AttachViewshed3DAnalysis(curViewshed3DAnalysisOther);
                }
                //关闭可视域分析状态
                bViewshed3DAnalysisCheck = false;
                bViewshed3DAnalysising = false;
            }
        }
    }
}
```

```
/// <summary>
/// 鼠标移动的时候
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void GlobeControl1OnMouseMove(object sender, MouseEventArgs e)
{
    //3D可视域分析
    {
        if (bViewshed3DAnalysisCheck && bViewshed3DAnalysising)
        {
            //如果是第一次分析
            if (curViewshed3DAnalysisOther == null)
            {
                GSOLayer resLayer = null;
                GSOPoint3d resIntersectPoint;
                globeControl1.Globe.HitTest(e.X, e.Y, out resLayer, out resIntersectPoint, false, true, 0);
                pntDistDirPoint = resIntersectPoint;
                curViewshed3DAnalysis.SetDistDirByPoint(resIntersectPoint);
            }
                //如果是其他分析
            else
            {
                GSOLayer resLayer = null;
                GSOPoint3d resIntersectPoint;
                globeControl1.Globe.HitTest(e.X, e.Y, out resLayer, out resIntersectPoint, false, true, 0);
```

```
                pntDistDirPoint = resIntersectPoint;

curViewshed3DAnalysisOther.SetDistDirByPoint(resIntersectPoint);
            }
        }
    }
}
```

还要添加一个清楚分析的事件：

```
globeControl1.Globe.ClearAnalysis();
//清空可视域分析
curViewshed3DAnalysis = curViewshed3DAnalysisOther = null;
```



11.10 挖坑

挖坑分析是运用于需要看到地面下的要素或者演示所使用。

首先，我们初始化按钮事件：

```
//绘制面
globeControl1.Globe.Action = EnumAction3D.TrackPolygon;
globeControl1.Globe.TrackPolygonTool.TrackMode = EnumTrackMode.GroundTrack;
TracPolygonType = "挖坑";

globeControl1.Refresh();
```

然后我们添加绘制面完成事件：

```
private void GlobeControl1OnTrackPolygonEndEvent(object sender,
TrackPolygonEndEventArgs e)
{
    GSOGeoPolygon3D polygon = e.Polygon;
    if (TracPolygonType == "挖坑")
    {
        //创建坑对象
        GSOGeoPit pit = new GSOGeoPit();
        pit.PitDepthUsing = true;   //是否使用深度
        pit.PitDepth = 10;          //深度
        pit.PitPolygon = polygon;   //坑的面
        globeControl1.Globe.AddPit("坑1",pit);
    }
}
```

这样就完成了一个坑。

*需要注意的是，挖的坑是根据当前地形生成的，如果地形网格精度变化的话，挖的坑也会出现问题，解决办法就是：如果地形精度变换，那么重新挖一个坑就可以了。

十二、输出

12.1 高清截图

本 SDK 提供底层的高清截图功能，可以根据当前球的大小进行缩放截图。

具体实现：

首先，为了防止截图比例不对，我们要获取当前球的 Panel 的大小，并且与我们设置放大倍数相乘：

```
//获得输出大小
float outWidth = (float)(panel1.Width*numericUpDown1.Value);
float outHeight = (float) (panel1.Height*numericUpDown1.Value);
//球输出（输出长像素，输出宽像素，输出地址）
bool isOK = globeControl1.Globe.OutputHighPic(outWidth, outHeight,
Application.StartupPath + "/images/test.jpg");
```

这样的话，就可以输出一张图片，输出图片支持*.PNG，*.BMP，*.JPG 格式。

12.2 影像拼接以及无效值过滤

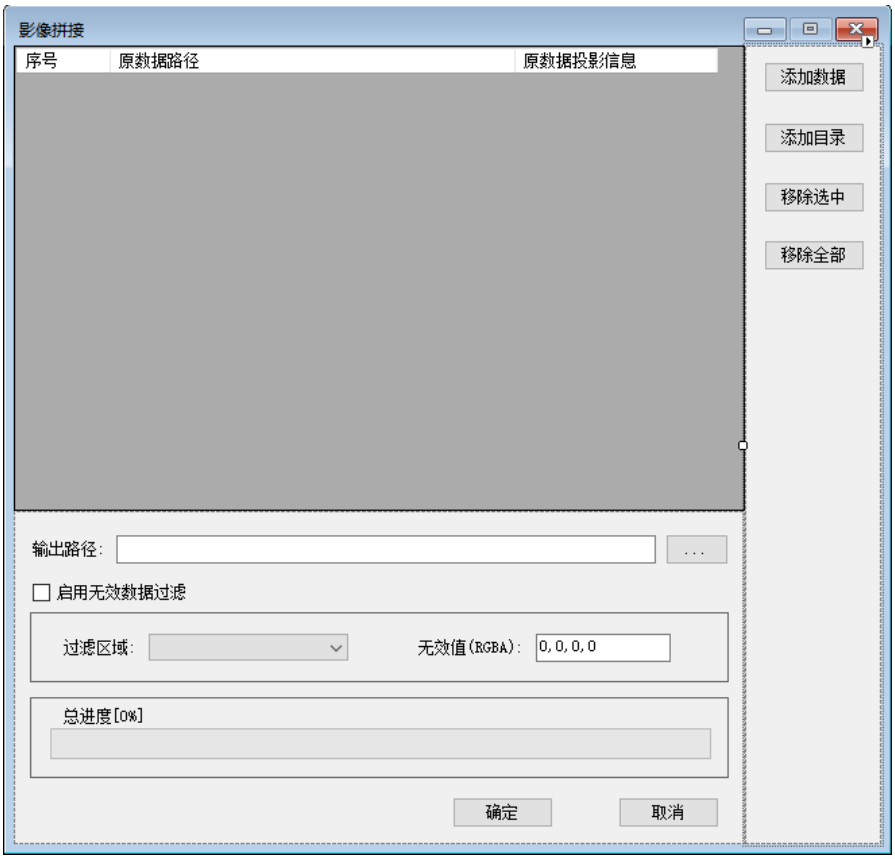
影像拼接功能可对多幅影像进行批量拼接，无效值过滤可以对影像中的无效 RGB 值进行过滤，以达到去影像黑边或白边。

首先在这个项目里，我们添加了以下新 DLL，有的大家可以选择添加，不是必须添加的。其中 DLL 为：

1. Microsoft.WindowsAPICodePack.dll
2. Microsoft.WindowsAPICodePack.Shell
3. PresentationFramework.dll

其中(1)(2)用作替换原生文件夹打开控件，使用方便。而(3)为帮助此控件获取当前句柄，可以选择添加

首先我们创建一个界面：



首先，我们先创建一个窗口，传入我们球控件。

```
GSOGlobeControl globeControl1;
public FrmDataJoinDom(GSOGlobeControl _globeControl1)
{
    InitializeComponent();
    globeControl1 = _globeControl1;
}
```

再将添加“添加数据”、“添加目录”按钮的代码，将文件数据添加到 GridView 中：

```
/// <summary>
/// 添加数据事件
/// </summary>
/// <param name="strDataPath">文件路径</param>
private void AddSrcData(string strDataPath)
{
    string strExt = Path.GetExtension(strDataPath);
    for (int i = 0; i < dataGridViewDomList.Rows.Count; i++)
    {
       object value = dataGridViewDomList.Rows[i].Cells[1].Value;
       if (strDataPath.Equals(value.ToString()) == true)
       {
           MessageBox.Show("文件已添加，不能重复添加同一份数据！","提示");
```

```
            return;
        }
    }

    int rowIndex = dataGridViewDomList.Rows.Add();
    dataGridViewDomList.Rows[rowIndex].Cells[0].Value = rowIndex + 1;
    dataGridViewDomList.Rows[rowIndex].Cells[1].Value = strDataPath;
    if (strExt.ToLower().Trim() == ".lrp")
    {
        dataGridViewDomList.Rows[rowIndex].Cells[2].Value = "经纬度数据";
    }
    else
    {
        string proj4 = GSODataEngineUtility.GetProj4FromDataFile(strDataPath);
        if (proj4 != "")
        {
            dataGridViewDomList.Rows[rowIndex].Cells[2].Value = "经纬度数据";
        }
        else if (proj4 == "")
        {
            dataGridViewDomList.Rows[rowIndex].Cells[2].Value = "无投影信息数据
";
        }
    }
}
/// <summary>
/// "添加数据"按钮
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonAddData_Click(object sender, EventArgs e)
{
    OpenFileDialog dlg = new OpenFileDialog();
    dlg.Filter = "支持格式
(*.lrp,*.tif,*.tiff,*.img,*.asc,*.raw,*.dem,*.adf,*.idr,*.sid,*.ecw,*.e00,*.e
rs,*.hdr,*.grd)|" +

"*.lrp;*.tif;*.tiff;*.img;*.asc;*.raw;*.dem;*.adf;*.idr;*.sid;*.ecw;*.e00;*.e
rs;*.hdr;*.grd|*.tif,*.tiff|*.tif;*.tiff|*.img|*.img|其它格式(*.*)|*.*";
    dlg.Multiselect = true;
    dlg.Title = "添加拼接数据";
    if (dlg.ShowDialog(this) == System.Windows.Forms.DialogResult.OK)
    {
```

```
        for (int i = 0; i < dlg.FileNames.Length; i++)
        {
            AddSrcData(dlg.FileNames[i]);
        }
    }
}
/// <summary>
/// 添加文件夹按钮
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonAddDataByDirectory_Click(object sender, EventArgs e)
{
    //调用DLL的新控件
    Microsoft.WindowsAPICodePack.Dialogs.CommonOpenFileDialog dlg = new
Microsoft.WindowsAPICodePack.Dialogs.CommonOpenFileDialog();
    dlg.IsFolderPicker = true;
    dlg.Multiselect = false;
    dlg.Title = "选择目录";
    if (dlg.ShowDialog(this.Handle) ==
Microsoft.WindowsAPICodePack.Dialogs.CommonFileDialogResult.Ok)
    {
        string directoryPath = dlg.FileName;
        string[] fileNames = Directory.GetFiles(directoryPath);
        for (int i = 0; i < fileNames.Length; i++)
        {
            AddSrcData(fileNames[i]);
        }
    }
}
```

开始转换的按钮事件：

```
private void buttonOK_Click(object sender, EventArgs e)
{
    if (dataGridViewDomList.Rows.Count == 0)
    {
        MessageBox.Show("请先添加数据！", "提示");
        return;
    }
    //输出路径
    string dataPatho_new = textBoxOutputPath.Text.Trim();
    if (dataPatho_new == "")
    {
```

```
        MessageBox.Show("请选择输出路径！","提示");
        return;
    }
    int r = 0;
    int g = 0;
    int b = 0;
    int a = 0;
    if (checkBoxUsingFilter.Checked)
    {
        string imageNoValue = textBoxImageNoValue.Text.Trim();
        string[] array = imageNoValue.Split(',');
        if (array.Length != 4)
        {
            MessageBox.Show("无效值不符合要求！", "提示");
            return;
        }
        else
        {
            if (int.TryParse(array[0], out r) == false
                || int.TryParse(array[1], out g) == false
                || int.TryParse(array[2], out b) == false
                || int.TryParse(array[3], out a) == false)
            {
                MessageBox.Show("无效值不符合要求！", "提示");
                return;
            }
        }
    }

    this.buttonOK.Enabled = false;
    this.buttonCancel.Enabled = false;
    this.panel1.Enabled = false;
    this.Cursor = Cursors.WaitCursor;
    GSORasterMerge rasterMerge = new GSORasterMerge();
    //无效值过滤选择按钮可用
    if (checkBoxUsingFilter.Checked == true)
    {
        //无效值过滤模式
        rasterMerge.SetUseNoValueFilterMode(comboBoxFilterRegion.SelectedIndex
+ 1);
    }
    if (checkBoxUsingFilter.Checked)
    {
```

```
        //无效值
        rasterMerge.SetImageNoValue(r, g, b, a);
    }
    //拼接进程
    rasterMerge.Stepped  += new SteppedEventHandler(rasterMerge_Stepped);
    for (int i = 0; i < dataGridViewDomList.Rows.Count; i++)
    {
        string dataPatho_old =
dataGridViewDomList.Rows[i].Cells[1].Value.ToString();
        rasterMerge.AddSrcData(dataPatho_old);
    }
    //保存生成文件
    rasterMerge.SetSavePath(dataPatho_new);
    bool isSaveSuccess = rasterMerge.MergeToLRPImage();
    this.Cursor = Cursors.Default;
    if (isFormclosing == false)
    {
        MessageBox.Show("数据拼接成功！");
    }

    this.panel1.Enabled = true;
    this.Close();
}
```

就这样，一份拼接和过滤过的影像就生成了！