# ArcEngine+C#　TIN 相关三维功能模块介绍（四）功能综合演示

作者：刘志远

Email：592418843@qq.com

## 1.简介

本篇打算对 TIN 的介绍做个小结，把前面将的功能都集中到一个小程序中。除了前面将的功能外，本篇还添加了如下一些功能。只要有数据，这些功能都是可以单独使用的，方便读者根据具体的需要直接参考相应代码。

- ➢ 由矢量点或线数据，通过核函数密度制图方法生成栅格数据（类 DEM）；
- ➢ 按给定间距，从栅格数据（DEM）中提取矢量等高线数据；
- ➢ 由栅格数据（DEM）生成 TIN 模型；
- ➢ 动态变化 DEM、TIN 的渲染样式；
- ➢ 矢量图层叠加到 TIN 模型；

本文主要对这几个功能进行简单的讲解，希望对接触 TIN 的朋友有所帮助。具体的实现，可以参看程序中相应的代码。同时，在压缩包中有个简短的“使用说明”，可以通过该程序尝试讲过的功能。

## 2.核函数密度制度生成“类 DEM”

通常情况下，我们通过对高程点数据进行插值操作（如反距离权插值、样条插值、克里金插值等）生成 DEM（数字高程模型）栅格数据，生成的栅格数据的像元灰度值即代表对应的高程值。

本文介绍的是密度制图，生成的栅格灰度值代表密度大小，所以这里成为“类 DEM”。密度制图有核函数密度制图（Kernal）和简单密度制图（Simple）两种方法，本文用的是 Kernal 方法。对于二次开发来说，由矢量数据生成栅格数据，不论是 DEM 也好，还是密度制图，只是调用相应接口的某个方法而已，内部的具体操作都是封装好的，程序员只需根据实际需要进行选择即可。这也体现了面向对象的巨大优势。主要代码如下：

```
        /// <summary>
        /// 核函数密度制图
        /// </summary>
        /// <param name="pFeatureLayer">进行密度分析的矢量数据图层</param>
        /// <param name="cellSize">输出栅格格网大小</param>
        /// <param name="radius">制图搜索半径</param>
        /// <param name="path">栅格文件保存路径　可以为空，则作为临时数据处理</param>
        /// <returns></returns>
        public IRaster DensityAnalyst(IFeatureLayer pFeatureLayer, double cellSize, double
radius, string path)
        {
            IFeatureClass pClass01 = pFeatureLayer.FeatureClass;
            IDensityOp pDensityOp = new RasterDensityOpClass();
            IRasterAnalysisEnvironment pEnv = pDensityOp as IRasterAnalysisEnvironment;

            Double double_cellSize = cellSize;
            object object_cellSize = (System.Object)double_cellSize;
            pEnv.SetCellSize(esriRasterEnvSettingEnum.esriRasterEnvValue, ref
object_cellSize);
```

```
            IFeatureClassDescriptor pFDescr = new FeatureClassDescriptorClass();
            pFDescr.Create(pFClass01, null, "NONE");

            System.Double double_radio_dis = radius;
            object object_radio_dis = (System.Object)double_radio_dis;
            object Missing = Type.Missing;
            //调用核函数密度制图方法生成栅格数据
            IRaster pRasOut1 = pDensityOp.KernelDensity(pFDescr as IGeoDataset, ref
object_radio_dis, ref Missing) as IRaster;

            //栅格值重计算，将栅格值缩放到合理范围（主要是纠正由于坐标系统不同而造成的数据夸
张变大）
            //如数据的坐标系等没有问题则可以不用重计算
            IMapAlgebraOp pMapAlgebraOp = (IMapAlgebraOp)new RasterMapAlgebraOp();
            IRasterAnalysisEnvironment pRasAnaEnv =
(IRasterAnalysisEnvironment)pMapAlgebraOp;
            pMapAlgebraOp.BindRaster(pRasOut1 as IGeoDataset, "R");
            string expresion = "[R] * 0.0000915"; //坐标系统转换参数
            IRaster pRasOut = pMapAlgebraOp.Execute(expresion) as IRaster;

            //输出栅格数据，否则存放为临时数据
            if (path != "")
            {
                string foldPath = System.IO.Path.GetDirectoryName(path);
                string rasterName = System.IO.Path.GetFileName(path);
                try
                {
                    IRasterBandCollection bandCollection = (IRasterBandCollection)pRasOut;
                    IRasterDataset dataset = bandCollection.Item(0).RasterDataset;
                    ITemporaryDataset temp = (ITemporaryDataset)dataset;
                    RasterWorkspaceFactory factory = new RasterWorkspaceFactory();
                    IWorkspace workSpace = factory.OpenFromFile(foldPath, 0);
                    temp.MakePermanentAs(rasterName, workSpace, "IMAGINE Image");
                }
                catch
                {
                    return pRasOut;
                }
            }
            return pRasOut;
        }
```

## 3.DEM 提取等高线

根据指定间隔，从 DEM 栅格数据中提取等高线数据。主要用了 ISurfaceOp 接口下的 Contour（）方法。主要代码如下：

```
    ISurfaceOp pSurfaceOP = new RasterSurfaceOpClass();
    IFeatureLayer pFL = new FeatureLayerClass();
    object Missing = Type.Missing;
    pFL.FeatureClass = pSurfaceOP.Contour(iRaster as IGeoDataset, interval, ref Missing) as
IFeatureClass;
```

## 4.由 DEM 生成 TIN

```
//***************生成TIN模型*********************************************
IGeoDataset pGeoData = iRaster as IGeoDataset;
IEnvelope pExtent = pGeoData.Extent;
IRasterBandCollection pRasBC = iRaster as IRasterBandCollection;
IRasterBand pRasBand = pRasBC.Item(0);
IRawPixels pRawPixels = pRasBand as IRawPixels;
IRasterProps pProps = pRawPixels as IRasterProps;

int iWid = pProps.Width;
int iHei = pProps.Height;

double w = iWid / 1000.0f;
double h = iHei / 1000.0f;

IPnt pBlockSize = new DblPntClass();
bool IterationFlag;

if (w < 1 && h < 1) //横纵都小于1000个像素
{
    pBlockSize.X = iWid;
    pBlockSize.Y = iHei;
    IterationFlag = false;
}
else
{
    pBlockSize.X = 1001.0f;
    pBlockSize.Y = 1001.0f;
    IterationFlag = true;
}

double cellsize = 0.0f;     //栅格大小
```

```
                IPnt pPnt1 = pProps.MeanCellSize(); //栅格平均大小
                cellsize = pPnt1.X;

                ITinEdit pTinEdit = new TinClass() as ITinEdit;
                pTinEdit.InitNew(pExtent);

                ISpatialReference pSpatial = pGeoData.SpatialReference;
                pExtent.SpatialReference = pSpatial;

                IPnt pOrigin = new DblPntClass();
                IPnt pPixelBlockOrigin = new DblPntClass();

                //栅格左上角像素中心坐标
                double bX = pBlockSize.X;
                double bY = pBlockSize.Y;

                pBlockSize.SetCoords(bX, bY);
                IPixelBlock pPixelBlock = pRawPixels.CreatePixelBlock(pBlockSize);

                object nodata = pProps.NoDataValue;     //无值标记
                ITinAdvanced2 pTinNodeCount = pTinEdit as ITinAdvanced2;
                int nodeCount = pTinNodeCount.NodeCount;

                object vtMissing = Type.Missing;

                object vPixels = null;      //格子
                if (IterationFlag)   //当为一个处理单元格子时
                {
                    pPixelBlockOrigin.SetCoords(0.0f, 0.0f);
                    pRawPixels.Read(pPixelBlockOrigin, pPixelBlock);

                    vPixels = pPixelBlock.get_SafeArray(0);
                    double xMin = pExtent.XMin;
                    double yMax = pExtent.YMax;
                    pOrigin.X = xMin + cellsize / 2;
                    pOrigin.Y = yMax - cellsize / 2;
                    bX = pOrigin.X;
                    bY = pOrigin.Y;

                    pTinEdit.AddFromPixelBlock(bX, bY, cellsize, cellsize, nodata, vPixels,
m_zTolerance, ref vtMissing, out vtMissing);
                }
                else  //当有多个处理单元格时，依次循环处理每个单元格
                {
```

```
            int i = 0, j = 0, count = 0;
            int FirstGoNodeCount = 0;
            while (nodeCount != FirstGoNodeCount)
            {
                count++;
                nodeCount = pTinNodeCount.NodeCount;
                //依次循环处理
                for (i = 0; i < h + 1; i++)
                {
                    for (j = 0; j < w + 1; j++)
                    {
                        double bX1, bY1, xMin1, yMax1;
                        bX1 = pBlockSize.X;
                        bY1 = pBlockSize.Y;

                        pPixelBlockOrigin.SetCoords(j * bX1, i * bY1);
                        pRawPixels.Read(pPixelBlockOrigin, pPixelBlock);
                        vPixels = pPixelBlock.get_SafeArray(0);

                        xMin1 = pExtent.XMin;
                        yMax1 = pExtent.YMax;

                        bX1 = pBlockSize.X;
                        bY1 = pBlockSize.Y;

                        pOrigin.X = xMin1 + j * bX1 * cellsize + cellsize / 2.0f;
                        pOrigin.Y = yMax1 + i * bY1 * cellsize - cellsize / 2.0f;

                        bX1 = pOrigin.X;
                        bY1 = pOrigin.Y;

                        pTinEdit.AddFromPixelBlock(bX1, bY1, cellsize, cellsize,
nodata, vPixels, m_zTolerance, ref vtMissing, out vtMissing);

                        FirstGoNodeCount = pTinNodeCount.NodeCount;
                    }
                }
            }
        }

        //保存TIN文件
        pTinEdit.SaveAs(tinFileName, ref vtMissing);
        pTinEdit.StopEditing(true);
```

## 5.动态变化 DEM、TIN 的渲染样式

动态变化 DEM、TIN 的渲染样式其实是根据用户的设定重新生成对应的渲染样式而已，这里就不贴代码了，具体的可以参考程序。

## 6.矢量图层叠加到 TIN 模型

矢量图层叠加到 TIN 模型，其实是改变图层的属性，将图层中对应的 ILayerExtensions 接口变为 I3DProperties 接口，然后将 I3DProperties. BaseSurface 属性设为 TIN 图层表面数据。具体代码如下：

```
/// <summary>
/// 将矢量图层叠加到TIN模型上
/// </summary>
/// <param name="pLayer">要叠加的图层</param>
/// <param name="pSceneControl">三维控件</param>
public void setLayerToTIN(ILayer pLayer, AxSceneControl pSceneControl)
{
    I3DProperties p3DProperties;
    p3DProperties = get3DProps(pLayer);
    p3DProperties.BaseOption = esriBaseOption.esriBaseSurface;
    ISurface pSurface = getTinlayer(pSceneControl).Dataset as ISurface;
    p3DProperties.BaseSurface = pSurface;
    p3DProperties.Apply3DProperties(pLayer);
}

/// <summary>
/// 恢复到平常样式
/// </summary>
/// <param name="pLayer"></param>
/// <param name="pSceneControl"></param>
public void setLayerOutTIN(ILayer pLayer, AxSceneControl pSceneControl)
{
    I3DProperties p3DProperties;
    p3DProperties = get3DProps(pLayer);
    p3DProperties.BaseOption = esriBaseOption.esriBaseExpression;
    p3DProperties.Apply3DProperties(pLayer);
}

/// <summary>
/// 获得普通要素图层的3D属性
/// </summary>
/// <param name="pLayer">普通矢量图层</param>
```

```
        /// <returns></returns>
        public I3DProperties get3DProps(ILayer pLayer)
        {
            ILayerExtensions pLayerExts = pLayer as ILayerExtensions;
            if (pLayerExts != null)
            {
                for (int i = 0; i < pLayerExts.ExtensionCount; i++)
                {
                    I3DProperties p3DProps = pLayerExts.get_Extension(i) as I3DProperties;
                    if (p3DProps != null)
                        return p3DProps;
                }
            }
            return null;
        }

       /// <summary>
        /// 返回TIN数据图层
        /// </summary>
        /// <param name="pSceneControl">三维控件</param>
        /// <returns></returns>
        public ITinLayer getTinlayer(AxSceneControl pSceneControl)
        {
            ITinLayer pTinlayer = null;
            IScene map = pSceneControl.Scene;
            if (map == null)
                return null;
            for (int i = 0; i < map.LayerCount; i++)
            {
                ILayer lyr = map.get_Layer(i);
                if (lyr is ITinLayer)
                {
                    pTinlayer = lyr as ITinLayer;
                    break;
                }
            }
            return pTinlayer;
        }
```

## 7.总结

至此，关于 TIN 的一些介绍就到这告一段落，文中没有阐述清楚的部分可以参看程序中的代码。当然，其中不足之处还请各位留言批评指正。