ArcEngine+C# TIN 相关三维功能模块介绍（三）
——TreeView 控件控制 TIN 颜色

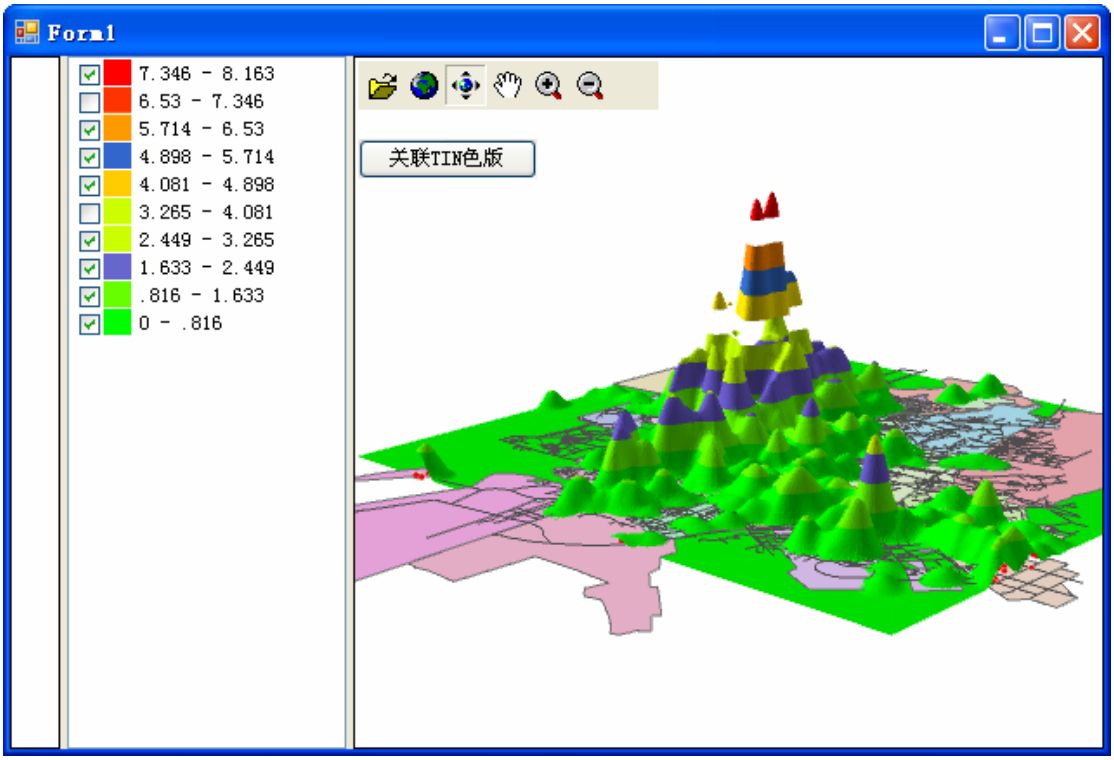
作者：刘志远

Email：592418843@qq.com

1.简介

本篇介绍如何写一个带复选框及颜色指示图标的 TreeView 控件，该控件与 TIN 模型分级设色方案中的各级颜色相关联。通过该 TreeView 控件复选框的勾选状态决定 TIN 模型对应颜色层的显示与隐藏，再结合颜色选择下拉框控件，双击 TreeView 中的颜色图标，弹出颜色选择对话框，选择颜色后即可实时改变 TIN 模型中对应分级的颜色，效果图如下。注意观察 TreeView 中复选框为空的颜色层对应 TIN 的样式，此外，那个蓝色和紫色的层是经过修改过后的结果：

颜色选择对话框



运行效果界面截图

2.思路分析

编写 TIN 颜色关联 TreeView 控件过程同上篇的编写颜色控件类似，先在项目下添加一个新的 UserControl 控件，再拖一个 TreeView 控件到 UserControl 上，将 TreeView 的 Dock 属性设为 Fill,，最后加入代码。

TIN 模型颜色可以通过 ITinRenderer 渲染器接口设置，ITinColorRampRenderer 接口控制渲染器中的色带，通过遍历获取或设置 ITinColorRampRenderer 中的颜色值，再将改变后的 ITinRenderer 重新赋给 TIN 模型就行了。隐藏效果只是将对应层的颜色值设为 NullColor 即可。

再稍微解释一下 TreeView 中颜色图标的实时联动更新。我是先生成一个动态的 Image 图片对象，该 Image 的颜色就是从 TIN 模型或颜色对话框中获得的颜色值，再依次将这些颜色图片存入一个 ImageList 对象中，再将该 ImageList 对象赋给 TreeView，通过 ImageList 中图片的索引与 TreeView 中的节点（Node）相关联，这样就达到实时更新显示的效果。

3.完整代码

下面是该控件的完整代码，具体的解释见代码注释：

```
using System;
using System.Drawing;
using System.Windows.Forms;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.Controls;
using ESRI.ArcGIS.Analyst3D;
namespace CaseAnalyse
{
    /// <summary>
    /// 模仿ArcGIS的TOCControl控件，对TIN模型分级渲染颜色进行关联，
    /// 并通过复选框的勾选状态控制对应色带的显隐，
    /// 同时可双击颜色图标，实时修改对应层的颜色
    /// </summary>
    public partial class ColorControlView : UserControl
    {
        /// <summary>
        /// 颜色图标清单
        /// </summary>
        private ImageList pImageList;

        /// <summary>
        /// 激活的节点
        /// </summary>
```

```
        private TreeNode pTreeNode;

        /// <summary>
        /// 含有TIN的三维控件
        /// </summary>
        static private AxSceneControl _pSceneControl;

        //构造函数
        public ColorControlView()
        {
            InitializeComponent();
            treeView1.CheckBoxes = true;
        }

        /// <summary>
        /// 初始化，关联TIN图层颜色列表与TreeView控件
        /// </summary>
        /// <param name="pSceneControl">含TIN图层的AxSceneControl控件</param>
        public void IniColorTree(AxSceneControl pSceneControl)
        {
            _pSceneControl = pSceneControl;

            ITinLayer pTinLayer = getTinLayer(_pSceneControl);
            if (pTinLayer == null)
                return;

            treeView1.Nodes.Clear();
            //给TreeView绑定一个图片列表控件，该控件记录着所有的颜色图片
            pImageList = new ImageList();
            pImageList.ImageSize = new Size(16, 15);
            treeView1.ImageList = pImageList;
            //获得TIN模型的渲染器，这里假设只有一个渲染层
            ITinRenderer pRenderNew = pTinLayer.GetRenderer(0);
            ITinColorRampRenderer pElevRenderer = pRenderNew as ITinColorRampRenderer;
            if (pElevRenderer == null)
                return;
            ISimpleFillSymbol pSymbol = null;

            Color pColor = new Color();
            string lable = "";
            //遍历渲染层的分级颜色，并用生成动态临时图标，将其加入ImageList中，以备TreeView
调用
            for (int i = 0; i < pElevRenderer.BreakCount; i++)
            {
```

```
            lable = pElevRenderer.get_Label(i);
            pSymbol = pElevRenderer.get_Symbol(i) as ISimpleFillSymbol;
            pColor = ColorTranslator.FromOle(pSymbol.Color.RGB);

            Image pImage = getImage(pColor);
            pImageList.Images.Add(pImage);

            TreeNode pTN = new TreeNode();
            pTN.Text = lable;
            pTN.Checked = true;
            pTN.ImageIndex = pImageList.Images.Count - 1;
            pTN.Tag = pColor;

            treeView1.Nodes.Add(pTN);
        }
    }

    /// <summary>
    /// 关联图标
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
    {
        treeView1.SelectedImageIndex = treeView1.SelectedNode.ImageIndex;
    }

    /// <summary>
    /// 复选框勾选事件
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void treeView1_AfterCheck(object sender, TreeViewEventArgs e)
    {
        //调用颜色显隐方法
        ShowOrHideColor();
        treeView1.SelectedNode = e.Node;
    }

    /// <summary>
    /// 双击节点事件
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
```

```
private void treeView1_DoubleClick(object sender, EventArgs e)
{
    if (treeView1.SelectedNode == null)
        return;

    pTreeNode = treeView1.SelectedNode;
    //调用改变颜色方法
    ChangeColor();
    //刷新颜色图标
    treeView1.SelectedNode = treeView1.Nodes[0];
    treeView1.SelectedNode = pTreeNode;
}

/// <summary>
/// 获得颜色图标
/// </summary>
/// <param name="clr">图标颜色</param>
/// <returns></returns>
private static Image getImage(Color clr)
{
    Panel pPanel = new Panel();
    pPanel.Visible = false;
    pPanel.Width = 16;
    pPanel.Height = 14;

    Bitmap pBitmap = new Bitmap(16, 14);
    Rectangle rc = new Rectangle(0, 0, 16, 14);
    pPanel.BackColor = clr;
    pPanel.DrawToBitmap(pBitmap, rc);
    Image pImage = (Image)pBitmap;

    return pImage;
}

/// <summary>
/// 切换点选色带对应TIN中区域的显示/消隐状态
/// 既通过给TIN一个新的渲染器来实现
/// </summary>
private void ShowOrHideColor()
{
    ITinLayer pTinLayer = getTinLayer(_pSceneControl);
    if (pTinLayer == null)
        return;
```

```
        ITinRenderer pRenderNew = pTinLayer.GetRenderer(0);
        ITinColorRampRenderer pElevRenderer = pRenderNew as ITinColorRampRenderer;
        ISimpleFillSymbol pSymbol = null;
        int ClassCount = treeView1.Nodes.Count;
        //遍历TreeView节点，并依次设置渲染器样式
        for (int i = 0; i < ClassCount; i++)
        {
            pSymbol = pElevRenderer.get_Symbol(i) as ISimpleFillSymbol;

            IRgbColor rgb = new RgbColorClass();
            rgb.Red = ((Color)treeView1.Nodes[i].Tag).R;
            rgb.Green = ((Color)treeView1.Nodes[i].Tag).G;
            rgb.Blue = ((Color)treeView1.Nodes[i].Tag).B;
            IColor pC = rgb as IColor;
            if (!treeView1.Nodes[i].Checked)
            {
                pC.NullColor = true;
                pC.Transparency = 0;
            }
            pSymbol = new SimpleFillSymbolClass();
            pSymbol.Color = pC;

            //设置渲染器样式
            pElevRenderer.set_Symbol(i, pSymbol as ISymbol);
        }

        //创建TinEdgeRendererClass 类型的Renderer
        pTinLayer.ClearRenderers();
        //插入一个渲染模型
        pTinLayer.InsertRenderer(pRenderNew, 0);

        //刷新渲染
        _pSceneControl.Scene.SceneGraph.Invalidate(pTinLayer, true, false);
        _pSceneControl.SceneViewer.Redraw(true);
        _pSceneControl.Scene.SceneGraph.RefreshViewers();
    }

    /// <summary>
    /// 改变点选色带对应TIN区域的颜色
    /// </summary>
    private void ChangeColor()
    {
        FrmChosePColor fChoseColor = new FrmChosePColor();
        if (fChoseColor.ShowDialog() == DialogResult.OK)
```

```
{
    Color color = fChoseColor.pureColorComboBox1.SelectedColor;
    IRgbColor rgb = new RgbColorClass();
    rgb.Red = color.R;
    rgb.Green = color.G;
    rgb.Blue = color.B;
    IColor pC = rgb as IColor;

    Image pImage = getImage(color);
    pImageList.Images.Add(pImage);
    pTreeNode.ImageIndex = pImageList.Images.Count - 1;
    pTreeNode.Tag = color;

    treeView1.Refresh();

    ITinLayer pTinLayer = getTinLayer(_pSceneControl);
    if (pTinLayer == null)
        return;

    ITinRenderer pRenderNew = pTinLayer.GetRenderer(0);
    ITinColorRampRenderer pElevRenderer = pRenderNew as ITinColorRampRenderer;
    ISimpleFillSymbol pSymbol = null;
    int ClassCount = treeView1.Nodes.Count;
    for (int i = 0; i < ClassCount; i++)
    {
        if (i != pTreeNode.Index)
            continue;
        pSymbol = new SimpleFillSymbolClass();

        if (!treeView1.Nodes[i].Checked)
        {
            pC.NullColor = true;
            pC.Transparency = 0;
        }

        pSymbol.Color = pC;
        pElevRenderer.set_Symbol(i, pSymbol as ISymbol);
    }

    //创建TinEdgeRendererClass 类型的Renderer
    pTinLayer.ClearRenderers();
    pTinLayer.InsertRenderer(pRenderNew, 0);//插入一个渲染模型

    //渲染的刷新方法.
```

```
                _pSceneControl.Scene.SceneGraph.Invalidate(pTinLayer, true, false);
                _pSceneControl.SceneViewer.Redraw(true);
                _pSceneControl.Scene.SceneGraph.RefreshViewers();
            }
        }

        /// <summary>
        /// 获得三维控件中的TIN图层
        /// </summary>
        /// <param name="pSceneControl">含TIN图层的AxSceneControl控件</param>
        /// <returns></returns>
        private static ITinLayer getTinLayer(AxSceneControl pSceneControl)
        {
            IScene pScene = pSceneControl.Scene;
            ITinLayer pTinLayer = null;
            for (int i = 0; i < pScene.LayerCount; i++)
            {
                ILayer lyr = pScene.get_Layer(i);
                if (lyr is ITinLayer)
                {
                    pTinLayer = lyr as ITinLayer;
                    break;
                }
            }
            return pTinLayer;
        }
    }
}
```

最后，只需要将该控件拖入相应的窗体中，调用控件的 IniColorTree()方法，传入参数 AxSceneControl 即可，如下：

```
        private void button1_Click(object sender, EventArgs e)
        {
            //关联色板
            this.colorControlView1.IniColorTree(this.axSceneControl1);
        }
```

4.下一篇简介

下一篇打算介绍如何从 DEM 栅格数据中提取等值线以及如何由 DEM 栅格数据生成 TIN 三维模型。最后把这个系列所设计的功能整合到一个程序中，以供感兴趣的朋友参考。