第 3 章

几何对象和空间坐标系

3.1 Geometry 对象

Geometry 是一个要素的基本组成部分，它确定了要素在地球上的位置。用户可以通过空间过滤器（ SpatialFilter）对这些要素进行空间查询操作，如查询与某个几何对象包含其他的几何形体等，以及返回需要的几何对象。这个工作对于 GIS 用户而言使用非常普遍。GIS 也可以对几何形体对象进行空间运算，如缓冲区分、相交、合并等，有些空间运算还会涉及拓扑关系等方面的内容，这些操作一般都在两个 Geometry 对象之间进行。

Geometry 是 Arc Engine 中使用最为广泛的对象集之一，用户在新建、删除、编辑或进行地理分析的时候，就是在处理一个包含几何形体的矢量对象。在 Geometry 模型中，几何形体对象分为两个层次，一个是构成要素形状的几何图形，另一个是组成这些形状的构件。前者包括 Point、Multipoint、Envelope、Polyline、Polygon 等。Geometry 的主要对象模型如图 3-1 所示。本章着重围绕这些主要对象进行介绍。

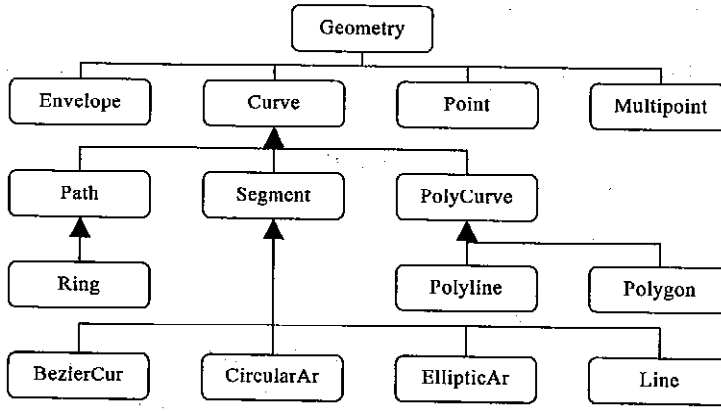


图 3-1　Geometry 主要对象模型

Point 对象是一个 0 维的几何图形，具有 x、y 坐标值，以及可选的属性，如高程（Z 值）、测量值（M）和 ID 号等，可用于描述需要精确定位的对象。

Multipoint 点集对象是无序点的群集，用于表示具有相同属性设置的同一组点。如一家公司不同的营业场所就可以使用点集来表示。

Envelope 包络线是一个矩形，用于表示要素的空间范围。它覆盖了几何对象的最小坐标和最大坐标，同时也记录了几何形体对象的 Z 值和 M 值的变化范围。所有的几何形体对象都拥有一个包络线，连其自己本身也有。

Polyline 多义线是一个有序路径（Path）的集合，这些路径既可以是连续的，也可以是离散的。这些对象可用于表示具有线状特征的对象，用户可以用单路径构成的多义线来表示简单线，也可以用具有多个路径的多义线来表示复杂线类型。

Polygon 多边形是环（Ring）的集合，环是一种封闭的路径。Polygon 可以由一个或者多个环组成，甚至环内嵌套环，但是内、外环之间不能重叠，它通常用来描述面状特性的要素。

Geometry 类是所有几何形体对象的父类，是一个抽象类，IGeometry 接口定义的属性和方法为所有的几何对象所拥有。如 Dimension 用于查询几何形体对象的纬度；Envelope 用于返回几何对象的包络线；GeometryType 用于返回对象的几何类型；IsEmpty 用于查看一个对象是否为空；SetEmpty 方法用于将一个几何对象设置为空；Project 方法用于设置一个几何对象的空间参考属性，用户可以产生或引用系统定义的空间参考。

Geometry 类型的集合接口主要有 IGeometryCollection、ISegmentCollection、IpointCollection 等。

IGeometryCollection 接口被多种几何对象继承实现，如 Polygons、Polylines、Multipoints、GeometryBags 等。该接口提供了方法，可以添加、改变和移除一个几何对象的组成元素，即子对象，如 Polygons 对象的组成元素 Ring 可以使用该接口的方法添加、改变和移除。几何对象都是有序的子对象的集合，每个子对象都有索引值，这个索引值确定了它们在组成方向上的排列顺序，可以通过 IGeometryCollection 的 Geometry 属性和索引值获取几何对象的子对象，GeometryCount 则用于返回这些子对象的数目。

IGeometryCollection 提供了一些方法可进行集合操作，如 AddGeometries、AddGeometry、AddGeometryCollection、InsertGeometries、InsertGeometriesCollection、SetGeometries 等。下面的代码演示了如何使用这些方法。

新建一个 VS2005 工程，命名为“lesson1”，在 Form1 窗体上添加一个 ToolbarControl 控件，一个 MapControl 控件，一个 LicenseControl1 控件，4 个 Button 控件，然后在 ToolbarControl 控件的属性里设置“伙伴控件”为 MapControl，添加全图、漫游、放大、缩小等工具，如图 3-2 所示。

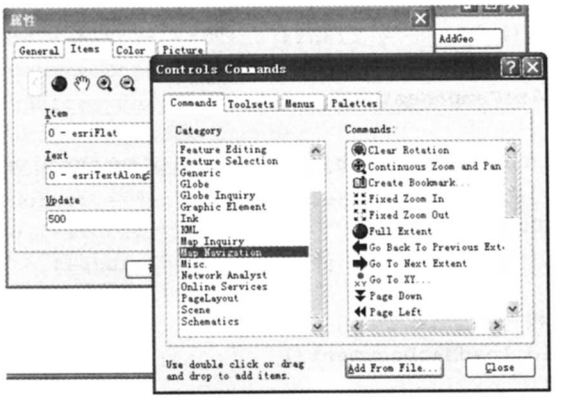


图 3-2　在工程 lesson1 中添加控件

修改 4 个 Button 控件的 Text 属性，窗体整体效果如图 3-3 所示。

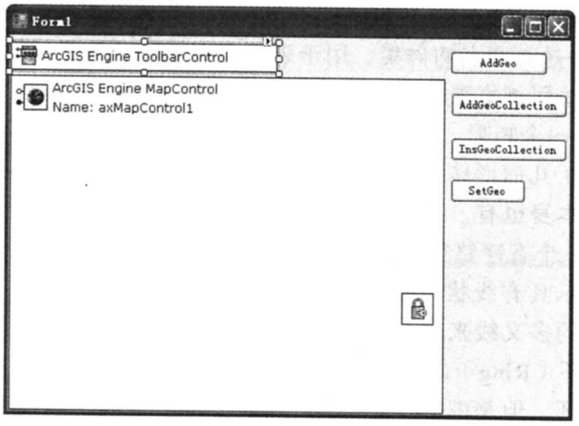


图 3-3　窗体整体效果

在代码窗口中添加如下代码。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;


namespace lesson1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            loadMapDocument();
        }

        //加载地图文档
        private void loadMapDocument()
        {
            System.Windows.Forms.OpenFileDialog openFileDialog;
```

```
            openFileDialog = new OpenFileDialog();
            openFileDialog.Title = "打开地图文档";
            openFileDialog.Filter = "map documents(*.mxd)|*.mxd";
            openFileDialog.ShowDialog();
            string filePath = openFileDialog.FileName;
            if (axMapControl1.CheckMxFile(filePath))
            {
                axMapControl1.MousePointer = esriControlsMousePointer. esriPoin-
terHour glass;
                axMapControl1.LoadMxFile(filePath, 0, Type.Missing);
                axMapControl1.MousePointer = esriControlsMousePointer.esriPointer
Default;
            }
            else
            {
                MessageBox.Show(filePath + "不是有效的地图文档");
            }
        }
        //添加实体对象到地图图层
        private void addFeature(string layerName,IGeometry geometry)
        {
            int i = 0;
            ILayer layer = null;
            for (i = 0; i < axMapControl1.LayerCount; i++)
            {
                layer = axMapControl1.Map.get_Layer(i);
                if (layer.Name.ToLower() == layerName)
                {
                    break;
                }
            }
            IFeatureLayer featureLayer = layer as IFeatureLayer;
            IFeatureClass featureClass = featureLayer.FeatureClass;
            IDataset dataset = (IDataset)featureClass;
            IWorkspace workspace = dataset.Workspace;
            //开始空间编辑
            IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
            workspaceEdit.StartEditing(true);
            workspaceEdit.StartEditOperation();
            IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
            IFeatureCursor featureCursor;
            //清除图层原有的实体对象
            featureCursor = featureClass.Search(null, true);
            IFeature feature;
            feature = featureCursor.NextFeature();
            while (feature!=null)
            {
                feature.Delete();
                feature = featureCursor.NextFeature();
            }
            //开始插入新的实体对象
```

```
            featureCursor = featureClass.Insert(true);
            featureBuffer.Shape = geometry;
            object featureOID = featureCursor.InsertFeature(featureBuffer);
            //保存实体
            featureCursor.Flush();
            //结束空间编辑
            workspaceEdit.StopEditOperation();
            workspaceEdit.StopEditing(true);
            System.Runtime.InteropServices.Marshal.ReleaseComObject (featureCursor);
        }
        //添加实体对象
        private void button1_Click(object sender, EventArgs e)
        {
            IGeometryCollection geometryCollection = new MultipointClass  ();
            IMultipoint multipoint;
            object missing = Type.Missing;
            IPoint point;
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i * 2, i * 2);

                geometryCollection.AddGeometry(point as IGeometry,ref missing,ref
missing);
            }
            multipoint = geometryCollection as IMultipoint ;
            addFeature("multipoint", multipoint as IGeometry);
            this.axMapControl1.Extent = multipoint.Envelope;
            this.axMapControl1.Refresh();
        }
        //添加实体对象集
        private void button2_Click(object sender, EventArgs e)
        {
            IGeometryCollection geometryCollection1 = new MultipointClass();
            IGeometryCollection geometryCollection2 = new MultipointClass();
            IMultipoint multipoint;
            object missing = Type.Missing;
            IPoint point;
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i * 2, i * 2);
                geometryCollection1.AddGeometry(point as IGeometry, ref missing,
ref missing);
            }
            geometryCollection2.AddGeometryCollection(geometryCollection1);
            multipoint = geometryCollection2 as IMultipoint;
            addFeature("multipoint", multipoint as IGeometry);
            this.axMapControl1.Extent = multipoint.Envelope;
            this.axMapControl1.Refresh();
        }
```

```
        //插入实体对象集
        private void button3_Click(object sender, EventArgs e)
        {
            IGeometryCollection geometryCollection1 = new MultipointClass();
            IGeometryCollection geometryCollection2 = new MultipointClass();
            IGeometryCollection geometryCollection3 = new MultipointClass();
            IGeometryCollection geometryCollection4 = new MultipointClass();
            IMultipoint multipoint;
            object missing = Type.Missing;
            IPoint point;
            //创建 3 个实体对象集
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i * 2, i);
                geometryCollection1.AddGeometry(point asIGeometry,ref missing, ref
missing);
            }
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i , i );
                geometryCollection2.AddGeometry(point as IGeometry, ref missing,
ref missing);
            }
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i, i * 2);
                geometryCollection3.AddGeometry(point as IGeometry, ref missing,
ref missing);
            }
            geometryCollection1.InsertGeometryCollection (1,geometryCollection2);
            geometryCollection1.InsertGeometryCollection (1,geometryCollection3);

            multipoint = geometryCollection1 as IMultipoint;
            addFeature("multipoint", multipoint as IGeometry);
            this.axMapControl1.Extent = multipoint.Envelope;
            this.axMapControl1.Refresh();
        }
        //用实体数组取代实体集中的实体对象
        private void button4_Click(object sender, EventArgs e)
        {
            IGeometryCollection geometryCollection1 = new MultipointClass();
            IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
            IGeometry[] geometryArray = new IGeometry[10];
            IMultipoint multipoint;
            object missing = Type.Missing;
            IPoint point;
            for (int i = 0; i < 10; i++)
            {
```

```
                point = new PointClass();
                point.PutCoords(i * 2, i * 2);
                geometryArray[i] = point as IGeometry;
            }
            geometryBridge.SetGeometries(geometryCollection1, ref geometryArray);
            multipoint = geometryCollection1 as IMultipoint;
            addFeature("multipoint", multipoint as IGeometry);
            this.axMapControl1.Extent = multipoint.Envelope;
            this.axMapControl1.Refresh();
        }
    }
}
```

ISegmentCollection 接口是 Segment 集合对象，由 Path、Polygon、Polyline 和 Ring 等 4 个类实现。使用这个接口可以处理 Segment 集合对象中的每个组成元素，该接口与 IGeometryCollection 类似，可以添加、改变和移除一个 Segment，每个 Segment 都有一个索引值。该接口也提供了一些方法，如 EnumSegments 用于返回一个 SegmentCollection 对象中的 Segment 对象，以枚举值返回；AddSegments 用于添加 Segment 数组对象；AddSegmentCollection 方法则可用于添加另一个 SegmentCollection 对象中的 Segment，即合并两个 SegmentCollection 对象。此外还提供了 QuerySegments、SetSegment、AddSegments 等方法。下面的代码将演示如何使用这些方法。

在 lesson1 工程 Form1 窗体上增加 3 个 Button 控件，修改 Button 控件的 Text 属性，界面效果如图 3-4 所示。

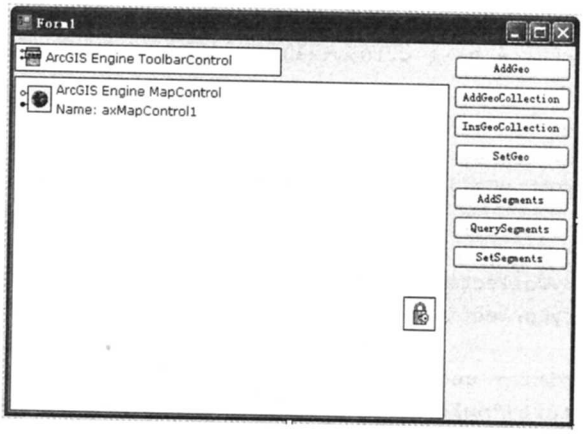


图 3-4　在 lesson1 工程窗体上增加控件

在代码窗口中添加如下代码。

```
//添加 SEGMENTS
        private void button5_Click(object sender, EventArgs e)
        {
            ISegment[] segmentArray = new ISegment[10];
            IPolyline polyline = new PolylineClass();
            for (int i = 0; i < 10; i++)
            {
                ILine line = new LineClass();
                IPoint fromPoint = new PointClass();
                fromPoint.PutCoords(i * 10, i * 10);
```

```
                IPoint toPoint = new PointClass();
                toPoint.PutCoords(i * 15, i * 15);
                line.PutCoords(fromPoint, toPoint);
                segmentArray[i] = line as ISegment;
            }
            ISegmentCollection segmentCollection = new PolylineClass();
            IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
            geometryBridge.AddSegments(segmentCollection, ref segmentArray);
            polyline = segmentCollection as IPolyline;
            addFeature("polyline", polyline as IGeometry);
            this.axMapControl1.Extent = polyline.Envelope;
            this.axMapControl1.Refresh();
        }
        //查询 segment
        private void button6_Click(object sender, EventArgs e)
        {
            ISegment[] segmentArray = new ISegment[10];
            for (int i = 0; i < 10; i++)
            {
                ILine line = new LineClass();
                IPoint fromPoint = new PointClass();
                fromPoint.PutCoords(i * 10, i * 10);
                IPoint toPoint = new PointClass();
                toPoint.PutCoords(i * 15, i * 15);
                line.PutCoords(fromPoint, toPoint);
                segmentArray[i] = line as ISegment;
            }
            ISegmentCollection segmentCollection = new PolylineClass();
            IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
            geometryBridge.AddSegments(segmentCollection, ref segmentArray);

            int index = 0;
            ISegment[] outputSegmentArray = new ISegment[segmentCollection.Segment
Count - index];
            for (int i = 0; i < outputSegmentArray.Length; i++)
            {
                outputSegmentArray[i] = new LineClass();
            }
            //查询 Segment
            geometryBridge.QuerySegments(segmentCollection, index, ref outputSegment
Array);
            String report = "";
            for (int i = 0; i < outputSegmentArray.Length; i++)
            {
                ISegment currentSegment = outputSegmentArray[i];
                ILine currentLine = currentSegment as ILine;
                report = report + "index = " + i + " , FromPoint X = " +
                currentLine.FromPoint.X + " , FromPoint Y = " + currentLine.
                FromPoint.X;
                report = report + " , ToPoint X = " + currentLine.ToPoint.X + " ,
                ToPoint Y = " + currentLine.ToPoint.X + "\n";
```

```
        }
        System.Windows.Forms.MessageBox.Show(report);

    }
    //用 Segment 数组取代 Segment 集中的 Segment 对象
    private void button7_Click(object sender, EventArgs e)
    {
        IPolyline polyline = new PolylineClass();
        ISegmentCollection segmentCollection = new PolylineClass();
        IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
        ISegment[] insertSegmentArray = new ISegment[5];
        for (int i = 0; i < 5; i++)
        {
            ILine insertLine = new LineClass();
            IPoint insertFromPoint = new PointClass();
            insertFromPoint.PutCoords(i, 1 );
            IPoint insertToPoint = new PointClass();
            insertToPoint.PutCoords(i*10,1 );
            insertLine.PutCoords(insertFromPoint, insertToPoint);
            insertSegmentArray[i] = insertLine as ISegment;
        }
        geometryBridge.SetSegments(segmentCollection, ref insertSegmentArray);
        polyline = segmentCollection as IPolyline;
        addFeature("polyline", polyline as IGeometry);
        this.axMapControl1.Extent = polyline.Envelope;
        this.axMapControl1.Refresh();
    }
```

IPointCollection 接口是点集合对象。任何一个几何对象都是由点构成，因此任何一个几何对象都有一个点集。该接口可被多种几何对象类实现，如 Multipoint、Path、Ring、Polyline 和 Polygon 等。这些几何类型都是 PointCollection 对象，可以通过该接口定义的方法获取、添加、插入、查询和移除集合中的某个顶点。和 IGeometryCollection、ISegmentCollection 一样，该接口也提供了类似的方法进行集合操作。下面的代码将演示如何使用 AddPointCollection、QueryPoints、UpdatePoint 等 3 个方法。

在 lesson1 工程 Form1 窗体上增加 3 个 Button 控件，修改 Button 控件的 Text 属性，界面效果如图 3-5 所示。

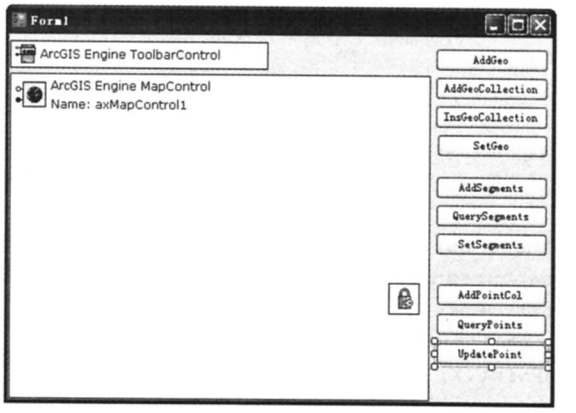


图 3-5　界面效果

在代码窗口中添加如下代码。

```
//添加点集
        private void button8_Click(object sender, EventArgs e)
        {
            IPointCollection4 pointCollection =new MultipointClass ();
            IPointCollection pointCollection2 = new MultipointClass();
            IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
            IPoint [] points = new PointClass[10];
            IMultipoint multipoint;
            object missing = Type.Missing;
            IPoint point;
            for (int i = 0; i < 10; i++)
            {
                point = new PointClass();
                point.PutCoords(i * 5, i);
                points[i] = point;
            }
            geometryBridge.SetPoints(pointCollection, ref points);
            pointCollection2.AddPointCollection(pointCollection);
            multipoint = pointCollection2 as IMultipoint;
            addFeature("multipoint", multipoint as IGeometry);
            this.axMapControl1.Extent = multipoint.Envelope;
            this.axMapControl1.Refresh();
        }
        //查询点对象
        private void button9_Click(object sender, EventArgs e)
        {
            IPoint point1 = new PointClass();
            point1.PutCoords(10, 10);
            IPoint point2 = new PointClass();
            point2.PutCoords(20, 20);
            IPoint[] inputPointArray = new IPoint[2];
            inputPointArray[0] = point1;
            inputPointArray[1] = point2;
            IPointCollection4 pointCollection = new MultipointClass();
            IGeometryBridge geometryBridge = new GeometryEnvironmentClass();
            geometryBridge.AddPoints(pointCollection, ref inputPointArray);

            int index = 0;
            IPoint[] outputPointArray = new IPoint[2];
            for (int i = 0; i < outputPointArray.Length; i++)
            {
              outputPointArray[i] = new PointClass();
            }

            geometryBridge.QueryPoints(pointCollection, index, ref outputPointArray);
            for (int i = 0; i < outputPointArray.Length; i++)
```

```
                {
                    IPoint currentPoint = outputPointArray[i];
                    if (currentPoint == null)
                    {
                        System.Windows.Forms.MessageBox.Show("Current point = null");
                    }
                    else
                    {
                System.Windows.Forms.MessageBox.Show("X = " + currentPoint.X + ", Y = " +
currentPoint.Y);
                    }
                }
            }
            //更新指定点对象
            private void button10_Click(object sender, EventArgs e)
            {
                IMultipoint multipoint;
                object missing = Type.Missing;
                IPoint point1 = new PointClass();
                point1.PutCoords(10, 10);
                IPoint point2 = new PointClass();
                point2.PutCoords(20, 20);
                IPointCollection pointCollection = new MultipointClass();
                pointCollection.AddPoint(point1, ref missing, ref missing);
                pointCollection.AddPoint(point2, ref missing, ref missing);
                point1 = new PointClass();
                point1.PutCoords(40, 10);
                pointCollection.UpdatePoint(1, point1);
                multipoint = pointCollection as IMultipoint;
                addFeature("multipoint", multipoint as IGeometry);
                this.axMapControl1.Extent = multipoint.Envelope;
                this.axMapControl1.Refresh();
            }
```

3.2 Envelope 对象

Envelope 也称包络线，是一个矩形区域，是每个几何形体的最小外接矩形。每个 Geometry 都拥有一个 Envelope，包括 Envelope 自身。IEnvelope 是包络线对象的主要接口，定义了 XMax、XMin、YMax、YMin、Height 和 Width 等属性，用于获取或设置一个存在的包络线对象的空间坐标。IEnvelope 接口提供了一些方法，如 Expand、offset、CenterAt、PutCoords 等。Expand 方法用于按比例缩放包络线的范围，产生一个新的包络对象；offset 是一个偏移方法，通过一个给定的（X，Y）移动包络线；CenterAt 方法则通过改变包络线的中心点来移动包络线；PutCoords 方法是通过指定的坐标点来构造包络线。

IEnvelope 接口还提供了两个拓扑运算方法 Intersect 和 Union。Intersect 用来计算两个包络线相交，返回相交部分作为结果；Union 则是合并两个包络线，并以两个包络线的最小外接矩形作为合并结果，如图 3-6 所示。

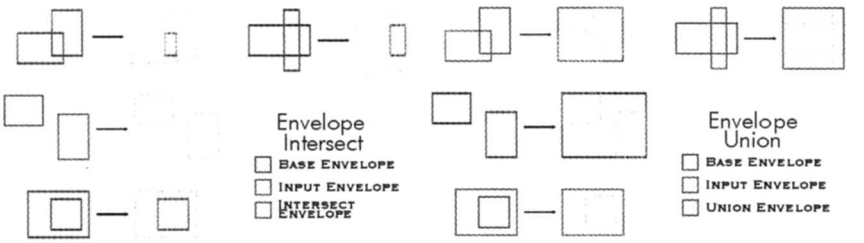


图 3-6　I Envelope 接口

下面的代码演示了如何使用 Envelope。

```
IEnvelope envelope1 = new EnvelopeClass();
IEnvelope envelope2 = new EnvelopeClass();
envelope1.PutCoords(100, 100, 200, 200);
envelope2.PutCoords(150, 150, 250, 250);
envelope1.Intersect(envelope2);
```

envelope1 为两个 Envelope 相交的交集。

```
IEnvelope envelope1 = new EnvelopeClass();
IEnvelope envelope2 = new EnvelopeClass();
envelope1.PutCoords(100, 100, 200, 200);
envelope2.PutCoords(150, 150, 250, 250);
envelope1.Union(envelope2);
```

两个 Envelope 合并为一个 Envelope。

3.3　Curve 对象

Curve 是几何体中重要的类型，除了点、点集、包络线外，可以将其他的几何形体都看做是曲线（Curve），如 Line、Polyline、Path、CircularArc、Polygon 等，这些对象都实现了 ICurve 接口。该接口提供了操作这些对象的属性和方法，但不能用于产生一个新的曲线对象。例如，Length 属性用于返回一个曲线对象的长度；FromPoint、ToPoint 属性用于获取或设置曲线的起点和终点；ReverseOrientation 方法用于改变一个曲线的节点次序，即将起点和终点对换，通过此方法可以改变曲线的方向；IsClosed 属性则用于说明曲线的起点和终点是否在同一个位置，即可以检查曲线是否闭合，如图 3-7 所示。

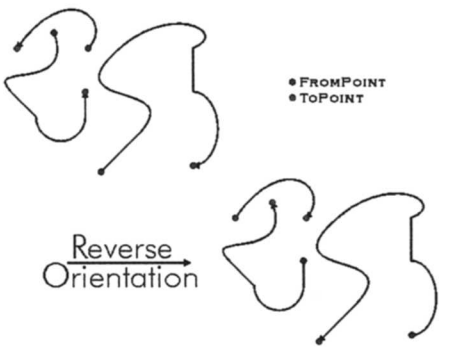


图 3-7　Curve 对象

3.3.1 Segment 对象

Segment 是由一个起点和终点，以及定义这两点之间的曲线函数组成的一纬几何形体对象。Segment 对象是一个抽象类，可以泛化为 Line、CircularArc、EllipticArc、BezierCurve 等 4 种类型，这 4 种类型将在本节的后一部分分别介绍。这些 Segment 对象可以独立存在，也可以用于构造其他的几何形体对象，如 Polyline、Polygon 等。由于 Segment 是从 Curve 继承而来，因此也拥有 Curve 的方法和属性，如 FromPoint、ToPoint 可用于确定曲线的起点和终点。

ISegment 是 Segment 对象的主要接口，该接口提供了两个方法：SplitAtDistance、SplitDivideLength，将一个 Segment 对象分割成了多个 Segment 对象。在分割 Segment 前会自动复制一份副本，然后对副本进行分割，分割操作对源对象不产生影响。 SplitAtDistance 方法是通过一个给定的长度或比率，在 Segment 对象的起点到终点方向上，在给定长度距离的位置上产生一个分割点。SplitDivideLength 是将一个 Segment 对象分割成多个新的 Segment，如图 3-8 所示。

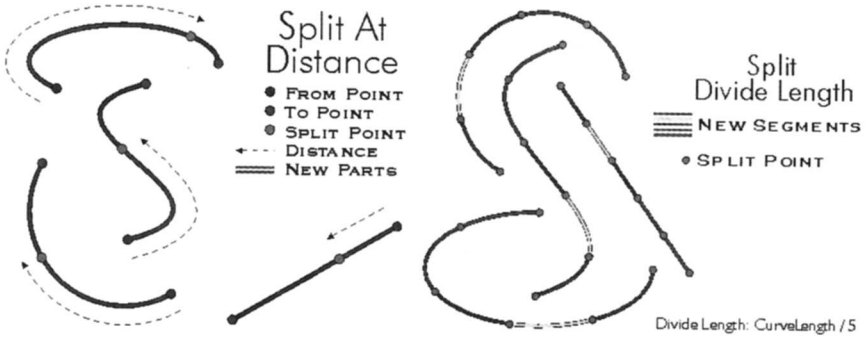


图 3-8 Segment 对象

下面的代码将演示如何使用该对象的一个常用方法和属性。

```
    IPoint fromPoint = new PointClass();
    fromPoint.PutCoords(0, 0);
    IPoint toPoint = new PointClass();
    toPoint.PutCoords(100, 0);
    ILine line = new LineClass();
    line.FromPoint = fromPoint;
    line.ToPoint = toPoint;
    double offset = 0;
    double length = 10;
    bool asRatio = false;
    int numberOfSplittedSegments;
    ISegment[] splittedSegments = new ISegment[10];
    for (int i = 0; i < 10; i++)
    {
        splittedSegments[i] = new PolylineClass() as ISegment;
    }
    ISegment segment = line as ISegment;
    IGeometryBridge2 geometryBridge = new GeometryEnvironmentClass();
    geometryBridge.SplitDivideLength(segment,offset,length,asRatio,outnumberOfSpli
ttedSegments, ref splittedSegments);
```

splittedSegments 数组保存打断后的 Segment 对象。

3.3.1.1　CircularArc 对象

CircularArc 对象是一个圆弧，是圆的一部分，如图 3-9 所示。

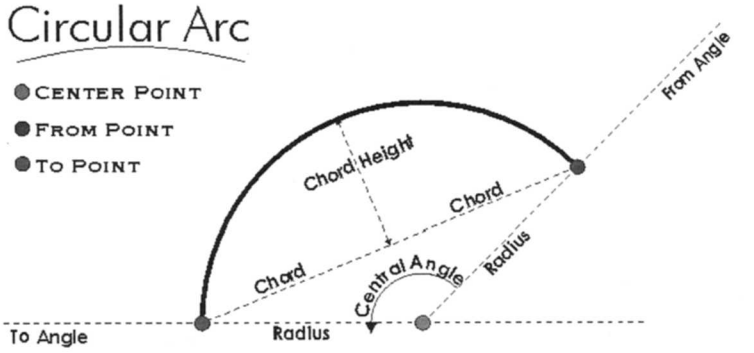


图 3-9　CircularArc 对象

CircularArc 是一个组件类。ICircularArc 是 CircularArc 的主要接口，提供了 FromAngle、ToAngle、CentralAngle、CenterPoint、ChordHeight 等属性，通过这些属性可以改变这个圆弧的形状。同时也提供了类型检查属性，如 IsLine、IsPoint、IsMinor、IsCounterClockwise 等。如果 IsLine 为真，则圆弧的半径无限大，即为直线；如果 IsPoint 为真，则圆弧的半径为 0，即为点；如果 IsMinor 为真，则 CentralAngle 小于半圆，即为劣弧；如果 IsCounterClockwise 为真，则 CentralAngle 为正值。通过接口提供的 Complement 方法，可以封闭该圆弧对象，产生一个圆对象。

CircularArc 对象可以使用 IConstructCircularArc 接口提供的方法进行构造，该接口提供了 34 种构造方法，如 ConstructCircle 构造器，该构造器通过指定一个 CenterPoint（圆心）和一个 Radius（半径）来构造；ConstructArcDistance 构造器，该构造器通过指定一个圆心、起始点、圆弧方向和圆弧来构造；ConstructChordDistance 构造器，该构造器通过指定一个起点、圆弧的弦长度、圆弧的方向和中心点来构造；ConstructFilletPoint 构造器，该构造器基于两个 Segment 对象以及内切弧，在两个 Segment 上的点可以产生两条线段和圆弧的内切线。在“帮助”中提供了这些方法的详细说明，读者可以查看“帮助”中的具体条目。

3.3.1.2　Line 对象

Line 是最简单的 Segment 对象，由起点和终点构成一条直线段，用于构造 Polyline、Polygon、Ring 和 Path 等对象。ILine 是 Line 对象的主要接口，定义了一系列用于构造和设置线段对象的属性和方法。如 QueryCoords 属性用于返回 Line 的起点和终点，PutCoords 属性用于设置 Line 的起点和终点坐标，Angle 属性用于返回 Line 与 x 轴的夹角。

除了 ILine 外，Line 还继承自 IConstructLine，该接口提供了两个方法：ConstructAngleBisector 和 ConstructExtended。ConstructAngleBisector 方法通过指定的 3 个点对象和新构造线的长度来构造，原理是：通过 3 个点构造一个夹角，在夹角的角平分线上根据线的长度构造一条新的线，如图 3-10 所示。

ConstructExtended 通过扩展一个已经存在的线段对象来产生一个新的线段。扩展类型将在后面进行详细的介绍。

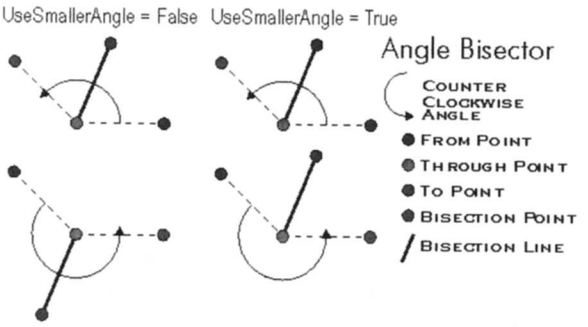


图 3-10　Line 对象

3.3.1.3　EllipticArc 对象

EllipticArc(椭圆弧)是椭圆的一部分。EllipticArc 对象与 CircularArc 对象类似。在 EllipticArc 方法中需要用到一个参数 ellipseStd，该参数用来改变坐标系和角度，它为 false 时，系统是标准的笛卡儿坐标系；它为 True 时，所有的角度都是相对坐标，0 度方向不在是笛卡儿直角坐标系，而是和椭圆的长轴一致，起点和终点坐标与中心点是相对的，如图 3-11 所示。

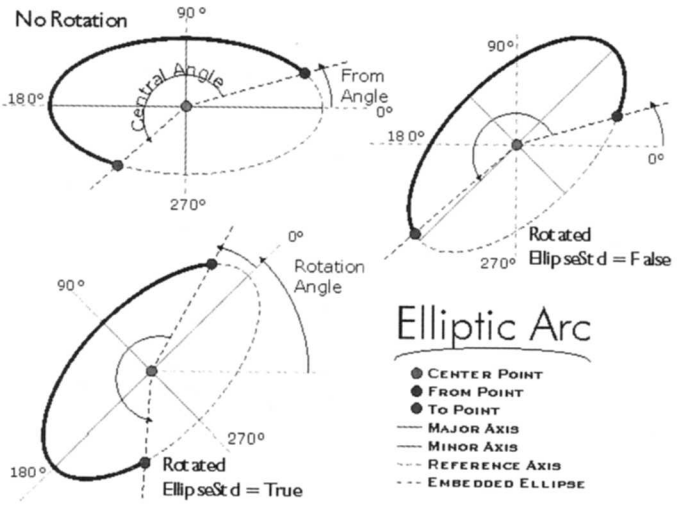


图 3-11　EllipticArc 对象

IEllipticArc 接口是 EllipticArc 的主要接口，提供了一些方法用于操作椭圆弧。如 GetAxes 方法用于得到椭圆的长半轴和短半轴以及比例，PutAxes 方法用于改变一个已经存在的椭圆弧对象的半轴长度。另外接口还提供了几个属性用于检查椭圆弧的情况，如 IsCircular 用于检查长半轴和短半轴的长度是否相等，即检查是否是圆；IsLine 如果为 True，则短半轴为 0，为一直线；IsPoint 如果为 True，则长半轴和短半轴都是 0，为一个点对象；ISCounterClockwise 如果为 True，则 CentralAngle 为正值；IsMinor 如果为 True，则弧的长度小于椭圆的一半，为劣弧。IEllipticArc 接口还提供了 Complement、PutCoord、QueryCoords、QueryCoordsByAngle 等方法，这些方法的使用同 ICircularArc 类似。

EllipticArc 可以用 IConstructEllipticArc 构造器来构造，该构造器提供了 ConstructEnvelope、ConstructquarterEllipse、ConstructTwoPointsEnvelope、ConstructUpToFivePoints 等 4 种方法。

ConstructEnvelope 方法，通过一个给定的包络线对象来产生一个内置的椭圆对象，这个椭圆对象是逆时针方向，如图 3-12 所示。

ConstructQuarterEllipse 方法，通过起点、终点和方向来构造一个椭圆弧，如图 3-13（1）所示。

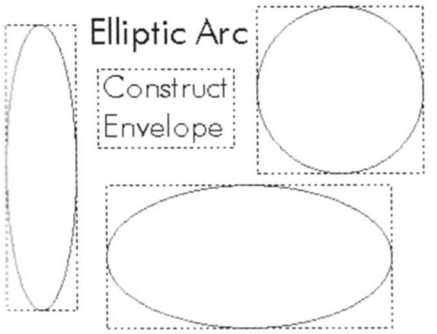


图 3-12　产生一个内置的椭圆对象

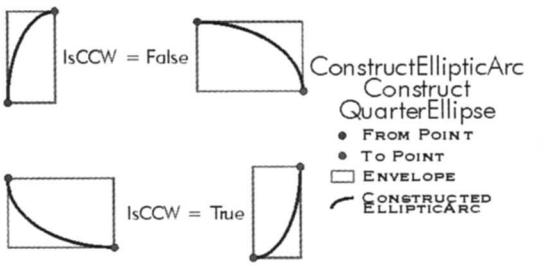


图 3-13　构造一个椭圆弧（1）

ConstructTwoPointsEnvelope 方法，通过起点、终点、包络线和方向来构造一个椭圆弧，如图 3-14（2）所示。

ConstructUpToFivePoints 方法，通过起点、终点、一个弧上的任意点以及两个椭圆对象上的点，来构造一个椭圆弧，如图 3-15（3）所示。

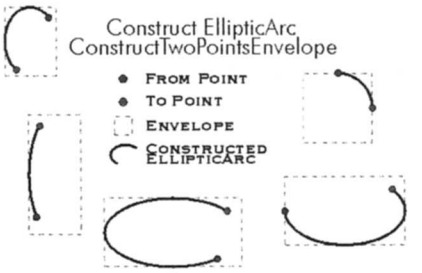


图 3-14　构造一个椭圆弧（2）

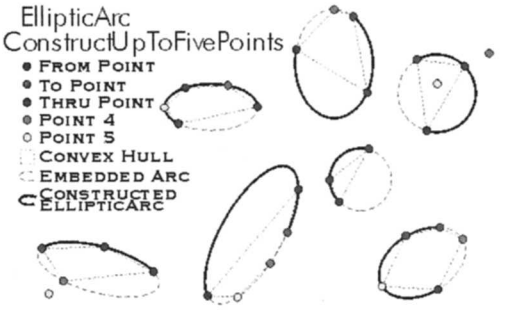


图 3-15　构造一个椭圆弧（3）

3.3.1.4　BezierCurve 对象

BezierCurve 对象定义一条两个节点之间的弧线。IBezierCurve 接口定义了设置和查询曲线属性的方法，通过 PutCoord 方法可以为曲线增加一个顶点。除了 PutCoord 方法外，Arc Engine 还提供了 IConstructBezierCurve 接口定义更高级方法，来实现 BezierCurve 曲线的构建。

3.3.2　Path 对象

Path 是连续多个 Segment 对象的集合，每个 Segment 通过首尾相连构成一条路径。IPath 是 Path 对象的主要接口，它定义了设置一个路径对象的多个方法，如 Generalizes 方法用于抽象化一个平滑的路径对象，即将一个平滑的曲线变成几条相连的线段；Smooth 方法则用于将一个非平滑的路径对象平滑；SmoothLocal 方法则只用于将某个 Segment 连接点处平滑。

Path 对象可以由两种途径来构造，其一，使用 ISegmentCollection 接口，通过接口的 AddSegment 方法为 Path 对象添加子元素；其二，使用 IConstructPath 接口，该接口提供了

ConstructRigidStretch 方法，用于旋转或缩放一个已经存在的路径对象的形状，或改变路径上顶点的位置，如图 3-16 所示。

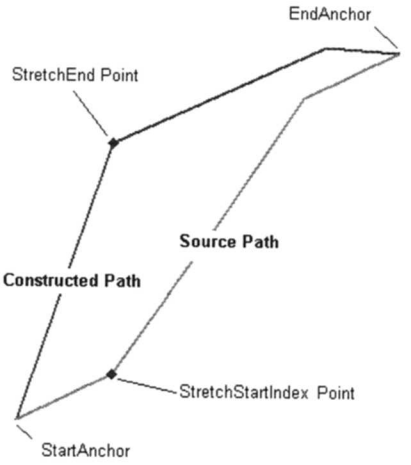


图 3-16　Path 对象

3.3.3　Ring 对象

Ring 对象是一种封闭的路径对象，它的起点和终点是同一个点。Ring 对象具有“内部”和“外部”属性，Polygon 是由 Ring 产生。由于 Ring 是封闭的 Path，因此其组成是一系列首尾相连的同方向的 Segment 对象，并且不能自相交。IRing 接口定义了 Ring 对象的处理方法，如 Close 方法用于检查 Ring 对象的起点和终点是否是同一个点，如果 Ring 不是闭合的，则会自动添加一条线段到 Ring 上，使 Ring 闭合。

在 Arc Engine 中提供了 3 种面状类型的对象：Envelope、Ring、Polygon。由于面状对象都有面积，因此这 3 种对象都实现了 IArea 接口来获取与面积相关的信息。IArea 接口提供了一些方法，如 Area 方法用于返回一个具有封闭特性几何形体对象的面积，Centroid 方法用于返回这些几何形体的重心，LabelPoint 方法用于返回几何形体的标注点等。

3.3.4　PolyCurve 对象

PolyCurve 是一个抽象类，泛化为 Polyline 和 Polygon。IPolyCurve 接口提供了处理这两种对象的方法，如 SplitAtDistance、SplitAtPoint，都是通过指定距离或指定点在 PolyCurve 对象上打断；Generalize 方法则用于对 PolyCurve 对象进行概化整形。关于 Polyline 和 Polygon 类，将在后面两节介绍。

3.4　点对象

点（Point）代表了一个 0 维的具有 X、Y 坐标的几何对象。点是没有任何形状的，可用于描述点类型的要素，而且 Geometry 中的任何类型都是用点来产生的。

构成几何形状的顶点存在着 3 种可以选择的属性，即 Z、M 和 ID。Z 值在大多数情况下都

可用于表示一个点的 Z 坐标，还可以将 Z 值作为一个点的辅助值来使用；M 即度量值，可以一个路径对象的线性度量，用于交通工程中一条公路的不同点的位置；ID 值即为一个点的唯一标识值。

点集（Multipoint）是具有相同属性的点的集合，用于构成高级几何对象、几何对象动态模拟等。

IPoint 接口定义了 Point 对象的属性和方法，点（Point）可以使用 PutCoords 方法创建，也可以使用 IConstructPoint 接口来创建。IConstructPoint 接口提供了 10 种创建点的方法，下面分别介绍。

（1）ConstructAlong 沿线创建法：基于一个曲线上的起始点对象，通过给定距离、比例和扩展类型，沿着曲线创建一个新点。如果距离比曲线的长度长，那么点将沿着它的切线生成。如图 3-17 所示。

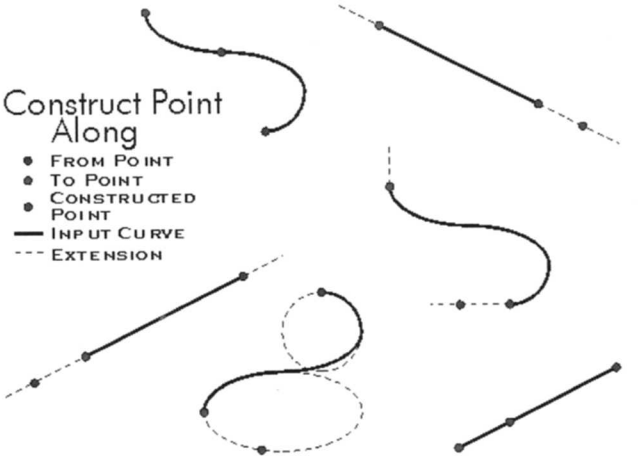


图 3-17　点对象

其扩展参数如下。

EsriNoExtension：没有扩展。在这种方式下构建点将始终在输入曲线上。如果输入距离小于 0，则构建点在曲线的起点；如果输入距离大于曲线的长度，则构建点在曲线的终点。

EsriExtendTangentAtFrom：起点切线。在这种方式下，如果输入距离小于 0，则构建点在曲线起点的切线且与曲线反方向的距离位置上。

EsriExtendTangentAtTo：终点切线。在这种方式下，如果输入距离大于曲线长度，则构建点放在曲线终点的切线且与曲线同方向的距离位置上。

EsriExtendEmbeddedAtFrom：与起点切线类似，但使用嵌入几何不是切线。

EsriExtendEmbeddedAtTo：与终点切线类似，但使用嵌入几何不是切线。

EsriExtendEmbedded：在这种方式下，根据输入距离自动使用 esriExtendEmbeddedAtFrom 或 esriExtendEmbeddedAtTo 创建构建点。

EsriExtendTangents：在这种方式下，根据输入距离自动使用 esriExtendTangentAtFrom 或 esriExtendTangentAtTo 创建构建点。

（2）ConstructAngleBisector 角平分线构建法：这种方法使用 3 个点，即起点（FromPoint）、经由点（Through Point）和终止点（ToPoint）。该方法通过平分 3 点形成的夹角，并设置一个距离在平分线上寻找一个点，如图 3-18 所示。

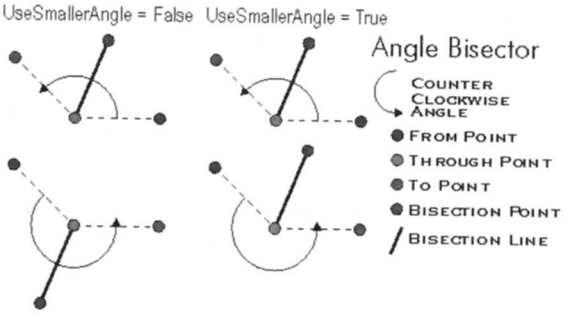


图 3-18　角平分线构建

（3）ConstructAngleDistance 构造角度距离法：通过一个给定点和一个相对点的绝对角度和距离构建一个点，如图 3-19 所示。

（4）ConstructAngleIntersection 构造角度交点法：通过给定的两个点和两个角度产生两条射线，然后在两条射线的交点处产生一个交点，如图 3-20 所示。

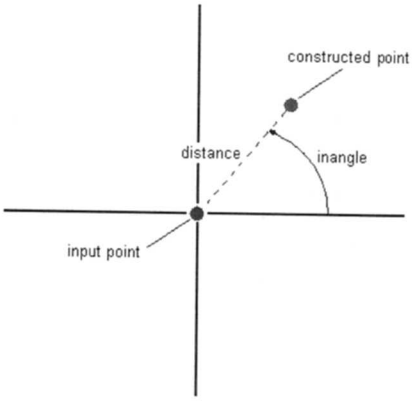


图 3-19　构建一个点

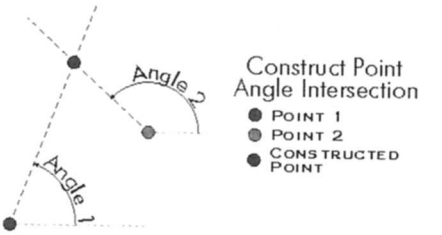


图 3-20　产生一个交点

（5）ConstructDeflection 构造偏转角度法：通过给定一条基准线、一个偏转角度和一个距离，沿着线的偏转角度的射线方向以给定距离位置构建一个点。如图 3-21 所示。

（6）ConstructDeflectionIntersection 构造偏转角交点法：通过指定一条线段作为基线，从该线段的起点和终点的偏转角度引出两个射线，然后在两条射线的交点处构建一个点。该方法的 bRightSide 指定偏转角在线段的左边或右边，如图 3-22 所示。

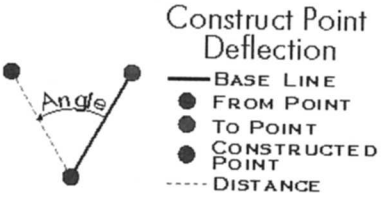


图 3-21　构建一个点

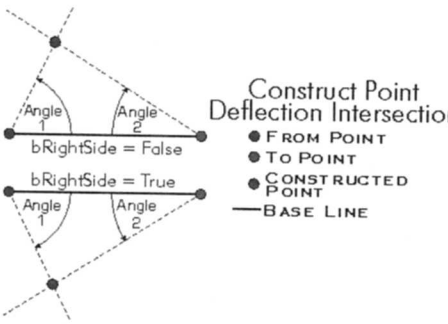


图 3-22　构造偏转角交点

（7）ConstructOffset 构造偏移点法：该方法通过指定一条曲线，沿着曲线的距离或比例进行水平偏移。可以设置参数 Offset 的值，如果值为正，则该点向右水平偏移；如果值为负，则向左偏移。如图 3-23 所示。

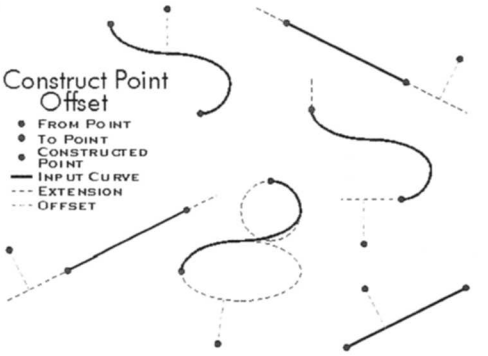


图 3-23　构造偏移点

其扩展参数如下。

EsriNoExtension：没有扩展。在这种方式下构建点将始终在输入曲线上。如果输入距离小于 0，则构建点在曲线的起点；如果输入距离大于曲线的长度，则构建点在曲线的终点。

EsriExtendTangentAtFrom：起点切线。在这种方式下，如果输入距离小于 0，则构建点在曲线起点的切线且与曲线反方向的距离位置上。

EsriExtendTangentAtTo：终点切线。在这种方式下，如果输入距离大于曲线长度，则构建点放在曲线终点的切线且与曲线同方向的距离位置上。

EsriExtendEmbeddedAtFrom：与起点切线类似，但使用嵌入几何不是切线。

EsriExtendEmbeddedAtTo：与终点切线类似，但使用嵌入几何不是切线。

EsriExtendEmbedded：在这种方式下，根据输入距离自动使用 esriExtendEmbeddedAtFrom 或 esriExtendEmbeddedAtTo 创建构建点。

EsriExtendTangents：在这种方式下，根据输入距离自动使用 esriExtendTangentAtFrom 或 esriExtendTangentAtTo 创建构建点。

EsriExtendAtFrom：在这种方式下，包括了 esriExtendTangentAtFrom 和 esriExtendEmbeddedAtFrom 。

EsriExtendAtT：在这种方式下，包括了 esriExtendTangentAtTo 和 esriExtendEmbeddedAtTo。

EsriExtendEmbeddedAtTo，在这种方式下，包括了 esriExtendTang entAtTo 和 EsriExtendEmbeddedAtTo，如图 3-24 所示。

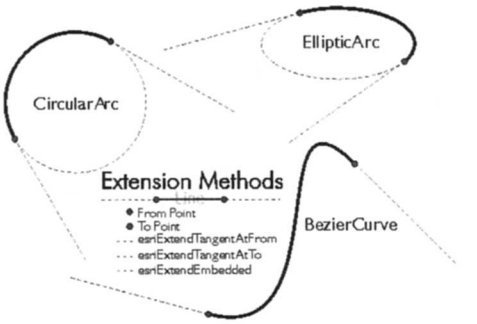


图 3-24　构造偏移点参数 esriExtendAtTo 的示例

（8）ConstructParallel 构造平行线上点法：该方法是给定一条路径 Path、一个参考点 Point 及一个距离或比率，在平行线上构建一个点，如图 3-25 所示。

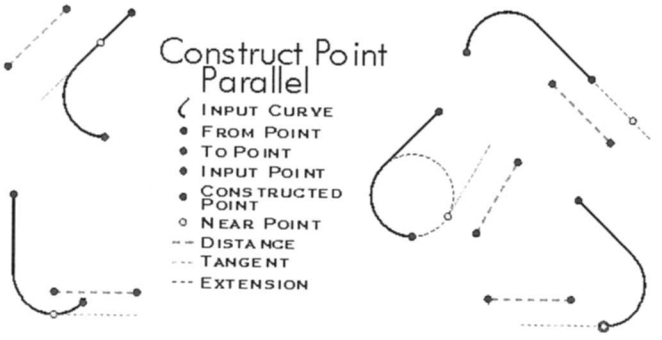


图 3-25　构造平行线上点法

其扩展参数同 ConstructOffset。

（9）ConstructPerpendicular 构造垂直线上点法：通过给定一条路径 Path、一个参考点 Point 及一个距离，在参考点与路径的垂直方向上，在给定距离位置构建一个点，如图 3-26 所示。

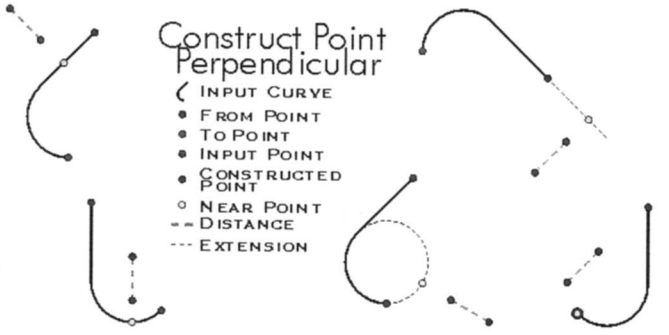


图 3-26　构造垂直线上点法

（10）ConstructThreePointResection 后方交会定点法：该方法即测量学中的后方交会定点法，即构建点对 3 个已知点进行观测，得到两个角度，然后确定构建点的位置，如图 3-27 所示。

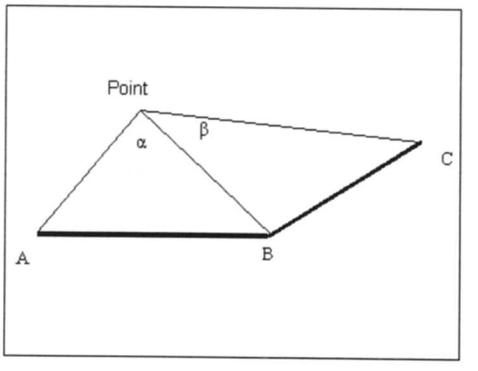


图 3-27　后方交会定点法

下面的工程将演示如何使用这些方法。

新建一个 VS2005 工程，工程命名为“lesson2”，在 Form1 窗体上添加一个 ToolbarControl 控件，一个 MapControl 控件，一个 LicenseControl1 控件，10 个 Button 控件，然后在 ToolbarControl

控件的属性里设置“伙伴控件”为 MapControl，界面效果如图 3-28 所示。

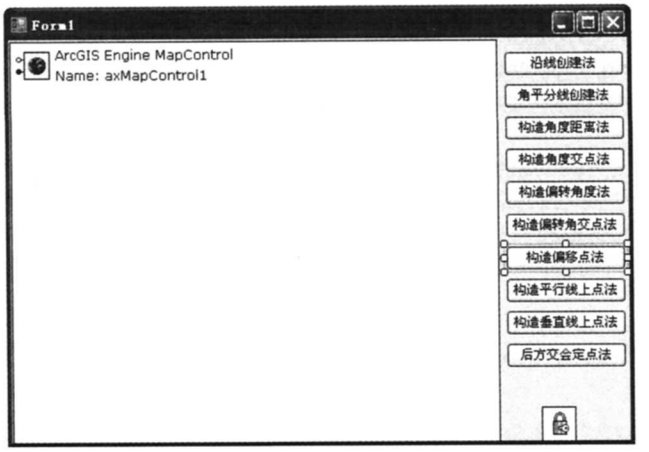


图 3-28　界面效果

在代码窗口中添加如下代码。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
namespace lesson2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            loadMapDocument();
        }
        //加载地图文档
        private void loadMapDocument()
        {
            System.Windows.Forms.OpenFileDialog openFileDialog;
            openFileDialog = new OpenFileDialog();
```

```
        openFileDialog.Title = "打开地图文档";
        openFileDialog.Filter = "map documents(*.mxd)|*.mxd";
        openFileDialog.ShowDialog();
        string filePath = openFileDialog.FileName;
        if (axMapControl1.CheckMxFile(filePath))
        {
            axMapControl1.MousePointer = esriControlsMousePointer.esriPointer Hourglass;
            axMapControl1.LoadMxFile(filePath, 0, Type.Missing);
            axMapControl1.MousePointer = esriControlsMousePointer.esriPointer Default;
        }
        else
        {
            MessageBox.Show(filePath + "不是有效的地图文档");
        }
    }
    //获取线实体
    private IFeature  getPolylineFeature(string layerName)
    {
        int i = 0;
        ILayer layer=null;
        for (i =0;i<axMapControl1.LayerCount ;i++)
        {
            layer = axMapControl1.Map.get_Layer(i);
            if (layer.Name.ToLower() == layerName)
            {
                break;
            }
        }
        IFeatureLayer featureLayer = layer as IFeatureLayer;
        IFeatureClass featureClass = featureLayer.FeatureClass;
        IQueryFilter queryFilter = new QueryFilterClass();
        queryFilter.WhereClause = "";
        IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
        IFeature feature = featureCursor.NextFeature();
        if (feature != null)
        {
            return feature;
        }
        return null;
    }
    //获取所有的点实体
    private System.Collections.ArrayList getPointFeature(string layerName)
    {
        int i = 0;
        ILayer layer = null;
        for (i = 0; i < axMapControl1.LayerCount; i++)
        {
            layer = axMapControl1.Map.get_Layer(i);
            if (layer.Name.ToLower() == layerName)
            {
                break;
```

```
        }
    }
    IFeatureLayer featureLayer = layer as IFeatureLayer;
    IFeatureClass featureClass = featureLayer.FeatureClass;
    IQueryFilter queryFilter = new QueryFilterClass();
    queryFilter.WhereClause = "";
    IFeatureCursor featureCursor = featureClass.Search(queryFilter, true);
    IFeature feature = featureCursor.NextFeature();
    System.Collections.ArrayList features = new System.Collections.ArrayList();
    while (feature != null)
    {
        features.Add(feature as object);
        feature = featureCursor.NextFeature();
    }
    return features;
}
//添加点实体
private void addFeature(string layerName,IPoint point)
{
    int i = 0;
    ILayer layer = null;
    for (i = 0; i < axMapControl1.LayerCount; i++)
    {
        layer = axMapControl1.Map.get_Layer(i);
        if (layer.Name.ToLower() == layerName)
        {
            break;
        }
    }
    IFeatureLayer featureLayer=layer as IFeatureLayer ;
    IFeatureClass featureClass = featureLayer.FeatureClass;
    IDataset dataset = (IDataset)featureClass;
    IWorkspace workspace = dataset.Workspace;
    IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
    workspaceEdit.StartEditing(true);
    workspaceEdit.StartEditOperation();
    IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
    IFeatureCursor featureCursor = featureClass.Insert(true);

    featureBuffer.Shape = point ;
    object  featureOID = featureCursor.InsertFeature(featureBuffer);
    featureCursor.Flush();
    workspaceEdit.StopEditOperation();
    workspaceEdit.StopEditing(true);
    System.Runtime.InteropServices.Marshal.ReleaseComObject(featureCursor);
}
//沿线创建法
private void button1_Click(object sender, EventArgs e)
{
    IFeature polylineFeature;
    polylineFeature = getPolylineFeature("polyline");
```

```
            if (polylineFeature != null)
            {
                ICurve curve = polylineFeature.Shape as ICurve;
                IConstructPoint constructPoint = new PointClass();
                constructPoint.ConstructAlong(curve,
esriSegmentExtension.esriNoExtension, 10, true );
                IPoint point1 = constructPoint as IPoint ;
                constructPoint = new PointClass();
                constructPoint.ConstructAlong(curve,
esriSegmentExtension.esriNoExtension, 20, false  );
                IPoint point2 = constructPoint as IPoint ;
                addFeature("point", point1);
                addFeature("point", point2);
            }
            axMapControl1.Refresh();
        }
        //角平分线创建法
        private void button2_Click(object sender, EventArgs e)
        {
            IPoint point1, point2, point3,point4;
            point1 = new PointClass();
            point2 = new PointClass();
            point3 = new PointClass();
            point1.PutCoords(0, 0);
            point2.PutCoords(0, 20);
            point3.PutCoords(20, 20);
            IConstructPoint constructPoint = new PointClass();
            constructPoint.ConstructAngleBisector(point1, point2, point3, 50, true);
            point4 = constructPoint as IPoint;
            addFeature("point", point1);
            addFeature("point", point2);
            addFeature("point", point3);
            addFeature("point", point4);
            axMapControl1.Refresh();
        }
        //构造角度距离法
        private void button3_Click(object sender, EventArgs e)
        {
            IPoint point1, point2;
            point1 = new PointClass();
            point1.PutCoords(0, 0);
            double distance = 20;
            double angle = 60;
            IConstructPoint constructPoint = new PointClass();
            double angleRad = angle * 2 * Math.PI / 360;
            constructPoint.ConstructAngleDistance(point1, angleRad, distance);
            point2 = constructPoint as IPoint ;
            addFeature("point", point1);
            addFeature("point", point2);
            axMapControl1.Refresh();
        }
```

```
        //构造角度交点法
        private void button4_Click(object sender, EventArgs e)
        {
            IPoint point1, point2;
            point1 = new PointClass();
            point2 = new PointClass();
            point1.PutCoords(0, 0);
            point2.PutCoords(10,0);
            double angleRed1 = 45 * 2 * Math.PI / 360;
            double angleRed2 = 60 * 2 * Math.PI / 360;
            IConstructPoint constructPoint=new  PointClass ();
            constructPoint.ConstructAngleIntersection
(point1,angleRed1 ,point2,angleRed2);

            addFeature ("point",point1 );
            addFeature ("point",point2 );
            addFeature ("point",constructPoint as IPoint  );
            axMapControl1.Refresh ();
        }
        //构造偏转角度法
        private void button5_Click(object sender, EventArgs e)
        {
            IPoint fromPoint = new PointClass();
            fromPoint.PutCoords(0, 0);
            IPoint toPoint = new PointClass();
            toPoint.PutCoords(1, 1);
            ILine line = new LineClass();
            line.PutCoords(fromPoint, toPoint);
            double distance = 1.4142135623731;
            double angle = Math.PI / 4;
            IConstructPoint constructionPoint = new PointClass();
            constructionPoint.ConstructDeflection(line, distance, angle);
            IPoint point = constructionPoint as IPoint;
            addFeature("point", fromPoint);
            addFeature("point", toPoint);
            addFeature("point", point);
            axMapControl1.Refresh();
        }
        //构造偏转角交点法
        private void button6_Click(object sender, EventArgs e)
        {
            IPoint fromPoint = new PointClass();
            fromPoint.PutCoords(0, 0);
            IPoint toPoint = new PointClass();
            toPoint.PutCoords(1, 1);
            ILine line = new LineClass();
            line.PutCoords(fromPoint, toPoint);
            double startAngle = Math.PI / 4;
            double endAngle = Math.PI / 4;
            IConstructPoint constructionPoint = new PointClass();
            constructionPoint.ConstructDeflectionIntersection(line, startAngle, endAngle,
false);
```

```
    IPoint point = constructionPoint as IPoint;
    addFeature("point", fromPoint);
    addFeature("point", toPoint);
    addFeature("point", point);
    axMapControl1.Refresh();
}
//构造偏移点法
private void button7_Click(object sender, EventArgs e)
{
    IPoint[] points = new IPoint[4];
    for (int i = 0; i < 4; i++)
    {
        points[i] = new PointClass();
    }
    points[0].PutCoords(0, 0);
    points[1].PutCoords(10, 0);
    points[2].PutCoords(20, 0);
    points[3].PutCoords(30, 0);
    IPointCollection polyline = new Polyline();
    IGeometryBridge geometryBride = new GeometryEnvironmentClass();
    geometryBride.AddPoints(polyline as IPointCollection4, ref points);
    IConstructPoint constructionPointRight = new PointClass();
    IConstructPoint constructionPointLeft = new PointClass();
    constructionPointRight.ConstructOffset(polyline as ICurve, esriSegment
    Extension.esriNoExtension, 15, false, 5);
    IPoint outPutPoint1 = constructionPointRight as IPoint;
    constructionPointLeft.ConstructOffset(polyline as ICurve, esriSegment
    Extension.esriNoExtension, 1, false, -5);
    IPoint outPutPoint2 = constructionPointLeft as IPoint;
    addFeature("point", points[0]);
    addFeature("point", points[1]);
    addFeature("point", points[2]);
    addFeature("point", points[3]);
    addFeature("point", outPutPoint1);
    addFeature("point", outPutPoint2);
    axMapControl1.Refresh();
}
//构造平行线上点法
private void button8_Click(object sender, EventArgs e)
{
    IPoint[] points = new IPoint[2];
    for (int i = 0; i < 2; i++)
    {
        points[i] = new PointClass();
    }
    points[0].PutCoords(0, 0);
    points[1].PutCoords(20, 0);


    ISegment segment;
```

```
            ILine line =new LineClass ();
            line.FromPoint =points [0];
            line.ToPoint =points [1];

            segment=line as ISegment ;
            IPoint fromPoint = new PointClass();
            fromPoint.X = points[0].X + 10;
            fromPoint.Y = points[0].Y + 5;
            IConstructPoint constructPoint = new PointClass();
            constructPoint.ConstructParallel(segment,
esriSegmentExtension.esriNoExtension, fromPoint, segment.Length);
            addFeature("point", points[0]);
            addFeature("point", points[1]);
            addFeature("point", constructPoint as IPoint );
            axMapControl1.Refresh();
        }
        //构造垂直线上点法
        private void button9_Click(object sender, EventArgs e)
        {
            IPoint[] points = new IPoint[2];
            for (int i = 0; i < 2; i++)
            {
                points[i] = new PointClass();
            }
            points[0].PutCoords(0, 0);
            points[1].PutCoords(20, 0);


            ISegment segment;
            ILine line = new LineClass();
            line.FromPoint = points[0];
            line.ToPoint = points[1];

            segment = line as ISegment;
            IPoint fromPoint = new PointClass();
            fromPoint.X = points[0].X + 10;
            fromPoint.Y = points[0].Y + 10;
            IConstructPoint constructPoint = new PointClass();
            constructPoint.ConstructPerpendicular (segment, esriSegmentExtension.
            esriNoExtension, fromPoint, 3,false );
            addFeature("point", points[0]);
            addFeature("point", points[1]);
            addFeature("point", constructPoint as IPoint);
            axMapControl1.Refresh();
        }
        //后方交会定点法
        private void button10_Click(object sender, EventArgs e)
        {

            IPoint[] points = new IPoint[3];
            for (int i = 0; i < 3; i++)
```

```
            {
                points[i] = new PointClass();
            }
            IConstructPoint constructPoint = new PointClass();
            points[0].PutCoords(0,10);
            points[1].PutCoords(20,20);
            points[2].PutCoords(0,0);
            double angle1=Math.PI /4;
            double angle2=Math.PI /4;
            double angle3 =0;
            try
            {
                constructPoint.ConstructThreePointResection(points[0],angle1,
                points [1],angle2, points[2], out angle3);
                addFeature("point", points[0]);
                addFeature("point", points[1]);
                addFeature("point", points[2]);
                addFeature("point", constructPoint as IPoint);
            }
            catch (Exception ex)
            {

            }
            axMapControl1.Refresh();
        }
    }
}
```

Multipoint对象是一个具有相同结构的点的集合,通过使用Add方法来添加一个点到集合中。这个过程需要使用 IPointCollection 接口来完成，在 IConstructMultipoint 接口中定义了“ConstructArcPoints”、“ConstructDivideEqual”、“ConstructDivideLength”、“ConstructIntersection”、“ ConstructTangent”等5种构造方法。

（1）ConstructArcPoints 构造圆弧点：通过给定的一段圆弧，返回该圆弧的起点、终点、圆心和切线的交点，如图3-29所示。

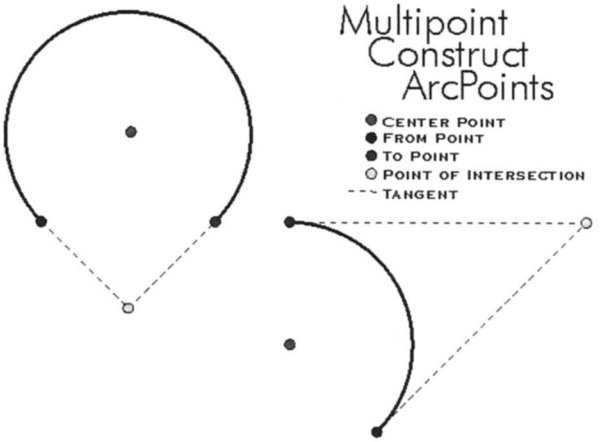


图3-29 构造圆弧点

（2）ConstructDivideEqual 构造等分点法：通过一条曲线和需要返回的点数来产生一个点集合，如图 3-30 所示。

（3）ConstructDivideLength 构造等长点法：通过给定的一条曲线对象和已经定义的间距长度，返回所有处于这条曲线上的点。

（4）ConstructIntersection 构造交点法：通过给定的两条曲线对象，以及这两条曲线的延长线、切线，计算这些线的交点，如图 3-31 所示。

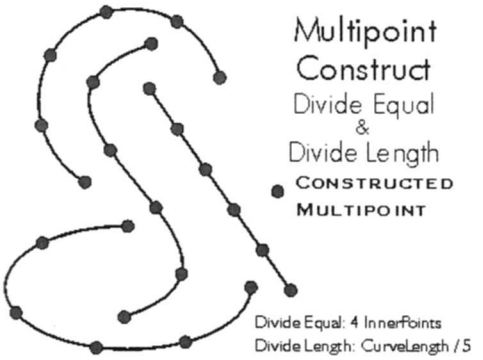


图 3-30　构造等分点

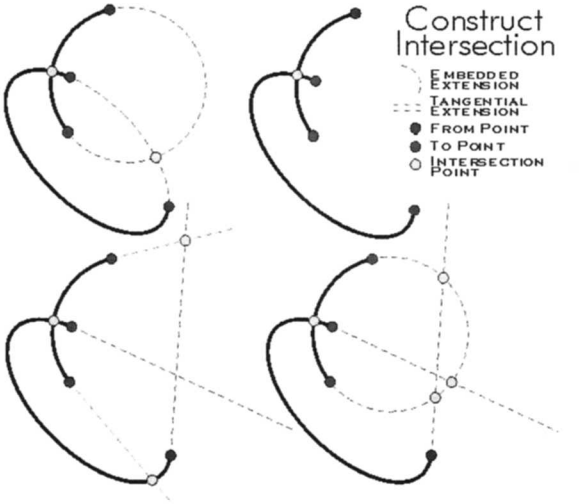


图 3-31　构造交点法

（5）ConstructTangent 构造切线点：通过一条曲线和一个指定的点，产生指定的点到曲线的切线，如图 3-32 所示。

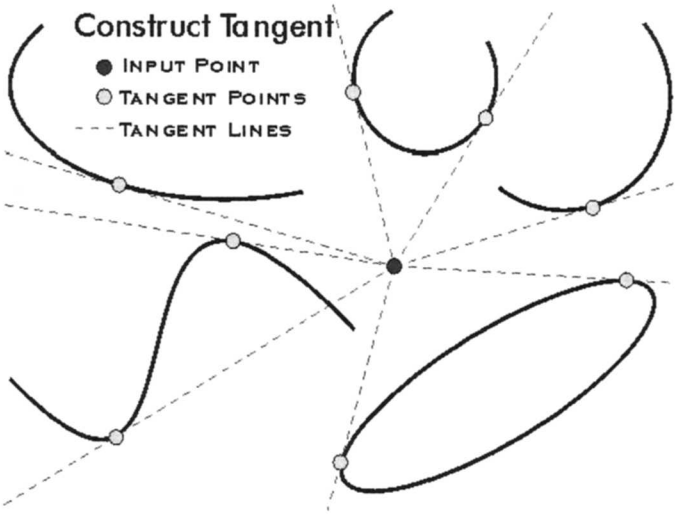


图 3-32　构造切线点

新建一个 VS2005 工程，工程命名为“lesson3”，在 FORM1 窗体上添加一个 ToolbarControl 控件，一个 MapControl 控件，一个 LicenseControl1 控件，5 个 Button 控件，然后在 ToolbarControl 控件的属性里设置“伙伴控件”为 MapControl，界面效果如图 3-33 所示。

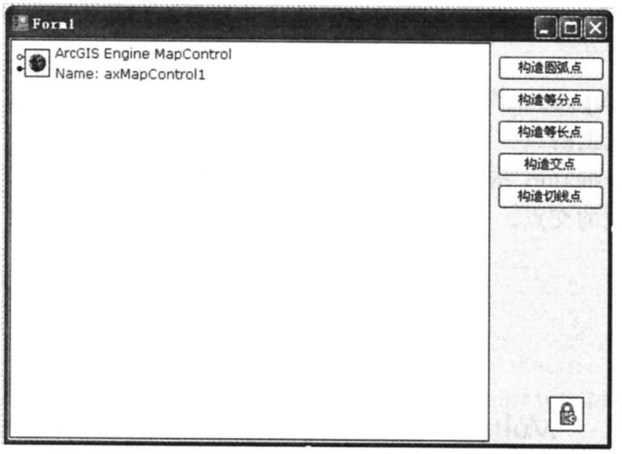


图 3-33　界面效果

在代码窗口中添加如下代码。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Controls;
using ESRI.ArcGIS.SystemUI;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Geodatabase;
namespace lesson2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        //加载地图文档
        private void loadMapDocument()
        {
            System.Windows.Forms.OpenFileDialog openFileDialog;
            openFileDialog = new OpenFileDialog();
            openFileDialog.Title = "打开地图文档";
            openFileDialog.Filter = "map documents(*.mxd)|*.mxd";
            openFileDialog.ShowDialog();
            string filePath = openFileDialog.FileName;
            if (axMapControl1.CheckMxFile(filePath))
            {
                axMapControl1.MousePointer = esriControlsMousePointer.esriPointer Hourglass;
```

```
                axMapControl1.LoadMxFile(filePath, 0, Type.Missing);
                axMapControl1.MousePointer = esriControlsMousePointer.esriPointer Default;
            }
            else
            {
                MessageBox.Show(filePath + "不是有效的地图文档");
            }
        }
        //添加点实体
        private void addFeature(string layerName, IPoint point)
        {
            int i = 0;
            ILayer layer = null;
            for (i = 0; i < axMapControl1.LayerCount; i++)
            {
                layer = axMapControl1.Map.get_Layer(i);
                if (layer.Name.ToLower() == layerName)
                {
                    break;
                }
            }
            IFeatureLayer featureLayer = layer as IFeatureLayer;
            IFeatureClass featureClass = featureLayer.FeatureClass;
            IDataset dataset = (IDataset)featureClass;
            IWorkspace workspace = dataset.Workspace;
            IWorkspaceEdit workspaceEdit = (IWorkspaceEdit)workspace;
            workspaceEdit.StartEditing(true);
            workspaceEdit.StartEditOperation();
            IFeatureBuffer featureBuffer = featureClass.CreateFeatureBuffer();
            IFeatureCursor featureCursor = featureClass.Insert(true);

            featureBuffer.Shape = point;
            object featureOID = featureCursor.InsertFeature(featureBuffer);
            featureCursor.Flush();
            workspaceEdit.StopEditOperation();
            workspaceEdit.StopEditing(true);

System.Runtime.InteropServices.Marshal.ReleaseComObject(featureCursor);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            loadMapDocument();
        }
        //构造圆弧点
        private void button1_Click(object sender, EventArgs e)
        {
            //构造一段圆弧
            IPoint centerPoint = new PointClass();
            centerPoint.PutCoords(10, 0);
            IPoint fromPoint = new PointClass();
```

```
            fromPoint.PutCoords(0, 0);
            IPoint toPoint = new PointClass();
            toPoint.PutCoords(0, 20);
            IConstructCircularArc circularArcConstruction = new CircularArcClass();
            circularArcConstruction.ConstructThreePoints(fromPoint, centerPoint,
to Point, false);
            //构造圆弧点
            IConstructMultipoint constructMultipoint = new MultipointClass();
            constructMultipoint.ConstructArcPoints(circularArcConstruction    as
ICircularArc);
            IPointCollection pointCollection = constructMultipoint as IPointCollection;
            for (int i = 0; i < pointCollection.PointCount; i++)
            {
                addFeature("point", pointCollection.get_Point(i)  );
            }

            axMapControl1.Refresh();

        }
        //构造等分点
        private void button2_Click(object sender, EventArgs e)
        {
            IPoint centerPoint = new PointClass();
            centerPoint.PutCoords(10, 10);
            IPoint fromPoint = new PointClass();
            fromPoint.PutCoords(0, 0);
            IPoint toPoint = new PointClass();
            toPoint.PutCoords(10, 20);
            ICircularArc circularArcConstruction = new CircularArcClass();
            circularArcConstruction.PutCoords(centerPoint,  fromPoint,  toPoint,
            esriArc Orientation.esriArcClockwise);
            IConstructMultipoint constructMultipoint = new MultipointClass();
            constructMultipoint.ConstructDivideEqual(circularArcConstruction  as
            ICurve, (int)circularArcConstruction.Length /5);
            IPointCollection pointCollection = constructMultipoint as IpointColl-
            ection;
            for (int i = 0; i < pointCollection.PointCount; i++)
            {
                addFeature("point", pointCollection.get_Point(i));
            }

            axMapControl1.Refresh();
        }
        //构造等长点
        private void button3_Click(object sender, EventArgs e)
        {
            IPoint centerPoint = new PointClass();
            centerPoint.PutCoords(10, 0);
            IPoint fromPoint = new PointClass();
            fromPoint.PutCoords(0, 0);
            IPoint toPoint = new PointClass();
```

```
            toPoint.PutCoords(0, 20);
            ICircularArc circularArcConstruction = new CircularArcClass();
            circularArcConstruction.PutCoords(centerPoint,fromPoint,toPoint,
            esriArcOrientation.esriArcClockwise);
            IConstructMultipoint constructMultipoint = new MultipointClass();
            constructMultipoint.ConstructDivideLength(circularArcConstruction as Icurve, 10);
            IPointCollection pointCollection = constructMultipoint as IPointCollection;
            for (int i = 0; i < pointCollection.PointCount; i++)
            {
                addFeature("point", pointCollection.get_Point(i));
            }

            axMapControl1.Refresh();

        }
        //构造交点
        private void button4_Click(object sender, EventArgs e)
        {
            IPoint[] points = new IPoint[4];
            for (int i = 0; i < 4; i++)
            {
                points[i] = new PointClass();
            }

            points[0].PutCoords(15, 10);
            points[1].PutCoords(20, 60);
            points[2].PutCoords(40, 60);
            points[3].PutCoords(45, 10);
            //构造 Bezier 曲线
            IBezierCurveGEN bezierCurve = new BezierCurveClass();
            bezierCurve.PutCoords(ref points);
            IPoint centerPoint = new PointClass();
            centerPoint.PutCoords(30, 30);
            IPoint fromPoint = new PointClass();
            fromPoint.PutCoords(10, 10);
            IPoint toPoint = new PointClass();
            toPoint.PutCoords(50, 10);
            //构造圆弧
            IConstructCircularArc circularArcConstruction = new CircularArcClass();
            circularArcConstruction.ConstructThreePoints(fromPoint, centerPoint,
toPoint, false);


            object param0;
            object param1;
            object isTangentPoint;
            IConstructMultipoint constructMultipoint = new MultipointClass();
            constructMultipoint.ConstructIntersection(circularArcConstruction as
ISegment, esriSegmentExtension.esriNoExtension, bezierCurve as ISegment, esri Seg
mentExtension.esriNoExtension, out param0, out param1, out isTangentPoint);
```

```
            IMultipoint multipoint = constructMultipoint as IMultipoint;
            IPointCollection pointCollection = multipoint as IPointCollection;
            for (int i = 0; i < pointCollection.PointCount; i++)
            {
                addFeature("point", pointCollection.get_Point(i));
            }

            axMapControl1.Extent = multipoint.Envelope;
            axMapControl1.Refresh();

        }
        //构造切线点
        private void button5_Click(object sender, EventArgs e)
        {
            IPoint[] points = new IPoint[4];
            for (int i = 0; i < 4; i++)
            {
                points[i] = new PointClass();
            }

            points[0].PutCoords(15, 10);
            points[1].PutCoords(20, 60);
            points[2].PutCoords(40, 60);
            points[3].PutCoords(45, 10);

            IBezierCurveGEN bezierCurve = new BezierCurveClass();
            bezierCurve.PutCoords(ref points);
            IConstructMultipoint constructMultipoint = new MultipointClass();
            constructMultipoint.ConstructTangent(bezierCurve as ICurve, points[1]);
            IMultipoint multipoint = constructMultipoint as IMultipoint;
            IPointCollection pointCollection = multipoint as IPointCollection;
            for (int i = 0; i < pointCollection.PointCount; i++)
            {
                addFeature("point", pointCollection.get_Point(i));
            }
            axMapControl1.Refresh();
        }
    }
}
```

3.5 线对象

Polyline（多义线）对象是相连或不相连的路径对象的有序集合，如图 3-34 所示。组成 Polyline 的 Path 对象都是有效的，Path 不会重合、相交或自相交。多个 Path 对象可以连接于一个节点，也可以是分离的，长度为 0 的 Path 对象是不被允许的。

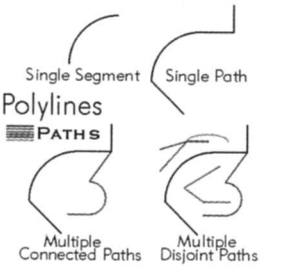


图 3-34 线对象

IPolyline 是 Polyline 类的主要接口，它定义了两个主要的方法，一个方法是使用一个路径对象给一个存在的 Polyline 整形；另一个方

法是 SimplifyNetwork，该方法主要用于清除 0 长度的 Segment，并重新组织 Polyline 中的 Segment。

Polyline 构造可以通过 IGeometryCollection 接口添加路径对象来产生，可以参阅 3.1 节的代码。

3.6　面对象

Polygon 对象是一个有序 Ring 对象的集合，如图 3-35 所示。

构造 Polygon 对象必须保证每个构成 Ring 都是有效的，Ring 之间的边界不能重合，外部 Ring 方向是顺时针，内部 Ring 方式是逆时针，不存在面积为 0 的 Ring。IPolygon 接口是 Polygon 类的主要接口，定义了一系列的属性和方法来控制 Ring。如 ExteriorRingCount 属性用于返回一个多边形的全部外部环的数目，InteriorRingCount 属性用于返回一个多边形内部环的数目，QueryExteriorRings 属性用于查询全部的外部 Ring，QueryInteriorRings 属性则用于查询全部的内部 Ring。Polygon 对象也可以使用 IGeometryCollection 接口的 AddSegment 方法添加 Ring 来产生，可以参阅 3.1 节的代码。

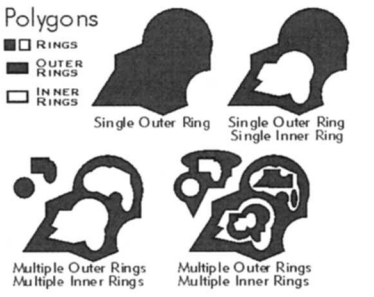


图 3-35　面对象

3.7　空间坐标系及变换

空间数据都有一个坐标系统（地理坐标系统或投影坐标系统），它定义了空间数据在地球上的位置。一幅地图上显示的空间数据地理坐标系是一致的，否则会造成数据无法正确拼合。

地理坐标系统也称为真实世界的坐标系，是确定地物在地球上位置的坐标系，以经纬度作为地图的存储单位。

投影坐标系统是将三纬地理坐标系统上的经纬网投影到二纬平面地图上使用的坐标系统，因此地理信息系统必然要考虑地图投影，地图投影的使用保证了空间信息在地域上的联系和完整性，在各类地理信息系统的建设过程中，都要考虑选择何种地图投影系统。

Arc Engine 提供了 3 种组件：ProjectedCoordinateSystem、GeographicCoordinateSystem 和 SpatialReferenceEnvironmentClass，这些组件可用于自定义坐标系统。下面的代码演示了如何使用这些组件。

```
//改变一个图层的空间参考
IFeatureLayer player ;
player = pMap.get_Layer(0) as IFeatureLayer;
IFeatureClass pFeatureClass;
pFeatureClass = player.FeatureClass;
IGeoDataset pGeoDataset;
pGeoDataset = pFeatureClass as IGeoDataset;
IGeoDatasetSchemaEdit pGeoDatasetEdit;
pGeoDatasetEdit = pGeoDataset as IGeoDatasetSchemaEdit;
if (pGeoDatasetEdit.CanAlterSpatialReference == true)
{
ISpatialReferenceFactory2 pSpatRefFact;
pSpatRefFact = new SpatialReferenceEnvironmentClass();
IGeographicCoordinateSystem pGeoSys;
pGeoSys =
```

```
    pSpatRefFact.CreateGeographicCoordinateSystem(4214);
    pGeoDatasetEdit.AlterSpatialReference(pGeoSys);
    }
    pActiveView.Refresh();
    //得到一个图层的空间参考
    IFeatureLayer pLayer;
    pLayer = pMap.get_Layer(0)as IFeatureLayer;
    IGeoDataset pGeoDataset;
    ISpatialReference pSpatialReference;
    pGeoDataset = pLayer as IGeoDataset;
    pSpatialReference = pGeoDataset.SpatialReference;
    MessageBox.Show(pSpatialReference.Name);
    //设置一个地图的空间参考，使用一个空间参考对话框
    IProjectedCoordinateSystem pSpatialReference;
    ISpatialReferenceDialog pDialog ;
    pDialog = new SpatialReferenceDialogClass();
    pSpatialReference = pDialog.DoModalCreate(true, false,false,0) as IprojectedCoordinate
System;
    pMap.SpatialReference = pSpatialReference;
    pActiveView.Refresh();
```

Arc Engine 的空间参考对象模型中有 3 个组件类：GeographicCoordinateSystem（地理坐标系统）、ProjectedCoordinateSystem（投影坐标系统）和 UnknownCoordinateSystem（未知坐标系统），这 3 个组件类都实现了 ISpatialRefrence 接口，该接口提供了操作方法和属性来设置一个数据集空间参考属性，如空间域和坐标精度等。如 Changed 是这个接口中最重要的一个方法，用于检查一个坐标系统中的参数是否发生了变化；GetDomain 和 SetDomain 方法分别用于获取和设置一个坐标系统的域范围等。

IGeographicCoordinateSystem 是 GeographicCoordinateSystem 类接口，提供了 CoordinateUnit（坐标系的角度单位）、Datum（椭球体）和 PrimeMeridian（本初子午线）等属性。

IProjectedCoordinateSystem 是 ProjectedCoordinateSystem 类的接口，该接口提供了新建一个投影坐标系统的方法。在一个新的投影坐标系统中，需要设置 projection（投影方式）、GeographicCoordinateSystem、CoordinateUnit 和 Parameters 等。

3.8 本章小结

本章介绍了基于 ArcEngine 的几何对象模型和空间参考内容，ArcEngine 的几何对象是一种逐级组合的关系，通过线段组合为路径，路径组合为多义线，环组合为多边形。本章详细介绍了这些几何对象的组件类以及组件类实现的接口，特别是对象的创建，并给出了各种对象创建的构造方法实例。