Python 基础开发帮助文档

Python 是一种类似于自然语言的脚本语言，因此，它具有语法简洁，开发简单的特点。python 运行时不必编译，运行时速度较快，最重要的特点是 python 的可嵌入性，可以把 Python 脚本嵌入到 C/C++程序，它能够把其他语言制作的各种模块，轻松地联结在一起，从而向程序用户提供脚本功能。

另外，python 的特点还有可移植性高以及丰富的库支持。正是基于以上 python 的特点，我们选用 python 来设计程序嵌入到 Ucancam 软件中，而 Ucan.pyd 是我们为用户封装的 python 模块,用户可以通过使用 python 加载 Ucan.pyd 模块，来快速实现编写用户自己需要绘图程序。在 Ucan.pyd 中提供了点(Point)、矢量(Vector)、点集合(PointList)、曲线集合(CurveList)以及二维塑造类(shape2)等一些点、矢量和曲线生成的方法，用户可以使用这些方法来编写自己需要的绘制图形的 python 程序来嵌入到 Ucancam 软件中，扩展绘图功能。用户可以在菜单中，选择 【文件 > 读入文件 > python file(*.py)】，来导入设计好的 python 文件到软件中。

目录

Point(float x ,float y ,float z)点

点 Point(float x ,float y ,float z)，根据指定 x ， y ,z 坐标来确定的点。当 x 和 y 赋值时，而 z 未赋值时，会默认 z=0,来定义一个点。其中，x ,y ， z 是点 Point p1 对象成员，可以使用 p1.x、p1.y、p1.z 的方式，对点 p 的 x，y ,z 坐标进行取值和赋值运算。例如，赋值运算：p1.x=4 p1.y=5 p1.z=6 ，取值运算：a=p1.x b=p1.y c=p1.z 。

定义点:

p1= Point(1,2,3)

x ,y ,z 均赋值。

结果：

p1.x=1 p1.y=2 p1.z=3

p1=Point(1,2)

x 和 y 赋值，而 z 未赋值。

结果：

p1.x=1 p1.y=2 p1.z=0

点的方法

点的方法，也就是点中的一些运算方法，通过这些方法来实现点之间的加减乘除运算操作等。其中的方法有：

Point operator+(Vector v1)

点p1与矢量v1的各坐标完成相加，生成新的点p2。其中“operator+” 为运算符号“+”。

使用方法:

```
p1=Point(0,1,2)
v1=Vector(1,2,3)
p2=p1+v1
```

结果：

p2.x=1 p2.y=3 p2.z=5

void operator+=(Vector v1)

点p1与矢量v1的各坐标完成相加，结果保存在p1中。其中“operator+=” 为运算符号“+=”。

使用方法：

```
p1=Point(0,1,2)
v1=Vector(1,2,3)
p1+=v1
```

结果：

```
p1.x=1        p1.y=3        p1.z=5
```

Point operator-(Vector v1)

点p1与矢量v1的各坐标完成相减，生成新的点p2。其中“operator-” 为运算符号“-”。

使用方法：

```
p1=Point(1,3,4)
v1=Vector(1,0,2)
p2=p1-v1
```

结果：

```
p2.x=0        p2.y=3        p2.z=2
```

void operator-=(Vector3 v1)

点p1与矢量v1的各坐标完成相减，结果保存在p1中。其中“operator-=” 为运算符号“-=”。

使用方法：

```
p1=Point(3,2,4)
v1=Vector(1,1,3)
p1-=v1
```

结果：

```
p1.x=2        p1.y=1        p1.z=1
```

Point operator*(float d)

点 p1 的各坐标同乘以浮点类型数 d，生成点 p2。其中“operator*” 为运算符号“*”。

使用参数：浮点类型d

使用方法：

```
p1=Point(1,2,3)
p2=p1*2
```

结果：

```
p2.x=2        p2.y=4        p2.z=6
```

void operator*=(float d)

点 p1 的各坐标同乘以浮点类型数 d。结果保存在 p1 中。其中“operator*=”为运算符号“*=”。

使用参数:浮点类型d

使用方法:

```
p1=Point(1,2,3)
p1*=2
```

结果:

```
p1.x=2        p1.y=4        p1.z=6
```

void operator/=(float d)

点 p1 的各坐标同除以浮点类型数 d。结果保存在 p1 中。其中“operator/=”为运算符号“/=”。

使用参数:浮点类型d

使用方法:

```
p1=Point(2,4,6)
p1/=2
```

结果:

```
p1.x=1        p1.y=2        p1.z=3
```

Vector operator-(Point p2)

点p2与p1相减，生成矢量v。其中“operator-” 为运算符号“-”。

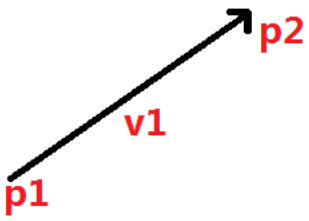
使用参数:浮点类型d

使用方法:

```
p1=Point(2,3,5)
p2=Point(1,1,2)
v1=p2-p1
```

结果:

```
v1.dx=1       v1.dy=2       v1.dz=3
```



float Distance(Point p2)

计算点p1与p2之间的距离

使用参数:点p2

使用方法:

```
p1=point3(3,0,0)
p2=point3(0,4,0)
distance=p1.Distance(p2)
```

结果:

```
distance=5.0 。
```

Vector(float dx ,float dy ,float dz)　矢量

矢量 Vector(float dx ,float dy ,float dz)，根据 dx ， dy ,dz 坐标生成矢量。当 dx,dy,dz 均未赋值时默认 dx=0,dy=0,,dz=0。当 dx 和 dy 赋值，而 dz 未赋值时，会默认 dz=0,来定义一个矢量。其中，dx ,dy ， dz 是矢量 Vector　v1 对象成员，可以使用 v1.dx、v1.dy、v1.dz 的方式，对矢量 v1 的 dx ， dy ,dz 坐标进行取值和赋值运算。例如，赋值运算：v1.dx=4　v1.dy=5　v1.dz=6 ，取值运算：a=v1.dx　b=v1.dy　c=v1.dz 。

定义矢量:

v1=Vector(1,2,3)

dx,dy,dz 均赋值

结果：

v1.dx=1　　v1.dy=2　　v1.dz=3

v1=Vector(1,2)

dx 和 dy 赋值，dz 未赋值

结果：

v1.dx=1　　v1.dy=2　　v1.dz=0

v1=Vector()：

dx,dy,dz 均未赋值

结果：

v1.dx=0　　v1.dy=0　　v1.dz=0

矢量的方法：

矢量的方法，也就是矢量中的一些运算方法，通过这些方法来实现矢量之间的加减乘除运算操作等。其中的方法有：

Vector operator+(Vector v2)

矢量v1与矢量v2的各坐标完成相加，生成矢量v。其中“operator+” 为运算符号“+”。

使用方法：

```
v1=Vector(0,1,2)
v2=Vector(1,2,3 )
v=v1+v2
```

结果：

```
v.dx=1    v.dy=3    v.dz=5
```

void operator+=(Vector v2)

矢量v1与矢量v2的各坐标相加，结果保存在v1中。其中“operator+=” 为运算符号“+=”。

使用方法：

```
v1=Vector(0,1,2)
v2=Vector(1,2,3)
v1+=v2
```

结果：

```
v1.dx=1    v1.dy=3    v1.dz=5
```

Vector operator-(Vector v2)

矢量v1与矢量v2的各坐标相减，生成矢量v。其中“operator-” 为运算符号“-”。

使用方法：

```
v1=Vector(3,4,5)
v2=Vector(1,2,3)
v=v1-v2
```

结果：

```
v.dx=2    v.dy=2    v.dz=2
```

void operator-=(Vector v2)

矢量v1与矢量v2的各坐标相减，结果保存在v1中。其中“operator-=” 为运算符号“-=”。

使用方法：

```
v1=Vector(3,4,5)
v2=Vector(1,2,3)
v1-=-v2
```

结果：

```
v1.dx=2    v1.dy=2    v1.dz=2
```

Vector operator*(float d)

矢量 v1 的各坐标同乘以浮点类型数 d, 生成矢量 v2。其中“operator*” 为运算符号“*”。

使用参数:浮点类型d

使用方法:

```
v1=Vector(1,2,3)
v2=v1*2
```

结果:

```
v2.dx=2    v2.dy=4    v2.dz=6
```

void operator*=(float d)

矢量 v1 的各坐标同乘以浮点类型数 d。结果保存在 v1 中。其中“operator*=” 为运算符号“*=”。

使用参数:浮点类型d

使用方法:

```
v1=Vector(1,2,3)
v1*=2
```

结果:

```
v1.dx=2    v1.dy=4    v1.dz=6
```

void operator/=(float d)

矢量 v1 的各坐标同除以浮点类型数 d。结果保存在 v1 中。其中“operator/=” 为运算符号“/=”。

使用参数:浮点类型d

使用方法:

```
v1=Vector(2,4,6)
v1/=2
```

结果:

```
v1.dx=1    v1.dy=2    v1.dz=3
```

float GetLength()

返回矢量 v1 的长度。

使用参数:无

使用方法:

```
v1=Vector3(3,4,0)
length=v1.GetLength()
```

结果:

length=5.0 。

float AngleToVec(Vector v2)

计算矢量 v1 与 v2 的夹角。

使用参数:矢量 v2

使用方法:

```
v1=Vector(3,0,0)
v2=Vector(3,3,0)
angle=v1.AngleToVec(v2)
```

结果:

angle=60°

PointList 点集合

点集合，用于存放点的集合。调用方法 append(Point p)来添加点到点集合中。

点集合的方法:

append(Point p1)

向点集合中添加一个点 p1。

使用参数: 点 p1

使用方法:

```
p1=Point3(15,10,14)
p2=Point3(5,7,6)
lst=PointList()
lst.append(p1)
lst.append(p2)
```

本示例中，分别创建两个点p1，p2与一个点集合lst，然后调用点集合中的append(Point p)方法命令，向点集合lst中添加两个点p1与p2。

CurveList 曲线集合

曲线集合，用于存放曲线的集合。调用方法 append(Curve)来添加一些列曲线到曲线集合中。

曲线集合的方法：

append(Curve)

向CurveList曲线集合中添加一系列曲线

使用参数：曲线（直线，弧线，圆，椭圆，贝塞尔曲线等）

使用方法：

```
p1=Point3(0,0,0)
p2=Point3(10,10,0)
p3=Point3(20,20,0)
p4=Point3(30,10,0)
CurveList=CurveList()
shape=Shape2()
Line=shape.CreateLine(p1,p2)
Arc=shape.CreateArc3Pt(p2,p3,p4)
CurveList.append(Line)
CurveList.append(Arc)
```

本示例中，分别创建四个三维点p1，p2，p3,p4与一个曲线集合CurveList和一个二维塑造类型shape，然后调用塑造类shape中绘制直线与绘制三点弧的命令生成直线与三点弧，再使用曲线集合中的append(Curve)方法命令来添加直线与三点弧线。

Shape2( ) 二维塑造类型

Shape2（）主要是开发中提供一些我们为用户封装好的二维情况下的绘图方法。

定义一个二维塑造类：

shape=Shape2()：

这样就定义了一个名为 shape 的二维塑造类。

二维塑造类的方法：

Shape2（）提供的二维情况下的绘图方法主要有：

CreatePoint(Point p1)

根据点p1，创建点。

使用参数:点 p1

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
shape=Shape2()
point=shape.CreatePoint(p1)
shape.AddCurveToSystem(point)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义一个点p1和一个塑造类型shape,然后用调用shape中绘制点的命令生成point,最后将point添加到系统中并显示出来。

CreateLine(Point p1, Point p2)

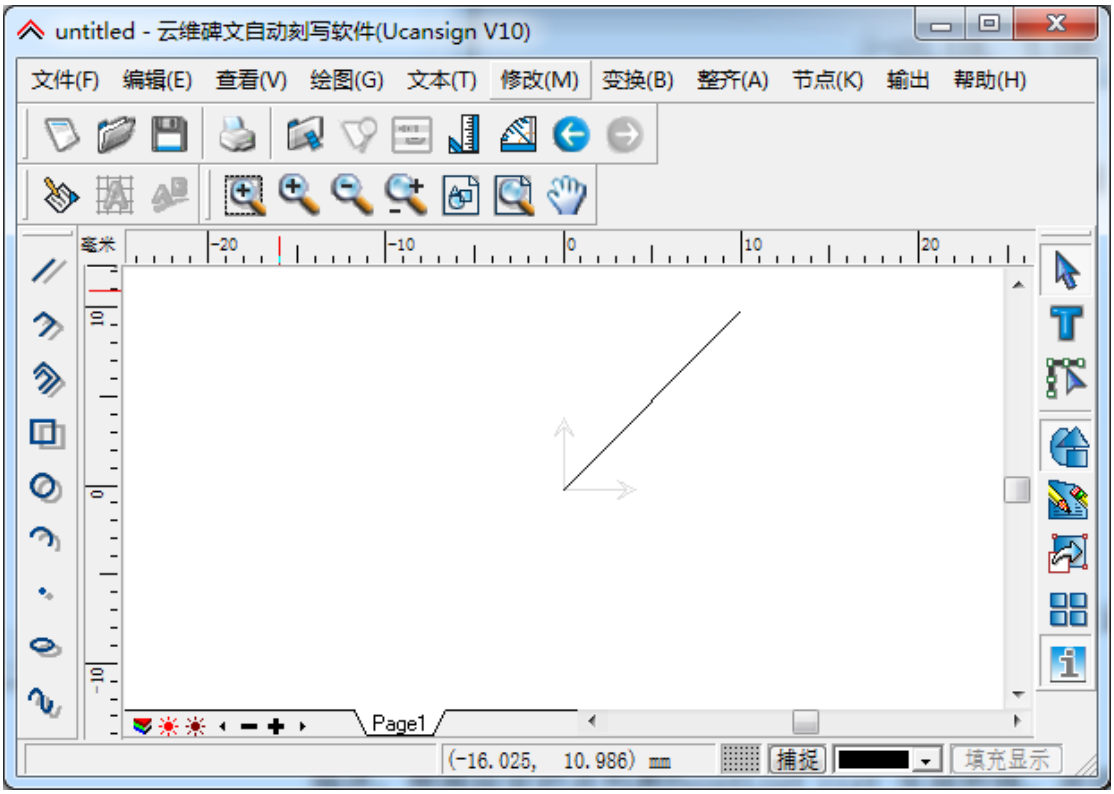
根据点p1与p2创建两点直线

使用参数: 点p1 和 p2

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(10,10,0)
shape=Shape2()
Line=shape.CreateLine(p1,p2)
shape.AddCurveToSystem(Line)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义两个点p1，p2和一个塑造类型shape,然后用调用shape中绘制直线的命令生成Line,最后将Line添加到系统中并显示出来。



CreatePolyline(PointList list)

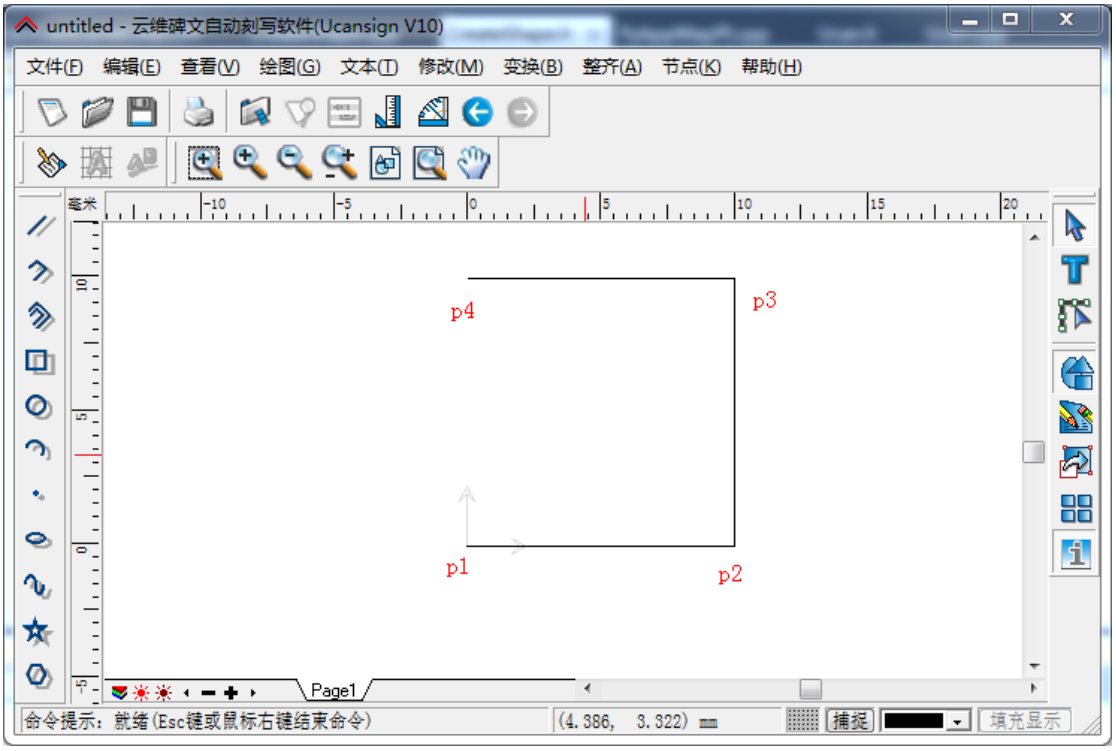
根据给定的点集合保存的点，生成折线。

使用参数： 点列表list

使用方法：

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(10,0,0)
p3=Point(10,10,0)
p4=Point(0,10,0)
lst=PointList()
lst.append(p1)
lst.append(p2)
lst.append(p3)
lst.append(p4)
shape=Shape2()
PolyLine=shape.CreatePolyline(lst)
shape.AddCurveToSystem(PolyLine)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义四个点p1，p2，p3，p4和一个点集合lst和一个塑造类型shape,然后调用点集合lst中append(Point p)方法向点集合添加四个点，调用shape中绘制折线的命令生成PolyLine,最后将PolyLine添加到系统中并显示出来。



CreateArc3Pt(Point p1, Point p2,Point p3)

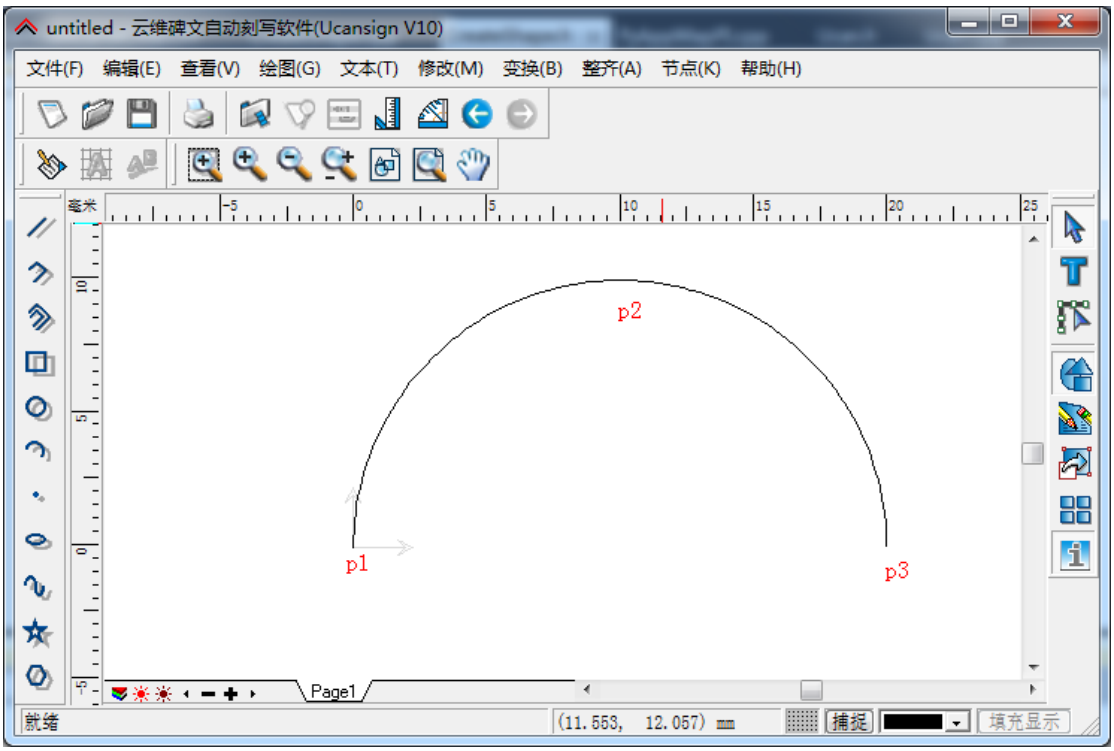
根据三个点生成弧线

使用参数: 点 p1 ， p2 ， p3

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(10,10,0)
p3=Point(20,0,0)
shape=Shape2()
Arc=shape.CreateArc3Pt(p1,p2,p3)
shape.AddCurveToSystem(Arc)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义三个点p1，p2，p3和一个塑造类型shape,然后用调用shape中绘制三点弧线的命令生成Arc,最后将Arc添加到系统中并显示出来。



CreateArcCSE(Point center, Point start,Point end,int dir)

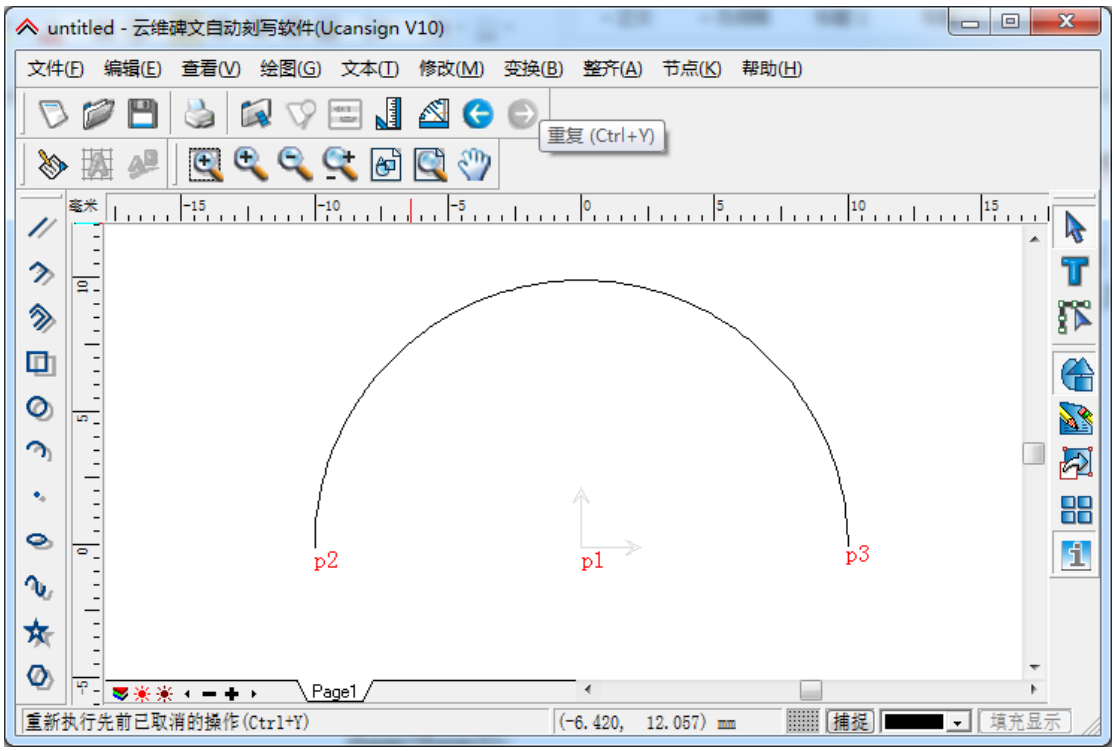
根据弧线的中心点center ,弧线的起始点 start 弧线的终点 end，以及绘制弧线的方向（dir=0时顺时针绘制，dir=1时逆时针绘制）生成弧线。

使用参数:点 center ， start ， end 和整数型 int dir

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(-10,0,0)
p3=Point(10,0,0)
dir=0
shape=Shape2()
Arc=shape.CreateArcCSE(p1,p2,p3,dir)
shape.AddCurveToSystem(Arc)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义三个点p1（弧线中心），p2(弧线起点)，p3(弧线终点)和nDir(绘制的方向)和一个塑造类型shape,然后用调用shape中绘制弧线的命令生成Arc,最后将Arc添加到系统中并显示出来。



CreateCircle(Point center, float rad,int nDir)

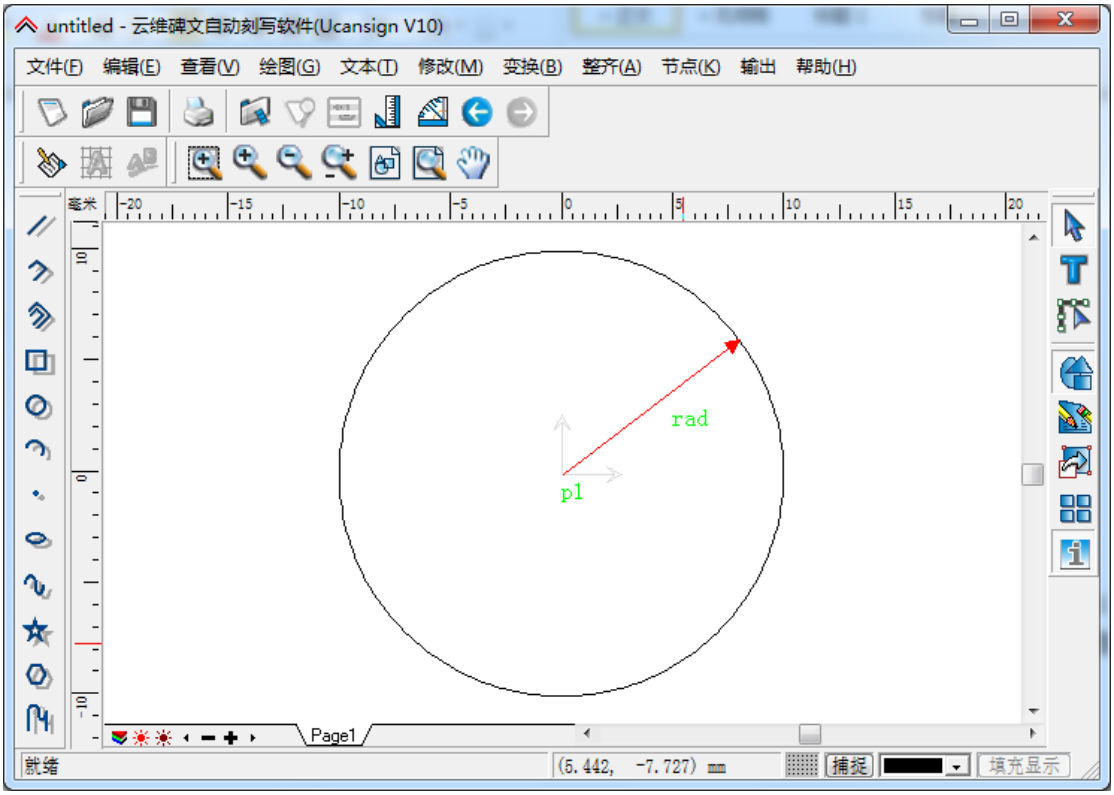
根据圆心点 center，半径 rad，以及绘制圆的方向（nDir=0时顺时针绘制，nDir=1时逆时针绘制）生成圆。

使用参数：点center，浮点类型 rad 以及int nDir

使用方法：

```
from Ucan import *
p1=Point(0,0,0)
rad=10
nDir=0
shape=Shape2()
Circle=shape.CreateCircle(p1,rad,nDir)
shape.AddCurveToSystem(Circle)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义一个点p1（圆心）和rad（半径）和nDir(绘制方向)和一个塑造类型shape,然后用调用shape中绘制圆的命令生成Circle,最后将Circle添加到系统中并显示出来。



CreateEllipse(Point center, float radX,float radY, float angX,int nDir)

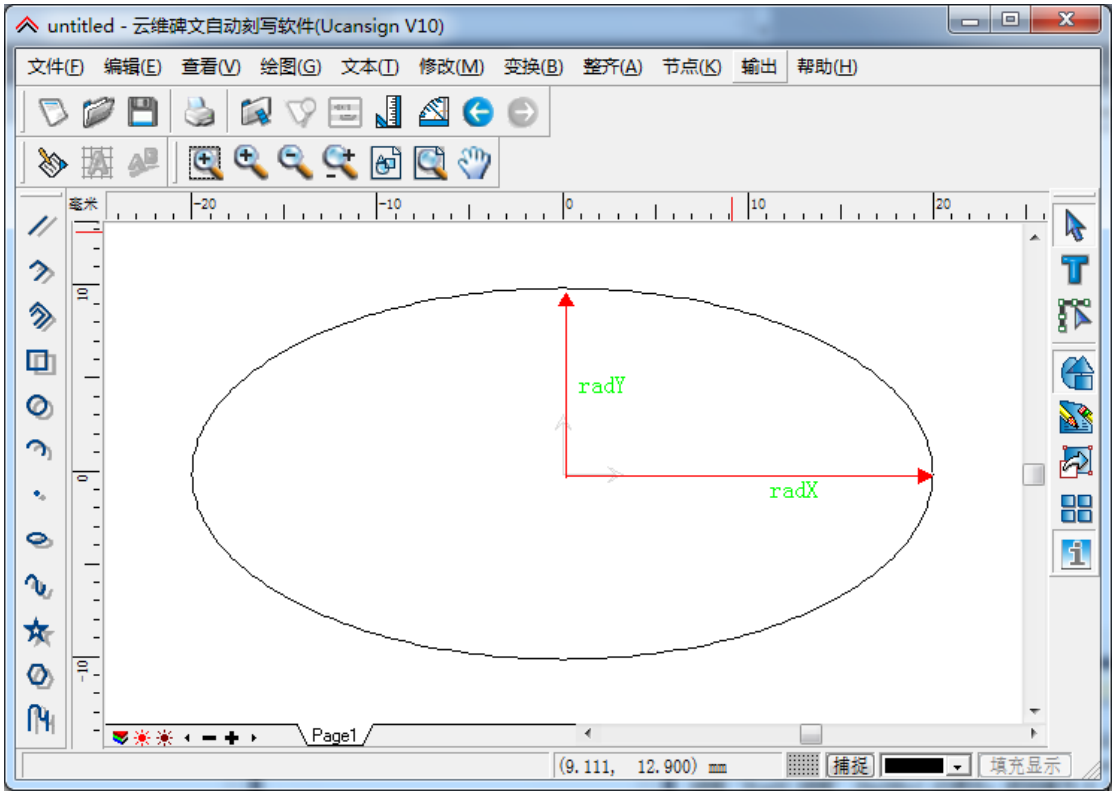
根据椭圆中心点 center ，长轴radX，虚轴radY,角度angX,以及绘制圆的方向（nDir=0时顺时针绘制，nDir=1时逆时针绘制）生成椭圆。

使用参数: 点 center，浮点类型radX, 浮点类型 radY, 浮点类型angleX 以及int nDir

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
radX=20
radY=10
angX=0
nDir=0
shape=Shape2()
Ellipse=shape.CreateEllipse(p1,radX,radY,angX,nDir)
shape.AddCurveToSystem(Ellipse)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义一个点p1（椭圆中心）和radX(椭圆长轴)和radY(椭圆短轴)和angX(椭圆与X轴的倾斜角度)和nDir（绘制方向）和一个塑造类型shape,然后用调用shape中绘制椭圆的命令生成Ellipse,最后将Ellipse添加到系统中并显示出来。



CreateBezier(PointList list)

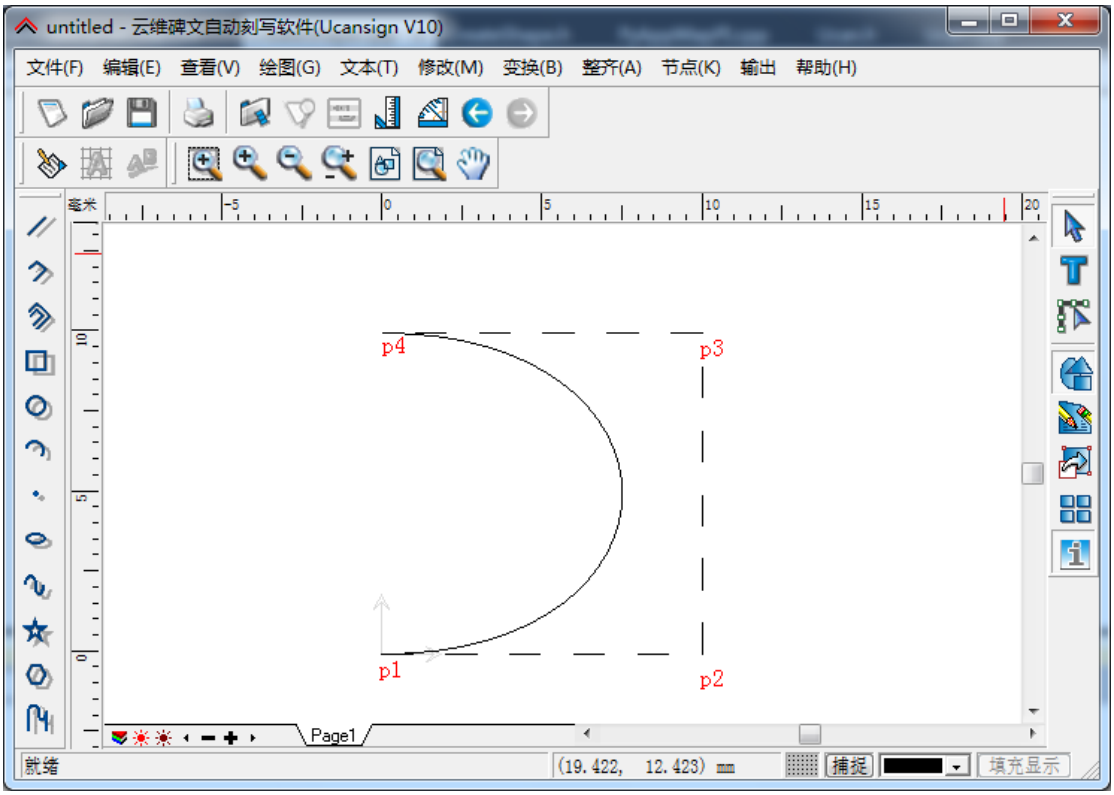
根据给定的点集合list 生成贝塞尔曲线

使用参数: 点集合 list

使用方法:

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(10,0,0)
p3=Point(10,10,0)
p4=Point(0,10,0)
lst=PointList()
lst.append(p1)
lst.append(p2)
lst.append(p3)
lst.append(p4)
shape=Shape2()
Bezier=shape.CreateBezier(lst)
shape.AddCurveToSystem(Bezier)
```

在本示例中，from Ucan import * 首先导入Ucan.Pyd，再定义四个点p1，p2，p3，p4和一个点集合lst和一个塑造类型shape,然后调用点集合lst中append(Point3 p)方法向点集合添加四个点，调用shape中绘制贝塞尔曲线的命令生成Bezier,最后将Bezier添加到系统中并显示出来。



CreateCombCurve(CurveList list)

根据曲线集合CurveList listz中保存的**首尾相接**的曲线生成一个组合曲线。

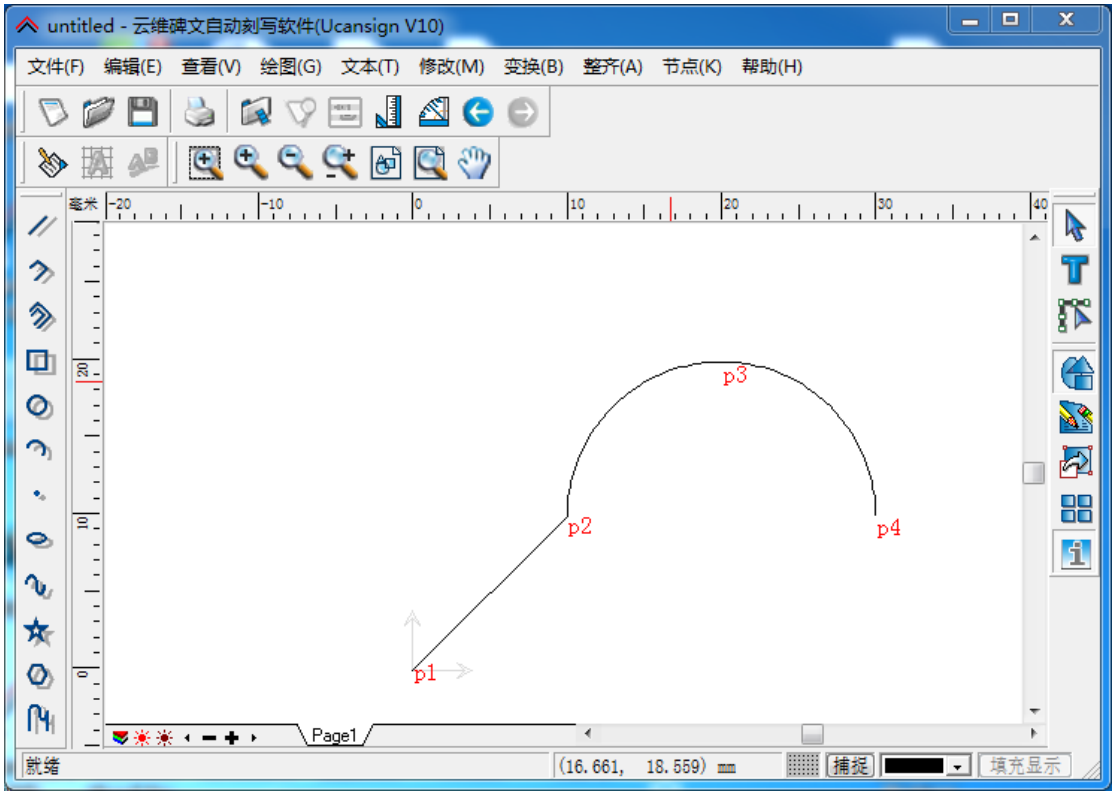
使用参数：曲线集合list

使用方法：

```
from Ucan import *
p1=Point(0,0,0)
p2=Point(10,10,0)
p3=Point(20,20,0)
p4=Point(30,10,0)
CurveList=CurveList()
shape=Shape2()
Line=shape.CreateLine(p1,p2)
Arc=shape.CreateArc3Pt(p2,p3,p4)
CurveList.append(Line)
CurveList.append(Arc)
CombCurve=shape.CreateCombCurve(CurveList)
shape.AddCurveToSystem(CombCurve)
```

本示例中，分别创建四个三维点p1，p2，p3，p4与一个曲线集合CurveList和一个二维塑造类型shape，然后调用塑造类shape中绘制直线与绘制三点弧的命令生成直线与三点弧，再使用曲线集合中的append(Curve)方法命令来添加直线与三点弧线，最后调用塑造类shape中

绘制组合曲线并将曲线添加到系统中显示出来。



AddCurveToSystem(CGeCurve* pGe)

将生成的曲线类型对象添加到UCancam软件中显示。

使用参数： CGeCurve * pGe