

《计算机图形学》

实 验 指 导 书

郝润平 编制

西北工业大学计算机学院

二〇一一年四月 修订

说 明

依照我院《计算机图形学》教学大纲规定及课堂教学内容，参阅其他院校本课程教学与实验课内容，并结合学生和实验室状况，特编制本实验指引书供学生在上机实验学时参照使用。

本实验大纲择选与课程教学较为密切若干实验，每个实验分别给出了参照机时数，实验时可依照详细机时安排状况选做。在附录中简朴地简介了 Open GL 和 DirectX 3D 两个三维图形开发平台基本概念和有关重要函数，供感兴趣同窗入门学习参照使用。

XRP 于.3.15

补充了 MFC 中几种惯用 DC 使用阐明等；(附录二 ~~13~~14)

补充了 VC 下 OpenGL 编程框架细节。(附录三)

于.4.24

补充 MFC 菜单使用；(附录二 10)

补充了对话框使用；(附录二 11)

鼠标拾取点示例。(附录二 12)

于.4.26

补充了工具条使用 (附录二 13)

于.3.14

增长三(二)9 修改 OnEraseBkgnd()函数，禁止重绘背景，避免动画时闪烁。

于 Sydney .1.18

目 录

实验一 直线生成算法	1
实验二 圆弧生成算法	2
实验三 窗口菜单交互界面	2
实验四 区域填充算法	3
实验五 裁剪算法	3
实验六 图形软件开发包/库应用	3
实验七 曲线生成算法	4
实验八 曲面生成算法	4
实验九 二维图形变换	5
实验十 三维图形变换	5
实验十一 真实感图形	6
附 录	7
一、C 语言中图形函数及其用法	7
二、VC 集成开发环境下基于 MFC 绘图	16
三、OpenGL 绘图	30
四、DirectX 3D 绘图	35
五、实验报告模板	39
六、《计算机图形学》上机实验评分原则	40

实验一 直线生成算法

一、实验教学目的与基本规定

1. 理解光栅图形显示屏工作原理和特点；
2. 学习 C/VC 环境下基本绘图办法；
3. 实践与巩固直线基本生成算法。
4. 掌握直线扫描转换算法原理及实现；

二、实验课程内容 (2 学时)

基于光栅图形显示屏，在 C 环境中生成不同粗细和线型任意直线。

1. 理解光栅图形显示屏特点；
2. 熟悉 C 环境下图形程序绘图办法；
3. 实践 DDA、中点及 Bresenham 基本算法，并至少用两种基本算法生成任意直线；

i. 任意斜率:

$-1 \leq k < 0$, $(50,400) \rightarrow (500,50)$;

$0 \leq k \leq 1$, 如 $(50,50) \rightarrow (300,200)$;

$1 < |k| < \infty$, 如 $(50,50) \rightarrow (300,400)$;

$k = \infty$, 如 $(200,50) \rightarrow (200,400)$ 。

ii. 任意起始点: 如 $(200,500) \rightarrow (100,200)$, $(300,200) \rightarrow (50,50)$

4. 改进办法，实现对直线实现线宽和线型(虚线、点划线等)控制。

三、实验参照

关于 C 环境下绘图简介请参见附录一，如下实例为绘制一条直线（TC 编译环境），供上机实验时参照使用。

```
#include "graphics.h"
```

```

#include "string.h"

main()
{
    int x0=50,y0=50,
    x1=450,y1=450,color=3;           //定义点(50,50), (450,450)和颜色 color
    int x,y;
    int gdriver=DETECT, gmode;       //定义图形驱动程序和显示模式
    initgraph(&gdriver,&gmode, ""); //图形方式初始化
    cleardevice();                   //清屏
    y=y0;
    for(x=x0;x<=x1;x++)
    {
        putpixel(x,y,color);        //用 colo 颜色绘制点(x,y)
        y=y+1;
    }
    getch();                         //让画面停住, 等按一下键盘再继续
    closegraph();                    //关闭图形方式
    return(0);
}

```

注：VC 集成开发环境下基于 MFC 绘图可参照附录二

实验二 圆弧生成算法

一、实验教学目的与基本规定

1. 学习圆(弧)基本生成算法;
2. 理解光栅图形显示屏工作原理和特点;
3. 实践圆(弧)基本生成算法;

4. 掌握圆弧扫描转换算法原理及实现;
5. 理解反走样技术。

二、实验课程内容 (2 学时)

基于光栅图形显示屏, 在 C 环境中生成任意圆弧, 并实践反走样技术。

1. 巩固 C 环境下绘图办法;
2. 用中点和 Bresenham 算法生成任意位置圆(弧);
3. 改进办法, 实现线宽和线型控制;
4. 运用反走样技术改进直线和圆弧生成算法。

实验三 窗口菜单交互界面

一、实验教学目的与基本规定

1. 理解窗口系统关于概念, 学习有关知识;
2. 理解和学习菜单有关与知识;
3. 理解交互重要意义。

二、实验课程内容 (3 学时)

MS-Windows 环境下窗口、菜单交互界面实现。

三、实验参照

Win32 编程方式参照教材 P₁₀₈₋₁₂₈ 关于内容, MFC 编程方式可参照附录二 10-12。

实验四 区域填充算法

一、实验教学目的与基本规定

1. 掌握区域填充基本算法原理；
2. 会使用字符图形输出。

二、实验课程内容 (4 学时)

1. 多边形扫描线、边标志及扫描线种子填充算法(至少实现两个算法)；
2. 在屏幕上输出矢量或点阵字符。

三、实验参照

参照教材关于内容。

实验五 裁剪算法算法

一、实验教学目的与基本规定

熟悉裁剪算法基本原理。

二、实验课程内容 (2 学时)

实现直线 Cohen-Sutherland、中点分割和参数化裁剪算法(至少实现两种裁剪算法)。

三、实验参照

参照教材关于内容。

实验六 图形软件开发包/库应用

一、实验教学目的与基本规定

1. 会使用一种图形软件开发包；
2. 学习窗口下图形界面设计；
3. 能使用图形软件开发包绘制简朴图形/形体。(为实验十一做准备)

二、实验课程内容 (3 学时)

学习并实践使用图形软件开发包(Open GL 或 DirectX 3D 等)绘图。

1. 熟悉图形开发平台程序设计过程；
2. 使用图形库绘制简朴图形/形体。

三、实验参照

关于 Open GL 或 DirectX 3D 绘图简介请参见附录三、四。

实验七 曲线生成算法

一、实验教学目的与基本规定

1. 掌握曲线生成基本算法原理；
2. 能实现曲线生成；
3. 掌握课本所简介图形算法原理和实现。

二、实验课程内容 (4 学时)

实现 Bezier 曲线和 B 样条曲线生成算法。

1. Bezier 曲线生成算法实现(参见教材 P304~306):

三次 Bezier 曲线及 de Casteljau 算法。

2. B 样条曲线生成算法实现(参见教材 P314~316):

B 样条曲线分割, 节点插入算法(选做)。

三、实验参照

参照教材关于内容。

实验八 曲面生成算法

一、实验教学目的与基本规定

1. 学习曲面生成、隐藏基本算法;
2. 理解 z 缓冲器算法应用;
3. 掌握课本所简介图形算法原理和实现。

二、实验课程内容 (4 学时)

曲面隐藏线消除和 z 缓冲器应用。

1. 曲面隐藏线消除算法;
2. 用 z 缓冲器算法绘出一种立方体真实感图形。

三、实验参照

参照教材关于内容。

实验九 二维图形变换

一、实验教学目的与基本规定

- 1.掌握图形变换基本算法原理；
- 2.实现若干典型二维图形变换算法。

二、实验课程内容 (4 学时)

- 1.生成前几次实验中基本图形；
- 2.对生成基本图形进行平移、旋转、放缩、对称等变换。

实验十 三维图形变换

一、实验教学目的与基本规定

1. 实习三维图形坐标系之间变换；
2. 三维图形几何变换；
3. 掌握三维图形坐标系之间变换算法及三维图形几何变换原理和实现。

二、实验课程内容 (4 学时)

实现三维图形坐标系之间变换（世界坐标、物坐标、屏幕坐标）以及三维图形几何变换。

三、实验参照

参照教材关于内容。

实验十一 真实感图形

一、实验教学目的与基本规定

初步实现真实感图形，并实践图形造型与变换等。

二、实验课程内容(6 学时)

运用几何造型，几何、投影及透视变换、真实感图形效果（消隐、纹理、光照等）关于知识实现。

1. 用给定地形高程数据绘制出地形图；
2. 绘制一(套)房间，参数自定。

三、实验参照

参照教材及附录三、四关于内容。

附 录

一、C 语言中图形函数及其用法

Turbo C 2.0/3.0具备丰富图形功能，它提供了70各种图形函数。在这里只简介最惯用一某些，别的图形函数及用法可参阅有关书籍。图形函数均在头文献"graphics.h"中定义，因此在程序中调用这些图形函数时，必要在程序文献开头写上文献包括命令：`#include "graphics.h"`。

1. 图形系统管理

PC机显示屏有两种工作模式，一种是图形模式，此外一种是文本模式（缺省模式）。在文本方式下，屏幕分为80列、25行，在该方式下，图形函数不能对的工作。因而在使用图形函数绘图之前，必要将屏幕显示适配器设立为图形模式，这就是普通所说"图形方式初始化"。在绘图工作完毕之后，又要使屏幕回到文本方式，以便进行文本方式下工作。

1) 图形方式初始化

图形方式初始化是通过函数`initgraph()`来完毕。其调用格式为：

```
Initgraph (*gdriver,*gmode,*path);
```

函数`initgraph()`功能是通过从磁盘上装入一种图形驱动程序来初始化图形系统，并将系统设立为图形方式。调用该函数必要用三个参数，其含义为：

gdriver是一种整型值，用来指定要装入图形驱动程序，如果给其赋值为`DETECT`（该值在头文献"graphics.h"已经中定义），则系统自动检测图形适配器最高辨别率模式，并装入相应图形驱动程序。

gmode是一种整型值，用来设立图形显示模式。不同图形驱动程序有不同图形显示模式，虽然是在同一种图形驱动程序下，也也许会有几种图形显示模式。图形显示模式决定了显示辨别率、可同步显示颜色多少、调色板设立方式以及存储图形一页数。

path是一种字符串，用来指明图形驱动程序所在途径。如果驱动程序就在顾客当前目录下，则该参数可觉得空字符串，否则应给出详细途径名。

以上简介了initgraph 函数中三个参数含义。注意，前两个参数事实上是整形指针，调用时应加上地址运算符"&"。例：

```
int gdriver=DETECT,gmode;  
  
initgraph(&gdriver,&gmode, "");
```

使用DETECT模式，由系统自动对硬件进行检测，并把图形显示模式设立为检测到驱动程序最高辨别率。

2) 关闭图形方式

在运营图形程序绘图结束后，要回到文本方式，以进行其他工作，这时应关闭图形方式。关闭图形方式要用函数closegraph()。其调用格式为：

```
closegraph();
```

函数closegraph ()作用是：释放所有图形系统分派存储区，恢复到调用initgraph()之前状态。函数closegraph()不需要参数。

2. 屏幕管理

Turbo C 2.0提供了11个函数用于对屏幕和视图区等进行控制管理。

1) 设立视图区

在图形模式下，可以用函数setviewport ()在屏幕上定义一种视图区（视图区相称于一种用于绘图窗口）。视图区位置用屏幕绝对坐标定义，并且可以把视图区设立为剪裁和不剪裁两种状态。函数setviewport ()调用格式为：

```
setviewport(x1,y1,x2,y2,c);
```

函数调用中五个参数均为整型，其中：

x1,y1:为视图区左上角坐标。X2,y2:为视图区右下角坐标。

C:为裁剪状态参数。当c=1时，则超过视图区图形某些被自动裁剪掉；当c=0 时，则对超过视图区图形不作裁剪解决。

注意 视图区建立后来，所有图形输出坐标都是相对于当前视图区，即视图区左上角为坐标

(0,0)点，而与图形在屏幕上位置无关。

2) 清除视图区

清除视图区可以使用函数`clearviewport()`。它作用是清除掉当前视图区，将当前点位置设立于屏幕左上角(0,0)点。执行后，原先设立视图区将不复存在。函数调用格式为：

```
clearviewport();
```

3) 清屏

清除屏幕使用函数`cleardevice()`。它作用是及时清除全屏幕内容，并将当前点位置设立为原点(0, 0)。但是其他图形系统设立保持不变，如线型，充填模式等；如果设立了视图区，则视图区设立不变，涉及当前点位置设立在视图区左上角。清屏函数调用格式为：

```
cleardevice();
```

3. 绘图函数

绘图函数是进行图形操作基本。用像素点几乎可以画出任何图形，但效率太低。为此TC提供了大量基本绘图函数，以以便图形设计。

在使用绘图函数时，要随时注意画图"当前点位置"，它是绘图起始位置。也就是说，图形总是从当前点开始画。画完一种图形后，有时当前点位置不变，仍在本来位置；有时则要把当前点移到新位置。此外，为了从指定位置开始作图，有时需要先变化当前点位置，然后再作图。在调用绘图函数时候要注意这些问题。

1) 直线类函数

用直线类函数绘制直线图形，可以用两种坐标：一种是绝对坐标；另一种是相对坐标。

Line() 函数 用`line()`函数可以在指定两点之间画一条直线段。其调用格式为：

```
line(x1,y1,x2,y2);
```

参数`x1,y1,x2,y2`均为整型，使用绝对坐标。其中`(x1,y1)`和`(x2,y2)`分别为直线两个端点坐标。用`line`函数画线时，其当前点位置不变。

例如：已知三角形两个顶点坐标分别为：`(x1,y1)`、`(x2,y2)`和`(x3,y3)`，

则用下面语句可以把该三点连成一种三角形：

```
line(x1,y1,x2,y2);
```

```
line(x2,y2,x3,y3);
```

```
line(x3,y3,x1,y1);
```

lineto () 函数

lineto () 函数用于从当前点位置到指定位置画一条直线，并变化当前点位置。在画线到指定点同步也把当前点位置移到了指定点（即直线终点）。其调用格式为：

```
lineto(x,y);
```

参数x,y为指定点坐标，均为整型。

moveto () 函数

函数moveto () 用于移动当前点位置，并不画线。其调用格式为：

```
moveto(x,y);
```

参数x,y用于指定新当前点位置坐标（用整型，使用绝对坐标）。调用成果是将当前点位置移到点（x,y）处。例如：

```
moveto(400,10);
```

成果是将当前点位置移到了（400，10）处。

moveto () 函数和lineto () 函数配合使用，可以在两点之间画直线。例如：

```
moveto(400,10);
```

```
lineto(240,360);
```

上面语句实现把当前点移到（400，10）处，然后从该点画线到（240，360）处。画线结束后，当前点位置在（240，360）处。

linereel () 函数

```
linereel
```

()函数用相对坐标画线。其功能是从当前点位置开始画线到指定点位置，该指定点位置坐标不是以绝对坐标形式给出，而是以其相对于当前点（即直线起点）位置坐标增量给出（相对坐标）。其调用格式为：

```
linere(dx,dy);
```

参数是相对于直线起点坐标增量。Linerel() 函数变化当前点位置到指定点处。

moverel () 函数

moverel () 函数功能与moveto () 函数相似，但它使用是相对坐标，它使当前点位置在x和y方向上分别移动一种增量。其调用格式为：

```
moverel(dx,dy);
```

参数dx,dy为整型，是相对于当前点位置增量。

下面举几种实际绘图例子，来阐明上面这些函数在使用上差别：

例1： 用line函数过四点画一种矩形。

```
#include "graphics.h"

#include "stdio.h"

main()
{
    int gdriver=DETECT,,gmode;

    initgraph(&gdriver,&gmode, "");

    cleardevice();

    line(100,100,100,400);

    line(100,400,300,400);

    line(300,400,300,100);

    line(300,100,100,100);

    getch();

    closegraph();
```

}

2) 圆弧类函数

circle () 函数 函数**circle ()**用于以指定圆心和半径方式画圆。其调用格式为

circle(x,y,r);

参数x,y,r均为整型。其中x,y为指定圆心坐标，r为圆半径。例如：

circle(300,200,100); 调用成果是：以点（300,200）为圆心，以100为半径画一种整圆。

arc () 函数 **arc()**函数用于画圆弧。其调用格式为:

arc(x,y,angs,ange,r);

函数调用时所需五个参数均为整型。其中：

x,y为圆弧所在圆圆心坐标。**angs**，**ange**分别为圆弧起始角和终结角（单位为度），r为圆弧半径。例如：

arc(300,200,90,180,200)成果是以点（300,200）为圆心，200为半径，从90度到180度画了四分之一圆弧。当圆弧起始角**angs=0**，终结角**angs=360**时，则可以画出一种整圆。

ellipse () 函数 函数**ellipes()**用于画椭圆弧或椭圆。其调用格式为：

ellipse(x,y,angs,ange,xr,yr);

函数参数均为整型。其中：

x,y为椭圆中心坐标，**angs**，**ange**分别为椭圆弧起始角和终结角（单位为度），**xr**，**yr**分别为椭圆水平轴半轴和垂直轴半轴。如果**angs=0**，**ange=360**，则可以画出一种完整椭圆。此外：

xr>yr：则画出长轴为水平方向椭圆或椭圆弧；

xr<yr：则画出长轴为垂直方向椭圆或椭圆弧；

xr=yr：则可以画出圆或圆弧。

例2：调用函数**ellipse**画出一种椭圆群。

```
#include "graphics.h"
```

```
#include "stdio.h"
```

```
main()
```

```
{
```

```
int
```

```

        a=150,i;

int gdriver=EDTECT,gmode;

initgraph(&gdriver,&gmode, "");

cleardevice()

for(i=10;i<=140;i+=10);

ellips(320,240,0,360,a-i,i);

getch();

closegraph();

}

```

3) 多边形

rectangle () 函数 函数rectangle () 用于绘制矩形。其调用格式为:

```
rectangle(x1,y1,x2,y2);
```

参数x1,y1,x2,y2均为整型。函数功能是以点(x1,y1)为矩形左上角点, 以点 (x2,y2) 为矩形右下角点, 画一种矩形。

drawpoly () 函数 函数drawpoly () 可用于画一条多边折线或者一种多边形。其调用格式为:

```
drawpoly(nps,*pxy);
```

它有两个参数。第一种参数nps是一种整型数据, 它表时所画多边折线顶点数, 第二个参数pxy是一种整型数组数组名, 该数组中存储了nps 个顶点坐标值序列。例如, 有一种名为d_poly数组中存储了 4 个顶点坐标为 [x1,y1,x2,y2,x3,y3,x4,y4], 则调用方式为: drawpoly(4,d_poly);

如果最后一种点坐标与第一种点坐标相似, 则运用drawpoly () 函数便可以画出一种封闭多边形。此时, 坐标点数目应当是多边形顶点数加 1 , 即让最后一种顶点和第一种点重叠。

例如, 有一种名为d_poly数组中存储了6个顶点坐标为 [x1,y1, x2,y2, x3,y3, x4,y4, x5,y5, x1,y1], 则调用方式为: drawpoly(6,d_poly); 便可以画出一种具备5个顶点封闭多边形。

4. 图形属性控制

图形属性控制涉及控制颜色和线型。颜色有背景色和前景色之分。背景色指屏幕颜色

（即绘图时底色），前景色指绘图时图形线条所用颜色。

任何绘图函数都是在当前颜色（涉及背景色和前景色）和线型状态下进行绘图。前面所举例子中没有提当前颜色和线型，是由于用了系统缺省值（系统缺省值是：背景色为黑色，前景色为白色，线型为实线）。如果绘图时要使用系统缺省值以外颜色和线型，则可以运用图形属性控制函数另行设立。

`setbkcolor()` 函数 函数 `setbkcolor()` 用于设立绘图时背景色。调用格式为：

```
setbkcolor(color);
```

参数 `color` 为一种整型数据，代表所取颜色，可以用整型常数（0~15），也可以用符号常数。

例如，要把背景色设立成浅蓝，可以如下调用：`setbkcolor(9);` 9表达浅蓝。

`setcolor()` 函数 函数 `setcolor()` 用于设立前景颜色，即绘图用颜色。调用格式为：

```
setcolor(color);
```

参数 `color` 含义和用法同 `setbkcolor()` 函数。

下面例程序可以演示 16 种颜色。

例3：显示16种颜色

```
#include "graphics.h"

#include "stdio.h"

main()

{

    int cb,cf;

    int gdriver=DETECT,gmode;

    initgraph(&drive,&gmode,"");

    cleardevice();

    for(cb=0;cb<=15;cb++)

    {
```

```

        setbkcolor(cb);

        for(cf=0;cf<=15;cf++)
        {

            setcolor(cf);

            circle(100+cf*25,240,100);

        }

        getch();

    }

    getch();

    closegraph();

}

```

setlinestyle () 函数 函数setlinestyle () 用于设立当前绘图所用线型和宽度，这些设立限于对直线类图形有效。其调用格式如下：

```
setlinestyle(sty,pat,b);
```

该函数所用三个函数含义如下：

sty为整型值，用来定义所画直线类型（0~4）；

pat为无符号整型数，该参数在需要顾客自定义线型时使用，如果是使用系统定义线型，则该参数可取 0 值；

b为整型数，指定所画直线粗细，以像素为单位（1或3）当函数setlinestyle第一种参数为USERBIT-LINE(或 4)时，可以由顾客自己定义直线类型。

例4：编写程序，显示系统预定义四种线型。

```

#include "graphics.h"

#include "stdiio.h"

```

```
main()
```

```
{
```

```

int i,j,c,x=50,y=50,k=1;

int gdrive=DETECT,gmode;

printf("input color number.\n")

scanf("%d",&c);

initgraph(&gdriver,&gmode, "");

cleardevice();

setbkcolor(11);

setcolor(c);

for(j=1;j<=2;j++)

{

for(i=0;i<4;i++)

{

setlinestyle(i,0,k);

rectangle(x,y,x+210,y+80);

x=x+110;

y=y+40;

}

k=3;

x=50;

y=250;

}

getch();

closegraph();

}

```

5. 充填

setfillstyle () 函数 函数setfillstyle () 用来设立当前充填模式和充填颜色，

以便用于充填一种指定封闭区域。其调用格式为：

```
setfillstyle(pattern,color);
```

参数pattern用于指定充填模式（取值： 0 ~ 1 1 ），第二个参数color指定充填用颜色。

Floodfill（）函数 函数floodfill（）实行对一指定区域进行充填操作，其充填模式和颜色已由setfillstyle（）函数指定。其调用格式为：

```
floodfill(x,y,bcolor);
```

参数x,y指位于充填区域内任意一点坐标，该点作为充填起始点。bcolor为充填区域边界颜色。

例5：下面例程序演示充填状况。

```
#include "graphics.h"

#include "stdio.h"

main()

{

int i,c,x=5,y=6;

int gdriver=DETECT,gmode;

printf("input color
```

```

    number.\n");

scanf("%d",&c);

initgraph(&gdriver,&gmode, "");

cleardevice();

setbkcolor(9);

for(i=c;i<c+8;i++)

{

setcolor(i);

rectangle(x,y,x+140,y+104);

x=x+70;

y=y+52;

setfillstyle(1,i);

floodfill(x,y,i);

}

getch();

closegraph();

}

```

6. 图形方式下文本输出

settextstyle(font,direction,csize)函数。设立字符显示风格。函数三个参数含义如下

font: 是一种整型数据，用于指明所要使用文字字体（0表达8*8位图字体；1表达三重矢量字体；2表达小号矢量字体；3表达无衬线矢量字体；4表达哥特矢量字体）。

direction: 是一种整型数据，表达指定文本输出方向（0表达从左向右输出；1表达从上向下输出）。

csize: 是一种整型数据，表达字符大小（实际是一种放大系数，表达对8*8点阵字符放大倍数，取值范畴是1~10）。

outtext(char *text); 在当前位置输出一种文本字符串。

outtextxy(int x,int y,char *text); 在x,y位置输出一种文本字符串。

综合举例

例6：渔网图案绘制

```
#include "graphics.h"

#include "stdio.h"

main()
{

    int x,y,x1,y1,x0=320,y0=50;

    int i,j,n=5,r=20;

    int
```

```

gdriver=DETECT,gmode;

initgraph(&gdriver,&gmode, "");

cleardevice();

setbkcolor(9);

for(i=0;i<2*n;i++)

{

    x1=x0-i*r;

    y1=y0+i*r;

    for(j=0;j<=n-1;j++)

    {

        x=x1+2*j*r;y=y1+2*j*r;

        arc(x,y,180,270,r);

        arc(x,y+2*r,0,90,r);

    }

}

x1=x0-2*r;

y1=y0;

for(i=0;i<2*n;i++)

{

    x1=x1+r;

    y1=y1+r;

    for(j=0;j<=n-1;j++)

    {

        x=x1-2*j*r;y=y1+2*j*r;

        arc(x,y,90,180,r);

```

$\text{arc}(x-2*r,y,270,360,r);$

```

    }

}

getch();

closegraph();

}

```

例7：图形模式下文本输出

```

#include "graphics.h"

#include "stdio.h"

#include "dos.h"

main()
{

    int i,t,x=300,y=50;

    int gdriver=DETECT,gmode;

    initgraph(&gdriver,&gmode, "");

    cleardevice();

    setbkcolor(9);

    setcolor(4);

    for(i=1;i<=10;i++)
    {

        x=x-15;

        y=y+15;

        settextstyle(1,0,i);

        cleardevice();

        outtextxy(x,y,"it is ok");

        delay(200); /* 时间延迟 */
    }
}

```

```
    }  
  
    getch();  
  
    closegraph();  
  
}
```

二、VC 集成开发环境下基于 MFC 绘图

0. MFC 简介

MFC (Microsoft Foundation Class Library)中各种类结合起来构成了一种应用程序框架，它目的就是让程序员在此基本上来建立 Windows 下应用程序，这是一种相对 SDK 来说更为简朴办法。由于总体上，MFC 框架定义了应用程序轮廓，并提供了顾客接口原则实现办法，程序员所要做就是通过预定义接口把详细应用程序特有东西填入这个轮廓。Microsoft Visual C++提供了相应工具来完毕这个工作：AppWizard 可以用来生成初步框架文献（代码和资源等）；资源编辑器用于协助直观地设计顾客接口；ClassWizard 用来协助添加代码到框架文献最后，编译，则通过类库实现了应用程序特定逻辑。

MFC 提供了一种 Windows 应用程序开发模式，对程序控制重要是由 MFC 框架完毕，并且 MFC 也完毕了大某些功能，预定义或实现了许多事件和消息解决，等等。框架或者由其自身解决事件，不依赖程序员代码；或者调用程序员代码来解决应用程序特定事件。

MFC 是 C++类库，程序员就是通过使用、继承和扩展恰当类来实现特定目。例如，继承时，应用程序特定事件由程序员派生类来解决，不感兴趣由基类解决。实现这种功能基本是 C++对继承支持，对虚拟函数支持，以及 MFC 实现消息映射机制。

1. 运用 VC 工程向导建立一种工程

工程名可觉得班号+学号+姓名+实验序号，如 8_04888_张三_1，这里以 test 为示例，如图 1 所示。

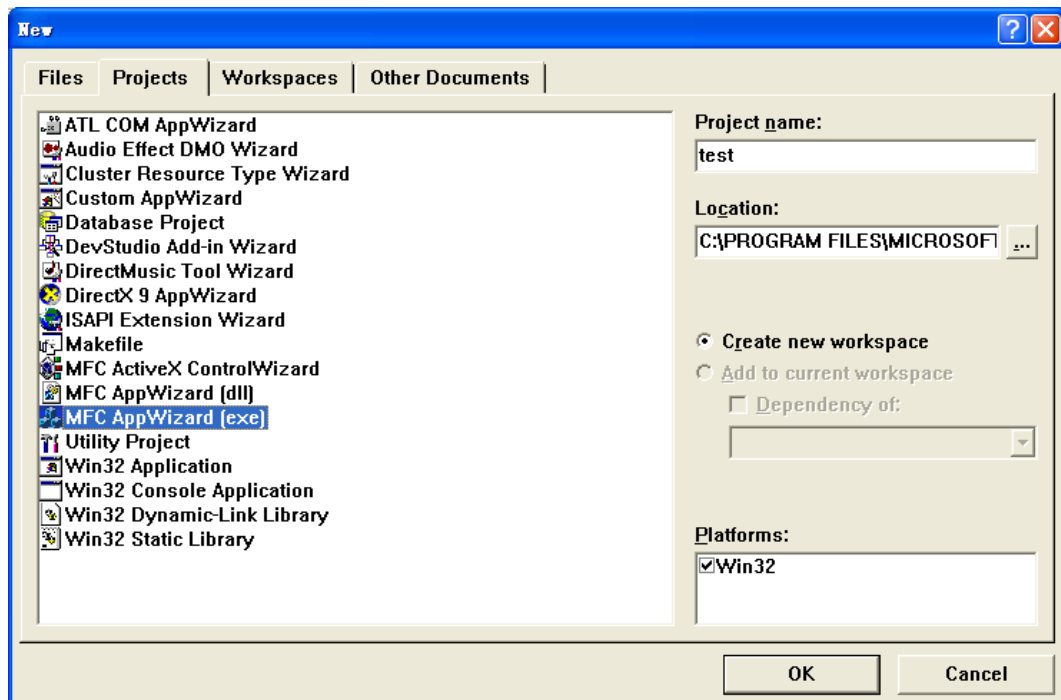


图 1 建立工程

2. 点击“OK”拟定后，如图 2 所示

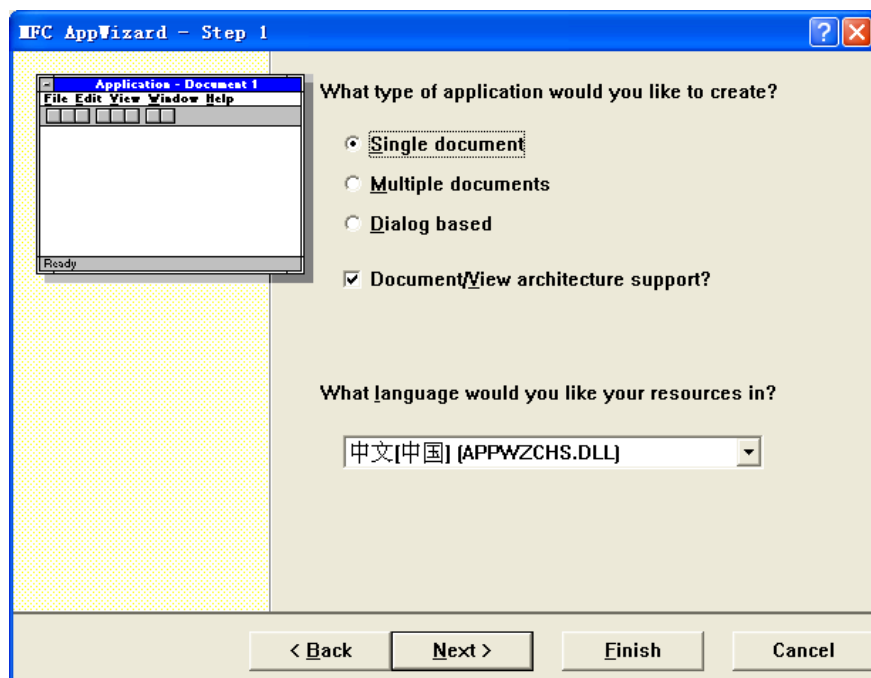


图 2 选取单文档

3. 选取“Single document”后，点击“Finish”，如图 3 所示

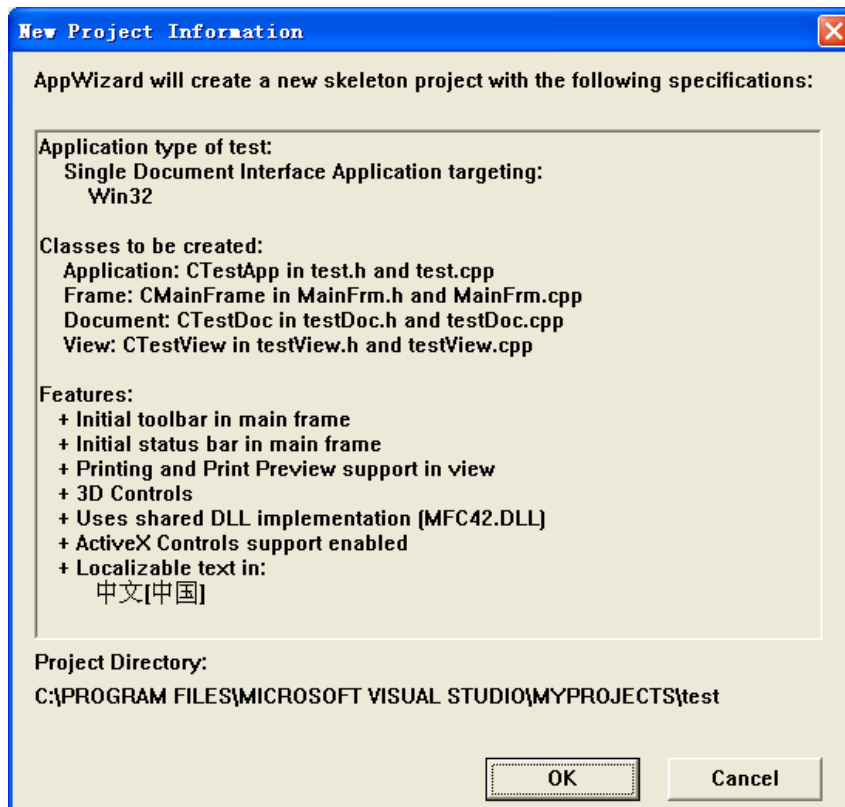


图 3 确认

4. 点击“OK”拟定后，在左则选取“ClassView”项，点击开“test class”前“+”，再点击“CTestView”类前“+”，双击“OnDraw(CDC* pDC)””，如图 4 所示

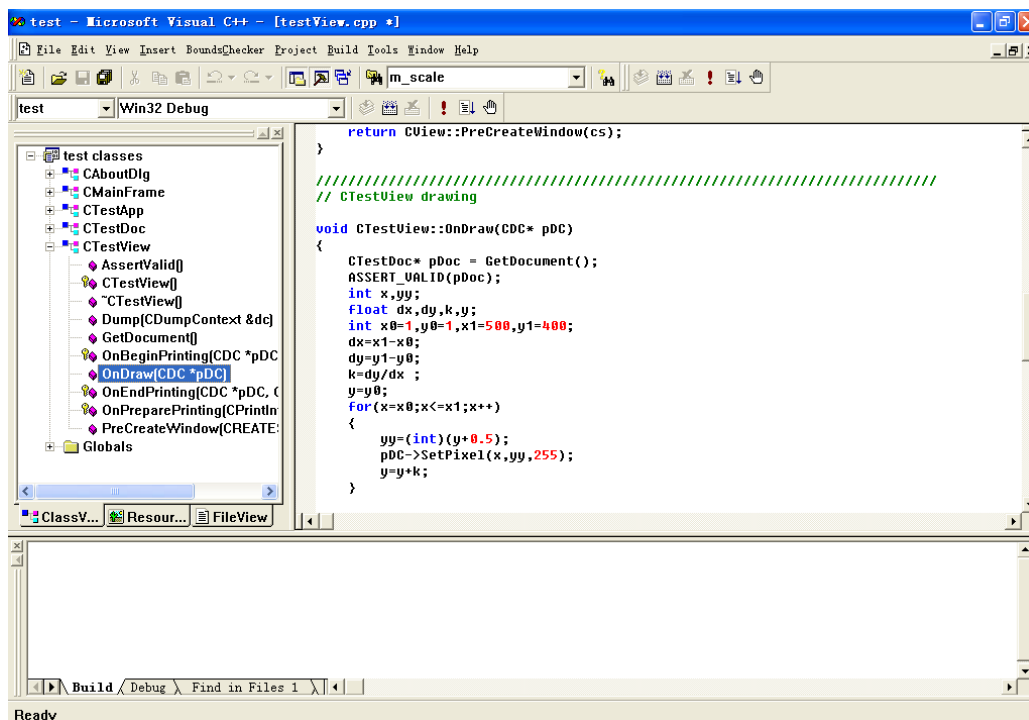


图 4 编写代码

5. 在 OnDraw()函数中使用 pDC 对象进行绘图，如

pDC->SetPixel(x,yy,255); 设立(x,yy)像素点颜色为 255。

可在其中加入自己代码，如 DDA 程序如下：

```
int x,yy;

float dx,dy,k,y;

int x0=1,y0=1,x1=500,y1=400;

COLORREF color=RGB(100,25,108); //定义一种 RGB 颜色常量

dx=x1-x0;

dy=y1-y0;

k=dy/dx ;

y=y0;

for(x=x0;x<=x1;x++)

{

    yy=(int)(y+0.5);

    pDC->SetPixel(x,yy, color255);

    y=y+k;

}
```

如图 4 所示。

此外，也可以在这里调用自己声明定义好函数，详见 7-9。

6. 点击菜单 “Build”中 “Build All”编译程序，如图 5 所示，无错误后可执行(Ctrl+F5)。

8. 返回 TestView.CPP 文献定义函数

```
void CTestView::DDALine(int x1,x0,int y1y0, int x2x0,int y2y0, COLORREFint color)

{

    加入自己程序代码。

}
```

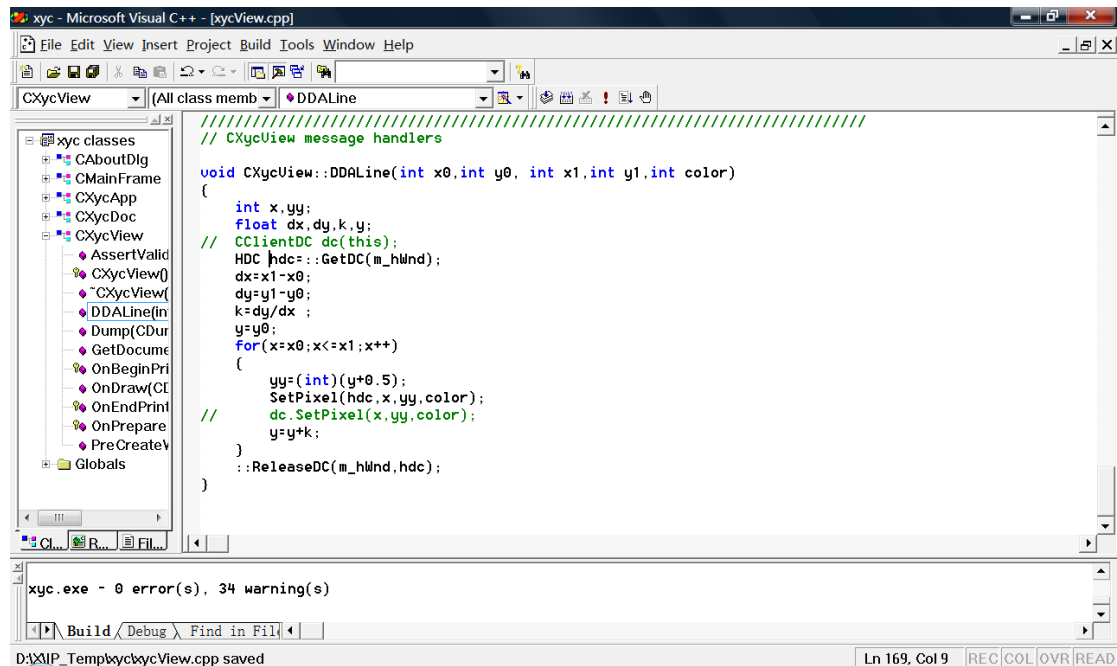


图 7 自定义函数

9. 在 OnDraw(CDC* pDC)中调用定义好函数，如 DDALine(100,100,400,500,200);

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。

如要下载或阅读全文，请访问：

<https://d.book118.com/588003001102006075>