Graph2D 使用说明

——一个面向 C++教学的简易图形库

胡少军，耿楠，蔡骋，魏蕾，耿耀君

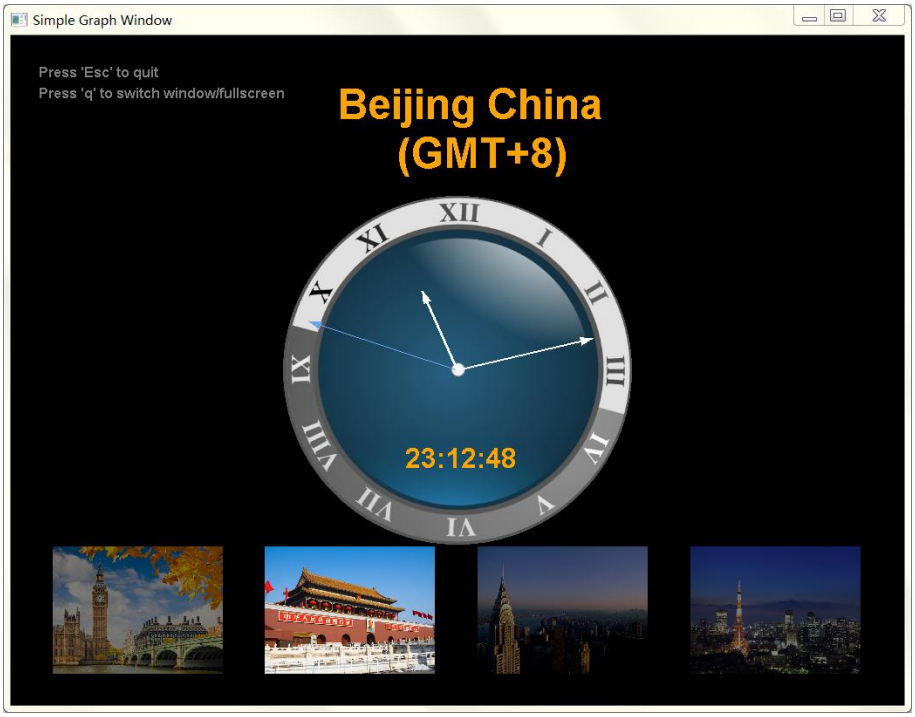

图 1：基于 Graph2D 开发的世界时钟界面

1. 简介

“面向对象”是软件设计中的一种重要思想，该思想的引入可实现将现实世界的解空间映射到软件系统的解空间，**提高代码重用性和软件开发效率**。C++作为一种面向对象编程语言，由于其**简洁、高效**的特点，在软件开发领域应用广泛，**适合描述面向对象程序设计**思想，也是许多高校计算机专业开设的核心课程。在 C++描述的面向对象程序设计的教学实践中，面向对象中的**抽象性、封装性、继承性与多态性是讲解重点**，相较于控制台程序中单纯的文本显示，**图形化显示有助于学生直观理解**面向对象程序设计中的**理论知识点**。例如，讲解类与对象时，可用圆类、矩形类、时钟类、俄罗斯方块类等进行举例；而在讲解继承与多态时，可基于形状类派生出多边形类、三角形类等。在 C++实现的**控制台程序中**，仅能从概念上对上述例子进行描述，学生无法直观看到具体绘制的图形。因此，如何在 C++描述的面向对象程序设计课程教学中**引入图形绘制与显示**是一个迫切问题。

当前，与 C++结合实现图形绘制与显示的途径有很多，如 MFC、Qt、wxWidgets、Windows API、OpenGL、DirectX 等，但考虑到开设这门课程之前，学生的先修课程许多仅为 C 语言等程序设计基础课，如果直接引入专业开发工具包或图形库，**易导致忽视面向对象程序设计教学的本质**。因此，我们选择图形绘制函数库主要有 3 个标准：

（1）图形模块能实现窗口图形绘制与显示，不能使用基于面向对象程序设计的图形库。如 MFC、Qt 和 wxWidgets 不适合 C++初学者，在掌握 C++描述的面向对象程序设计方法后再学习比较合适。

（2）图形模块简洁，程序入口为 main 函数而非 WinMain 函数，避免使用专用图形库，避免学生花费太多精力在理解图形库或 Windows 宏、函数的使用上。底层图形库如 OpenGL 或 DirectX 提供了丰富的二维和三维图形显示功能，但这些图形库的使用需预先了解相关计算机图形学知识，而 Windows API 中存在大量的宏、数据类型与函数，入口为 WinMain 函数，也不适合 C++初学者。

（3）支持当前高分辨率、真彩色显示模式，提供键盘、鼠标交互和双缓存功能，便于图形和动画的交互与显示。Borland C++中曾提供一个名为 graphics 的图形库，该图形库使用简捷，但仅支持 640×480 的 VGA 显示模式且显示颜色数有限，也不能直接提供图像、键盘、鼠标交互和双缓存功能，因此目前这一图形显示方式已被抛弃。

按照上述 3 个标准，提供图形用户界面，且能与 C++结合的商业化或专业化软件包、图形库，均存在**不适合 C++面向对象程序设计教学**的问题。鉴于此，在总结、分析上述软件包和图形库的基础上，课程组结合 OpenGL 和 graphics 库的特点，开发了一个**面向教学的 Graph2D 图形库**，该图形库**淡化了图形用户界面的设计**，不要求学生掌握图形学和图像处理的知识，**突出了基本图形、文字和图像的设置与显示**。图 1 是采用 Graph2D 图形库实现的一个简易世界时钟系统截图，该系统包含线、矩形、圆、文字、图像等基本图形的创建与显示，可动态显示不同时区当前时间，通过键盘可控制时钟的移动，通过鼠标可选择不同时区。由于采用 Graph2D 简易图形库，实现该系统仅需 167 行代码。

1.1 功能与结构图

Graph2D 参考 Borland C++中 graphics 库，通过名字空间(namespace)封装 OpenGL 中的图形初始化、基本图元绘制、键盘和鼠标交互等函数，为用户提供一个简易的开发界面与接口，图元绘制函数的命名多沿袭传统 graphics 库的命名。如图 2 结构图所示，Graph2D 包含图形初始化、窗口设置、基本图元绘制、字体创建与文本显示、图像读入与显示、键盘鼠标

交互等功能模块。

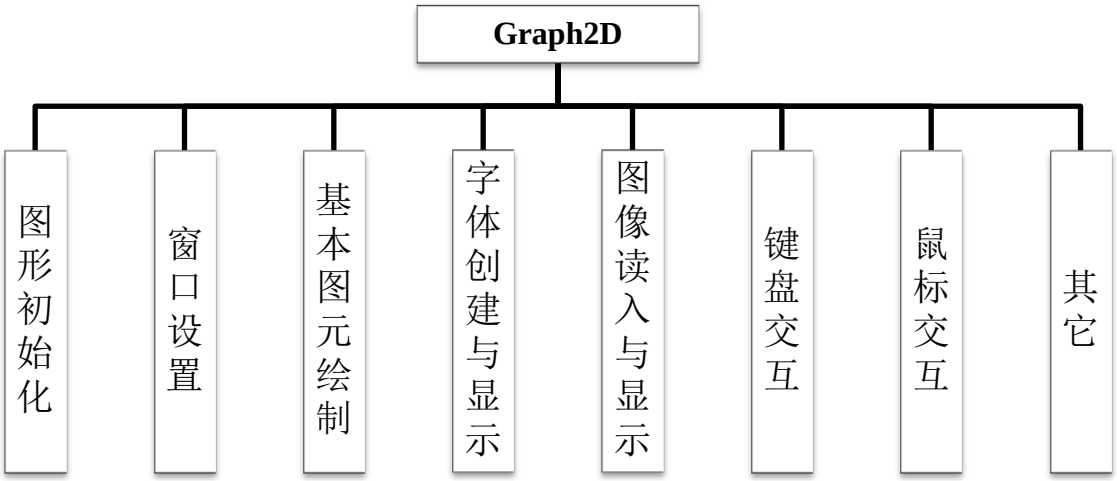

图 2：Graph2D 功能结构图

1.2 坐标系统和函数命名

坐标系统采用**二维笛卡尔右手坐标系**，其中 X 轴指向右方，Y 轴指向上方，图 3 是调用 Graph2D 中的 showCoordinate 函数显示的窗口坐标系，其中原点在左下位置。

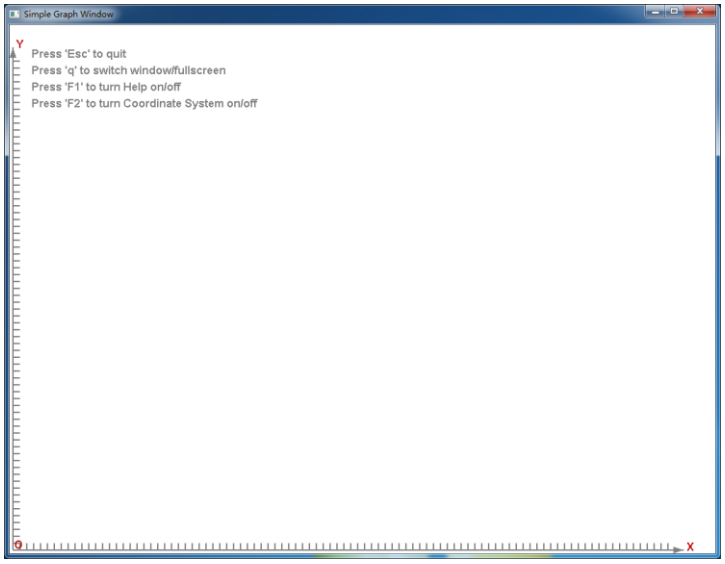

图 3：Graph2D 窗口坐标

函数命名采用**骆驼式命名法**，即当函数名由一个或多个单词连在一起时，第一个单词以小写字母开始，后面的单词首字母均为大写，如上述显示坐标系的 showCoordinate 函数。

1.3 图形库的配置与使用

Graph2D 库包含 graph2d.h, libgraph2d.a 和 graph2d.dll 三个文件。在 Code::Blocks 集成开

发环境中配置 Graph2D 图形库前需下载已编译过的 mingw 版的 freeglut 包，（www.transmissionzero.co.uk/software/freeglut-devel/），下载后将 freeglut 中的*.h 文件拷贝到.\mingw\include\GL 目录下，将 graph2d.h 拷贝到.\mingw\include 目录，libfreeglut.a 和 libgraph2d.a 拷贝至.\mingw\lib 目录，最后将 glut32.dll 和 graph2d.dll 拷贝到.\windows\system 或.\windows\system32 目录。为简化配置过程，课程组提供了一个安装程序可实现上述文件的自动拷贝。

完成文件拷贝后，运行 Code::Blocks，选择 C++控制台程序框架，根据向导生成一个典型的 C++程序“Hello world！”，修改代码如下：

```
01    #include <Graph2D.h>
02    using namespace graph;
03    void display() {
04        circle(512, 384, 100);
05        putText(480, 384, "Hello world!");
06    }
07    int main(int argc, char *argv[]) {
08        initGraph(display);
09        return 0;
10    }
```

代码 1：“Hello world!”图形绘制

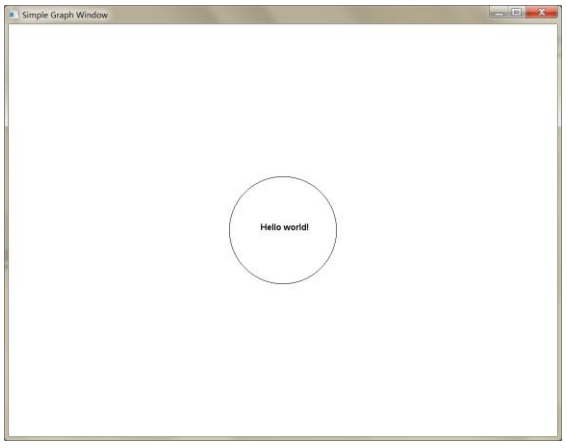

图 4：“Hello world!”图形窗口

如果直接运行程序将提示链接错误，在 Code::Blocks 中，还需显式指定图形程序链接库文件，在菜单 Settings->Compiler and debugger->Link settings 中添加 **libgdi32、libopengl32、libglu32、libfreeglut 和 libgraph2d**。

代码 1 编译运行后的结果如图 4 所示，与控制台程序框架类似，Graph2D 也使用了名字

空间和 main()函数，但最终能实现图形用户界面的显示，该界面中输出“Hello world!”文本和一个圆，**按 Esc 键可退出界面，按 F1 可显示帮助信息，按 F2 可显示坐标系，按‘q’键可在窗口和全屏间切换**。代码 1 中仅包含 10 条语句，结构清晰，与控制台程序中先包含 iostream，再使用名字空间 std，最后用 cout 输出“Hello world!”流程类似，适合初学者使用。

1.4 运行机制

代码 1 中的 initGraph 函数参数中包含了一个名为 display 的函数指针，即**回调函数**。回调函数不由用户直接调用，而是在特定事件或条件发生时（如键盘被按下、鼠标单击等），由操作系统调用并给该函数发送一个消息，用于对该事件或条件进行响应。除了 display 函数外，还可以继续在 initGraph 中添加键盘、鼠标交互等回调函数。Graph2D 中图形程序的运行机制可用流程图 5 表示，为了对各种消息进行监听，更新图形窗口，**display 等回调函数均处于一个循环体内**，除非收到退出窗口消息才停止调用回调函数。同时，为保证图形更新的流畅性，在 display 函数中使用了封装后的 OpenGL 交换双缓存的函数。

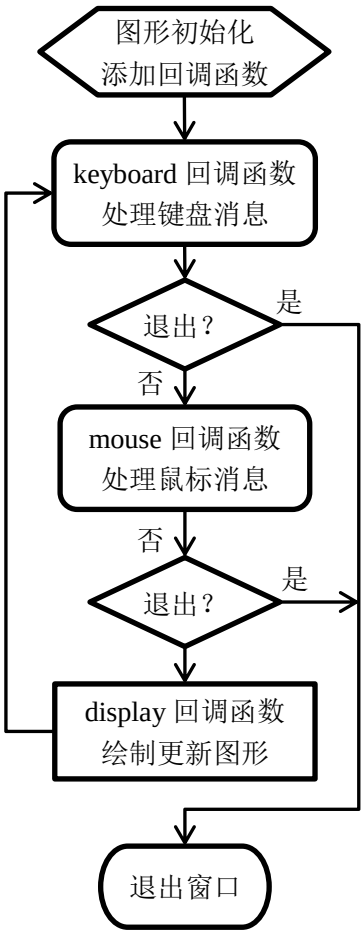

图 5：Graph2D 运行流程图

2. 图形函数使用说明

如图 2 结构图所示，Graph2D 图形函数分为 8 类，下面将按类别对每个函数的基本功能、参数与返回值等进行说明，为方便用户理解，在每类函数后面提供了一段示例代码。

2.1 图形初始化

（1）图形系统初始化

```
void initGraph(void (*display)(),
               void (*keyboard)(unsigned char key)=NULL,
               void (*mousePress)(int message, int x, int y)=NULL,
               void (*mouseMove)(int x, int y)=NULL,
               void (*specialKeyboard)(int key)=NULL);
```

参数与返回值：函数包含 5 个函数指针，分别对应图形窗口绘制与更新函数 display，键盘响应函数 keyboard，鼠标按键响应函数 mousePress，鼠标移动响应函数 mouseMove 和键盘特殊键响应函数 specialKeyboard。

说明：该函数用于构建图形窗口，嵌入回调函数。第 1 个参数不能缺省，后面 4 个参数均可缺省，因此用户至少要定义一个用于图形显示与更新的 display 函数。

（2）示例代码

仅带单一参数 initGraph 函数的使用例如代码 1 所示。代码 2 中的 initGraph 函数使用了全部参数，该代码实现了一个交互式圆的绘制，鼠标移动到圆内时绘制实体圆，否则绘制线框圆，鼠标单击将修改圆心位置，按键‘w’、‘s’、‘a’、‘d’和‘↑’、‘↓’、‘←’、‘→’用于控制圆的上下左右移动。

```
01  #include <Graph2D.h>
02  using namespace graph;
03  int g_x=500, g_y=350;
04  bool g_bSolid = false;
05  void display() {
06      if (g_bSolid) fillCircle(g_x, g_y, 100);
07      else circle(g_x, g_y, 100);
08  }
09  void keyboard(unsigned char key) {
10      if (key=='a') g_x-=10;
```

```
11          else if (key=='d') g_x+=10;
12          else if (key=='w') g_y+=10;
13          else if (key=='s') g_y-=10;
14      }
15      void mousePress(int message, int x, int y) {
16          if (message == LEFT_BUTTON_DOWN) {
17              g_x = x; g_y = y;
18          }
19      }
20      void mouseMove(int x, int y) {
21          double r = sqrt((x-g_x)*(x-g_x)+(y-g_y)*(y-g_y));
22          if (r<100) g_bSolid = true;
23          else g_bSolid = false;
24      }
25      void specialKeyboard(int key) {
26          if (key==KEY_LEFT) g_x-=10;
27          else if (key==KEY_RIGHT) g_x+=10;
28          else if (key==KEY_UP) g_y+=10;
29          else if (key==KEY_DOWN) g_y-=10;
30      }
31      int main(int argc, char *argv[]) {
32          initGraph(display, keyboard, mousePress, mouseMove, specialKeyboard);
33          return 0;
34      }
```

代码 2：可交互圆的绘制

2.2 窗口设置

（1）设置窗口背景颜色

void **setBkColor**(unsigned char red, unsigned char green, unsigned char blue);

void **setBkColor**(unsigned long color);

参数与返回值：两个重载函数均可设置窗口背景颜色，其中第 1 个函数中对应的 3 个参数为红绿蓝 3 色分量，取值范围均为[0, 255]；第 2 个函数对应参数可用一个 16 进制宏表示，例如红色 RED 对应 0XFF0000，类似的宏还包括 BLACK、GREEN、BLUE、GREY、PURPLE、

WHITE、CYAN、MAGENTA、YELLOW、ORANGE 和 BROWN。

说明：一般在 initGraph 前调用。

（2）设置窗口标题

void **setWinTitle**(char strTitle[]);

参数与返回值：默认窗口标题为“Simple Graph Window”，通过 strTitle 字符数组指定新的窗口标题。

说明：一般在 initGraph 前调用。

（3）设置窗口大小

void **setWinSize**(int width, int height);

参数与返回值：默认窗口大小为 1024*768，通过 width 和 height 分别设置新窗口的宽度与高度。

说明：一般在 initGraph 前调用。

（4）获取窗口宽度

int **getWinWidth**();

参数与返回值：返回当前窗口宽度。

说明：无调用限制。

（5）获取窗口高度

int **getWinHeight**();

参数与返回值：返回当前窗口高度。

说明：无调用限制。

（6）用背景颜色清屏

void **clearWindow**();

参数与返回值：无。

说明：在 display 函数中调用后，前面绘制的所有图形将被清除。

（7）设置全屏

void **setFullScreen**();

参数与返回值：无。

说明：进入全屏模式，一般在 initGraph 前调用。

（8）退出全屏

void **exitFullScreen**();

参数与返回值：无。

说明：退出全屏，切换到窗口模式，一般在 keyboard 回调函数中使用。

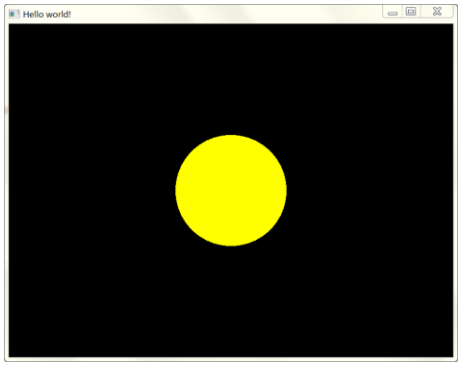

图 6：自定义窗口及图形绘制结果

（9）示例代码

代码 3 通过使用窗口函数实现了一个标题为“Hello world!”，背景为黑色，大小为 800*600 的图形窗口，并在窗口中心位置绘制了一个黄色填充圆（如图 6 所示），缩放窗口后绘制的圆依然处于窗口中心位置。

```
01    #include <Graph2D.h>
02    using namespace graph;
03    void display(){
04        setColor(YELLOW);
05        fillCircle(getWinWidth()/2, getWinHeight()/2, 100);
06    }
07    int main(int argc, char *argv[]){
08        setWinTitle("Hello world!");
09        setWinSize(800, 600);
10        setBkColor(BLACK);
11        initGraph(display);
12        return 0;
13    }
```

代码 3：窗口设置相关函数使用例

2.3 基本图元绘制

（1）显示像素点

void **putPixel**(int x, int y);

参数与返回值：在坐标(x, y)处显示一个像素点。

说明：在 display 中调用。

（2）设置线型为实线

void **setSolidLines**();

参数与返回值：无。

说明：设置当前线型为实线，默认线型为实线，该函数在 display 中调用。

（3）设置线型为虚线

void **setDottedLines**();

参数与返回值：无。

说明：在 display 中调用。

（4）设置线宽

void **setLineWidth**(int width);

参数与返回值：设置当前线型为虚线。

说明：在 display 中调用。

（5）设置图元颜色

void **setColor**(unsigned char red, unsigned char green, unsigned char blue);

void **setColor**(unsigned long color);

参数与返回值：与 setBkColor 类似，两个重载函数均可设置图元绘制颜色，其中第 1 个函数中对应的 3 个参数为红绿蓝 3 色分量，取值范围为[0, 255]；第 2 个函数对应参数可用一个 24 位的 16 进制数表示，如 0XFFFF00 对应为橙色。

说明：该函数可设置点、线、基本形状、文本以及与图像的混合色，在 display 中调用。

（6）绘制线段

void **line**(double startX, double startY, double endX, double endY);

参数与返回值：startX 和 startY 对应线段起点，endX 和 endY 对应线段终点。

说明：在 display 中调用。

（7）绘制带箭头线段

void **arrowLine**(double startX, double startY, double endX, double endY,

double s1=5.0, double s2=10.0);

参数与返回值：startX 和 startY 对应线段起点，endX 和 endY 对应线段终点，s1 位箭头宽度，s2 为箭头长度。

说明：在 display 中调用。

（8）绘制矩形线框

void **rectangle**(double left, double bottom, double right, double top);

参数与返回值：left 和 bottom 对应矩形左下角坐标，right 和 top 对应右上角坐标。

说明：在 display 中调用。

（9）绘制三角形

void **triangle**(double x0, double y0, double x1, double y1, double x2, double y2);

参数与返回值：(x0, y0)、(x1, y1)和(x2, y2)分别对应三角形的三个顶点坐标。

说明：在 display 中调用。

（10）绘制多边形

void **polygon**(int numPoints, double x[], double y[]);

参数与返回值：numPoints 为多边形顶点数，x[]和 y[]是用于存储多边形顶点 x 坐标和 y 坐标的数组。

说明：在 display 中调用。

（11）绘制圆

void **circle**(double cx, double cy, double radius);

参数与返回值：cx 和 cy 为圆心坐标，radius 为半径。

说明：在 display 中调用。

（12）绘制椭圆

void **ellipse**(double cx, double cy, double xRadius, double yRadius);

参数与返回值：cx 和 cy 为椭圆圆心坐标，xRadius 为 x 方向轴半径，yRadius 为 y 方向轴半径。

说明：在 display 中调用。

（13）绘制圆弧

void **arc**(double cx, double cy, double radius, double startAngle, double endAngle);

参数与返回值：cx 和 cy 为圆心坐标，radius 为半径，startAngle 和 endAngle 对应圆弧的起始角度。

说明：在 display 中调用。

（14）绘制椭圆环

void **donut**(double cx, double cy, double xRadius, double yRadius, double ratio);

参数与返回值：cx 和 cy 为圆心坐标，xRadius 为 x 方向轴半径，yRadius 为 y 方向轴半

径，ratio 为内圆半径与外圆半径的比例。

说明：在 display 中调用。

（15）绘制填充矩形

void **fillRectangle**(double left, double bottom, double right, double top);

参数与返回值：left 和 bottom 对应矩形左下角坐标，right 和 top 对应右上角坐标。

说明：在 display 中调用。

（16）绘制填充三角形

void **fillTriangle**(double x0, double y0, double x1, double y1, double x2, double y2);

参数与返回值：(x0, y0)、(x1, y1)和(x2, y2)分别对应三角形的三个顶点坐标。

说明：在 display 中调用。

（17）绘制填充多边形

void **fillPolygon**(int numPoints, double x[], double y[], bool bConvex=true);

参数与返回值：numPoints 为多边形顶点数，x[]和 y[]是用于存储多边形顶点 x 坐标和 y 坐标的数组，默认情况下 bConvex 为真，即绘制凸多边形，若为凹多边形，则将 bConvex 设为假，采用扫描线算法填充。

说明：在 display 中调用，对于凹多边形的填充，暂不支持图像邦定功能。

（18）绘制填充圆

void **fillCircle**(double cx, double cy, double radius);

参数与返回值：cx 和 cy 为圆心坐标，radius 为半径。

说明：在 display 中调用。

（19）绘制填充椭圆

void **fillEllipse**(double cx, double cy, double xRadius, double yRadius);

参数与返回值：cx 和 cy 为椭圆圆心坐标，xRadius 为 x 方向轴半径，yRadius 为 y 方向轴半径。

说明：在 display 中调用。

（20）绘制扇形

void **sector**(double cx, double cy, double radius, double startAngle, double endAngle);

参数与返回值：cx 和 cy 为圆心坐标，radius 为半径，startAngle 和 endAngle 对应圆弧的起始角度。

说明：在 display 中调用。

（21）绘制填充椭圆环

void **fillDonut**(double cx, double cy, double xRadius, double yRadius, double ratio);

参数与返回值：cx 和 cy 为圆心坐标，xRadius 为 x 方向轴半径，yRadius 为 y 方向轴半径，ratio 为内圆半径与外圆半径的比例。

说明：在 display 中调用。

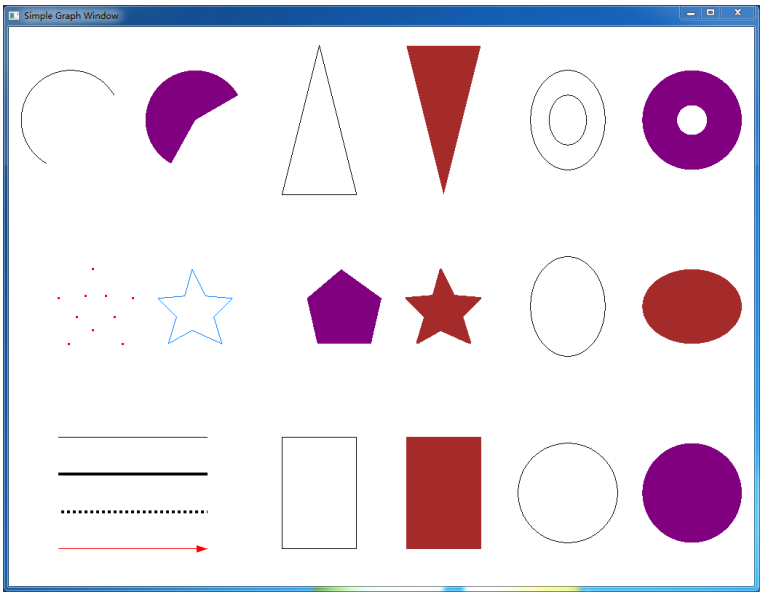

图 7：点、线、面等基本图元绘制结果

（22）示例代码

图 7 是代码 4 编译运行后的结果，设置绘制图元颜色即可使用用户自定义函数（如第 10 行），也可使用系统预定义的颜色（如第 20 行），点的绘制代码采用 putPixel 函数，与之类似，line 和 arrowLine 用于绘制线段和箭头线，此外本例还演示了矩形、圆、椭圆、三角形、圆弧、扇形和多边形的绘制，其中凸多边形和凹多边形的绘制分别采用了不同参数的 fillPolygon 函数（第 51 行和第 57 行）。

```
01    #include <Graph2D.h>
02    using namespace graph;
03    void display() {
04        int w = getWinWidth(),    h = getWinHeight(), i;
05        double dx = 1.0/3, dy = 1.0/3;
06        const int n = 10;
07        double poly[n][2] = {{0, 0.61},{0.25, 0.36},{0.14, 0},{0.46, 0.18},{0.86, 0},
08                             {0.75, 0.36}, {1, 0.61}, {0.64, 0.64}, {0.46, 1.0},{0.36, 0.64}};
09        double tmpPolyX[n], tmpPolyY[n];
```

```
10          setColor(0, 128, 255);
11          for(i=0; i<n; i++)
12              putPixel((0.2+0.3*poly[i][0])*dx*w, (1.3+0.4*poly[i][1])*dy*h);
13          setColor(BLACK);
14          line(0.2*dx*w, 0.8*dy*h, 0.8*dx*w, 0.8*dy*h);
15          setLineWidth(4);
16          line(0.2*dx*w, 0.6*dy*h, 0.8*dx*w, 0.6*dy*h);
17          setDottedLines();
18          line(0.2*dx*w, 0.4*dy*h, 0.8*dx*w, 0.4*dy*h);
19          setLineWidth(1);
20          setColor(RED);
21          setSolidLines();
22          arrowLine(0.2*dx*w, 0.2*dy*h, 0.8*dx*w, 0.2*dy*h);
23          setColor(BLACK);
24          rectangle(1.1*dx*w, 0.2*dy*h, 1.4*dx*w, 0.8*dy*h);
25          setColor(BROWN);
26          fillRectangle(1.6*dx*w, 0.2*dy*h, 1.9*dx*w, 0.8*dy*h);
27          setColor(BLACK);
28          circle(2.25*dx*w, 0.5*dy*h, 0.2*dx*w);
29          setColor(PURPLE);
30          fillCircle(2.75*dx*w, 0.5*dy*h, 0.2*dx*w);
31          setColor(BLACK);
32          ellipse(2.25*dx*w, 1.5*dy*h, 0.15*dx*w, 0.2*dx*w);
33          setColor(BROWN);
34          fillEllipse(2.75*dx*w, 1.5*dy*h, 0.2*dx*w, 0.15*dx*w);
35          setColor(BLACK);
36          donut(2.25*dx*w, 2.5*dy*h, 0.15*dx*w, 0.2*dx*w, 0.5);
37          setColor(PURPLE);
38          fillDonut(2.75*dx*w, 2.5*dy*h, 0.2*dx*w, 0.2*dx*w, 0.3);
39          setColor(BLACK);
40          triangle(1.1*dx*w, 2.1*dy*h, 1.4*dx*w, 2.1*dy*h, 1.25*dx*w, 2.9*dy*h);
41          setColor(BROWN);
42          fillTriangle(1.6*dx*w, 2.9*dy*h, 1.9*dx*w, 2.9*dy*h, 1.75*dx*w, 2.1*dy*h);
43          setColor(BLACK);
```

```
44          arc(0.25*dx*w, 2.5*dy*h, 0.2*dx*w, 30, 240);
44          setColor(PURPLE);
45          sector(0.75*dx*w, 2.5*dy*h, 0.2*dx*w, 30, 240);
46          for(i=0; i<n; i++) {
47              tmpPolyX[i] = (0.6+0.3*poly[i][0])*dx*w;
48              tmpPolyY[i] = (1.3+0.4*poly[i][1])*dy*h;
49          }
50          setColor(BLACK);
51          polygon(n, tmpPolyX, tmpPolyY);
52          for(i=0; i<n/2; i++) {
53              tmpPolyX[i] = (1.2+0.3*poly[2*i][0])*dx*w;
54              tmpPolyY[i] = (1.3+0.4*poly[2*i][1])*dy*h;
55          }
56          setColor(PURPLE);
57          fillPolygon(n/2, tmpPolyX, tmpPolyY);
58          for(i=0; i<n; i++) {
59              tmpPolyX[i] = (1.6+0.3*poly[i][0])*dx*w;
60              tmpPolyY[i] = (1.3+0.4*poly[i][1])*dy*h;
61          }
62          setColor(BROWN);
63          fillPolygon(n, tmpPolyX, tmpPolyY, false);
64      }
65      int main(){
66          initGraph(display);
67          return 0;
68      }
```

代码 4：基本图形绘制测试

2.4 文本设置与显示

（1）设置字号与字体

int **createFont**(int pixHeight=24, const char style[]="Arial",

int weight=FW_NORMAL, bool bItalic=false);

参数与返回值：pixHeight 为字体高度（单位为像素），默认高度为 24；style 为字体，默认为“Arial”，也可设为“Times New Roman”、“宋体”、“黑体”等；weight 为字体粗度，

默认为正常粗度，也可设为加粗（对应宏为 BOLD）；bItalic 为倾斜模式，默认为非斜体。函数将返回创建字体标记号，若创建失败则返回-1。

说明：在 initGraph 前调用，暂不支持汉字显示。

（2）设置文本颜色

说明：文本颜色设置函数与基本图元颜色设置函数相同，均为 setColor()。

（3）使用指定字体

void **useFont**(int fontID);

参数与返回值：fontID 对应 createFont 中创建返回的字体标记号。

说明：在 display 中调用。

（4）显示文本

void **putText**(int x, int y, const char strText[]);

参数与返回值：在屏幕坐标(x, y)处显示字符串 strText。

说明：在 display 中调用，若不预先调用 createFont 和 useFont 函数，将使用默认“Arial”24 号字体。

（5）获取指定字体宽度

int **getFontWidth**(int fontID);

参数与返回值：返回 fontID 对应字体宽度。

说明：在 display 中调用。

（6）获取指定字体高度

int **getFontHeight**(int fontID);

参数与返回值：返回 fontID 对应字体高度。

说明：在 display 中调用。

图 8：不同字体与风格测试结果

（7）示例代码

图 8 显示了“Arial”和“Times New Roman”两种字体，黑体和加粗显示等风格也得到体现，具体实现参见代码 5。

```
01    #include <Graph2D.h>
02    using namespace graph;
03    int g_fontID[4];
04    int g_imgID;
05    void display() {
06        int w = getWinWidth(), h = getWinHeight();
07        showImage(g_imgID);
08        setColor(BROWN);
09        useFont(g_fontID[0]);
10        putText(w/32, 6*h/7, "The quick brown fox");
11        setColor(BLACK);
12        useFont(g_fontID[1]);
13        putText(w/32, 5*h/7, "jumps over");
14        setColor(ORANGE);
15        useFont(g_fontID[2]);
16        putText(w/32, 4*h/7, "the lazy dog.");
17        setColor(BLACK);
18        useFont(g_fontID[1]);
19        putText(w/4, 3*h/7, "0123456789");
20        useFont(g_fontID[3]);
21        putText(2*w/3, 2*h/7, "Z,Z,Z,Z,Z");
22    }
23    int main() {
24        g_fontID[0] = createFont(36, "Arial", BOLD);
25        g_fontID[1] = createFont(48, "Arial");
26        g_fontID[2] = createFont(32, "Times New Roman", BOLD);
27        g_fontID[3] = createFont(24, "Times New Roman", BOLD, true);
28        g_imgID = loadImage(".\\images\\foxoverdog.bmp");
29        setWinSize(600, 600);
30        initGraph(display);
31        return 0;
```

```
32      }
```

代码 5：字体创建与显示测试

2.5 图像载入与填充

（1）载入图像

int **loadImage**(char *fileName, bool bGrey=false);

参数与返回值：读入 fileName 对应的 BMP 真彩色图像文件，bGrey 若为真值，则将读入图像转化为灰度图像，否则为原图像。该函数返回读取图像的标记号，若图像读取失败，返回-1。

说明：在 initGraph 前调用，暂不支持真彩色 BMP 格式以外图像文件的读写。

（2）用指定图像填充基本图元

void **bindImage**(int imageID, bool bBlend=false, double startU=0.0, double startV=0.0,
double endU=1.0, double endV=1.0);

参数与返回值：使用标记号为 imageID 的图像填充后续图像，如果 bBlend 为真，则图像将与 setColor 指定颜色混合后再填充后续图元对象（如矩形、圆、凸多边形等），默然情况下图像不与设定颜色混合；startU、startV、endU、endV 分别指定从图像中截取矩形区域子图像的左下角和右上角坐标，坐标范围量化为[0, 1]，默认情况下获取的是整幅图像。

说明：在 display 中调用。

（3）显示指定图像

void **showImage**(int imageID, double startU=0.0, double startV=0.0,
double endU=1.0, double endV=1.0);

参数与返回值：在屏幕窗口上显示标记号为 imageID 的图像，startU、startV、endU、endV 分别指定从图像中截取矩形区域子图像的左下角和右上角坐标，坐标范围量化为[0, 1]，默认情况下显示整幅图像。

说明：在 display 中调用。

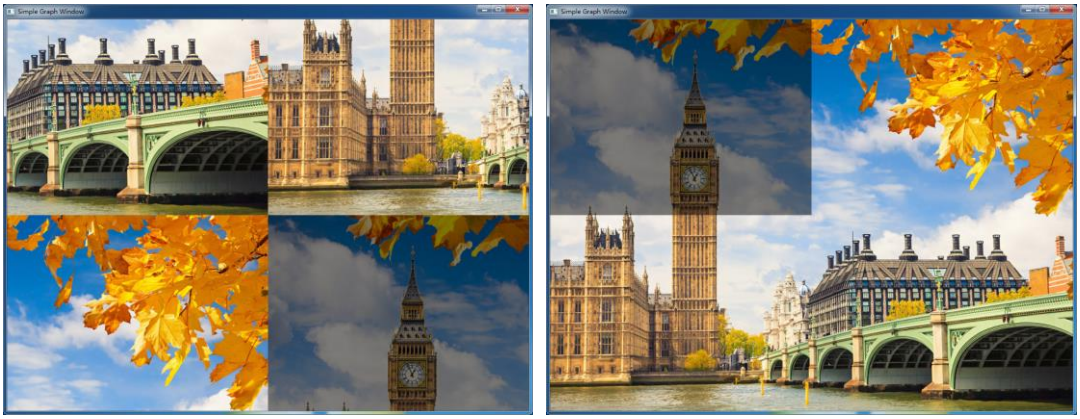

图 9：图像操作测试结果（左：拼图初始状态；右：完成状态）

（4）示例代码

代码 6 实现了一个简易拼图游戏，运行结果如图 9 所示，拼图被分成 4 块，设定初始状态后，采用鼠标交互实现子块移动，拼图完成后从控制台输出提示信息。上述游戏的实现用到了图像读入、分块显示、图像绑定及颜色混合等函数。

```
01  #include <Graph2D.h>
02  using namespace graph;
03  int g_imgID = -1;
04  int g_empty = 3;
05  int g_arr[4] = {2,3,0,1};
06  double g_pos[4][4] = {{0,0,0.5,0.5},{0.5,0,1,0.5},{0.5,0.5,1,1},{0,0.5,0.5,1}};
07  int g_w, g_h;
08  void display() {
09      g_w = getWinWidth();      g_h = getWinHeight();
10      for(int i=0; i<4; i++) {
11          if (g_arr[i]==g_empty) setColor(GREY);
12          else setColor(WHITE);
13          bindImage(g_imgID, true, g_pos[g_arr[i]][0], g_pos[g_arr[i]][1],
14                  g_pos[g_arr[i]][2], g_pos[g_arr[i]][3]);
15          fillRectangle(g_pos[i][0]*g_w, g_pos[i][1]*g_h, g_pos[i][2]*g_w, g_pos[i][3]*g_h);
16      }
17  }
18  void mousePress(int message, int x, int y) {
19      int i, j, k=-1;
20      if (message==LEFT_BUTTON_DOWN) {
21          for(i=0; i<4; i++)
```

```
                if(g_pos[i][0]*g_w<=x&&x<=g_pos[i][2]*g_w
22                 &&g_pos[i][1]*g_h<=y&&y<=g_pos[i][3]*g_h)
23                     break;
24             if (i!=4) {
25                 j = i-1;
26                 if (j<0) j=j+4;
27                 if (g_arr[j]==g_empty)    k = j;
28                 j = i+1;
29                 if (j>=4) j=j-4;
30                 if (g_arr[j]==g_empty)    k = j;
31             }
32         }
33         if (k!=-1) {
34             j = g_arr[k];
35             g_arr[k] = g_arr[i];
36             g_arr[i] = j;
37             for(i=0; i<4; i++)
38                 if (g_arr[i]!=i) break;
39             if (i==4) printf("拼图完成\n");
40         }
41     }
42     int main(int argc, char *argv[]) {
43         g_imgID = loadImage(".\\images\\london.bmp");
44         initGraph(display, NULL, mousePress);
45         return 0;
46     }
```

代码 6：图像读入、显示与交互测试

2.6 键盘交互

（1）普通键盘消息处理函数

void **keyboard**(unsigned char key);

参数与返回值：若产生普通键盘消息，如“A-Z”、“a-z”、“0-9”等键被按下，操作系统将调用 keyboard 函数，其中 key 中存储按键的 ASCII 值。

说明：函数指针传递给 initGraph 中的第 2 个参数。

（2）特殊键盘消息处理函数

void **specialKeyboard**(int key);

参数与返回值：若产生特殊键盘消息，如“F1-F10”、 “↑↓←→”等键被按下，操作系统将调用 specialKeyboard 函数，其中 key 中存储特殊按键的标记号。

说明：函数指针传递给 initGraph 中的第 5 个参数。

（3）示例代码

参考 2.1 节图形初始化示例代码。

2.7 鼠标交互

（1）鼠标单击消息处理函数

void **mousePress** (int message, int x, int y);

参数与返回值：若产生鼠标按键消息，如左、中、右键被按下，操作系统将调用 mousePress 函数，其中 message 中存储按键的标记号，(x, y)为按键按下时鼠标的位置。

说明：函数指针传递给 initGraph 中的第 3 个参数。

（2）鼠标移动消息处理函数

void **mouseMove** (int x, int y);

参数与返回值：若产生鼠标移动消息，操作系统将调用 mouseMove 函数，其中 (x, y) 为鼠标移动时所在位置。

说明：函数指针传递给 initGraph 中的第 4 个参数。

（3）示例代码

参考 2.1 节图形初始化示例代码。

2.8 其它

（1）设置帧速率

void **setFPS**(int framePerSec);

参数与返回值：设置屏幕更新频率为 framePerSec，默然帧速率为每秒 40 帧。

说明：在 initGraph 前调用，如果图像绘制过于复杂，即使设置高帧速率也无法提高画面更新速度。

（2）获取帧速率

int **getFPS**();

参数与返回值：返回当前帧速率。

说明：在 display 中调用。

（3）显示坐标系

void **showCoordinate**();

参数与返回值：显示当前坐标系，默认情况下不显示。

说明：在 display 中调用，按 F2 键将调用该函数。

（4）显示帮助信息

void **showHelp**();

参数与返回值：显示帮助信息，默认情况下不显示。

说明：在 initGraph 前调用，按 F1 键将调用该函数。

（5）保存屏幕图像至文件

void **saveScreenImage**(char *fileName);

参数与返回值：实现抓屏功能，保存屏幕图像到 BMP 图像文件 fileName，fileName 后缀名为 bmp。

说明：在 keyboard 回调函数中使用。

图 10：图像载入、绑定与保存结果（左：原始图像；右：保存图像）

（6）示例代码

编译运行代码 7，读入图 10（左）图像，风格化处理后的结果如图 10（右）所示，保存屏幕图像不需使用抓屏软件，在 keyboard 回调函数中处理‘s’或‘S’按键消息，然后调用 **saveScreenImage** 实现图像保存。

```
01    #include <Graph2D.h>
02    using namespace graph;
03    int g_imgID;
```

```
04    void display() {
05        setColor(ORANGE);
06        bindImage(g_imgID, true);
07        fillCircle(getWinWidth()/2, getWinHeight()/2, 300);
08    }
09    void keyboard(unsigned char key) {
10        if (key=='s' || key=='S')
11            saveScreenImage(".\\images\\londonCopy.bmp");
12    }
13    int main() {
14        g_imgID = loadImage(".\\images\\london.bmp");
15        initGraph(display, keyboard);
16        return 0;
17    }
```

代码 7：图像载入、绑定与保存测试

3. 结语

C++描述的面向对象程序设计课程存在**理论复杂**、**抽象性高**的特点，为在教学中**直观体现**面向对象程序设计的思想，课程组结合 OpenGL 和 VGA 模式下的 graphics 库，开发了一个**面向教学的简易图形库 Graph2D**，该图形库结构简单清晰，易学易用，课程组已在多年教学和综合实践中得到检验。实践表明，保留以 main 函数为主线，同时提供具备基本图形绘制和图像显示功能的图形库，**增加了学习的趣味性**，调动了学生的积极性，有助于学生专注本课程的核心内容，避免花费过多时间去理解窗口程序和图形用户界面的设计。此外，Graph2D 也是一个过渡型图形库，通过该图形库，学生熟悉了图形绘制的基本流程，便于进一步过渡到专业软件开发包（Qt 或 MFC）或图形库（OpenGL，DirectX）的学习。

西北农林科技大学

信息工程学院

计算机科学系

面向对象程序设计课程组

2014年3月初稿