用 C++Builder4.0 结合 OpenGL 开发三维图形

骆 锐 郑 军 (湖南大学工程软件研究所 长沙 410082)

摘要:本文系统地介绍了 C++Builder4.0 和 OpenGL 在用户图形界面和三维图形技术方面功能和作用,并为它们的混合编程提供了一个基本的框架。

关键字:C++Builder4.0 OpenGL 混合编程

一、概述

以前开发复杂三维图形只能在图形工作站上进行,由于条件的限制,使用的人很少。随着微机硬件设备的发展和软件功能的完善,使得将只有在工作站上才能实现的图形功能移植到微机平台上成为了可能。本文介绍的就是在微机上利用 C++ Builder 和 OpenGL 混合编程技术开发高质量三维图形的基本方法。

二、C++Builder 简述

C++Builder 是 Inprise 公司推出的一款高性能全新的可视化的集成开发环境。它使用了 Microsoft Windows 图形用户界面的许多先进特性和设计思想,采用了弹性的,可重用的和完整的面向对象程序语言(Object-Oriented Programming Language),为我们提拱了一种方便,快捷的 Windows 应用程序开发工具。特别是 C++Builder4.0,内置了 100 多个完全封装了 Win9X 公用特性且具有完全可扩展性(包括支持 ActiveX)的控件,用户利用它们可以非常容易的开发出自己所需的界面。同时 C++Build4.0 还具有一个专业 C++ 开发环境所应提供的全部功能:快速,高效,灵活的编译优化,逐步连接,CPU 透视,命令行工具等,可实现可视化的编程环境和功能强大的编程语言的完美结合。

三、OpenGL 简述

OpenGL 是 SGI 公司开发的一个非常优秀的开放式三维图形软件库,目前已成为高质量三维图形的工业标准。同时它又是一个 API(Application Programming Interface)--与硬件无关的编程接口,可以在不同的硬件平台上得以实现。

OpenGL 包括 100 多个图形函数,开发人员可以用这些函数来构造景物模型,进行三维图形实时交互软件的开发。现在很多三维图形软件,如 3Dmax 可以方便地建模,但难以对其进行控制。用 OpenGL 就可以方便地控制模型,制作 CAD,制作三维动画,实现虚拟仿真,进行影视采集,使我们制作出的三维图形更逼真,更真实。在实际应用中,很多优秀的工具软件和游戏软件都是用 OpenGL 开发的。

OpenGL 具有绘制复杂三维图形的各种功能,方便地实现三维图形的交互操作,而程序员只需很少的计算机图形学知识即可生成十分精美的图形。它的主要功能有:

(1) 建模 提供绘制三维点,线,多边形及复杂曲线曲面的函数;

(2) 变换 包括三维基本变换与投影变换;

(3) 着色,光照,消影;

(4) 混合,平滑,雾化;

(5) 纹理映射,位图。

四、混合编程及具体应用

微软公司已经将 OpenGL 图形库链接到 Windows 95 OSR2 版本(Windows 98 即自带 OpenGL 库)中,这就为混合编程提供了可能,使得开发者可以充分利用 OpenGL 的强大功能来设计三维图形。

在图形软件的开发中,除了强大的功能外,良好的交互性也是十分必要的。C++Builder4.0 就是一个非常优秀的图形用户界面开发软件,结合 OpenGL 的图形处理功能,就能给程序员广阔的开发空间。

在混合编程中,我们是以 C++Builder 为基础,OpenGL 镶嵌在其中,而关键在于怎样使它们连接起来,做到用户用 C++Builder 发出命令,OpenGL 响应绘图。因为两者都是以 C++ 语言为基础发展起来的,所以程序在接口上没什么问题。

以下是一个在 C++Builder4.0 环境下编写的完整源程序，并运行通过。

```
//- - - - - - - - - - - - - - - - - - - - - - - - -
#include <vcl.h>
#include <gl/gl.h>
#include <gl/glu.h>
#pragma hdrstop
HDC hdc;
HGLRC hglrc;
HWND hWnd;
RECT oldrect;

void mydraw();
void draw-object(void);
void initlights(void);

#include "Draw.h"
#pragma package(smart-init)
#pragma resource "*.dfm"

TForm1 *Form1;
//- - - - - - - - - - - - - - - - - - - - - - - - -
--fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void --fastcall TForm1::FormCreate(TObject *Sender)
{
    int iPixelFormat;
    hdc = GetDC(Handle);
    PIXELFORMATDESCRIPTOR pfd = {
      sizeof(PIXELFORMATDESCRIPTOR), // size of this pfd
      1, // version number
      PFD-DRAW-TO-WINDOW | // support window
      PFD-SUPPORT-OPENGL | // support OpenGL
      PFD-DOUBLEBUFFER, // double buffered
      PFD-TYPE-RGBA, // RGBA type
      24, // 24-bit color depth
      0, 0, 0, 0, 0, 0, // color bits ignored
      0, // no alpha buffer
      0, // shift bit ignored
      0, // no accumulation buffer
      0, 0, 0, 0, // accum bits ignored
      32, // 32-bit z-buffer
      0, // no stencil buffer
      0, // no auxiliary buffer
      PFD-MAIN-PLANE, // main layer
      0, // reserved
      0, 0, 0 // layer masks ignored
      };
    iPixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, iPixelFormat, &pfd);
    hglrc = wglCreateContext(hdc);
    wglMakeCurrent(hdc, hglrc);

    RECT oldrect;
    oldrect = GetClientRect();

    glMatrixMode(GL-PROJECTION);
    glLoadIdentity();
    glOrtho(-oldrect.right/150., oldrect.right/150., -
oldrect.bottom/150.,
      oldrect.bottom/150., -1000, 1000);
    glMatrixMode(GL-MODELVIEW);
    glLoadIdentity();
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void --fastcall TForm1::FormResize(TObject *Sender)
{
    RECT rect;
    rect = GetClientRect();
    glViewport(0, 0, rect.right, rect.bottom);
    if((oldrect.right > rect.right) || (oldrect.bottom >
rect.bottom))
      FormPaint(Sender);
    oldrect.right = rect.right;
    oldrect.bottom = rect.bottom;
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void --fastcall TForm1::FormDestroy(TObject *Sender)
{
hglrc = wglGetCurrentContext();
```

```
hdc = wglGetCurrentDC();

wglMakeCurrent (NULL, NULL) ;
if(hglrc)
  wglDeleteContext (hglrc);
if(hdc)
  ReleaseDC(hWnd, hdc);
PostQuitMessage(0);
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void --fastcall TForm1::FormPaint(TObject * Sender)
{
    PAINTSTRUCT ps;
    hdc = BeginPaint(hWnd, &ps);
    mydraw();
    EndPaint(hWnd, &ps);
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void mydraw()
{
      initlights();
      glClear ( GL-COLOR-BUFFER-BIT | GL-DEPTH-
BUFFER-BIT);
    glLoadIdentity();
    glPushMatrix();
      glRotatef(50.0,0.0,1.0,0.0);
      glRotatef( - 5.0,1.0,0.0,0.0);
      draw-object();
    glPopMatrix();
    glFlush();
    SwapBuffers(wglGetCurrentDC());
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void initlights(void)
{
    GLfloat light-ambient[] = {0.3,0.3,0.2};
    GLfloat light-diffuse[] = {1.0,1.0,1.0};
    GLfloat light-position[] = { - 2.0, - 2.0, - 2.0,
1.0 };
      glLightfv(GL-LIGHT0, GL-AMBIENT, light-am-
bient);
      glLightfv(GL-LIGHT0, GL-DIFFUSE, light-dif-
fuse);
      glLightfv(GL-LIGHT0, GL-POSITION, light-po-
sition);
    glLightModeli(GL-LIGHT-MODEL-TWO-SIDE, GL-
TRUE);
    glEnable(GL-LIGHTING);
    glEnable(GL-LIGHT0);
    glDepthFunc(GL-LESS);
    glEnable(GL-DEPTH-TEST);
    glColorMaterial ( GL-FRONT-AND-BACK, GL-DIF-
FUSE);
    glEnable(GL-COLOR-MATERIAL);
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
void draw-object(void)
{
    static GLfloat p1[] = {2.165,0.0,1.25}, p2[] = { -
2.165,0.0,1.25},
    p3[] = {0.0,0.0, - 2.5}, p0[] = {0.0,2.5,0.0};
    static GLfloat c1[] = {0.0,1.0,0.0}, c2[] = {1.0,
0.0,0.0},
    c3[] = {0.0,0.0,1.0}, c0[] = {1.0,1.0,0.0};
  glBegin(GL-TRIANGLES);
    glColor3fv ( c1 ); glVertex3fv ( p1 ); glColor3fv ( c2 );
glVertex3fv(p2);glColor3fv(c3);glVertex3fv(p3);
    glColor3fv ( c0 ); glVertex3fv ( p0 ); glColor3fv ( c1 );
glVertex3fv(p1);glColor3fv(c2);glVertex3fv(p2);
    glColor3fv ( c0 ); glVertex3fv ( p0 ); glColor3fv ( c2 );
glVertex3fv(p2);glColor3fv(c3);glVertex3fv(p3);
    glColor3fv ( c0 ); glVertex3fv ( p0 ); glColor3fv ( c3 );
glVertex3fv(p3);glColor3fv(c1);glVertex3fv(p1);
  glEnd();
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
```

以上程序的运行结果是在 Form1 窗体上画一个三棱锥，另外还加了颜色和光照以突出效果。

在程序的开头，包含了一些必要的头文件和初始定义，在 Form1 中添加 FormCreate()，FormPaint()，FormResize()，FormDestroy()四个事件。

FormCreate()是应用程序处理的第一个事件，也是十分关键的一步，在这里我们进行 OpenGL 的初始化工作。GetDC()函数用来获取设备描述表(device context)

的句柄，然后在 PIXELFORMATDESCRIPTOR 结构中定义所需的像素格式，调用 ChoosePixelFormat()函数返回一个像素格式指数，并将该指数传递给 SetPixelFormat()，把这个像素格式设置成设备描述表的当前像素格式。完成这些之后就可以创建一个适合于该设备的 OpenGL 绘图描述表(render context)了，并且继承了设备描述表像素格式，用 wglMakeCurrent()在设备描述表和绘图描述表之间建立联系。最后定义 OpenGL 的视见区和投影方式，GetClientRect()用以获取当前 Form1 窗口大小的数值。

FormPaint()事件是当显示窗口被改变(如窗口大小变化)时重画，更新窗口的。BeginPaint()找出屏幕上那些地方发生了变化，用当前的更新的信息来填充此结构，EndPaint()用来指示 Windows 已更新了屏幕，OpenGL 的绘图语句在 FormPaint()事件中的 BeginPaint()与 EndPaint()之间进行调用。

FormDestroy()可以干净地关闭应用程序，清除窗口，释放系统资源。

FormResize()是重新定位窗口，程序判断窗口发生变化时作出的处理。

五、结论

在这里，我们以 C++Builder4.0 为工作平台提供了一个 OpenGL 图形软件开发的框架和一些基本的思想，开发者可以基于以上模块根据自己的需要添加控件(菜单，按钮等)，定义更多的系统资源和响应函数，同时也可以绘制复杂的三维图形，实现更为强大的功能。

参考文献

[1] Weo M, Neider J, Davis T, OpenGL Programming Guide, Silicon Graphics Inc, 1997

[2] Kempf R, Frazier C, OpenGL Reference Manual, Silicon Graphics Inc, 1996

[3] OpenGL Programming Guide, Addison-Wesley Publishing Company

[4] 广正工作室，《C++Builder 实用教程》，机械工业出版社，1998

(来稿时间:1999 年 4 月)