Motif、OpenGL 及混合编程

周军 成艾国 (湖南大学工程软件研究所 410082)

摘要:本文对 Motif 与 OpenGL 作了一次较系统的阐述,明确了它们分别在图形用户界面和三维图形技术方面的地位,进而从实践中对它们之间进行混合编程的重要性、可能性作了论述。

关键字:Motif OpenGL 混合编程

1.引言

Motif 与 OpenGL 在工作站上应用得极为广泛,现在 Microsoft 等公司引用了 Motif 和 OpenGL 标准,并将它们应用到微机上,如 Windows 95 或 Windows NT 版本,使图形用户界面系统和三维图形技术发生了一次大的飞越。

2.Motif 简述

Motif 是开放软件基金会 OSF(Open Software Foundation)于 1989 年推出的一个图形用户界面(GUI)系统。它融合了多种 GUI 产品中的优点,得到 OSF 所有成员及广大第三方厂商的支持。Motif 极有可能成为将来唯一的图形用户界面的标准。

Motif 引入了面向对象的程序设计方法,提供了一套构造更高层界面构件的数据结构和方法,遵从"只提供机制不提供风格"的原则,界面风格就由 Motif 对象元类的集合来决定。对象元类就是一些预先定义好的用户界面构件,如菜单,按钮,标尺,对话框等,同时 Motif 提供了功能强大的函数库,利用简单的函数就可创建不同属性的构件及获取,修改属性。

与其他的事件相比,Motif 提供了一种更方便的事件处理方法。当一事件发生之后,Motif 会自动确定出这个事件发生在哪个对象元窗口之中和这个对象元中注册了哪些包括回调函数、动作函数和事件处理函数在内的用户函数,进而确定是否在该事件发生时应执行的函数。这种过程被称为事件分发。

3.OpenGL 简述

OpenGL 是 SGI 公司在推出 GL(Graphics Library)三维图形库以来,进一步发展起来的三维图形开发和应用工具。现在 OpenGL 已被公认为是高性能图形和交互式视景处理的通用标准。目前 SGI, Microsoft, SUN, DEC, HP 等公司都采用了 OpenGL 标准。

看一种软件是否能表现出三维真实感图形,主要反映在可视化(Visualization),动画(Animation)和虚拟实现(Virtual Reality)三大技术上,在这方面,SGI 公司已走在世界的最前列。同其他三维图形工具软件包相比,OpenGL 在交互式图形建模能力和编程方便程度上具有不可动摇的地位。OpenGL 能灵活方便地实现二维和三维的高级图形技术,在性能上表现得异常优越,包括建模、变换、光线处理、色彩处理、动画、纹理映射、雾效果等。OpenGL 能形成逼真的三维环境,给我们一种全新的真实感。

4.混合编程

在应用开发中,特别是在图形软件的设计方面,要想设计出卓越的软件,一方面就得具有良好的用户界面,因为"好的界面是软件成功的一半";另一方面,也就是另一半,就得具有无可比拟的功能设计。OpenGL 是一种开放式的软件,是一种程序设计语言,这就给予了程序员以广阔的开发空间,能设计出功能强大的模块。这样,OpenGL 就迫切需要一个用户界面,来展示出它在三维图形上的优点。Motif 则是一个优良的图形用户界面开发工具,并具有与其他语言的接口。作者在应用探索中,已成功地完成了这一方面的连接,并开发出了一系列实用软件。

5.具体应用

在 Motif 和 OpenGL 混合编程中,关键之处在于怎样使它们连接起来,做到 Motif 发出命令,OpenGL 响应绘图;OpenGL 需求信息,Motif 给予请示。在这里,我们可以假设存在一个对象元,Motif 能对它进行管理,OpenGL 能将图形映射上去,问题就解决了。

以下是一个完整的源程序,并运行通过。

```
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <GL/GLwMDrawA.h>
#include <GL/gl.h>
```

```
#include <GL/glu.h>

static void input(), draw-scene(), do-resize(), init-window();
static GLXContext glx-context;

void main(int argc, char * * argv) {
  Arg args[20];
  int n;
  Widget glw, toplevel, form;
  static XtAppContext app-context;
  static String fallback-resources[] = {
    " * glwidget * width: 280", " * glwidget * height: 200",
    " * glwidget * rgba: TRUE", " * glwidget * allocateBackground: TRUE",
    NULL };
  toplevel = XtAppInitialize( &app-context, "Mixed", NULL, 0, &argc, argv,
fallback-resources, NULL, 0);
  n = 0;
  form = XmCreateForm(toplevel, "form", args, n);
  XtManageChild(form);

  n = 0;
  glw = GLwCreateMDrawingArea(form, "glwidget", args, n);
  XtManageChild (glw);
  XtAddCallback(glw, GLwNexposeCallback, draw-scene, NULL);
  XtAddCallback ( glw, GLwNresizeCallback, do-resize, NULL);
  XtAddCallback(glw, GLwNginitCallback, init-window, NULL);
  XtAddCallback ( glw, GLwNinputCallback, input, NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app-context);
}
static void input(Widget w, XtPointer client-data, XtPointer call) {
  GLwDrawingAreaCallbackStruct * call-data;
  call-data = (GLwDrawingAreaCallbackStruct * ) call;
  switch(call-data->event->type) {
    case KeyRelease: /* 在此加入事件处理函数 */
      break;
    case ButtonPress: /* ..... */
      break;
    case Expose: /* ..... */
      break;
    default: /* ..... */
      break; }
}

static void draw-scene(Widget w, XtPointer client-data, XtPointer call) {
  static char firstTime = 0x1;
  GLwDrawingAreaCallbackStruct * call-data;
  call-data = (GLwDrawingAreaCallbackStruct * ) call;
  GLwDrawingAreaMakeCurrent(w, glx-context);
  if (firstTime) {
    glViewport(0, 0, call-data->width, call-data->height);
    glMatrixMode(GL-PROJECTION);
    glLoadIdentity();
    gluOrtho2D( -(GLdouble) call-data->width/2.0, (GLdouble)call-data->width/2.0, -(GLdouble)call-data->height/2.0,(GLdouble) call-data->height/2.0 );
    glMatrixMode(GL-MODELVIEW);
    glLoadIdentity();
    firstTime = 0; glClearColor(0.0,0.0,1.0,0.0);
    glClear(GL-COLOR-BUFFER-BIT);
  }
}
static void do-resize(Widget w, XtPointer client-data, XtPointer call) {
  GLwDrawingAreaCallbackStruct * call-data;
  call-data = (GLwDrawingAreaCallbackStruct * ) call;
  GLwDrawingAreaMakeCurrent(w, glx-context);
  glViewport(0, 0, call-data->width, call-data->height);
  glMatrixMode(GL-PROJECTION);
  glLoadIdentity();
  gluOrtho2D( -(GLdouble) call-data->width/2.0, (GLdouble)call-data->width/2.0, -(GLdouble)call-data->height/2.0,(GLdouble) call-data->height/2.0 );
  glMatrixMode(GL-MODELVIEW);
```

```
  glLoadIdentity();
}

static void init-window(Widget w, XtPointer client-data,
XtPointer call-data){
  Arg args[1];
  XVisualInfo * vi;
  GLUquadricObj * quadObj;
  XtSetArg(args[0], GLwNvisualInfo, &vi);
  XtGetValues(w, args, 1);
  glx-context = glXCreateContext(XtDisplay(w), vi, 0,
GL-FALSE);
}
```

程序很简明地阐述了对一个单绘图窗口的创建和管理。总的来看，Motif 为基础，OpenGL 镶嵌在其中，两者紧密相关。Motif 与 OpenGL 语言都是以标准 C 语言为基础发展起来的。

在程序的开头，包含了一些必需的头文件。在 main( ) 中完全依照 Motif 语言编写。在初始创建顶层对象元(toplevel，它为所有对象元的祖父)的同时，我们为以下与 OpenGL 有关的对象元注册了四种资源，您也可以在以后的过程中增加，修改相关资源。另外，在程序中只创建了一个对象元 Form，并以它为父对象元创建一个 OpenGL 绘图区域。读者可依自己的需要创建一些构件，如菜单，标题，按钮等，来扩充自己的程序功能，在此不在详述。

程序用 GLwCreateMDrawingArea( ) 创建了一个 GLwMDrawinArea 对象元，返回给变量 glw，glw 就是 Motif 与 OpenGL 的混合交点。此对象元有点象 Motif 中的 GC(Graphics Context)对象元，不过 GC 为 Motif 内部对象元，不能对它进行复杂的操作。管理对象元之后，还必需对新建的 glw 进行初始化，暴露，加其他回调函数等操作。GLw MDrawinArea 对象元主要有四种回调资源：

GLwNginitCallback 其回调函数为 init-window( )。在对象元 glw 未进行初始化之前，所有关于 OpenGL 的操作都不能实现。在 init-window( ) 中使用 glXCreateContext( ) 去创建一个新的 GLX 使用现场(redering context)，并返回它的操作柄(handler)给 glx-context，这样，glw 才能响应事件。使用现场就是图元所能显示并能进行图像等操作的场所。

GLwNexposeCallback 其回调函数为 draw-scene( )。它能响应暴露事件，一般表现为重画屏幕。当一对象元初次实现时，也称为暴露。函数中使用了 GLwDrawingAreaMakeCurrent( )，具有与它功能相同的还有 GLXMakeCurrent( )，它们都能指定一个可画体(drawable)为当前的绘图现场，前者指定一个可画对象元，而后者还能指定一个窗口或屏幕。draw-scene( ) 的后半部是说明怎样去确定视点(glViewpoint( ))和投影方式(gluOrtho2D( ))。

GLwNresizeCallback 其回调函数为 do-resize( )。当对象元的尺寸改变时调用此函数，在函数中得重新设置当前绘图现场，视点和投影。

GLwNinputCallback 其回调函数为 input( )。此函数为响应事件而进行有关操作。这在程序中是一个很重要的部分，它能响应各种事件类型，如键盘，鼠标，暴露等事件。这里，在 switch...case 语句下，没有加入一些事件处理函数，读者可将一些功能模块，如响应鼠标指点画图等，挂在适当的位置。

对一个新创建的 GLwDrawingArea 对象元，必须为 GLwNginitCallback、GLwNexpose Callback 和 GLwNresizeCallback 注册回调函数，它才能实现；而为了使它有所活动，给 GLwNinputCallback 注册回调函数十分重要。

注意，在程序编译和连接时，要同时连上 Motif 与 OpenGL 相关的库。

6. 结论及探讨

在这里，我们给予您一个图形软件开发的框架和一些基本的思路。

以上是针对单窗口绘图，对于多窗口绘图(即几个不同绘图区域能同时显示不同的图形)，我们只要把握到以下原则：在绘图时，只能有唯一的当前绘图现场。这样，我们就可以频繁使用 glXMakeCurrent( ) 或 GLwDrawingAreaMakeCurrent( ) 来解决问题。

在 Motif 与 OpenGL 混合编程中，除了利用 Xt 工具箱及对象元外，还可以使用与 XLib 函数库有关的操作来处理。

参考文献

[1] OpenGL Porting Guide

[2] OSF/Motif Programmer's Guide , Open Software Foundation

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

(来稿时间：1997 年 11 月)