一个面向对象的计算机图形系统的实现*

李文辉　庞云阶　全炳哲

（吉林大学计算机科学系　长春　130023）

摘要　本文介绍了作者设计并实现的一个面向对象的计算机图形系统. 由于系统采用了面向对象的系统设计思想，使用户可以通过高层概念来设计对象模型，且模型设计直观、自然，避免了在传统图形系统中，要求用户使用低层图形概念进行建模. 同时也保证了系统的易扩充性和易复用性，使本系统真正成为图形应用支持系统. 用户只要加入适当的领域知识，本系统很容易扩充为一个面向领域的图形系统.

关键词　对象，面向对象，计算机图形系统，约束.

面向对象方法是从 80 年代中期开始被广泛、深入研究的一种新的软件开发方法. 它与其他软件开发方法的主要区别在于：它是以数据为中心进行软件开发活动的，而基于功能分解技术的结构化设计方法是以功能为中心进行软件开发活动. 一般认为基于功能分解技术的软件开发方法有 4 点不足：(1) 难以适应系统进化中的变动. (2) 系统由单个功能刻画. (3) 它着眼于功能，对数据结构考虑不够. (4) 对软件构件的易复用性未予足够重视.

所以，它不适于开发大型软件，但对于小型软件，则不失为一种有效的方法，它能更直接、快捷地达到目标. 面向对象技术的核心是数据抽象和继承，它着重考虑问题空间和解空间的“自然”对应性、软件的可扩充性及可复用性.

计算机图形学与面向对象的方法的结合是非常自然的，它易于将人的思维模型平滑而自然地转换成图形系统的语言结构. 现在，已经有了很多有关面向对象图形学的研究，这些研究可以分成以下几个方面：

* 面向对象的图形库的研究. GraphPak[1]是一个可以提供 2D 图形对象的面向对象系统，这些对象可以被结合以形成更复杂的图形对象. Williams 在文献[2]中给出了一个可以提供 3D 图形对象的面向对象系统.

* 面向对象的用户界面研究. 图形是用户界面的一个很重要的组成部分，图形用户界面也是当前编程环境中必不可少的部分. 面向对象和用户界面的设计有着某种自然的联系，面向对象可以把用户界面和应用计算的代码分开，提供更有效的用户界面.[3]

* 本文研究部分得到国家教委留学回国人员科研基金资助. 作者李文辉，1961 年生，讲师，主要研究领域为计算机图形学，人工智能. 庞云阶，1939 年生，教授，主要研究领域为计算机图形学，人工智能. 全炳哲，1961 年生，副教授，主要研究领域为面向对象技术，程序自动化，人工智能.

本文通讯联系人：李文辉，长春 130023，吉林大学计算机科学系

本文 1995－07－17 收到修改稿

* 以面向对象的方法来实现某一特别的功能.

* 用面向对象的思想来实现图形标准，Peter Wisskirchen[4]很早就开始进行这方面的研究，设计了一个面向对象的图形系统 GEO++.

* 用面向对象的方法来实现的图形系统. 本文我们要介绍的面向对象的图形系统 OOCGS (object-oriented computer graphics system)就是一个例子.

1 OOCGS 系统的结构

OOCGS 系统是一个面向对象的图形系统，它可以支持广泛的图形应用，由于它采用了面向对象的技术，就使用户在进行图形设计和图形操作时更加直观和自然. 同时，利用面向对象技术中的继承机制可易于实现系统的扩充.

在面向对象的系统实现中，最关键的问题就是如何定义和设计系统中的各种类及把这些类组成相应的类层次结构. 在本系统中，我们主要定义了 3 个重要的类，分别为：应用类、图形类和涂抹类. 每个类都有一些相应的子类. 子类划分的依据是针对不同的类的特点而有所不同的. 应用类的划分是依据应用对象的特点来进行的. 比如说，当我们来表示一个风景造型时，可以把此应用类划分为如图 1 所示的几个子类：

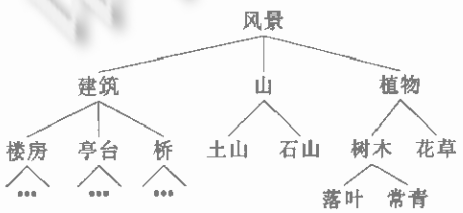

图1 一个应用类的类层次结构

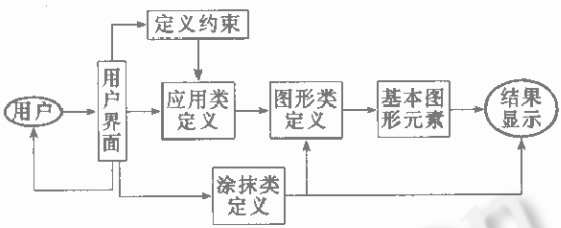

图2 OOCGS系统的结构

图形类则是以图形的表示方法来划分的，同一类的图形对象具有相同的数据表示形式. 例如，我们可以用点表、线表、面表和解析的方式来表示图形，那么，依据不同的表示方式，则可分为不同的图形类.

涂抹类是用来对应用对象进行涂抹显示的类，它也有一些子类，每一个子类都代表一个涂抹方式，比如 PhongRend 是一个以 Phong 为模型的涂抹方式. 本系统依据不同的要求，而设计了很多的涂抹方式，即涂抹子类，它们的根类是 Renderer.

上面这 3 种主要的类，我们在后面还要详细地讨论. 有这 3 种主要的类就组成一个对应用系统的模拟，整个系统的结构如图 2 所示.

本系统是使用 Smalltalk-80 语言在 Sun 工作站上实现的，下面我们将详细讨论本系统的部分实现.

2 应用类及图形类的定义和实现

应用类是本系统的抽象程度最高的类，它主要表示用户的特殊的应用对象，它是用户数据的载体. 用户的具体数据和要求，就是由此类的对象来传给系统的. 虽然，各种应用的对象不尽相同，但它们也具有很多共同的属性和方法. 因此我们设计了一个最高的应用类 ApplicationObject，它的定义如下：

```
Object SubclassNames: #ApplicationObject
        instanceVariableNames: 'render,listobject,material,jointlist'
        classVariables: ' '
        pooloictionaries: ' '
        category: 'Graphics System'

ApplicationObject methods:

        insert: aGraphicsObject.
        getGraphicsObject:aInteger.
        setMaterial: aMaterial.
        getMaterial.
        setRender: aRender.
        getRender.
        setJointlist:aJointlist.
        getJointlist
        ... ...
```

我们来看一下这个类的定义，首先，它有几个变量，分别为：material，render，listobject及jointlist. render指示此客体被涂抹的模式，listobject是一个表，它用来指示组成此应用对象的所有图形对象. jointlist也是一个表结构，表中的每一个元素都是Joint Class的实例. Joint Class描述的是客体间各组成部分的关系，它指明了被关联的2个客体及被关联的类型，还有满足这种关联的参数的取值范围. 我们还为这个类定义了一些相应的方法来处理本类对象. 例如，app－Obj－1 insert: aGraphicsObject，表示给一个应用对象app－Obj－1插入一个图形对象aGraphicsObject.

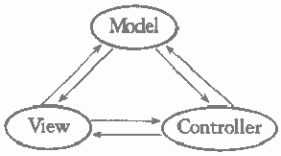

图3 Smalltalk的MVC模式

前面定义的类ApplicationObject是对所有的应用类的抽象定义，但对于每一个具体的应用，我们可以定义新的应用子类，它们继承了超类的所有特性，同时也扩充了必要的方法. 我们可以把应用对象的设计看成是一个建模的过程，我们利用Smalltalk中的MVC模式来使模型Model和视口View及控制Controller联系在一起，共同作用，来实现系统的功能. MVC模式提供了Model，View和Controller之间很灵活的通讯，如图3所示.

它的每条路径都有确定的含义. 这种使建模、显示和控制分开设计的方法，将使用户在设计时能更好地集中精力于Model的设计，因为View和Controller是有现存的类的，用户只要选择使用即可.

图形类是用来模拟应用类的重要部分，是系统中具有几何意义的类，它也是连接应用类和涂抹类的桥梁，它的功能是把应用对象的数据过渡转化为涂抹类可以接受的数据形式.

每一个应用对象都有一个对应的图形对象，这一图形对象是它的相对应的应用对象的图形体现. 用户可以方便地更换应用对象的图形对象，以选择更为准确的图形对象来描述应用对象. 我们来定义名为GraphicsObject的图形类，定义如下：

```
Object SubclassNames: # GraphicsObject
        instanceVariableNames:' geometry,material,trmatrix,render'
        classvariableNames:' '
        pooloictionaries:' '
        category:' Graphics System"

GraphicsObject methods:
        initial.
        setGeometry: aGeometry.
        setMaterial: aMaterial.
        setTransformation: aTransformation mod: aInteger.
        setRender: aRender.
        updateGeometry: aGeometry mod: aInteger.
        ... ...
```

在类 GraphicsObject 中，几个事实变量分别有以下含义：geometry 用来指示一个几何对象；material 用来指示本图形对象的材料；trmatrix 代表对此图形对象进行变换的变换矩阵；render 给出了本图形对象的涂抹方式. 同时，也定义了几个相应的方法来处理图形对象. 由于有了这个一般的图形类的定义，要想生成一个图形对象则可很容易地用类来产生. 例如：

```
aGra－Obj－1 ← GraphicsObject new.
aGra－Obj－1 initial.
```

则产生一个基本的图形对象 aGra－Obj－1，然后可对其设置一些必要的数据，如：

```
aGra－Obj－1    setGeometry: aGeo－1.
aGra－Obj－1    setRender: aRender－1.
aGra－Obj－1    setLight: aLight－1.
aGra－Obj－1    setMaterial: aMaterial－1.
aGra－Obj－1    setTransformation: aScale－1 mod:3.
```

当然，在方法中所使用的对象数据都是其对应类的实例，这些对象必须在被使用前已被赋予相应的数据. 当一个图形对象已被设置完毕之后，则可利用下面的方法把其连到对应的应用对象上：

```
app－Obj－1    setGraphics: aGra－Obj－1.
```

如果要改变一个应用对象的图形对象的信息，只要再把一个新的图形对象连到它自身就可以实现了.

前面给出了一个具有一般性的图形类的定义，它是图形类的总根，对于不同的图形对象，可以定义不同的图形类，但它们必须是 GraphicsObject 的子类，且继承了超类的所有特征，当然也可能由于自身的特点，而具有特殊的变量和方法，也可修改它所继承的方法，这在面向对象的环境中是非常容易实现的.

3　Render 类的定义和实现

类 Render 是一个用于涂抹的类，它的作用是用其定义的方式来描绘应用对象，当然，这

个类是不能直接处理应用对象的，它所能处理的是图形类的对象. Render 是所有涂抹子类的超类，每个子类就是一种涂抹方式，那么，每当要增加一种新的涂抹方式，只需要创建一个 Render 的子类，定义相应的方法，再加上它从超类所继承来的方法和变量，则构成了一个新的涂抹子类. 本系统现有的涂抹方式，依据不同的需要，主要分为几种，比如，对真实感要求很高的涂抹方式、一般要求的涂抹方式、线框方式、实时涂抹方式.

实际上，并不是对任意的图形对象，涂抹类都能处理，那么，就存在着一种把一般的图形对象转化为涂抹类可以接受的数据形式的转化机制. 我们定义了一些涂抹类可以接受的最基本的图形元素，称其为涂抹基元. 那么对于一般的图形，我们可以把其分解为一些涂抹基元的组合，分别对其进行涂抹处运. 类 Render 的定义可简示如下：

```
Object SubclassNames: #Render
instanceVariableNames:' applobj,light,viewpoint,viewdirection,window'
        classvariableNames:' '
        pooloictionaries:' '
        category:' Graphics System"
Render methods:
        initial.
        setApplobj:aApplobj.
        setLight: aLight.
        setViewpoint:aPoint.
        setViewdirection:aVector.
        setWindow:aWindow.
        pick.
        render.
        ... ...
```

这个定义包含了以下几个内容，它有一个 ApplicationObject 类的实例，这个实例作为要被涂抹的对象；同时它也定义了一个视点及视线方向；另外还有光线的信息. 因为要把被涂抹的对象显示在一个窗口中，所以每一个 Render 都有一个窗口信息，由于一个对象可以被不同的 Render 的实例来进行处理，那么，同一个对象就可以通过不同的涂抹特性得以在不同的窗口中显示. 这里的 Window 类是对窗口数据及操作的描述，它的方法主要有：resize，move，iconize，pop－up 和 pop－down 等. Render 中定义的 pick 方法是允许用户拾取应用对象中的各子组件，这对于各组件的进一步定义和修改及显示都是有意义的.

下面看一下与涂抹类相关的另一个类 Material 的定义：

```
Object SubclassNames: #Material
instanceVariableNames:' specular,diffuse,shininess,transparence,ambient'
        classvariableNames:' '
        pooloictionaries:' '
        category:' Graphics System"
Material methods:
        initial mode:anInteger.
        setSpecular: aSpecular.
```

```
setDiffuse:aDiffuse.
setShininess:aShininess.
setTransparence:aTransparence.
setAmbient: aAmbient.
... ...
```

这个类定义了对象的材料特性，比如金属、塑料或木质材料等. 只要改变不同的参数就可以实现. 同时，系统中也预先定义了一些常用的材料，通过调用方法 initial mode:aNumber 就可得到，这里，整数 aNumber 用来标识已定义的材料，它的范围是 0～20.

4　对象模型中约束的应用

模型中的约束，就是指模型中各部分之间应该保持的关系. 在模型的定义中引入约束是很有意义的，它将提高模型定义的一致性及有效性，且可以使模型的定义更简单.[5]因为设计者只需给出模型各组成部分的关系的描述，系统中的解约束模块将会被启动，以求得满足设计要求的模型状态. 由于用户不用进行具体的设计，就使得设计过程更为方便. 即用户可以使用高层几何概念来设计模型，而不是用各个实例的具体位置、方向、尺寸等数据来定义模型的组成.

4.1　几何结构支架

在本系统中，一个应用对象可以由一些其他对象组成，我们对这些对象的组成施加一定的约束，以达到模型的要求. 这里，约束是定义在各对象所在的坐标系之间的，而不是直接定义于各对象之间. 这样就使得解约束技术更为简单且更适于交互.

系统的建模过程，实际上是生成图形对象的一个几何结构的过程. 在我们的系统中，图形对象是由一些被称作体素的基本几何体来构成的，如何安排这些体素呢？我们是通过定义一个几何结构支架来实现的. 几何结构支架，就是空间的一个几何结构，它由一些节点组成，每个节点代表一个体素的坐标系原点. 其中有一个为根节点，其它的节点是相对于根节点定义的. 在世界坐标系中的一个变换，则是由一个节点到根节点的全部中间变换的级联而得到的. 这种几何结构支架，不仅被用作内部数据结构，同时也可被显示为 3D 的结构.

这种几何结构支架有 2 方面的含义：即拓扑和几何意义. 它的拓扑结构是由此结构支架的构成定义的，即支架中各节点的相互连接的方式. 拓扑结构是在产生支架时定义的，但也可以在以后通过对支架的结构的编辑而进行修改；它的几何意义，主要是指节点在世界坐标系中的位置、方向及标量值. 一个几何结构支架 GeoFrame 的定义如下：

```
Object SubclassNames: #GeoFrame
instanceVariableNames:' root,numnode'
  ... ...
GeoFrame methods:
initial.
select x: x1 y: y1.
append: parentnode.
delete: aNode.
draw.
```

```
getRoot.
relink old: node1 new: node2.
    ... ...
```

4.2 约束的定义

约束作为系统中的一个类而存在,具体的约束是约束类的实例.约束类可以定义如下:

```
Object SubclassNames: #Constraint
instanceVariableNames:' node,referencenode,value,type'
... ...
Constraint methods:
initial node:node1 referen: node2 type:integer1 value:real1.
setConstraintnode:node1.
setReferencenode:node1.
setType: aType.
setValue: real1.
evaluate.
display.
... ...
```

这个定义的含义为:对定义中的 node 进行约束,type 是指约束类型,它相对于某一个节点被定义,这个节点就为 referencenode,这里的 value 定义一个值,但它只对一些约束有意义.

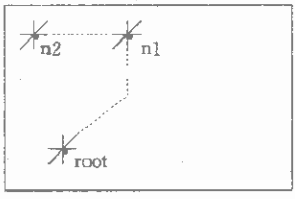

图4 一个受束的几何结构支架例子

以上是约束的内部数据结构的定义,但在使用时,用户不用直接对程序进行编辑,只要通过人机界面以图形的方式直观地定义约束,然后,相应的内部结构定义由系统产生.例如,在图 4 中的 geoFrame 中:

当用户想使 n1. y_pointto→root 时,则可以用鼠标选择节点 n1,然后在菜单中选择 y_pointto 时约束,并选择为参考点,则定义了一个约束.此时,这个约束将以程序码的形式加到模型的程序描述中,作为被束节点的属性.

本系统定义了几类基本的约束,这些约束可以用很简单的方法来求解.对于复杂的约束,我们可以把其分为一些基本约束的组合,即对同一节点施以不同的约束.例如,要使一个点在一个已知的 block 内部,那么,这可由 6 个约束组成,即 2 个定义它的 X 的最大、最小值,另 2 个定义它的 Y 的最大、最小值,最后 2 个定义它的 Z 的最大、最小值.

4.3 约束求解

在对约束的定义中,已经定义了一个方法 evaluate,它就是用来求解约束的,一些具体的约束类都是它的超类 Constraint 的子类,这样,我们把对每个具体类型约束的求解放到各具体的类的定义中去.开始解约束时,一般从模型的根节点开始,那么,全部其它的节点则以用户定义的顺序求解.一个节点的 3D 变换是由它的父节点的 3D 变换和此节点的局部变换级联得来的.约束的求解顺序依赖于约束自身的阶数,对于高阶约束,即它是约束的约束,不能给出一个唯一的答案,它一般是约束了一个范围.如 xy—distance 约束,则可以有许多的

情况满足此约束.对这类约束的求解,有2个原则:(1) 延缓求解,(2) 最小化扰乱.

延缓求解意味着:一个高阶约束的求解,直到被应用到此节点的全部低级的约束已经被求解了,再去解此高阶约束的解.最小化扰乱,就是在多解情况下,用户选出一个相对于父节点有最小变换的节点位置.

在几何结构支架中的全部节点都被赋值后,系统开始计算附着在节点上的对象.

上面求解约束的方法,实际上是采用了已知状态的传播技术,所以,具有求解速度快的特点,但如果在约束网络中出现环时,这个直接的求解方法则不能奏效,这时就要求助于数值方法 Relaxation 来求解.

4.4 一致性及有效性的保护

为了保证系统的正确使用,也为了方便对系统不熟悉的用户对系统的使用,我们利用约束为系统设计了一个保护神(Demon),它可以及时阻止用户的误操作,并给出一定的提示.具体来讲,它可以保证:① 值的范围的有效性;② 数据类型的正确性;③ 在交互设计过程中,由已知的知识来保证系统的一致性.

例如,在我们的图形系统中,有这样的保护措施:

(1) 在某一最大、最小角度范围内的旋转及以 RGB 形式给出的颜色值的范围 0.0～1.0.

(2) 对参数的赋值的类型的控制以保证输入正确的数据类型.这对交互式系统的应用是非常重要的,阻止用户输入错误的数据类型,以保护系统的正确运行.

(3) 保护系统中各层之间的一致性,比如当我们在图形方式下对模型进行了交互操作,那么,模型便得到了修改.同时,这种修改也要传递到模型的程序描述中去,以保持各种表示的一致性.同样,反方向的修改也要保持统一.

(4) 当模型被交互定义及修改后,它对应的程序描述应具有一致性,即能够对整个程序进行分析,以找出不一致的部分并调整之.

5 结　论

OOCGS 系统是采用面向对象技术的应用图形支持系统,具有比一般的标准图形包提供更多的应用工具的特点,使用户可以在应用层来定义应用模型.同时,它又不是针对某一应用设计的,所以,它也有很大的应用范围.

由于采用了面向对象程序设计思想,使系统的扩充更为方便、模型的定义更为直观.在系统中加入约束技术,使得模型的描述更加准确、简便,且可以使系统保持一致.当然,本系统还在不断的完善中,下一步要对人机界面给予更多的关注.同时,对约束的求解技术也在不断的探索中,我们也正在研究在系统中加入动画模拟的功能.

致谢　这一研究的部份工作是本文第一作者在德国 Bremen 大学进修期间完成的.在系统的研制过程中,得到了 F. Nake 教授的直接指导和帮助,Dr. Hans, Mr. Ecke 对本文的工作也提出了很多宝贵的建议,在此一并表示感谢.

参考文献

1 Craighill Nancy Knolle, Fong Martin W. GraphPak, a 2D graphics class library. In: Computer Graphics Through Object—Oriented Programming, ed. Steve Cunningham, 1992.

2 Williams Peter. The graphics class library for the visualization workbench. The Center for Supercomputer Research and Development, the University of Illinois at Urbana—Champaign, Internal Report, 1991.

3 Myers B A, Giuse D A, Dannenberg R B *et al*. Garnet: comprehensive support for graphical, highly interactive user interface. IEEE Computer, November 1990.

4 Wisskirchen Peter. Object—oriented graphics. Springer—Verlag, 1990.

5 Emmerik M J G M van. An interactive graphical system for modeling with 3D constraints. In: Proceedings Computer Graphics International'90, Spring—Verlag, 1990.

6 Borning Alan. Defining constraint graphically. In: the CHI'86 Proceedings, 1986.

AN OBJECT—ORIENTED COMPUTER GRAPHICS SYSTEM

Li Wenhui Pang Yunjie Quan Bingzhe

(*Department of Computer Science Jilin University Changchun* 130023)

Abstract This paper introduced an object—oriented graphics support system. The authors used object—oriented concept to design this system, so that one can design graphical application model at high level of abstraction and translate his meaning into a model maturely. The system is designed to be easily extensible and reuse. By adding some special knowledge about some area, the user can tailor the system into a more special one.

Key words Object, object—oriented, computer graphics system, constraint.