УДК 004.045

А.П. ПОБЕГАЙЛО

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ОПИСАНИЮ ПРОЕКТИВНОЙ ГЕОМЕТРИИ

Object-oriented description of geometric objects in two-dimensional projective space is presented. The proposed interfaces can be used in object-oriented computer graphics systems. Classes of geometric transformations and objects are described using C++ programming language.

Прогресс в технологии программирования привел к объектно-ориентированному программированию. Некоторые аспекты его использования в компьютерной графике рассмотрены в работах [1–6]. Примерами развитых объектно-ориентированных графических систем являются OpenGL и DirectX, а также объектно-ориентированный интерфейс Java 3D API. Проблема заключается в том, что в существующих системах компьютерной графики применяется примитивный подход к описанию геометрических объектов, в котором используются только конкретные геометрические объекты, например векторы и матрицы, и нет концептуального описания интерфейсов геометрических объектов. Целью данной статьи является строгое и математически обоснованное объектно-ориентированное описание геометрических объектов, которое базируется на классической теории групп Ли и может быть использовано в объектно-ориентированных системах компьютерной графики. В данной статье классы геометрических объектов описаны с использованием языка C++. Интерфейсы геометрических объектов описаны полностью. Реализация методов опущена, так как алгоритмы геометрических преобразований хорошо известны из аналитической геометрии. Для примера рассмотрено объектно-ориентированное описание проективной геометрии на плоскости.

Онтологический подход к проектированию программных систем

В настоящее время при проектировании программных систем необходимо разрабатывать довольно сложные протоколы и интерфейсы, что требует точного определения абстрактных понятий из предметной области, т. е. присутствует некоторый философский подход к проектированию систем. Используя философские термины, которые, впрочем, сейчас также широко применяются и в исследованиях по моделированию систем, можно сказать, что процесс разработки программных систем есть комбинация онтологического и эпистемологического подходов [7, 8]. Другими словами, модель системы есть синтез системы классов, описывающих абстрактные и реальные объекты прикладной области.

Онтологический подход к разработке программной системы подразумевает построение ее концептуальной модели. *Онтология* означает систему концепций и отношений между ними, причем определение этой системы носит декларативный характер. Каждая концепция рассматривается как сущность, которая обладает свойствами и связями с другими концепциями. Однако заметим, что концепция не подразумевает способ ее реализации, она может быть реализована как абстрактным, так и реальным классом.

В то же время проблематично построить реальную систему, которая описывается только концепциями, или интерфейсами. Как правило, при проектировании системы также прорабатываются реальные классы, которые не содержат виртуальных функций. Можно сказать, что их проработка требует применения эпистемологического подхода к моделированию системы.

С учетом этих замечаний вначале рассмотрим несколько общих свойств объектов, которые используются при построении любых концепций.

Первое свойство заключается в том, что все объекты имеют структуру. С этой точки зрения они делятся на простые и сложные, причем простые объекты являются элементарными, из которых строятся сложные объекты.

Второе свойство описывает активность объектов, на основании чего объекты классифицируются как активные и пассивные. Активные объекты действуют на пассивные и вызывают изменение их состояния.

Следует отметить, что два данных свойства являются относительными, т. е. на разных уровнях абстракции или в разных контекстах один и тот же объект может иметь противоположные свойства.

Третье свойство описывает отношение объектов к реальному миру, который они моделируют. С этой точки зрения объекты делятся на конкретные и абстрактные, причем конкретные имеют структуру и обычно описываются с использованием предопределенных типов данных. Предполагается, что абстрактные объекты определяются только своими свойствами и об их структуре нельзя сказать ничего определенного. В объектно-ориентированных языках программирования абстрактные объекты обычно специфицируются только интерфейсами и не имеют конкретной реализации, например, в языке C++ такие объекты описываются абстрактными классами, содержащими чисто виртуальные функции.

Онтологический подход к проектированию системы классов, описывающих геометрию, совпадает с определением геометрии, которое было дано Ф. Клейном в его Эрлангенской программе. Согласно Ф. Клейну, геометрия определяется как группа преобразований некоторого пространства и множество объектов этого пространства, инвариантных относительно этой группы преобразований. С. Ли разработал теорию групп непрерывных преобразований, которые сейчас и называются группами Ли.

Классификация геометрических преобразований

Дадим определение группы Ли [9–11]. Понятие группы Ли является обобщением идеи непрерывного геометрического преобразования, имеющего групповые свойства. Группой Ли называется дифференцируемое многообразие G, которое одновременно является группой с гладкими операциями умножения

$$G \times G \to G : (g_2, g_1) \mapsto g_2 g_1$$

и инвертирования

$$G \to G : g \mapsto g^{-1}$$

элементов.

Нейтральный элемент группы Ли G обозначается через e.

Геометрические преобразования действуют на точки, принадлежащие некоторому множеству. Предположим, что это множество является гладким многообразием, которое обозначим M. Действием группы Ли G на гладкое многообразие M называется гладкое отображение

$$G \times M \to M : (g, x) \mapsto x,$$

которое удовлетворяет следующим условиям:

$$(e, x) = x$$

для любых $x \in M$ и

$$(g_2, (g_1, x)) = (g_2 g_1, x)$$

для любых $x \in M$ и $g_2, g_1 \in G$.

Теперь перейдем к классификации геометрических преобразований. В данной статье рассматривается классификация коллинеарных преобразований проективной плоскости. Поэтому в качестве многообразия M возьмем двумерное проективное пространство RP^2, а в качестве группы преобразований рассмотрим группу проективных преобразований $GP(2, R)$.

Принимая во внимание замечания из предыдущего раздела, определим абстрактную геометрию на проективной плоскости двумя классами: классом точек, которые задаются однородными координатами, и абстрактным классом точечных преобразований. Приведем интерфейсы этих классов.

```
struct Point
{
   double wx, wy, w ;
   // конструкторы
   Point ( ) { }
   Point (const double& _wx, const double& _wy, const double& _w) ;
   // операторы, члены класса
   Point& operator += ( const Point& ) ;
   Point& operator –= ( const Point& ) ;
   Point& operator *= ( const double& ) ;
};

// операторы, не члены класса
Point operator + ( const Point&, const Point& ) ;
Point operator – ( const Point&, const Point& ) ;
Point operator * ( const double&, const Point& ) ;
Point operator * ( const Point&, const double& ) ;

class PointTransformation
{
public:
   // деструктор
   virtual ~PointTransformation ( ) { }
   // оператор, член класса, преобразующий точку
   virtual Point operator ( )  ( const Point& ) const = 0 ;
};
```

Точки рассматриваются как простые пассивные объекты, которые являются базисом для построения всех сложных пассивных геометрических объектов, а класс точечных преобразований – как базовый класс для всех конкретных геометрических преобразований.

Теперь определим группы геометрических преобразований, каждая из которых определяет конкретную геометрию на проективной плоскости. Приведем иерархическую классификацию групп геометрических преобразований проективной плоскости, которые являются подгруппами группы проективных преобразований:

```
Проективные преобразования
└─ Аффинные преобразования
    ├─ Линейные преобразования
    └─ Эквиаффинные преобразования
        ├─ Деформации
        └─ Движения
            ├─ Вращения
            └─ Переносы
```

Здесь в иерархии подгруппы расположены ниже, кроме того, линейные преобразования содержат следующие подгруппы:

```
Линейные преобразования
└─ Деформации
    └─ Вращения
```

Используя данную классификацию, можно определить классы геометрических преобразований. Так, класс проективных преобразований плоскости определяется следующим образом:

```
class Projection: public PointTransformation
{
protected:
   // матрица однородных преобразований
  double  xx,  xy,  tx,
          yx,  yy,  ty,
          px,  py, sw;
public:
   // конструкторы
      .   .   .
   // деструктор
   virtual ~Projection ( )   { }
   // оператор, член класса, проектирующий точку
   virtual Point operator ( )  ( const Point& ) const;
   // инверсия проективного преобразования
   friend Projection operator~ ( const Projection& ) ;
   // композиция проективных преобразований
   friend Projection operator* ( const Projection&, const Projection&) ;
} ;
```

Аналогично могут быть определены классы других геометрических преобразований. Однако следует учитывать, что классы деформаций и вращений наследуются от двух базовых классов. Поэтому для этих базовых классов нужно применять виртуальное наследование, например,

```
class Motion: virtual public EquaffineTransformation
{
public:
   // конструкторы
      .   .   .
   // деструктор
   virtual ~Motion ( )  { }
   // оператор, член класса, передвигающий точку
   virtual Point operator ( )  ( const Point& ) const ;
   // инверсия движения
   friend Motion operator~ ( const Motion  & ) ;
   // композиция движений
   friend Motion operator* (const Motion  &, const Motion & ) ;
} ;
```

Замечание. Для каждого из классов наследников операторы члены и друзья класса реализуются более эффективно, учитывая структуру матрицы однородных преобразований.

Классификация геометрических объектов

Следуя определению геометрии, данному Ф. Клейном, можно сказать, что каждый класс геометрических преобразований определяет некоторый класс пассивных геометрических объектов, которые инвариантны относительно этих преобразований. Следовательно, класс геометрических объектов должен содержать функцию, параметром которой является допустимое геометрическое преобразова-

ние. Так, класс объектов, которые инвариантны относительно движений, определяется следующим образом:

```
class MovedObjects
{
  // деструктор
  virtual  ~MovedObjects ( )  { }
  // функция, член класса, изменяющая объект
  virtual void transformed ( const Motion& ) = 0 ;
} ;
```

Из этого интерфейса также следует, что объекты, которые допускают движения, также допускают переносы и повороты, так как эти геометрические преобразования являются наследниками движения. Класс конкретных объектов, инвариантных относительно движения, определяется как наследник абстрактного класса MovedObjects.

* * *

В статье представлено объектно-ориентированное описание геометрических преобразований и объектов проективной плоскости. Разработанные интерфейсы могут быть использованы в объектно-ориентированных системах компьютерной графики.

1. Alberti M.A., Bastioli E., Marini D. // The Visual Computer. 1995. Vol. 15. P. 378.
2. Silver D. // IEEE Comput. Graph. Appl. 1995. Vol. 15. P. 54.
3. Zaharia M.D. // Technical Report IAS-UVA-02-05. Informatics Institute. University of Amsterdam. The Netherlands. August 2002.
4. Fontijne D., Dorst L. // IEEE Computer Graphics and Applications. 2003. Vol. 23. P. 68.
5. Dorst L., Mann S. // Ibid. 2002. Vol. 22. P. 24.
6. Mann S., Dorst L. // Ibid. 2002. Vol. 22. P. 58.
7. Guarino N. // Formal Ontology in Information Systems. Proceedings of FOIS'98, Trento, Italy, 6–8, June, 1998. Amsterdam, 1998. P. 3.
8. Devedžić V. Understanding Ontological Engineering // Communications of the ACM. 2002. Vol. 45. № 4ve. P. 136.
9. Серр Ж.-П. Алгебры Ли и группы Ли: Пер. с англ. и франц. М., 1969.
10. Понтрягин Л.С. Непрерывные группы. 4-е изд. М., 1984.
11. Уорнер Ф. Основы теории гладких многообразий и групп Ли: Пер. с англ. М., 1987.

Поступила в редакцию 25.09.07.

Александр Павлович Побегайло – кандидат технических наук, доцент кафедры технологии программирования.