УДК 514.1

А. П. ПОБЕГАЙЛО

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ МОДЕЛИРОВАНИЕ СИМПЛИЦИАЛЬНЫХ КОМПЛЕКСОВ

Рассмотрены вопросы моделирования объектно ориентированных интерфейсов симплициальных комплексов для систем геометрического моделирования и компьютерной графики. Предложена классификация симплициальных комплексов, используемых в системах компьютерной геометрии и графики. На основе данной классификации разработаны интерфейсы симплексов и симплициальных комплексов, базирующиеся на шаблоне проектирования `Composite`, широко используемом в объектно ориентированном программировании для моделирования иерархических структур объектов. Исследованы ограничения на операции модификации предложенных симплициальных комплексов, для определения которых применяется эйлерова характеристика симплициального комплекса. Для моделирования интерфейсов симплексов и симплициальных комплексов используется язык программирования C++, являющийся практически стандартом для разработки системного программного обеспечения.

Ключевые слова: геометрическое моделирование; шаблон проектирования `Composite`; симплексы; симплициальные комплексы; эйлерова характеристика многогранника.

The paper presents an approach to object-oriented modeling of simplicial complexes. The classification of simplicial complexes, used in computer graphics and geometric design, is presented. Using this classification, the interfaces of simplicial complexes are developed. These interfaces are based on the design pattern `Composite` which is intended for developing interfaces of hierarchical structures in object-oriented programming. Constraints on operations for modification of simplicial complexes are considered. The constraints are defined using Euler characteristics of simplicial complexes. The proposed interfaces are described by means of the programming language C++ which is now widely used in systems of computer graphics and geometric desing.

Key words: geometric modeling; design pattern `Composite`; simplex; simplicial complex; Euler formula.

В геометрическом моделировании симплициальные комплексы используются для построения моделей геометрических объектов, состоящих из симплексов, и дальнейшей обработки этих геометрических объектов. Большое количество работ посвящено концептуальным вопросам, связанным с моделированием топологии геометрических объектов [1–3], а также алгоритмическим аспектам организации структур данных, предназначенных для хранения симплициальных комплексов [4–6]. Эти вопросы очень важны, так как определяют производительность системы геометрического моделирования. Но не менее важно создание архитектуры системы. Особенно это актуально при разработке объектно ориентированных систем, когда требуются абстрактные интерфейсы для взаимодействия объектов системы. Данная задача не может быть тривиальной, так как базируется на классификации объектов системы и определении абстрактных классов или интерфейсов, которые являются корнем дерева классификации. Работ, посвященных решению этой задачи, не так много, можно отметить некоторые работы [7–9], в которых для моделирования симплициальных комплексов предлагается использовать шаблоны языка программирования C++, а также систему геометрического моделирования ACIS [10], в которой симплексы определяются классами языка программирования C++, без применения абстрактных симплексов и симплициальных комплексов.

Представленный в настоящей работе подход к разработке объектно ориентированных интерфейсов симплициальных комплексов для систем геометрического моделирования отличается от других, во-первых, использованием шаблона проектирования `Composite` для моделирования симплексов и, во-вторых, построением иерархии симплициальных комплексов, в которой для контроля операций над симплексами комплекса используется эйлерова характеристика симплициального комплекса. Для моделирования интерфейсов симплексов и симплициальных комплексов применяется язык программирования C++, который практически является стандартом для разработки системного программного обеспечения.

Шаблон проектирования `Composite`

Для моделирования симплициальных комплексов используется шаблон проектирования `Composite` [11], модель которого показана на диаграмме классов (рис. 1). Этот шаблон предназначен для компоновки схожих объектов в древовидные структуры. Другим подходом к компоновке объектов в древовидные структуры является использование шаблона `GeneralTree`, который рассматривается в [12]. Концептуально этот шаблон эквивалентен шаблону `Composite`, но предполагает реализацию дерева с использованием связных списков.

Кратко опишем назначение классов, входящих в шаблон проектирования `Composite`. Абстрактный класс `Component` объявляет интерфейс для компонуемых объектов и предоставляет реализацию операций, общих для всех компонентов. Класс `Leaf` представляет узлы композиции, которые не имеют потомков, и определяет поведение таких узлов. Класс `Composite` хранит и определяет поведение компонентов, у которых есть потомки, а также реализует операции для их управления.

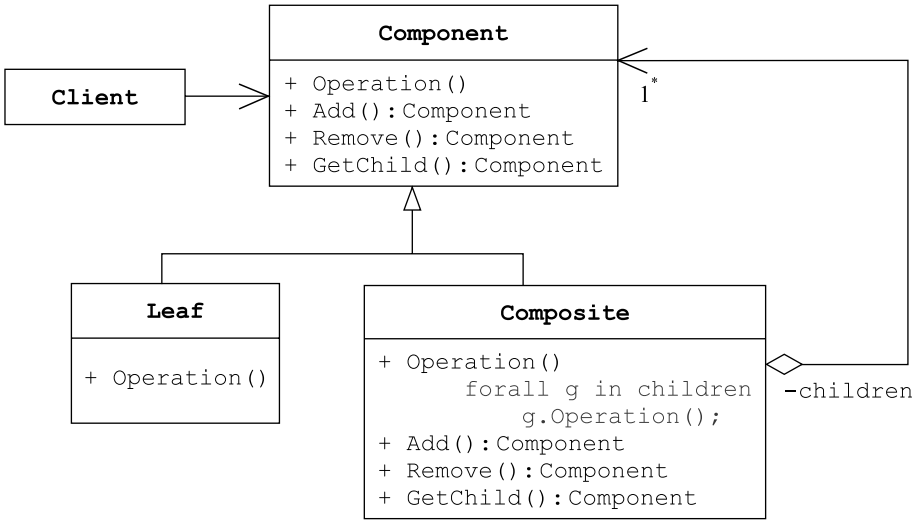

Рис. 1. Диаграмма классов шаблона проектирования `Composite`

Клиенты используют интерфейс класса `Component` для взаимодействия с объектами в составной структуре. Если получатель запроса – составной объект, то он обычно переадресует запрос своим потомкам, возможно выполняя некоторые дополнительные операции до или после его переадресации. Если получателем является лист, то он и обрабатывает этот запрос.

Моделирование симплексов

Применим шаблон проектирования `Composite` для определения программных интерфейсов симплексов. Для этого прежде всего дадим определение абстрактного симплициального комплекса [13, 14]. Абстрактным симплициальным комплексом K называется такая совокупность конечных множеств, что если множество $\alpha \in K$ и множество $\beta \subseteq \alpha$, то $\beta \in K$. Множество $\alpha \in K$ называется абстрактным симплексом, а каждое непустое подмножество множества α есть грань этого симплекса. Размерность абстрактного симплекса – число, на единицу меньшее количества его граней. Объединение всех множеств симплициального комплекса K называется множеством вершин этого симплициального комплекса.

Теперь для моделирования абстрактных симплексов нужно выбрать представление для множества вершин, на которых заданы эти симплексы. Для разработки программных интерфейсов возможны три варианта определения вершин симплексов: точки евклидова пространства, указатели и целые числа. В первом случае получим геометрические симплексы. Этот подход используется редко, так как требует слишком много памяти для хранения симплициальных комплексов и, кроме того, замедляет их обработку из-за медленных операций над действительными числами. Поэтому на практике обычно останавливаются на одном из двух оставшихся вариантов. Использование указателей подразумевает реализацию симплициальных комплексов при помощи списковых структур, организованных в деревья. А выбор целых чисел подразумевает, что результаты хранятся в структурах данных, доступ к элементам которых выполняется через индексы, – как правило, это массивы. Для организации симплексов очевиден выбор целых чисел, так как симплексы одной размерности имеют фиксированное количество граней. Кроме того, в настоящее время этот подход нашел широкое применение при моделировании высокопроизводительных структур данных для симплициальных комплексов [15, 16]. Заметим также, что для нужд компьютерной геометрии и графики достаточно рассмотреть абстрактные симплексы размерностей 0, 1 и 2, которые будут называться соответственно вершинами, сегментами и треугольниками по аналогии с геометрическими симплексами таких же размерностей.

Принимая во внимание вышесказанное, установим иерархию интерфейсов симплексов, используемых в компьютерной геометрии и графике таким образом, как это показано на рис. 2.

Здесь `Simplex` – это абстрактный класс, который является базовым для всех конкретных симплексов. С учетом того, что симплексы определяются на множестве целых чисел, этот интерфейс может быть найден следующим образом:

```
class Simplex
{
public:
      virtual int simplexNum() const = 0;
      virtual void setSimplex(const int&, const Simplex&) = 0;
      virtual const Simplex& getSimplex(const int&) const = 0;
};
```

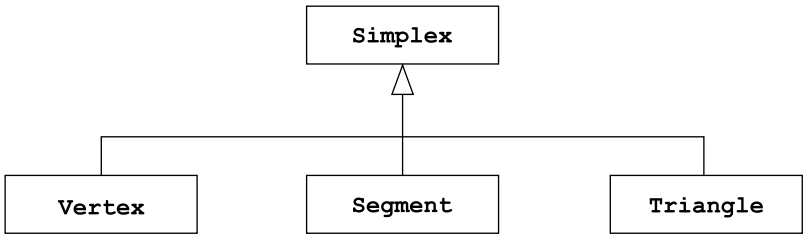

Рис. 2. Иерархия интерфейсов симплексов

Каждая функция класса `Simplex` имеет определенное назначение: функция `simplexNum` возвращает количество граней симплекса, `setSimplex` устанавливает грань симплекса, а `getSimplex` возвращает ссылку на грань симплекса.

Наследниками класса `Simplex` являются классы, обусловливающие интерфейсы конкретных симплексов. Для примера приведем интерфейсы классов для программирования вершин и сегментов. Интерфейсы для симплексов более высоких размерностей определяются аналогично интерфейсу класса сегментов. Интерфейс класса вершин устанавливают следующим образом:

```
class Vertex: public Simplex
{
     int idx;
public:
     virtual int simplexNum() const;
     virtual void setSimplex(const int&, const Simplex&);
     virtual const Simplex& getSimplex(const int&) const;
     virtual void setIdx(const int&);
     virtual const int& getIdx() const;
};
```

Класс `Vertex` содержит объекты, которые являются листами иерархии объектов, определяющих симплексы. Поэтому для этого класса определяются функции `setIdx` и `getIdx`, которые предназначены для работы с индексами, указывающими на индексированные точки евклидова пространства. Поскольку объекты этого класса не имеют вложенных симплексов, то наследуемые функции `setSimplex` и `getSimplex` обычно просто выбрасывают исключения. Теперь, например, определим интерфейс класса сегментов, который описывает нелистовые объекты иерархии симплексов.

```
class Segment: public Simplex
{
     Vertex v[2];
public:
     virtual int simplexNum() const;
     virtual void setSimplex(const int&, const Simplex&);
     virtual const Vertex& getSimplex(const int&) const;
     virtual void setSegment(const Vertex[2]);
     virtual void setVertex(const int&, const Vertex&);
};
```

Как видим, класс `Segment` содержит два индексированных объекта класса `Vertex` и дополнительные функции `setSegment` и `setVertex`, которые упрощают конструирование сегментов. Другие классы из иерархии симплексов определяются аналогично. В результате получим интерфейсы для единообразного иерархического построения симплексов через агрегацию симплексов, размерность которых на единицу меньше размерности конструируемого симплекса.

Моделирование симплициальных комплексов

С точки зрения структурной организации симплициальные комплексы отличаются от симплексов нерегулярной структурой. Симплициальный комплекс может содержать произвольное количество симплексов различной размерности. Кроме того, при конструировании симплициального комплекса симплексы могут как включаться в этот комплекс, так и исключаться из него. Принимая во внимание эти структурные характеристики, определим интерфейс абстрактного симплициального комплекса следующим образом:

```
class SimplicialComplex: public Simplex
{
public:
     virtual void insSimplex(const int&, const Simplex&) = 0;
     virtual void delSimplex(const int&) = 0;
};
```

Здесь функции `insSimplex` и `delSimplex` предназначены соответственно для включения симплексов в комплекс и для исключения их из него. Как видим, для определения интерфейса абстрактного симплициального комплекса используется интерфейс абстрактного симплекса `Simplex`, от которого он наследует функции для доступа к симплексам, уже включенным в комплекс.

Конкретные симплициальные комплексы наследуются от абстрактного класса `SimplicialComplex`. Иерархия конкретных симплициальных комплексов, применяемых в компьютерной графике, показана на рис. 3.

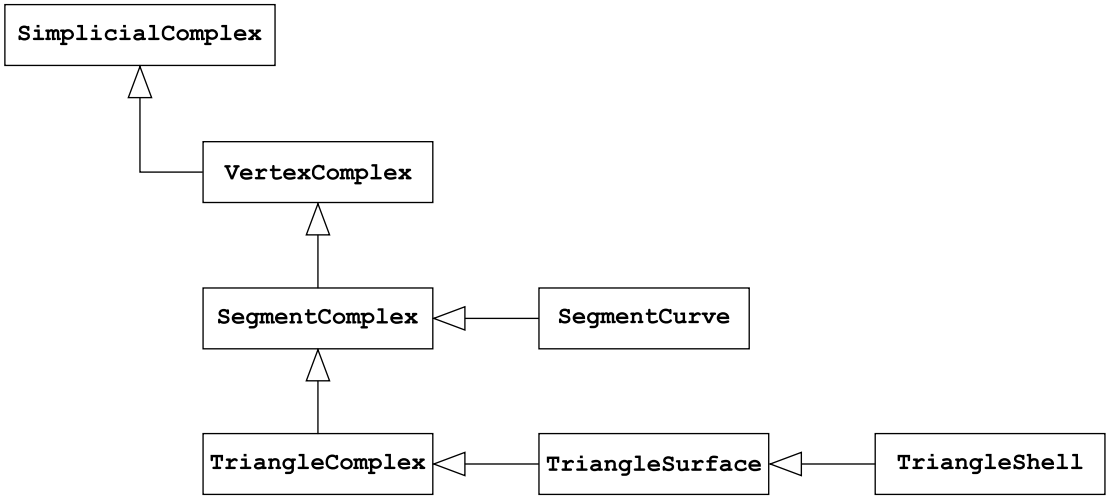

Рис. 3. Иерархия симплициальных комплексов

Поскольку компьютерная графика занимается только изображением геометрических объектов, то для моделирования твердых тел достаточно использовать их представление оболочками. Приведем объектно ориентированные интерфейсы симплициальных комплексов `VertexComplex`, `SegmentComplex` и `TriangleComplex`, предназначенные для моделирования соответственно комплексов, включающих вершины, вершины и сегменты, а также вершины, сегменты и треугольники.

```
class VertexComplex: public SimplicialComplex
{
public:
      virtual void insVertex(const int&, const Vertex&);
      virtual void delVertex(const int&);
      virtual void setVertex(const int&, const Vertex&);
      virtual const Vertex& getVertex(const int&) const;
};


class SegmentComplex: public VertexComplex
{
public:
virtual void insSegment(const int&, const Segment&);
      virtual void delSegment(const int&);
      virtual void setSegment(const int&, const Segment&);
      virtual const Segment& getSegment(const int&) const;
};


class TriangleComplex: public SegmentComplex
{
public:
      virtual void insTriangle(const int&, const Triangle&);
      virtual void delTriangle(const int&);
      virtual void setTriangle(const int&, const Triangle&);
      virtual const Triangle& getTriangle(const int&) const;
};
```

Заметим, что для краткости изложения эти интерфейсы содержат только не наследуемые от базовых классов функции.

Теперь нужно отметить важный момент, касающийся ограничений или условий, которым должны удовлетворять операции вставки и удаления симплекса в описанные комплексы. При модификации

комплекса вставляемые в него геометрические симплексы должны склеиваться по своим граням и не могут пересекаться между собой.

Интерфейсы остальных классов иерархии, а именно `SegmentCurve`, `TriangleSurface` и `TriangleShell`, отличаются от интерфейсов базовых классов только тем, что замещают виртуальные функции базовых классов, предназначенные для модификации симплициальных комплексов. Замещаемые виртуальные функции вставки и удаления симплексов в комплекс изменяют семантику этих операций. Для определения ограничений замещаемых функций используется эйлерова характеристика симплициального комплекса [17].

Для класса `SegmentCurve` вставка сегмента в симплекс выполняется лишь в том случае, если он граничит только с другим сегментом, уже вставленным в этот симплекс, что можно проверить, вычислив эйлерову характеристику ломаной линии:

$$\chi = k_0 - k_1,$$

где k_0 – количество вершин ломаной линии; k_1 – количество сегментов, из которых состоит эта ломаная линия. Очевидно, что для ломаной линии данная характеристика должна быть равна 1.

Для класса `TriangleSurface` вставка треугольника в симплекс выполняется только в том случае, если каждая из его сторон граничит не более чем с одним другим треугольником. Вставка треугольника в класс `TriangleShell` выполняется только в том случае, если эйлерова характеристика этого симплициального комплекса равна 2. Для односвязного многогранника эйлерову характеристику можно вычислить следующим образом:

$$\chi = k_0 - k_1 + k_2,$$

где k_0 – количество вершин многогранника; k_1 – количество сегментов, которые являются границами граней; k_2 – количество граней этого многогранника.

Таким образом, в статье рассмотрены вопросы моделирования объектно ориентированных интерфейсов симплициальных комплексов для систем геометрического моделирования и компьютерной графики. Предложенные интерфейсы базируются на шаблоне проектирования `Composite`, который используется в объектно ориентированном программировании для моделирования иерархических структур. Предложена классификация симплициальных комплексов для систем компьютерной геометрии и графики.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Weiler K. J.* Topological structures for geometric modeling : Ph. D. thesis : TR-86032. New York, 1986.

2. *Chen J., Akleman E.* Topologically Robust Mesh Modeling: Concepts, Data Structures, and Operations // Intern. J. Shape Model. 2000. Vol. 5, № 2. P. 149–177.

3. *Dalstein B., Ronfard R., Van de Panne M.* Vector Graphics Complexes // ACM Transactions on Graphics. 2014. Vol. 33, iss. 4. Art. № 133.

4. *Weiler K.* Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments // IEEE Computer Graphics & Applications. 1985. Vol. 5, iss. 1. P. 21–40.

5. *De Floriani L., Hui A.* Data structures for simplicial complexes: an analysis and a comparison // Proc. of the 3rd Eurographics Symp. on Geometry Proc. (Vienna, 4–6 July 2005). Vienna, 2005. P. 119–128.

6. *De Floriani L., Hui A., Panozzo D., Canino D.* A Dimension-Independent Data Structure for Simplicial Complexes // Proc. of the 19th Intern. Meshing Roundtable. Berlin ; Heidelberg, 2010. P. 403–420.

7. *Kettner L.* Using generic programming for designing a data structure for polyhedral surfaces // Computational Geometry. 1999. Vol. 13, № 1. P. 65–90.

8. *Fabri A., Giezeman G.-J., Kettner L., Schirra S., Schönherr S.* On the design of CGAL, a computational geometry algorithms library // Software: Practice and Experience. 2000. Vol. 30, iss. 11. P. 1167–1202.

9. *Shiue L.-J., Peters J.* A mesh refinement library based on generic design // Proc. of the 43rd annual Southeast regional conf. (Kennesaw, 18–20 March 2005). New York, 2005. Vol. 1. P. 104–108.

10. *Corney J., Lim T.* 3D modeling with ACIS. Coburg, 2002.

11. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб., 2007.

12. *Preiss B. R.* Data Structures and Algorithms with Object Oriented Design Patterns in C++. Wiley, 1998.

13. *Mankres J. R.* Elements of Algebraic Topology. Menlo Park, 1984.

14. *Понтрягин Л. С.* Основы комбинаторной топологии. М., 1986.

15. *Alumbaugh T. J., Jiao X.* Compact Array-Based Mesh Data Structures // Proc. of the 14th Intern. Meshing Roundtable (San Diego, 11–14 Sept. 2005). San Diego, 2005. P. 485–503.

16. *Aleardi L. C., Devillers O.* Explicit array-based compact data structures for triangulations // Proc. of the 22nd Intern. Symp. on Algorithms and Computation (Yokohama, 5–8 Dec. 2011). Berlin ; Heidelberg, 2011. P. 312–322.

17. *Wilson P. R.* Euler Formulas and Geometric Modeling // IEEE Computer Graphics & Animation. 1985. Vol. 5, № 8. P. 24–36.

Поступила в редакцию 05.11.2014.

Александр Павлович Побегайло – кандидат технических наук, доцент кафедры технологий программирования факультета прикладной математики и информатики БГУ.