УДК 004.92 DOI: 10.18698/2541-8009-2017-1-57

ПРЕОБРАЗОВАНИЯ ОБЪЕКТОВ 3D-ПРОСТРАНСТВА

Е.В. Копьев asadiR.KEV@gmail.com

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Алгоритмы компьютерной графики сегодня позволяют решать одну и ту же задачу несколькими способами. Эти алгоритмы можно ранжировать в порядке удаления от примитивов. Предложенная концепция позволит реализовать модульную библиотеку и эффективно вести разработку на любом уровне удаленности от примитивов. Такое решение ускорит написание кода, упростит отладку, даст возможность качественно сравнивать эффективности комбинированных подходов. Описан формат хранения 3D-моделей. Дано определение класса базисных объектов и их конструкторов. Определено множество составных функций. Рассмотрен ряд задач компьютерной графики. Приведены примеры использования данной концепции. Предложены идеи для дальнейших разработок

Ключевые слова

Машинная графика, алгоритмы компьютерной графики, 3D-редакторы, модульная библиотека алгоритмов, неструктурированные топологии, STL

Поступила в редакцию 08.12.2016

Во многих отраслях возникает необходимость редактировать объекты в трехмерном пространстве. Множество преобразований может варьироваться в зависимости от прикладной отрасли. Например, в медицине подобная технология может быть использована для создания планировщика хирургических операций.

В данной области такая технология позволяет создавать имплантаты, идеально подходящие для конкретного человека. При моделировании имплантатов возникает необходимость: создавать отверстия в телах, выполнять лепку, размещать текстурные метки на поверхностях и т. д.

Целью данной работы является описание концепции, позволяющей выполнять преобразования над объектами 3*D*-пространства на любом уровне удаленности от примитивов. Это значительно уменьшит сложность разработки программ, использующих такие алгоритмы, так как упростит отладку и оптимизацию.

Представления трехмерных объектов в виртуальном пространстве. В компьютерной графике непрерывные поверхности аппроксимируются дискретными, т. е. поверхности ставится в соответствие конечный набор фрагментов, называемых полигонами. Трехмерной моделью (телом) называют объем, заключенный в замкнутую поверхность. *STL* — это формат файла для хранения 3*D*-моделей на жестком диске. В *STL*-файле информация о модели хранится

в виде списка треугольных граней и их нормалей. Иногда вершинам треугольника также ставят в соответствие цвета. Таким образом, описывают поверхность модели.

Треугольная грань — это треугольник, лежащий в какой-то фиксированной плоскости в трехмерном пространстве. Ее нормаль вычисляется с помощью векторного произведения двух векторов, полученных из вершин грани путем обхода против часовой стрелки (по правилу правого винта). На одну и ту же плоскость в пространстве можно смотреть с двух сторон. Пусть имеется некоторая грань *ABC* с вершинами *A*, *B*, *C* и нормалью *N*. Лицевой стороной грани *ABC* будем называть ту, которая видна из точки $V = A + N$. Некоторые программные обеспечения отображают только лицевые стороны граней.

Импорт *STL*-файла подразумевает построение по нему определенной топологии. Здесь под топологией подразумевается способ хранения поверхности в оперативной памяти. Естественно, что топология подбирается исходя из требований конкретной задачи. Классическим примером задания топологии является полигональная сетка. Если в полигональной сетке может быть задан естественный порядок, позволяющий за константное время ссылаться на соседние полигоны, то ее называют структурированной. Например, сфера. В ней полигоны можно отсортировать по двум сферическим координатам ψ и φ. Предполагая, что полигоны хранятся в двумерном массиве *M*, соседние к $M[i][j]$ полигоны можно получить, изменяя *i* или *j* на единицу. Структурированная сетка позволяет эффективно хранить поверхности, однако с помощью нее можно задать лишь простые модели. Если такого естественного порядка не существует, то полигональную сетку называют неструктурированной. В ней явно хранятся ссылки на соседние полигоны. Таким образом, неструктурированная полигональная сетка представляет собой граф, вершинами которого являются грани, а ребрами отношение смежности.

Для представления дискретных геометрических моделей предполагается использовать неструктурированные полигональные сетки, состоящие из треугольных граней. Однако основные описанные идеи можно применить и к другим топологиям.

Базисные объекты. Базисный объект (БО) — это абстракция над элементарными геометрическими объектами, имеющими дискретное представление. Один БО может состоять из других. По этой причине имеет смысл говорить о классе (семействе) базисных объектов (рис. 1) *BOS (Base ObjectS)*.

БО выбирают так, чтобы можно было на высоком уровне преобразовывать объекты, минуя такие примитивы как точки или треугольники. Данное семейство будет строиться, начиная с множества точек (векторов) *PS (PointS)*. Из точек можно получить множество *LS (LineS)*, включающее в себя прямые, лучи и отрезки. Отрезок у которого начало и конец совпадают является точкой. Таким образом, $PS \subset LS \subset BOS$. С помощью отрезков можно задать произвольную ломаную (дискретную кривую) в пространстве. Множество таких кривых обозначается аббревиатурой *CS (CurveS)*. Дискретная кривая может быть прямой, поэтому $PS \subset LS \subset CS \subset BOS$. С помощью кривых задается множество *PLS (PLaneS)*, состоящее из плоскостей и полигонов, с помощью замкнутой кривой, лежащей в

одной плоскости полигон. Делая допущение о том, что любая плоскость — это дискретная кривая из двух непараллельных звеньев, можно утверждать, что $PS \subset LS \subset PLS \subset CS \subset BOS$. Из полигонов можно получить произвольную дискретную поверхность. Множество, состоящее из поверхностей и их фрагментов, обозначается аббревиатурой *MS (MesheS)*. Так как поверхность может состоять из вырожденных треугольников, то $PS \subset LS \subset PLS \subset CS \subset MS \subset BOS$ (1). Рассмотренные выше множества включают в себя как бесконечные объекты (луч, прямая, плоскость, поверхность и т. д.), так и конечные (отрезок, треугольная грань, фрагмент поверхности и т. д.). Далее множество с «индексом плюс» — это подмножество соответствующего множества, состоящее только из конечных элементов. Если обозначить множество замкнутых кривых за *CCS (Closed CurveS)*, то $PS \subset LS_+ \subset PLS_+ \subset CCS \subset CS_+ \subset MS_+ \subset BOS$. Очевидно, что только замкнутым поверхностям соответствуют модели, а поскольку поверхность замкнутая, то она конечная. Для любой конечной кривой можно подобрать замкнутую поверхность так, чтобы кривая лежала в этой поверхности. Пусть множество замкнутых поверхностей — *CMS (Closed MesheS)*, тогда $PS \subset LS_+ \subset PLS_+ \subset CCS \subset CS_+ \subset CMS \subset MS_+ \subset BOS$ (2).

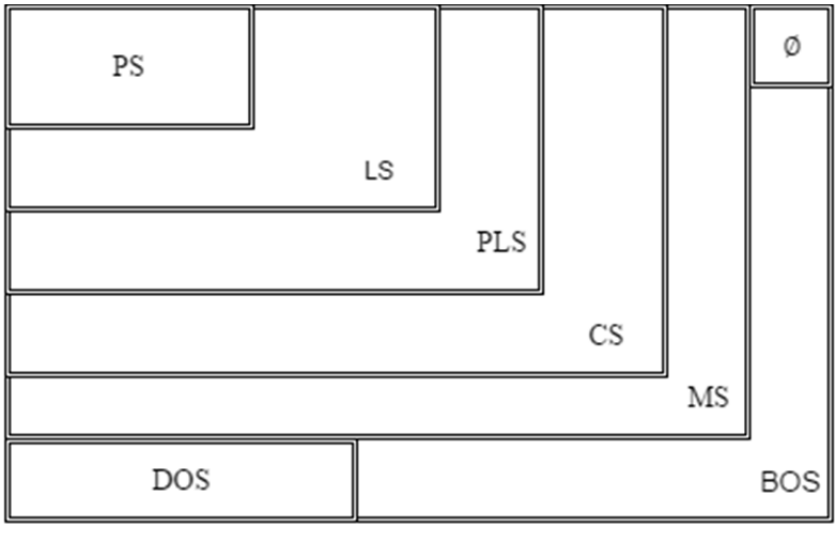

Рис. 1. Класс базисных объектов

Далее возникнет необходимость в результате преобразования БО возвращать объект данных (ОД). ОД представляет собой целочисленный кортеж. Множество ОД, обозначенное как *DOS (Data ObjectS)*, и пустое множество ∅, также включают в *BOS*.

Конструктор — это метод, позволяющий получить БО. Используя формулы (1) и (2), любому БО можно сопоставить кортеж точек. Подобного рода кортеж можно считать частью сигнатуры конструктора для рассматриваемого БО. Создавать БО из точек не всегда удобно, например, поверхность удобнее задавать треугольниками. По этой причине предлагается ввести конструкторы, позволя-

ющие генерировать БО из любых составных частей (т. е. из любых БО, находящихся в формулах (1), (2) левее, чем рассматриваемый объект). Конструктором объекта *X* уровня *Y* (обозначается как *CONS#X#Y)* называют конструктор, принимающий как аргументы элементы множества *Y*. Таким образом, используя введенную терминологию, можно назвать конструктор, генерирующий поверхности из точек, конструктором *MS* уровня *PS* (или *CONS#MS#PS*).

CONS#X#Y не обязательно один. Например, существует две реализации *CONS#PLS#PS*. Одна из них принимает три точки, другая два вектора. *CONS#LS#PS* может принимать на вход две точки, или точку и направляющий вектор. Если обозначить множество всех конструкторов как *CONS (CONStructors)*, то можно утверждать, что $BOS \subset CONS$.

Составные функции (СФ). В общем виде множество составных функций *CFS (Composite FunctionS)* задается следующим образом:

$$CFS = \{ f: (COS \cup BOS)^i \rightarrow (COS \cup BOS)^j \mid i, j \in [0, \infty) \}.$$

Частным случаем составной функции является конструктор БО, поэтому $BOS \subset CONS \subset CFS$. При этом можно ввести следующие критерии качества реализации этих функций: количество используемой памяти, алгоритмическая сложность, вычислительная устойчивость. Как правило, СФ используют другие СФ в процессе генерации результата. Множество СФ, используемых некоторой фиксированной СФ *F*, называется описателем этой функции *(DESCR#F)*, элементы которого обозначаются как g_i.

Для того, чтобы расширять *CFS* вводится множество *COS (Custom ObjectS)*. Изначально, *COS* — пустое множество. Пусть *Cl* — некоторый класс объектов. Если для него существует конструктор $f: (COS \cup BOS)^i \rightarrow Cl$, то *Cl* может быть добавлен в *COS*.

Очевидно, что для решения одной и той же задачи могут быть использованы разные подходы. Как правило, не существует универсального метода, который был бы лучше по всем критериям. Более того, для каждой конкретной задачи предметной области формулируются собственные критерии эффективности. По этой причине хотелось бы иметь модульную систему, позволяющую быстро сравнивать необходимые характеристики для разных видов решений и подбирать самые подходящие.

Пусть существует некоторый список задач компьютерной графики, каждой из которых можно поставить в соответствие некоторую составную функцию. Тогда множество допустимых реализаций этих функций, обозначенное как *ICFS (Implementations of CFS)*, является расширением множества *CFS*, т. е. $CFS \subset ICFS$. Пусть есть некоторая составная функция *F*, тогда для построения ее реализации *RF* нужно также реализовать все $g_i \in DESCR\#F$.

Резюмируя, можно утверждать, что любой БО — это СФ, описатель которой состоит из конструкторов. Таким образом, для упомянутой выше модульной системы предлагается использовать СФ в качестве элементарных абстрактных модулей. Их реализации могут быть весьма различными, и легко редактируются на любом уровне удаленности от примитивов.

Задачи компьютерной графики. Для наиболее полного описания картины предлагается рассмотреть некоторые задачи компьютерной графики (ЗКГ), которые могут быть реализованы на основе предложенной ранее концепции. Предполагая, что множества, описанные ранее, можно рассматривать как типы данных, установим соответствия между ЗКГ и СФ (рис. 2). Здесь «?» обозначает опциональный символ, «*» — итерацию, «|» — альтернативу, «DESCR»— описатель составной функции. Далее ссылки на СФ из рис. 2 будут указаны в круглых скобках.

```
1.(BOS) affineTransformation(BOS);
2.(PS?,DOS) segmentsIntersection(LS,LS);
3.(LS,DOS) planesIntersection(PLS,PLS);
4.(PS?,DOS) intersectionOfLineAndTriangle(LS,PLS); DESCR={2,9};
5.(OBBT) buildOBBT(MS);
6.(PS) intersectionOfLineAndMesh(LS,MS); DESCR={4,5};
7.(PS|LS|PLS|∅,DOS) intersectionOfTwoTriangles(PLS,PLS);
8.(CS*,MS*) intersectionOfTwoMeshes(MS,MS); DESCR={5,7};
9.(DOS) inclusionOfPointIn2DClosedCurve(PS,CCS); DESCR={2};
10.  (DOS) inclusioOfPointIn3DClosedMesh(Point,CMS);
11.  (PLS*) triangulation2D(PS*); DESCR={2};
12.  (CMS*) triangulation3D(PS*);
13.  (CMS*) booleanOperation(CMS,CMS,DOS); DESCR={8,11};
```

Рис. 2. Листинг ЗКГ и СФ

Аффинные преобразования (1) включают: сдвиг вдоль вектора, масштабирование, поворот относительно произвольной оси. При этом применять их можно к любому базисному объекту.

В ряде алгоритмов появляется необходимость перебирать объекты для попарного пересечения за $O(n^2)$. Существует более эффективное решение (5), строящее специальную структуру OBBT (Oriented Box Bounding Tree) [1]. Такой подход существенно снижает количество операций над примитивами. Данная методика может быть задействована при поиске кривой пересечения двух поверхностей (8) для того, чтобы минимизировать количество операций пересечения треугольников (7) [2]. Стоит отметить, что метод (5) является конструктором класса пользовательских объектов OBBT.

Триангуляция (11) позволяет строить треугольники по точкам на двумерной плоскости. Триангуляция Делоне предполагает, максимизацию минимального угла сгенерированных треугольников [3, 4]. Для получения такой триангуляции могут быть использованы различные алгоритмы: итерационный, инкрементный или «разделяй и властвуй». Также существует 3*D*-аналог (12), позволяющий из набора точек в пространстве получать тетраэдры [5].

С помощью булевых операций над моделями (13) можно получить замкнутую поверхность, являющуюся результатом объединения, пересечения или дополнения двух моделей [6–8].

Примеры использования составных операций. РФС для описанных выше ЗКГ могут быть использованы для решения широкого спектра задач. Например,

лепка 3*D*-моделей может быть реализована через СФ (1) и (6). Используя булевы преобразования, можно создавать отверстия различных форм, добавлять недостающие фрагменты, делить модели на части. При необходимости можно дробить треугольники на составные части, применяя триангуляцию. Через СФ (8) можно реализовать автоматическое доведение деталей до нужной позиции в пространстве так, чтобы поверхности не пересекались или пересекались по поверхности нужной площади. Разбиение модели на тетраэдры можно использовать для решения задач динамики и прочности конструкций.

Таким образом, любой высокоуровневой задаче можно сопоставить СФ с некоторым описателем. Оформив ЗКГ таким образом, можно будет включать в описатели ЗКГ еще более высокого уровня. И библиотека может расширяться пользовательскими СФ. Размещение библиотеки подобного рода под лицензией Open Source также может значительно ускорить темпы ее развития.

В отличие от существующих решений [9–12], представленная концепция предлагает унифицированный подход для написания, отладки и оптимизации программ, реализующих ЗКГ. Более того, такой подход дает возможность пользователю быстро и самостоятельно расширять функционал библиотеки, а также редактировать уже готовые РСФ.

В дальнейшем будет выполнено *UML*-проектирование данной библиотеки, которое поможет выявить сильные и слабые стороны данного решения, а также подготовиться к этапу реализации идеи.

Литература

1. *Lin M.C., Manocha D., Gottschalk S.* OBBTree: a hierarchical structure for rapid interference detection // SIGGRAPH 96. Proc. 23rd Annual Conf. on Computer Graphics and Interactive Techniques. 1996. P. 171–180.
2. *Moller T.* A fast triangle-triangle intersection test. Stanford: Standford university press, 1997.
3. *Laszlo M.S.* Computational geometry and computer graphic in C++. Prentice Hall, 1995. 282p.
4. *Lee D.T., Schachter B.J.* Two algorithms for constructing a delaunay triangulation // International Journal of Computer and Information Sciences. 1980. Vol. 9. No. 3. P. 219–242. URL: http://link.springer.com/article/10.1007/BF00977785 DOI: 10.1007/BF00977785
5. *Delaunay M.P.* Treangulation in 3D. University of West Bohemia in Pilsen, 2002.
6. *Dutr P., Adams B.* Interactive boolean operations on surfel-bounded solids // SIGGRAPH. 2003. P. 651–656.
7. *Schifko M., Juttler B., Kornberger B.* Industrial application of exact Boolean operations for meshes // SCCG. Proc. of the 26th Spring Conf. on Computer Graphics. 2010. P.165–172. URL: http://dl.acm.org/citation.cfm?id=1925089 DOI: 10.1145/1925059.1925089
8. *Mei G., Tipper J.C.* Simple and robust Boolean operations for triangulated surfaces. Cornell University, 2013. URL: https://arxiv.org/abs/1308.4434 (дата обращения: 02.01.2017).
9. CGAL: The computational geometry algorithms library. URL: http://www.cgal.org (дата обращения: 03.12.2016)
10. VTK: The visualization toolkit. URL: http://www.vtk.org (дата обращения: 03.12.2016).
11. DGtal: Digital geometry tools and algorithms. URL: http://dgtal.org (дата обращения: 03.12.2016).
12. The VCGLibrary. URL: http://vcg.isti.cnr.it/vcglib (дата обращения: 03.12.2016).

Копьев Евгений Владимирович — студент кафедры «Теоретическая информатика и компьютерные технологии», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Ю.Т. Каганов, канд. техн. наук, доцент кафедры «Теоретическая информатика и компьютерные технологии», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

3D-SPACE OBJECT CONVERSION

E.V. Kop'ev asadiR.KEV@gmail.com

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

Computer graphics algorithms at present day allow us to solve one and the same problem in several ways. These algorithms can be ranked in order of distance from the primitives. The proposed concept will implement a modular library and efficiently conduct the work at any level of distance from the primitives. This decision will accelerate the coding, simplify debugging, will make it possible to qualitatively compare the effectiveness of the combined approaches. The study describes the 3D-model storage format and gives the definition of the class of basis objects and their designers. Moreover, we define the set of blending functions and consider some problems in computer graphics. Finally, we give examples of using this concept and propose ideas for further development

Keywords

Computer graphics, computer graphics algorithms, 3D-editors, modular algorithm library, unstructured topology, STL

References

[1] Lin M.C., Manocha D., Gottschalk S. OBBTree: a hierarchical structure for rapid interference detection. *SIGGRAPH 96. Proc. 23rd Annual Conf. on Computer Graphics and Interactive Techniques*, 1996, pp. 171–180.

[2] Moller T. A fast triangle-triangle intersection test. Stanford, Standford university press, 1997.

[3] Laszlo M.S. Computational geometry and computer graphic in C++. Prentice Hall, 1995. 282 p.

[4] Lee D.T., Schachter B.J. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*, 1980, vol. 9, no. 3, pp. 219–242. URL: http://link.springer.com/article/10.1007/BF00977785 DOI: 10.1007/BF00977785

[5] Delaunay M.P. Treangulation in 3D. University of West Bohemia in Pilsen, 2002.

[6] Dutr P., Adams B. Interactive boolean operations on surfel-bounded solids. *SIGGRAPH*, 2003, pp. 651–656.

[7] Schifko M., Juttler B., Kornberger B. Industrial application of exact Boolean operations for meshes. *SCCG. Proc. of the 26th Spring Conf. on Computer Graphics*, 2010, pp.165–172. URL: http://dl.acm.org/citation.cfm?id=1925089 DOI: 10.1145/1925059.1925089

[8] Mei G., Tipper J.C. Simple and robust Boolean operations for triangulated surfaces. Cornell University, 2013. URL: https://arxiv.org/abs/1308.4434 (accessed 02.01.2017).

[9] CGAL: The computational geometry algorithms library. URL: http://www.cgal.org (accessed 03.12.2016)

[10] VTK: The visualization toolkit. URL: http://www.vtk.org (accessed 03.12.2016).

[11] DGtal: Digital geometry tools and algorithms. URL: http://dgtal.org (accessed 03.12.2016).

[12] The VCGLibrary. URL: http://vcg.isti.cnr.it/vcglib (accessed 03.12.2016).

Kop'ev E.V. — student of Theoretical Informatics and Computer Technology Department, Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — Yu.T. Kaganov, Cand. Sc. (Eng.), Assoc. Professor of Theoretical Informatics and Computer Technology Department, Bauman Moscow State Technical University, Moscow, Russian Federation.