электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.02

Построение трехмерных моделей, посредством булевых операций

***Тарасенко С. В.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

***Тарасенко Е. В.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»

Научный руководитель: Русакова З.Н.
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
irudakov@mail.ru

Введение

Конструктивная геометрия (Constructive Solid Geometry, *CSG*) – особая техника в компьютерной графике, значительно облегчающая восприятие сложных трехмерных моделей. С ее помощью можно представить модель практически любой формы, использовав теоретико-множественных операций над более простыми объектами в трёхмерном пространстве. Эти простые объекты называются *примитивами*. Главная особенность примитивов в CSG – они должны быть *сплошными*, т.е. для любой точки пространства можно однозначно определить находится она внутри или снаружи примитива. В качестве стандартного набора обычно используют простые геометрические фигуры: шар, конус, прямоугольный параллелепипед, пирамида и цилиндр. Их комбинируют при помощи булевых операций и получат требуемую трехмерную модель.

Из двух примитивов можно получить новый объект посредством следующих *булевых операций над множествами*:

- *Объединение* (новое тело состоит из областей, лежащих внутри первого, внутри второго или внутри обоих тел).
- *Пересечение* (новое тело состоит из областей, лежащих одновременно внутри обоих тел).
- *Разность* (новое тело состоит из областей, лежащих внутри первого тела, но не лежат внутри второго).

Остальные операции можно представить в виде их комбинации.

CSG-модели облегчают восприятие комплексных объектов. Это достигается путем

представления результирующего объекта в виде дерева, листья которого — примитивы, а узлы — булевы операции (Рис. 1).

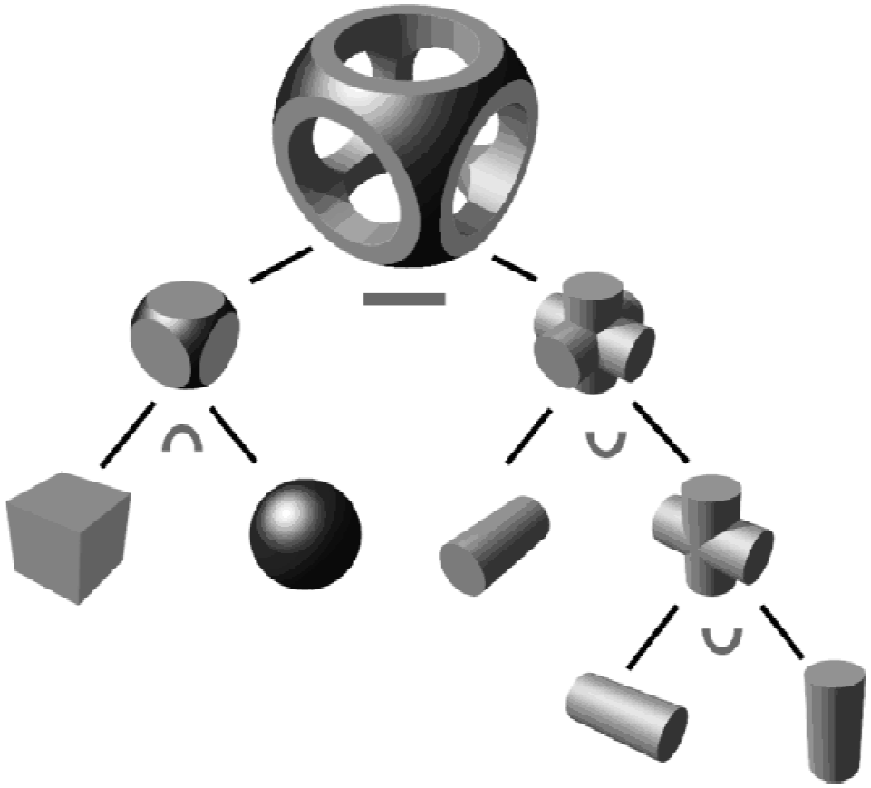

Рис. 1. CSG-дерево

Таким образом, любая CSG-модель представляет собой некую булеву функцию над примитивами.

Одной из важных задач в CSG является нормализация деревьев. Она значительно упрощает рендеринг результирующих моделей и является непосредственной частью некоторых алгоритмов построения CSG-моделей.

CSG-дерево называется *нормальным* если оно представляет собой объединение поддеревьев, в каждом из которых:

- в качестве операций используются только пересечение и разность;
- вторым операндом каждой из операций может быть только примитив.

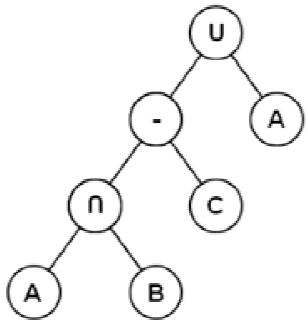

Рис. 2. Нормальная форма CSG-дерева

На Рис. 2 представлено CSG-дерево в нормальной форме.

Алгоритмы построения CSG-моделей

Алгоритмы построения CSG-модели могут работать в пространстве изображения или в пространстве объекта. К методам, работающим в пространстве изображения относится алгоритм Goldfeather, описанный в [1] и метод SCS (Sequenced Convex Subtraction) описанный в [2]. Оба этих алгоритма проводят нормализацию CSG-дерева (в случае SCS на верхних уровнях вместо операции объединения используется разность) и работают с буфером глубины (Z-буфером). Алгоритмы, работающие в пространстве объекта, рассчитывают модель до ее вывода на экран, что дает большую гибкость [3]. Дальнейшая отрисовка осуществляется любым алгоритмом удаления невидимых линий и поверхностей.

Алгоритм работы с полигональными объектами.

Этот алгоритм является ярким представителем объектного подхода в графике. Главным, и единственным, его условием, помимо условий CSG, является полигональное представление объектов. Поэтому он может работать с невыпуклыми примитивами. Алгоритм позволяет абстрагироваться от выполняемой операции, и работать непосредственно с объектами. Для каждого примитива он определяет два набора треугольников, пространственно ориентированных относительно второго примитива. Программисту достаточно выбрать по одному набору для каждого примитива, и отрисовав их, получить результат операции. В качестве полезного «побочного» эффекта данного алгоритма можно выделить нахождение линии пересечения двух примитивов, с которой можно работать в дальнейшем, либо ограничиться только ее нахождением, если того требует ситуация.

Работа алгоритма.

Алгоритм работает в 3 этапа:

1. нахождение линии пересечения двух заданных примитивов;
2. разбиение треугольников каждой поверхности вдоль линии пересечения;
3. ориентация каждого треугольника одного примитива относительно другого (внутри или снаружи);

Рассмотрим этапы этого алгоритма подробнее:

1) Для нахождения линии пересечения двух полигональных объектов, нужно найти все отрезки попарных пересечений входящих в них полигонов. Поскольку любой многоугольник можно разбить на треугольники, будем считать, что наши полигоны — треугольники. Следовательно задача сводится к нахождению отрезка пересечения двух треугольников в пространстве. Воспользуемся методом, предложенным T. Möller в [4], немного его усовершенствовав. Работая с парой треугольников из каждого примитива,

сравним их граничные объемы. Для этого достаточно сравнения всех трех измерений максимальных координат одного треугольника, с минимальными другого. Если пересечение объемов отсутствует, т.е. хотя бы в одном измерении минимальное значение координаты одного треугольника получилось больше максимального значения в другом треугольнике, то текущие треугольники точно не пересекаются. На Рис. 3. таким измерением является координата x.

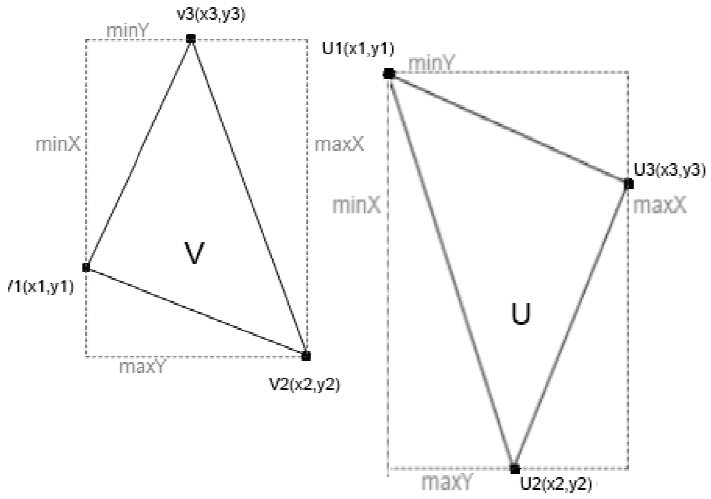

Рис. 3. Граничные «объемы» двух двухмерных треугольников

Далее проверяется положение каждого треугольника относительно плоскости другого. Для этого, сначала вычисляется уравнение плоскости проверяемого треугольника: $f(p) = (N, q_0) - (N, p)$, где N – нормаль этого треугольника, q_0 - некоторая его точка, p — точка другого треугольника. С помощью этой функции можно определить отклонение точки от заданной плоскости. Если ее значения для всех точек другого треугольника одного знака, то проверяемый треугольник лежит по одну сторону от плоскости другого треугольника. Иначе он ее пересекает.

Двумя рассмотренными простыми тестами мы отсекаем громадное число треугольников, которые не нужно обрабатывать, тем самым значительно ускоряем работу алгоритма.

По полученным значениям отклонения можно выбрать все отрезки проверяемого треугольника, которые пересекают плоскость второго, т.е. отрезки p_1p_2, для которых

$f(p_1) * f(p_2) < 0$.

Зададим уравнение линии пересечения плоскостей двух треугольников в параметрическом виде: $L = O + tD$, где O — одна из точек на линии, а $D = N_1 \times N_2$– вектор пересечения плоскостей. Искомый отрезок будет иметь координаты вершин $A = O + t_2 D$и

$B = O + t_1 D$(Рис. 4).

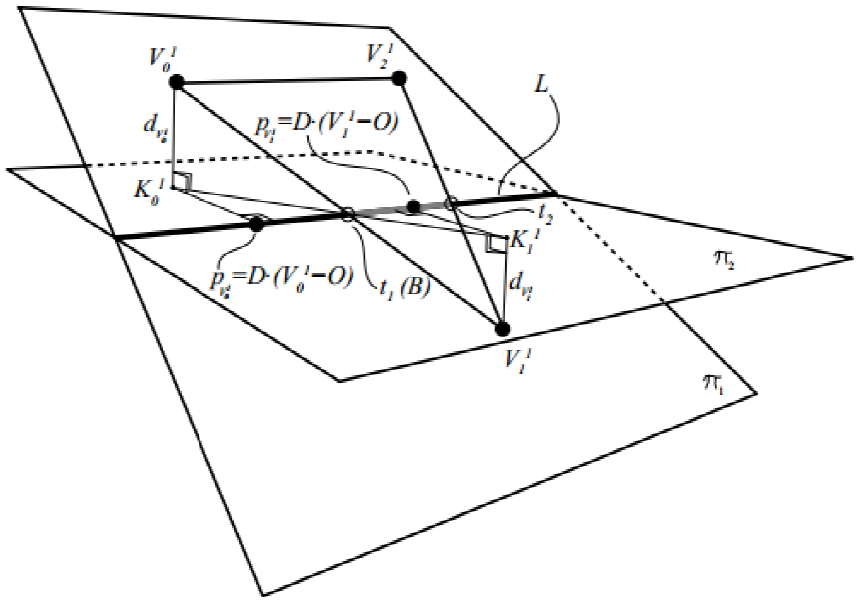

Рис. 4. Поиск пересечения треугольников

Найдем t_1. Для этого необходимо найти параметры $p_{v_1^0}$и $p_{v_1^1}$, соответствующие проекциям точек V_0 и V_1 на прямую L по формуле$p_{v_1^i} = D \cdot (V_1^i - O)$, для $i = 1, 2$.

Рассмотрим подобные треугольники $\Delta V_0 K_0 B$ и $\Delta V_1 B K_1$. Из подобия имеем соотношение$\frac{d_{v_0^1}}{d_{v_1^1}} = \frac{K_0 B}{K_1 B}$, где $d_{v_1^i}$— расстояние от точки V_i до ее проекции на плоскость. Из подобия $\Delta P_0 K_0 B$ и $\Delta P_1 K_1 B$ имеем соотношение$\frac{(p_{v_0^1} - t_1)}{(t_1 - p_{v_1^1})} = \frac{K_0 B}{K_1 B}$.

Выражаем t_1 :

$$t_1 = \frac{d_{v_1^1} \cdot p_{v_0^1} + d_{v_0^1} \cdot p_{v_1^1}}{d_{v_0^1} + d_{v_1^1}}$$

Аналогично находим t_2.

После нахождения t_1 и t_2, нужно задать явное соответствие между отрезком и треугольниками, которым он принадлежит.

2) Теперь необходимо разбить каждый треугольник, с которым соотнесен хотя бы один отрезок пересечения, на группы треугольников таким образом, чтобы каждый из отрезков являлся стороной двух смежных треугольников. Пример разбиения представлен на Рис. 5.

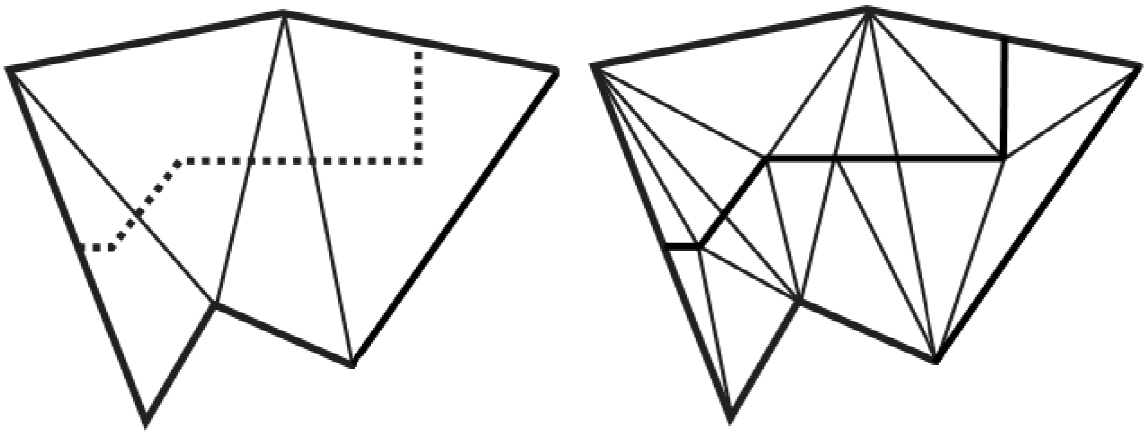

Рис. 5. Разбиение треугольников вдоль линии пересечения.

Соберем все смежные отрезки, лежащие на одной прямой в пределах одного треугольника, в один отрезок. Достаточно проверить коллинеарность их векторов, и если они коллинеарны, сравнить концы отрезков. Параллельные отрезки, у которых совпадает один из концов, можно объединить в один, исключив общую точку. Проделав это для всех пересекаемых треугольников, получится «линия» пересечения двух примитивов.

Перейдем непосредственно к разбиению. Разбиение замкнутой области на группу треугольников называется *триангуляцией*. Воспользуемся идеями итерационного метода триангуляции Б.Н. Делоне (описание триангуляции и различные методы ее построения описано [5]).

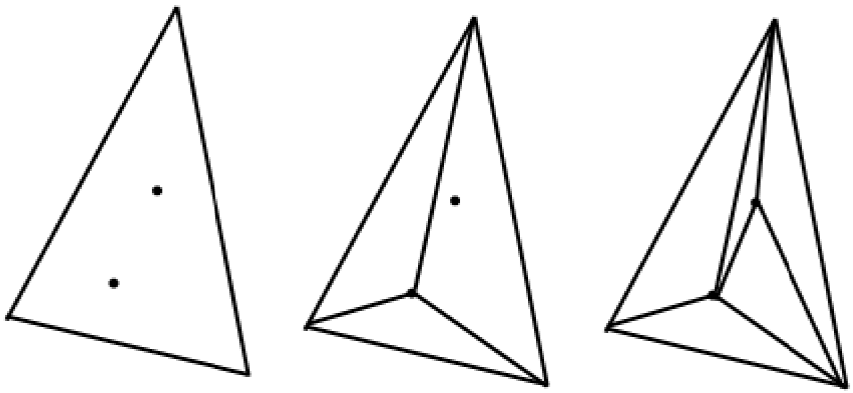

Рис. 6. Итеративная триангуляция

Пусть есть множество треугольников, в начальный момент времени содержащее только исходный треугольник, и множество точек, по которым нужно этот треугольник разбить.

Алгоритм:

1. Выбрать первую точку.
2. Проверить ее принадлежность каждому треугольнику из множества

треугольников.

3. Если точка не принадлежит ни одному треугольнику, перейти на шаг 7.

4. Если найден треугольник, которому принадлежит точка, то разбить его по этой точке на три треугольника и удалить из множества проверяемых.

5. Проверить площадь каждого из трех полученных треугольников.

6. Каждый треугольник с ненулевой площадью добавить к множеству проверяемых.

7. Если есть еще точки, выбрать следующую и перейти на шаг 2.

8. Результат — множество проверяемых треугольников.

Алгоритм достаточно прост для реализации и выполняет необходимые для триангуляции требования: отрезки, по которым пересекаются исходные треугольники в примитивах, должны быть сторонами новых треугольников в полученном наборе. Пример триангуляции приведен на Рис. 6.

Разбив все треугольники, получаем две группы треугольников для каждого примитива. Остается определить их ориентацию.

3) В данной части алгоритма решается задача пространственной локализации каждого треугольника одного примитива относительно другого. Поскольку после предыдущего шага нет ни одного треугольника, пересекаемого линией пересечения, для определения положения треугольника достаточно локализовать лишь одну точку — его барицентр. Треугольники требуется ориентировать по свойству внутри или снаружи. Используем алгоритм испускания луча (ray-casting), применяемый для решения подобных задач в компьютерной графике.

Этот алгоритм решает задачу локализации посредством испускания произвольного луча и подсчета количества пересечений с граничной областью. Если число пересечений четно, то проверяемая точка лежит вне области. На Рис. 7. приведена иллюстрация двумерного варианта ray-casting.

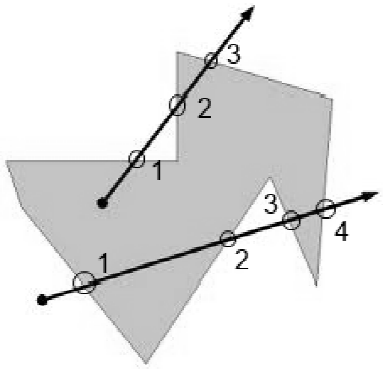

Рис. 7. Локализация точки испусканием луча

За точку испускания луча принимается центр масс треугольника, а за направление — нормаль к его плоскости.

Для подсчета количества пересечений необходимо проверить пересечение луча с каждым треугольником другого примитива и подсчитать количество пересечений с внутренними частями треугольников (если учитывать пересечения со сторонами, может возникнуть ложная ситуация, когда один луч дважды пересекает одно и тоже ребро в разных треугольниках).

Для локализации пересечений воспользуемся алгоритмом, предложенным T. Möller и B. Trumbore в [6]. Он работает с *барицентрическими координатами* — отношением площадей треугольников, на которые разбивается исходный, к его площади. Рассмотрим Рис. 8. По свойствам барицентрических координат $z(u,v) = u * V2 + v * V1 + (1 - u - v) * V0$, где *z(u,v)* — точка пересечения луча с треугольником. С другой стороны $z(t) = P + t * D$- параметрическое уравнение прямой, проходящей через *z* и *p*.

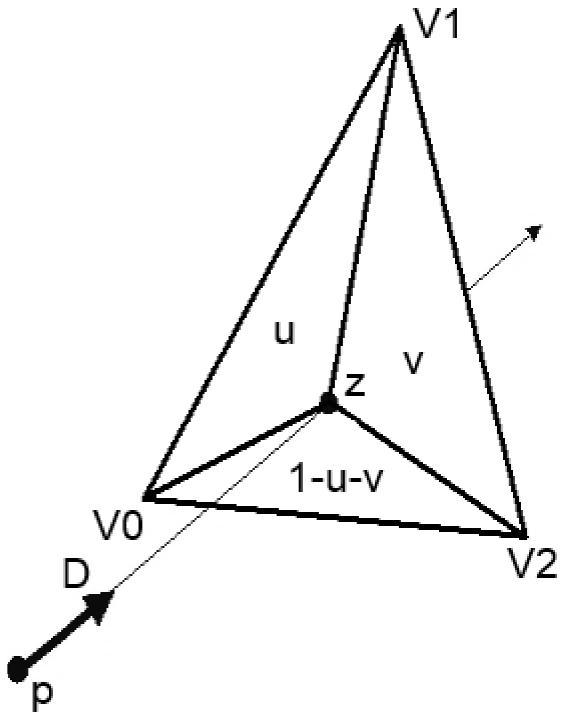

Рис. 8. Барицентрические координаты

Приравняв левые части уравнений и проведя алгебраически преобразования, получим:

$$u = \frac{[D \times (V2 - V0)] \cdot (P - V0)}{[D \times (V2 - V0)] \cdot (V1 - V0)}$$

$$v = \frac{[(P - V0) \times (V1 - V0)] \cdot D}{[D \times (V2 - V0)] \cdot (V1 - V0)}$$

$$t = \frac{[(P - V0) \times (V1 - V0)] \cdot (V2 - V0)}{[D \times (V2 - V0)] \cdot (V1 - V0)}$$

При выполнении условий $0 < u < 1$, $0 < v < 1$и $0 \leqslant u \leqslant 1$треугольник и луч пересекаются.

Примечание. Алгоритм испускания луча может быть ускорен при помощи BVH деревьев. Тогда на пересечение будет проверяться не все треугольники, а только попадавшие в определенный граничный объем.

Таблица 1

Связь ориентации внутри/снаружи с булевыми операциями.

	Примитив A	Примитив B
$A \cup B$	снаружи	снаружи
$A \cap B$	внутри	внутри
$A \setminus B$	снаружи	внутри
$B \setminus A$	внутри	снаружи

В *Таблице 1* показана зависимость булевых операций от ориентации групп треугольников в примитивах.

Сравнение алгоритмов построения CSG-модели

Преимущества и недостатки вышеописанных алгоритмов приведены в *Таблице 2*.

Выбор алгоритма и зависит от поставленной задачи.

Таблица 2

Сравнение алгоритмов

	Алгоритмы, основанные на Z-буфере.	Алгоритм, основанный на трассировке лучей.	Алгоритм, работающий с полигональными объектами.
преимущества	- первые, из известных алгоритмов;	- наиболее реалистичное изображение за счет трассировки;	- разовое вычисление всех операций; - возможность работы с линией пересечения; - возможность выбора алгоритма визуализации;
недостатки	- Z-коллизия — неверное отображения близких друг к другу плоскостей; - пересчет операции на каждом кадре;; - сложность работы с невыпуклыми объектами; - требуют нормализации;	- пересчет операции на каждом кадре;	- при большом количестве треугольников время вычесчения может существенно возрасти; - затраты памяти на хранение четырех групп треугольников;

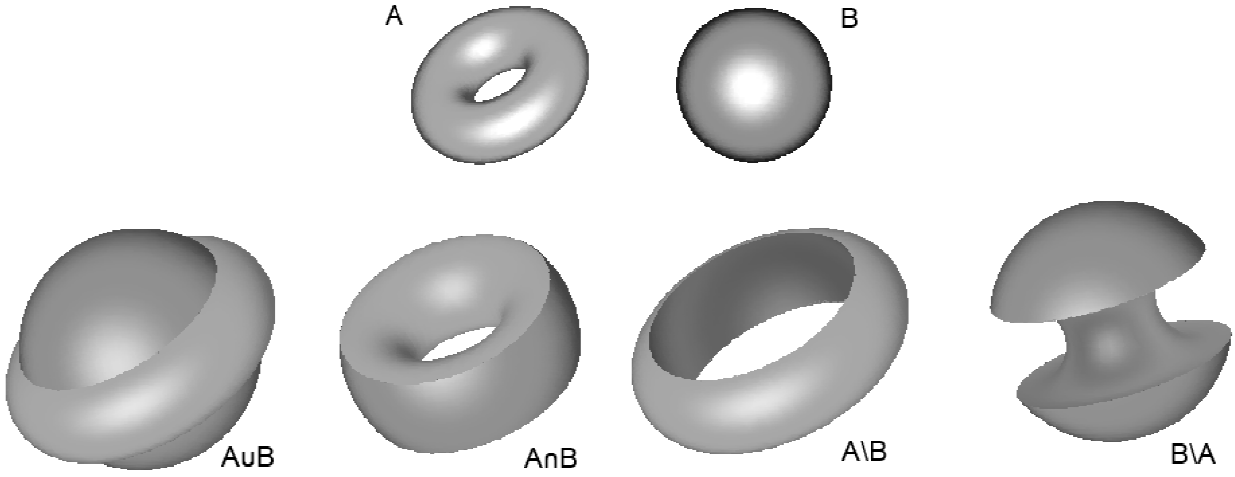

Рис. 9. Пример работы алгоритма.

Вывод

В данной статье были рассмотрены основы конструктивной твердотельной геометрий и был предложен алгоритм, позволяющий выполнять булевы операции над сплошными телами, что является необходимым при построении CSG-модели. Были подробно рассмотрены следующие этапы алгоритма: поиск линии пересечения двух треугольников, триангуляция, поиск пересечения луча с треугольником. В результате проведенного сравнения данного алгоритма с другими известными методами была выявлена его конкурентоспособность при решении определенного класса задач, в частности тех, в которых

присутствует необходимость интерактивного взаимодействия с получаемой моделью.

Список литературы

1. Goldfeather J., Hultquist J.P., Fuchs H. Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System. // Computer Graphics, 1986. V. 20, I. 4, pp. 107-116. DOI: 10.1145/15886.15898.
2. Stewart N., Leach G., John S. An Improved Z-Buffer CSG Rendering Algorithm. // HWWS '98 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS, 1998. pp. 25-30. DOI: 10.1145/285305.285308.
3. Laidlaw D.H., Trumbore W.B., Hughes J.F. Constructive solid geometry for polyhedral objects // Computer Graphics, 1986. V. 20.I. 4. P. 161-170. DOI: 10.1145/15922.15904.
4. Möller T. A fast triangle-triangle intersection test // Journal of Graphics Tools. 1997. V. 2. I. 2. P. 25-30. DOI: 10.1080/10867651.1997.10487472.
5. Скворцов А.В. Триангуляция Делоне и ее применение. Томск:ТГУ, 2002. С. 128.
6. Möller T., Trumbore B. Fast, minimum storage ray-triangle intersection // Journal of Graphics Tools, 1997. V. 2. I. 1. P. 21-28. DOI: 10.1080/10867651.1997.10487468.