



Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Курсовая работа

**Декомпозиция невыпуклых объектов
для решения задач обнаружения пересечений**

Гончаренко Дмитрий Александрович
320 группа

Научный руководитель: с.н.с. Игнатенко А.В.

Москва, 2018

Аннотация

В данной работе поднимается актуальная проблема в компьютерной графике – *выпуклое разбиение фигур*¹, использующееся для дальнейшего поиска пересечений объекта в сцене. Рассматриваются алгоритмы декомпозиции и методы решения поставленной задачи, в том числе иерархия ограничивающих объемов. Реализована декомпозиция сложнопolygonальных объектов на основе библиотеки HASC на языке C++. Проведена сравнительная характеристика точного и приближительного разбиений на примере 14 невыпуклых объектов.

¹здесь и далее **выпуклое разбиение** - разбиение произвольной фигуры на только выпуклые части

Содержание

1	Введение	3
2	Предметная область	3
2.1	Понятие выпуклости	3
2.2	Проверка на выпуклость	4
3	Задача	5
3.1	Цели работы	5
3.2	Постановка задачи	5
4	Алгоритмы разбиения	5
4.1	Триангуляция на плоскости	5
4.1.1	Триангуляция фигур с отверстиями	6
4.1.2	Триангуляция Делоне	7
4.2	Трехмерная декомпозиция	8
4.2.1	NP-полная задача разбиения невыпуклых многогранников	8
4.3	Точное и приближительное разбиения	9
5	Иерархия ограничивающих объемов	10
5.1	Ограничивающие объемы	10
5.2	Стратегии построения дерева иерархии	12
5.3	Стратегии обхода дерева иерархии	12
5.4	Использование BVH-дерева в задачах обнаружения пересечений	13
6	Реализация алгоритма разбиения	14
6.1	Инструменты	14
6.2	Алгоритм	14
6.3	Результаты	14
7	Заключение	15
8	Список литературы	16

1 Введение

Компьютерная графика все сильнее проникает в науку, медицину, сферу развлечений. Поиск столкновений, вычисление расстояний и глубины проникновения объектов - задачи, остающиеся актуальными уже не первый десяток лет. Появление GPU, многопоточной структуры для массивных параллельных вычислений, стало, несомненно, прорывом для трехмерной графики в целом. Сейчас повсеместно используются графические процессоры для ускорения вычислений объемных объектов. CPU и комбинированные решения GPU и CPU используются для достижения наилучшей точности в обработке графической информации.

В своей работе я рассматриваю задачи, возникающие при попытке обнаружения точного пересечения объектов. Одной из таких задач является проблема поиска столкновений нетривиальных фигур. Сложность вычисления в случае невыпуклых объектов резко возрастает. Популярным и наиболее действенным решением на текущий момент считается разбиение (декомпозиция) фигуры на выпуклые части, что значительно упрощает процесс обнаружения пересечений.

Ниже описаны основные задачи, с которыми можно столкнуться при изучении поставленного вопроса. Такие как поиск оптимальных алгоритмов, NP-сложность разбиения трехмерных объектов, точная декомпозиция многогранников (ECD) и выбор стратегий иерархии ограничивающих объемов.

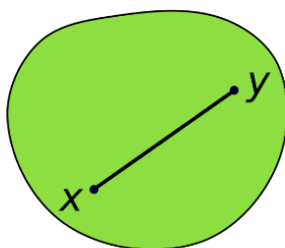
2 Предметная область

2.1 Понятие выпуклости

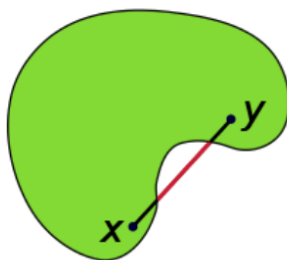
Для начала, введем общее понятие выпуклости.

Выпуклым называется множество, в котором все точки отрезка, образуемого любыми двумя точками данного множества, также принадлежат данному множеству.

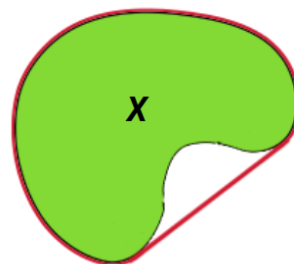
Выпуклой оболочкой (convex hull) множества X называется наименьшее выпуклое множество, содержащее X .



Convex Set



Non-Convex Set



Convex Hull

2.2 Проверка на выпуклость

Для того чтобы определить, является ли выбранный многоугольник или многогранник выпуклыми, используют различные алгоритмы. Рассмотрим некоторые из них.

Многоугольник. На плоскости используют следующие методы определения выпуклости:

1. Для каждого ребра вычислить, лежат ли все несмежные вершины строго по одну сторону от выбранного ребра. Сложность $O(n^2)$. (см. рис 1)
2. Для каждого внутреннего угла проверить, меньше ли он 180° . Сложность $O(n)$. Работает только для фигур без самопересечений (пр. "звезда" - невыпуклая).
3. Прodelать циклический путь по всем смежным вершинам. Должно быть только две смены направления.

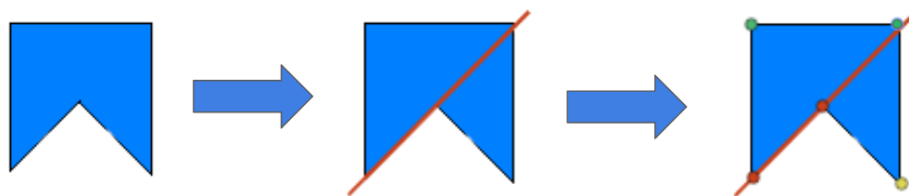


Рис. 1: Невыпуклый многоугольник с опорной прямой (красная линия)

Многогранник. В трехмерном случае выделяются следующие алгоритмы:

1. Для каждой грани вычислить, лежат ли все несмежные вершины строго по одну сторону от этой грани. Сложность $O(n^2)$. (см. рис. 2)
2. Для каждой грани F многогранника P вычислить центроид² C . Для каждой соседней грани G проверить, лежит ли их C за опорной плоскостью G . Если для всех граней это выполняется, то фигура предполагается выпуклой, иначе она является невыпуклой. Сложность $O(n)$.

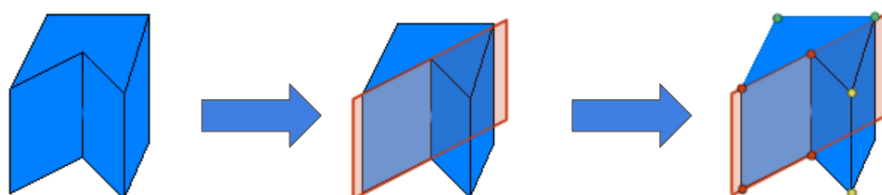


Рис. 2: Невыпуклый многогранник с опорной плоскостью (красный прямоугольник)

²центроид (или барицентр) - центр масс

3 Задача

3.1 Цели работы

В точном обнаружении пересечений невыпуклых фигур возникают проблемы падения скорости и возрастания сложности вычислений. С этой проблемой справляются разбиение фигуры на упрощенные выпуклые части и иерархия ограничивающих объемов. В свою очередь задачи оптимального выпуклого разбиения фигур в трехмерном пространстве являются нетривиальными. Целью работы является исследование алгоритмов выпуклого разбиения и реализация выбранного метода на C++.

3.2 Постановка задачи

1. Рассмотреть основные методы декомпозиции, такие как триангуляция Делоне, тетрадаризация, методы точного и приближительного разбиения.
2. Исследовать NP-полную задачу о невозможности разбиения невыпуклых многогранников.
3. Привести возможные алгоритмы ее решения.
4. Сравнить стратегии построения иерархий ограничивающих объемов.
5. Реализовать выбранный алгоритм разбиения.

4 Алгоритмы разбиения

4.1 Триангуляция на плоскости

Триангуляция – процесс/результат разбиения геометрической фигуры на симплексы³. Одним из алгоритмов триангуляции многоугольника является *алгоритм обрезки ушей*⁴ (рис. 3). Сформулируем соответствующую теорему.

Теорема Мейстерса (о двух ушах)⁴

Любой простой n-угольный полигон с $n > 3$ имеет как минимум два уха.

Суть алгоритма Находится очередное ухо и отрезается от многоугольника. После этого операция повторно применяется к оставшемуся многоугольнику до тех пор, пока не останется один треугольник. рис. 3

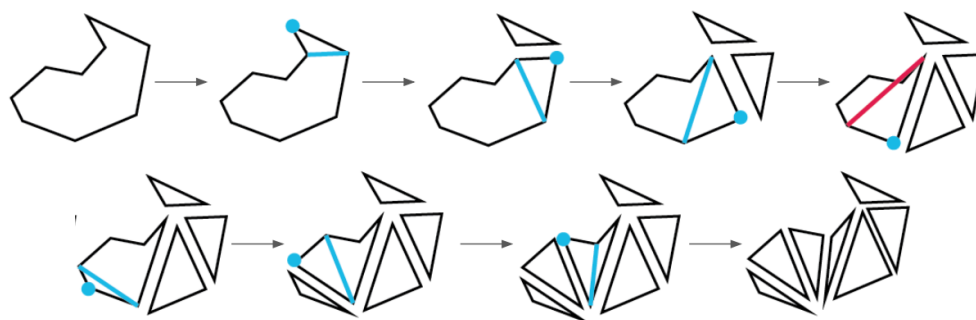


Рис. 3: Алгоритм отрезания ушей (Ear Cutting Algorithm)

³Симплекс (n-мерный тетраэдр) – фигура, являющаяся n-мерным обобщением треугольника.

⁴Ушами многоугольника называют треугольники, две стороны каждого из которых являются сторонами многоугольника, а третья лежит полностью внутри него.

Преимущества и недостатки Этот способ работает только для многоугольников без дырок. Он прост в реализации, но медленнее, чем некоторые другие алгоритмы. Алгоритм работает за время $O(n^2)$. Результирующая декомпозиция не является оптимальной.

Оптимизация Не вызывает сомнения, что разбиение полученное методом отрезания ушей может быть выполнено не единственным способом. Еще в прошлом веке Хертелем и Мелхорном был придуман оптимизирующий *алгоритм удаления возможных диагоналей*. Его суть заключается в следующем: в процессе триангуляции удаляются все возможные диагонали, убрав которые не появляются невыпуклые вершины. Другими словами, полученная декомпозиция остается выпуклой. рис. 4

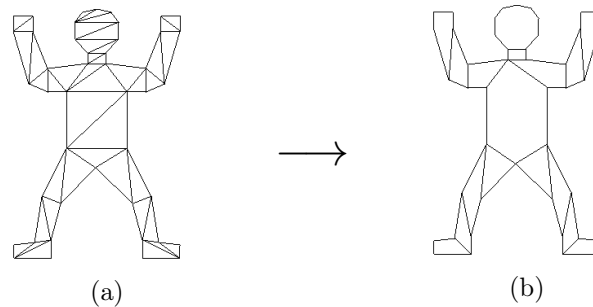


Рис. 4: Алгоритм Hertel-Mehlhorn. (a) - триангуляция, (b) - после работы алгоритма

Предложенный алгоритм можно встроить непосредственно в процесс триангуляции. Для этого достаточно на каждом шаге проверять, возможно ли объединить только что отрезанное ухо с предыдущим многоугольником. Если да - объединять с предыдущим, иначе - предыдущим многоугольником становится отрезанный треугольник [1]. Как показывает эвристика, полученная триангуляция методом удаления лишних диагоналей близка к оптимальной.

4.1.1 Триангуляция фигур с отверстиями

Из предыдущего параграфа стало ясно, что алгоритм отрезания ушей перестает функционировать в случае присутствия в геометрии . Для разрешения этой ситуации потребуются или дополнительное разбиение фигуры на несколько монотонных⁵ частей, или вырезание из многоугольника канала нулевой ширины с перенумерацией вершин (рис. 5).

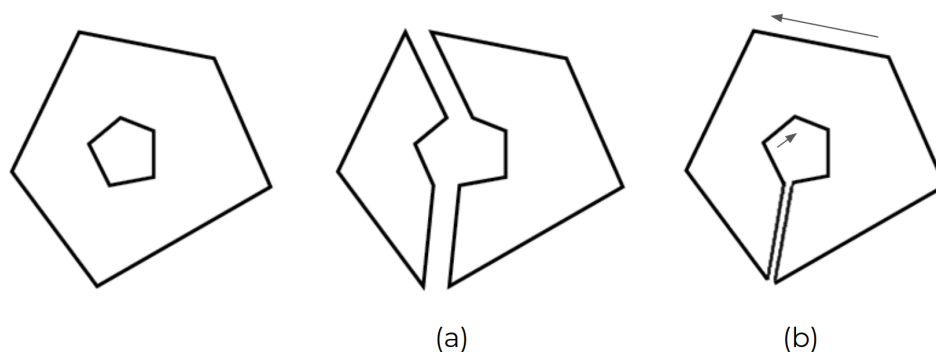


Рис. 5: (a) - декомпозиция полигона на монотонные части, (b) - “канал” нулевой ширины

⁵Многоугольник называется **монотонным**, если его граничная ломаная имеет не более двух точек пересечения с прямой, перпендикулярной данной.

4.1.2 Триангуляция Делоне

Триангуляция Делоне – триангуляция для заданного множества точек S на плоскости, при которой для любого треугольника все точки из S за исключением точек, являющихся его вершинами, лежат вне окружности, описанной вокруг треугольника. Из каждой точки порождается окружность, проходящая через две ближайшие точки (рис. 6а).

Интересная реализация метода на изображениях представлена на рис. 6б. [10]

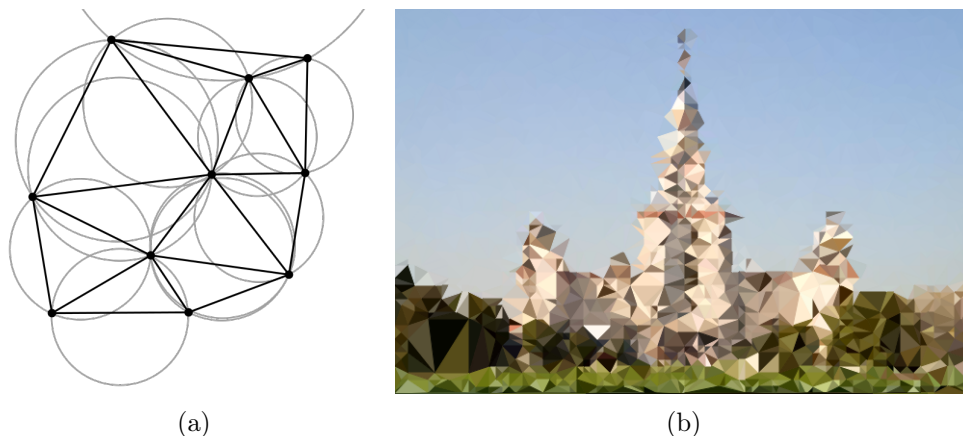


Рис. 6: (а) - пример триангуляции Делоне, (b) - триангуляция картинки Делоне

Свойства Триангуляция Делоне –

- взаимно однозначно соответствует диаграмме Вороного для того же множества точек.
- единственна, если никакие четыре точки не лежат на одной окружности.
- максимизирует минимальный угол среди всех углов всех построенных треугольников, тем самым избегаются «тонкие» треугольники.
- максимизирует сумму радиусов вписанных шаров.
- минимизирует дискретный функционал Дирихле.
- минимизирует максимальный радиус минимального объемлющего шара.
- на плоскости обладает минимальной суммой радиусов окружностей, описанных около треугольников, среди всех возможных триангуляций.

Алгоритм Задача о нахождении триангуляции Делоне множества точек в d -мерном Евклидовом пространстве может быть сведена к задаче поиска выпуклой оболочки этого множества точек в $(d+1)$ -мерном пространстве. Для этого необходимо каждой точке p ввести дополнительную координату $|p|^2$. Возвращаясь к пространству предыдущей размерности нужно удалить последнюю координату. В связи с уникальностью выпуклой оболочки, триангуляция тоже будет уникальной (рис. 7). Реализация на github [9].

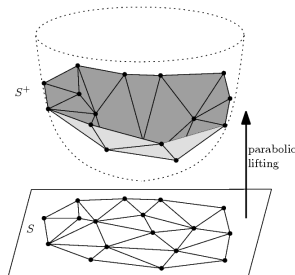


Рис. 7: Триангуляция Деланау через выпуклую оболочку

4.2 Трехмерная декомпозиция

В отличие от разбиения многоугольников, рассмотренного выше, не каждую фигуру в трехмерном пространстве можно триангулировать тетраэдрами без введения вспомогательных условий. Примером может послужить *скрученная призма* – невыпуклый призматический многогранник, полученный из однородной q -угольной путём деления боковых граней диагональю и вращения верхнего основания, обычно на угол π/q радиан, в направлении, при котором стороны становятся вогнутыми (рис. 8). Скрученная призма не может быть разбита на тетраэдры без введения новых вершин.

4.2.1 NP-полная задача разбиения невыпуклых многогранников

Скрученная призма с параметром $q = 3$ называется *многогранником Шёнхардта* (рис. 8). На его основе было доказано, что триангуляция невыпуклого многогранника относится к классу NP-полных задач. В свою очередь задача о минимальном разбиении выпуклого многогранника относится к классу NP-сложных задач [3].

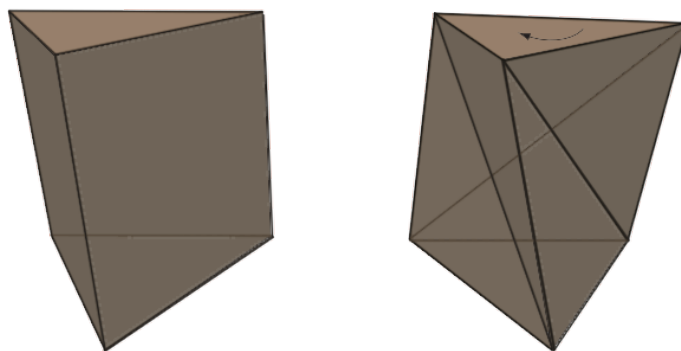


Рис. 8: Многогранник Шёнхардта
(невозможно триангулировать)

Проблема		
Невозможность триангуляции невыпуклого многогранника		
Решение	Описание	Недостатки
Точки Штейнера	Ввести дополнительные точки разбиения вне объекта	Поиск расположения
Полигональный суп	Представить объект, в виде набора полигонов	Потеря информации
CSG объекты	Использовать технологию Constructive Solid Geometry (см. рис. 9)	Численная неточность

Конструктивная блочная геометрия (CSG) является одним из возможных способом моделирования в трёхмерной графике. Она позволяет создать сложную сцену или объект с помощью битовых операций, комбинируя несколько иных упрощенных объектов (см. рис. 9).

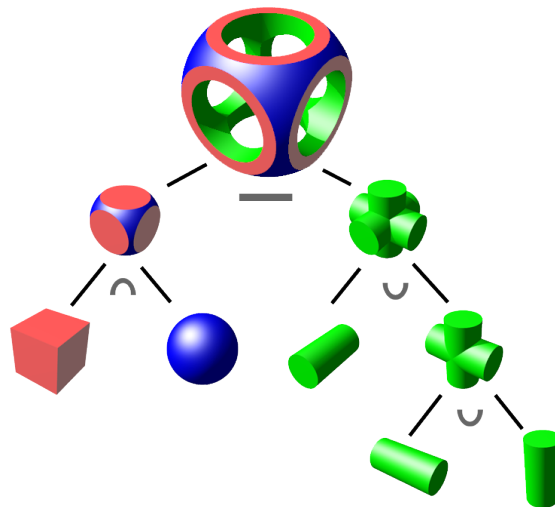


Рис. 9: Пример КБГ. Двоичное дерево, где “листья” – это объекты, а “узлы” – операции

4.3 Точное и приближенное разбиения

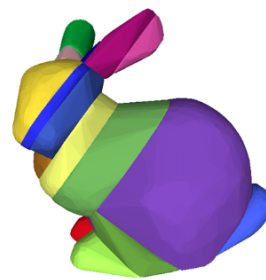
В некоторых ситуациях при декомпозиции сложных моделей, можно сильно сэкономить на вычислительных ресурсах, если вместо *точного разбиения* считать *приближенное*. Во многих задачах такой подход, помимо увеличения эффективности и удобного иерархического представления, даст результат, не превышающий пределы допустимой точности [5].



исходная модель



7611 частей



20 частей

Рис. 10: Точное и приближенное выпуклое разбиение

Введем понятие *меры вогнутости*.

Декомпозиция многогранника:

$$D(P) = \{P_i \mid \cup_i P_i = P \text{ and } \forall_{i \neq j} P_i \cap P_j = \emptyset\}$$

Мера вогнутости многогранника P:

$$\text{concave}(P) = \max_{x \in \partial P} \{\text{concave}(x)\}$$

Приближенная декомпозиция P:

$$ACD_\tau(P) = \{P_i \mid P_i \in D(P) \text{ and } \text{concave}(P_i) \leq \tau\}$$

Точная декомпозиция P:

$$ECD(P) = ACD_0(P)$$

Обозначим через H_p выпуклую оболочку фигуры P .

Вогнутость (concavity) P можно измерить как расстояние от точки x многогранника P до H_p . Определим следующие вспомогательные понятия:

Мост (bridge) – выпуклая оболочка граней, соединяющих несмежные вершины границы многогранника ∂P .

Карман (pocket) – часть границы ∂P которая не находится на выпуклой оболочке границы ∂H_p .

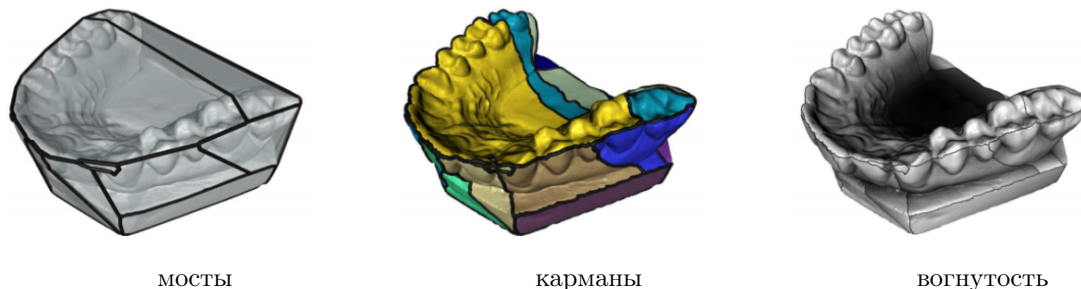


Рис. 11: Алгоритм вычисления вогнутости фигуры

Алгоритм Алгоритм вычисления вогнутости фигуры (рис. 11) заключается в следующем. Для объекта строятся мосты и карманы таким образом, чтобы между каждым карманом и мостом можно было установить взаимнооднозначную уникальную связь. Вогнутость $concave(x)$ подсчитывается как расстояние от точки x кармана до соответствующего моста β_x .

$$concave(x) = distance(x, H_p) = distance(x, \beta_x)$$

На правом изображении рисунка 11 темный цвет означает более высокое значение меры вогнутости, светлый - низкое.

5 Иерархия ограничивающих объемов

В задачах поиска пересечений с большим числом одновременно тестируемых объектов существует смысл определять, а какие объекты в данный момент времени в принципе могут пересечься и соответственно нужно тестировать. Для этого задача обнаружения коллизий разбивается на две фазы: *широкая фаза (broad phase)* и *узкая фаза (narrow phase)*. Широкую фазу отвели на проверку возможности пересечения между телами, для этого достаточно вписать объект в некоторую простую фигуру, *ограничивающий объем*, проверка на пересечение с которым будет занимать существенно меньше времени. В узкую фазу тестируется уже непосредственное пересечение пары объектов, в случае успеха широкой фазы для этих фигур.

5.1 Ограничивающие объемы

Теперь строго определим ограничивающий объем.

Ограничивающий объем множества точек – замкнутая фигура содержащая все точки этого множества. Существует много различных типов ограничивающих объемов с разной степенью ограничения и скорости тестирования, ниже на рис. 12 представлены некоторые из них.



Рис. 12: Примеры типов ограничивающих объемов

Чем сложнее объем - тем он лучше ограничивает тестируемое тело, тем медленнее происходит проверка.

Ограничивающие объемы могут стать полезны не только в широкой фазе, но и в узкой. Для этого вводится древовидная структура – *иерархия ограничивающих объемов (bvh)*. Некоторый объем ограничивающий объект или его часть называется *узлом* иерархии. Сложный объект (root) делится на простые дочерние, причем на каждом уровне дети покрывают все вершины своего родителя (рис. 13).

Функция стоимости Для вычисления сложности построенной иерархии ограничивающих объемов используется функция стоимости:

$$T = N_v C_v + N_p C_p + N_u C_u, \text{ где}$$

- N_v — количество bv-пар, проверяемых на пересечение
- C_v — стоимость проверки bv-пары на пересечение
- N_p — количество тестируемых примитивов
- C_p — стоимость теста примитивов
- N_u — количество узлов для обновления
- C_u — стоимость обновления узла

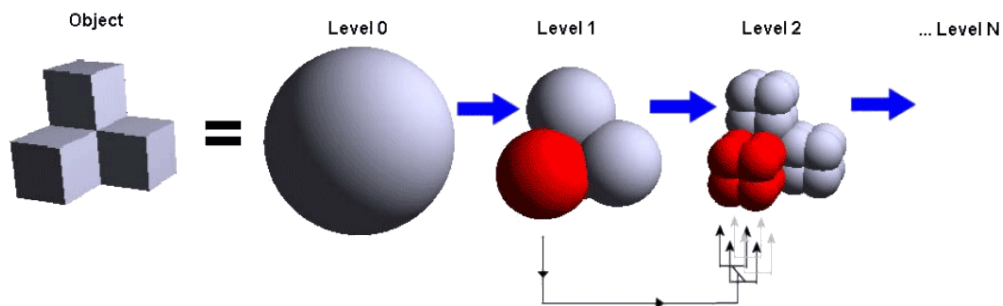


Рис. 13: Модель иерархии сверху-вниз (top-down)

5.2 Стратегии построения дерева иерархии

Как уже было сказано в предыдущем параграфе BVH можно представить в виде дерева. Эвристически показано что наиболее эффективное bvh-дерево для обнаружения пересечений является бинарным. При увеличении n-арности будет уменьшаться высота и время обхода, но возрастать время на проверку узла. Существует несколько разных алгоритмов построения иерархии (рис. 14).

Стратегия сверху-вниз (Top-down) Исходное множество ограничивается неким объемом (корень). До тех пор пока не выполнится условие, например не закончатся примитивы для разбиения, происходит рекурсивное деление множества на два. Для эффективного выбора места разбиения можно использовать центр масс (рис. 14a).

Достоинства. Простота реализации, высокая скорость построения.

Недостатки. В результате получается не всегда оптимальное решение.

Стратегия снизу-вверх (Bottom-up) Каждый примитив в множестве ограничивается неким объемом (листья). Узлы объединяются под общими ограничивающими объемами, до тех пор пока не будет достигнут корень (рис. 14b).

Достоинства. Зачастую результат построения превосходит метод top-down.

Недостатки. Скорость построения уступает предыдущей стратегии. Сложнее реализация.

Стратегия вставками (Insertion) В некой случайной последовательности для каждого объекта происходит поиск его места и вставка в изначально пустое дерево. Чем больше и дальше объект, тем он окажется выше (рис. 14c).

Достоинства. Удобен для динамических систем, высокая скорость обновления дерева.

Недостатки. Дерево растет максимально медленно. Сложная реализация.

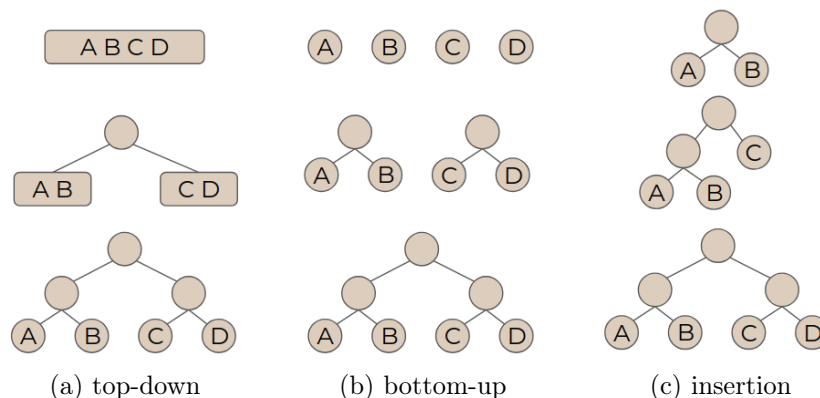


Рис. 14: Стратегии построения bvh-дерева

5.3 Стратегии обхода дерева иерархии

Существует два основных подхода в обходе bvh-деревьев. *Depth-first search* (поиск в глубину) – самый широко используемый алгоритм в задачах обнаружения пересечений (рис. 15a). *Breadth-first search* (поиск в ширину) – требует больше памяти, удобен для прерываемых систем поиска столкновений (рис. 15b).

Подходы приведенные выше не зависят от данных в структуре, а зависят только от самой структуры. Из-за чего эти методы также называют “слепыми”. С другой стороны есть

Best-first search (поиск по наилучшему совпадению) (рис. 15с). Жадный алгоритм, который всегда перемещается к узлу наилучшему по какому-то заданному критерию, например по расстоянию до цели.

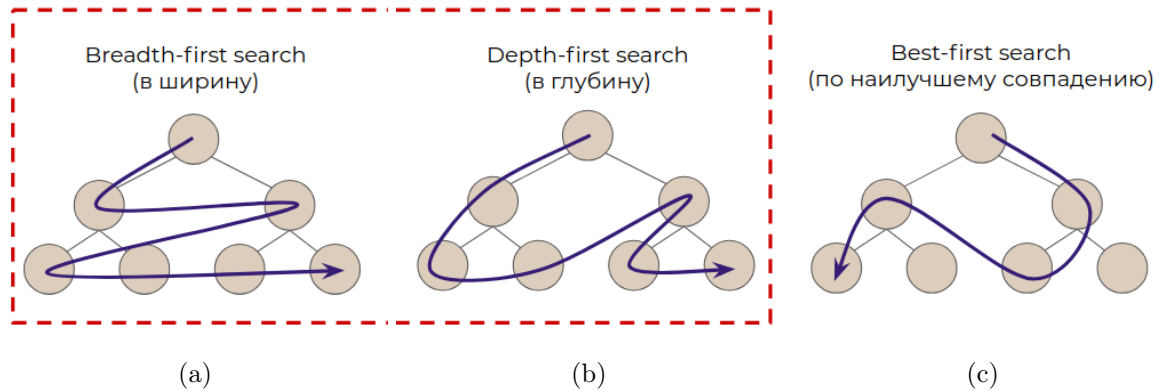


Рис. 15: Стратегии обхода bvh-дерева. Выделенные методы - “слепые”

5.4 Использование BVH-дерева в задачах обнаружения пересечений

Суть алгоритма В начале на пересечение тестируются корни двух объектов. Если корни пересеклись происходит рекурсивная проверка левого и правого дочернего поддерева на пересечение. В зависимости от выбранного метода обхода будет получаться разная производительность. Наиболее часто для задач обнаружения столкновений используется алгоритм поиска в глубину, как наиболее универсальный.

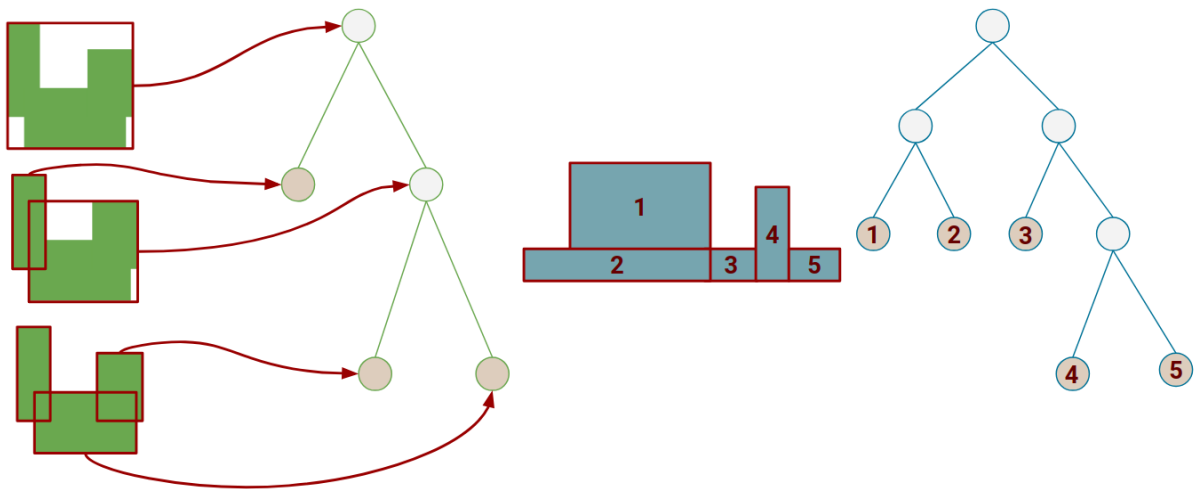


Рис. 16: Алгоритм построения bvh-дерева по объекту

6 Реализация алгоритма разбиения

Для реализации алгоритма приближенного разбиения была выбрана библиотека с открытым исходным кодом HACD, как наиболее близкая к статье Jyh-Ming Lien[5] о выпуклой декомпозиции.

6.1 Инструменты

Я использовал:

- ОС: Ubuntu 16.04
- Intel Core i5-6200U 2.3 MHz
- Графический редактор Blender 2.79b - для визуализации результата
- HACD - библиотека приближенной декомпозиции
- Библиотеки OBJLoader и OFFLoader для загрузки объектных файлов в программу
- Язык C++, std11

6.2 Алгоритм

На вход программа получает объектный файл в формате .off или .obj.

Вызывается функция Scene::parseOFF() или Scene::parseOBJ(), которая создает Object obj и распаршивает объектный файл. После предварительной конфигурации вызывается функция obj->compute() и начинается процесс декомпозиции. В результате на выходе получается .off файл, который можно открыть в графическом редакторе Blender.

6.3 Результаты

Сравнение точной и приближенной декомпозиции:


Models															
	Full model			Solid						Surface					
				ACD _{0.2}		ACD _{0.02}		ECD		ACD _{0.2}		ACD _{0.02}		ECD	
<i>models</i>	$ r $ %	$ e $	S	$ P_i $	S	$ P_i $	S	$ P_i $	S	$ P_i $	S	$ P_i $	S	$ P_i $	S
dinopet	34.9%	9,895	201 KB	13	252 KB	67	577 KB	5,607	38 MB	12	205 KB	62	226 KB	1,297	224 KB
elephant	30.4%	10,197	206 KB	13	338 KB	136	1.4 MB	5,349	50 MB	15	215 KB	123	250 KB	1,306	229 KB
bull	42.5%	18,594	379 KB	12	481 KB	211	2.3 MB	12,210	102 MB	12	388 KB	191	446 KB	3,486	444 KB
inner ear	34.0%	48,354	1.0 MB	31	1.4 MB	181	3.6 MB	14,591	171 MB	26	1.0 MB	89	1.1 MB	6,360	1.2 MB
horse	34.4%	59,541	1.3 MB	8	1.4 MB	77	2.4 MB	24,044	527 MB	8	1.3 MB	47	1.3 MB	8,095	1.4 MB
screw-dr	45.5%	81,450	1.8 MB	1	1.8 MB	44	3.0 MB	43,180	2.0 GB	1	1.8 MB	9	1.8 MB	15,052	2.1 MB
bunny	40.5%	104,496	2.3 MB	6	2.5 MB	178	6.6 MB	46,728	2.8 GB	6	2.3 MB	97	2.4 MB	16,549	2.7 MB
teeth	45.5%	349,806	7.9 MB	11	9.4 MB	307	18.8 MB	135,224	7.5 GB	29	8.0 MB	131	8.2 MB	67,059	9.4 MB
female	38.8%	365,163	8.5 MB	5	8.7 MB	67	10.9 MB	145,085	7.2 GB	5	8.5 MB	50	8.6 MB	51,580	9.3 MB
venus	43.8%	403,026	9.3 MB	3	9.5 MB	273	32.8 MB	166,555	18.2 GB	3	9.3 MB	164	9.6 MB	72,190	9.6 MB
armadillo	41.4%	518,916	12.1 MB	11	12.1 MB	98	14.2 MB	726,240	20+ GB	11	12.2 MB	85	12.4 MB	89,839	14.1 MB
david	38.7%	748,893	18.0 MB	models are						10	18.0 MB	170	18.3 MB	85,132	20.1 MB
dragon	42.8%	1,307,170	31.7 MB	not closed						12	31.8 MB	237	32.1 MB	246,053	37.3 MB

Рис. 17: Таблица декомпозиции 14 моделей

В таблице 17:

$|r|$ % - процент невыпуклости модели

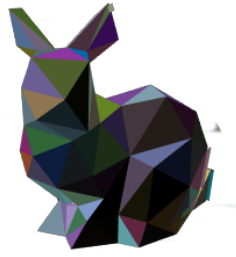
$|e|$ - число ребер модели

S - физический размер модели

$|P_i|$ - число компонент декомпозиции модели



(а) исходный



(b) 200 частей



(с) 3000 частей

Рис. 18: Работа программы

Время работы Время декомпозиции при изменении параметра точности:

- 10 частей - real 0m4.282s
- 200 частей - real 0m11.341s
- 3000 частей - real 0m43.883s
- 7611 частей - real 36m40.183s

7 Заключение

Подведем итоги. Были рассмотрены основные методы декомпозиции, такие как триангуляция Делоне, тетрайдризация, методы точного и приближенного разбиения. Приведены три возможных алгоритма решения NP-полной задачи о невозможности разбиения невыпуклых многогранников, такие как технология CSG, точки Штейнера, полигональный суп. Были сравнены оптимальные стратегии построения и обхода дерева иерархий ограничивающих объемов. Реализован алгоритм приближенного разбиения, работающий в связке с графическим редактором blender.

После изучения темы было обнаружено, что приближенное разбиение с заранее выставленной точностью может решить большинство задач декомпозиции, при этом сильно сэкономив на вычислительных ресурсах и времени разбиения.

8 Список литературы

- [1] **Christer Ericson**. *Real-Time Collision Detection*. Sony Computer Entertainment, Santa Monica, USA, December, 2004.
- [2] **Stephen A. Ehmann, Ming C. Li**. *Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition*. University of North Carolina, Chapel Hill, July, 2002.
- [3] **J.Ruppert, R.Seidel**. *On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra*. University of California, Berkele, December, 1992
- [4] **S. Quinlan**. *Efficient Distance Computation between Non-Convex Objects*. Stanford University, May, 1994
- [5] **Jyh-Ming Lien**. *Approximate Convex Decomposition of Polyhedra*. 2006
- [6] **Xinyu Zhang, Minkyong Lee, Young J. Kim**. *Interactive Continuous Collision Detection for Non-Convex Polyhedra*. 2006
- [7] **Rui S.V. Rodrigues, José F.M. Morgado, Abel J.P. Gomes**. *Part-Based Mesh Segmentation: A Survey*. 2018
- [8] **Michael Manzke**. *Bounding Volume Hierarchies presentation*. The University of Dublin, 2015
- [9] Visualization of Delaunay triangulation computed via the convex hull of lifted 2D points to a 3D paraboloid - <https://github.com/jmlien/cshape2d>
- [10] Delaunay image triangulation - <https://github.com/snorpey/triangulate-image>
- [11] Approximate Convex Decomposition of Polygons - <https://github.com/jmlien/acd2d>
- [12] Bounding Mesh Library - <https://github.com/gaschler/bounding-mesh>
- [13] HACD Library - <https://github.com/kmammou/v-hacd>