Санкт-Петербургский Государственный Университет Математико-механический факультет
Кафедра системного программирования

Построение мотоциклетного графа

Курсовая работа студента 445 группы

Титова Артем Юрьевича

Научный руководитель .... Вяткина К. В.

Санкт-Петербург 2011 год

Введение 3
Основные определения 3
Практическое применение 4
1 Постановка задачи 7
2 Описание алгоритма 8
2.1 Основные понятия 8
2.2 Алгоритм 9
3 Результаты 10
Заключение 14
Список литературы 15

Введение

В данной курсовой работе я расскажу о мотоциклетных графах, а именно об одном сложном алгоритме их построения. Мотоциклетный граф используется в компьютерной графике для решения различных задач[5], я же рассмотрю его применение для построения таких важных структур как прямолинейные скелеты, построение которых является моей конечной целью.

Основные определения

Мотоциклетный граф (motorcycle graph, Рис. 1) - это граф, полученный следующим образом: у нас есть набор из n мотоциклов M_i, для каждого из которых задано начально положение и вектор скорости. Мотоциклы начинают движение. Если два мотоцикла встречаются в одной точке, то они сталкиваются и останавливаются. Если мотоцикл попадает на след другого мотоцикла, то он сталкивается с его следом и останавливается.

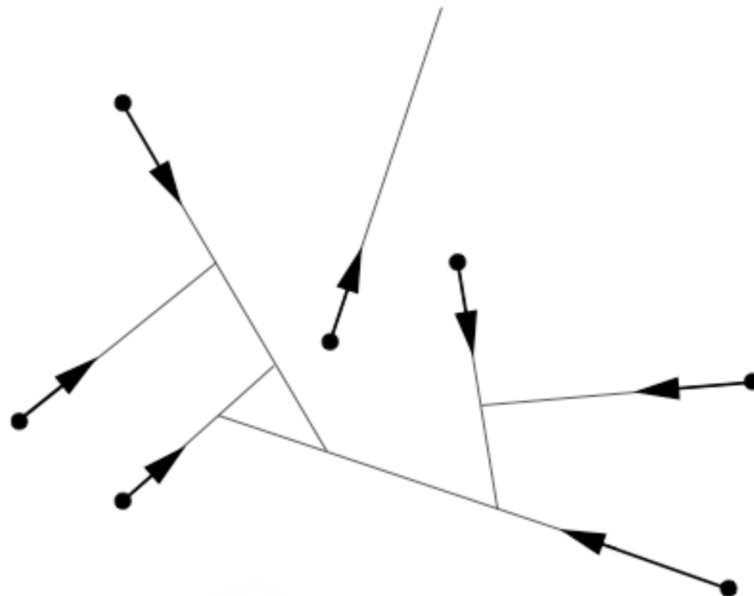


Рис.1. Мотоциклетный граф

Прямолинейный скелет многоугольника (straight skeleton, Рис. 3) это граф, получающийся в результате следующего процесса: берем все ребра и начинаем двигать их внутрь многоугольника, таким образом, чтобы они оставались параллельны изначальным ребрам

многоугольника. При таком движение вершины многоугольника, а также точки, образовавшиеся в результате разрывания вершинами других ребер, будут вычерчивать на плоскости граф, который и будет исходным (Рис. 2).

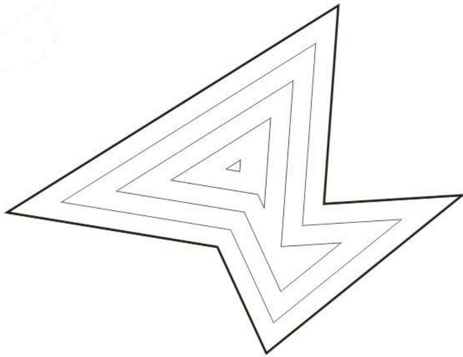


Рис. 2. Процесс построения прямолинейного скелета

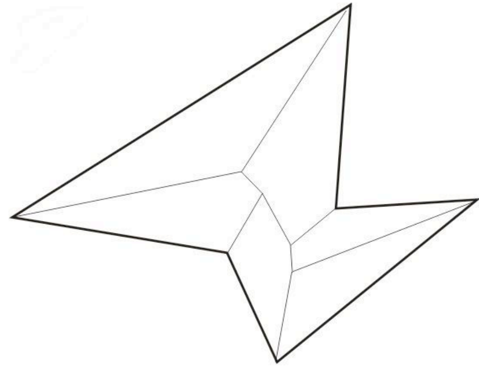


Рис. 3. Прямолинейный скелет

Практическое применение

Моей конечной целью является реализация алгоритма построения прямолинейного скелета, а построение мотоциклетного графа является важным промежуточным звеном этого алгоритма. Поэтому я больше расскажу о практическом применении прямолинейных скелетов, а так как наиболее эффективные методы их построения в качестве вспомогательной структуры используют мотоциклетный граф, то можно сказать, что это также и практическое применение мотоциклетных графов.

Одним из красивых применений прямолинейного скелета является метод, использующий его для построения крыши. Если взять многоугольник, построить его прямолинейный скелет, а затем добавить ребрам, помимо движения внутрь многоугольника, движение вверх с той же скоростью, то у нас получится крыша, грани которой будут находиться под углом в 45° к плоскости многоугольника [6] (Рис. 4).

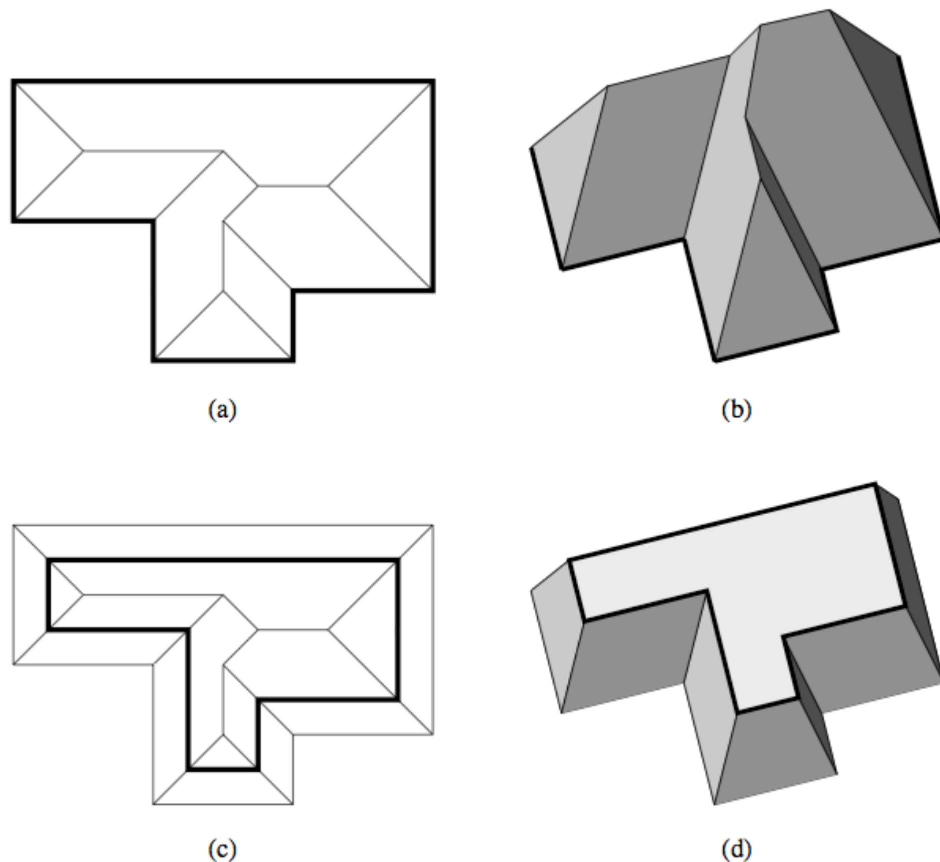


Рис. 4. (a) Прямолинейный скелет для многоугольника. (b) Крыша, построенная на основе прямолинейного скелета. (c) Положение ребер многоугольника в момент времени t при построении прямолинейного скелета. (d) Крыша, построенная к моменту времени t на основе прямолинейного скелета.

Также через прямолинейные скелеты можно решать следующую задачу: есть лист бумаги, нам хочется вырезать в нем заданный многоугольник, при этом сложив его как-то и сделав один прямолинейный разрез [3]

Но одним из самых важных примеров использования прямолинейных скелетов - построение приближения поверхности по ее сечениям[1]. При таком подходе берутся два близлежащих сечения, они аппроксимируются многоугольниками, накладываются друг на друга (верхний проецируется на нижний), после чего определяется их симметрическая разность. Для многоугольников, составляющих симметрическую разность строятся прямолинейные скелеты, и производится триангуляция граней, прилегающих к скелетам. Затем учитывается высота сечений и получается кольцо (Рис. 5). Процесс повторяется для всех пар близлежащих сечений, пока не будет построена вся поверхность. Это

используется, например, в медицине для построения 3D моделей органов человека (Рис. 6, 7).

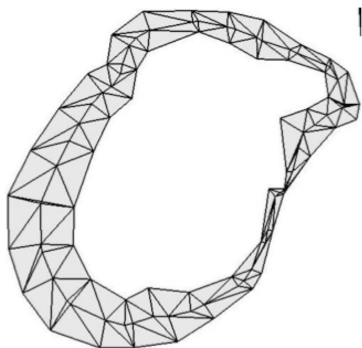


Рис. 5 [1].

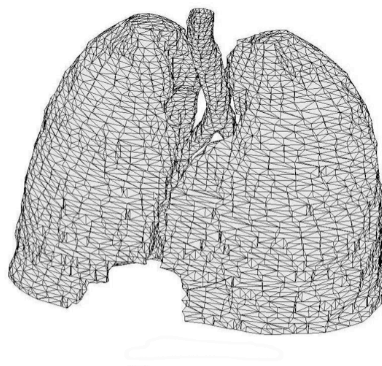


Рис . 6 [1].

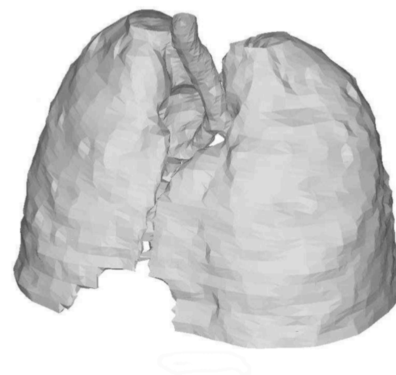


Рис. 7 [1].

1 Постановка задачи

Изначально дается число n - количество мотоциклов, а также n мотоциклов M_i, каждый из которых задается начальной точкой p_i и вектором скорости v_i. Помимо этого вводятся следующие ограничения:

1. все скорости постоянны
2. никакие два мотоцикла не движутся по одной прямой

Задача состоит в том, чтобы построить мотоциклетный граф для этих мотоциклов, реализовав довольно сложный алгоритм [4].

2 Описание алгоритма

2.1 Основные понятия

Итак пусть M_i это мотоцикл, заданный начальной точкой p_i и вектором скорости v_i. Тогда будем говорить, что L_i ***опорная прямая для мотоцикла*** M_i, если она проходит через точку p_i и параллельна вектору v_i. Будем говорить, что луч $T_i(t)$ ***траектория или след мотоцикла*** M_i, если $T_i(t) = p_i + v_i * t$. Заметим, что мотоцикл может столкнуться до того, как достигнет точки $T_i(t)$. ***Точкой столкновения*** является точка пересечения T_i и T_j для некоторых мотоциклов M_i и M_j. Если из мотоциклов M_i и M_j мотоцикл M_i достигает точки столкновения раньше, чем мотоцикл M_j, то он продолжает движение, а мотоцикл M_j сталкивается с его следом и останавливается.

Теперь обратимся к некоторой дополнительной структуре. Нам понадобится разбиение плоскости, называемое $(1/\sqrt{n})$-разбиение. Это разбиение на непересекающиеся треугольники обладает следующим свойством: нам дано n прямых на плоскости, тогда $(1/\sqrt{n})$-разбиение плоскости индуцированное этими прямыми будет разбиение, у которого каждая ячейка содержит не более $(1/\sqrt{n})$ фрагментов этих прямых. Построение такого разбиение достаточно сложно, оно описывается в статье [2]. В этой же работе используется сильно упрощенное разбиение плоскости на треугольники – все мотоциклы описываются квадратом, который разрезается прямыми на треугольники. При этом планируется реализация оригинального разбиения в будущем.

Теперь введем несколько определений. Будем говорить, что мотоцикл M_i ***активен*** в клетке разбиения C в момент времени t, если он находится в C в момент времени t или был в C до момента времени t. Легко заметить, что мотоцикл может взаимодействовать с другими мотоциклами в клетке C, если он находится в ней сейчас, или был там раньше. Этот факт обуславливает то, что мы называем такие клетки активными. Теперь определим два типа рассматриваемых событий:

- ***Переход (switching event) (i, C, t)*** – происходит в момент времени t, когда мотоцикл M_i впервые оказывается в клетке C
- ***Столкновение (impact event) (i, j, t)*** – происходит в момент времени t, когда $T_i(t)=T_i \cap T_j=T_j(t')$ для некоторого $t' \in [0, t]$ (т.е. мотоцикл M_i столкнулся с мотоциклом M_j или со следом мотоцикла M_j)

Итак, пусть K – разбиение плоскости индуцированное опорными прямыми. Теперь давайте будем эмулировать процесс движения мотоциклов по плоскости. Все события-переходы мы можем сгенерировать на этапе инициализации, а в основном этапе алгоритма мы сгенерируем некоторое подмножество столкновений, которого будет достаточно, чтобы построить требуемый граф. Для этого определим ***локальные пересечения*** A(C) для каждой клетки C из K.

2.2 Алгоритм

Вначале мы построим разбиение K. Затем создадим очередь с приоритетом Q – в ней мы будем хранить события в порядке времени. Далее для всех клеток из K добавим в очередь события переходов. Затем сгенерируем события столкновений для клеток, в которых изначально находились мотоциклы и добавим их в очередь. Для этого вычислим для этих клеток локальные пересечения, и в них найдем столкновения мотоциклов (Рис. 8).

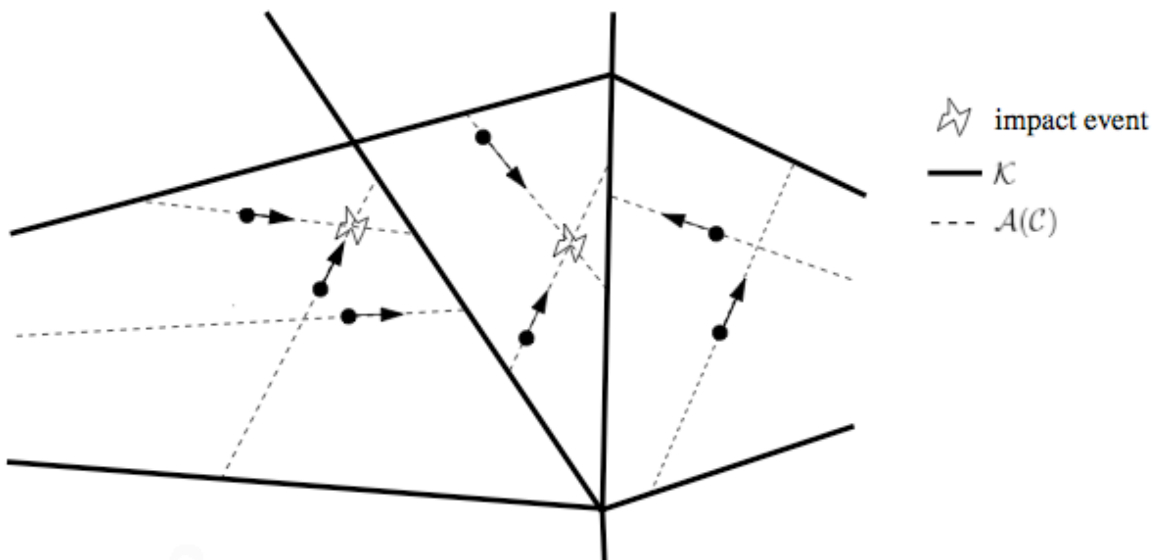
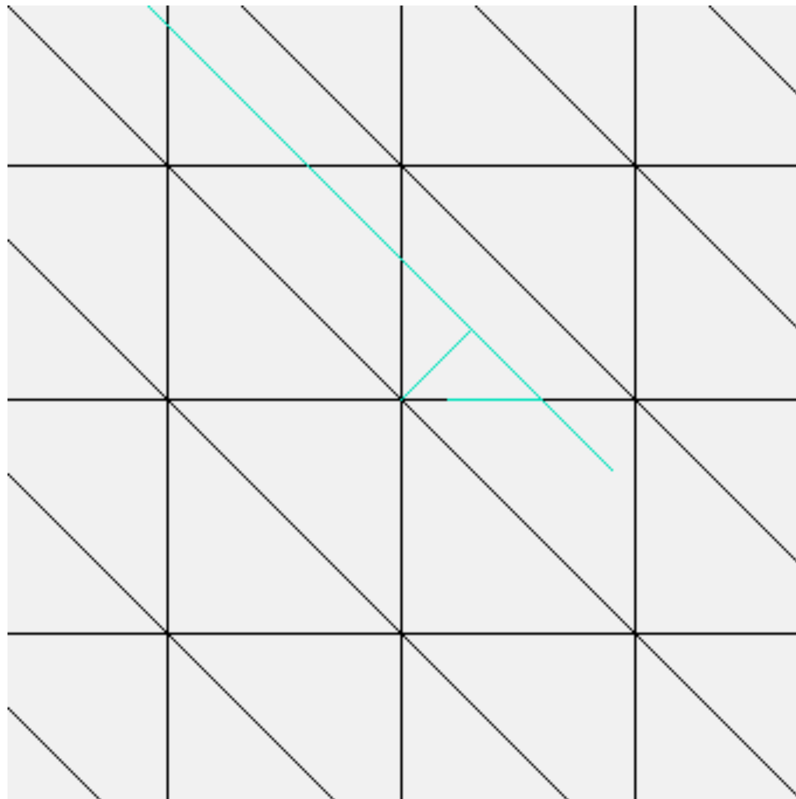


Рис. 8. Этап инициализации. На приведенном рисунке на данном этапе произошло два столкновения

В основном цикле алгоритма мы будем доставать из очереди события и проверять: если это событие перехода, то обновим локальное пересечение соответствующей клетки и добавим в очередь новые столкновения, если же это столкновение (i, j, t), то добавим в ответ ребро, соединяющее начальную точку мотоцикла i с точкой $T_i(t)$, и удалим из очереди все события, связанные с мотоциклом i.

3 Результаты

Результатом моей работы стала работающая программа на языке Java, которая строит мотоциклетный граф. В качестве разбиения плоскости, как я уже говорил, было взято более простое, а именно я заключаю все начальные точки мотоциклов в некоторый квадрат, который разбивается на треугольники. В остальном программа работает в соответствие с алгоритмом. Также приложение позволяет наглядно увидеть процесс построение графа по описанному алгоритму, так как отображает на экране его основные этапы.



Архитектурно программа разбита на два модуля. Первый реализует сам алгоритм, оперируя довольно высокоуровневыми понятиями, такими как клетки разбиения, мотоциклы (Листинг. 1).

Листинг 1. Основной цикл алгоритма

```
while (!queue.isEmpty()) {
	// вытаскиваем следующее по времени событие
	final Event e = queue.pop();
	// если один из мотоциклов, участвующих в этом событии уже разбился, то 
игнорируем его
	if (crashedMotorcycles.contains(e.getCurrentMotorcycle()) ||
	  (e instanceof ImpactEvent && crashedMotorcycles.contains(((ImpactEvent) e).getJ())))
```

```
        {
            continue;
        }
        // если это switching event
        if (e instanceof SwitchingEvent) {
            final SwitchingEvent event = (SwitchingEvent) e;
            drawer.addDrawable(event.getCell(), Color.BLUE);
            // то добавляем в local arrangement клетки новый сегмент, получая список impact
event's
            final List<ImpactEvent> impacts =
event.getCell().updateLocalArrangement(event.getCurrentMotorcycle());
            // добаляем impact event's в очередь
            for (final ImpactEvent i : impacts) {
                queue.push(i.getTime(), i);
            }
            drawer.removeDrawable(event.getCell());
        }
        if (e instanceof ImpactEvent) {
            // если это impact event
            final ImpactEvent event = (ImpactEvent) e;
            // то добавляем кусок от начальной точки мотоцикла, до места его столкновения
в мотоциклетный граф
            final Segment s = new Segment(event.getCurrentMotorcycle().getInitPoint(),
                    event.getCurrentMotorcycle().getPointForTime(event.getTime()));
            motorcycleGraph.add(s);
            drawer.addDrawable(s, Color.decode("#00dead"));
            // помечаем, что мотоцикл разбился
            crashedMotorcycles.add(event.getCurrentMotorcycle());
        }
}
```

Второй же реализует низкоуровневые объекты, такие как внутренние треугольники, прямые, отрезки, нахождение точек пересечения прямых, лучей, отрезков (Листинг 2).

Листинг 2. Класс, описывающий треугольник

```
package ru.spbsu.utils.geometry.objects;

import ru.spbsu.utils.geometry.Drawable;
import ru.spbsu.utils.geometry.GU;
import ru.spbsu.utils.geometry.objects.simple.Point;
import ru.spbsu.utils.geometry.objects.simple.Segment;
import ru.spbsu.utils.graphic.Graphics2DWrapper;

import java.awt.*;

/**
 * Date: 4/18/11
 * Time: 12:39 AM
 *
 * @author Artem Titov
```

```java
 */
public class Triangle extends AbstractGeometryObject implements Drawable {
    private final Point v1;
    private final Point v2;
    private final Point v3;

    public Triangle(final Point v1, final Point v2, final Point v3) {
        this.v1 = v1;
        this.v2 = v2;
        this.v3 = v3;
    }

    public Point getV1() {
        return v1;
    }

    public Point getV2() {
        return v2;
    }

    public Point getV3() {
        return v3;
    }

    public boolean containsPoint(final Point p) {
        final double s = this.getSquare();
        final double s1 = new Triangle(v1, v2, p).getSquare();
        final double s2 = new Triangle(v2, v3, p).getSquare();
        final double s3 = new Triangle(v3, v1, p).getSquare();
        return GU.equals(s, s1 + s2 + s3);
    }

    public double getSquare() {
        final double a = new Segment(v1, v2).length();
        final double b = new Segment(v2, v3).length();
        final double c = new Segment(v3, v1).length();
        if (b * c != 0) {
            final double cosA = (b * b + c * c - a * a) / (2 * b * c);
            final double sinA = Math.sqrt(1 - cosA * cosA);
            return 0.5 * b * c * sinA;
        } else {
            return 0;
        }
    }

    public Segment getFirstSide() {
        return new Segment(v1, v2);
    }

    public Segment getSecondSide() {
        return new Segment(v2, v3);
    }
```

```
    public Segment getThirdSide() {
        return new Segment(v3, v1);
    }

    public boolean isEdge(Segment segment) {
        return new Segment(v1, v2).equals(segment) || new Segment(v2, v3).equals(segment) || new
Segment(v3, v1).equals(segment);
    }

    @Override
    public boolean equals(Object o) {
        if (o == null || !(o instanceof Triangle)) {
            return false;
        }
        final Triangle t = (Triangle) o;
        return (this.v1.equals(t.getV1()) && this.v2.equals(t.getV2()) && this.v3.equals(t.getV3()))
||
                (this.v1.equals(t.getV1()) && this.v2.equals(t.getV3()) && this.v3.equals(t.getV2())) ||
                (this.v1.equals(t.getV2()) && this.v2.equals(t.getV1()) && this.v3.equals(t.getV3())) ||
                (this.v1.equals(t.getV2()) && this.v2.equals(t.getV3()) && this.v3.equals(t.getV1())) ||
                (this.v1.equals(t.getV3()) && this.v2.equals(t.getV1()) && this.v3.equals(t.getV2())) ||
                (this.v1.equals(t.getV3()) && this.v2.equals(t.getV2()) && this.v3.equals(t.getV1()));
    }

    @Override
    public void draw(Graphics2DWrapper graphics, Color color) {
        new Segment(v1, v2).draw(graphics, color);
        new Segment(v2, v3).draw(graphics, color);
        new Segment(v3, v1).draw(graphics, color);
    }
}
```

Изначально оценка на время построение $O(n\sqrt{n} \log n)$. У меня же получилось реализовать несколько худший вариант. Это связано с тем, что было использовано неоптимальное разбиение плоскости, и после реализации последнего оценка должна будет достигнуть теоретической.

Наряду с алгоритмом был реализован некоторый общий компонент, представляющей собой набор классов и методов для работы с геометрическими объектами, которые активно используются для решения поставленной задачи.

Исходные коды доступны в svn http://code.google.com/p/project-banch/source/browse/trunk/StraightSkeleton/, ривизия 291. Я принимал участие под учётной записью titov.artem.u@gmail.com

Заключение

В будущем я хочу реализовать правильное разбиение плоскости, а также построить на основе мотоциклетного графа прямолинейный скелет. Помимо этого хочется обиться теоретической оценки сложности алгоритма на практике и провести сравнение с другими существующими алгоритмами построения скелета.

Список литературы

[1] G. Barequet, M.T. Goodrich, A. Levi-Steiner, and D. Steiner, Contour interpolation by straight skeletons, Graphical Models (GM), vol. 66 (4), 245-260, July 2004

[2] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9:145–158, 1993.

[3] E. D. Demaine, M. L. Demaine, and A. Lubiw. Folding and cutting paper. In Japan Conf. Discrete Comput. Geom., pages 104–118, 1998.

[4] Siu-Wing Cheng, A. Vigneron. Motorcycle Graphs and Straight Skeletons. Algorithmica, 47 (2007), 159-182.

[5] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle graphs: canonical quad mesh partitioning. Proc. 6th Symp. Geometry Processing, Copenhagen, Denmark, 2008. Computer Graphics Forum 27(5):1477-1486, 2008.

[6] D. Eppstein, J. Erickson. Raising Roofs, Crashing Cycles, and Playing Pool: Applications of a Data Structure for Finding Pairwise Interactions. 14th ACM Symp. Comp. Geom., Minneapolis, 1998, pp. 58-67. Disc. Comp. Geom. 22(4):569-592, 1999 (special issue for SCG 1998).