Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig

Masterarbeit

Computational aspects of maxmin (length) triangulations

or: How Greg Found Large Triangles

Winfried Hellmann

2013-07-12 (DRAFT)

Betreuer: Prof. Dr. Sándor Fekete

for my daughter—in the hope that she will understand sometime

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben. Bei den Experimenten sind keine unbeteiligten Dreiecke zu Schaden gekommen.

Braunschweig, den 11. Juli 2013

Abstract

Maxmin length triangulations are just awesome.

Zusammenfassung

Maxmin Triangulationen sind einfach super.

Contents

1. Introduction

...

1 pages

Triangulations, that is subdividing the plane (or a polygon) into triangles, are a popular topic in computational geometry — not only because of their connections to other problems but also due to their practical applications. They can be helpful as a preprocessing in other algorithms or as a tool in geometric proofs. One popular example is the artgallery problem . Another area where triangulations are widely used is mesh generation and approximation of complex geometric structures.

cite

cite

For different use cases the objectives for a triangulation vary. For example, the widely known Delaunay triangulation [2, Section 9.2] tends to avoid ‘"skinny’" triangles and is therefore useful for meshes. One can image many different properties to be optimized: Edge lengths, triangle area, angle, and degree in a vertex are some of them. Many have already been looked into, but for some of them no application is known by now — so they remain theoretical problems. In chapter 3 we will have a brief overview of different kinds of triangulations.

This thesis will focus on the MaxMin Length Triangulation (MMLT). Stated an open problem in 1991 [9], it has been proven to be NP-complete in 2012 [11]. However our assumption is that the hard instances are rare and that random instances can be solved in polynomial time on average. Therefore we provide an algorithmic idea in chapter 4 and its implementation in chapter 5.

Even though there seems to be no known application of MMLT yet, it is Greg’s [18] preferred kind of triangulation.

2. Integer Programming

8 pages:

glue text

Definition 2.1 ((Linear) Integer Program (IP))
An IP is a system of integer variables $x \in \mathbb{Z}^n$ with a set of constraints on them and often an objective function $c^T x$. We consider only the case where the constraints are linear: $Ax \leq b$ or $Ax \geq b$.

glue text

Problem 2.2 (IP in canonical form [22])

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z} \end{aligned}$$

or

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \geq b \\ & x \leq 0 \\ & x \in \mathbb{Z} \end{aligned}$$

Theorem 2.3
Solving IPs is NP-hard.

Proof:
Even the special case where there is no objective function, only binary variables, and only equality constraints is NP-complete [16]. Therefore the more general problem is NP-hard.

2.1. Vertex Cover

glue text

Definition 2.4 (Vertex Cover)
Given an undirected graph $G = (V, E)$, a vertex cover $V_{\text{cover}} \subseteq V$ for G is a vertex set such that every edge $e \in E$ is incident to at least one vertex $v \in V_{\text{cover}}$:

$$\forall e \in E : \exists v \in V_{\text{cover}} : v \in e$$

glue text

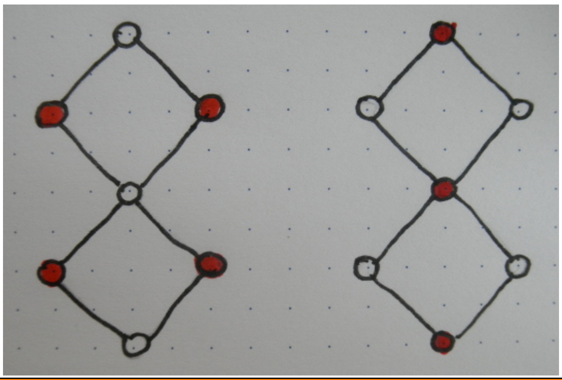
Definition 2.5 (Minimal Vertex Cover)
A vertex cover V_{cover} for an undirected graph $G = (V, E)$ is minimal if there is no vertex $v \in V_{\text{cover}}$ such that $V_{\text{cover}} \setminus \{v\}$ remains a vertex cover.

glue text

Definition 2.6 (Minimum Vertex Cover)
A vertex cover V_{cover} for an undirected graph $G = (V, E)$ is minimum if there is no other vertex cover V'_{cover} for G which has fewer vertices:

$$\forall\ V'_{\text{cover}} \text{ vertex cover} : |V_{\text{cover}}| \leq |V'_{\text{cover}}|$$

glue text



replace

Figure 2.1.: Example of minimal vs. minimum vertex cover

Problem 2.7 (IP Formulation of Minimum Vertex Cover)

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} x_v \\ \text{subject to } \forall \{v, w\} \in E : \quad & x_v + x_w \geq 1 \\ \forall v \in V : \quad & x_v \in \{0, 1\} \end{aligned}$$

glue text

Theorem 2.8
Vertex Cover is NP-complete. [16]

glue text

Definition 2.9 (*Well-covered* Graph)
An undirected graph $G = (V, E)$ is *well-covered* iff every minimal vertex cover for G is also a minimum vertex cover for G. [23]

glue text

Theorem 2.10
In a *well-covered* graph, all minimal vertex covers have the same cardinality. [23]

2.2. Independent Set

glue text

Definition 2.11 (Independent Set)
Given an undirected graph $G = (V, E)$, an independent set $V_{\text{IS}} \subseteq V$ is a vertex set such that no two vertices $v, w \in V_{\text{IS}}$ are incident to the same edge $\{v, w\} \in E$:

$$\forall v \in V_{\text{IS}} : \forall \{v, w\} \in E : w \notin V_{\text{IS}}$$

glue text

Definition 2.12 (Maximal/Maximum Independent Set)
For an undirected graph $G = (V, E)$, an independent set $V_{\text{IS}} \subseteq V$ is maximal if there is no vertex $v \in V \setminus V_{\text{IS}}$ such that $V_{\text{IS}} \cup \{v\}$ remains an independent set. It is maximum if there is no independent set ${V_{\text{IS}}}'$ with larger cardinality.

glue text

Problem 2.13 (IP Formulation of Maximum Independent Set)

$$\begin{aligned}
\text{maximize} \quad & \sum_{v \in V} x_v \\
\text{subject to } \forall \{v, w\} \in E : \quad & x_v + x_w \leq 1 \\
\forall v \in V : \quad & x_v \in \{0, 1\}
\end{aligned}$$

glue text

Theorem 2.14 (Independent Set and Vertex Cover)
For an undirected graph $G = (V, E)$, $V_{\text{IS}} \subseteq V$ is a maximum independent set iff $V_{\text{cover}} = V \setminus V_{\text{IS}}$ is a minimum vertex cover.

Proof:
Let V_{cover} be a vertex cover for G.

$$\begin{aligned}
\forall e \in E : \exists v \in V_{\text{cover}} : v \in e &\iff \forall \{v, w\} \in E : v \in V_{\text{cover}} \vee w \in V_{\text{cover}} \\
&\iff \forall \{v, w\} \in E : \neg(v \notin V_{\text{cover}} \wedge w \notin V_{\text{cover}}) \\
&\iff \forall \{v, w\} \in E : \neg(v \in (V \setminus V_{\text{cover}}) \wedge w \in (V \setminus V_{\text{cover}})) \\
&\iff \forall v \in (V \setminus V_{\text{cover}}) : \forall \{v, w\} \in E : w \notin (V \setminus V_{\text{cover}}) \\
&\iff (V \setminus V_{\text{cover}}) \text{ independent set}
\end{aligned}$$

Assume V_{cover} is a minimum vertex cover for G and the independent set $V_{\text{IS}} = V \setminus V_{\text{cover}}$ is not maximum. Then there is an independent $V'_{\text{IS}} \subseteq V$ with $|V_{\text{IS}}| < |V'_{\text{IS}}|$. But then for the vertex cover $V'_{\text{cover}} = V \setminus V'_{\text{IS}}$ the following holds: $|V'_{\text{cover}}| < |V_{\text{cover}}|$—which is a contradiction to V_{cover} being minimum. The same argumentation applies in the other direction.

glue text

Theorem 2.15
For a *well-covered* graph $G = (V, E)$, every maximal independent set has the same size and is therefore maximum.

Proof:
Theorem 2.15 follows directly from definition 2.9 and theorems 2.10 and 2.14.

Algorithmus 2.1 : Greedy algorithm for independent set

Input : Undirected graph $G = (V, E)$

Output : Maximal independent set $V_{\text{IS}} \subseteq V$ for G

Set $V_{\text{IS}} = \emptyset$

foreach $v \in V$ **do**

  if $\forall \{v, w\} \in E : (w \notin V_{\text{IS}})$ **then**

    Set $V_{\text{IS}} = V_{\text{IS}} \cup \{v\}$

return V_{IS}

examples: fig. 2.1

glue text

glue text

Theorem 2.16 (Correctness and Complexity of Algorithm 2.1)
Algorithm 2.1 always finds a maximal independent set in $O(|E|)$ time.

Proof:
Because the vertices are processed sequentially, for every edge $\{v, w\} \in E$ at most one of v and w is added to V_{IS}. Therefore V_{IS} is an independent set. Additionally every vertex $v \in V$ is processed and if there is no $w \in V_{\text{IS}}$ with $\{v, w\} \in E$ then $v \in V_{\text{IS}}$. So V_{IS} is maximal.

The for-loop runs $|V|$ times but the if-statement is only executed twice for every edge $e \in E$. Every other statement runs in $O(1)$ time. Thus algorithm 2.1 needs $O(|E|)$ time.

glue text

Theorem 2.17
For a *well-covered* graph algorithm 2.1 always finds a maximum independent set in $O(|E|)$ time.

Proof:

...

proof

2.3. Set Cover

...

evaluate if we really need set cover

Problem 2.18 (Minimum Weight Set Cover)

Given: A "universe" (set) of objects U, subsets $S = \{S_i\}$ such that $\bigcup\limits_{S_i \in S} S_i = U$, and a weight function $c : S \to \mathbb{R}_+$

Sought: A set $R \subseteq S$ which covers the universe, i.e. $\bigcup\limits_{S_i \in R} S_i = U$ and minimizes $\sum\limits_{S_i \in R} c(S_i)$

Problem 2.18 is NP-hard [17] and the related decision problem was already one of the problems Karp has shown to be NP-complete [16].

3. Triangulations

...

4 pages

glue text

Definition 3.1 (Triangulation)
Given the complete undirected graph

$$K_{|V|} = (V, E = \{e = \{v, w\} : v, w \in V \land v \neq w\})$$

of a vertex set V and a set of conflicts $X \subseteq \{\{e_i, e_j\} : e_i, e_j \in E \land e_i \neq e_j\}$. A *triangulation* $T(V, X) \subseteq E$ of V with respect to X is a maximum set of non-conflicting edges:

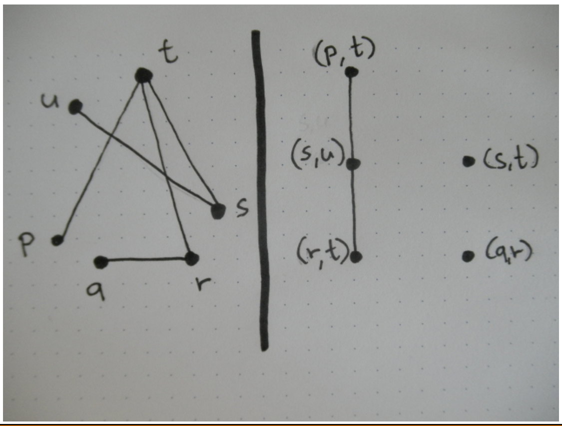
$$e_i \in T(V, X) \iff e_i \in E \land \forall e_j \in T(V, X) : \{e_i, e_j\} \notin C$$

glue text

Definition 3.2 (Conflict Graph)
The conflict graph $G_{\text{conf}}(E, X) = (E, X)$ for a set of edges E and conflicts $X \subseteq \{\{e_i, e_j\} : e_i, e_j \in E \land e_i \neq e_j\}$ is an undirected graph with E as vertices and X as edges.

glue text

Theorem 3.3 (Equality of Triangulation and Maximum Independent Set)
Every triangulation $T(V, X)$ of a vertex set V with respect to conflicts X is a maximum independent set for the conflict graph $G_{\text{conf}}(E, X)$ and vice versa. Hereby E are the edges of the complete undirected graph $K_{|V|}$ for V.



replace

Figure 3.1.: Example of a conflict graph

Proof:
...

proof

Theorem 3.4
From theorem 3.3 and theorem 2.17 follows that finding the triangulation $T(V, X)$ of a vertex set V with respect to conflicts X can be calculated in polynomial time.

glue text

Definition 3.5 (Triangulation with Forbidden Edges)
A *triangulation with forbidden edges* $T(V, X, F)$ is a triangulation of the vertex set V with respect to conflicts X which does not contain any of the edges in F:

$$\forall e \in F : e \notin T(V, X, F)$$

glue text

Theorem 3.6
The decision problem whether a triangulation with forbidden edges $T(V, X, F)$ exists is NP-complete.

cite Lloyd: On Triangulations of a Set of Points in the Plane, triangulation existence problem

Definition 3.7 (Constrained Triangulation)
...

definition

3.1. Point Set Triangulations

glue text

Definition 3.8 (Planar Points)
A *planar set of points* or *set of points in the plane* P is a set of points with two coordinates:

$$P \subseteq \{p = (p_x, p_y) : p_x, p_y \in F\}$$

We do not make any assumptions on the coordinates besides F being a field, e.g. the real numbers $\mathbb{R}$. Furthermore, because P is a set, no duplicate points $p = (p_x, p_y) \in P$ and $p' = (p'_x, p'_y) \in P$ with $p_x = p'_x$ and $p_y = p'_y$ are allowed. Note however that we do not require the points in P to be in general position as that would forbid some interesting instances.

glue text

Definition 3.9 (Line Segments)
A *line segment* $s = (p, q)$ is determined by its endpoints $p, q \in P$ with P being a point set. For compatibility with other definitions, s is directed from p to q, i.e. $(p, q) \neq (q, p)$ and contains all points "between" p and q:

$$m \in s \iff \exists a \in [0, 1] : m = p + a \cdot (q - p)$$

glue text

Definition 3.10 (Crossing)
Two line segments $s_i = (p_i, q_i)$ and $s_j = (p_j, q_j)$ with different slope are *crossing*, if their intersection is not empty and not an endpoint, i.e.

$$s_i, s_j \textit{ crossing} \iff (p = s_i \cap s_j) \wedge (|s_i \cap s_j| = 1) \wedge (p \notin \{p_i, q_i, p_j, q_j\})$$

Two segments s_i and s_j are *non-crossing* if they are not *crossing*. A set S of segments is *crossing* if at least two segments $s_i, s_j \in S$ are *crossing*. It is *non-crossing* if each pair $s_i, s_j \in S$ is *non-crossing*.

glue text

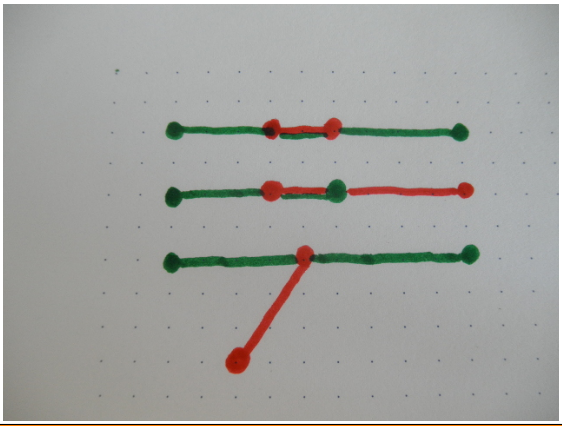
Definition 3.11 (Overlapping Segments)
Given a point set P and a line segment $s = (p, q)$ with $p, q \in P$. s is *overlapping* in P iff there is a point $p' \in P$ which lies in its interior:

$$s \textit{ overlapping} \iff \exists p' \in P : (p' \in s) \wedge (p' \notin \{p, q\})$$

s is *non-overlapping* if it is not *overlapping*.

glue text

Definition 3.12 (Topological Representation)
A vertex set $V(P)$ represents a point set P iff there is exactly one vertex $v_p \in V(P)$ for each point $p \in P$ and v_p can be identified by p and vice versa.



replace

Figure 3.2.: Examples of overlapping segments.

Definition 3.13 (Point Set Triangulation)
The triangulation $T(P)$ of a point set P is a triangulation with forbidden edges $T(P) = T(V, X, F)$ where the vertex set $V = V(P)$ represents P, the conflicts X are all *crossing* segments and the forbidden edges F are the *overlapping* segments:

$$\begin{aligned} \{\{v_p, v_q\}, \{v_{p'}, v_{q'}\}\} \in X &\iff (p, q), (p', q') \textit{ crossing} \\ \{v_p, v_q\} \in F &\iff (p, q) \textit{ overlapping} \end{aligned}$$

For convenience we define $s = (p, q) \equiv e = \{v_p, v_q\}$ such that $s \in T(P) \iff e \in T(P)$. For a slightly different yet equivalent definition of a point set triangulation see also [2, Section 9.1]. Note that for P in general position F is empty.

Theorem 3.14
A point set triangulation can be calculated in polynomial time.

Proof:
...
proof

Definition 3.15 (Constrained Point Set Triangulation)
...
definition

Theorem 3.16
A constrained point set triangulation can be calculated in polynomial time.

Proof:
...
proof

3.2. Polygon Triangulations

definitions (and related work?)

3.3. Edge Flipping

```
- what is a flip?
- flip graph complexity
- local vs. global optimal
- hint to edge flipping paper
```

notes

3.4. Related Work

After class was over, Greg asked Mrs. Minerva if there are different kinds of triangulations. She replied that the problem of triangulating has kept researchers busy for over 100 years already [13] and that people have found different aspects in that a triangulation can be optimal.

The most famous class is the Delaunay triangulation [2, Section 9.2]. It forces every circumcircle of a triangle to be empty of other points and therefore maximizes the minimum angle [2, Theorem 9.9]. There is an edge flipping algorithm which calculates it in $O(n \log n)$ expected time using $O(n)$ space [2, Theorem 9.12].

There are several other triangulations which can be computed in polynomial time. Minimizing the maximum edge length in $O(n^2)$ times was one of the first results [9]. The counterpart of a Delaunay triangulation, minimizing the maximum angle, takes $O(n^2 \log n)$ time and $O(n)$ space [3]. The same approach can also produce triangulations which maximize the minimum height of a triangle. Finally, the same reference shows also that minimizing the maximum slope and minimizing the maximum eccentricity can both be done in $O(n^3)$ time and $O(n^2)$ space. A triangulation which minimizes or maximizes the area of triangles can be computed in $O(n^2 \log n)$ time with $O(n^2)$ space. [24]

Other triangulations have been proven NP-hard or NP-complete. One of them is to minimize the edge length sum (also known as the minimum weight triangulation) which is NP-hard [21]. Maximizing the minimum edge length was stated an open problem [9] but 20 years later it has been shown that it is NP-complete [11]. The latter one remains NP-hard for polygons with holes and interior points [5] but can be solved in $O(n^3)$ time for simple polygons and even in linear time for convex polygons [14].

4. MaxMin Length Triangulation

...

2 pages

During lunch, Greg thought about the different types of triangles Mrs. Minerva had told him about (chapter 3) and which ones he liked the most. On one hand he liked triangles with long edges, on the other hand Greg was very interested in NP-complete problems. Therefore he decided to find out more about MaxMin Length Triangulation (MMLT), which forces the shortest edge to be as long as possible and was proven NP-complete [11].

Problem 4.1 (MMLT)

Given: Set of points in the plane P and implicitly their induced segments S_P (see **??**)

Sought: Triangulation $T_{\text{opt}} \subseteq S_P$ of P which maximizes $\min\limits_{s \in T_{\text{opt}}} |s|$ with $|s|$ being the length of the segment s

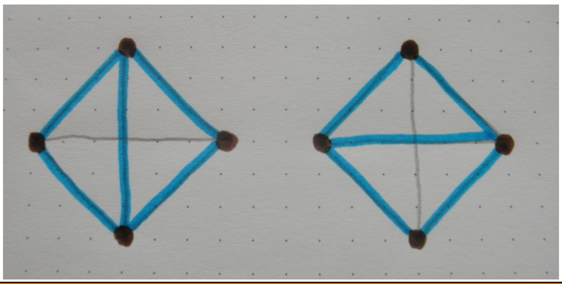
When playing around with some small instances [1], Greg discovered two properties of MMLTs. An optimal solution need not be unique as fig. 4.1 shows. Additionally fig. 4.2 is an example where an edge flip is necessary in a local optimum to gain global optimality. Therefore it is not always possible to retrieve an optimal MMLT solution by starting with an arbitrary triangulation and applying locally optimal edge flips.

explain edge flips

Definition 4.2 (Edge Length Order)
Given a set of edges E representing a set of line segments S and two edges $e_{s_i}, e_{s_j} \in E$ representing two line segments $s_i, s_j \in S$, respectively. Let $|s|$ for $s \in S$ be the segment length and $s_i < s_j$ the lexicographical order of $s_i, s_j \in S$. The edge length order is defined as

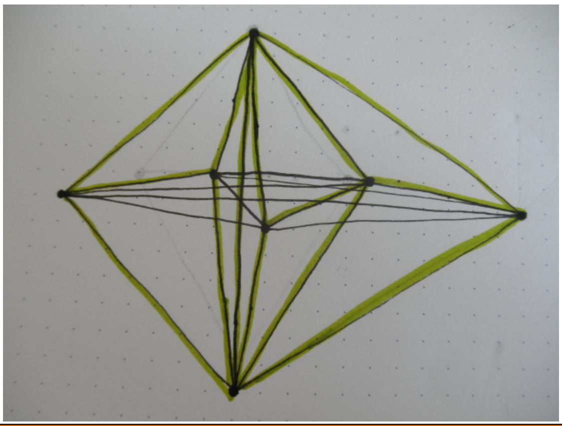
$$e_{s_i} < e_{s_j} \iff |s_i| < |s_j| \vee ((|s_i| = |s_j|) \wedge (s_i < s_j)).$$

[1] Greg drew them on napkins in the cafeteria.



replace

Figure 4.1.: Example of different optimal solutions for the same point set.



replace

Figure 4.2.: Example of necessary locally non-optimal flips.

Definition 4.3 (Edge Length Index)
Given a set of edges E representing a set of line segments S and an edge $e \in E$. The edge length index $|e|$ is the index of e in E sorted by edge length order.

Problem 4.4 (MaxMin Edge Length Index Triangulation (MMELIT))

Given: Set of points in the plane P

Sought: Topological triangulation $T_{\text{opt}} = (V, E)$ of P which maximizes the minimum edge length index: $\min_{e \in E} |e|$.

Theorem 4.5
The optimal edge in a MMELIT solution is unique.

Proof:
...

proof

Theorem 4.6
Every optimal MMELIT solution is an optimal MMLT solution.

Proof:
...

proof

```
IP:
- maximize min. edge index
- no conflicting edges
- for every edge: either edge or crossing edge picked
- n^2 variables, O(n^4) restrictions
```

notes

4.1. Separators

He tried to identify the good, the bad, and the ugly edges. Bad are clearly all short edges as they can worsen the MMLT solution. Step by step, Greg found the set of all segments which were shorter than a certain threshold and named them the “short segments”.

Definition 4.7 (Short Edges)
Given a set of edges E representing a set of line segments S. Short edges within E are all edges with an edge length index smaller than a certain threshold:

$$E_{\text{short}}(P, \ell) := \{e \in E : |e| < \ell\}$$

The next observation Greg made is that there were some long segments which cross the short segments. Greg called them “separators” as they separate the endpoints of short segments. When the shortest segment which is part of the MMLT solution has separators, it can (under certain conditions) be replaced by a longer segment to improve the solution. Therefore separators are the good segments.

Definition 4.8 (Separators)
Given a set of edges E representing a set of line segments S and a set of conflicts $C \subseteq E^2$. The set of separators $E_{\text{sep}}(E, C, e)$ for an edge $e \in E$ are all edges that improve the MMELIT solution, i.e. all in C with e *conflicting* edges which have a higher edge length index:

$$E_{\text{sep}}(E, C, e) := \{e_{\text{sep}} \in E : |e| < |e_{\text{sep}}| \wedge \{e, e_{\text{sep}}\} \in C\}$$

Finally, there are the ugly segments which are short but have no separators such that they can not be replaced [2]. A special case are segments which do not cross at all — for example those on the convex hull. The ugliest segment is the shortest of all ugly segments e_{nose}.

[2]. . . which is not their fault!

Definition 4.9 (Shortest Non-separable Edge)
Given a set of edges E representing a set of line segments S and a set of conflicts $C \subseteq E^2$. The *shortest non-separable edge* e_{nose} is the edge with the smallest edge length index which has no separators:

$$e_{\text{nose}} := \operatorname*{arg\,min}_{e \in E : E_{\text{sep}}(E,C,e) = \emptyset} |e|$$

Greg pities the ugly segments because nobody likes them even though they are not bad. So he tries to find something where they are good at. When the boy thinks back to the art class, he remembers that a triangulation $T \subseteq S_P$ of a point set P is a maximum set of non-crossing segments (**??**). So any segment $s \in S_P$ that is not crossed by another segment $s_\times \in T$ has to be part of T itself. A similar property applies to the ugly segments in a MMELIT: Every segment $s \in S_P$ with no separators is either part of an optimal MMELIT solution T_{opt} or crosses a shorter (or equal length) segment $s_\times \in T_{\text{opt}}$.

Theorem 4.10 (upper bound)
Given a set of edges E representing a set of line segments S and a set of conflicts $C \subseteq E^2$. Let T_{opt} be an optimal MMELIT solution for P. Every segment s without separators is an upper bound for T_{opt}:

$$\forall s \in S_P,\ E_{\text{sep}}(P,s) = \emptyset : \min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| \leq |s|$$

which is equivalent to

$$\forall s \in S_P :\ \neg \exists E_{\text{sep}} \in S_P : |s| < |E_{\text{sep}}| \wedge\ s, E_{\text{sep}}\ \textit{crossing} \implies \min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| \leq |s|$$

Proof:
Assume

$$\exists s \in S_P,\ E_{\text{sep}}(P, s) = \emptyset : \min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| > |s|.$$

This implies $s \notin T_{\text{opt}}$ and

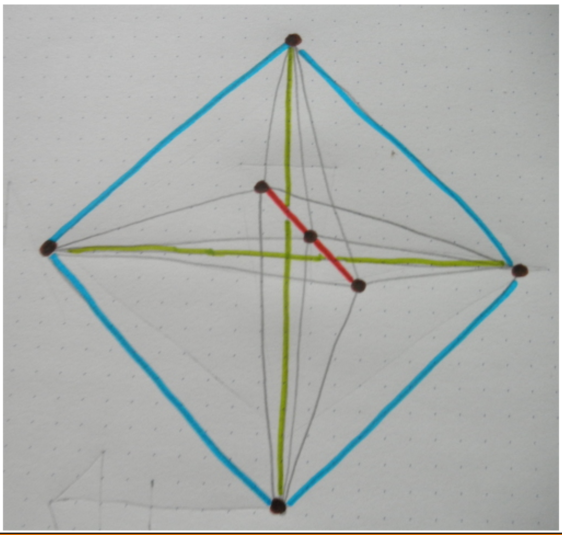
$$\forall s' \in S_P : |s'| \leq |s| \implies s' \notin T_{\text{opt}}.$$

With $E_{\text{sep}}(P, s) = \emptyset$ it follows that

$$\forall s_\times \in S_P : s, s_\times \textit{ crossing} \implies s_\times \notin T_{\text{opt}}.$$

This means that for T_{opt} to be a triangulation s has to be in T_{opt}—which is a contradiction.

Unfortunately, Greg soon found an example where the ugly segments did not help: fig. 4.3. In this instance, there are short segments which have conflicting (crossing) separators. This forces one short segment to be part of the optimal triangulation as only one of the separators can be selected. Therefore the upper bound for an optimal MMLT solution in theorem 4.10 is not tight.



replace

Figure 4.3.: Example where the upper bound from theorem 4.10 is not tight. One of the short segments (red) is part of an optimal MMLT solution because their separators (green) cross. Thus the shortest segment in the solution is shorter than the shortest one of all segments without separators (blue).

Greg remembered that he had met conflicting segments in fig. 4.2 already. By conflicting he meant that the separator E_{sep} of a segment s either crossed any segment $s_\times$ shorter than s or the separator which replaced $s_\times$.

Definition 4.11 (Conflicting Segments)
Given planar point set P and its induced segments S_P. The (potentially) conflicting segments for a segment $s \in S_P$ are:

$$S_{\text{conf}}(P, s) := \bigcup_{s_\times \in E_{\text{short}}(|s|)} s_\times \cup E_{\text{sep}}(s_\times)$$

Greg immediately started to rethink his earlier understanding of ugly segment. His opinion was that short segments with only conflicting separators were as ugly as those with no separators at all. The latter ones are even a special case of the first group. So he changed theorem 4.10 to take conflicting separators into account. A segment may be part of an optimal MMLT solution because all of its separators interfere with other segments. The following theorem assumes knowledge of the optimal triangulation, so conflicting segments can only be found iteratively after the triangulation has already be calculated for shorter segments. However, we will see how conflicts can be formulated independently later on.

Theorem 4.12 (tight upper bound)
Given a point set P and its induced segments S_P. Let T_{opt} be an optimal MMLT solution for P. Every segment s without non-conflicting separators is an upper bound for T_{opt}:

$$E'_{\text{sep}}(P, s) := \{E_{\text{sep}} \in E_{\text{sep}}(P, s) : E_{\text{sep}} \cup (T_{\text{opt}} \cap S_{\text{conf}}(P, s)) \textit{ non-crossing}\}$$

$$\forall s \in S_P : E'_{\text{sep}}(P, s) = \emptyset \implies \min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| \leq |s|$$

can we get T_{opt} out of the bound?

Proof:
This proof is very similar to the proof of theorem 4.10. Assume

$$\exists s \in S_P : E'_{\text{sep}}(P,s) = \emptyset \wedge \min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| > |s|.$$

That implies $s \notin T_{\text{opt}}$ and

$$\forall s' \in S_P : |s'| \leq |s| \implies s' \notin T_{\text{opt}}.$$

If $E_{\text{sep}}(P,s) = \emptyset$ itself, according to theorem 4.10 it holds that

$$\min_{s_{\min} \in T_{\text{opt}}} |s_{\min}| \leq |s|$$

which is a contradiction.

Now assume $E_{\text{sep}}(P,s) \neq \emptyset$. The following holds:

$$\forall E_{\text{sep}} \in E_{\text{sep}}(P,s) : E_{\text{sep}} \cup (T_{\text{opt}} \cap S_{\text{conf}}(P,s)) \text{ } \textit{crossing}.$$

With T_{opt} being a triangulation this implies

$$\forall E_{\text{sep}} \in E_{\text{sep}}(P,s) : E_{\text{sep}} \notin T_{\text{opt}}.$$

Therefore

$$\forall s_c \in S_P,\ s, s_c \textit{ crossing} : s_c \notin T_{\text{opt}}.$$

That means that s has to be in T_{opt} – which is a contradiction.

Theorem 4.13 (tightness)
The bound of theorem 4.12 is tight. That is, let T be a feasible MMLT solution and $s_{\min} = \arg\min_{s \in T} |s|$ with no non-conflicting separators:

$$\neg\exists E_{\text{sep}} \in E_{\text{sep}}(s_{\min}) : E_{\text{sep}} \cup (T_{\text{opt}} \cap S_{\text{conf}}(s_{\min})) \text{ } \textit{non-crossing}$$

Then T is optimal.

Proof:

Assume not optimal, compare optimal, prove not optimal

4.2. Set Cover

As a consequence of theorem 4.12 with theorem 4.13, we introduce the following smaller (in terms of input and output size) problem Non-Conflicting Separators (NCS). For $S = \{s \in S_P : |s| \leq |e_{\text{nose}}|\}$ an optimal solution S_{opt} for NCS has the same shortest segment as an optimal MMLT solution T_{opt}:

$$\min_{s \in S_{\text{opt}}} |s| = \min_{s \in T_{\text{opt}}} |s|$$

Problem 4.14 (NCS)

Given: Set of short segments S and their separators E_{sep}.

Sought: Set of *non-crossing* segments $S_{\text{opt}} \subseteq S \cup \bigcup_{s \in S} E_{\text{sep}}(s)$ which contains for each segment $s \in S$ either the segment itself or at least one separator $E_{\text{sep}} \in E_{\text{sep}}(s)$ and maximizes $\min_{s \in S_{\text{opt}}} |s|$.

The NCS problem can be modeled as a weighted set cover problem. Therefore it is also NP-hard.

```
IP:
- maximize min. edge index
- no conflicting edges
- for every short edge: either edge or separator picked
- worst case: n^2 variables, O(n^4) restrictions
- random points: O(n) variables, < O(n^3) restrictions?
```

notes

Problem 4.15

Given: Set of short segments S and their separators E_{sep}.

Sought: here comes the reduction

4.3. Algorithm

Now that Greg knew all about MMLT he could start developing algorithm 4.1. It consists of three main components: Constructing the partial intersection graph, running a binary search over the short segments and solving the smaller NCS problem in each step, and using the resulting segments in a constrained triangulation.

mention segment index and introduce notation $S_P[i]$

Algorithmus 4.1 : MMLT algorithm

Input : Planar point set P

Output : An optimal MMLT

solution for P

Generate the induced segments S_P for P

Find the shortest non-separable segment e_{nose} and calculate intersections for $E_{\text{short}}(|e_{\text{nose}}|)$ and their separators

Set the lower bound $\mathsf{lb} = 0$

Set the upper bound ub such that $S_P[\mathsf{ub}] = e_{\text{nose}}$

Set $\mathsf{last} = \emptyset$

while $\mathsf{lb} < \mathsf{ub}$ **do**

    Set $\mathsf{mid} = \lceil \frac{\mathsf{lb}+\mathsf{ub}}{2} \rceil$

    Find optimal NCS solution S_{opt} for $E_{\text{short}}(P, |S_P[\mathsf{ub}|)]$ without using any segment from $E_{\text{short}}(P, |S_P[\mathsf{mid}|)]$

    if *there is a solution* S_{opt} **then**

        Set $\mathsf{last} = S_{\text{opt}}$

        Maximize lb such that $|S_P[\mathsf{lb}]| = \min_{s \in S_{\text{opt}}} |s|$

        if $\mathsf{lb} > \mathsf{ub}$ **then**

            $\mathsf{lb} = \mathsf{ub}$

    else

        if $\mathsf{lb} < \mathsf{mid}$ **then**

            Set $\mathsf{ub} = \mathsf{mid} - 1$

        else

            Set $\mathsf{ub} = \mathsf{lb}$

return *constrained triangulation* T_{opt} *with* last *as constraints*

4.4. Intersection Graph

For a point set P with n points in the plane the induced segments (see **??**) have $\Theta(n^4)$ intersections. [19] Thus calculating the intersection graph takes $\Omega(n^4)$ time.

```
- link to sweep line algorithm
- sweep line takes O(n^4 \log n) time => brute force is faster
- adjacency list, because sparse (experiment? proof?)
```

notes

Algorithmus 4.2 : intersection algorithm

Input : Set of segments S

Output : Intersection graph $G = (S, C)$ where $C \subseteq S^2$ are the intersections

Set $C = \emptyset$

foreach $s \in S$ **do**

  foreach $s_\times \in S$ *with* $|s| \leq |s_\times|$ **do**

    if s *and* $s_\times$ *are crossing* **then**

      Add $\{s, s_\times\}$ to C

return $G = (S, C)$

5. Implementation

...

2 pages

To get his algorithm implemented Greg asked Winfried Hellmann, a friend of his who studied computer science. He instantly agreed and started to plan the structure. There were two main components: Geometry and Optimization.

To let the program run close to the hardware layer (which usually leads to fast execution times), the code was written in C++. First attempts to use Python instead (for the sake of clarity and better readability) stumbled over the non-readiness of the CGAL bindings and the lack of good alternatives.

5.1. Geometry

The geometry part consists now of the basics like number, point, and segment types, data structures for triangulation and convex hull, and the intersection algorithm. For most of it Winfried made extensive use of CGAL, which will be introduced in section 5.1.1.

5.1.1. CGAL

CGAL [4] is an Open Source library (mainly) for computational geometry written in C++. It includes most of the common algorithms in the field and also offers efficient data structures. Through the use of C++ templates it is flexible and extendable: For example it is common to adjust the underlying number types to the application.

5.1.2. Kernel

A kernel in CGAL is something like a computational geometry operating system: It holds the basic type definitions like numbers, points, lines, and line segments. Basic operations such as intersection, angle calculations, or comparisons are also part of it.

In our application we use the built-in `Exact_predicates_inexact_constructions_kernel`[1] [10], which uses double as a number type and is not capable of constructing new objects from existing ones accurately. Both properties lead to faster execution time yet do not produce wrong results in our case.

On top of the CGAL kernel there are two modifications: One is for printing points and segments without the need to use streams, the other one to output them to SVG (see also sections 5.3.2 and 5.3.6). Additionally segments are indexed by length and have the information whether they overlap with other segments attached to them.

[1]Thanks to Michael Hemmer for making me aware that I should use it!

5.1.3. Triangulation

CGAL brings along a constrained triangulation already [7] which triangulates a point set with respect to a given mandatory set of (non-crossing) edges. For this application the class was extended to be drawable to SVG and to find the shortest edge which is part of the triangulation.

5.1.4. Convex Hull

This class directly calls the `convex_hull_2` function of CGAL [8] which itself defaults to the algorithm of Akl & Toussaint [1]. The only extension is that the class serves as container which holds the output points and contains a function to find the shortest segment within.

For the algorithm this class is not necessary as its bound is worse than the one of the SAT solution. It is left in the implementation though as a measure of quality for the other bound.

5.1.5. Intersection Graph

```
- crossing segments as conflict edges
- adjacency list
- maybe later replaced by boost graph?
```

5.2. Optimization

The optimization part itself consists mainly of data structures for SAT problems and solutions, an interface for SAT solvers and the solvers themselves (currently only CPLEX).

Only segment indices and intersections are passed to the SAT problem. This is because geometry does not influence the problem, but only topology. Also it is easier that way to keep track of which segments take part in the restrictions.

5.2.1. SAT problem

This class serves two purposes: To grant an interface to the relevant data for the SAT (i.e. segments and intersections) and to set the short segment range.

5.2.2. SAT solution

The SAT solution class mainly just stores the segment indices derived from solving the SAT problem — which can be none if no feasible solution is found. Additionally it offers methods for drawing short segments and separators and for finding the shortest segment of the solution.

5.2.3. SAT solver

To unify the way solving the SAT problem is done, there are three interfaces: The base SAT solver and two derived interfaces for decision and optimization problems. They all share methods for adding forbidden segments, intersection and separation restrictions and for running the actual solving. Furthermore there is a method for binding short segments to the objective function for optimization problems.

5.2.4. CPLEX

IBM ILOG CPLEX [15] is a commercial optimization suite written in C. It contains a standalone tool for solving optimization problems and also includes libraries for being used in other programs or even other programming languages. According to [20] CPLEX is one of the two fastest MILP solvers.

In our application we use the Concert API for C++ [6] to access CPLEX. It allows for adding variables and restrictions to a model, extracting them to more efficient data structures and then running several solving algorithms on it.

5.3. Remains

And then there is the rest: A controller class for the whole algorithm which combines all the other components and utility classes for reading input files, debug output and assertions, test case generation, and drawing certain states of the algorithm to SVG.

5.3.1. Boost

```
https://www.boost.org/

  - boost spirit for JSON parsing
  - maybe later: boost datetime and format for logging,
    boost graph for intersection graph and SVG output
```

5.3.2. Qt

```
https://qt-project.org/

  - event-based programming framework
  - logging
  - SVG output
  - simplified build system and IDE
```

5.3.3. JSON Parser

```
  - JSON spirit (url!)
  - built-in JSON of Qt 5 has bug => no STL/boost/CGAL compatibility
```

5.3.4. Logger

```
- conditional output of different information
- used for evaluation of algorithm output
```

5.3.5. Point Generator

```
- CGAL point generator
- evenly distributed points in (unit) square
```

5.3.6. SVG Painter

```
- conditional algorithm state output
- example?
```

6. Results

...

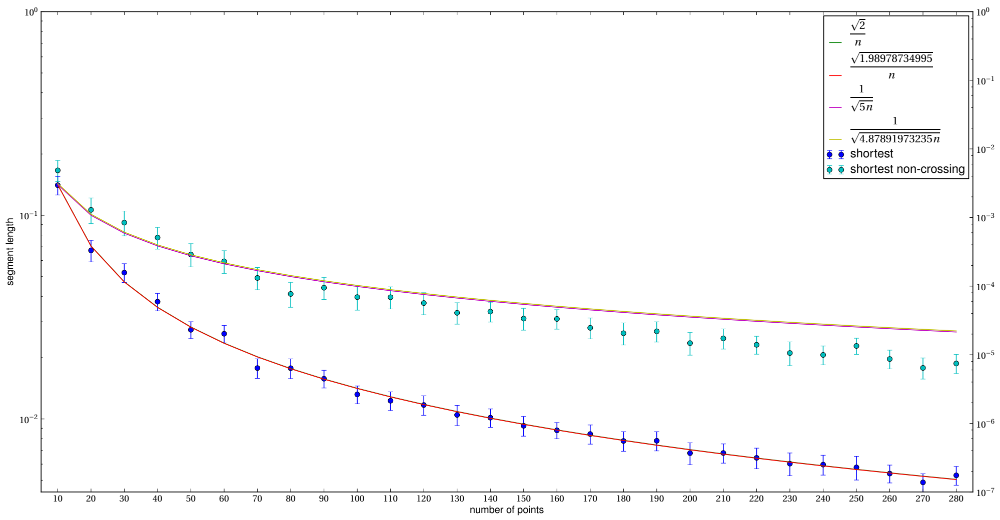
3 pages

some very nice graphs

6.1. Technical Details

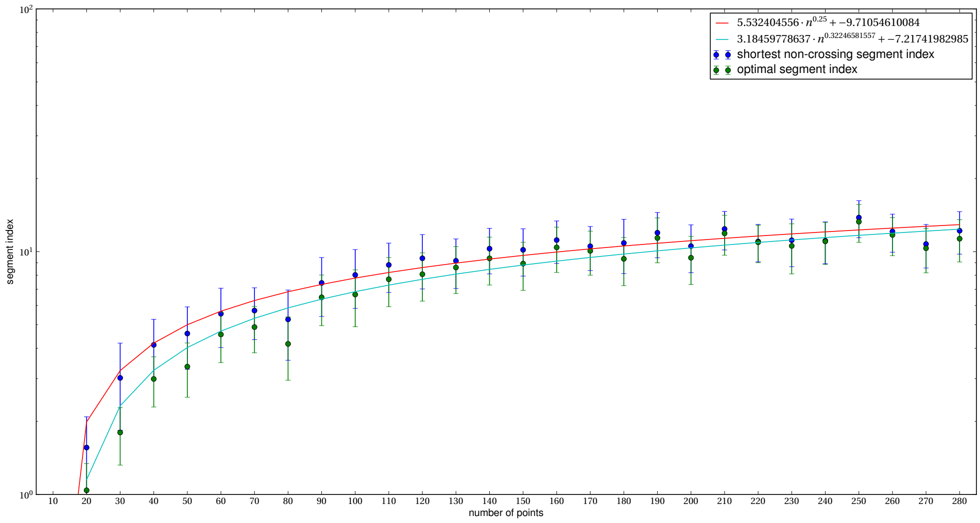
- Intel® Core™ 2 Duo CPU E6850
- 2 GB RAM
- CGAL version

6.2. Segment Lengths



replace

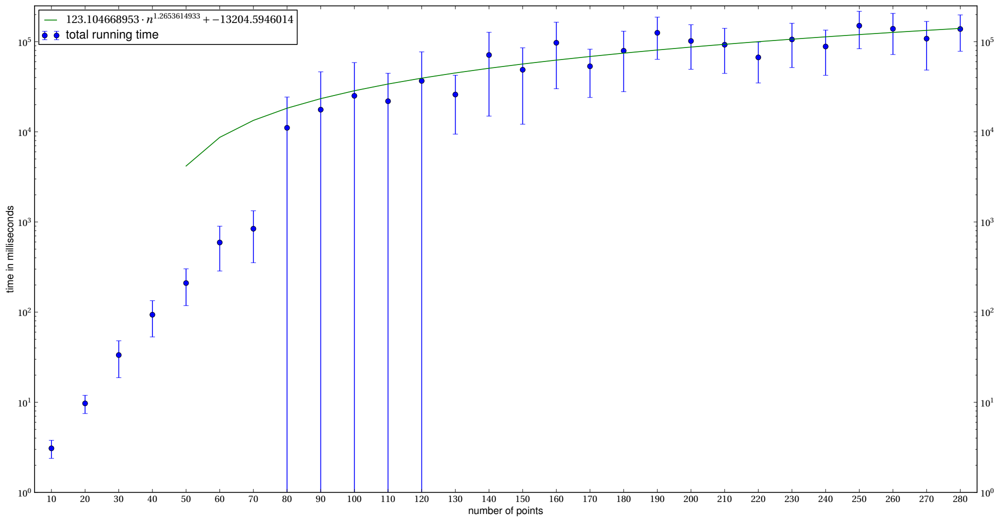
Figure 6.1.: Comparison of segment lengths



replace

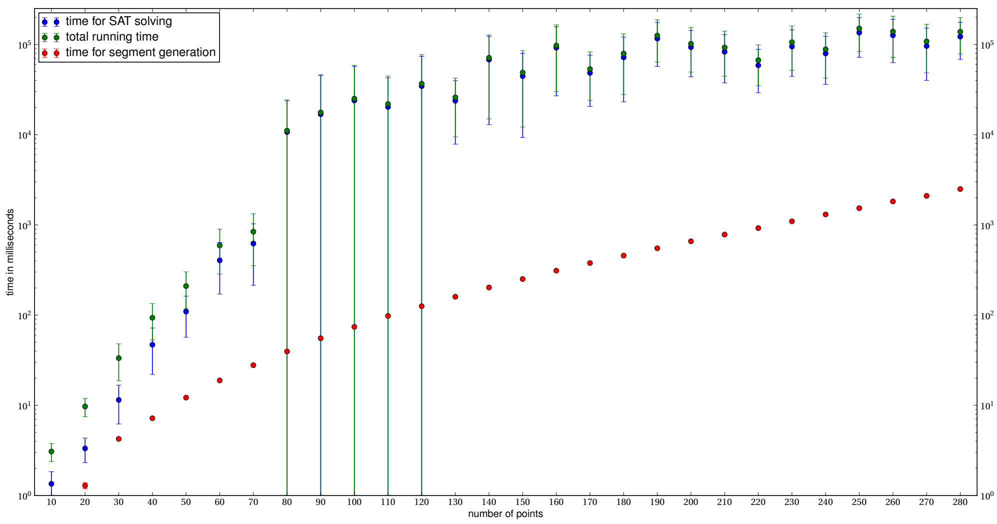
Figure 6.2.: Comparison of segment indices

6.3. Execution Time



replace

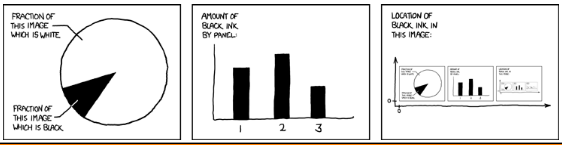
Figure 6.3.: Total execution time



replace

Figure 6.4.: Comparison of execution times

6.4. Aborted instances



replace

Figure 6.5.: Number of aborted instances

7. Conclusion

...

2 pages

With the MaxMin Length Triangulation (MMLT) algorithm at hand Greg made his way to world domination.

be conclusive

A. Documentation

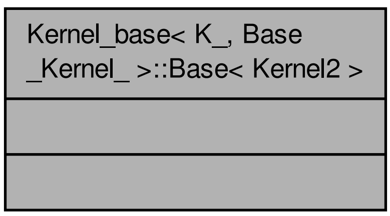
```
- generated with Doxygen
- link to git
- see implementation chapter
```

Contents

1 Class Documentation

1.1 Kernel_base< K_, Base_Kernel_ >::Base< Kernel2 > Struct Template Reference

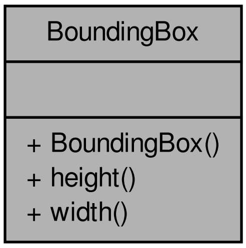
Collaboration diagram for Kernel_base< K_, Base_Kernel_ >::Base< Kernel2 >:



1.2 BoundingBox Class Reference

```
#include <bounding_box.h>
```

Collaboration diagram for BoundingBox:



Public Member Functions

- BoundingBox (const PointSet &points)
- Number height () const
- Number width () const

1.2.1 Constructor & Destructor Documentation

1.2.1.1 BoundingBox::BoundingBox (const PointSet & *points*) [inline]

1.2.2 Member Function Documentation

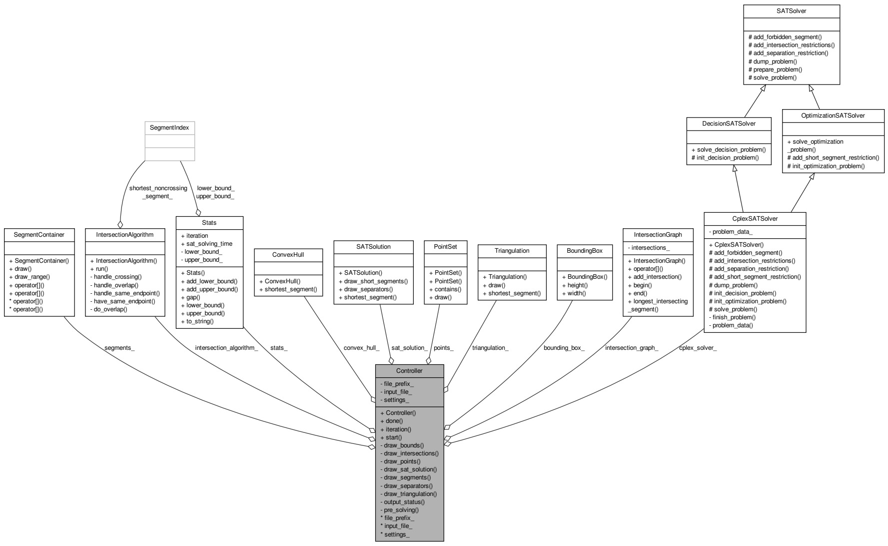
1.2.2.1 Number BoundingBox::height ( ) const [inline]

1.2.2.2 Number BoundingBox::width ( ) const [inline]

1.3 Controller Class Reference

```
#include <controller.h>
```

Collaboration diagram for Controller:



Public Member Functions

- Controller (const QString &file_prefix, QFile &input_file, const QSettings &settings)
- void done ()
- bool iteration ()
- bool start ()

Private Member Functions

- void draw_bounds () const
- void draw_intersections () const
- void draw_points (SVGPainter &painter) const
- void draw_sat_solution () const
- void draw_segments (SVGPainter &painter) const
- void draw_separators () const
- void draw_triangulation () const
- void output_status () const
- void pre_solving ()

Private Attributes

independent members

- CplexSATSolver cplex_solver_
- IntersectionAlgorithm intersection_algorithm_
- SATSolution sat_solution_
- Stats stats_

input parameters

- const QString & file_prefix_
- QFile & input_file_
- const QSettings & settings_

dependent on input parameter

- const PointSet points_

dependent on input points

- const BoundingBox bounding_box_
- const ConvexHull convex_hull_
- SegmentContainer segments_
- Triangulation triangulation_

dependent on segments

- IntersectionGraph intersection_graph_

1.3.1 Constructor & Destructor Documentation

1.3.1.1 Controller::Controller (const QString & *file_prefix*, QFile & *input_file*, const QSettings & *settings*)

1.3.2 Member Function Documentation

1.3.2.1 void Controller::done ( )

called after the algorithm finished

1.3.2.2 void Controller::draw_bounds ( ) const [private]

1.3.2.3 void Controller::draw_intersections ( ) const [private]

1.3.2.4 void Controller::draw_points (SVGPainter & *painter*) const [private]

1.3.2.5 void Controller::draw_sat_solution ( ) const [private]

1.3.2.6 void Controller::draw_segments (SVGPainter & *painter*) const [private]

1.3.2.7 void Controller::draw_separators ( ) const [private]

1.3.2.8 void Controller::draw_triangulation ( ) const [private]

1.3.2.9 bool Controller::iteration ( )

run next iteration

Returns

true if next iteration should be triggered

1.3.2.10 void Controller::output_status ( ) const [private]

dumps the current algorithm status

1.3.2.11 void Controller::pre_solving ( ) [private]

does some pre-processing

1.3.2.12 bool Controller::start ( )

start the algorithm

Returns

true if iteration should be triggered

1.3.3 Member Data Documentation

1.3.3.1 const BoundingBox Controller::bounding_box_ [private]

1.3.3.2 const ConvexHull Controller::convex_hull_ [private]

1.3.3.3 CplexSATSolver Controller::cplex_solver_ [private]

1.3.3.4 const QString& Controller::file_prefix_ [private]

1.3.3.5 QFile& Controller::input_file_ [private]

1.3.3.6 IntersectionAlgorithm Controller::intersection_algorithm_ [private]

1.3.3.7 IntersectionGraph Controller::intersection_graph_ [private]

1.3.3.8 const PointSet Controller::points_ [private]

1.3.3.9 SATSolution Controller::sat_solution_ [private]

1.3.3.10 SegmentContainer Controller::segments_ [private]

1.3.3.11 const QSettings& Controller::settings_ [private]

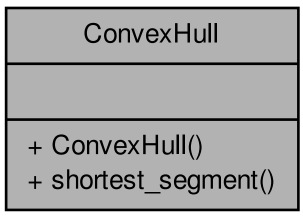
1.3.3.12 Stats Controller::stats_ [private]

1.3.3.13 Triangulation Controller::triangulation_ [private]

1.4 ConvexHull Class Reference

#include <convex_hull.h>

Collaboration diagram for ConvexHull:



Public Member Functions

- ConvexHull (const PointSet &points)
- const SegmentIndex & shortest_segment (const SegmentContainer &segments) const

1.4.1 Constructor & Destructor Documentation

1.4.1.1 ConvexHull::ConvexHull (const PointSet & *points*)

compute convex hull of given point set

1.4.2 Member Function Documentation

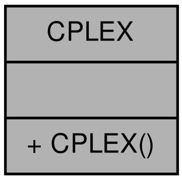
1.4.2.1 const SegmentIndex & ConvexHull::shortest_segment (const SegmentContainer & *segments*) const

find the convex hull segment with minimum length

1.5 CPLEX Class Reference

`#include <concert.h>`

Collaboration diagram for CPLEX:



Public Member Functions

- CPLEX ()

1.5.1 Detailed Description

ugly CPLEX code is not our fault helper class for CPLEX concert API

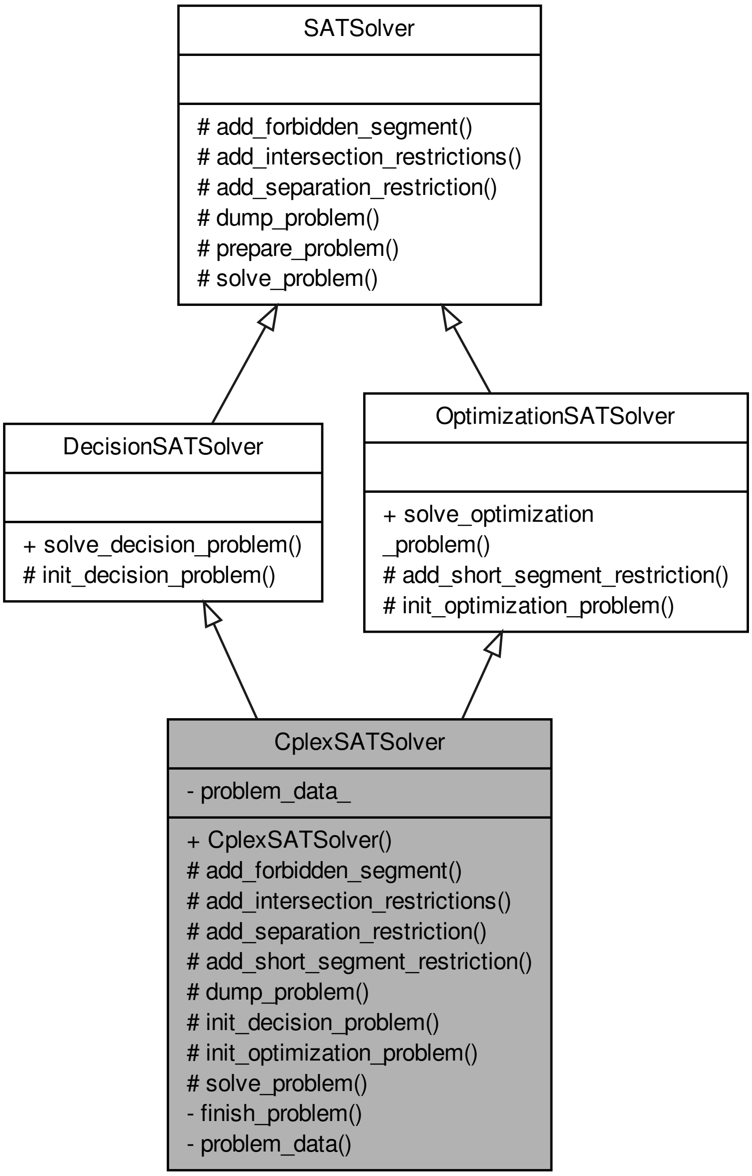
1.5.2 Constructor & Destructor Documentation

1.5.2.1 CPLEX::CPLEX ( ) [inline]

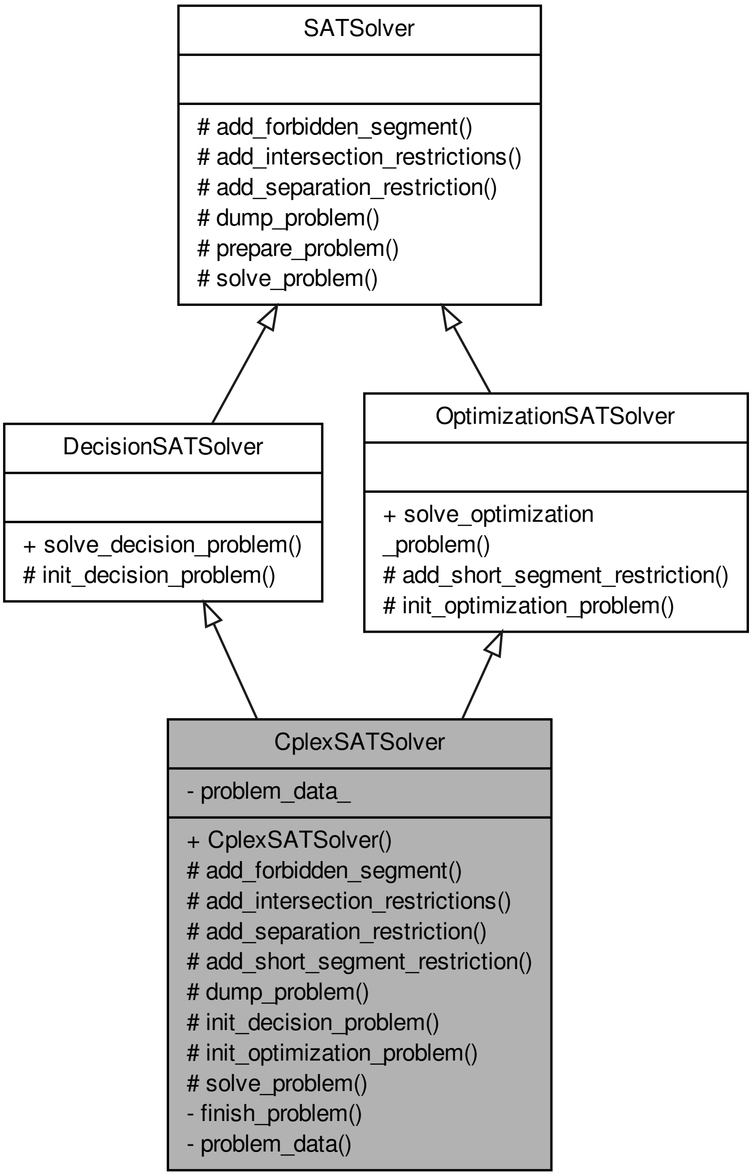
1.6 CplexSATSolver Class Reference

`#include <cplex_sat_solver.h>`

Inheritance diagram for CplexSATSolver:



Collaboration diagram for CplexSATSolver:



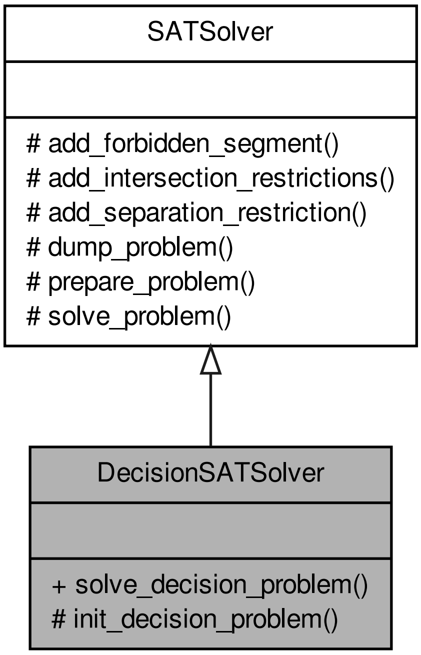
Classes

- struct ProblemData

Public Member Functions

- CplexSATSolver ()

Collaboration diagram for DecisionSATSolver:



Public Member Functions

- void solve_decision_problem (const QSettings &settings, const QString &file_prefix, const SATProblem &problem, SATSolution &solution)

Protected Member Functions

- virtual void init_decision_problem (const SATProblem ∗problem)=0

1.6.1 Detailed Description

interface for decision SAT solvers

1.6.2 Member Function Documentation

1.6.2.1 virtual void DecisionSATSolver::init_decision_problem (const SATProblem ∗ *problem*) [protected], [pure virtual]

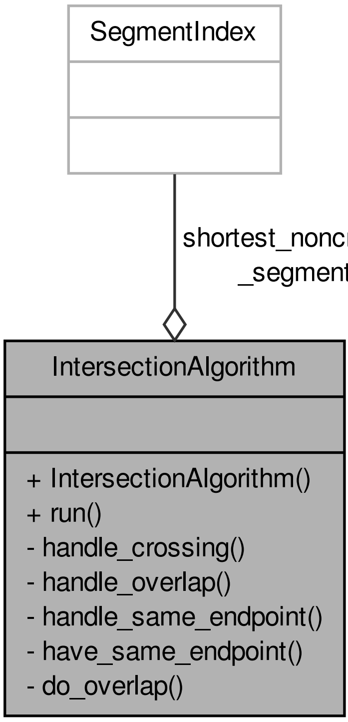
Implemented in CplexSATSolver.

1.6.2.2 void DecisionSATSolver::solve_decision_problem (const QSettings & *settings,* const QString & *file_prefix,* const SATProblem & *problem,* SATSolution & *solution*)

1.7 IntersectionAlgorithm Class Reference

`#include <intersection_algorithm.h>`

Collaboration diagram for IntersectionAlgorithm:



Public Member Functions

- IntersectionAlgorithm ()
- void run (IntersectionGraph &igraph, SegmentContainer &segments)

Public Attributes

- SegmentIndex shortest_noncrossing_segment_

Private Member Functions

- void handle_crossing (IntersectionGraph &igraph, const Segment &s1, const Segment &s2)
- void handle_overlap (IntersectionGraph &igraph, const Segment &s1, const Segment &s2)
- void handle_same_endpoint (const Segment &s1, const Segment &s2) const
- bool have_same_endpoint (const Segment &s1, const Segment &s2) const
- bool do_overlap (Segment &s1, Segment &s2) const

1.7.1 Constructor & Destructor Documentation

1.7.1.1 IntersectionAlgorithm::IntersectionAlgorithm ( )

1.7.2 Member Function Documentation

1.7.2.1 bool IntersectionAlgorithm::do_overlap (Segment & *s1*, Segment & *s2*) const [private]

checks if two segments overlap

Returns

the outer segment

1.7.2.2 void IntersectionAlgorithm::handle_crossing (IntersectionGraph & *igraph*, const Segment & *s1*, const Segment & *s2*) [private]

segments cross

1.7.2.3 void IntersectionAlgorithm::handle_overlap (IntersectionGraph & *igraph*, const Segment & *s1*, const Segment & *s2*) [private]

segments intersect but do not cross

1.7.2.4 void IntersectionAlgorithm::handle_same_endpoint (const Segment & *s1*, const Segment & *s2*) const [private]

segments have the same end point

1.7.2.5 bool IntersectionAlgorithm::have_same_endpoint (const Segment & *s1*, const Segment & *s2*) const [private]

checks if two segments share an endpoint

Returns

the endpoint

1.7.2.6 void IntersectionAlgorithm::run (IntersectionGraph & *igraph*, SegmentContainer & *segments*)

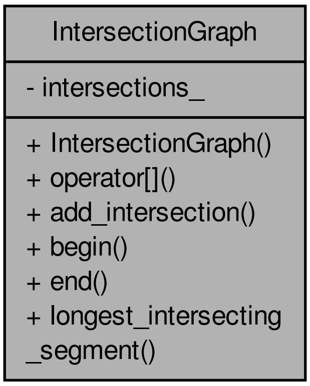
1.7.3 Member Data Documentation

1.7.3.1 SegmentIndex IntersectionAlgorithm::shortest_noncrossing_segment_

1.8 IntersectionGraph Class Reference

`#include <intersection_graph.h>`

Collaboration diagram for IntersectionGraph:



Public Member Functions

- IntersectionGraph (const SegmentIndex &size)
- const Intersections & operator[] (const SegmentIndex &index) const
- void add_intersection (const Segment &s1, const Segment &s2)
- IntersectionsVector::const_iterator begin () const
- IntersectionsVector::const_iterator end () const
- const SegmentIndex & longest_intersecting_segment (const SegmentIndex &index) const

Private Attributes

- IntersectionsVector intersections_

1.8.1 Constructor & Destructor Documentation

1.8.1.1 IntersectionGraph::IntersectionGraph (const SegmentIndex & *size*)

default constructor

1.8.2 Member Function Documentation

1.8.2.1 void IntersectionGraph::add_intersection (const Segment & *s1,* const Segment & *s2*)

add two intersecting segments to the graph

1.8.2.2 IntersectionsVector::const_iterator IntersectionGraph::begin ( ) const [inline]

1.8.2.3 IntersectionsVector::const_iterator IntersectionGraph::end ( ) const [inline]

1.8.2.4 const SegmentIndex & IntersectionGraph::longest_intersecting_segment (const SegmentIndex & *index*) const

1.8.2.5 const Intersections& IntersectionGraph::operator[] (const SegmentIndex & *index*) const [inline]

Returns

all intersecting segments for a segment

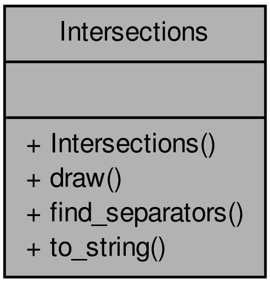
1.8.3 Member Data Documentation

1.8.3.1 IntersectionsVector IntersectionGraph::intersections_ [private]

1.9 Intersections Class Reference

`#include <intersections.h>`

Collaboration diagram for Intersections:



Public Member Functions

- Intersections ()
- void draw (QPainter &painter, const SegmentContainer &segments) const
- void find_separators (const SegmentIndex &segment_index, const SegmentContainer &segments, std::vector< SegmentIndex > &separators) const
- QString to_string (const SegmentContainer &segments) const

1.9.1 Detailed Description

sorted set of intersecting segments

1.9.2 Constructor & Destructor Documentation

1.9.2.1 Intersections::Intersections ( ) [inline]

default constructor

1.9.3 Member Function Documentation

1.9.3.1 void Intersections::draw (QPainter & *painter*, const SegmentContainer & *segments*) const

draws intersections using QPainter

1.9.3.2 void Intersections::find_separators (const SegmentIndex & *segment_index,* const SegmentContainer & *segments,* std::vector< SegmentIndex > & *separators*) const

finds all separators for a given segment and stores them in the passed container

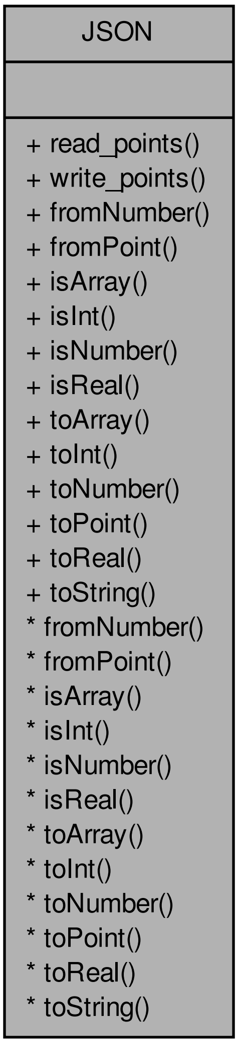
1.9.3.3 QString Intersections::to_string (const SegmentContainer & *segments*) const

output intersections to QString

1.10 JSON Class Reference

```
#include <json.h>
```

Collaboration diagram for JSON:



Static Public Member Functions

- template<typename OutputIterator >
  static bool read_points (QFile &file, OutputIterator output)
- template<typename Container >
  static bool write_points (const std::string &file_name, Container points)

helper functions

- static JSONValue fromNumber (const Number &value)
- static JSONArray fromPoint (const Point &point)
- static bool isArray (const JSONValue &value)
- static bool isInt (const JSONValue &value)
- static bool isNumber (const JSONValue &value)
- static bool isReal (const JSONValue &value)
- static const JSONArray & toArray (const JSONValue &value)
- static int toInt (const JSONValue &value)
- static Number toNumber (const JSONValue &value)
- static Point toPoint (const JSONValue &value)
- static double toReal (const JSONValue &value)
- static const std::string & toString (const JSONValue &value)

1.10.1 Member Function Documentation

1.10.1.1 JSON::JSONValue JSON::fromNumber (const Number & *value*) [static]

1.10.1.2 JSON::JSONArray JSON::fromPoint (const Point & *point*) [static]

1.10.1.3 bool JSON::isArray (const JSONValue & *value*) [static]

1.10.1.4 bool JSON::isInt (const JSONValue & *value*) [static]

1.10.1.5 bool JSON::isNumber (const JSONValue & *value*) [static]

1.10.1.6 bool JSON::isReal (const JSONValue & *value*) [static]

1.10.1.7 template<typename OutputIterator > static bool JSON::read_points (QFile & *file,* OutputIterator *output*) [inline], [static]

1.10.1.8 const JSON::JSONArray & JSON::toArray (const JSONValue & *value*) [static]

1.10.1.9 int JSON::toInt (const JSONValue & *value*) [static]

1.10.1.10 Number JSON::toNumber (const JSONValue & *value*) [static]

1.10.1.11 Point JSON::toPoint (const JSONValue & *value*) [static]

1.10.1.12 double JSON::toReal (const JSONValue & *value*) [static]

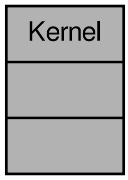
1.10.1.13 const std::string & JSON::toString (const JSONValue & *value*) [static]

1.10.1.14 template<typename Container > static bool JSON::write_points (const std::string & *file_name,* Container *points*) [inline], [static]

1.11 Kernel Struct Reference

```
#include <kernel.h>
```

Collaboration diagram for Kernel:



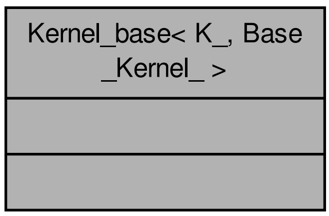
1.11.1 Detailed Description

customized kernel

1.12 Kernel_base< K_, Base_Kernel_ > Class Template Reference

```
#include <kernel.h>
```

Collaboration diagram for Kernel_base< K_, Base_Kernel_ >:



Classes

- struct Base

1.12.1 Detailed Description

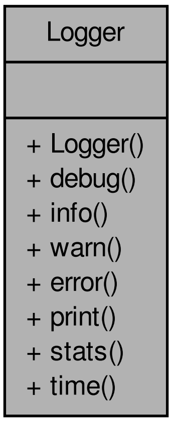
template< typename K_, typename Base_Kernel_ >class Kernel_base< K_, Base_Kernel_ >

kernel base with customized PointC2 and SegmentC2

1.13 Logger Class Reference

```
#include <logger.h>
```

Collaboration diagram for Logger:



Public Member Functions

- Logger ()
- void debug (const QString &message) const
- void info (const QString &message) const
- void warn (const QString &message) const
- void error (const QString &message) const
- void print (const QString &message) const
- void stats (const Stats &stats) const
- void time (const QString &identifier, int milliseconds) const

1.13.1 Constructor & Destructor Documentation

1.13.1.1 Logger::Logger ( )

1.13.2 Member Function Documentation

1.13.2.1 void Logger::debug (const QString & *message*) const

1.13.2.2 void Logger::error (const QString & *message*) const

1.13.2.3 void Logger::info (const QString & *message*) const

1.13.2.4 void Logger::print (const QString & *message*) const

1.13.2.5 void Logger::stats (const Stats & *stats*) const

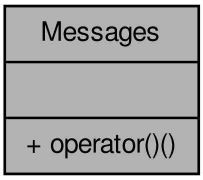
1.13.2.6 void Logger::time (const QString & *identifier,* int *milliseconds*) const

1.13.2.7 void Logger::warn (const QString & *message*) const

1.14 Messages Class Reference

```
#include <logger.h>
```

Collaboration diagram for Messages:



Public Member Functions

- QString operator() (const char ∗text)

1.14.1 Detailed Description

helper class for string literals

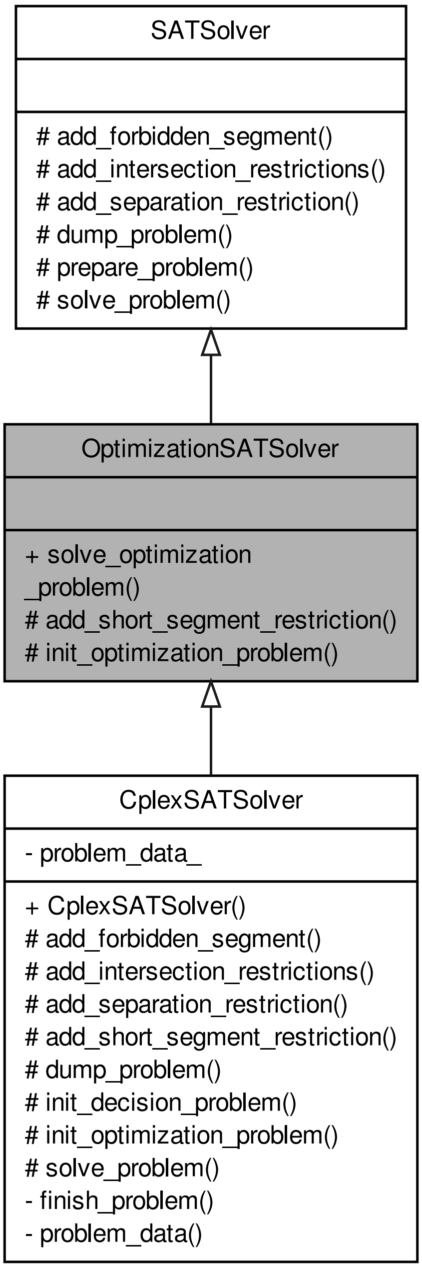
1.14.2 Member Function Documentation

1.14.2.1 QString Messages::operator() (const char ∗ *text*) [inline]

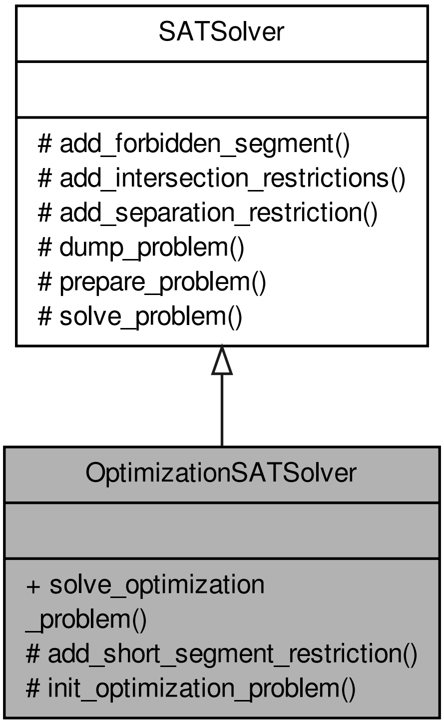
1.15 OptimizationSATSolver Class Reference

`#include <sat_solver.h>`

Inheritance diagram for OptimizationSATSolver:



Collaboration diagram for OptimizationSATSolver:



Public Member Functions

- void solve_optimization_problem (const QSettings &settings, const QString &file_prefix, const SATProblem &problem, SATSolution &solution)

Protected Member Functions

- virtual void add_short_segment_restriction (const SATProblem ∗problem, const SegmentIndex &index)=0
- virtual void init_optimization_problem (const SATProblem ∗problem)=0

1.15.1 Detailed Description

interface for optimization SAT solvers

1.15.2 Member Function Documentation

1.15.2.1 virtual void OptimizationSATSolver::add_short_segment_restriction (const SATProblem ∗ *problem*, const SegmentIndex & *index*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.15.2.2 virtual void OptimizationSATSolver::init_optimization_problem (const SATProblem * *problem*) [protected], [pure virtual]

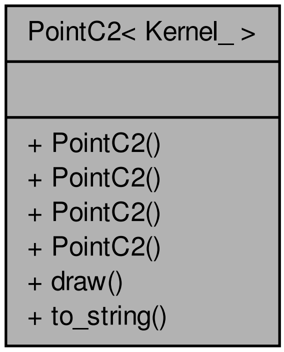
Implemented in CplexSATSolver.

1.15.2.3 void OptimizationSATSolver::solve_optimization_problem (const QSettings & *settings*, const QString & *file_prefix*, const SATProblem & *problem*, SATSolution & *solution*)

1.16 PointC2< Kernel_ > Class Template Reference

```
#include <point.h>
```

Collaboration diagram for PointC2< Kernel_ >:



Public Member Functions

- PointC2 ()
- PointC2 (const CGAL::Origin &origin)
- PointC2 (const FT &x, const FT &y)
- PointC2 (const FT &hx, const FT &hy, const FT &hw)
- void draw (QPainter &painter) const
- QString to_string () const

Private Types

- typedef Kernel_::FT FT
- typedef CGAL::PointC2< Kernel_ > PointBase

1.16.1 Detailed Description

template< class Kernel_ >class PointC2< Kernel_ >

customized point type

1.16.2 Member Typedef Documentation

1.16.2.1 template<class Kernel_ > typedef Kernel_::FT PointC2< Kernel_ >::FT [private]

1.16.2.2 template<class Kernel_ > typedef CGAL::PointC2<Kernel_> PointC2< Kernel_ >::PointBase [private]

1.16.3 Constructor & Destructor Documentation

1.16.3.1 template<class Kernel_ > PointC2< Kernel_ >::PointC2 ( ) [inline]

empty constructor

1.16.3.2 template<class Kernel_ > PointC2< Kernel_ >::PointC2 (const CGAL::Origin & *origin*) [inline]

origin constructor

1.16.3.3 template<class Kernel_ > PointC2< Kernel_ >::PointC2 (const FT & *x*, const FT & *y*) [inline]

Cartesian constructor

1.16.3.4 template<class Kernel_ > PointC2< Kernel_ >::PointC2 (const FT & *hx*, const FT & *hy*, const FT & *hw*) [inline]

homogeneous constructor

1.16.4 Member Function Documentation

1.16.4.1 template<class Kernel_ > void PointC2< Kernel_ >::draw (QPainter & *painter*) const

draw segment using given QPainter

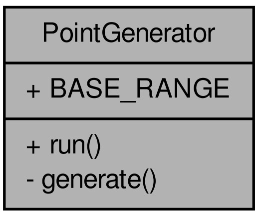
1.16.4.2 template<class Kernel_ > QString PointC2< Kernel_ >::to_string ( ) const

dump point to QString

1.17 PointGenerator Class Reference

`#include <point_generator.h>`

Collaboration diagram for PointGenerator:



Static Public Member Functions

- static void run (const QSettings &settings)

Static Public Attributes

- static const double BASE_RANGE = 100.0

Static Private Member Functions

- template<typename GeneratorType >
  static void generate (const QString &base_name, std::size_t num_points, std::size_t num_iterations)

1.17.1 Member Function Documentation

1.17.1.1 template<typename GeneratorType > static void PointGenerator::generate (const QString & *base_name*, std::size_t *num_points*, std::size_t *num_iterations*) `[inline], [static], [private]`

1.17.1.2 static void PointGenerator::run (const QSettings & *settings*) `[inline], [static]`

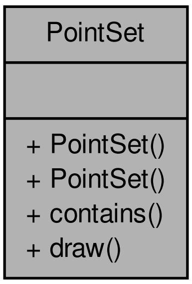
1.17.2 Member Data Documentation

1.17.2.1 const double PointGenerator::BASE_RANGE = 100.0 `[static]`

1.18 PointSet Class Reference

`#include <point_set.h>`

Collaboration diagram for PointSet:



Public Member Functions

- PointSet ()
- PointSet (QFile &input_file)
- bool contains (const Point &point) const
- void draw (QPainter &painter) const

1.18.1 Detailed Description

(sorted) set of points

1.18.2 Constructor & Destructor Documentation

1.18.2.1 PointSet::PointSet ( )

empty set

1.18.2.2 PointSet::PointSet (QFile & *input_file*)

read points from file

1.18.3 Member Function Documentation

1.18.3.1 bool PointSet::contains (const Point & *point*) const [inline]

shortcut for STL count()

Returns

true if point is in set

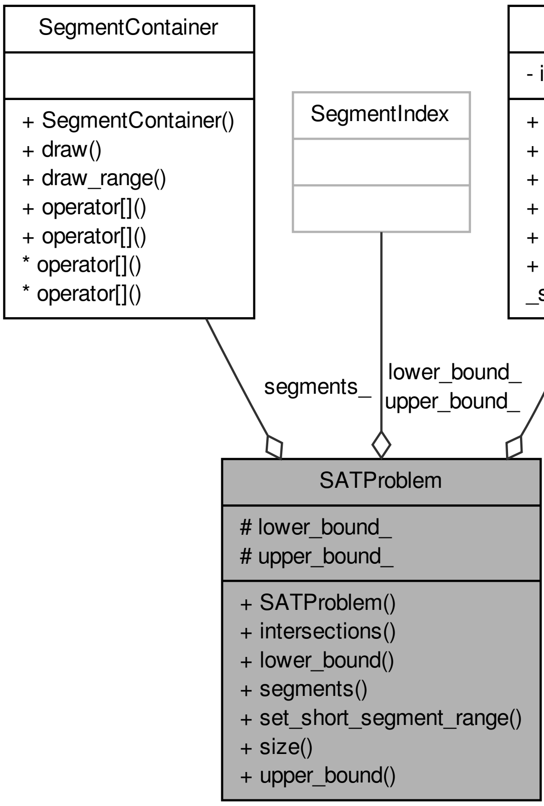
1.18.3.2 void PointSet::draw (QPainter & *painter*) const

output point set using QPainter

1.19 SATProblem Class Reference

```
#include <sat_problem.h>
```

Collaboration diagram for SATProblem:



Public Member Functions

- SATProblem (const IntersectionGraph &igraph, const SegmentContainer &segments)
- const Intersections & intersections (const SegmentIndex &index) const
- const SegmentIndex & lower_bound () const
- const SegmentContainer & segments () const
- void set_short_segment_range (const SegmentIndex &lower_bound, const SegmentIndex &upper_bound)
- SegmentIndex size () const
- const SegmentIndex & upper_bound () const

Protected Attributes

- const IntersectionGraph & igraph_
- const SegmentContainer & segments_
- SegmentIndex lower_bound_
- SegmentIndex upper_bound_

1.19.1 Constructor & Destructor Documentation

1.19.1.1 SATProblem::SATProblem (const IntersectionGraph & *igraph*, const SegmentContainer & *segments*)

default constructor

1.19.2 Member Function Documentation

1.19.2.1 const Intersections& SATProblem::intersections (const SegmentIndex & *index*) const [inline]

1.19.2.2 const SegmentIndex& SATProblem::lower_bound ( ) const [inline]

1.19.2.3 const SegmentContainer& SATProblem::segments ( ) const [inline]

1.19.2.4 void SATProblem::set_short_segment_range (const SegmentIndex & *lower_bound,* const SegmentIndex & *upper_bound*)

set range of segments to consider

1.19.2.5 SegmentIndex SATProblem::size ( ) const [inline]

1.19.2.6 const SegmentIndex& SATProblem::upper_bound ( ) const [inline]

1.19.3 Member Data Documentation

1.19.3.1 const IntersectionGraph& SATProblem::igraph_ [protected]

1.19.3.2 SegmentIndex SATProblem::lower_bound_ [protected]

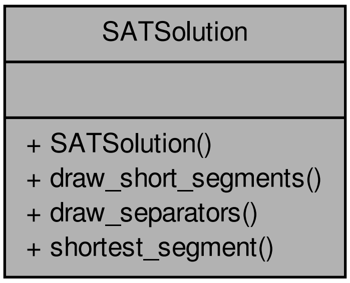
1.19.3.3 const SegmentContainer& SATProblem::segments_ [protected]

1.19.3.4 SegmentIndex SATProblem::upper_bound_ [protected]

1.20 SATSolution Class Reference

`#include <sat_solution.h>`

Collaboration diagram for SATSolution:



Public Member Functions

- SATSolution ()
- void draw_short_segments (QPainter &painter, const SegmentIndex &num_short_segments, const SegmentContainer &segments) const
- void draw_separators (QPainter &painter, const SegmentIndex &num_short_segments, const SegmentContainer &segments) const
- const SegmentIndex & shortest_segment () const

1.20.1 Constructor & Destructor Documentation

1.20.1.1 SATSolution::SATSolution ( ) [inline]

1.20.2 Member Function Documentation

1.20.2.1 void SATSolution::draw_separators (QPainter & *painter*, const SegmentIndex & *num_short_segments*, const SegmentContainer & *segments*) const

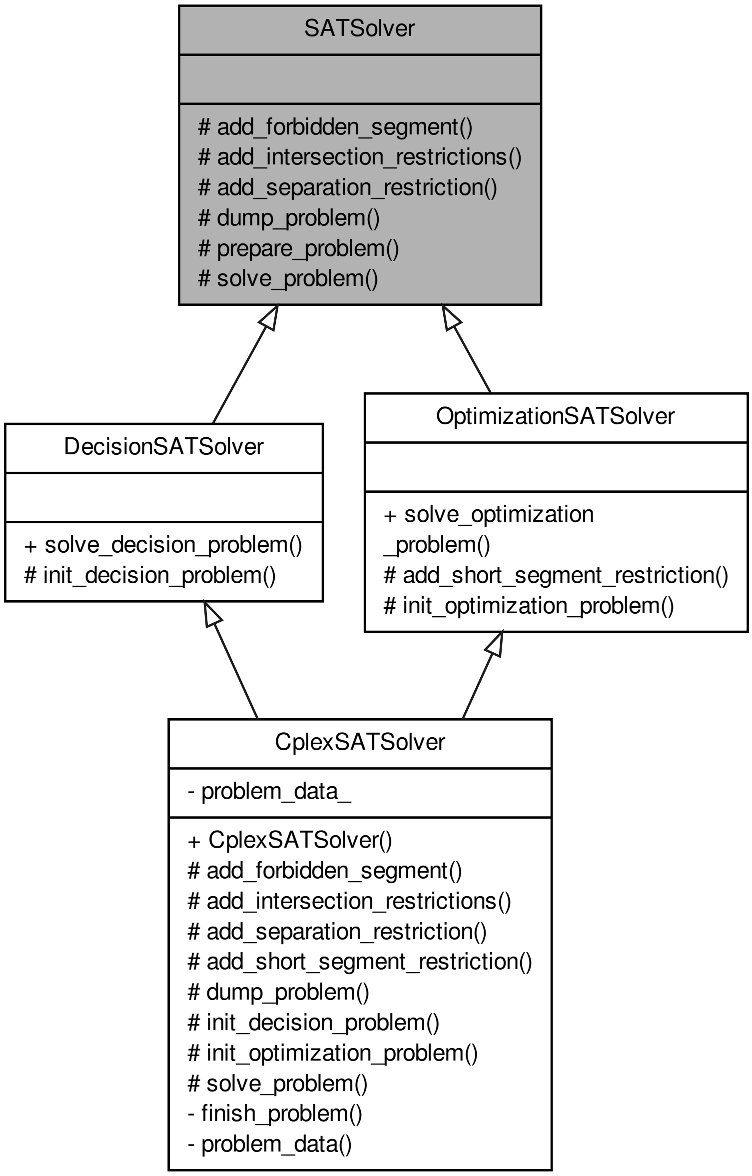
1.20.2.2 void SATSolution::draw_short_segments (QPainter & *painter*, const SegmentIndex & *num_short_segments*, const SegmentContainer & *segments*) const

1.20.2.3 const SegmentIndex & SATSolution::shortest_segment ( ) const

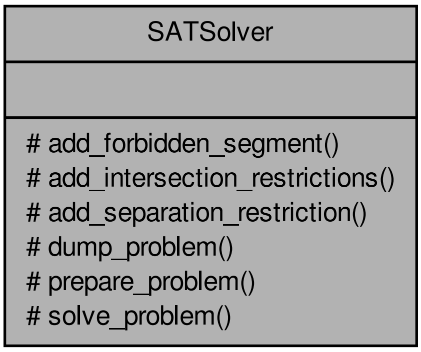
1.21 SATSolver Class Reference

```
#include <sat_solver.h>
```

Inheritance diagram for SATSolver:



Collaboration diagram for SATSolver:



Protected Member Functions

- virtual void add_forbidden_segment (const SATProblem ∗problem, const SegmentIndex &index)=0
- virtual void add_intersection_restrictions (const SATProblem ∗problem, const SegmentIndex &index, const Intersections &igroup)=0
- virtual void add_separation_restriction (const SATProblem ∗problem, const SegmentIndex &index, const std::vector< SegmentIndex > &separators)=0
- virtual void dump_problem (const QString &file_prefix, const SATProblem ∗problem)=0
- void prepare_problem (const SATProblem &problem)
- virtual void solve_problem (const SATProblem ∗problem, SATSolution &solution)=0

1.21.1 Detailed Description

interface for SAT solvers

1.21.2 Member Function Documentation

1.21.2.1 virtual void SATSolver::add_forbidden_segment (const SATProblem ∗ *problem,* const SegmentIndex & *index*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.21.2.2 virtual void SATSolver::add_intersection_restrictions (const SATProblem ∗ *problem,* const SegmentIndex & *index,* const Intersections & *igroup*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.21.2.3 virtual void SATSolver::add_separation_restriction (const SATProblem ∗ *problem,* const SegmentIndex & *index,* const std::vector< SegmentIndex > & *separators*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.21.2.4 virtual void SATSolver::dump_problem (const QString & *file_prefix,* const SATProblem ∗ *problem*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.21.2.5 void SATSolver::prepare_problem (const SATProblem & *problem*) [protected]

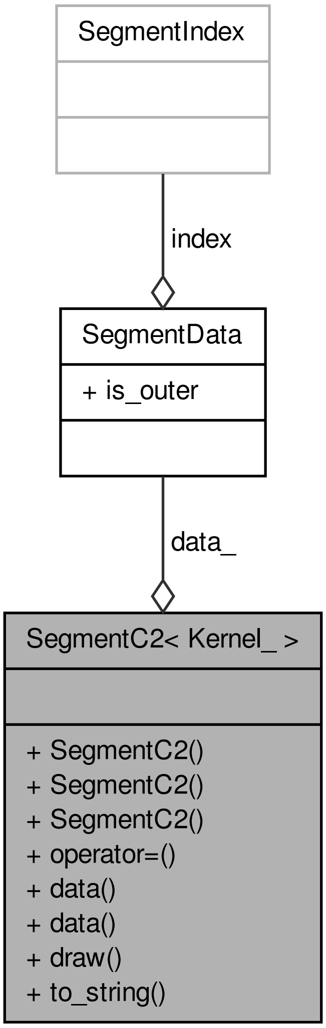
1.21.2.6 virtual void SATSolver::solve_problem (const SATProblem * *problem,* SATSolution & *solution*) [protected], [pure virtual]

Implemented in CplexSATSolver.

1.22 SegmentC2< Kernel_ > Class Template Reference

`#include <segment.h>`

Collaboration diagram for SegmentC2< Kernel_ >:



Public Member Functions

- SegmentC2 ()
- SegmentC2 (const Point_2 &source, const Point_2 &target)

- SegmentC2 (const SegmentC2 &other)
- SegmentC2 & operator= (const SegmentC2 &other)
- SegmentData & data ()
- const SegmentData & data () const
- void draw (QPainter &painter) const
- QString to_string () const

Private Attributes

- SegmentData data_

1.22.1 Detailed Description

template<class Kernel_>class SegmentC2< Kernel_ >

customized segment type

1.22.2 Constructor & Destructor Documentation

1.22.2.1 template<class Kernel_ > SegmentC2< Kernel_ >::SegmentC2 ( ) [inline]

empty constructor

1.22.2.2 template<class Kernel_ > SegmentC2< Kernel_ >::SegmentC2 (const Point_2 & *source*, const Point_2 & *target*) [inline]

base constructor

1.22.2.3 template<class Kernel_ > SegmentC2< Kernel_ >::SegmentC2 (const SegmentC2< Kernel_ > & *other*) [inline]

copy constructor

1.22.3 Member Function Documentation

1.22.3.1 template<class Kernel_ > SegmentData& SegmentC2< Kernel_ >::data ( ) [inline]

getter for attached data

1.22.3.2 template<class Kernel_ > const SegmentData& SegmentC2< Kernel_ >::data ( ) const [inline]

constant getter for attached data

1.22.3.3 template<class Kernel_ > void SegmentC2< Kernel_ >::draw (QPainter & *painter*) const

draw segment using given QPainter

1.22.3.4 template<class Kernel_ > SegmentC2& SegmentC2< Kernel_ >::operator= (const SegmentC2< Kernel_ > & *other*) [inline]

assignment operator

1.22.3.5 template<class Kernel_ > QString SegmentC2< Kernel_ >::to_string ( ) const

dump segment to QString

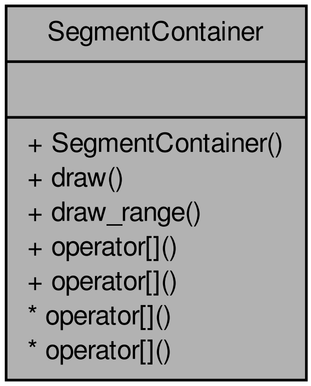
1.22.4 Member Data Documentation

1.22.4.1 template<class Kernel_ > SegmentData SegmentC2< Kernel_ >::data_ [private]

1.23 SegmentContainer Class Reference

#include <segment_container.h>

Collaboration diagram for SegmentContainer:



Public Member Functions

- SegmentContainer (const PointSet &points)
- void draw (QPainter &painter) const
- void draw_range (QPainter &painter, const SegmentIndex &lower_bound, const SegmentIndex &upper_-bound) const

access i-th shortest segment

these operators assume that the segment set is not changed after construction

- Segment & operator[] (const SegmentIndex &index)
- const Segment & operator[] (const SegmentIndex &index) const

1.23.1 Detailed Description

container of segments sorted by length

1.23.2 Constructor & Destructor Documentation

1.23.2.1 SegmentContainer::SegmentContainer (const PointSet & *points*)

construct segments for all point pairs from set

1.23.3 Member Function Documentation

1.23.3.1 void SegmentContainer::draw (QPainter & *painter*) const

draws all segments

1.23.3.2 void SegmentContainer::draw_range (QPainter & *painter,* const SegmentIndex & *lower_bound,* const SegmentIndex & *upper_bound*) const

draw a range of segments

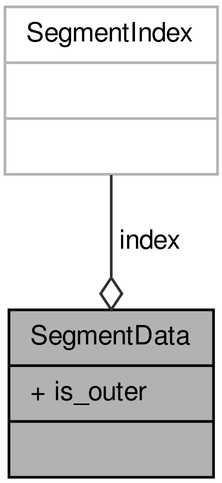
1.23.3.3 Segment & SegmentContainer::operator[] (const SegmentIndex & *index*)

1.23.3.4 const Segment & SegmentContainer::operator[] (const SegmentIndex & *index*) const

1.24 SegmentData Struct Reference

`#include <segment.h>`

Collaboration diagram for SegmentData:



Public Attributes

- SegmentIndex index
- bool is_outer

1.24.1 Detailed Description

data attached to a segment

1.24.2 Member Data Documentation

1.24.2.1 SegmentIndex SegmentData::index

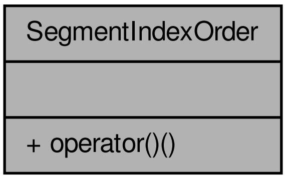
1.24.2.2 bool SegmentData::is_outer

true if the segment includes another

1.25 SegmentIndexOrder Struct Reference

```
#include <orders.h>
```

Collaboration diagram for SegmentIndexOrder:



Public Member Functions

- CGAL::Comparison_result operator() (const Segment &s, const Segment &t) const

1.25.1 Detailed Description

CGAL order for Segment by index

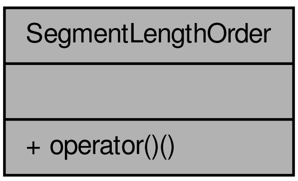
1.25.2 Member Function Documentation

1.25.2.1 CGAL::Comparison_result SegmentIndexOrder::operator() (const Segment & *s*, const Segment & *t*) const

1.26 SegmentLengthOrder Struct Reference

```
#include <orders.h>
```

Collaboration diagram for SegmentLengthOrder:



Public Member Functions

- CGAL::Comparison_result operator() (const Segment &s, const Segment &t) const

1.26.1 Detailed Description

CGAL order for Segment by length

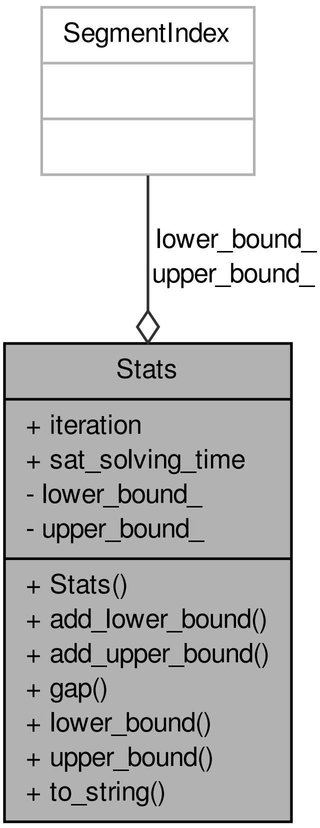
1.26.2 Member Function Documentation

1.26.2.1 CGAL::Comparison_result SegmentLengthOrder::operator() (const Segment & *s*, const Segment & *t*) const

1.27 Stats Class Reference

`#include <stats.h>`

Collaboration diagram for Stats:



Public Member Functions

- Stats ()
- void add_lower_bound (const SegmentIndex &bound)

- void add_upper_bound (const SegmentIndex &bound)
- SegmentIndex gap () const
- const SegmentIndex & lower_bound () const
- const SegmentIndex & upper_bound () const
- QString to_string () const

Public Attributes

- size_t iteration
- quint64 sat_solving_time

Private Attributes

- SegmentIndex lower_bound_
- SegmentIndex upper_bound_

1.27.1 Constructor & Destructor Documentation

1.27.1.1 Stats::Stats ( ) [inline]

1.27.2 Member Function Documentation

1.27.2.1 void Stats::add_lower_bound (const SegmentIndex & *bound*) [inline]

1.27.2.2 void Stats::add_upper_bound (const SegmentIndex & *bound*) [inline]

1.27.2.3 SegmentIndex Stats::gap ( ) const [inline]

1.27.2.4 const SegmentIndex& Stats::lower_bound ( ) const [inline]

1.27.2.5 QString Stats::to_string ( ) const [inline]

1.27.2.6 const SegmentIndex& Stats::upper_bound ( ) const [inline]

1.27.3 Member Data Documentation

1.27.3.1 size_t Stats::iteration

1.27.3.2 SegmentIndex Stats::lower_bound_ [private]

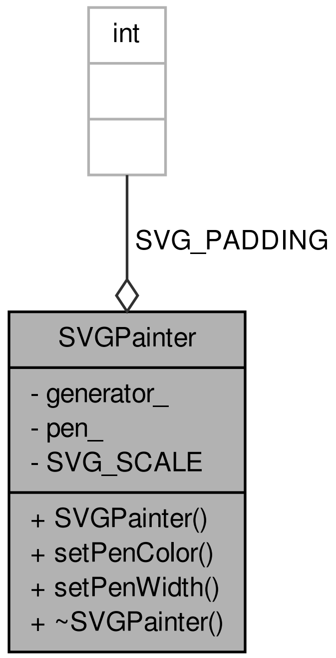
1.27.3.3 quint64 Stats::sat_solving_time

1.27.3.4 SegmentIndex Stats::upper_bound_ [private]

1.28 SVGPainter Class Reference

```
#include <svg_painter.h>
```

Collaboration diagram for SVGPainter:



Public Member Functions

- SVGPainter (const QString &file_prefix, const QString &file_name, const BoundingBox &bbox)
- void setPenColor (const QColor &color)
- void setPenWidth (int width)
- ∼SVGPainter ()

Private Attributes

- QSvgGenerator generator_
- QPen pen_

Static Private Attributes

- static const int SVG_PADDING = 10
- static const double SVG_SCALE = 4.0

1.28.1 Constructor & Destructor Documentation

1.28.1.1 SVGPainter::SVGPainter (const QString & *file_prefix*, const QString & *file_name*, const BoundingBox & *bbox*)

1.28.1.2 SVGPainter::∼SVGPainter ( )

1.28.2 Member Function Documentation

1.28.2.1 void SVGPainter::setPenColor (const QColor & *color*)

1.28.2.2 void SVGPainter::setPenWidth (int *width*)

1.28.3 Member Data Documentation

1.28.3.1 QSvgGenerator SVGPainter::generator_ [private]

1.28.3.2 QPen SVGPainter::pen_ [private]

1.28.3.3 const int SVGPainter::SVG_PADDING = 10 [static], [private]

padding for SVG images

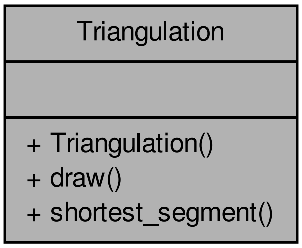
1.28.3.4 const double SVGPainter::SVG_SCALE = 4.0 [static], [private]

scale for SVG images

1.29 Triangulation Class Reference

`#include <triangulation.h>`

Collaboration diagram for Triangulation:



Public Member Functions

- Triangulation (const PointSet &points)
- void draw (QPainter &painter) const
- const SegmentIndex & shortest_segment (const SegmentContainer &segments) const

1.29.1 Constructor & Destructor Documentation

1.29.1.1 Triangulation::Triangulation (const PointSet & *points*)

default constructor

1.29.2 Member Function Documentation

1.29.2.1 void Triangulation::draw (QPainter & *painter*) const

draw triangulation segments using given QPainter

1.29.2.2 const SegmentIndex & Triangulation::shortest_segment (const SegmentContainer & *segments*) const

Glossary

Glossary

e_{nose} . . . . shortest non-separable edge

e_{sep} . . . . separating edge

$s_{\times}$ . . . . crossing segment

IP . . . . Integer Program

IS . . . . Independent Set

MMELIT . . . . MaxMin Edge Length Index Triangulation

MMLT . . . . MaxMin Length Triangulation

NCS . . . . Non-Conflicting Separators

Properties

conflicting . . . . see **??**

crossing . . . . see definition 3.10

non-crossing . . . . see definition 3.10

non-overlapping . . . . see definition 3.11

overlapping . . . . see definition 3.11

well-covered . . . . see definition 2.9

Sets

E_{sep} . . . . set of separators (see definition 4.8)

E_{short} . . . . set of short edges (see definition 4.7)

G_{conf} . . . . conflict graph (see definition 3.2)

S_{conf} . . . . set of potentially conflicting separators (see theorem 4.12)

S_{opt} . . . . an optimal set of segments

T_{opt} . . . . an optimal triangulation

V_{cover} . . . . vertex cover (see **??**)

Bibliography

[1] Selim G. Akl and Godfried T. Toussaint. „A Fast Convex Hull Algorithm". In: *Information Processing Letters* 7.5 (1978), pp. 219–222.

[2] Mark de Berg et al. *Computational Geometry: Algorithms and Applications*. Third. Springer-Verlag, 2008, p. 386. URL: `http://www.cs.uu.nl/geobook/`.

[3] Marshall W. Bern et al. „Edge Insertion for Optimal Triangulations." In: *Discrete & Computational Geometry* 10 (1993), pp. 47–65. DOI: `10.1007/BF02573962`.

[4] *CGAL - Computational Geometry Algorithms Library*. CGAL Open Source Project. 2013. URL: `https://www.cgal.org/` (visited on 2013-06-19).

[5] Christiane Schmidt. „Maxmin Length Triangulation in Polygons". In: *Proceedings of the 28th European Workshop on Computational Geometry*. 2012, pp. 121–124.

[6] *Concert Technology for C++ users*. Version 12.2. IBM Corporation. URL: `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/topic/ilog.odms.cplex.help/Content/Optimization/Documentation/CPLEX/_pubskel/CPLEX197.html` (visited on 2013-06-25).

[7] *Constrained_triangulation_2<Traits,Tds,Itag>*. Version 4.2. CGAL Open Source Project. 2013-04-09. URL: `https://www.cgal.org/Manual/latest/doc_html/cgal_manual/Triangulation_2_ref/Class_Constrained_triangulation_2.html` (visited on 2013-06-19).

[8] *convex_hull_2*. Version 4.2. CGAL Open Source Project. 2013-04-09. URL: `https://www.cgal.org/Manual/latest/doc_html/cgal_manual/Convex_hull_2_ref/Function_convex_hull_2.html` (visited on 2013-06-19).

[9] Herbert Edelsbrunner, Tiow, and Seng Tan. „A Quadratic Time Algorithm for the MinMax Length Triangulation". In: *SIAM J. Comput* 22 (1991), pp. 414–423.

[10] *Exact_predicates_inexact_constructions_kernel*. Version 4.2. CGAL Open Source Project. 2013-04-09. URL: `https://www.cgal.org/Manual/latest/doc_html/cgal_manual/Kernel_23_ref/Class_Exact_predicates_inexact_constructions_kernel.html` (visited on 2013-06-19).

[11] Sándor P. Fekete. „The Complexity of MaxMin Length Triangulation". In: *CoRR* abs/1208.0202 (2012).

[12] Peter Greenberg. „Piecewise SL_2Z Geometry". In: *Transactions of the American Mathematical Society* 335.2 (1993), pp. 705–720. DOI: `10.2307/215440`.

[13] David Hilbert. *Grundlagen der Geometrie*. 2nd ed. B. G. Teubner, 1903, p. 39.

[14] Shiyan Hu. „A linear time algorithm for max-min length triangulation of a convex polygon“. In: *Inf. Process. Lett.* 101.5 (2007-03), pp. 203–208. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2006.09.014. URL: http://dx.doi.org/10.1016/j.ipl.2006.09.014.

[15] *IBM CPLEX Optimizer - United States*. Version 17. IBM Corporation. 2013-05-09. URL: https://www.ibm.com/software/commerce/optimization/cplex-optimizer/ (visited on 2013-06-25).

[16] Richard M. Karp. „Reducibility Among Combinatorial Problems“. In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, p. 94. ISBN: 0-306-30707-3.

[17] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. 5th ed. Springer Publishing Company, Incorporated, 2012, p. 414. ISBN: 9783642244872.

[18] Sue Wheeler Lloyd. *Greg Likes Triangles for Lunch*. Authorhouse, 2011. ISBN: 978-1452098838.

[19] László Lovász et al. „Convex quadrilaterals and k-sets“. In: *Contemporary Mathematics Series, 342, AMS 2004*. 2004, pp. 139–148.

[20] H. Mittelmann. *Mixed Integer Linear Programming Benchmark (MIPLIB2010)*. 2013-05-25. URL: http://plato.asu.edu/ftp/milpc.html (visited on 2013-06-25).

[21] Wolfgang Mulzer and Günter Rote. „Minimum-weight triangulation is NP-hard“. In: *CoRR* abs/cs/0601002 (2006).

[22] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications Inc., 2000. ISBN: 978-0486402581.

[23] Michael D. Plummer. „Some covering concepts in graphs“. In: *Journal of Combinatorial Theory* 8.1 (1970), pp. 91–98. ISSN: 0021-9800. DOI: 10.1016/S0021-9800(70)80011-4. URL: http://www.sciencedirect.com/science/article/pii/S0021980070800114.

[24] Tzvetalin Simeonov Vassilev. „Optimal Area Triangulation“. PhD thesis. University of Saskatchewan, Saskatoon, 2005. URL: http://ecommons.usask.ca/bitstream/handle/10388/etd-08232005-111957/thesisFF.pdf.

Liste der noch zu erledigenden Punkte