B-Spline methods for the design of smooth spatial paths with obstacle avoidance

Stefano Martina
stefano.martina@stud.unifi.it

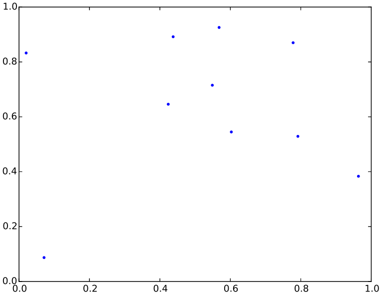
UNIVERSITÀ DEGLI STUDI FIRENZE

15 July 2016

Voronoi diagrams

Input: Set of points in plane (or space) called sites

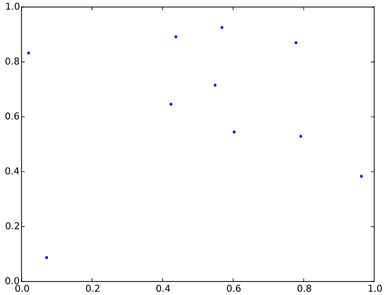
Output: partition of the plane (or space) such that each point of a region is closer to a certain site respect to others

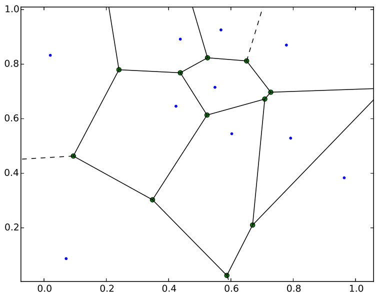


Voronoi diagrams

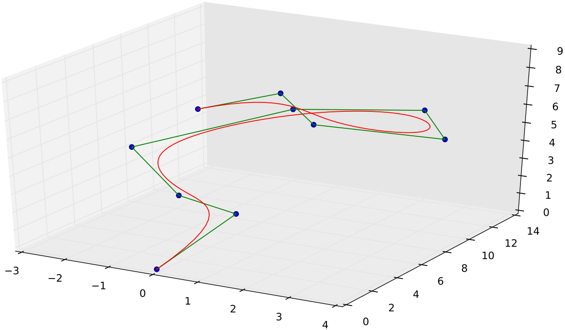
Input: Set of points in plane (or space) called sites

Output: partition of the plane (or space) such that each point of a region is closer to a certain site respect to others



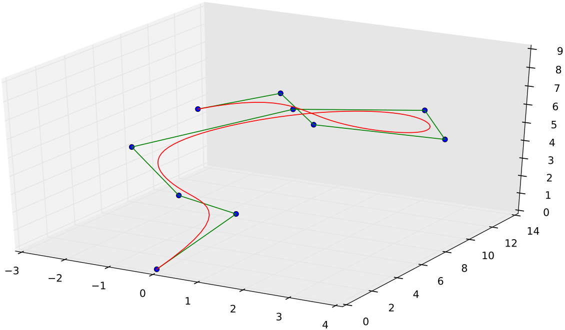


B-spline curves



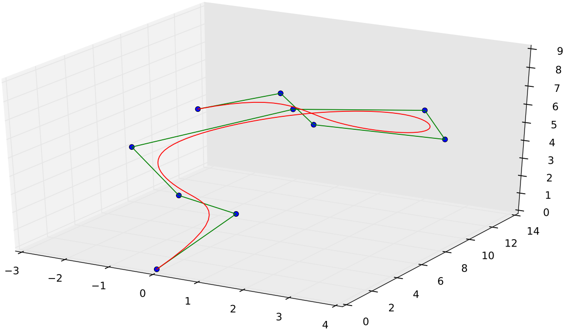
- ✔ Piecewise polynomial parametric curves $\boldsymbol{S} : [a, b] \to \mathbb{E}^3$
  $\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$
- ✔ Prescribed regularity
- ✔ Follow the shape of a control poligon
- ✔ Can interpolate the extremes of control polygon

B-spline curves



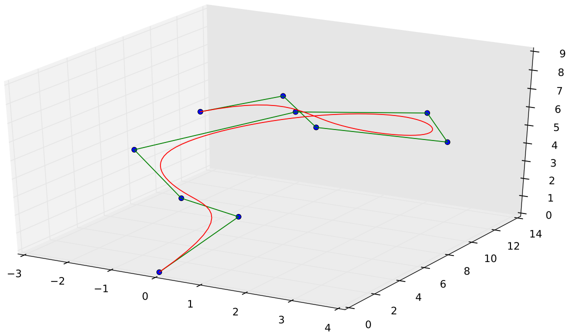
- ✔ Piecewise polynomial parametric curves $\boldsymbol{S} : [a, b] \rightarrow \mathbb{E}^3$
  $\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$
- ✔ Prescribed regularity
- ✔ Follow the shape of a control poligon
- ✔ Can interpolate the extremes of control polygon

B-spline curves



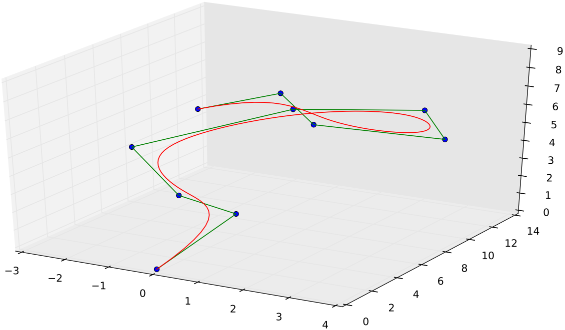
- Piecewise polynomial parametric curves $\boldsymbol{S} : [a, b] \to \mathbb{E}^3$
  $\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$
- Prescribed regularity
- Follow the shape of a control poligon
- Can interpolate the extremes of control polygon

B-spline curves



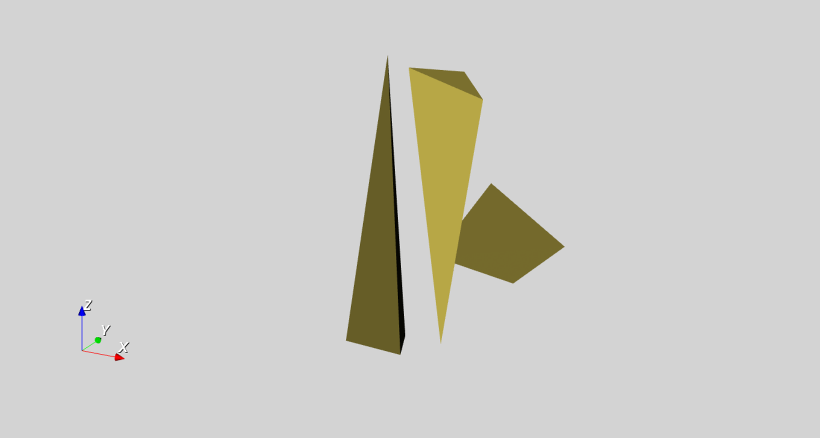
- ✔ Piecewise polynomial parametric curves $\boldsymbol{S} : [a, b] \rightarrow \mathbb{E}^3$
  $\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$
- ✔ Prescribed regularity
- ✔ Follow the shape of a control poligon
- ✔ Can interpolate the extremes of control polygon

B-spline curves

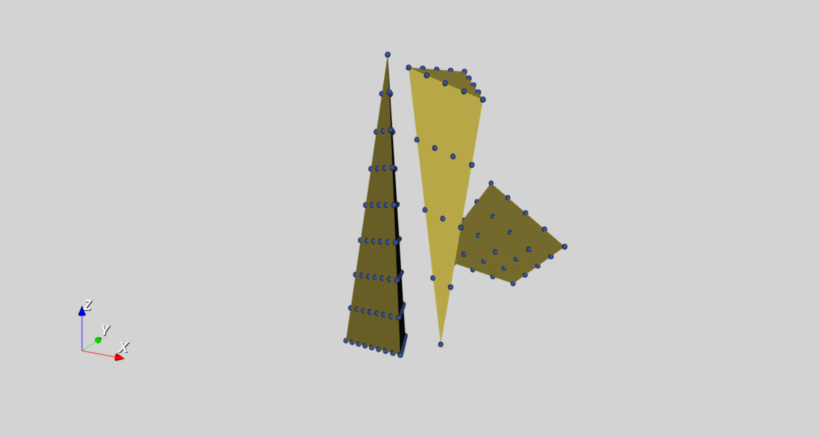


- ✔ Piecewise polynomial parametric curves $\boldsymbol{S} : [a, b] \to \mathbb{E}^3$
  $\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$
- ✔ Prescribed regularity
- ✔ Follow the shape of a control poligon
- ✔ Can interpolate the extremes of control polygon

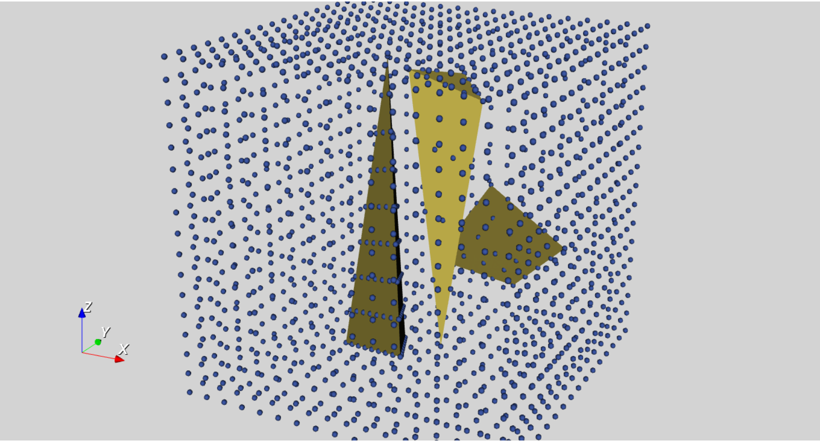
Basic structure



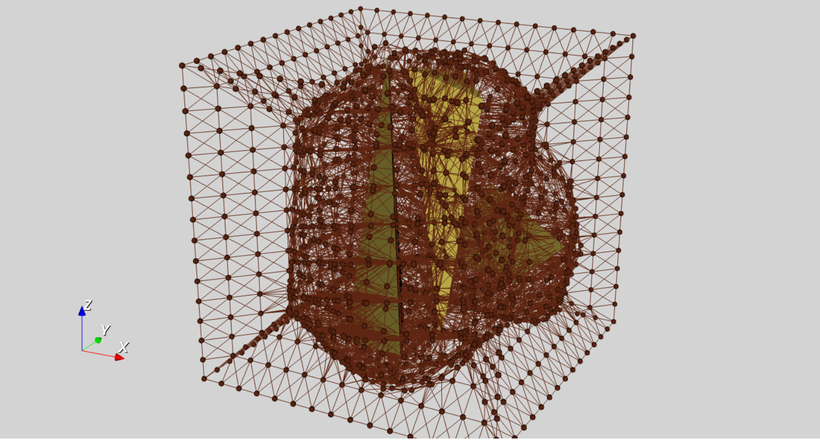
Basic structure



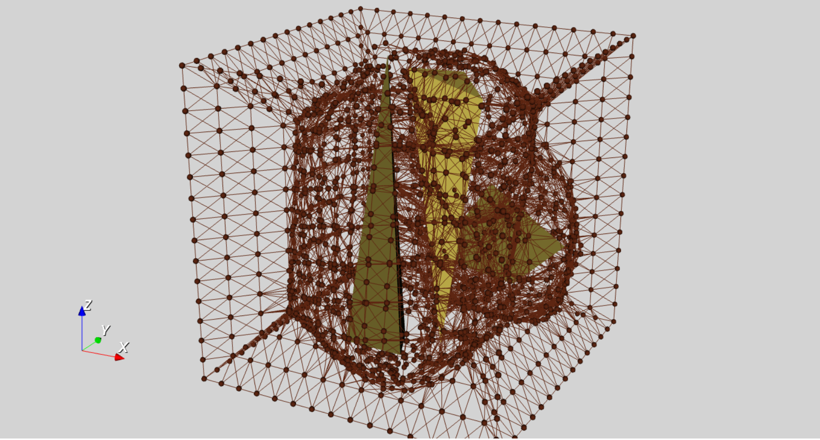
Basic structure



Basic structure



Basic structure



Improvement

Idea

Smoother curve instead of polygonal chain

- ✔ Use a B-Spline that
  - interpolate the start and end
  - Shortest path as control polygon

Improvement

Idea

Smoother curve instead of polygonal chain

- ✔ Use a B-Spline that
  - interpolate the start and end
  - Shortest path as control polygon

Improvement

Idea

Smoother curve instead of polygonal chain

- ✔ Use a B-Spline that
  - interpolate the start and end
  - Shortest path as control polygon

Improvement

Idea

Smoother curve instead of polygonal chain

- ✔ Use a B-Spline that
  - interpolate the start and end
  - Shortest path as control polygon

Problem

✔ Control polygon obstacle-free by construction

- (because is pruned of arcs that cross obstacles)

✔ Curve may intersect an obstacle

Problem

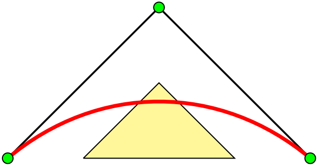
- ✔ Control polygon obstacle-free by construction
  - (because is pruned of arcs that cross obstacles)
- ✔ Curve may intersect an obstacle

Problem

- ✔ Control polygon obstacle-free by construction
  - (because is pruned of arcs that cross obstacles)
- ✔ Curve may intersect an obstacle

Problem

- ✔ Control polygon obstacle-free by construction
  - (because is pruned of arcs that cross obstacles)
- ✔ Curve may intersect an obstacle



Solution

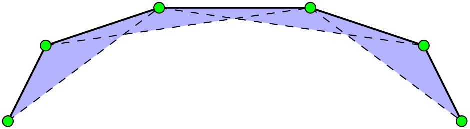
- A B-Spline of degree m is contained inside the union of convex hulls of $m + 1$ consecutive vertices

Idea

- Use a quadratic B-Spline to smooth the path
- Achieve convex hulls free from intersection with obstacles

Solution

- ✔ A B-Spline of degree m is contained inside the union of convex hulls of $m+1$ consecutive vertices

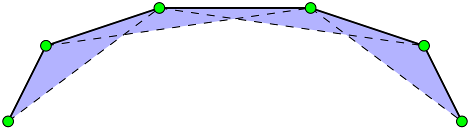


Idea

- ✔ Use a quadratic B-Spline to smooth the path
- ✔ Achieve convex hulls free from intersection with obstacles

Solution

- ✔ A B-Spline of degree m is contained inside the union of convex hulls of $m+1$ consecutive vertices

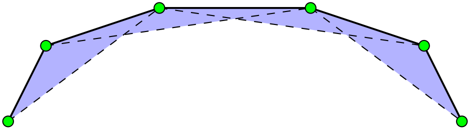


Idea

- ✔ Use a quadratic B-Spline to smooth the path
- ✔ Achieve convex hulls free from intersection with obstacles

Solution

✔ A B-Spline of degree m is contained inside the union of convex hulls of $m+1$ consecutive vertices



Idea

✔ Use a quadratic B-Spline to smooth the path

✔ Achieve convex hulls free from intersection with obstacles

First implementation

Graph transformation ($G \to G_t$)

- ✔ Triples (a, b, c) of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(a, b, c) \to (b, c, d)$
  - weighted with the distance of the edge $a \leftrightarrow b$ in G

- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples

First implementation

Graph transformation ($G \to G_t$)

- ✔ Triples $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \to (\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$
  - weighted with the distance of the edge $\boldsymbol{a} \leftrightarrow \boldsymbol{b}$ in G

- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples

First implementation

Graph transformation ($G \to G_t$)

- ✔ Triples $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \to (\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$
  - weighted with the distance of the edge $\boldsymbol{a} \leftrightarrow \boldsymbol{b}$ in G

- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples

First implementation

Graph transformation ($G \rightarrow G_t$)

- ✔ Triples (a, b, c) of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(a, b, c) \rightarrow (b, c, d)$
  - weighted with the distance of the edge $a \leftrightarrow b$ in G

- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples

First implementation

Graph transformation ($G \to G_t$)

- ✔ Triples $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \to (\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$
  - weighted with the distance of the edge $\boldsymbol{a} \leftrightarrow \boldsymbol{b}$ in G

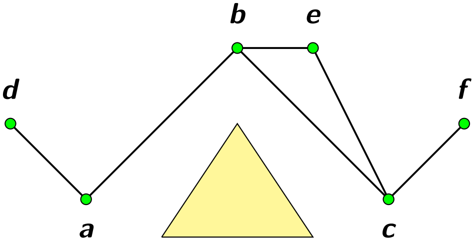
- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples

First implementation

Graph transformation ($G \to G_t$)

- ✔ Triples $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of neighboring nodes in G become nodes in G_t
- ✔ Arcs in G_t between triples in the form $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \to (\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d})$
  - weighted with the distance of the edge $\boldsymbol{a} \leftrightarrow \boldsymbol{b}$ in G

- ✔ Prune all the triples that intersect an obstacle
- ✔ Shortest path in the remaining triples



Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Creation of G_t	$\mathcal{O}(k^3\|O\|)$
Pruning of G_t	$\mathcal{O}(\|O\|^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\|O\| + k^2\|O\| \log(k^2\|O\|))$
Total	$\mathcal{O}(k^2\|O\|^2 + k^3\|O\|)$
Total (k constant)	$\mathcal{O}(\|O\|^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O|)$
  - (same of Dijkstra with constant degree)

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Creation of G_t	$\mathcal{O}(k^3\|O\|)$
Pruning of G_t	$\mathcal{O}(\|O\|^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\|O\| + k^2\|O\| \log(k^2\|O\|))$
Total	$\mathcal{O}(k^2\|O\|^2 + k^3\|O\|)$
Total (k constant)	$\mathcal{O}(\|O\|^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O|)$
  - (same of Dijkstra with constant degree)

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Creation of G_t	$\mathcal{O}(k^3\|O\|)$
Pruning of G_t	$\mathcal{O}(\|O\|^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\|O\| + k^2\|O\| \log(k^2\|O\|)$
Total	$\mathcal{O}(k^2\|O\|^2 + k^3\|O\|)$
Total (k constant)	$\mathcal{O}(\|O\|^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O|)$
  - (same of Dijkstra with constant degree)

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
Pruning of G	$\mathcal{O}(k\lvert O\rvert^2)$
Creation of G_t	$\mathcal{O}(k^3\lvert O\rvert)$
Pruning of G_t	$\mathcal{O}(\lvert O\rvert^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\lvert O\rvert + k^2\lvert O\rvert \log(k^2\lvert O\rvert))$
Total	$\mathcal{O}(k^2\lvert O\rvert^2 + k^3\lvert O\rvert)$
Total (k constant)	$\mathcal{O}(\lvert O\rvert^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ Scene construction
  - $\mathcal{O}(\lvert O\rvert^2)$
- ✔ Routing
  - $\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
  - (same of Dijkstra with constant degree)

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
Pruning of G	$\mathcal{O}(k\lvert O\rvert^2)$
Creation of G_t	$\mathcal{O}(k^3\lvert O\rvert)$
Pruning of G_t	$\mathcal{O}(\lvert O\rvert^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\lvert O\rvert + k^2\lvert O\rvert \log(k^2\lvert O\rvert))$
Total	$\mathcal{O}(k^2\lvert O\rvert^2 + k^3\lvert O\rvert)$
Total (k constant)	$\mathcal{O}(\lvert O\rvert^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ Scene construction
  - $\mathcal{O}(\lvert O\rvert^2)$
- ✔ Routing
  - $\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
  - (same of Dijkstra with constant degree)

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
Pruning of G	$\mathcal{O}(k\lvert O\rvert^2)$
Creation of G_t	$\mathcal{O}(k^3\lvert O\rvert)$
Pruning of G_t	$\mathcal{O}(\lvert O\rvert^2 k^2)$
Routing in G_t	$\mathcal{O}(k^3\lvert O\rvert + k^2\lvert O\rvert \log(k^2\lvert O\rvert))$
Total	$\mathcal{O}(k^2\lvert O\rvert^2 + k^3\lvert O\rvert)$
Total (k constant)	$\mathcal{O}(\lvert O\rvert^2)$

- O set of obstacles
- k maximum degree in G
- Scene construction
  - $\mathcal{O}(\lvert O\rvert^2)$
- Routing
  - $\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
  - (same of Dijkstra with constant degree)

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple

- ✔ new vertices easy calculated after collision check

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple

- ✔ new vertices easy calculated after collision check

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple

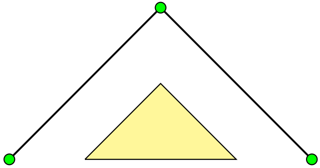
- ✔ new vertices easy calculated after collision check

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple



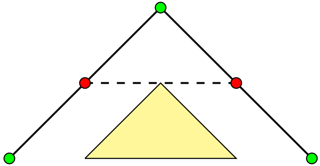
- ✔ new vertices easy calculated after collision check

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple



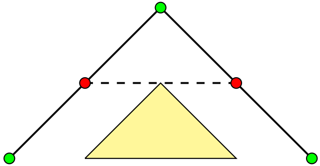
- ✔ new vertices easy calculated after collision check

Second implementation

Reason

- ✔ First implementation interesting
- ✘ but rejects many paths

- ✔ Shortest path on G
- ✔ Add aligned control vertices when an obstacle intersects a triple



- ✔ new vertices easy calculated after collision check

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
Pruning of G	$\mathcal{O}(k\lvert O\rvert^2)$
Routing in G	$\mathcal{O}(k\lvert O\rvert + \lvert O\rvert \log \lvert O\rvert)$
Path cleaning	$\mathcal{O}(\lvert P\rvert \cdot \lvert O\rvert) = \mathcal{O}(\lvert O\rvert^2)$
Total	$\mathcal{O}(k\lvert O\rvert^2)$
Total (k costant)	$\mathcal{O}(\lvert O\rvert^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ P set of control vertices
- ✔ Scene construction
  - $\mathcal{O}(\lvert O\rvert^2)$
- ✔ Routing
  - $\mathcal{O}(\lvert O\rvert \log \lvert O\rvert + \lvert P\rvert\ \lvert O\rvert)$

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Routing in G	$\mathcal{O}(k\|O\| + \|O\| \log \|O\|)$
Path cleaning	$\mathcal{O}(\|P\| \cdot \|O\|) = \mathcal{O}(\|O\|^2)$
Total	$\mathcal{O}(k\|O\|^2)$
Total (k costant)	$\mathcal{O}(\|O\|^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ P set of control vertices
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O| + |P| \; |O|)$

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\lvert O\rvert \log \lvert O\rvert)$
Pruning of G	$\mathcal{O}(k\lvert O\rvert^2)$
Routing in G	$\mathcal{O}(k\lvert O\rvert + \lvert O\rvert \log \lvert O\rvert)$
Path cleaning	$\mathcal{O}(\lvert P\rvert \cdot \lvert O\rvert) = \mathcal{O}(\lvert O\rvert^2)$
Total	$\mathcal{O}(k\lvert O\rvert^2)$
Total (k costant)	$\mathcal{O}(\lvert O\rvert^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ P set of control vertices
- ✔ Scene construction
  - $\mathcal{O}(\lvert O\rvert^2)$
- ✔ Routing
  - $\mathcal{O}(\lvert O\rvert \log \lvert O\rvert + \lvert P\rvert\ \lvert O\rvert)$

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\vert O\vert \log \vert O\vert)$
Pruning of G	$\mathcal{O}(k\vert O\vert^2)$
Routing in G	$\mathcal{O}(k\vert O\vert + \vert O\vert \log \vert O\vert)$
Path cleaning	$\mathcal{O}(\vert P\vert \cdot \vert O\vert) = \mathcal{O}(\vert O\vert^2)$
Total	$\mathcal{O}(k\vert O\vert^2)$
Total (k costant)	$\mathcal{O}(\vert O\vert^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ P set of control vertices
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O| + |P| \; |O|)$

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Routing in G	$\mathcal{O}(k\|O\| + \|O\| \log \|O\|)$
Path cleaning	$\mathcal{O}(\|P\| \cdot \|O\|) = \mathcal{O}(\|O\|^2)$
Total	$\mathcal{O}(k\|O\|^2)$
Total (k costant)	$\mathcal{O}(\|O\|^2)$

- O set of obstacles
- k maximum degree in G
- P set of control vertices
- Scene construction
  - $\mathcal{O}(|O|^2)$
- Routing
  - $\mathcal{O}(|O| \log |O| + |P| \, |O|)$

Complexity

Description	Cost
Creation of G	$\mathcal{O}(\|O\| \log \|O\|)$
Pruning of G	$\mathcal{O}(k\|O\|^2)$
Routing in G	$\mathcal{O}(k\|O\| + \|O\| \log \|O\|)$
Path cleaning	$\mathcal{O}(\|P\| \cdot \|O\|) = \mathcal{O}(\|O\|^2)$
Total	$\mathcal{O}(k\|O\|^2)$
Total (k costant)	$\mathcal{O}(\|O\|^2)$

- ✔ O set of obstacles
- ✔ k maximum degree in G
- ✔ P set of control vertices
- ✔ Scene construction
  - $\mathcal{O}(|O|^2)$
- ✔ Routing
  - $\mathcal{O}(|O| \log |O| + |P| \; |O|)$

Increase degree

Continuity

✔ Using quadratic B-Splines means C^1 continuity

✘ Not enough

✘ If we increase the B-Spline degree → convex hull not planar anymore

- convex hull formed of union of tetrahedra

Solution

✔ Add aligned vertices in control polygon

- then increase the degree

Increase degree

Continuity

✔ Using quadratic B-Splines means C^1 continuity

✘ Not enough

✘ If we increase the B-Spline degree $\rightarrow$ convex hull not planar anymore

- convex hull formed of union of tetrahedra

Solution

✔ Add aligned vertices in control polygon

- then increase the degree

Increase degree

Continuity

- ✔ Using quadratic B-Splines means C^1 continuity
- ✘ Not enough

- ✘ If we increase the B-Spline degree $\rightarrow$ convex hull not planar anymore
  - convex hull formed of union of tetrahedra

Solution

- ✔ Add aligned vertices in control polygon
  - then increase the degree

Increase degree

Continuity

- ✔ Using quadratic B-Splines means C^1 continuity
- ✘ Not enough

- ✘ If we increase the B-Spline degree $\rightarrow$ convex hull not planar anymore
  - convex hull formed of union of tetrahedra

Solution

- ✔ Add aligned vertices in control polygon
  - then increase the degree

Increase degree

Continuity

- ✔ Using quadratic B-Splines means C^1 continuity
- ✘ Not enough

✘ If we increase the B-Spline degree $\rightarrow$ convex hull not planar anymore
- convex hull formed of union of tetrahedra

Solution

- ✔ Add aligned vertices in control polygon
  - then increase the degree

Increase degree

Continuity

✔ Using quadratic B-Splines means C^1 continuity

✘ Not enough

✘ If we increase the B-Spline degree → convex hull not planar anymore

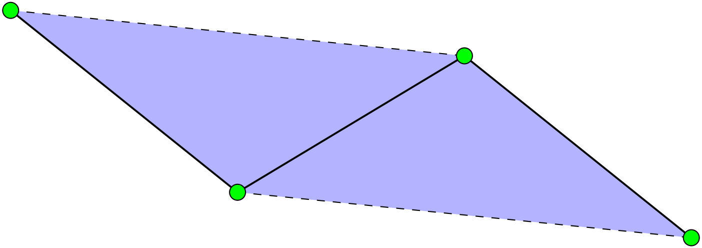
- convex hull formed of union of tetrahedra

Solution

✔ Add aligned vertices in control polygon

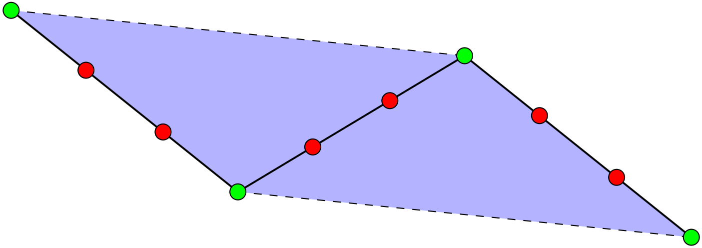
- then increase the degree

Example: quadratic to quartic (m=2 $\rightarrow$ m=4)



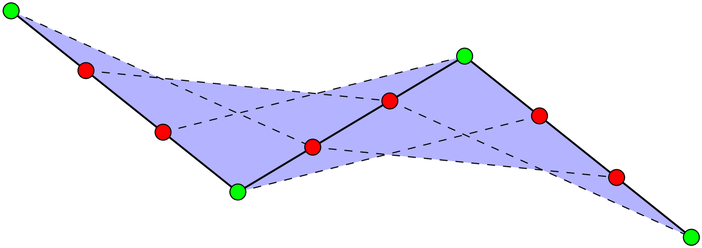
✔ Add 2 vertices per edge

Example: quadratic to quartic (m=2 $\rightarrow$ m=4)



✔ Add 2 vertices per edge

Example: quadratic to quartic (m=2 $\rightarrow$ m=4)



✔ Add 2 vertices per edge

Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of consecutive points in path
- ✔ If no obstacles intersect the triangle $\rightarrow$ the triple is simplified to a single edge $(\boldsymbol{a}, \boldsymbol{c})$

☛ After simplification, new neighbouring triples need to be obstacle-free

Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple (a, b, c) of consecutive points in path
- ✔ If no obstacles intersect the triangle → the triple is simplified to a single edge (a, c)

☛ After simplification, new neighbouring triples need to be obstacle-free

Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple (a, b, c) of consecutive points in path
- ✔ If no obstacles intersect the triangle $\rightarrow$ the triple is simplified to a single edge (a, c)

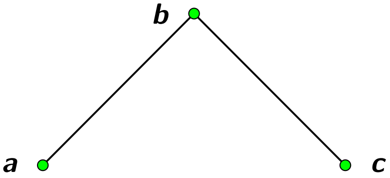
☛ After simplification, new neighbouring triples need to be obstacle-free

Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of consecutive points in path
- ✔ If no obstacles intersect the triangle $\rightarrow$ the triple is simplified to a single edge $(\boldsymbol{a}, \boldsymbol{c})$
- ☛ After simplification, new neighbouring triples need to be obstacle-free

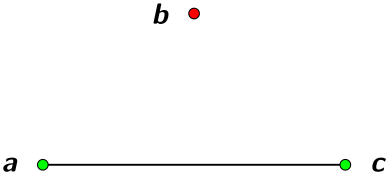


Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple (a, b, c) of consecutive points in path
- ✔ If no obstacles intersect the triangle $\rightarrow$ the triple is simplified to a single edge (a, c)
- ☛ After simplification, new neighbouring triples need to be obstacle-free

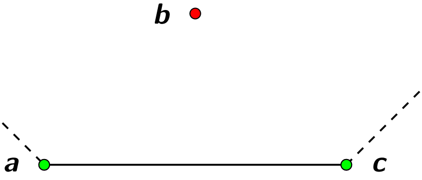


Post Processing

Purpose

- ✔ Simplify the control polygon
- ✔ Remove useless turns

- ✔ For each triple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ of consecutive points in path
- ✔ If no obstacles intersect the triangle $\rightarrow$ the triple is simplified to a single edge $(\boldsymbol{a}, \boldsymbol{c})$
- ☛ After simplification, new neighbouring triples need to be obstacle-free



Optimization method

Problem

$$\underset{P}{\text{minimize}} \quad \alpha \cdot \max_u[\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u[\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S})$$

$$\text{subject to} \quad \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Optimization method

Problem

$$\begin{aligned} \underset{P}{\text{minimize}} \quad & \alpha \cdot \max_u [\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u [\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S}) \\ \text{subject to} \quad & \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0 \end{aligned}$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Optimization method

Problem

$$\underset{P}{\text{minimize}} \quad \alpha \cdot \max_u[\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u[\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S})$$

$$\text{subject to} \quad \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Optimization method

Problem

$$\underset{P}{\text{minimize}} \quad \alpha \cdot \max_u[\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u[\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S})$$

$$\text{subject to} \quad \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Optimization method

Problem

$$\underset{P}{\text{minimize}} \quad \alpha \cdot \max_u[\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u[\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S})$$

$$\text{subject to} \quad \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Optimization method

Problem

$$\underset{P}{\text{minimize}} \quad \alpha \cdot \max_u[\kappa_{\boldsymbol{S}}(u)] + \beta \cdot \max_u[\tau_{\boldsymbol{S}}(u)] + \gamma \cdot len(\boldsymbol{S})$$

$$\text{subject to} \quad \left| \boldsymbol{S}(u) \cap \bigcup_{i \in I} obstacle_i \right| = 0$$

- ✔ Relax constraint: $L(P, \lambda) = gain(P) + \lambda \cdot constraint(P)$
- ✔ Saddle point $L(P^*, \lambda) \leq L(P^*, \lambda^*) \leq L(P, \lambda^*)$
- ✔ Simulated annealing finds saddle point that minimizes *gain*

Drawbacks

- ✘ slower respect to the other methods
- ✘ *gain* and *constraint* are calculated in a discrete way

Technologies

Technologies

python™ 3

SciPy

Technologies

python™ 3

SciPy

NetworkX

Technologies

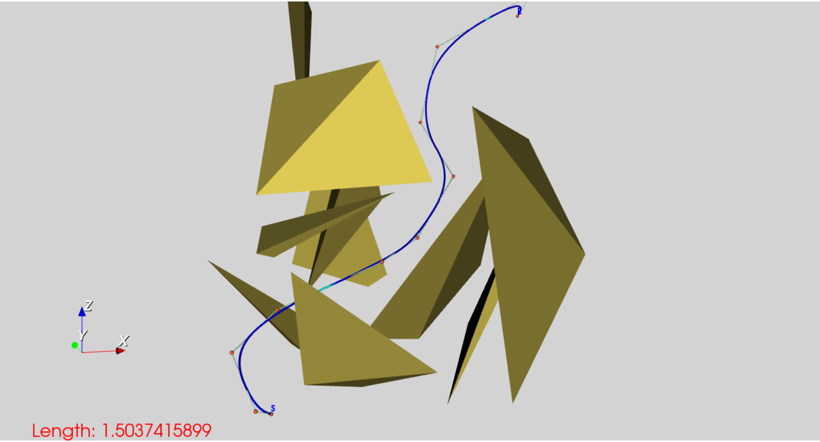
python 3

SciPy

NetworkX

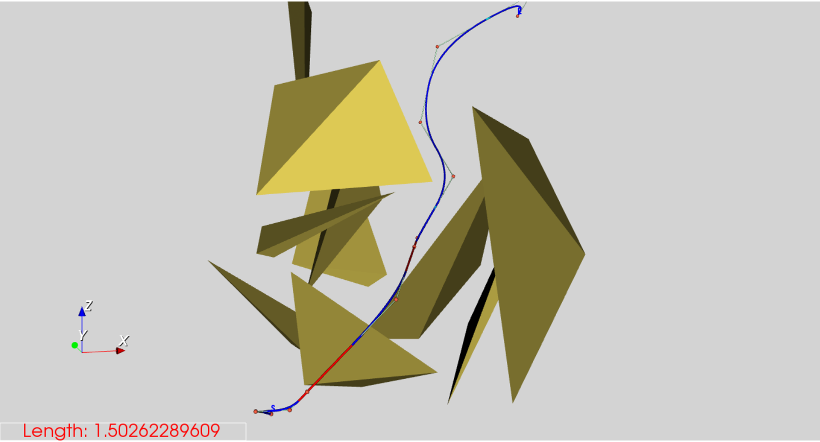
Visualization Toolkit

Example



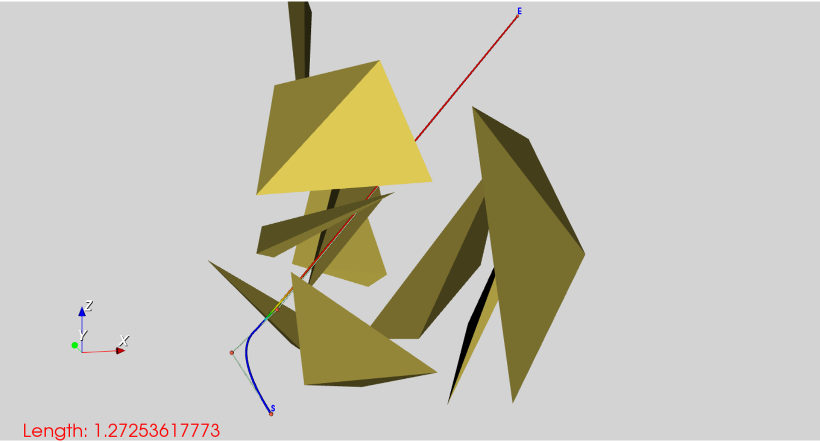
✔ Method 1, no post processing

Example



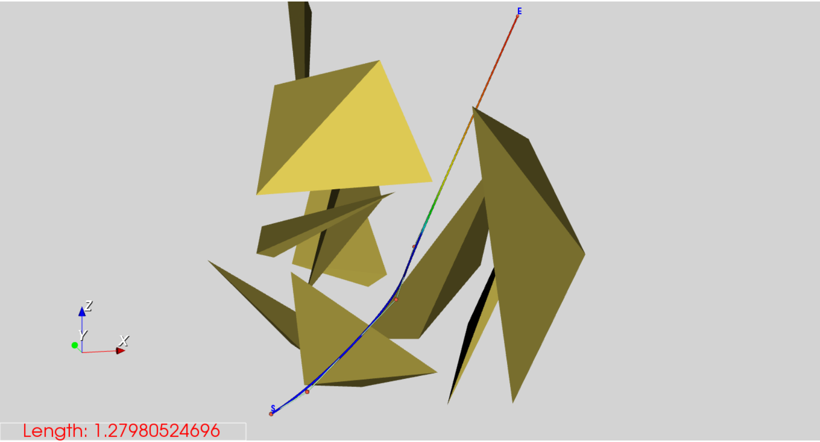
✔ Method 2, no post processing

Example



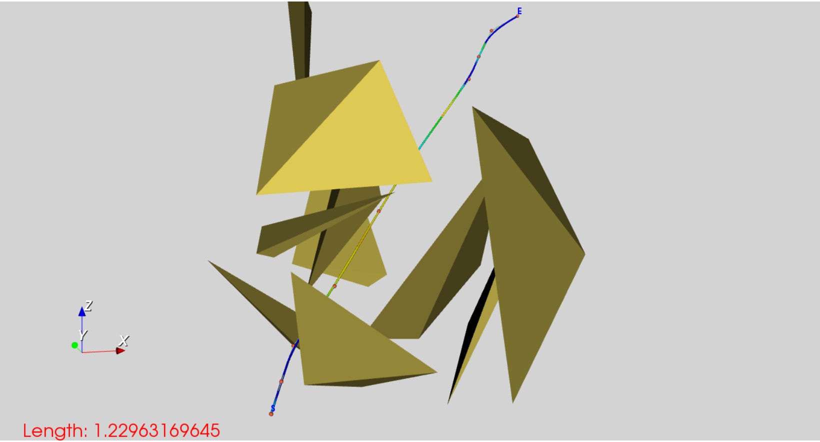
✔ Method 1, with post processing

Example



✔ Method 2, with post processing

Example



✔ Method 3

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other . . .
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other . . .
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other . . .
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other . . .
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

Future improvements

- ✔ Change underlying structure
  - different attach point
  - visibility graph
  - rapidly exploring random tree (RRT)
  - other ...
- ✔ Improve degree increase
  - without aligned vertices
  - like second solution but with quadruple/quintuples of vertices
- ✔ Improve post processing
  - make a symmetric algorithm
- ✔ Another optimization process
  - output of other solutions as initial state
  - moves in a restricted space

The End.

Questions? Thank you!

The End.

Questions? Thank you!

The End.

Questions? Thank you!

B-spline curves details

- ✔ Degree m
- ✔ Extended partition (of parametric space $[a, b]$)

$$T = \{t_0, \ldots, t_{m-1}, t_m, \ldots, t_{n+1}, t_{n+2}, \ldots, t_{n+m+1}\}$$

$$t_0 \leq \cdots \leq t_{m-1} \leq t_m (\equiv a) < \cdots < t_{n+1} (\equiv b) \leq t_{n+2} \leq \cdots \leq t_{n+m+1}$$

- ✔ $n + 1$ basis (of $S_{m,\tau} = P_{m,\tau} \cap C^{m-1}$)

$$N_{i,1}(u) = \begin{cases} 1, & \text{if} \quad t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \qquad i = 0, \ldots, n+m$$

$$N_{i,r}(u) = \omega_{i,r-1}(u) \cdot N_{i,r-1}(u) \; + \; (1 - \omega_{i+1,r-1}(u)) \cdot N_{i+1,r-1}(u)$$

i=0,...,n+m+1-3, r=2,...,m+1

$$\omega_{i,r}(u) = \begin{cases} \frac{t - t_i}{t_{i+r} - t_i}, & \text{if } t_i \neq t_{i+r} \\ 0, & \text{otherwise} \end{cases}$$

- ✔ B-spline curve $\boldsymbol{S} : [a, b] \subset \mathbb{R} \rightarrow \mathbb{E}^d$

$$\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$$

B-spline curves details

- ✔ Degree m
- ✔ Extended partition (of parametric space $[a, b]$)

$$T = \{t_0, \dots, t_{m-1}, t_m, \dots, t_{n+1}, t_{n+2}, \dots, t_{n+m+1}\}$$

$$t_0 \le \dots \le t_{m-1} \le t_m (\equiv a) < \dots < t_{n+1} (\equiv b) \le t_{n+2} \le \dots \le t_{n+m+1}$$

- ✔ $n+1$ basis (of $S_{m,\tau} = P_{m,\tau} \cap C^{m-1}$)

$$N_{i,1}(u) = \begin{cases} 1, & \text{if} \quad t_i \le t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \qquad i = 0, \dots, n+m$$

$$N_{i,r}(u) = \omega_{i,r-1}(u) \cdot N_{i,r-1}(u) + (1 - \omega_{i+1,r-1}(u)) \cdot N_{i+1,r-1}(u)$$

i=0,...,n+m+1-3, r=2,...,m+1

$$\omega_{i,r}(u) = \begin{cases} \frac{t-t_i}{t_{i+r}-t_i}, & \text{if } t_i \ne t_{i+r} \\ 0, & \text{otherwise} \end{cases}$$

- ✔ B-spline curve $\boldsymbol{S} : [a, b] \subset \mathbb{R} \to \mathbb{E}^d$

$$\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$$

B-spline curves details

- ✔ Degree m
- ✔ Extended partition (of parametric space $[a, b]$)

$$T = \{t_0, \dots, t_{m-1}, t_m, \dots, t_{n+1}, t_{n+2}, \dots, t_{n+m+1}\}$$

$$t_0 \leq \dots \leq t_{m-1} \leq t_m (\equiv a) < \dots < t_{n+1} (\equiv b) \leq t_{n+2} \leq \dots \leq t_{n+m+1}$$

- ✔ $n+1$ basis (of $S_{m,\tau} = P_{m,\tau} \cap C^{m-1}$)

$$N_{i,1}(u) = \begin{cases} 1, & \text{if} \quad t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \qquad i = 0, \dots, n+m$$

$$N_{i,r}(u) = \omega_{i,r-1}(u) \cdot N_{i,r-1}(u) \; + \; (1 - \omega_{i+1,r-1}(u)) \cdot N_{i+1,r-1}(u)$$

i=0,...,n+m+1-3, r=2,...,m+1

$$\omega_{i,r}(u) = \begin{cases} \frac{t-t_i}{t_{i+r}-t_i}, & \text{if } t_i \neq t_{i+r} \\ 0, & \text{otherwise} \end{cases}$$

- ✔ B-spline curve $\boldsymbol{S} : [a, b] \subset \mathbb{R} \rightarrow \mathbb{E}^d$

$$\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$$

B-spline curves details

- ✔ Degree m
- ✔ Extended partition (of parametric space $[a, b]$)

$$T = \{t_0, \ldots, t_{m-1}, t_m, \ldots, t_{n+1}, t_{n+2}, \ldots, t_{n+m+1}\}$$

$$t_0 \leq \cdots \leq t_{m-1} \leq t_m (\equiv a) < \cdots < t_{n+1} (\equiv b) \leq t_{n+2} \leq \cdots \leq t_{n+m+1}$$

- ✔ $n+1$ basis (of $S_{m,\tau} = P_{m,\tau} \cap C^{m-1}$)

$$N_{i,1}(u) = \begin{cases} 1, & \text{if} \quad t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \qquad i = 0, \ldots, n+m$$

$$N_{i,r}(u) = \omega_{i,r-1}(u) \cdot N_{i,r-1}(u) + (1 - \omega_{i+1,r-1}(u)) \cdot N_{i+1,r-1}(u)$$

i=0,. . . ,n+m+1-3, r=2,. . . ,m+1

$$\omega_{i,r}(u) = \begin{cases} \frac{t-t_i}{t_{i+r}-t_i}, & \text{if } t_i \neq t_{i+r} \\ 0, & \text{otherwise} \end{cases}$$

- ✔ B-spline curve $\boldsymbol{S} : [a, b] \subset \mathbb{R} \to \mathbb{E}^d$

$$\mathbf{S}(u) = \sum_{i=0}^{n} \mathbf{v_i} \cdot N_{i,m+1}(u)$$

Useful properties of B-spline curves

- ✔ Interpolates extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- ✔ Continuity C^{m-1} between polynomials
- ✔ Contained in convex hulls of $m+1$ consecutive vertices

- ✔ Touches segment between m aligned vertices
- ✔ Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}{\left\|\dot{S}(u)\right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{S}(u), \ddot{S}(u), \dddot{S}(u)\right]}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2} = \frac{\left(\dot{S}(u) \wedge \ddot{S}(u)\right) \cdot \dddot{S}(u)}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}$$

Useful properties of B-spline curves

- **Interpolates** extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- **Continuity** C^{m-1} between polynomials
- Contained in convex hulls of $m+1$ consecutive vertices

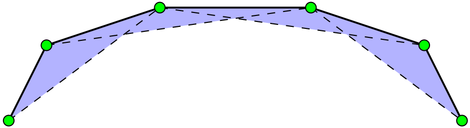
- Touches segment between m aligned vertices
- Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}{\left\|\dot{S}(u)\right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{S}(u), \ddot{S}(u), \dddot{S}(u)\right]}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2} = \frac{\left(\dot{S}(u) \wedge \ddot{S}(u)\right) \cdot \dddot{S}(u)}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}$$

Useful properties of B-spline curves

- ✔ Interpolates extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- ✔ Continuity C^{m-1} between polynomials
- ✔ Contained in convex hulls of $m+1$ consecutive vertices



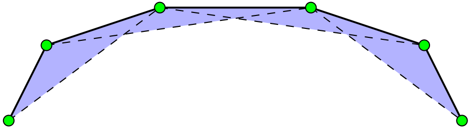
- ✔ Touches segment between m aligned vertices
- ✔ Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}{\left\|\dot{S}(u)\right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{S}(u), \ddot{S}(u), \dddot{S}(u)\right]}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2} = \frac{\left(\dot{S}(u) \wedge \ddot{S}(u)\right) \cdot \dddot{S}(u)}{\left\|\dot{S}(u) \wedge \ddot{S}(u)\right\|_2}$$

Useful properties of B-spline curves

- ✔ Interpolates extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- ✔ Continuity C^{m-1} between polynomials
- ✔ Contained in convex hulls of $m+1$ consecutive vertices



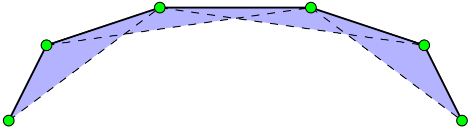
- ✔ Touches segment between m aligned vertices
- ✔ Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\| \dot{\mathbf{S}}(u) \wedge \ddot{\mathbf{S}}(u) \right\|_2}{\left\| \dot{\mathbf{S}}(u) \right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{\mathbf{S}}(u), \ddot{\mathbf{S}}(u), \dddot{\mathbf{S}}(u) \right]}{\left\| \dot{\mathbf{S}}(u) \wedge \ddot{\mathbf{S}}(u) \right\|_2} = \frac{\left(\dot{\mathbf{S}}(u) \wedge \ddot{\mathbf{S}}(u) \right) \cdot \dddot{\mathbf{S}}(u)}{\left\| \dot{\mathbf{S}}(u) \wedge \ddot{\mathbf{S}}(u) \right\|_2}$$

Useful properties of B-spline curves

- ✔ Interpolates extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- ✔ Continuity C^{m-1} between polynomials
- ✔ Contained in convex hulls of $m+1$ consecutive vertices



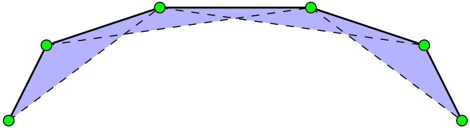
- ✔ Touches segment between m aligned vertices
- ✔ Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\| \dot{S}(u) \wedge \ddot{S}(u) \right\|_2}{\left\| \dot{S}(u) \right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{S}(u), \ddot{S}(u), \dddot{S}(u)\right]}{\left\| \dot{S}(u) \wedge \ddot{S}(u) \right\|_2} = \frac{\left(\dot{S}(u) \wedge \ddot{S}(u)\right) \cdot \dddot{S}(u)}{\left\| \dot{S}(u) \wedge \ddot{S}(u) \right\|_2}$$

Useful properties of B-spline curves

- ✔ Interpolates extremes if $t_0 = \cdots = t_m$ and $t_{n+1} = \cdots = t_{n+m+1}$
- ✔ Continuity C^{m-1} between polynomials
- ✔ Contained in convex hulls of $m+1$ consecutive vertices



- ✔ Touches segment between m aligned vertices
- ✔ Lays in segment between $m+1$ aligned vertices

$$\kappa(u) = \frac{\left\|\dot{\boldsymbol{S}}(u) \wedge \ddot{\boldsymbol{S}}(u)\right\|_2}{\left\|\dot{\boldsymbol{S}}(u)\right\|_2^3}$$

$$\tau(u) = \frac{\det\left[\dot{\boldsymbol{S}}(u), \ddot{\boldsymbol{S}}(u), \dddot{\boldsymbol{S}}(u)\right]}{\left\|\dot{\boldsymbol{S}}(u) \wedge \ddot{\boldsymbol{S}}(u)\right\|_2} = \frac{\left(\dot{\boldsymbol{S}}(u) \wedge \ddot{\boldsymbol{S}}(u)\right) \cdot \dddot{\boldsymbol{S}}(u)}{\left\|\dot{\boldsymbol{S}}(u) \wedge \ddot{\boldsymbol{S}}(u)\right\|_2}$$

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices $\rightarrow$ nodes
   - cells edges $\rightarrow$ arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Background

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices → nodes
   - cells edges → arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Background

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices → nodes
   - cells edges → arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Background

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices $\rightarrow$ nodes
   - cells edges $\rightarrow$ arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Background

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices → nodes
   - cells edges → arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Background

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

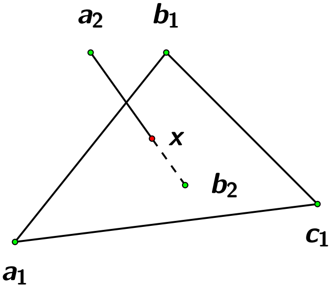
1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices → nodes
   - cells edges → arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Main problem

Path planning from a start point to an end point in 3D space with obstacles using Voronoi diagrams.

1. Distribute points in obstacles surfaces
   - and bounding box
2. Voronoi diagram using those points
3. Transform Voronoi diagram in graph
   - cells vertices → nodes
   - cells edges → arcs (infinite edges ignored)
4. Prune arcs that cross obstacles
5. Attach start and end
6. Shortest path from start to end
   - Dijkstra's algorithm

Intersection segment-triangle



$$\begin{cases} \alpha \boldsymbol{a_2} + \beta \boldsymbol{b_2} = \gamma \boldsymbol{a_1} + \delta \boldsymbol{b_1} + \zeta \boldsymbol{c_1} \\ \alpha + \beta = 1 \\ \gamma + \delta + \zeta = 1 \end{cases}$$

$$\begin{cases} \alpha \geq 0 \\ \beta \geq 0 \\ \gamma \geq 0 \\ \delta \geq 0 \\ \zeta \geq 0. \end{cases}$$