Algorithmique de la planification de mouvement

Mattéo Clémot

Juin 2019

Introduction

- Déterminer / approximer des trajectoires pour un système cinématique en respectant des contraintes de non-collision avec un environnement.
- Applications : robotique, désassemblage industriel...

Problème

- Donnée d'un espace où se trouvent les positions du système cinématique.
- Donnée de l'espace des configurations : positions respectant les contraintes de non-collision avec l'environnement.

Problème

- Donnée d'un espace où se trouvent les positions du système cinématique.
- Donnée de l'espace des configurations : positions respectant les contraintes de non-collision avec l'environnement.

Systèmes cinématiques

- Bras robotique dans le plan à d segments, avec une origine éventuellement translatable. Obstacle représenté par un ensemble de segments.
- Condition d'appartenance : non-collision avec l'environnement et non-autocollision, par tests d'intersection de segments (système de Cramer).
- Robot polygonal, obstacles représentés par un ensemble de points.
- Condition d'appartenance : non-appartenance des points au polygone vu comme union de triangles (coordonnées barycentriques).

Systèmes cinématiques

- Bras robotique dans le plan à d segments, avec une origine éventuellement translatable. Obstacle représenté par un ensemble de segments.
- Condition d'appartenance : non-collision avec l'environnement et non-autocollision, par tests d'intersection de segments (système de Cramer).
- Robot polygonal, obstacles représentés par un ensemble de points.
- Condition d'appartenance : non-appartenance des points au polygone vu comme union de triangles (coordonnées barycentriques).

Méthode générale

- Construction d'un graphe :
  - les nœuds sont des positions valides
  - une arête représente une transition valide du système entre les deux positions
- Recherche d'un chemin dans ce graphe entre le départ et le but :
  - algorithme de Dijkstra
  - algorithme A* (heuristique : distance physique entre nœuds)

Méthode générale

- Construction d'un graphe :
  - les nœuds sont des positions valides
  - une arête représente une transition valide du système entre les deux positions
- Recherche d'un chemin dans ce graphe entre le départ et le but :
  - algorithme de Dijkstra
  - algorithme A* (heuristique : distance physique entre nœuds)

Discrétisation naïve

- "Grille" de même dimension que l'espace, les voisins sont les cellules adjacentes.

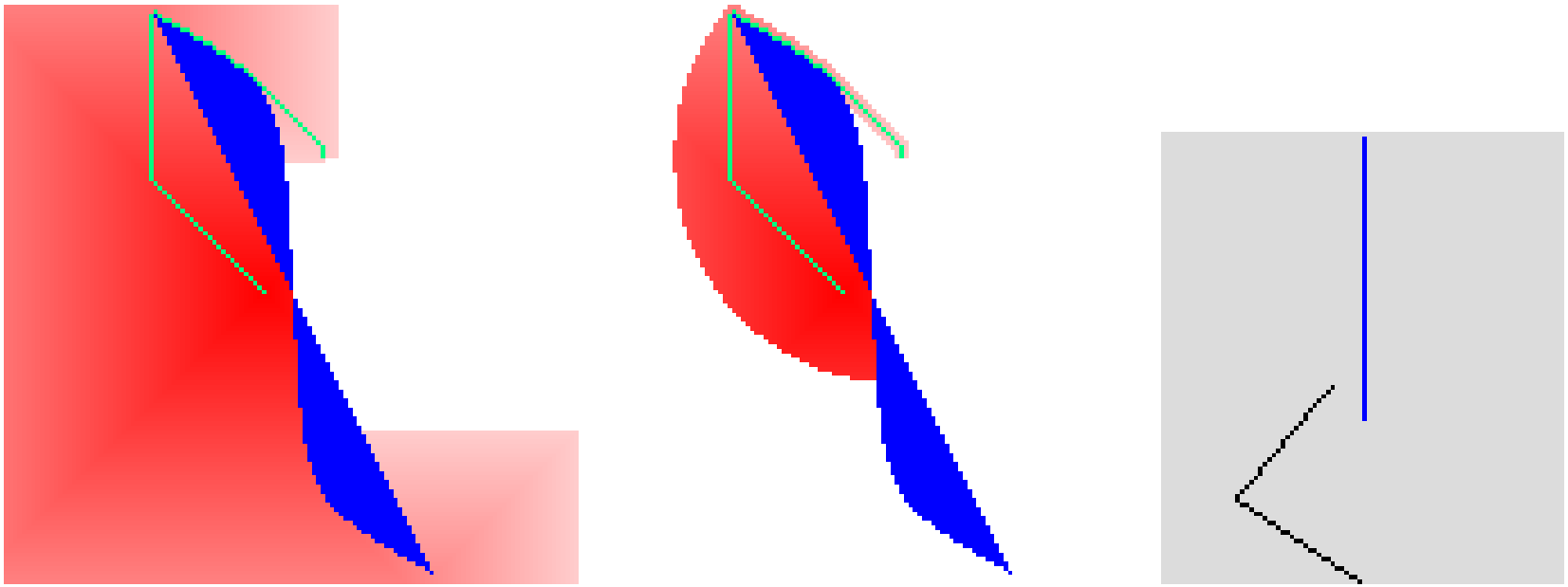


Figure – Recherche par algorithmes de Dijkstra et A* pour une discrétisation naïve ($d = 2$) des configurations d'un bras robotique.

Partition de l'espace

- Partition récursive de l'espace en utilisant un arbre dont les nœuds, représentant des boîtes dans l'espace, sont d'arité 0 ou 2^d.
- Découpage d'un nœud lorsque les sommets sont ni tous valides ni tous invalides.
- Arrêt en fixant une hauteur maximale ou une taille de boîte minimale.

Partition de l'espace

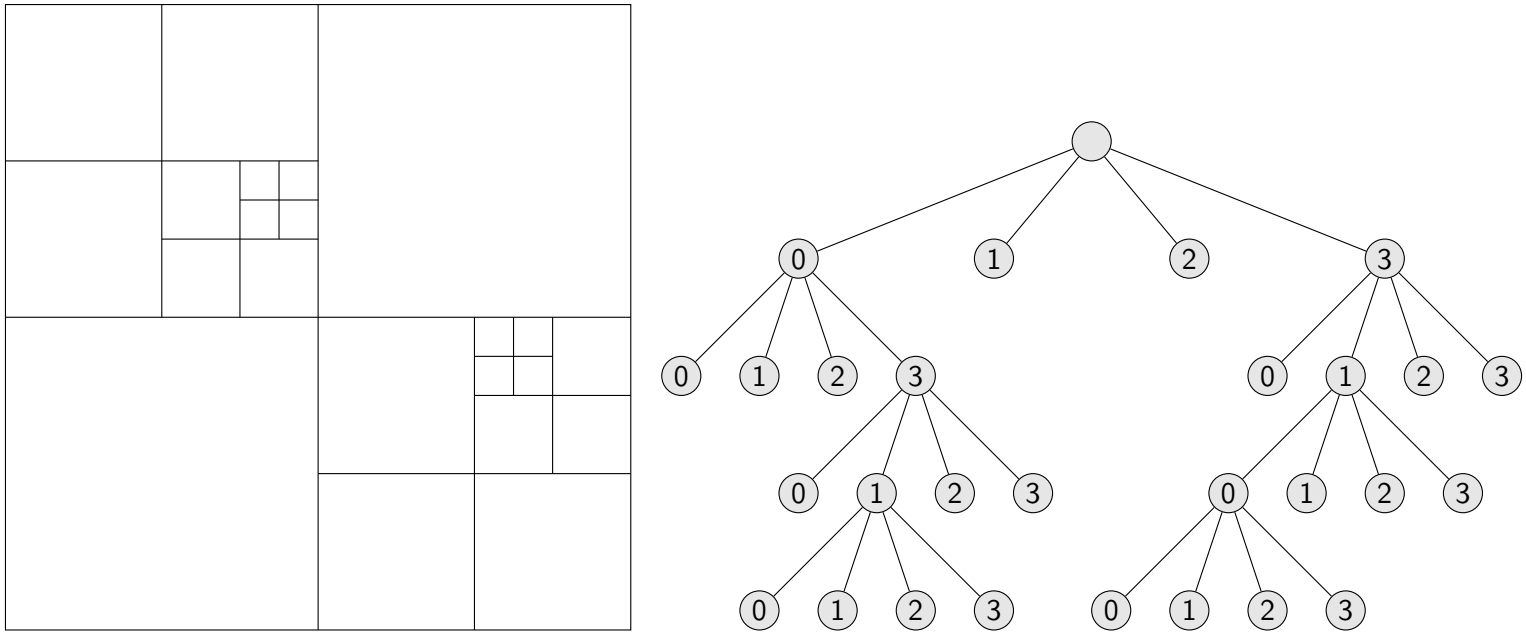


Figure – Partition de l'espace ($d = 2$) et *quadtree* associé

Taille de l'arbre

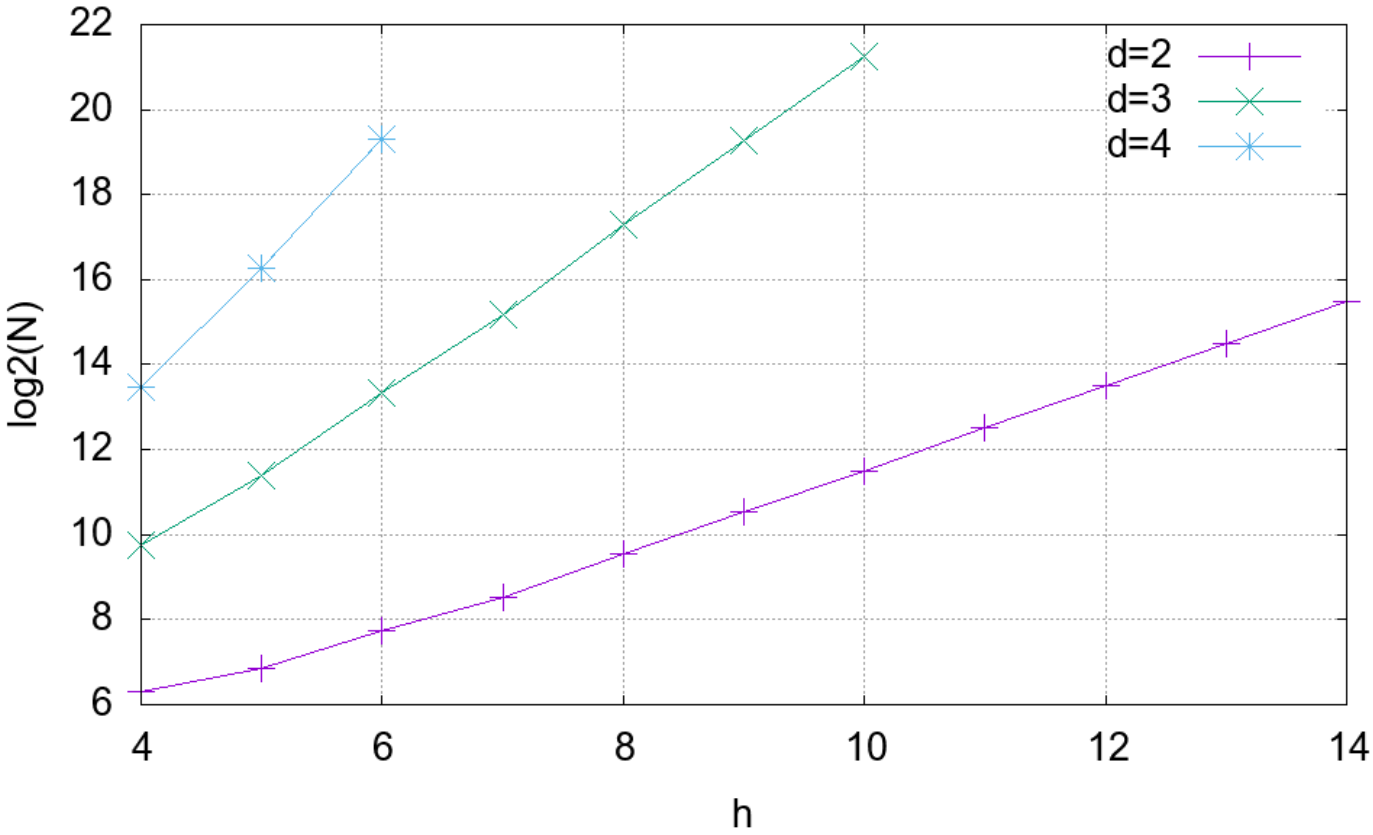


Figure – Mesure de la taille de l'arbre en fonction de la hauteur fixée et de la dimension.

Taille de l'arbre

- Comportement en $2^{(d-1)h}$, au lieu de 2^{dh} pour la discrétisation naïve.
- Les nœuds se concentrent à la frontière de l'espace de configuration, hypersurface de dimension $d-1$.

Parcours du graphe

- Graphe dont les nœuds sont les feuilles de l'arbre, deux nœuds étant reliés si les boîtes associées ont une frontière en commun.

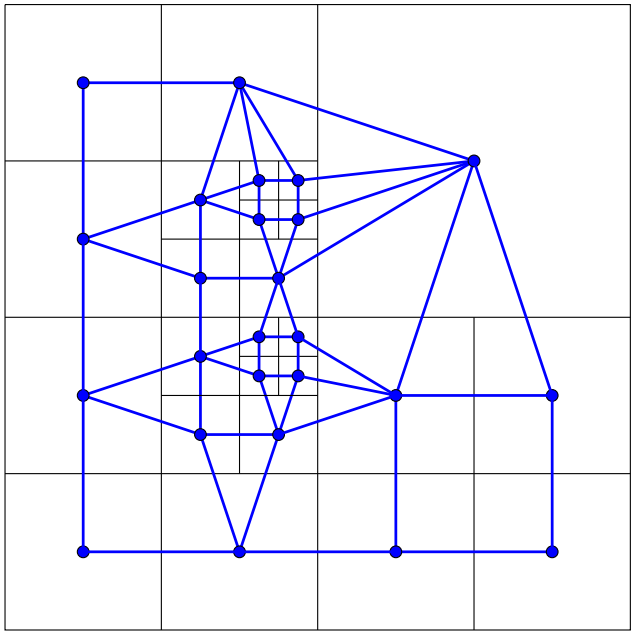
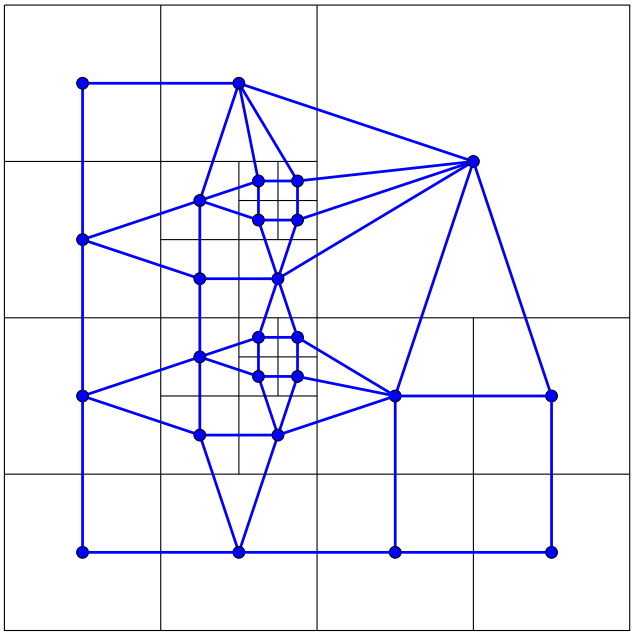


Figure – Graphe associé à une discrétisation de l'espace ($d = 2$).

- Pas de stockage du graphe, voisins déterminés lors de son exploration.

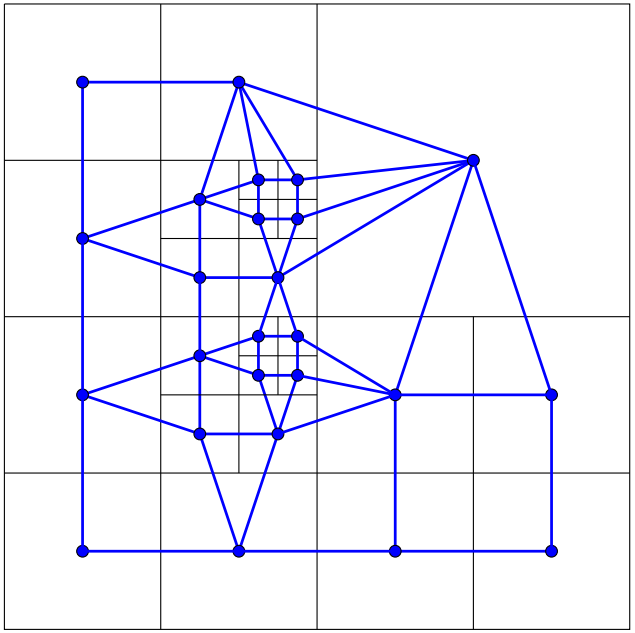
Détermination des voisins

- Codage de la position relative d'un enfant par un entier de $[\![0, 2^d - 1]\!]$ ou par son écriture binaire dans $\{0, 1\}^d$.
- Dans une direction $i \in [\![0, d-1]\!]$, procédure de descente récursive, en appelant la descente sur les 2^{d-1} enfants de même bit i que le nœud de départ, et en partant du nœud frère obtenu en inversant le bit i du nœud de départ.



Détermination des voisins

- Voisins indirects : remontée de l'arbre en mémorisant la suite de position relatives dans une pile jusqu'à disposer d'un nœud frère dans l'autre sens de la direction i, puis redescente en dépilant puis inversant le bit i.
- De nouveau descente récursive pour finir.



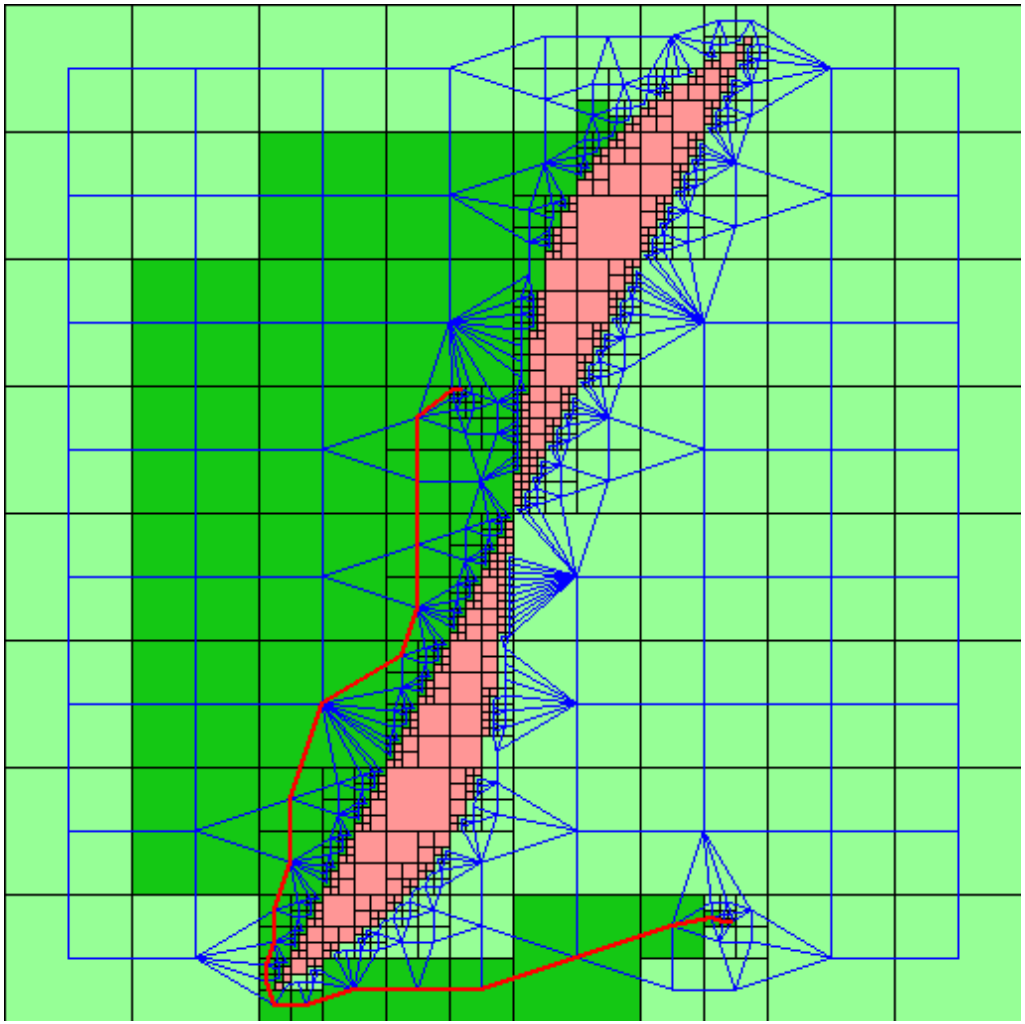
Détermination des voisins

- Coût de la détermination des voisins d'un nœud à profondeur k, $\mathcal{O}(d(k + 2^{(d-1)(h-k+1)}))$.
- Coût de l'exploration à profondeur k ($2^{(d-1)k}$ nœuds) : $\mathcal{O}(dk2^{(d-1)k} + d2^{(d-1)(h+1)})$.
- Au final exploration en $\mathcal{O}(dh2^{(d-1)(h+1)})$.

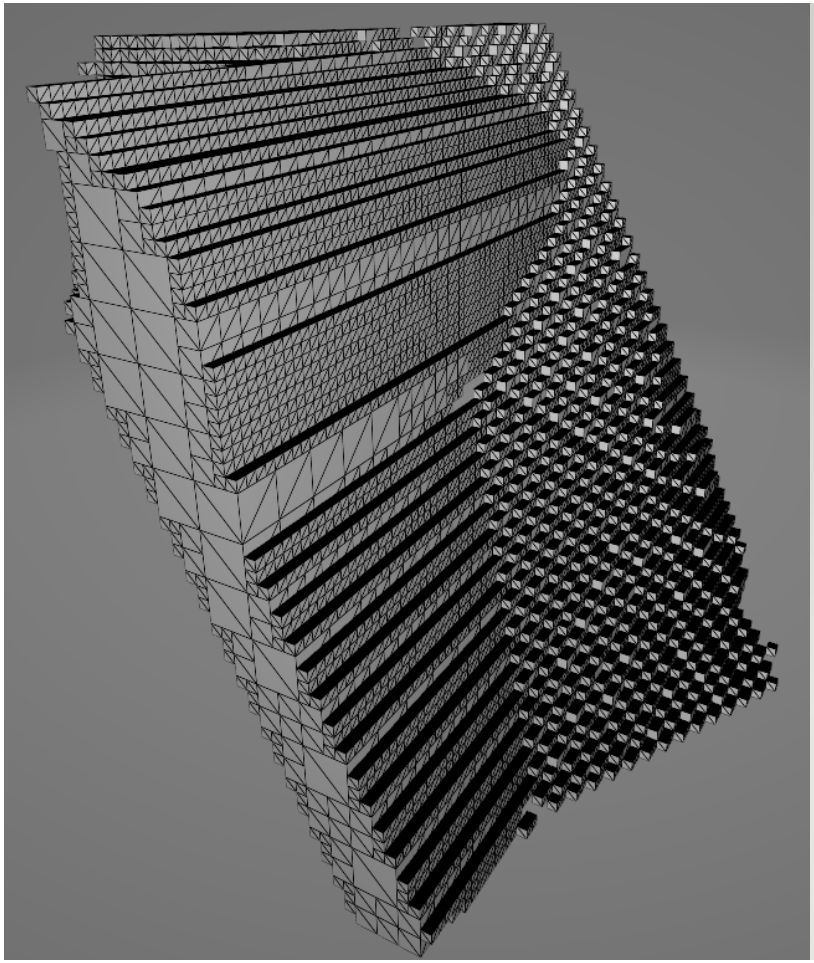
Divers

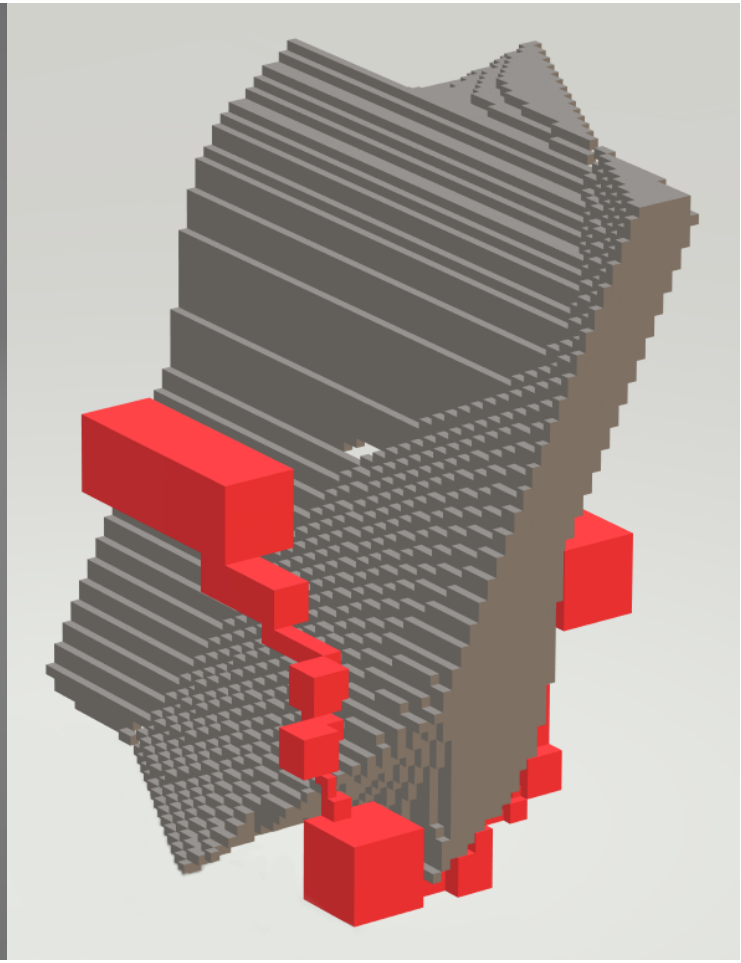
- Insertion récursive du départ et du but.
- Indexation des feuilles par un parcours en profondeur pour disposer d'un accès en coût constant via un tableau de pointeurs.

Visualisation du parcours en dimension 2



Visualisation du parcours en dimension 3





Exemple-type d'utilisation du code

```
Environnement2D Tree::env (-3., 3., -3., 3.); //environnement du bras robotique
const int num = pow(2,7); //fixation de la hauteur
//definition des parametres de discretisation
Discretization Tree::dis ({3, {2.*M_PI/3., 2.*M_PI/3., 2.*M_PI/3.}, {num,num,num}});
uint Tree::min_depth = 3; //profondeur minimale des feuilles

//mise en place des obstacles de l'environnement
Tree::env.setObstacles({Segment({{-.5,1.1},{-.5,3.}}),Segment({{.5,1.1},{.5,3.}})});
//construction de l'arbre
Tree t;
//insertion tardive des configurations initiale (s) et finale (g)
Tree* s = t.insert(systemToCoord(System({-0.2,-1,-1}), Tree::dis));
Tree* g = t.insert(systemToCoord(System({1.3,1.5,0.5}), Tree::dis));
//indexation des feuilles pour l'acces rapide
t.build_indices();

//recherche d'un chemin de s a g
std::vector<uint> path;
getPath_astar(s, g, path);

//dessin des etapes du chemin
for(uint i=0; i<path.size(); ++i)
    Tree::env.drawSystem(coordToSystem(Tree::nodes[path[i]]->get_bary(), Tree::dis),
    "D:/config"+QString::number(i)+".png", 200);
```

Dimension 2

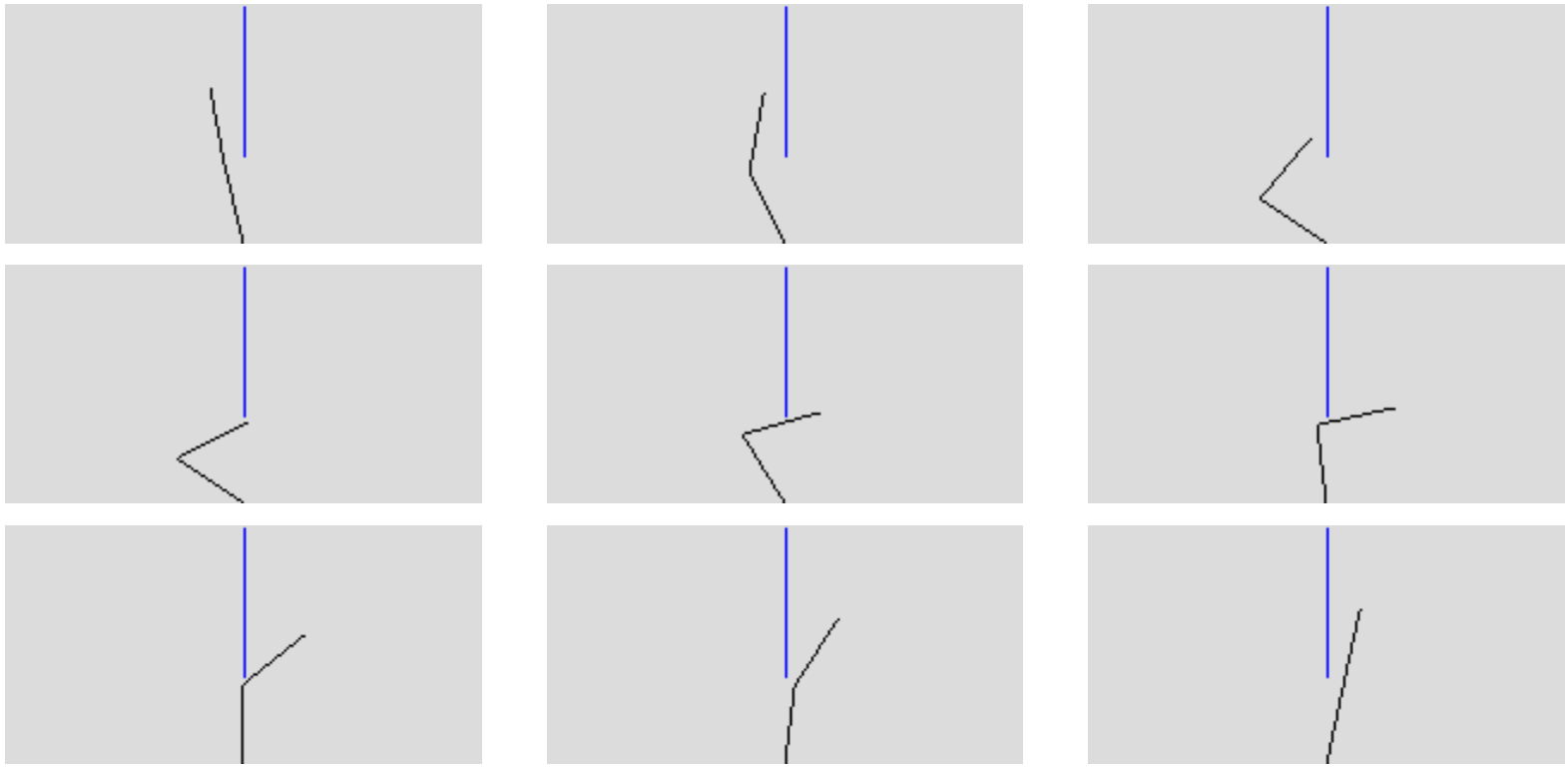


Figure – Résolution d'un problème de dimension 2.

Dimension 3

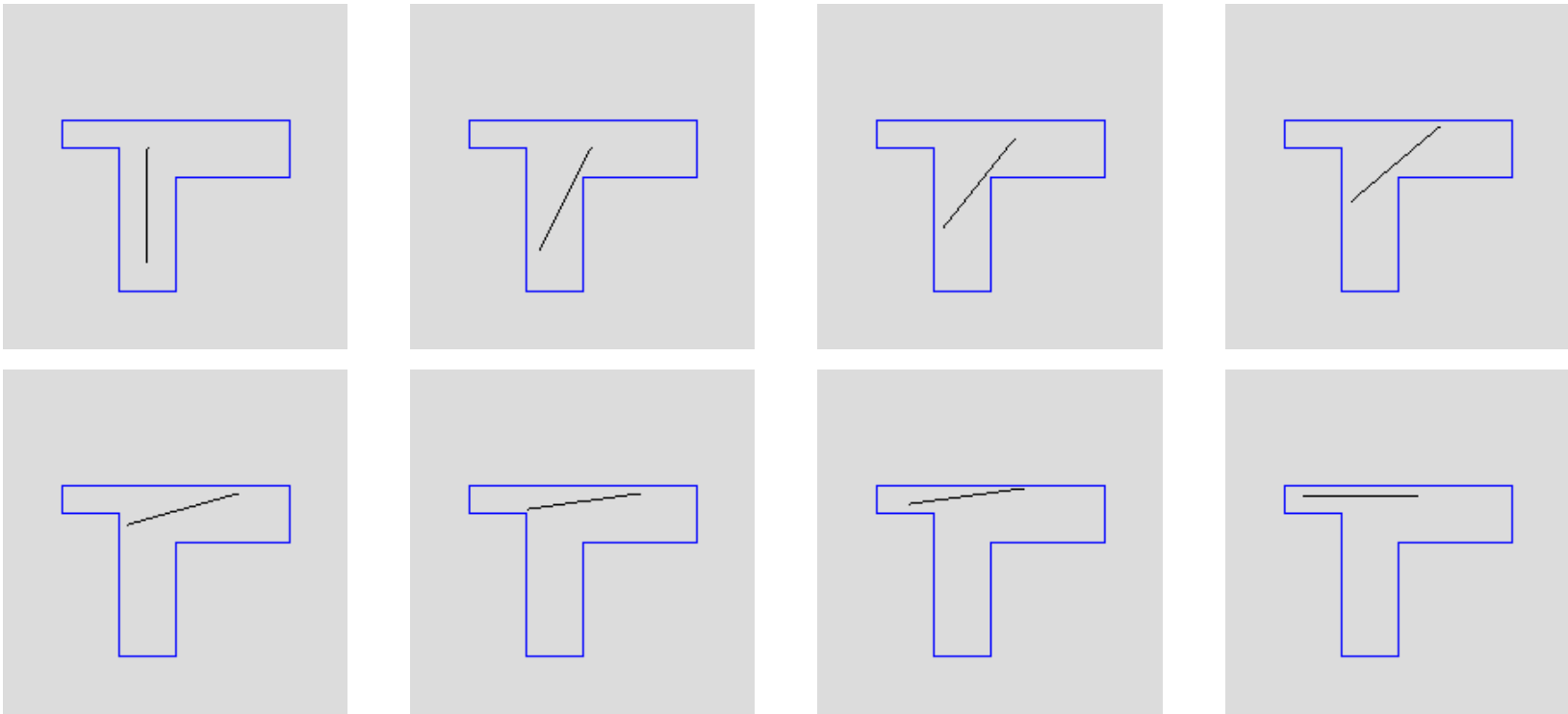


Figure – Résolution d'un problème de dimension 3.

Dimension 4

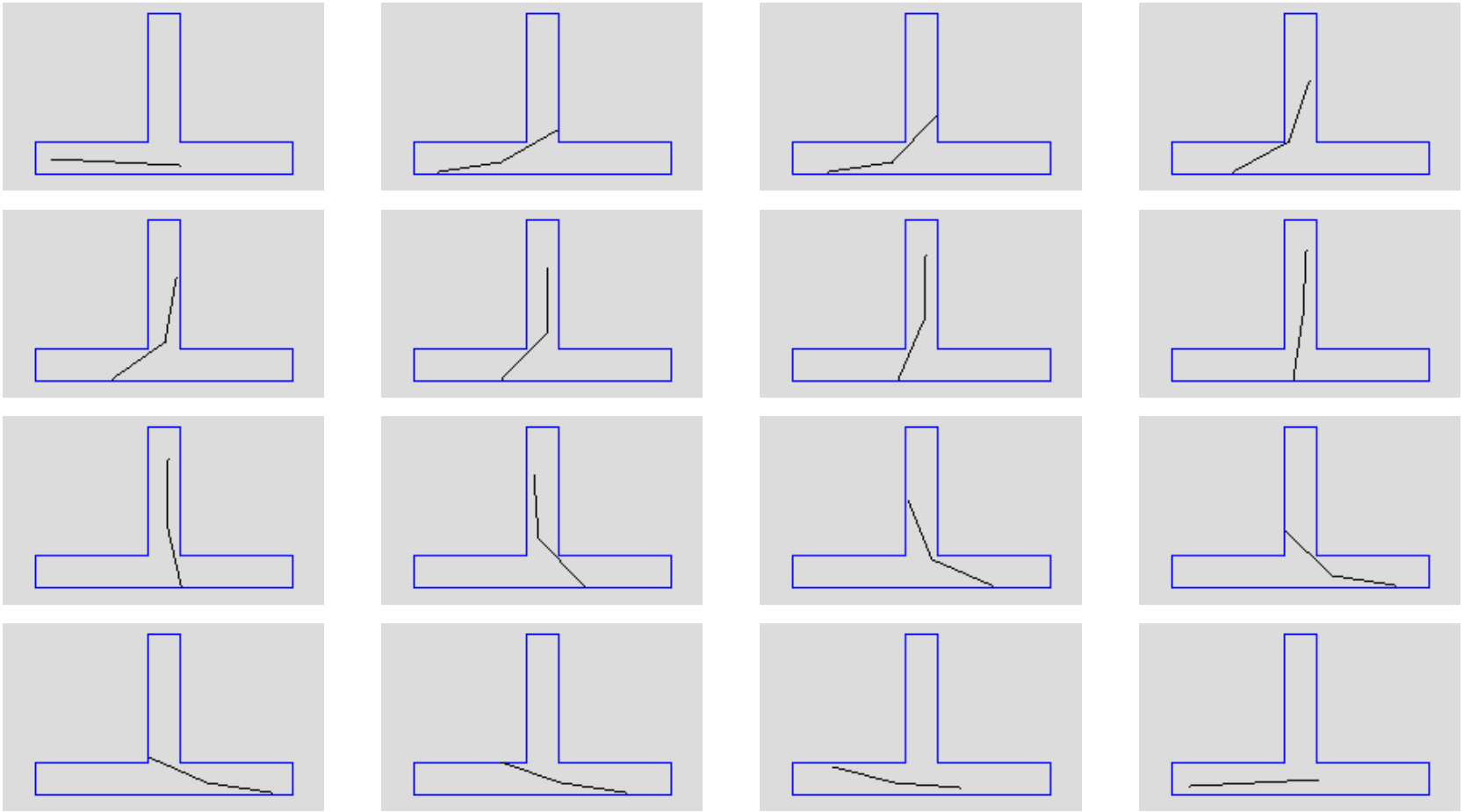


Figure – Résolution d'un problème de dimension 4.