Conception, spécification et preuve formelle d'algorithmes de calcul d'enveloppe convexe avec des cartes combinatoires en Coq

Christophe Brun, Jean-François Dufourd, Nicolas Magaud

LSIIT UMR 7005 CNRS-ULP
Université Louis Pasteur de Strasbourg

Projet Galapagos - Visioconférence - Lundi 21 janvier 2008

- Objectifs :
  - Améliorer les techniques de programmation en modélisation et en géométrie algorithmique

- Moyens utilisés :
  - Spécifications formelles et preuves de programmes : calcul des constructions inductives et preuves interactives en Coq
  - Modélisation géométrique à base topologique : cartes combinatoires orientées plongées

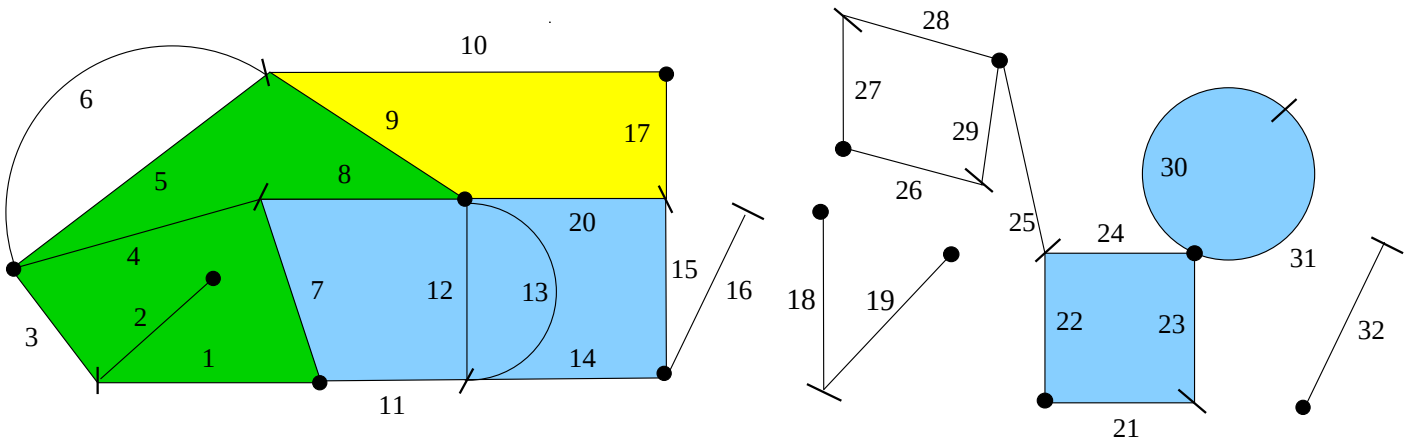
- Etude de cas : Algorithme incrémental de calcul de l'enveloppe convexe d'un ensemble fini de points du plan

- Etapes de développement :
  1. Conception d'un algorithme fonctionnel
  2. Extraction automatique d'un programme en Ocaml (saisie des données et visualisation du résultat)
  3. Preuve formelle de sa correction : mise en évidence de nombreuses propriétés topologiques et géométriques

- Algorithmes de calcul d'enveloppes convexes
  - David Pichardie et Yves Bertot, « Formalizing Convex Hull Algorithms ». *Theorem Proving in Higher Order Logics*. LNCS. Springer-Verlag, 2001.
  - Laura Meikle et Jacques Fleuriot, « Mechanical Theorem Proving in Computational Geometry ». *Automated Deduction in Geometry*. LNCS. Springer-Verlag, 2004.
- Cartes combinatoires par induction structurelle en Coq
  - Jean-François Dufourd, « Design and formal proof of a new optimal image segmentation program with hypermaps ». *Pattern Recognition*. Elsevier, 2007.
  - Jean-François Dufourd, « A hypermap framework for computer-aided proofs in surface subdivisions, Genus theorem and Euler formula ». *ACM SAC*. ACM Press, 2007.

Définition 2.1 (Hypercarte)

- Une hypercarte est une structure algébrique $M = (D, \alpha_0, \alpha_1)$, où D est un ensemble fini dont les éléments sont appelés des brins et α_0, α_1 sont des permutations dans D.
- Si $y = \alpha_k(x)$, y est le k-successeur de x, x est le k-prédécesseur de y, et x et y sont dits k-liés ou k-cousus.



Définition 2.2 (Orbite)

Soient D un ensemble fini, σ une permutation dans D et x un élément de D. L'orbite de x par rapport à σ est le sous-ensemble de D des éléments accessibles depuis x, noté $\langle\sigma\rangle(x)$, défini par $\langle\sigma\rangle(x) = \{x, \sigma(x), \sigma^2(x), ..., \sigma^k(x)\}$, où k est le plus petit entier naturel tel que $\sigma^{k+1}(x) = x$.

Définition 2.3 (Cellules topologiques)

Dans l'hypercarte $M = (D, \alpha_0, \alpha_1)$, $\langle\alpha_0\rangle(x)$ représente l'arête du brin x, $\langle\alpha_1\rangle(x)$ son sommet, $\langle\alpha_1^{-1} \circ \alpha_0^{-1}\rangle(x)$ sa face, et $\langle\alpha_0, \alpha_1\rangle(x)$ sa composante connexe. Arêtes, sommets et faces sont aussi appelés k-cellules topologiques de la carte pour k = 0, 1 et 2 respectivement.

- Définition de types

```
Definition dart := nat.
Definition nil := 0%nat.

Definition point := (R * R)%type.

Inductive dim : Set := di0 : dim | di1 : dim.
```

- Décidabilité de l'égalité

```
Lemma eq_dim_dec :
  forall (i:dim)(j:dim), {i=j} + {~i=j}.

Definition eq_dart_dec := eq_nat_dec.
```

- Définition du type `fmap` de carte libre

```
Inductive fmap : Set :=
    V : fmap
  | I : fmap -> dart -> point -> fmap
  | L : fmap -> dim -> dart -> dart -> fmap.
```

- Définition de prédicats et de fonctions

```
Fixpoint exd (m:fmap)(x:dart) {struct m} : Prop :=
  match m with
    V => False
  | I m0 x0 _ => x = x0 \/ exd m0 x
  | L m0 _ _ _  => exd m0 x
  end.
```

```
Fixpoint A (m:fmap)(k:dim)(x:dart) {struct m} : dart :=
  match m with
    V => nil
  | I m0 x0 _ _ => A m0 k x
  | L m0 k0 x0 y0 =>
    if (eq_dim_dec k k0)
    then if (eq_dart_dec x x0) then y0
         else A m0 k x
    else A m0 k x
  end.

Definition succ (m:fmap)(k:dim)(x:dart) : Prop :=
  A m k x <> nil.

Lemma succ_dec: forall (m:fmap)(k:dim)(x:dart),
  {succ m k x} + {~succ m k x}.
```

- Définition des préconditions `prec_I` et `prec_Lq`

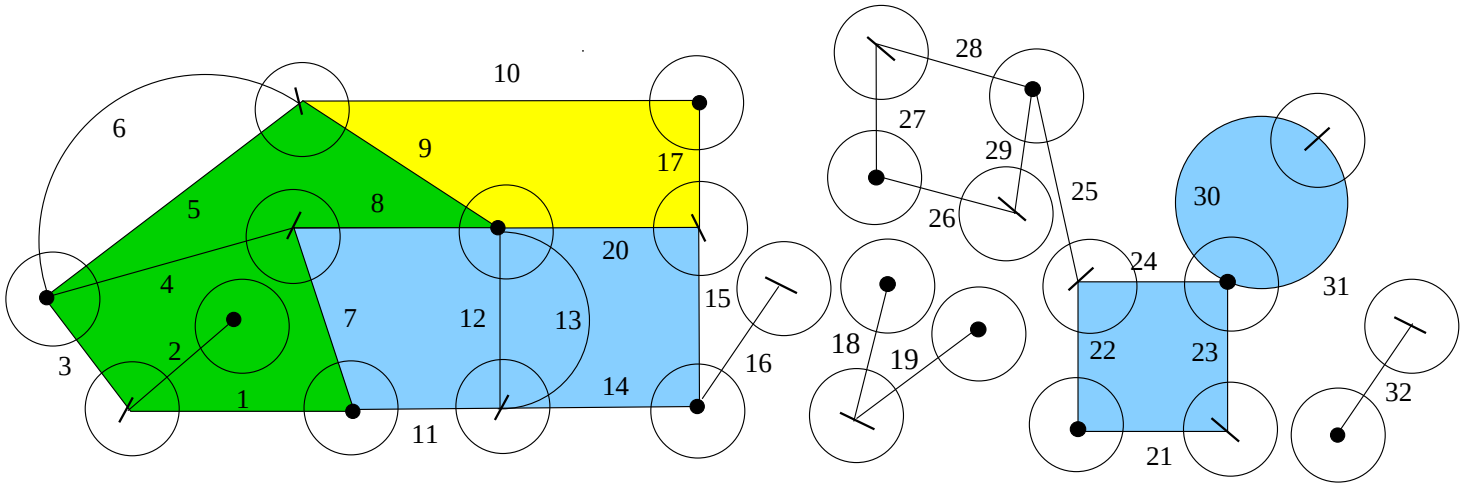
```
Definition prec_I (m:fmap)(x:dart) : Prop :=
  x <> nil /\ ~ exd m x .

Definition prec_Lq (m:fmap)(k:dim)(x y:dart) : Prop :=
  exd m x /\ exd m y /\ ~ succ m k x /\ ~ pred m k y.
```

- Définition de l'invariant `inv_qhmap`

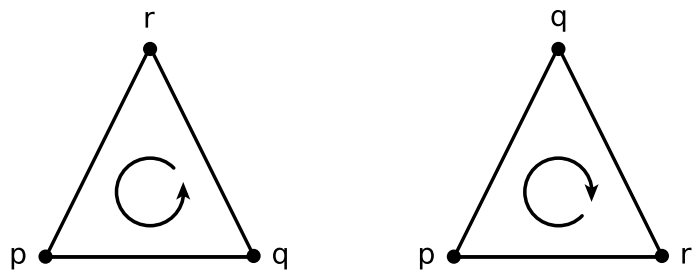
```
Fixpoint inv_qhmap (m:fmap) {struct m} : Prop :=
  match m with
    V => True
  | I m0 x0 _ => inv_qhmap m0 /\ prec_I m0 x0
  | L m0 k0 x0 y0 => inv_qhmap m0 /\ prec_Lq m0 k0 x0 y0
  end.
```

- Définition de l'invariant `inv_hmap`

```
Definition inv_hmap (m:fmap) : Prop :=
  inv_qhmap m /\ forall (x:dart)(k:dim),
  exd m x -> succ m k x /\ pred m k x.
```



- Prédicat géométrique d'orientation



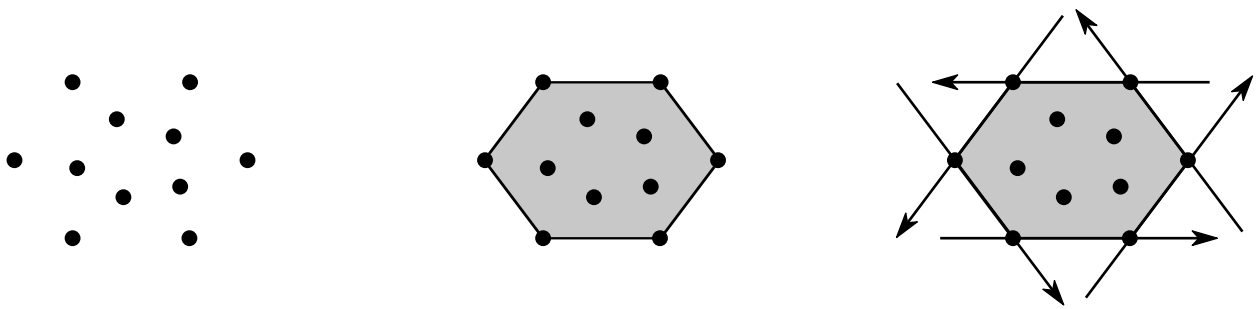
- Axiomatisation de Knuth :
  distinction entre prédicats et calculs numériques
  - Résolution des problèmes de robustesse
  - Isoler les calculs numériques
  - Focaliser principalement sur les aspects algorithmiques

- Points en position générale :
  tous distincts 2 à 2 et jamais 3 points alignés

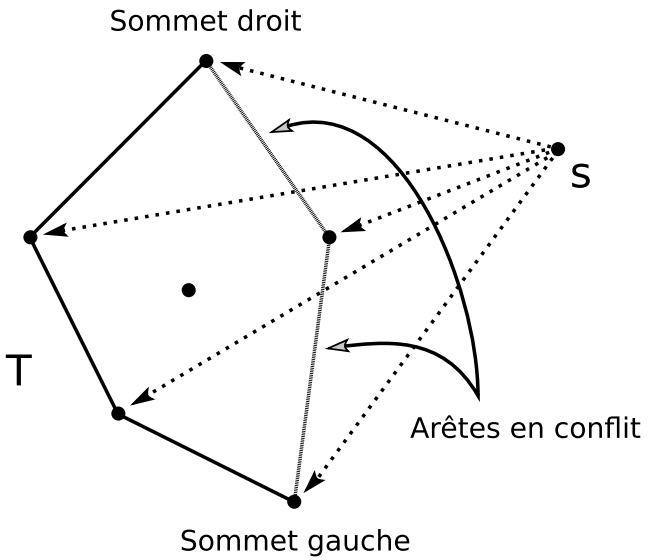
Soit P un ensemble fini de points du plan euclidien E.

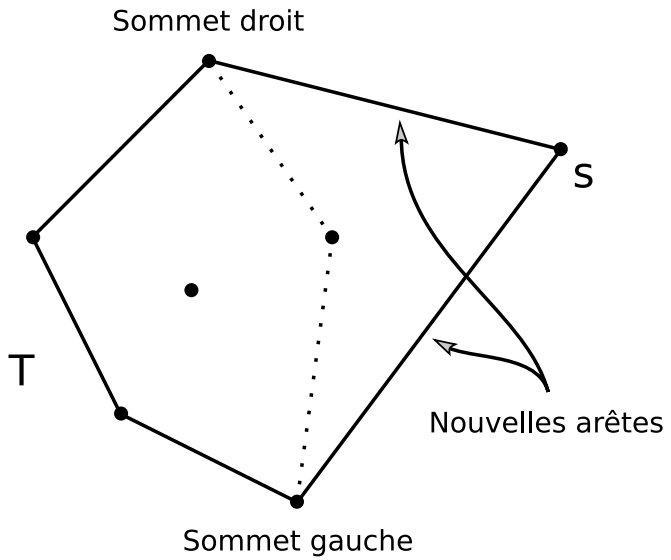
Définition 4.1 (Enveloppe convexe)

L'enveloppe convexe de P est le polygone convexe T dont les sommets sont des points de P et tel que, pour toute arête $[t_i t_{i+1}]$ de T et pour tout point p de P différents de t_i et de t_{i+1}, on a ccw$(t_i, t_{i+1}$,p). Autrement dit, tous les points p de P différents de t_i et de t_{i+1} se situent à gauche de la droite orientée engendrée par le vecteur $\overrightarrow{t_i t_{i+1}}$.

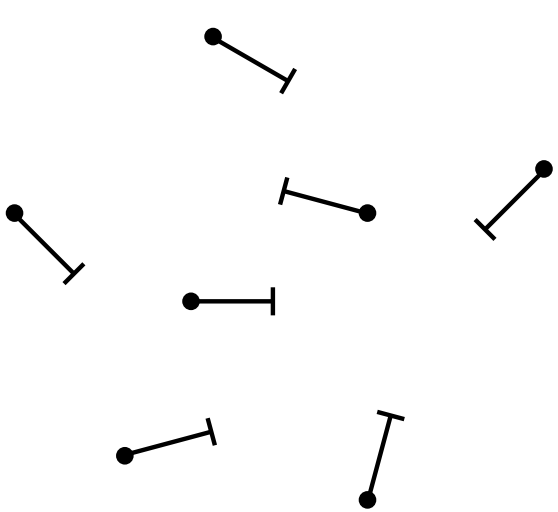


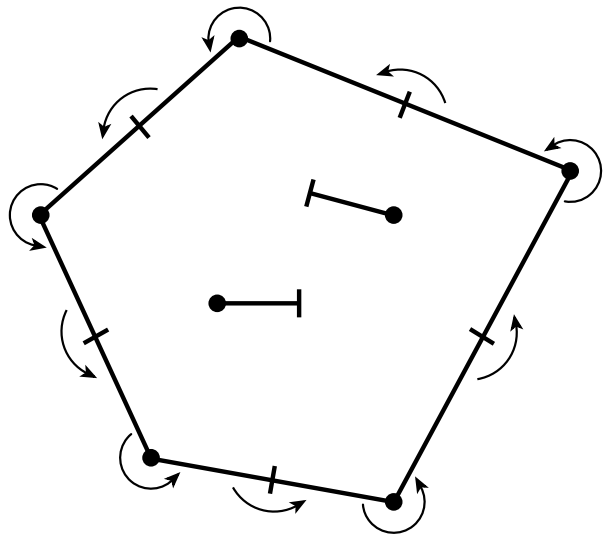
- L'algorithme incrémental
  - Construction progressive d'une enveloppe convexe par l'insertion successive des points les uns apres les autres
  - Si point intérieur alors étape suivante sinon suppression des arêtes en conflit et ajout des deux arêtes le reliant aux sommets gauche et droit



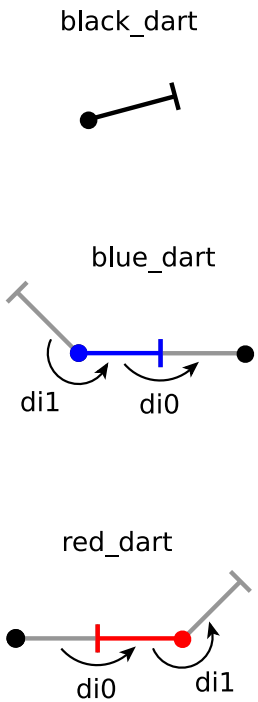


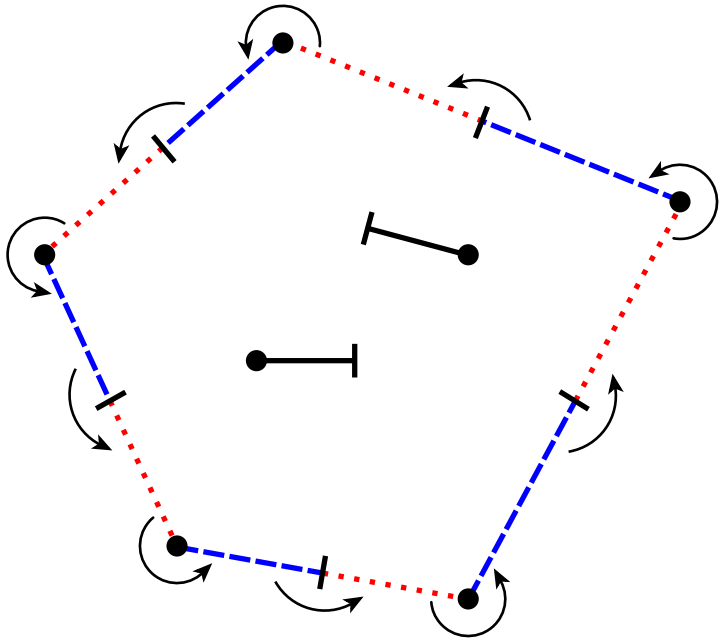
- Représentation sous forme de carte
  - L'ensemble fini de points du plan :
    un point = un brin isolé sans couture
  - L'enveloppe convexe : polygone topologique orienté
    (quasi-permutations)





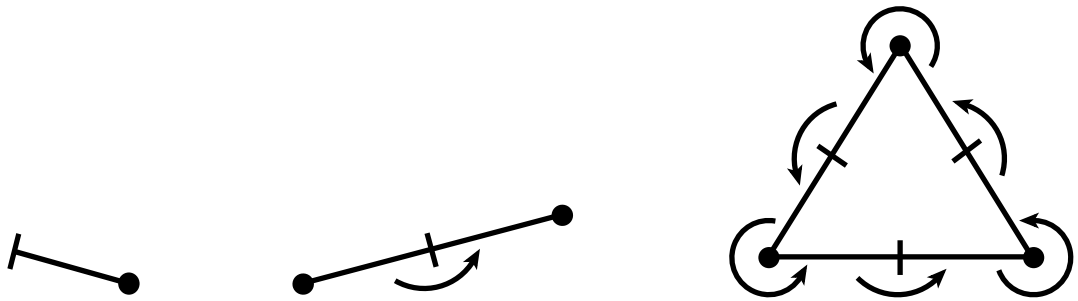
- Classification très précise des brins en 3 catégories :
  - **brins noirs** = brins isolés sans couture
  - brins bleus = brins avec un 1-prédécesseur et un 0-successeur
  - brins rouges = brins avec un 0-prédécesseur et un 1-successeur





- Définition de la fonction principale CH

```
Definition CH (m:fmap) : fmap :=
  match m with
    V => V
  | I V x p => I V x p
  | I (I V x1 p1) x2 p2 => L (I (I V x1 p1) x2 p2) di0 x1 x2
  | I (I (I m0 x1 p1) x2 p2) x3 p3 =>
    CHI m0 (CH3 x1 p1 x2 p2 x3 p3 (max_dart m)) ((max_dart m)+4)
  | _ => V
  end.
```



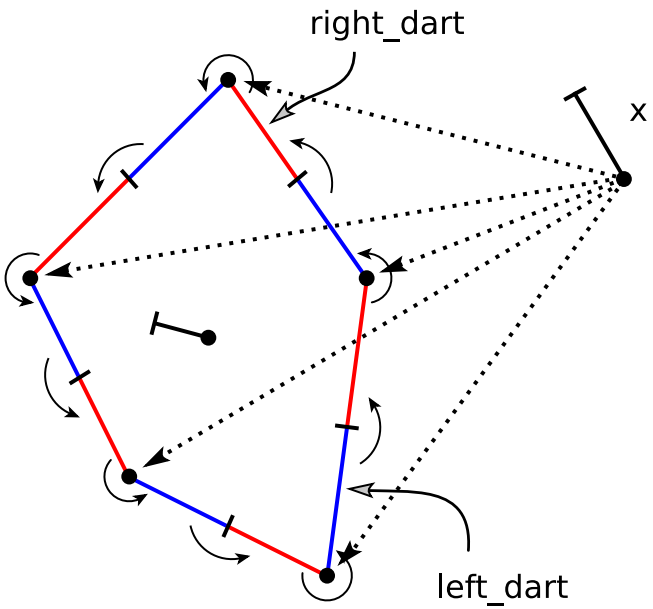
- Définition de la fonction incrémentale CHI

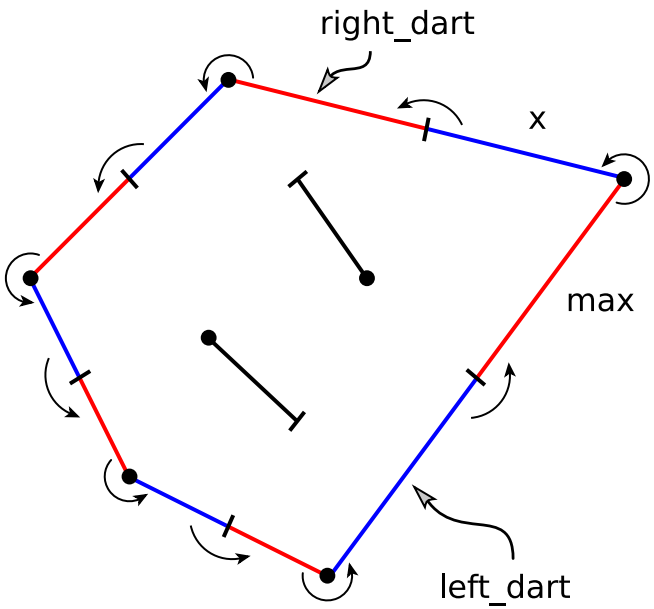
```
Fixpoint CHI (m1:fmap)(m2:fmap)(max:dart) {struct m1} : fmap :=
  match m1 with
    V => m2
  | I m0 x p => CHI m0 (CHID m2 m2 x p max) (max+1)
  | _ => V
  end.
```

- Définition de la fonction CH3 qui construit un triangle orienté

```
Definition CH3 (x1:dart)(p1:point)(x2:dart)(p2:point)
  (x3:dart)(p3:point)(max:dart) : fmap :=
  let m0 := (I (I (I V x1 p1) x2 p2) x3 p3) in
  let m1 := L (I m0 (max+1) p1) di1 (max+1) x1 in
  let m2 := L (I m1 (max+2) p2) di1 (max+2) x2 in
  let m3 := L (I m2 (max+3) p3) di1 (max+3) x3 in
  if (ccw_dec p1 p2 p3) then
    L (L (L m3 di0 x1 (max+2)) di0 x2 (max+3)) di0 x3 (max+1)
  else L (L (L m3 di0 x1 (max+3)) di0 x2 (max+1)) di0 x3 (max+2).
```

- Définition de la fonction d'insertion `CHID`

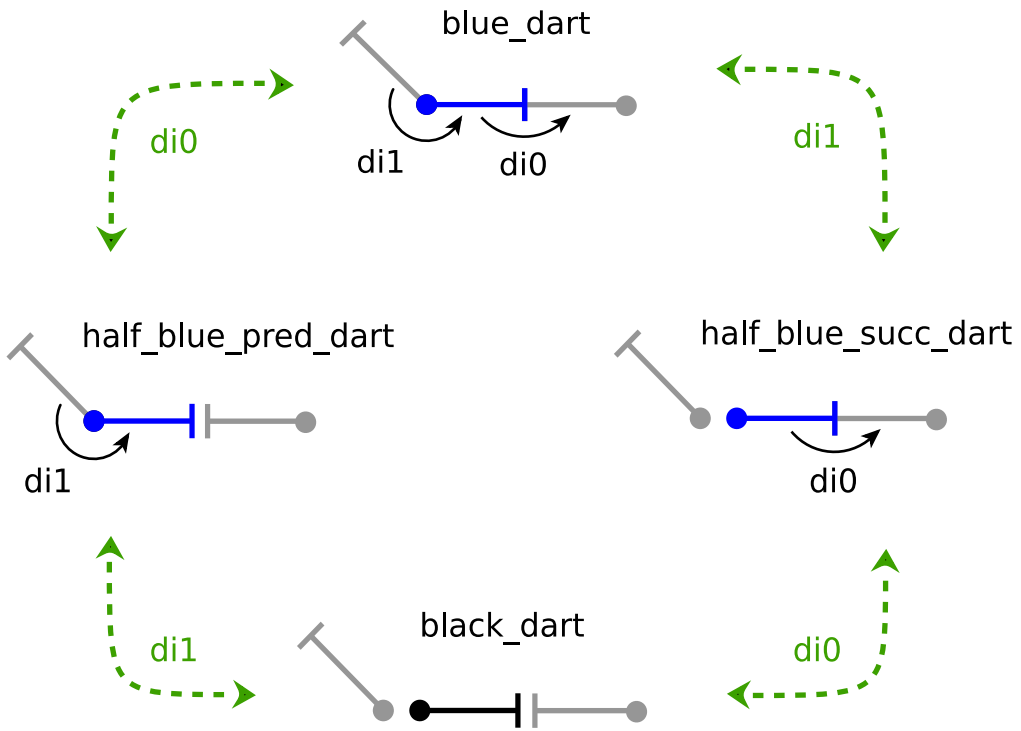




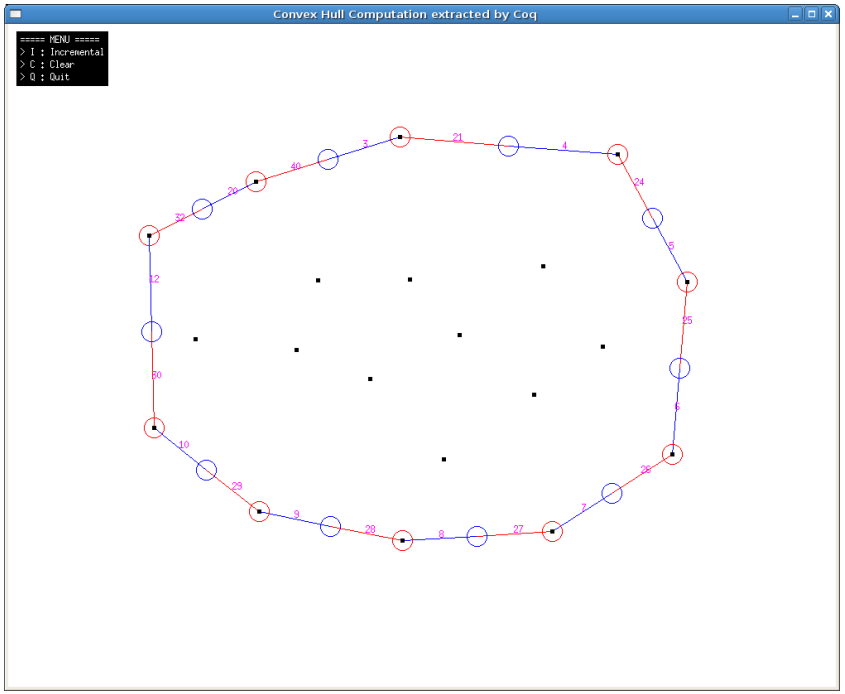
```
Fixpoint CHID (m:fmap)(mr:fmap)(x:dart)(p:point)(max:dart)
  {struct m} : fmap :=
  match m with
    V => I V x p
  | I m0 x0 p0 =>
    if (blue_dart_dec mr x0) then
      if (ccw_dec (fpoint mr x0) (fpoint mr (A mr di0 x0)) p) then
        (I (CHID m0 mr x p max) x0 p0)
      else if (left_dart_dec mr p x0) then
(L (L (I (I (CHID m0 mr x p max) x0 p0) max p) di1 max x) di0 x0 max)
           else (I (CHID m0 mr x p max) x0 p0)
    else if (red_dart_dec mr x0) then [...]
         else (I (CHID m0 mr x p max) x0 p0)
  | L m0 di0 x0 y0 =>
    if (ccw_dec (fpoint mr x0) (fpoint mr y0) p) then
      (L (CHID m0 mr x p max) di0 x0 y0)
    else (CHID m0 mr x p max)
  | L m0 di1 x0 y0 => [...]
  end.
```

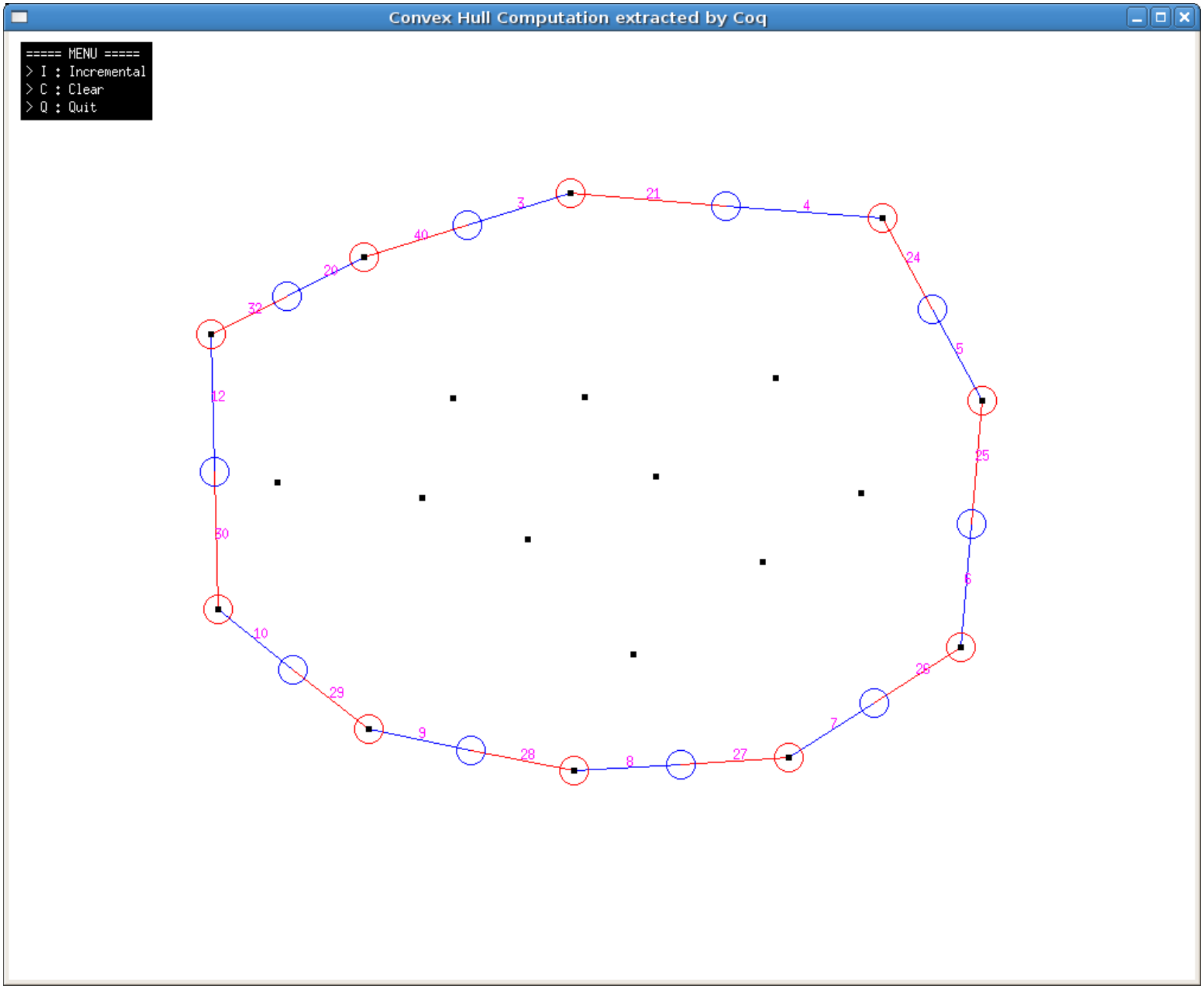
- Description de l'algorithme par induction structurelle
  - Traitement indéterministe et séparé des brins et des coutures
  - Pas de stratégie de parcours dans la carte
  - Nécessite une carte de référence `mr` pour les tests et une notion de sous-carte définie par le prédicat `submap`
- Carte orientée avec des orbites non closes
  - Moins de liaisons à casser
  - Orientation immédiate

- Modification de la classification des brins : perte ou création de nouvelles liaisons



- Mécanisme d'extraction
  - Engendrer automatiquement des programmes certifiés en Objective Caml
- Interface graphique
  - Saisie des données (les points du plan)
  - Visualisation du résultat (l'enveloppe convexe)




Convex Hull Computation extracted by Coq
===== MENU =====
> I : Incremental
> C : Clear
> Q : Quit
3
21
4
40
24
20
5
32
12
25
30
6
10
26
29
7
9
28
8
27

- Correction totale = terminaison + correction partielle
  - Terminaison : imposée et garantie par Coq
  - Correction partielle
    - Propriétés topologiques
    - Propriétés géométriques
- Preuve par induction structurelle sur les cartes
- Préconditions sur la carte initiale

```
Definition prec_CH (m:fmap) : Prop :=
  inv_qhmap m /\ linkless m /\
  distinct_emb m /\ noalign m.
```

- Propriétés topologiques
  - Conservation des brins de la carte initiale dans la carte finale

```
Lemma inv_exd : forall (m:fmap)(x:dart),
  inv_qhmap m -> linkless m ->
  exd m x -> exd (CH m) x.
```

  - Conservation de l'invariant des quasi-hypercartes

```
Lemma inv_qhmap_CH : forall (m:fmap),
  inv_qhmap m -> linkless m -> inv_qhmap (CH m).
```

  - Raffinement des cartes par ajout d'invarients
    - inv_gmap : pas de points fixe + liaisons = demi-involutions
    - inv_gmap_orient : lignes brisées orientées par les coutures
    - inv_gmap_poly : uniquement des brins noirs, bleus ou rouges
  - Une seule composante connexe après suppression des brins intérieurs isolés (relation d'Euler, théorème du genre)

```
Lemma inv_exd : forall (m:fmap)(x:dart),
  inv_qhmap m -> linkless m -> exd m x -> exd (CH m) x.
Proof.
induction m.
 (* Case 1 : m = V *)
 (* Case 2 : m = I *)
 induction m.
  (* Case 2.1 : m = I V *)
  (* Case 2.2 : m = I I *)
  induction m.
   (* Case 2.2.1 : m = I I V *)
   (* Case 2.2.2 : m = I I I *)
    (* H1 : x = d *)
    (* H2 : x = d0 *)
    (* H3 : x = d1 *)
    (* H4 : exd m x *)
   (* Case 2.2.3 : m = I I L *)
  (* Case 2.3 : m = I L *)
 (* Case 3 : m = L *)
Qed.
```

```
Lemma inv_exd_m1 :
  forall (m1:fmap)(m2:fmap)(max:dart)(x:dart),
  inv_qhmap m1 -> linkless m1 ->
  exd m1 x -> x < max ->
  (black_dart m2 x) \/ (blue_dart m2 x) \/
  (half_blue_succ_dart m2 x) \/ (half_blue_pred_dart m2 x) ->
  exd (CHI m1 m2 max) x.

Lemma inv_exd_m2 :
  forall (m1:fmap)(m2:fmap)(max:dart)(x:dart),
  inv_qhmap m1 -> linkless m1 ->
  exd m2 x ->  x < max ->
  (black_dart m2 x) \/ (blue_dart m2 x) \/
  (half_blue_succ_dart m2 x) \/ (half_blue_pred_dart m2 x) ->
  exd (CHI m1 m2 max) x.
```

```
Lemma lemma01 :
forall (m:fmap)(mr:fmap)(x:dart)(tx:tag)(px:point)(max:dart)(d:dart),
  exd m d -> (black_dart mr d) \/ (blue_dart mr d) \/
  (half_blue_succ_dart mr d) \/ (half_blue_pred_dart mr d) ->
  exd (CHID m mr x tx px max) d.

Lemma lemma02 :
forall (m:fmap)(mr:fmap)(x:dart)(tx:tag)(px:point)(max:dart)(d:dart),
  submap m mr -> d <> max ->
  (black_dart mr d) \/ (blue_dart mr d) \/
  (half_blue_succ_dart mr d) \/ (half_blue_pred_dart mr d) ->
  (black_dart (CHID m mr x tx px max) d) \/
  (blue_dart (CHID m mr x tx px max) d) \/
  (half_blue_succ_dart (CHID m mr x tx px max) d) \/
  (half_blue_pred_dart (CHID m mr x tx px max) d).
```

- Propriétés géométriques
  - Unicité et équivalence de l'existance des extrémités gauche et droite pour les arêtes en conflit
  - Bon plongement des brins par rapport à leures coutures

```
Fixpoint inv_emb (m:fmap) {struct m} : Prop :=
  match m with
    V => True
  | I m0 x t p => inv_emb m0
  | L m0 di0 x y => inv_emb m0 /\
    (fpoint m0 x) <> (fpoint m0 y)
  | L m0 di1 x y => inv_emb m0 /\
    (fpoint m0 x) = (fpoint m0 y)
  end.

Lemma inv_emb_CH : forall (m:fmap),
  prec_CH m -> inv_emb (CH m).
```

- Propriétés géométriques [suite]
  - Caractérisation de l'enveloppe convexe

```
Definition convex (m:fmap) : Prop :=
  forall (x:dart), exd m x -> blue_dart m x ->
  forall (z:dart), exd m z ->
  (fpoint m z) <> (fpoint m x) ->
  (fpoint m z) <> (fpoint m (A m di0 x)) ->
  ccw (fpt m x) (fpt m (A m di0 x)) (fpt m z).

Lemma convex_CH : forall (m:fmap),
  inv_qhmap m -> linkless m -> convex (CH m).
```

- Interdépendance forte entre topologie et géométrie

8. Conclusion : Bilan

- Description formelle d'un algorithme incrémental de calcul d'enveloppe convexe
  - basé sur une modélisation à base topologique (cartes combinatoires)
  - programmé par induction structurelle dans le langage fonctionnel de Coq
- Extraction d'un prototype de construction exécutable en OCaml
- Certification formelle : démonstration de la terminaison et mise en évidence de nombreuses propriétés topologiques et géométriques

- Utilisation de l'induction noethérienne pour parcourir les cartes dans la suite des brins et pas par sélection désordonnée
- Généralisation des travaux
  - Autres algorithmes de calcul d'enveloppe convexe (parcours de Graham ou marche de Jarvis)
  - Dimensions supérieures
  - Subdivisions plus complexes (triangulation de Delaunay ou diagramme de Voronoi)