OI-OPPA. European Social Fund
Prague & EU: We invest in your future.

Delaunay triangulace pomocí DeWall algoritmu

István Módos

modosist@fel.cvut.cz

DeWall algoritmus

- Divide & Conquer
- zobecněný algoritmus Delaunay triangulace pro body v d-dimenzionálním prostoru

Kroky algoritmu, funkce *dewall*

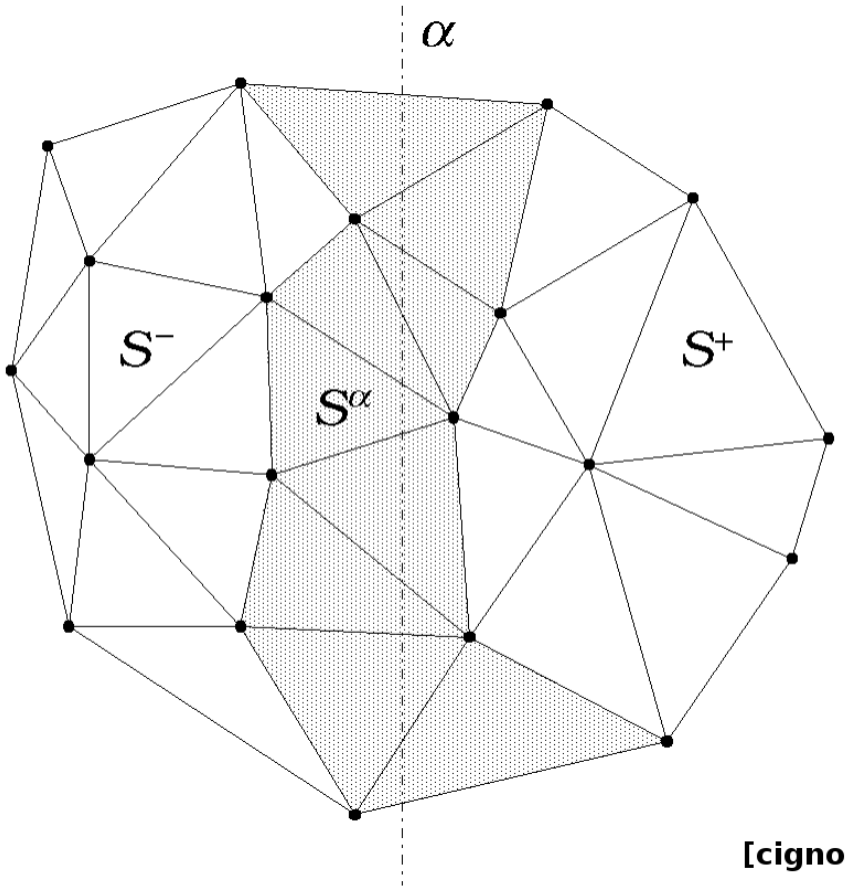
Vstup: body P, které mají být triangulovány
Výstup: triangulace vstupních bodů P

1. Nalezni přímku α, která rozdělí množinu P na dvě disjunktní množiny P_1 a P_2
2. Sestroj *simplexovou stěnu* Σ_α
3. Rekurzivně aplikuj funkci *dewall* předchozí kroky na množiny P_1 a P_2, čímž dostaneme triangulace Σ_1 a Σ_2
4. Vrať sjednocení triangulací $\Sigma_\alpha, \Sigma_1, \Sigma_2$

Simplexová stěna Σ_α

Předpokládejme, že jsme v *i*-tém rekurzivním volání algoritmu nalezli triangulaci Σ_α. Množina trojúhelníků, které jsou protínány přímkou α, se nazývá *simplexová stěna*.

Příklad simplexové stěny



[cignoni]

Konstrukce simplexové stěny

1. Nalezení prvního trojúhelníku, který protíná stěnu
2. Inkrementální konstrukce zbylých trojúhelníků z hran, které jsou protínány rozdělující přímkou

Vytvoření prvotního trojúhelníku

1. Nalezni bod $\mathbf{p}_1$, který je nejblíže k přímce α
2. Nalezni bod $\mathbf{p}_2$, který je nejblíže k bodu $\mathbf{p}_1$ a neleží s ním ve stejné polorovině
3. Nalezni další bod $\mathbf{p}_3$ takový, že kružnice opsaná trojúhelníku $\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3$ má minimální poloměr

Nalezení zbylých trojúhelníků simplexové stěny

Zavedmě si tři seznamy hran:

- AFL_α - seznam aktivních hran, které jsou protínány přímkou α
- AFL_1 - seznam hran, jejichž všechny vrcholy se nacházejí v množině P_1
- AFL_2 - seznam hran, jejichž všechny vrcholy se nacházejí v množině P_2

Při prvním volání funkce *dewall* vytvoříme prvotní trojúhelník, jehož hranami naplníme příslušné seznamy.

Nalezení zbylých trojúhelníků simplexové stěny

Mějme hranu $\mathbf{e} := (\mathbf{p}_b, \mathbf{p}_e)$, která je protínána přímkou α. Budeme hledat takový bod $\mathbf{p} \in P_1 \cup P_2$, který minimalizuje *Delaunay vzdálenost*

$$dd(\mathbf{e}, \mathbf{p}) = \begin{cases} r(\mathbf{e}, \mathbf{p}) & \text{pokud } \mathbf{p}_b\mathbf{p}_e\mathbf{c}(\mathbf{e}, \mathbf{p}) \text{ tvoří pravou otočku} \\ -r(\mathbf{e}, \mathbf{p}) & \text{jinak} \end{cases}$$

kde $\mathbf{c}()$ vrátí střed kružnice opsané trojúhelníku $\mathbf{p}_b\mathbf{p}_e\mathbf{p}$ a $r()$ vrátí její poloměr. Hledané body $\mathbf{p}$ ještě omezíme na ty, pro které platí, že orientace $\mathbf{p}_b\mathbf{p}_e\mathbf{p}$ tvoří pravou otočku.
Pokud bod $\mathbf{p}$ minimalizuje Delaunay vzdálenost, potom trojúhelník $\mathbf{p}_b\mathbf{p}_e\mathbf{p}$ je prvkem simplexové stěny a zároveň i výsledné triangulace.

Nalezení zbylých trojúhelníků simplexové stěny

Do doby, než je seznam AFL_α prázdný, tak provádíme

1. Odstraň první hranu $e := (\mathbf{p}_b, \mathbf{p}_e)$ z AFL_α
2. Vytvoř nový trojúhelník t simplexové zdi z hrany e (viz. předchozí slide)
3. Pokud $t \neq \mathit{null}$, tak všechny hrany tohoto trojúhelníka vlož do příslušných seznamů (pokud se daná hrana v seznamu již nachází, tak ji odstraň a znovu nepřidávej)

Máme simplexovou stěnu, co dál?

Nyní rekurzivně zavoláme funkci *dewall* na množiny P_1 a P_2 (za předpokladu, že seznam hran AFL_1, resp. AFL_2, není prázdný), čímž získáme triangulace Σ_1 a Σ_2. Výsledná triangulace v i-tém volání funkce *dewall* je sjednocení triangulací $\Sigma_\alpha \cup \Sigma_1 \cup \Sigma_2$, která je vrácena volající funkci.

Pseudokód

```
Function DeWall (P : point_set, AFL : (d-1)face_list) : d-simplex_list;
    var f : (d-1)face; AFL_α, AFL_1, AFL_2 : (d-1)face_list;
        t : d-simplex; Σ : d-simplex_list; α : splitting_plane;
    begin
        AFL_α, AFL_1, AFL_2:=emptylist;
        Pointset_Partition(P, α, P_1, P_2);
    /* Simplex Wall Construction */
        if AFL = ∅ then
                t:=MakeFirstSimplex(P, α);
                AFL:=(d-1)faces(t); Insert(t,Σ);
        for each f ∈ AFL do
                if IsIntersected(f,α) then Insert(f, AFL_α);
                if Vertices(f) ⊂ P_1 then Insert(f, AFL_1);
                if Vertices(f) ⊂ P_2 then Insert(f, AFL_2);
```

Pseudokód - pokračování

```
while AFL_α ≠ ∅ do
        f:=Extract(AFL_α);
        t:=MakeSimplex(f, P);
        if t ≠ null then
                Σ:=Σ ∪ {t};
                for each f': f' ∈ (d−1)faces(t) AND f' ≠ f do
                    if IsIntersected(f',α) then Update(f',AFL_α)
                    if Vertices(f') ⊂ P_1 then Update(f',AFL_1)
                    if Vertices(f') ⊂ P_2 then Update(f',AFL_2);
    /* Recursive Triangulation */
    if AFL_1 ≠ ∅ then Σ:=Σ ∪ DeWall(P_1,AFL_1);
    if AFL_2 ≠ ∅ then Σ:=Σ ∪ DeWall(P_2,AFL_2);
    DeWall:=Σ;
end;
```

[cignoni]

Nalezení přímky α

Přímku se snažíme volit tak, aby v každém volání funkce *dewall* měly množiny P_1 a P_2 přibližně stejně velkou mohutnost. Směr přímky volíme cyklicky tak, aby byla ortogonální k jednotlivým osám prostoru.

Příklad

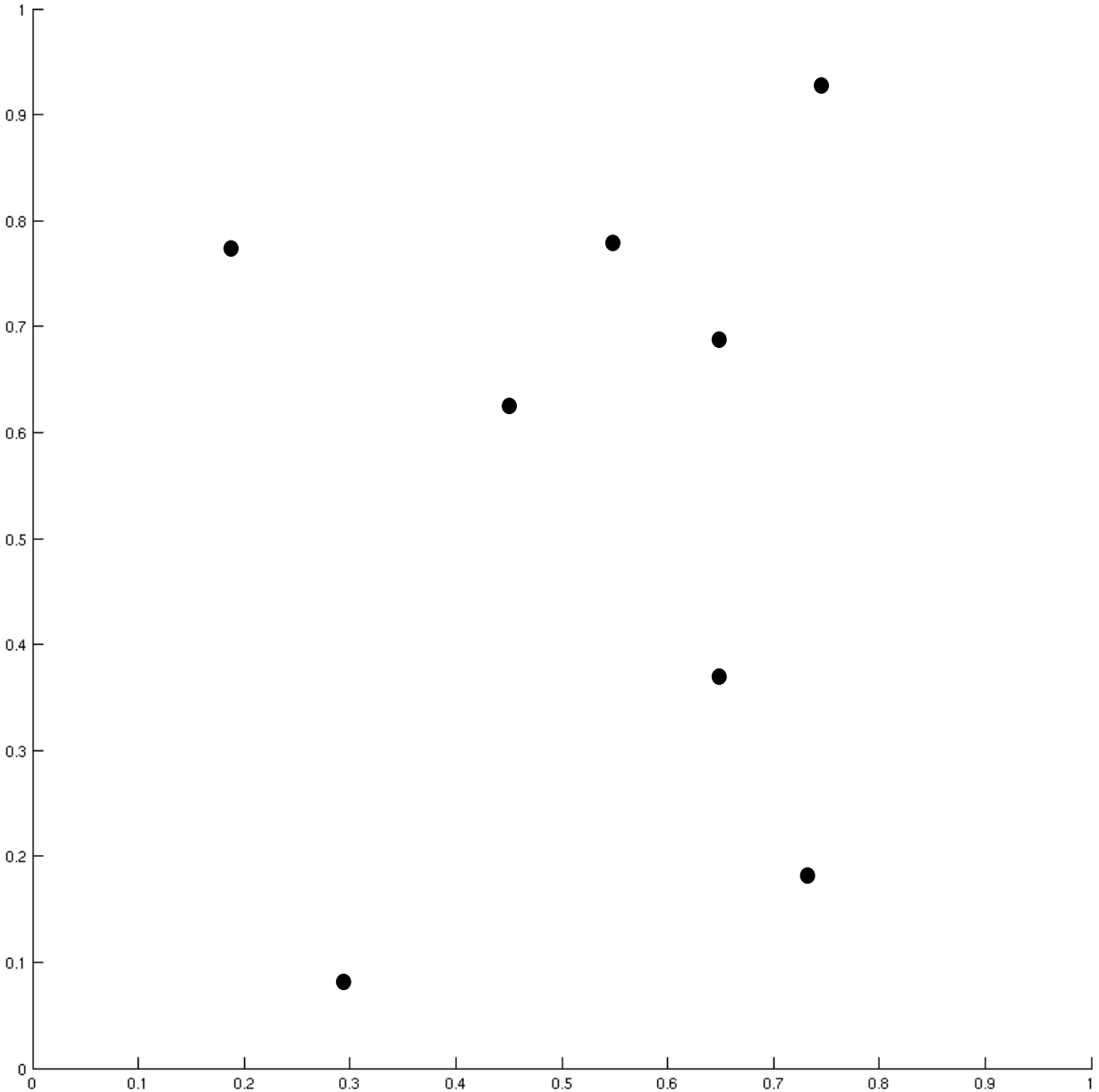
Zpracované hrany

Hrany v AFL $_\alpha$

Hrany v AFL $_1$

Hrany v AFL $_2$

Zpracovávaná hrana



Příklad

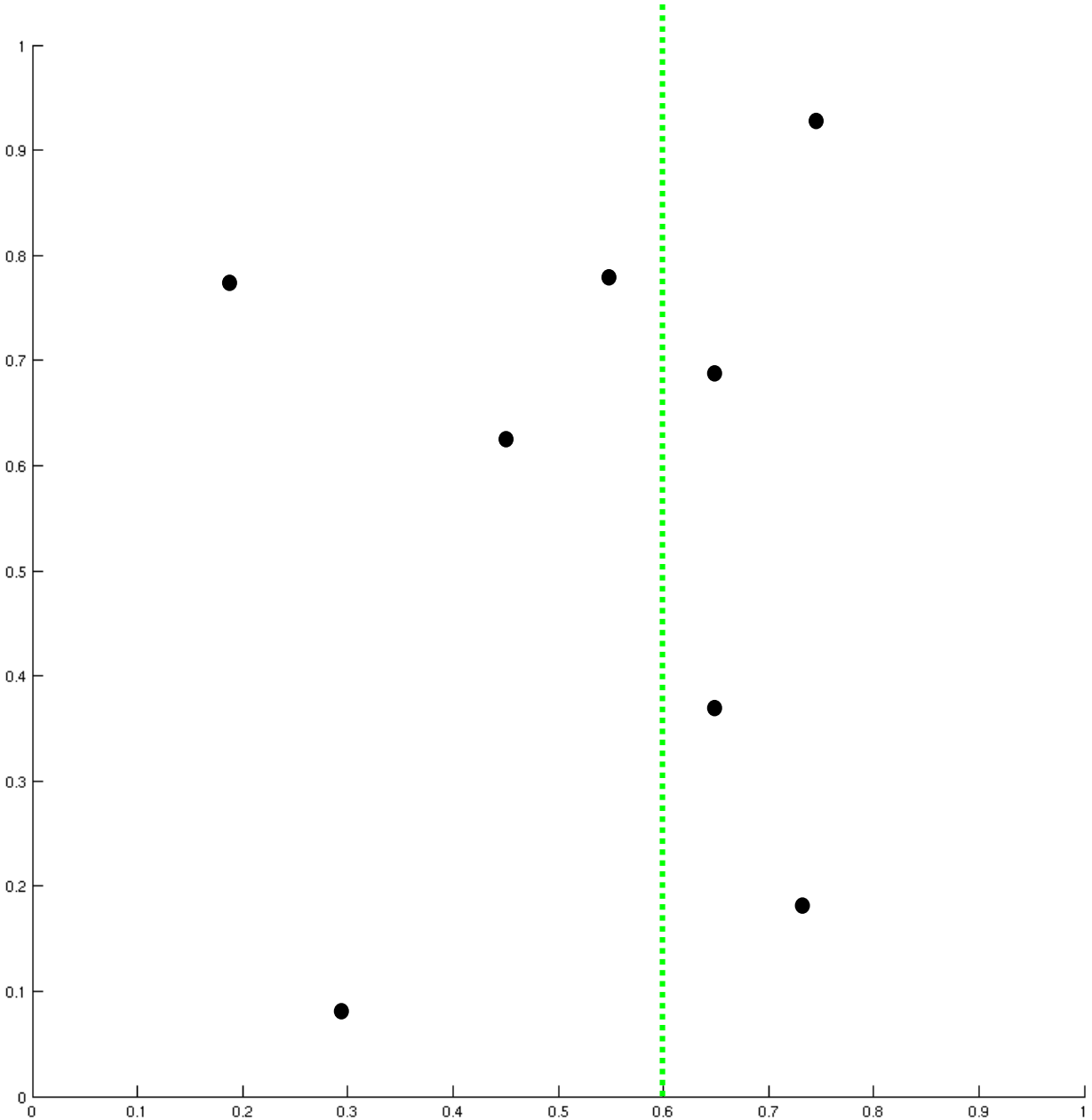
Zpracované hrany

Hrany v AFL_{α}

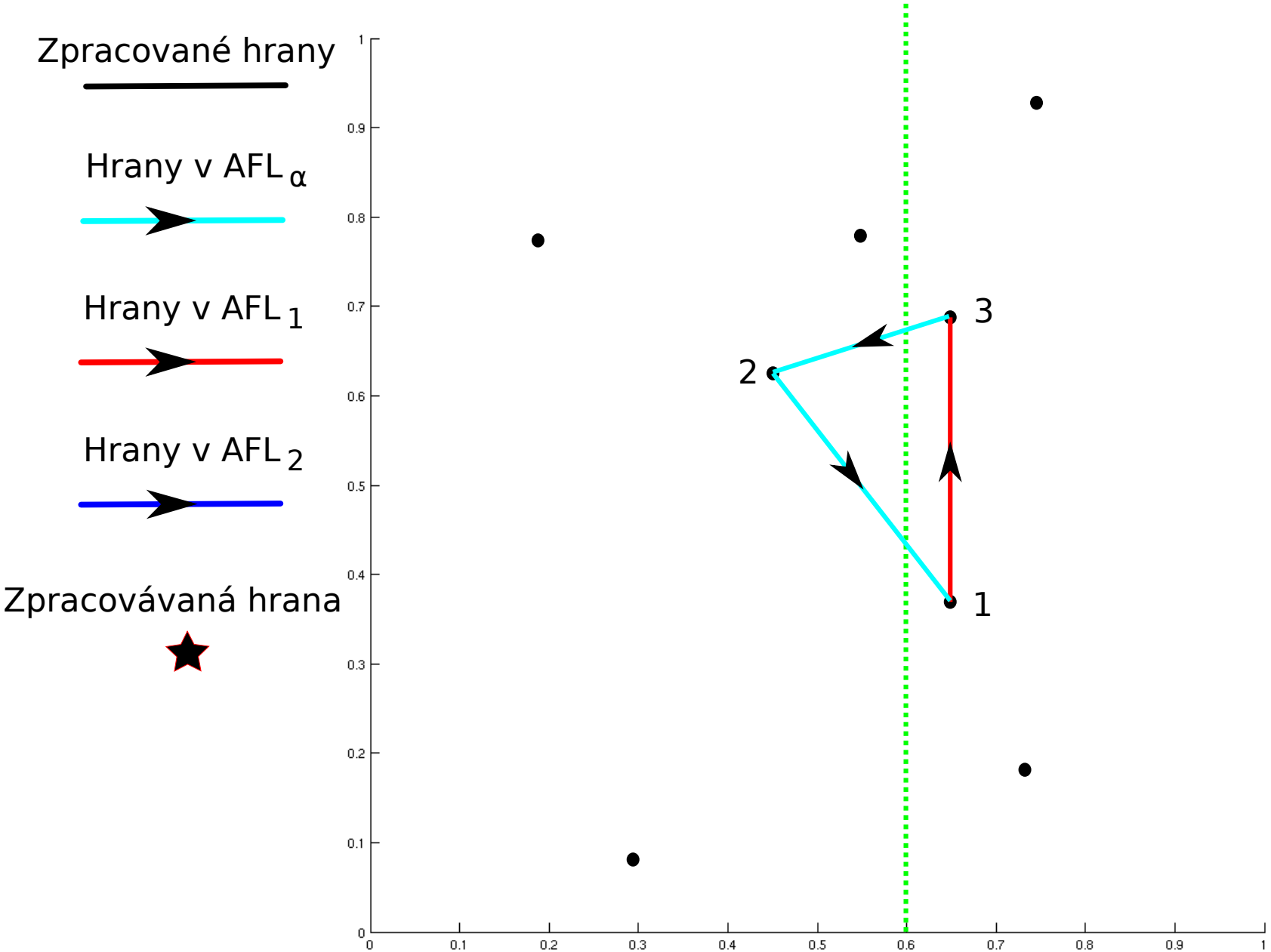
Hrany v AFL_1

Hrany v AFL_2

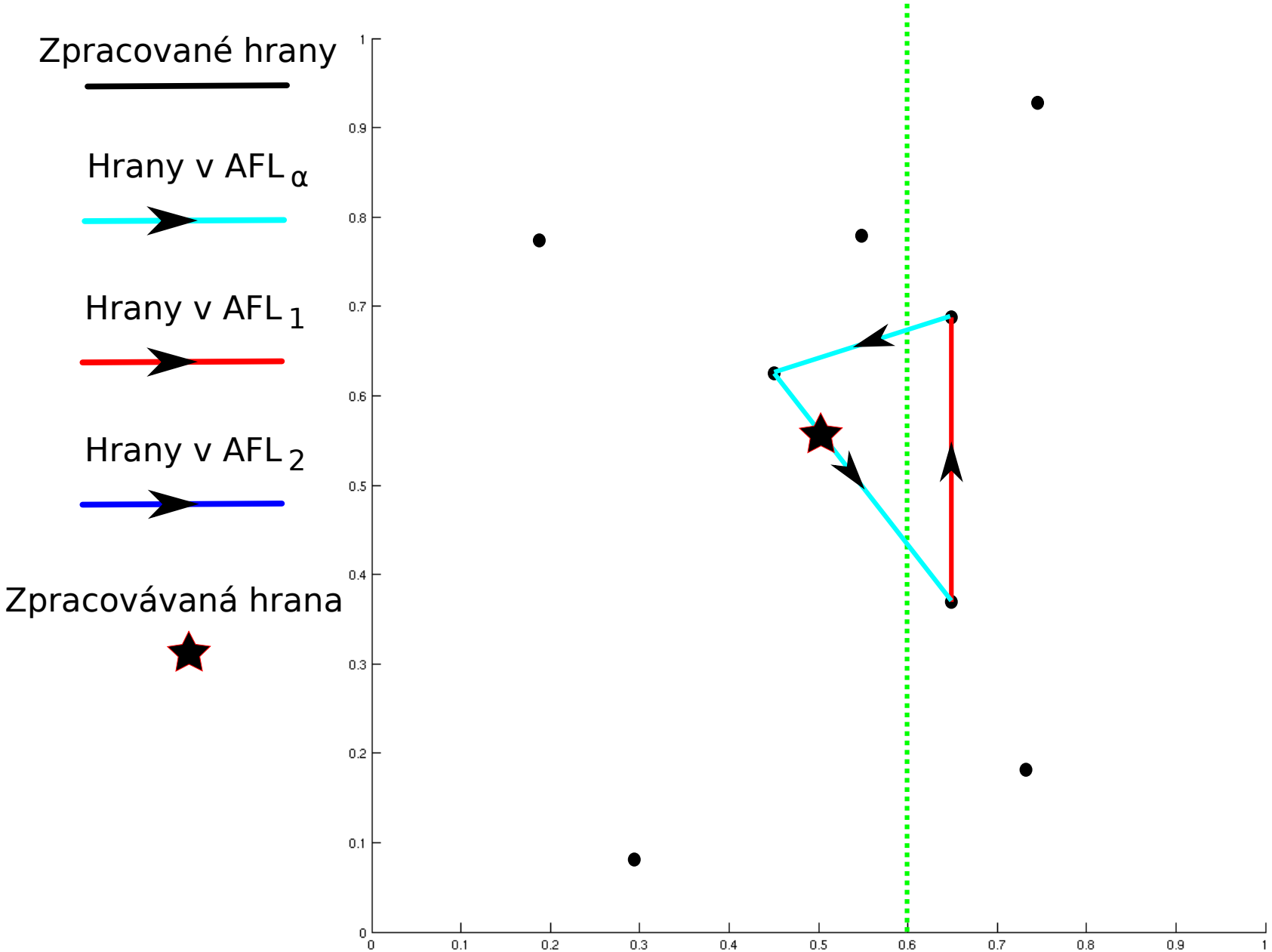
Zpracovávaná hrana



Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
1
2
3
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

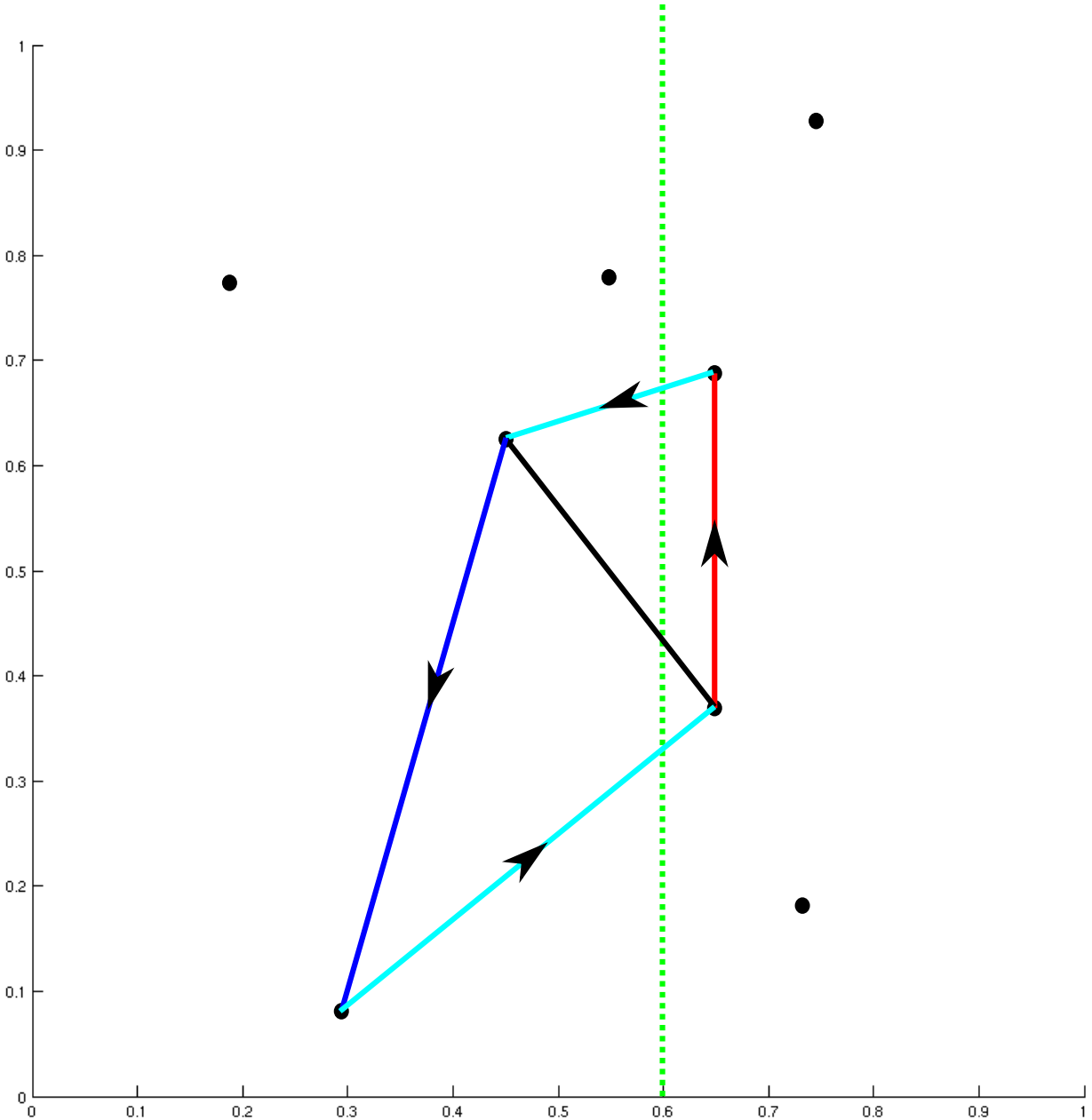
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

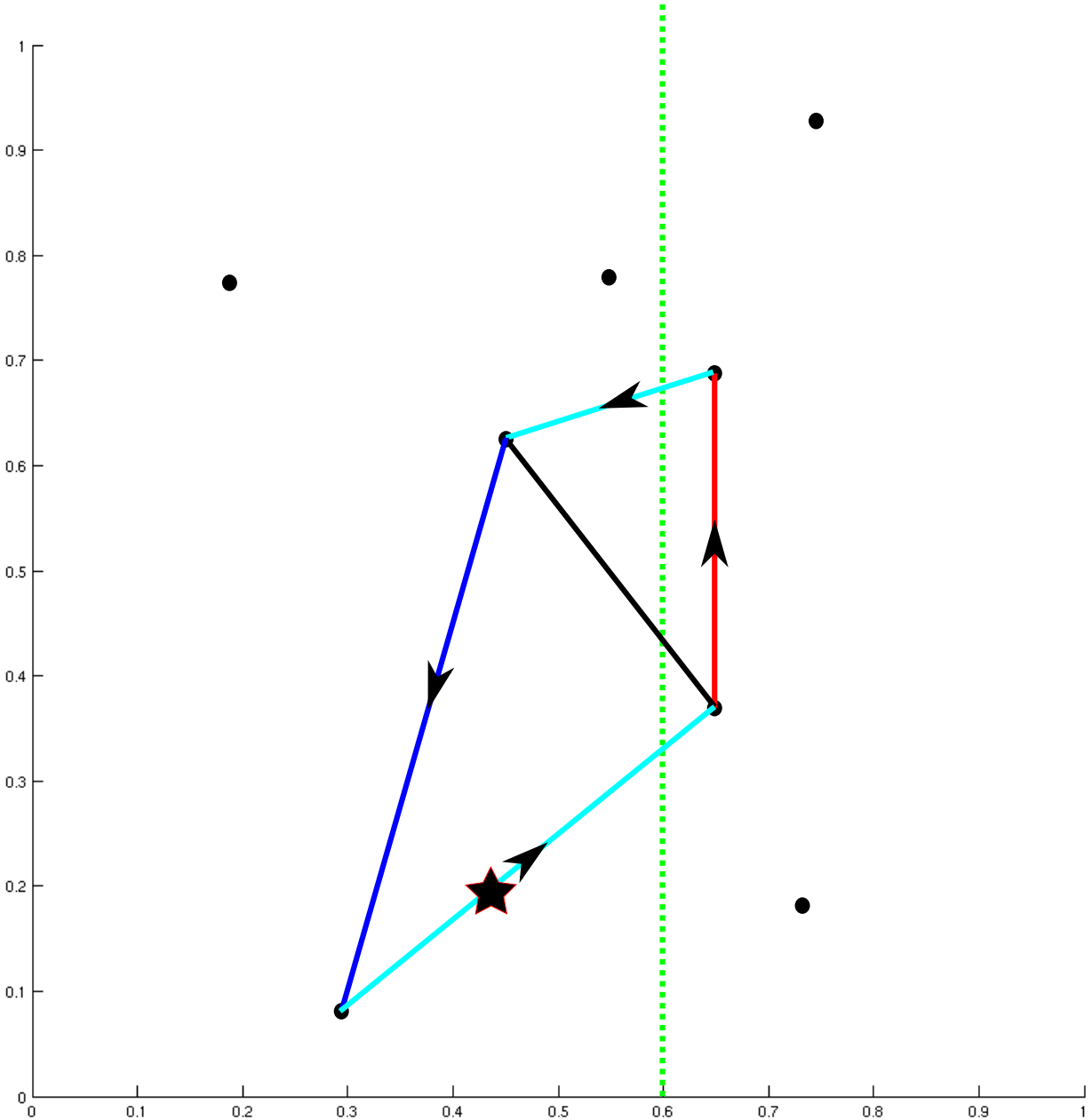
Zpracované hrany

Hrany v AFL_α

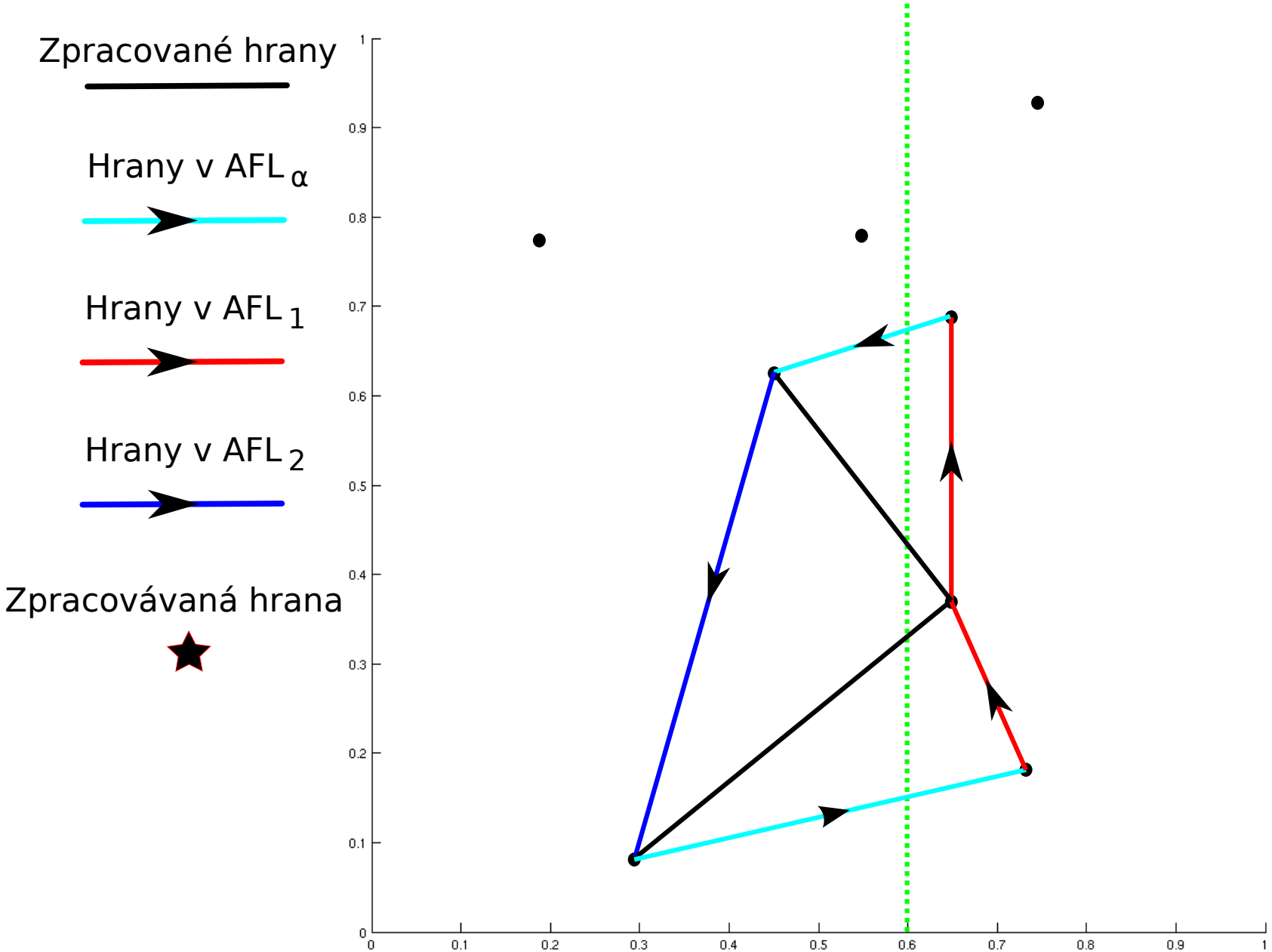
Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

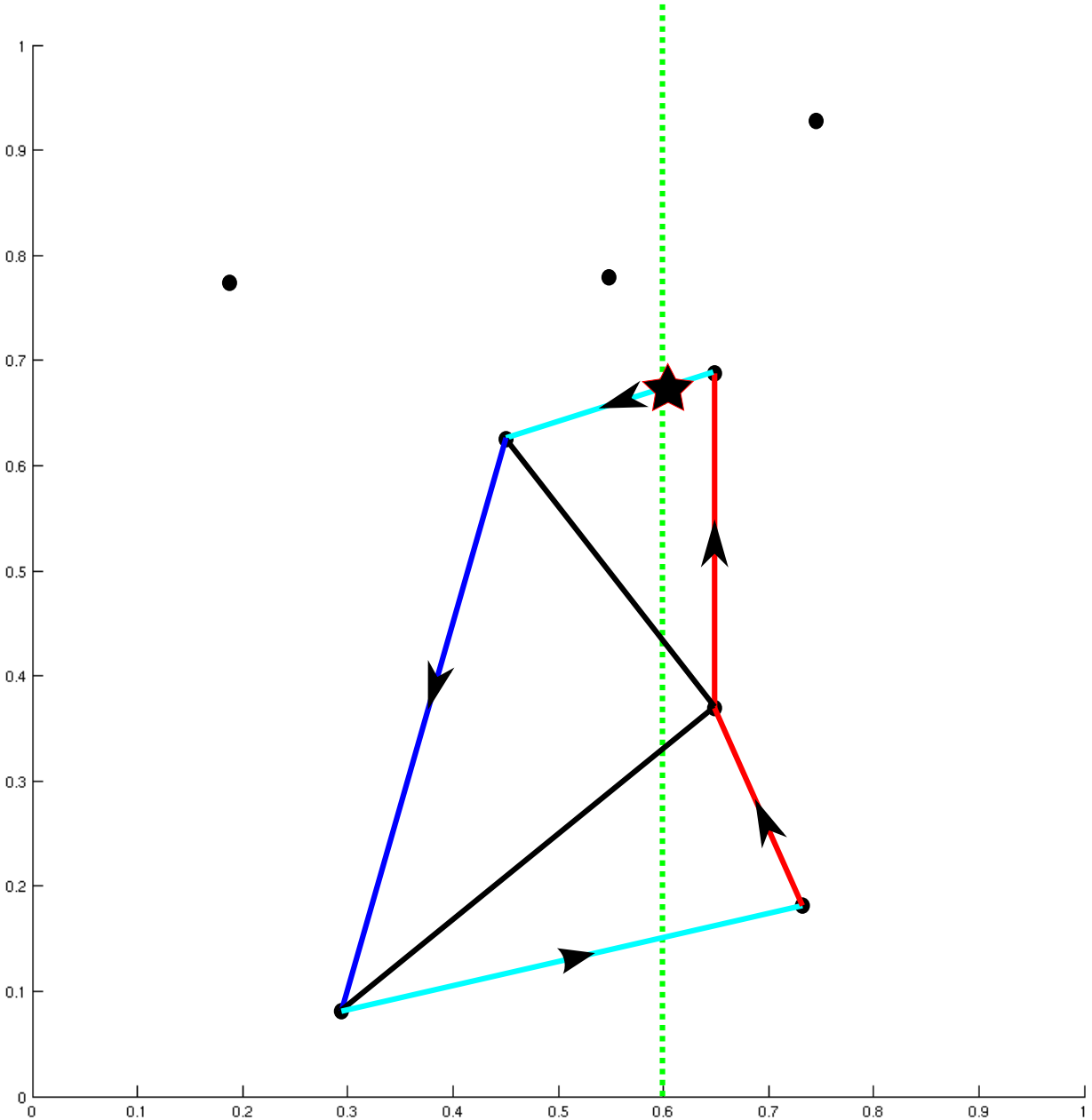
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

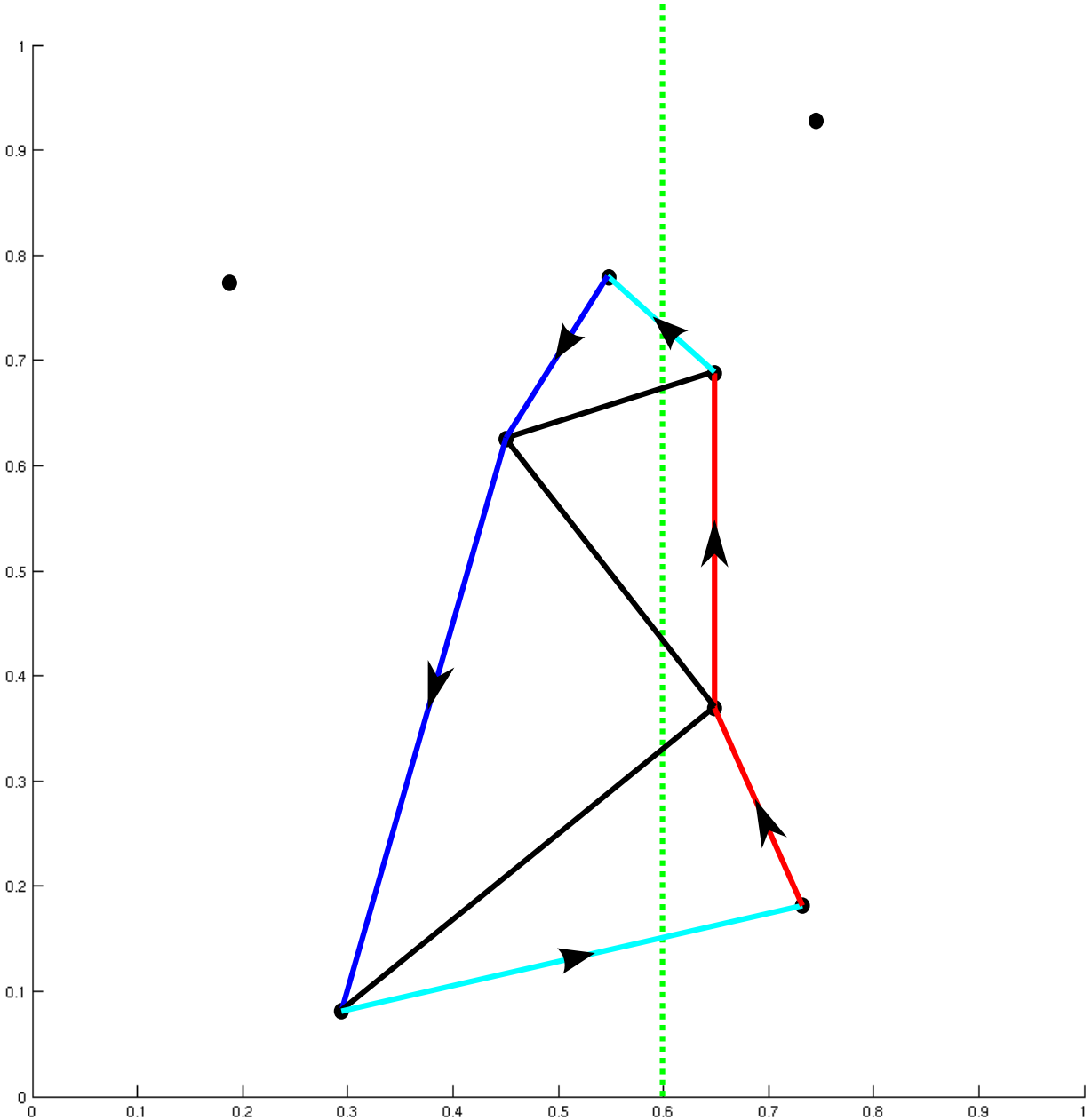
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

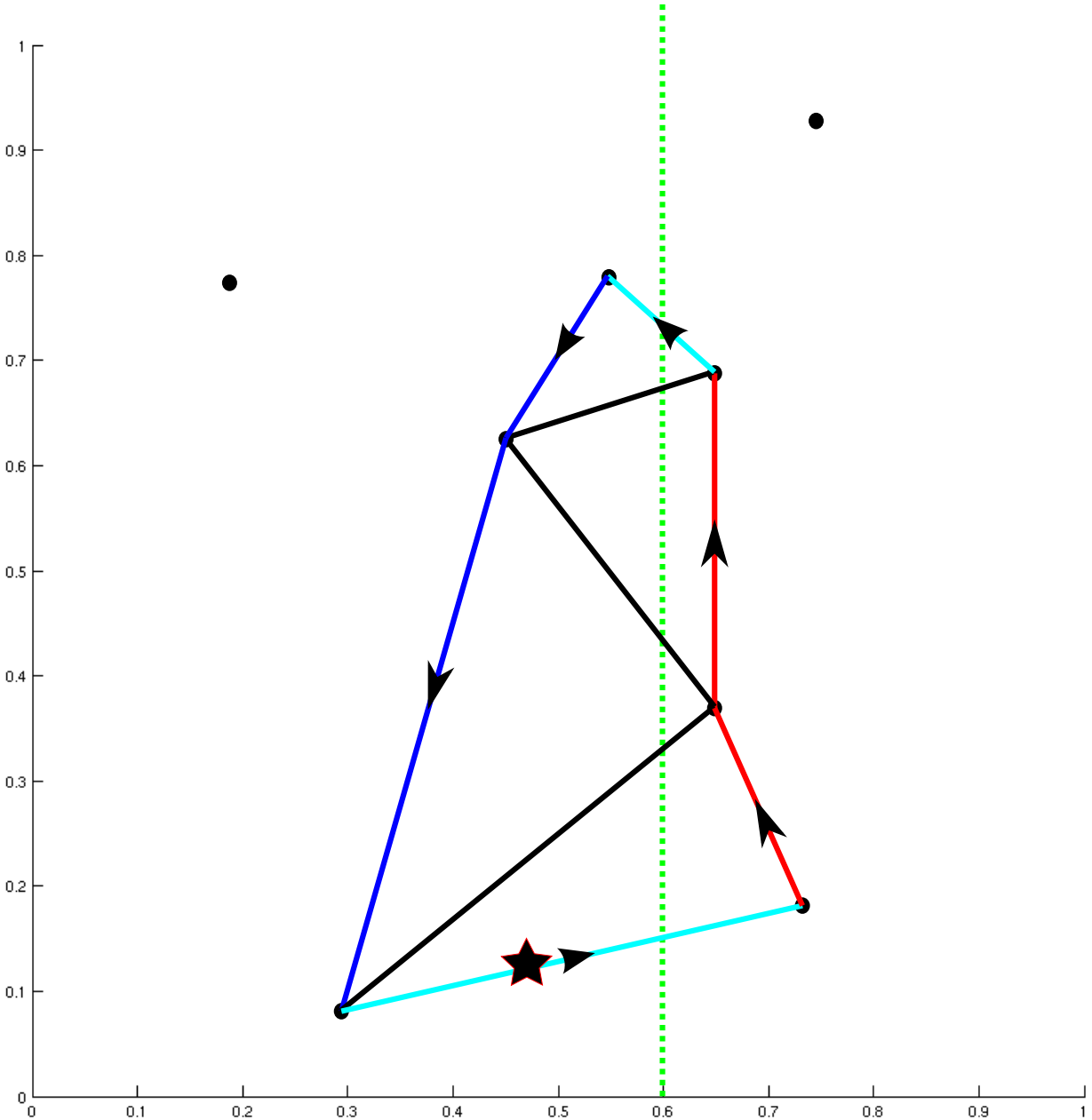
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

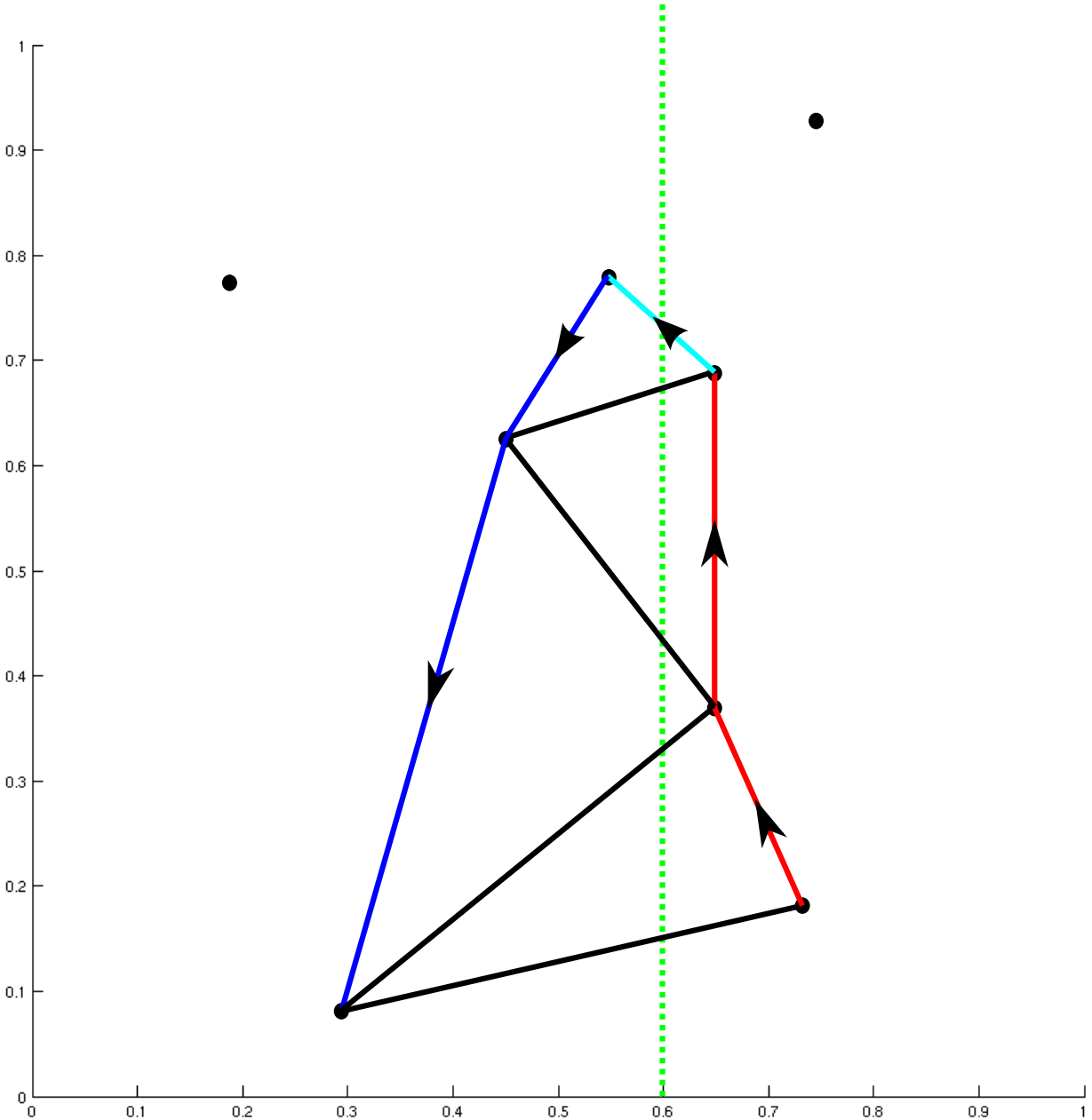
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

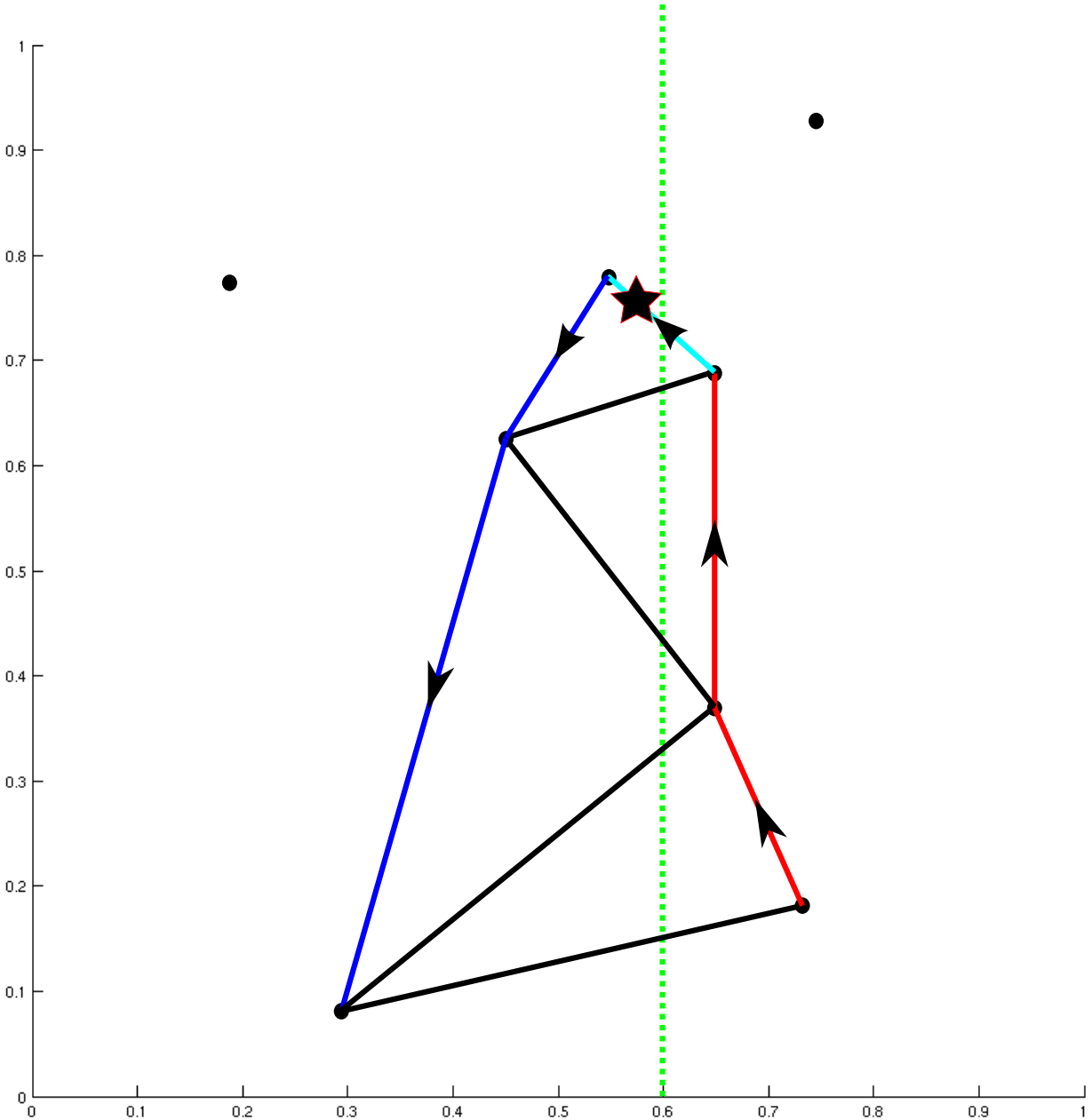
Zpracované hrany

Hrany v AFL_α

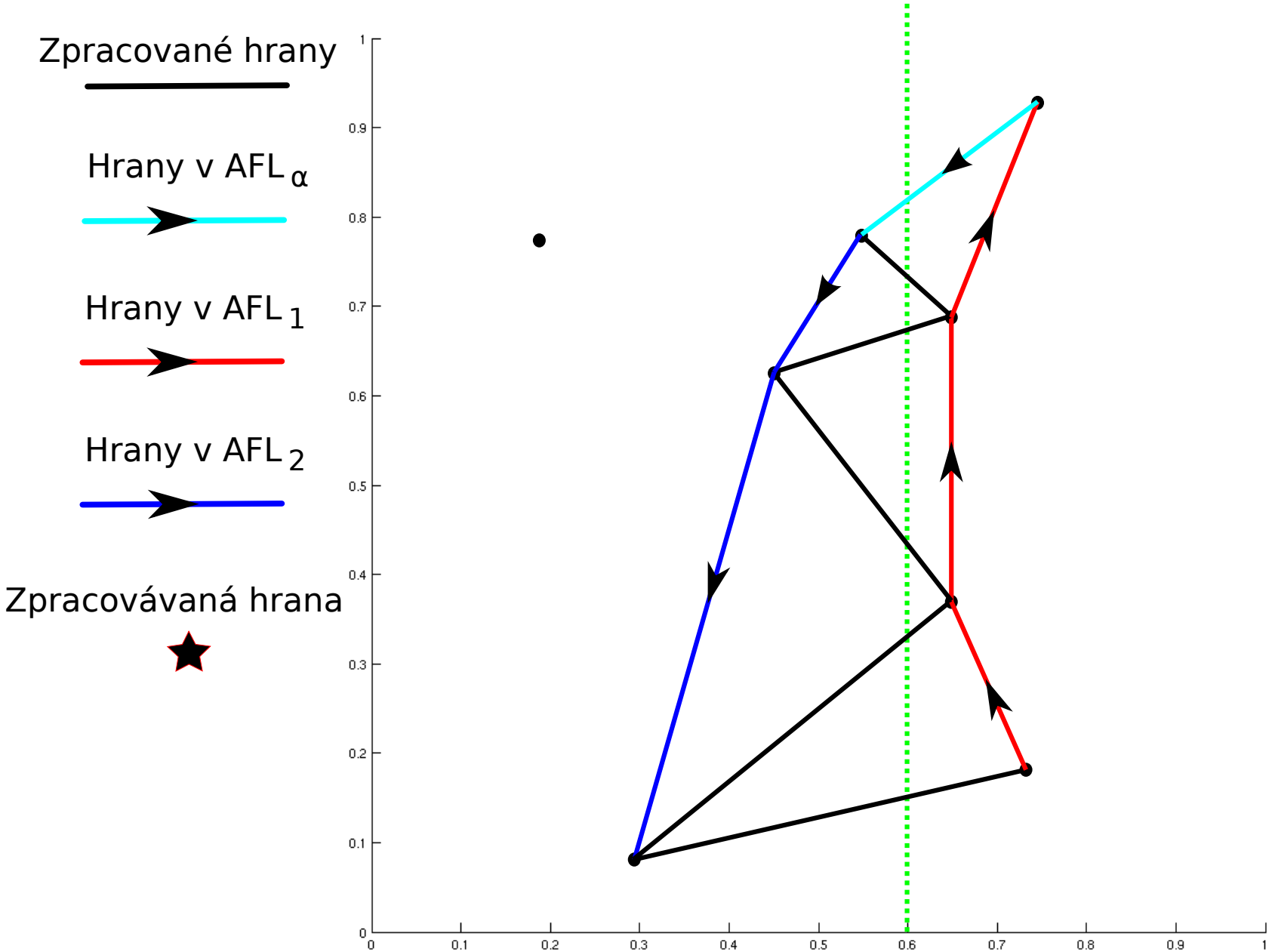
Hrany v AFL_1

Hrany v AFL_2

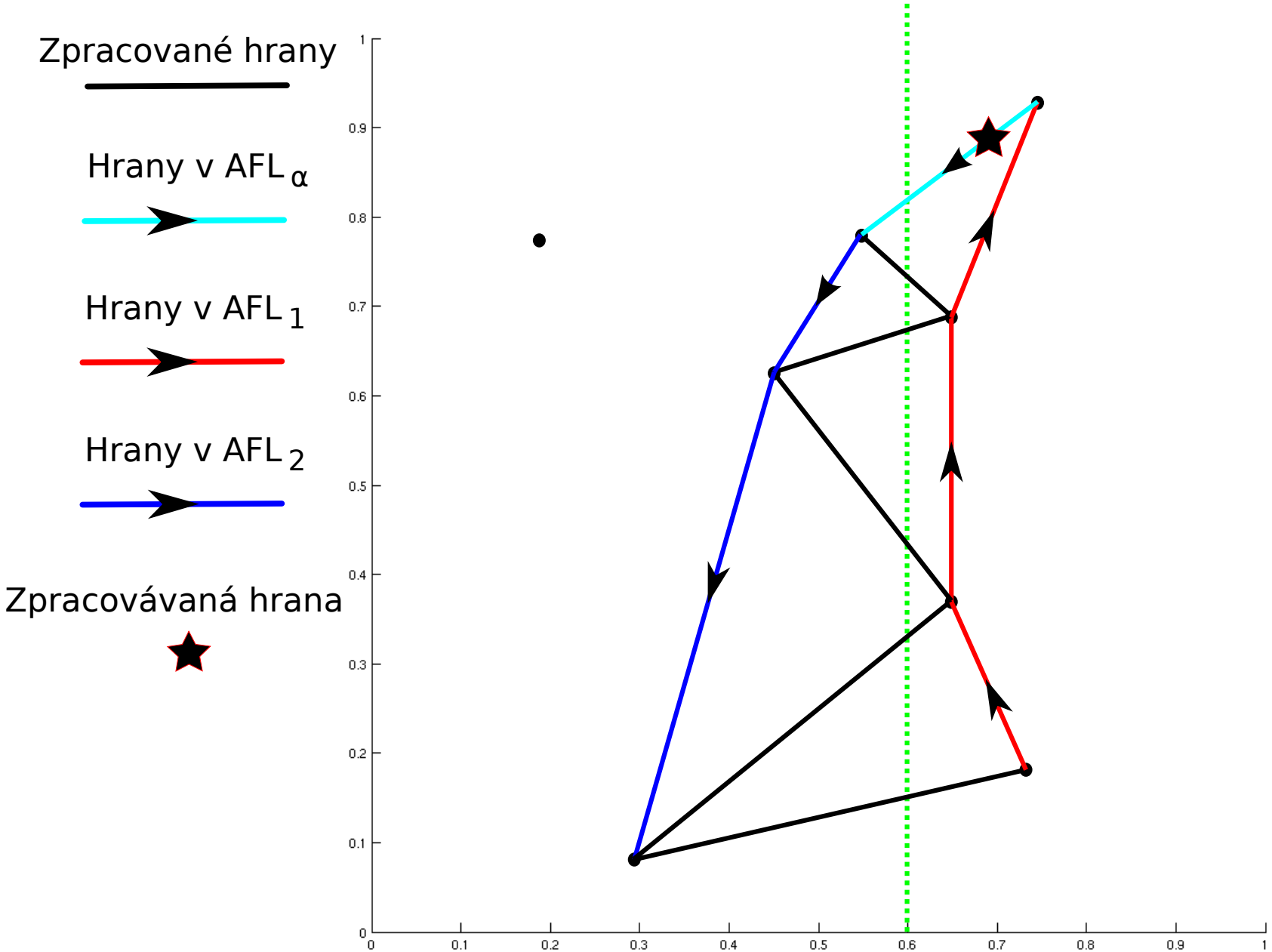
Zpracovávaná hrana



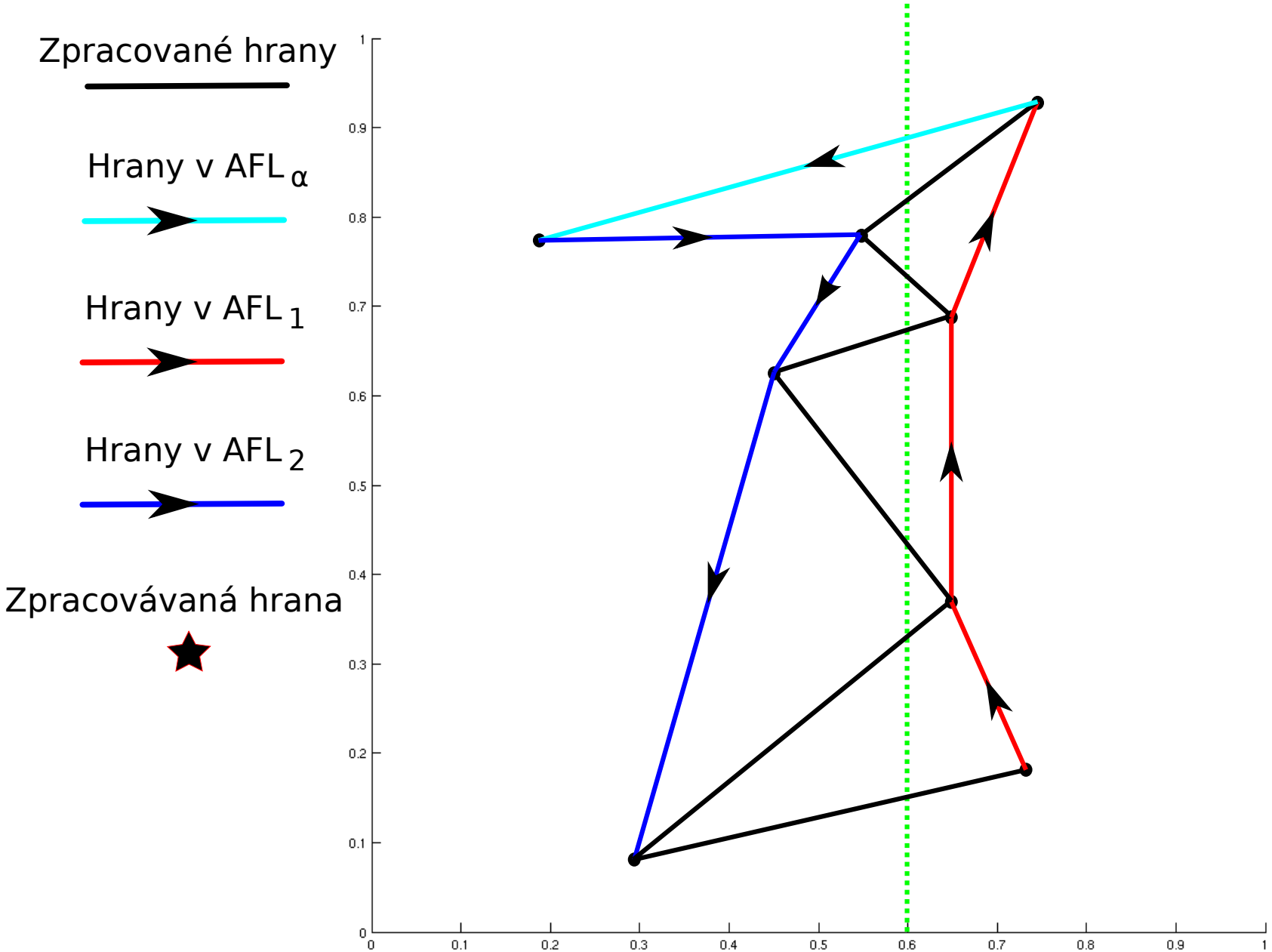
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

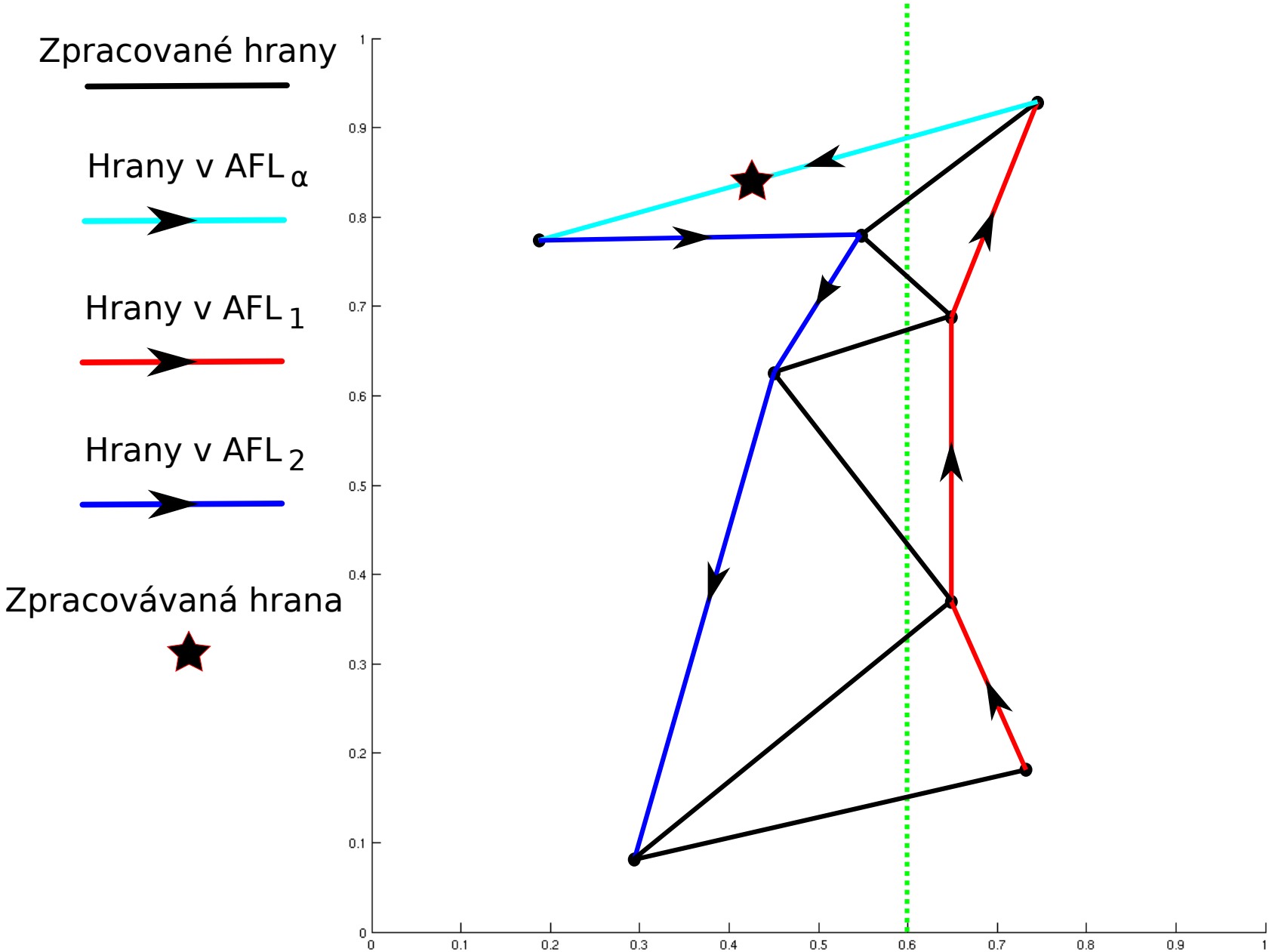
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

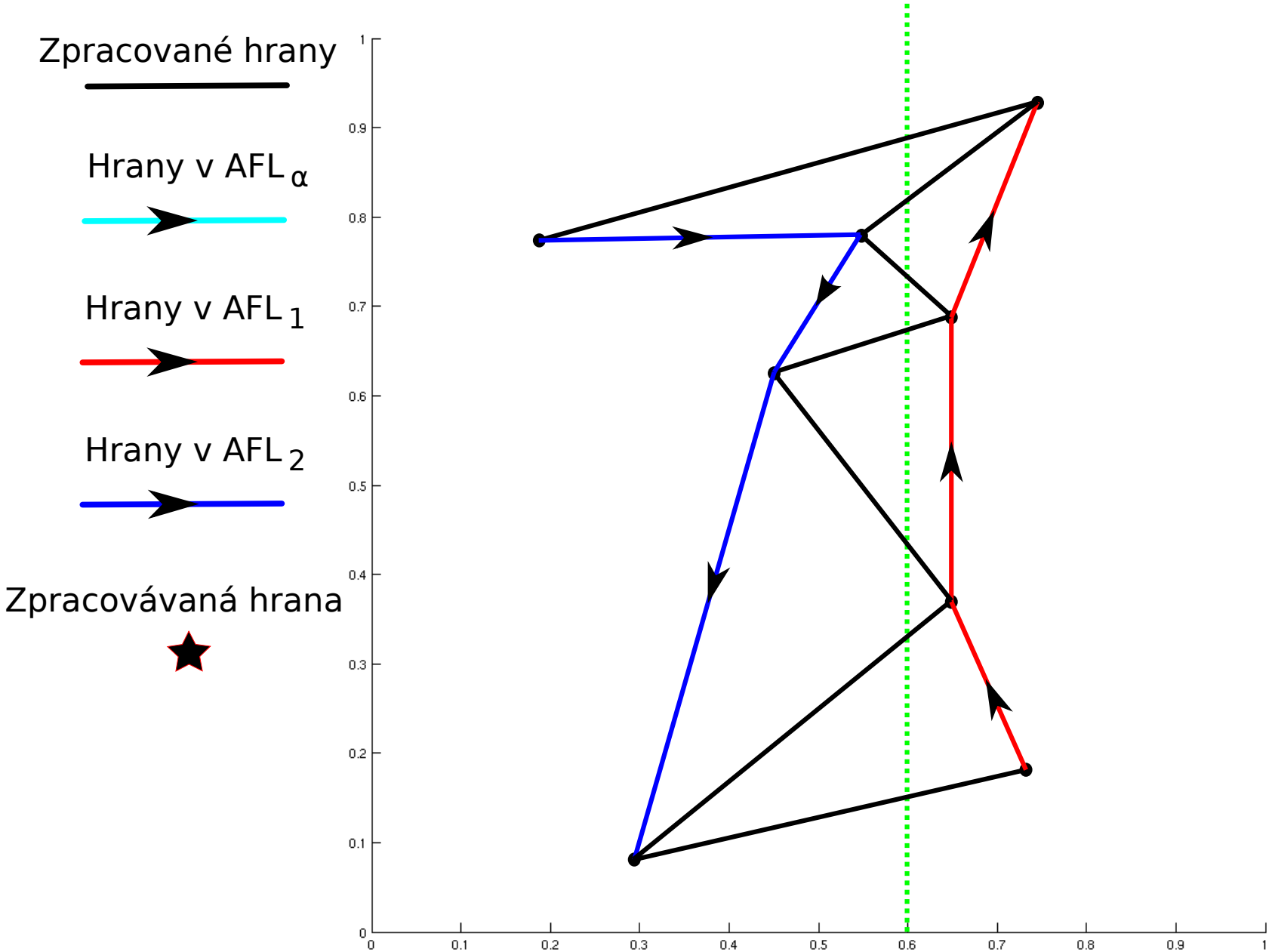
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

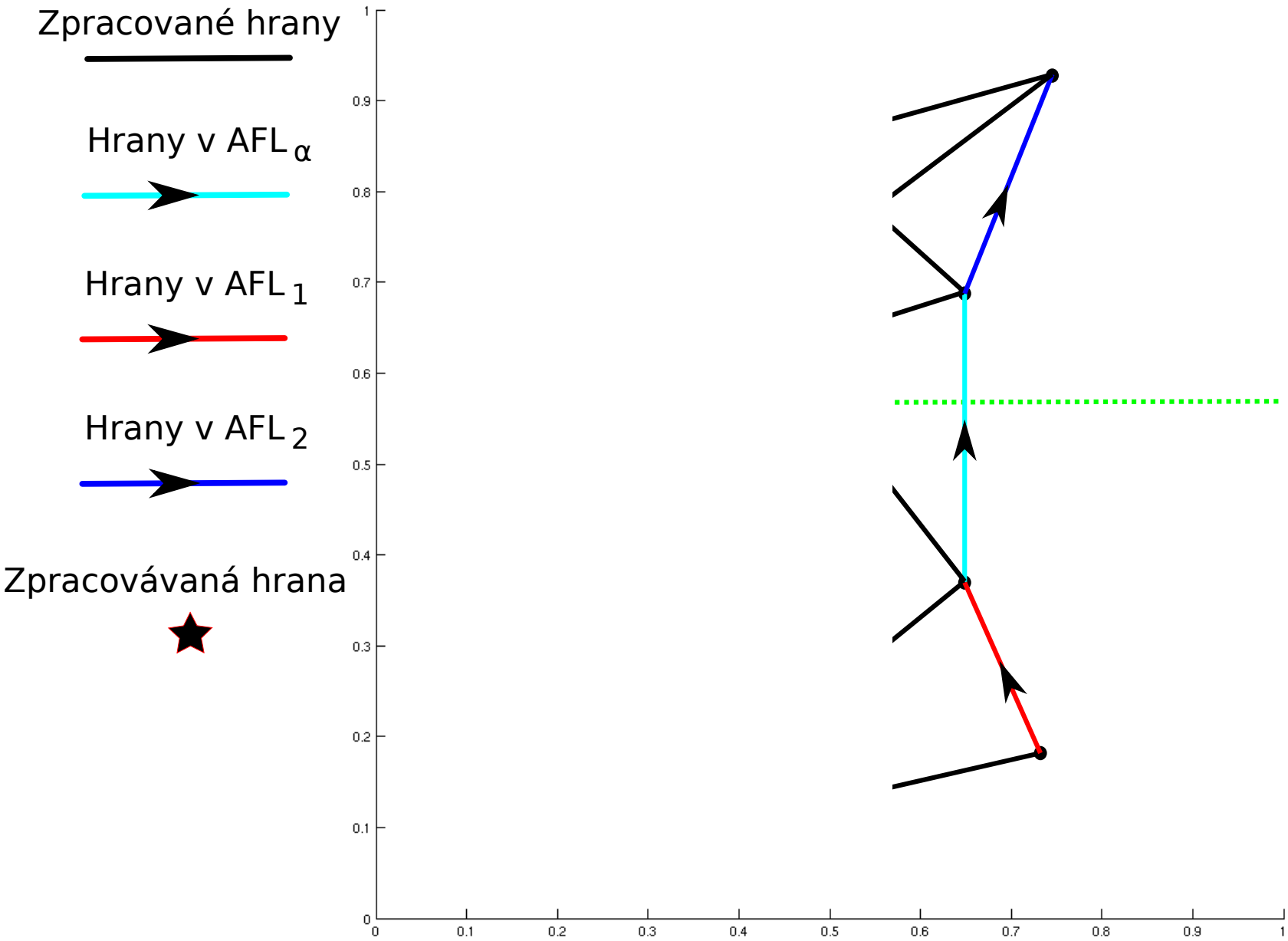
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

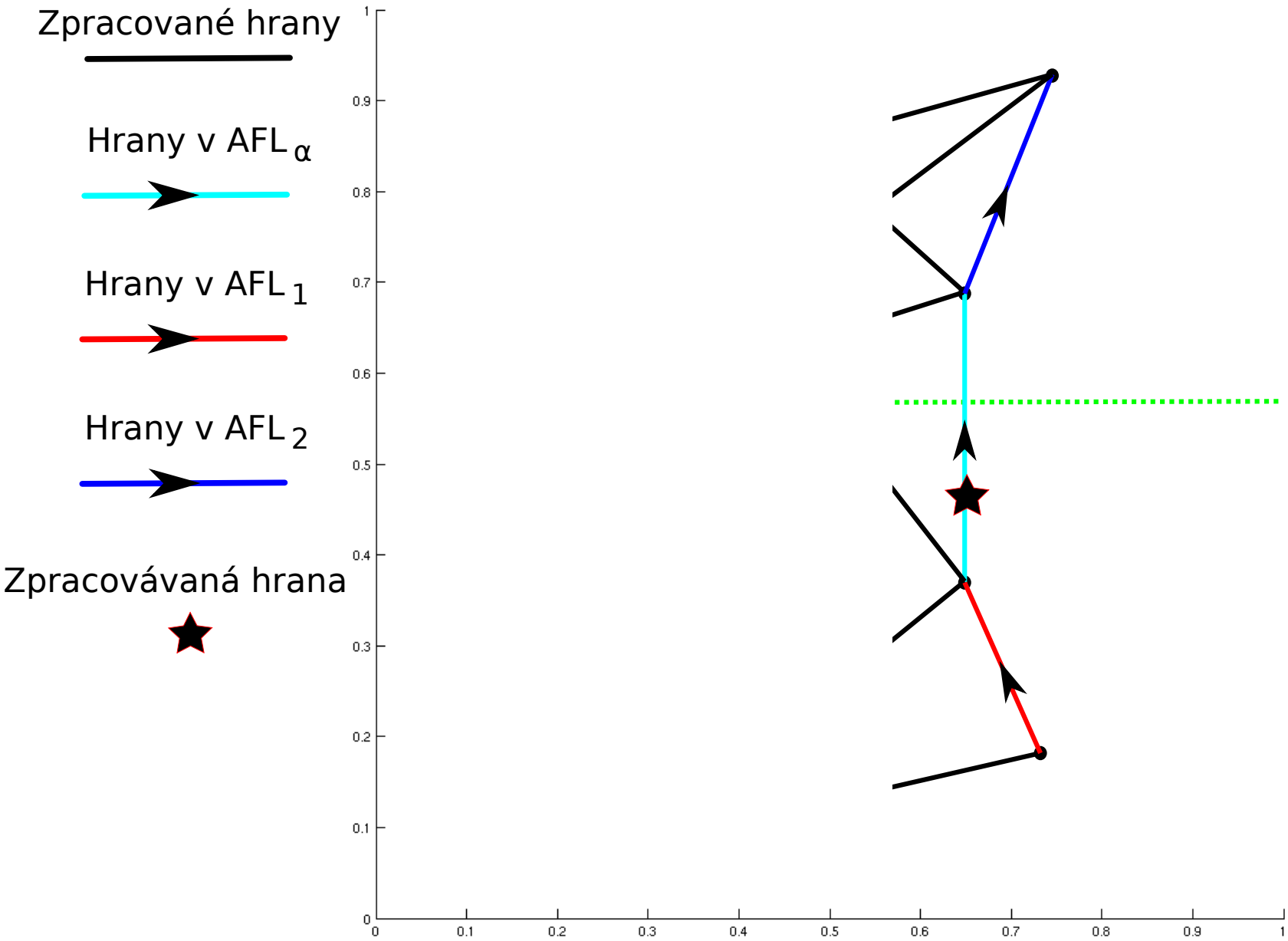
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

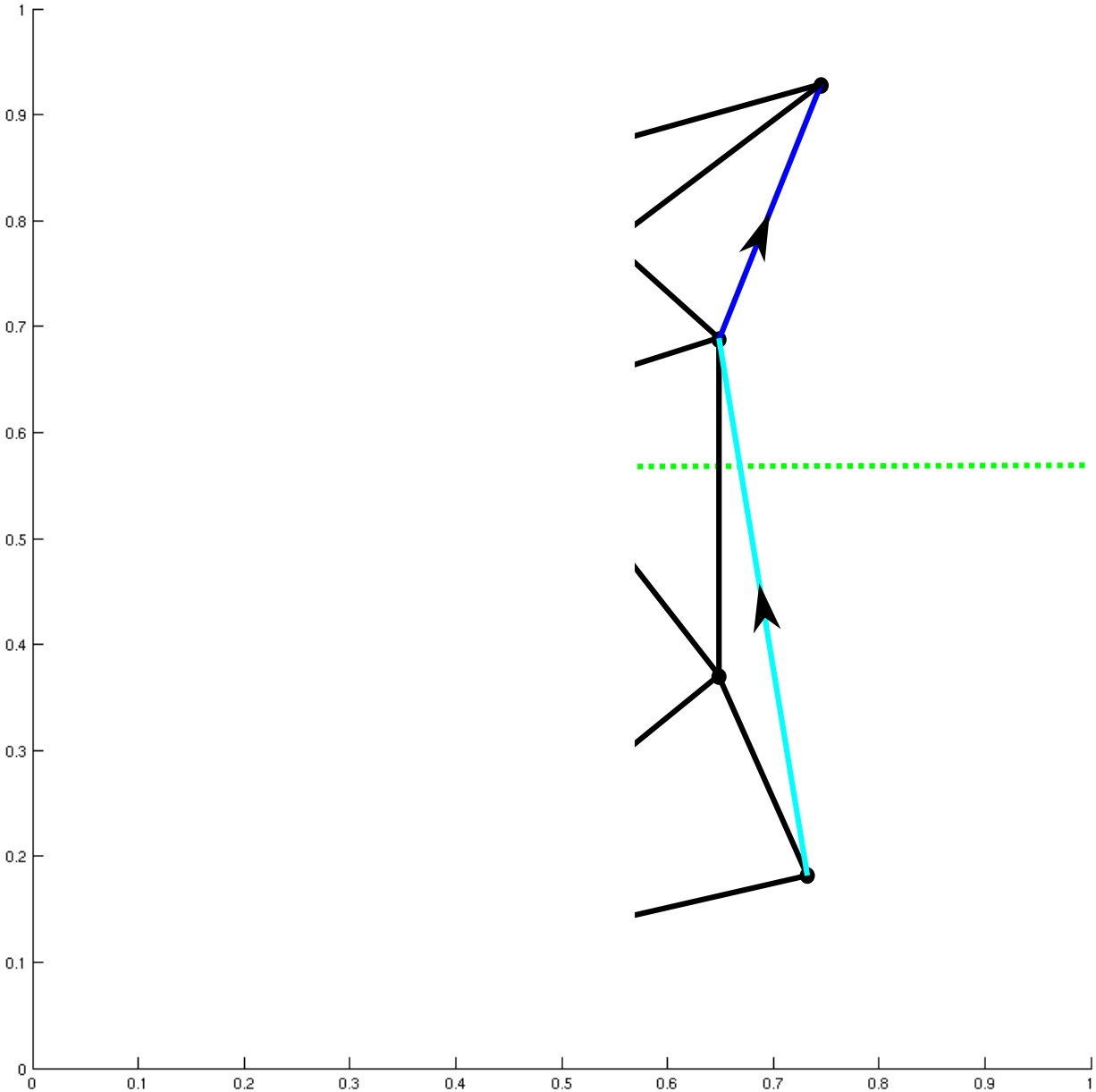
Zpracované hrany

Hrany v AFL $_\alpha$

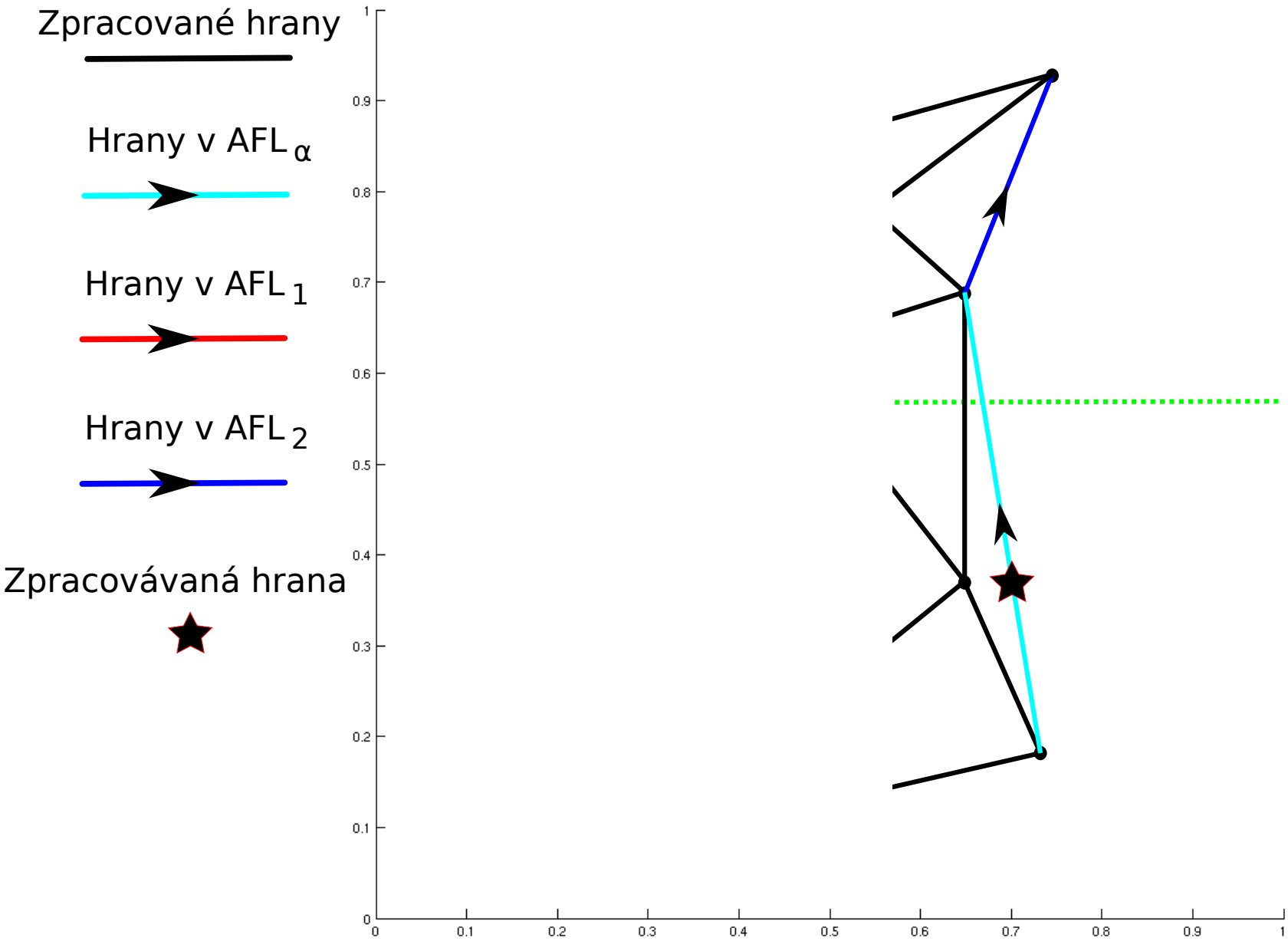
Hrany v AFL $_1$

Hrany v AFL $_2$

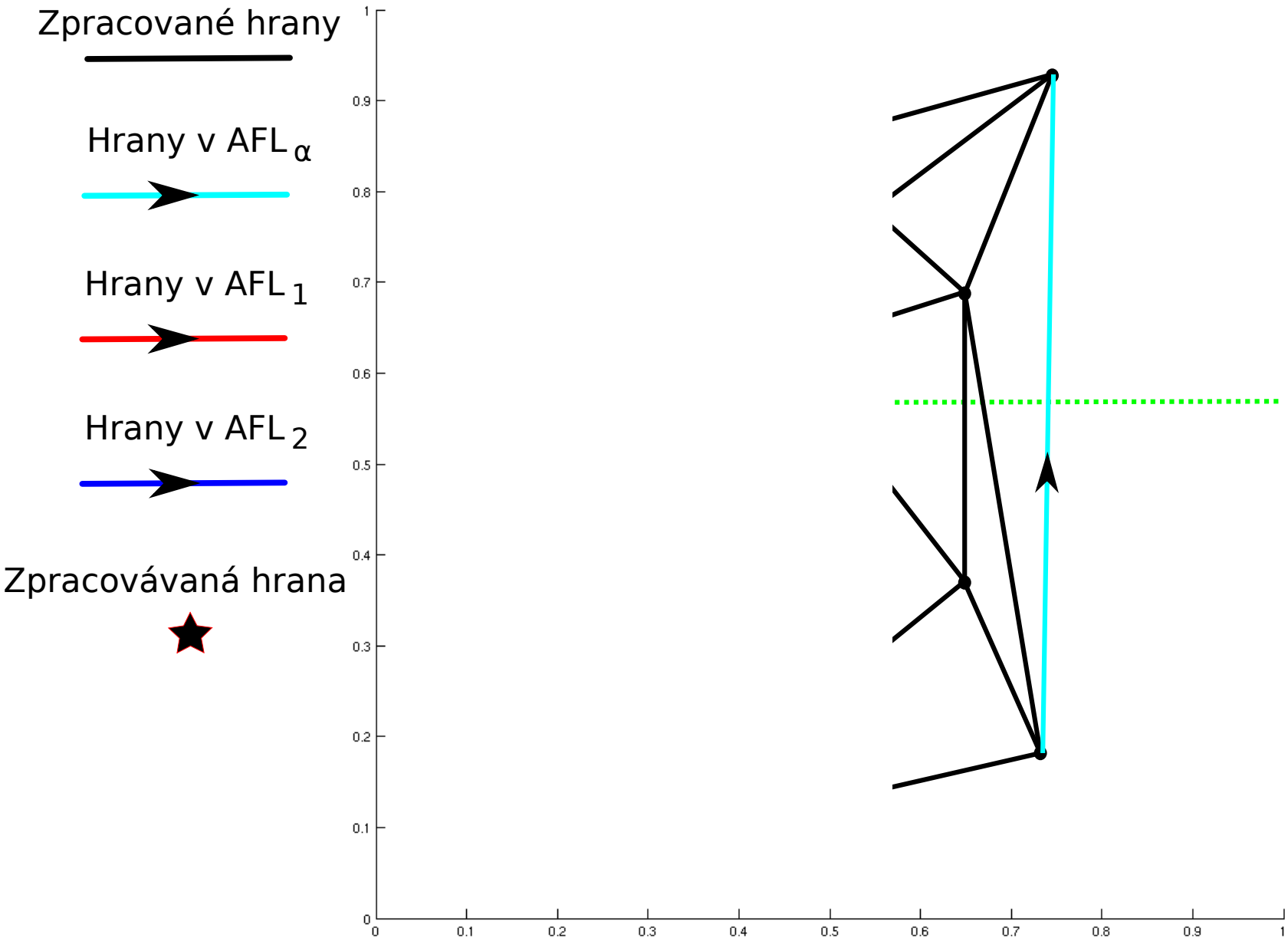
Zpracovávaná hrana



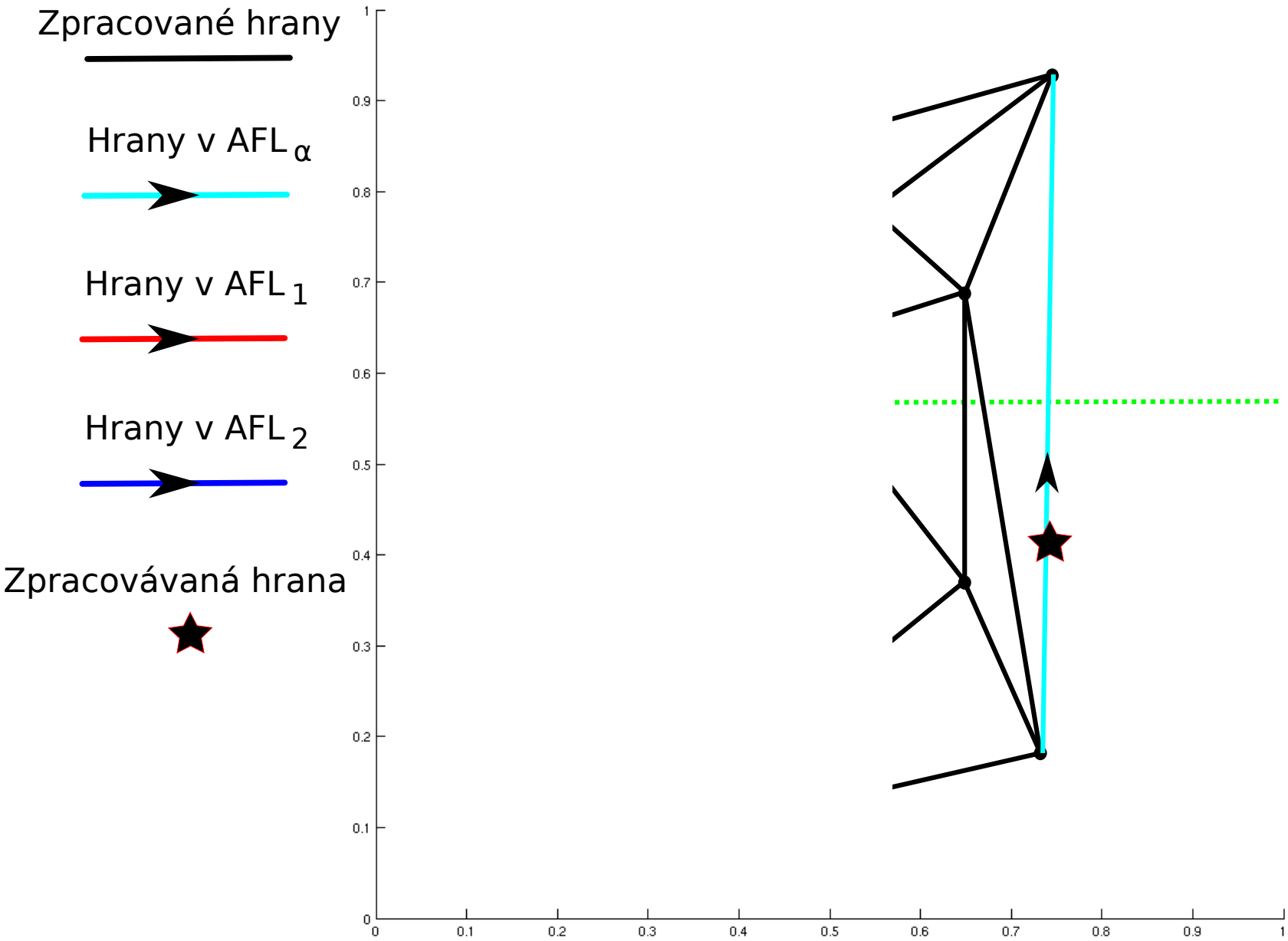
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

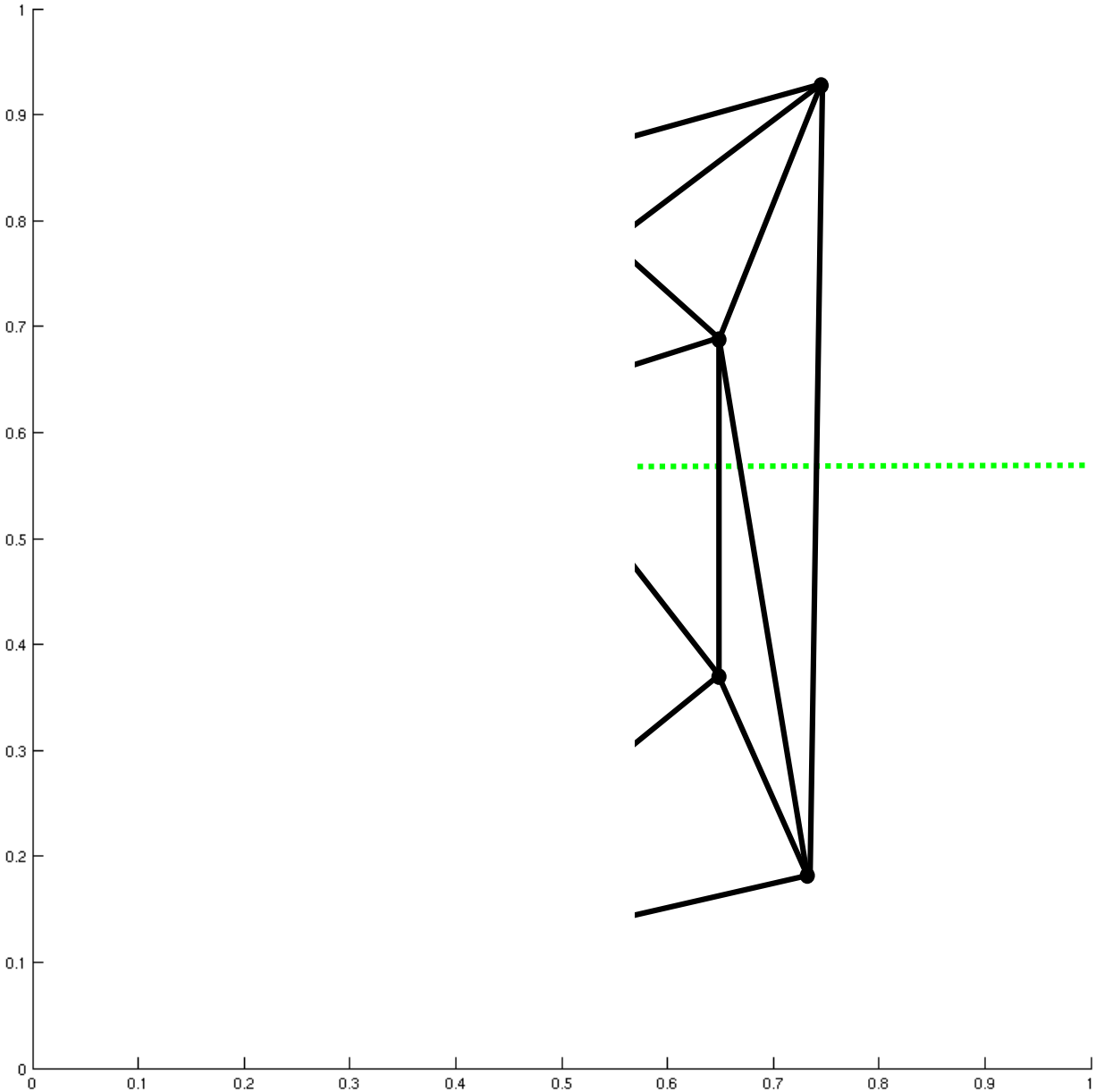
Zpracované hrany

Hrany v AFL_α

Hrany v AFL_1

Hrany v AFL_2

Zpracovávaná hrana



Příklad

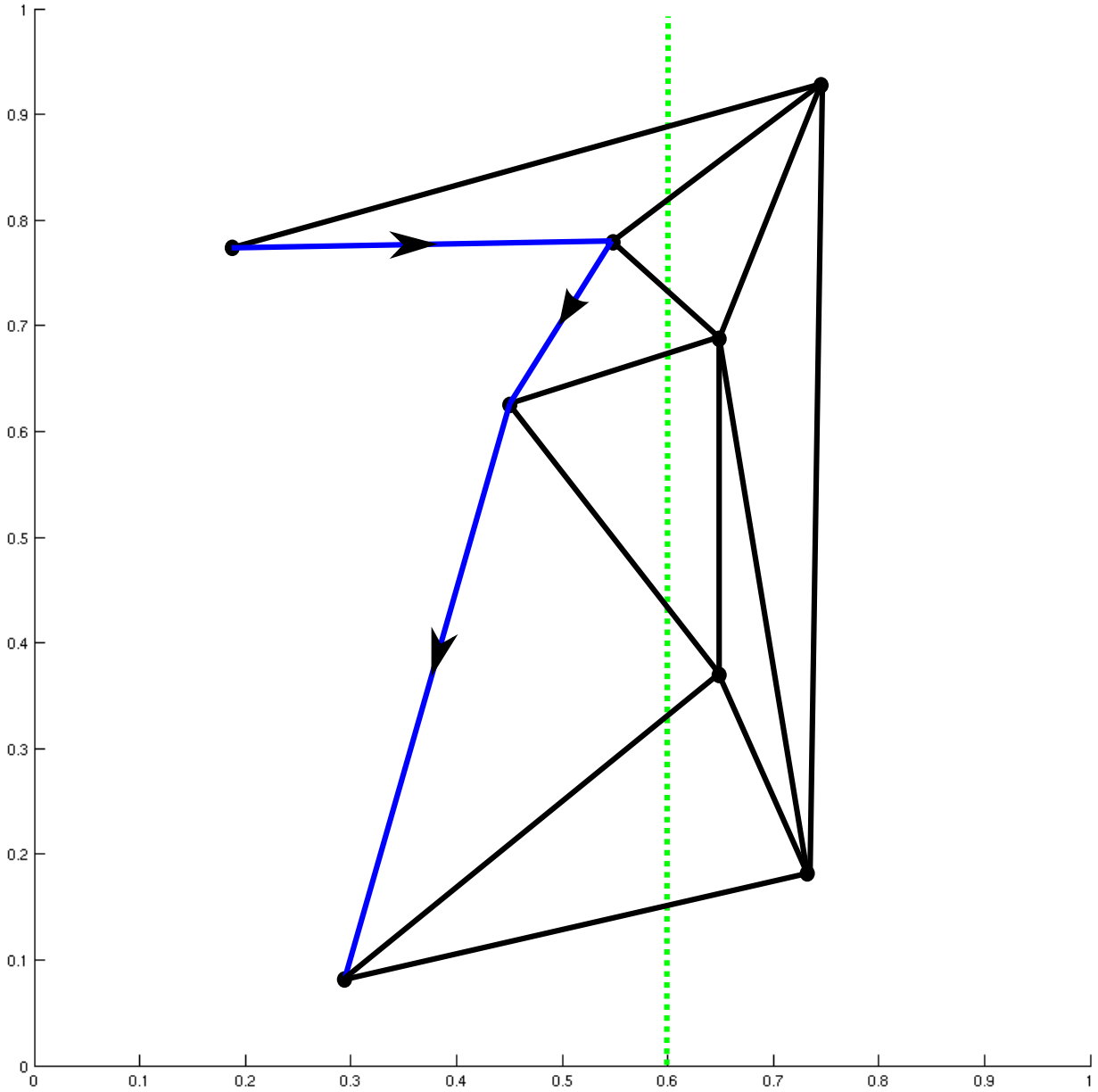
Zpracované hrany

Hrany v AFL_α

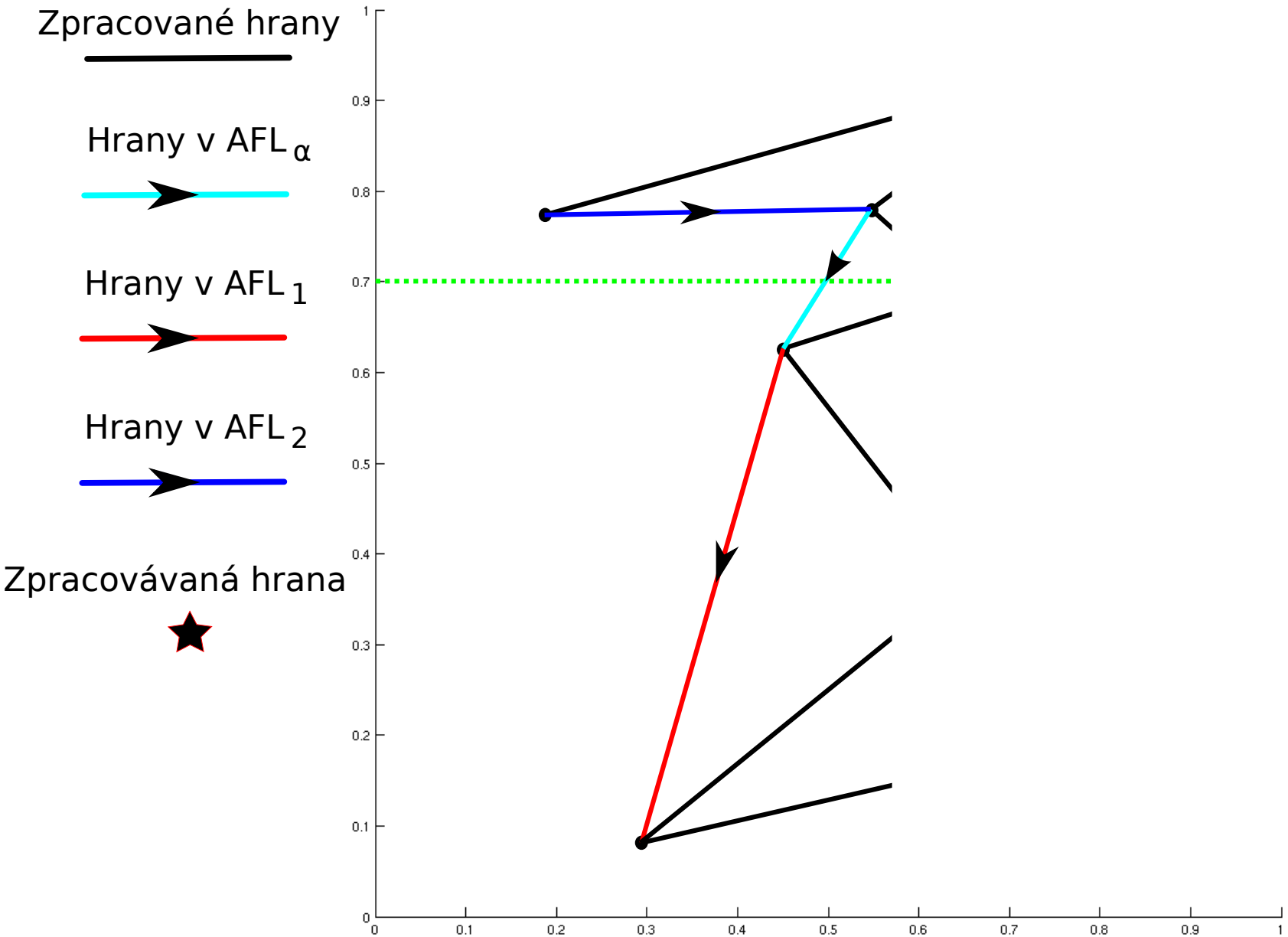
Hrany v AFL_1

Hrany v AFL_2

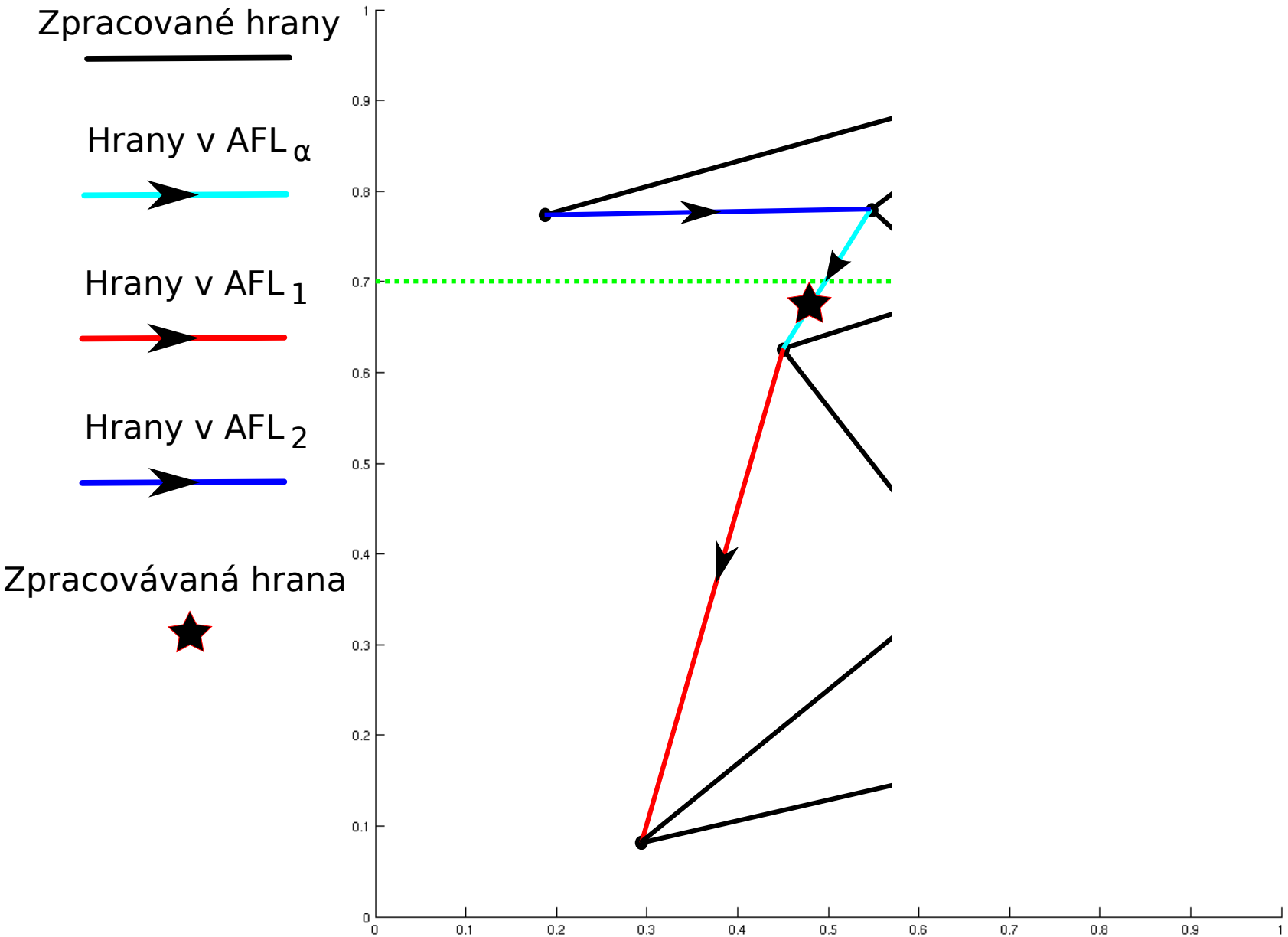
Zpracovávaná hrana



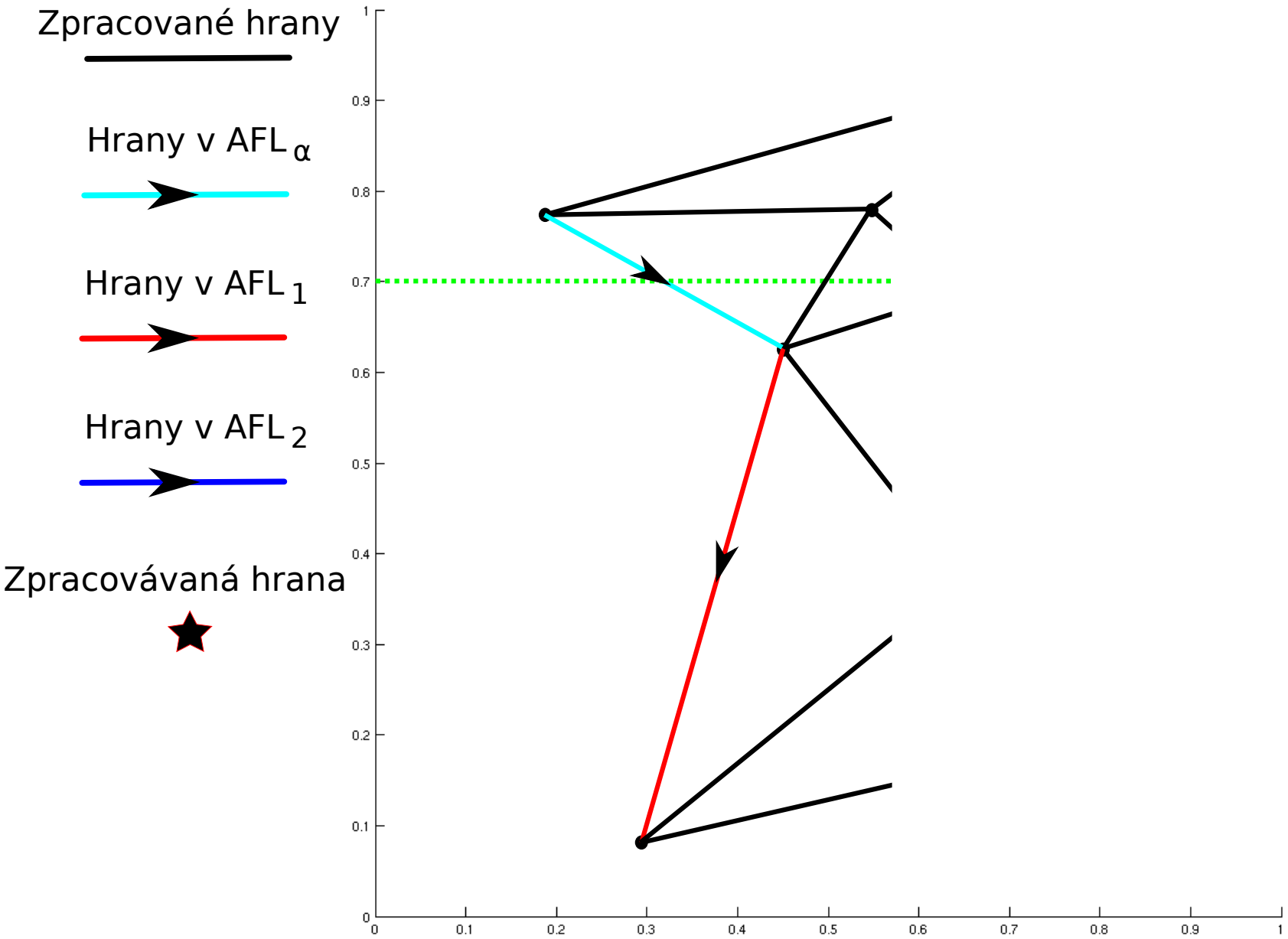
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

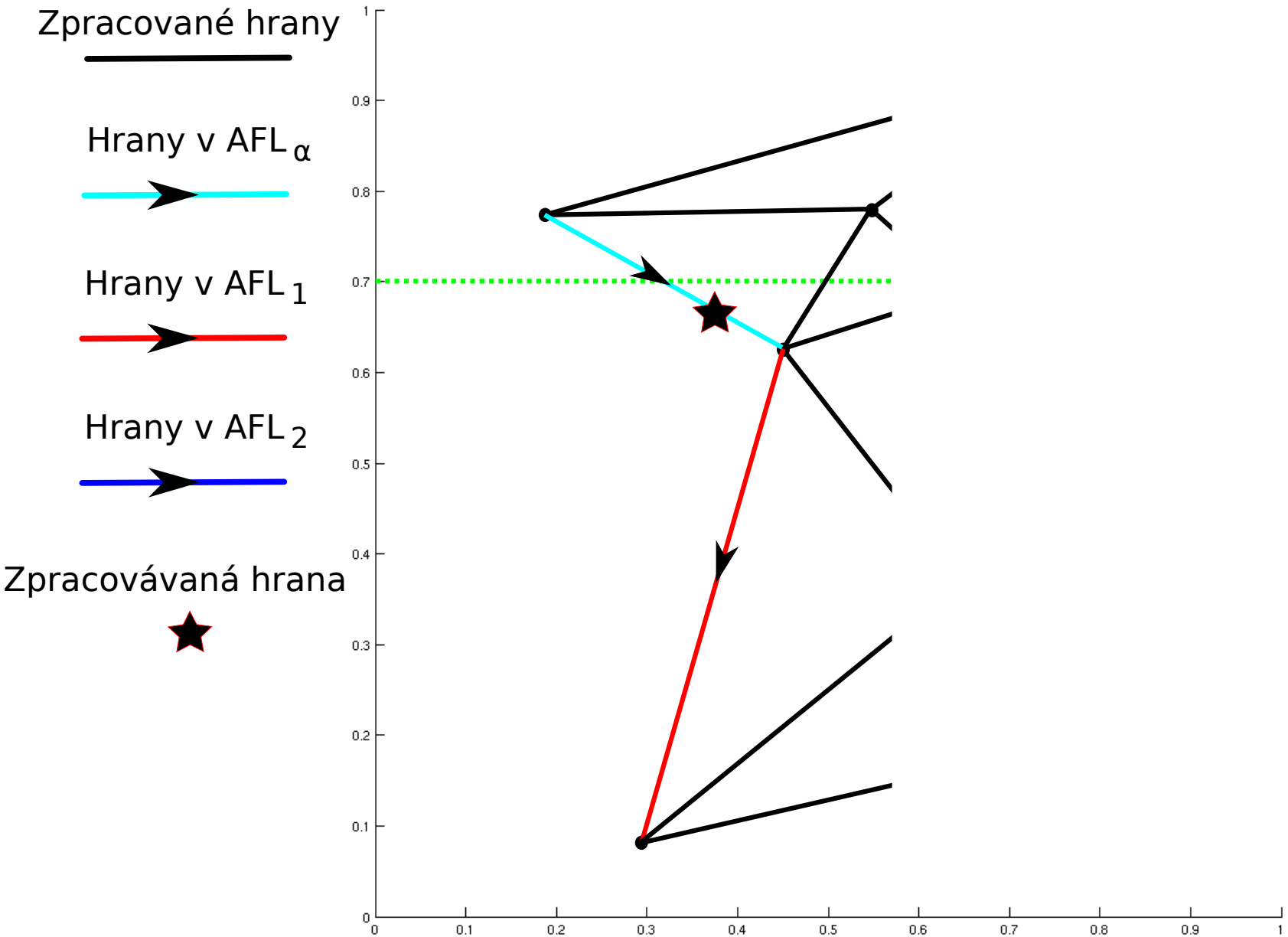
Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

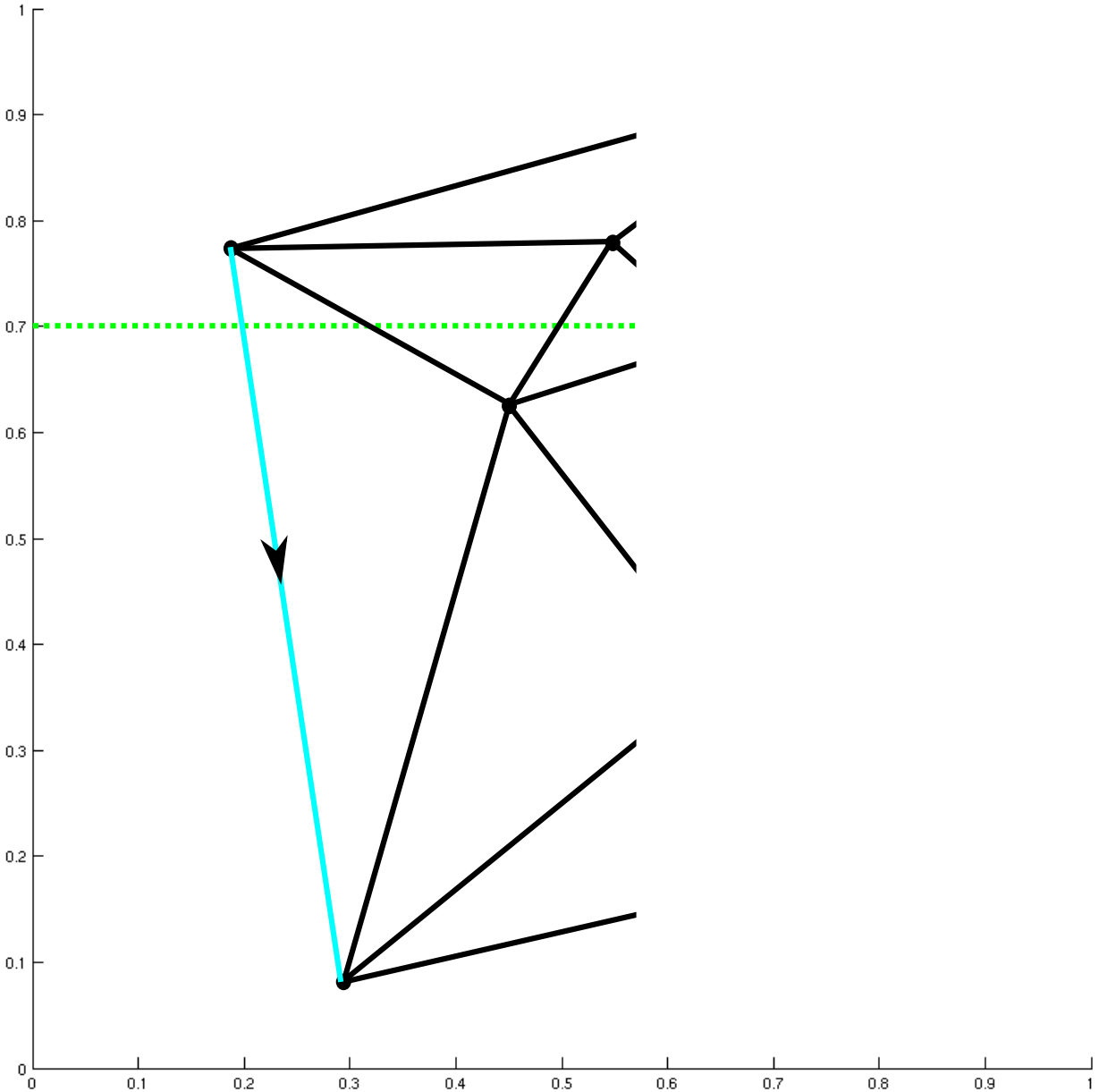
Zpracované hrany

Hrany v AFL $_\alpha$

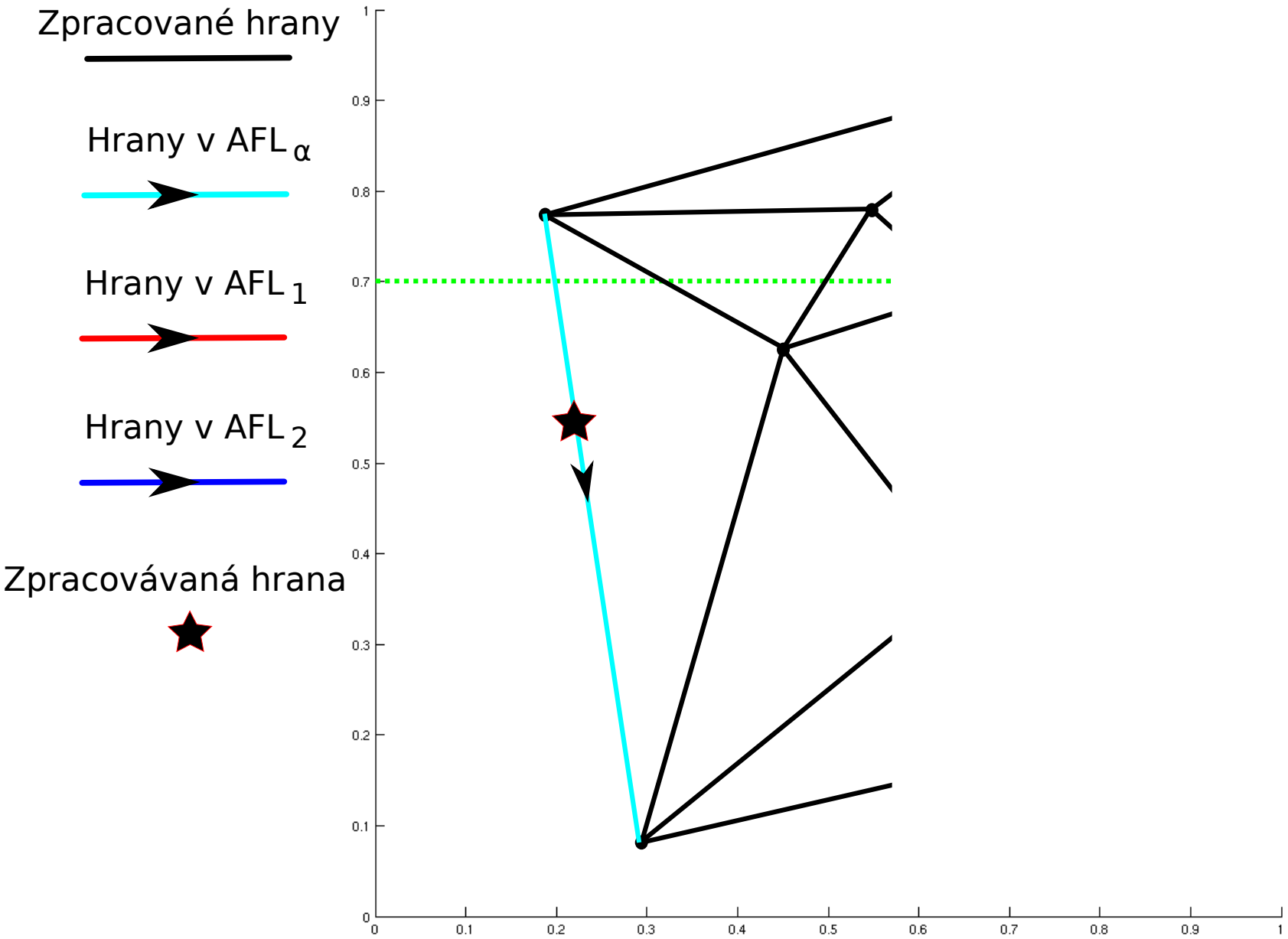
Hrany v AFL $_1$

Hrany v AFL $_2$

Zpracovávaná hrana



Příklad


Zpracované hrany
Hrany v AFL α
Hrany v AFL 1
Hrany v AFL 2
Zpracovávaná hrana
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Příklad

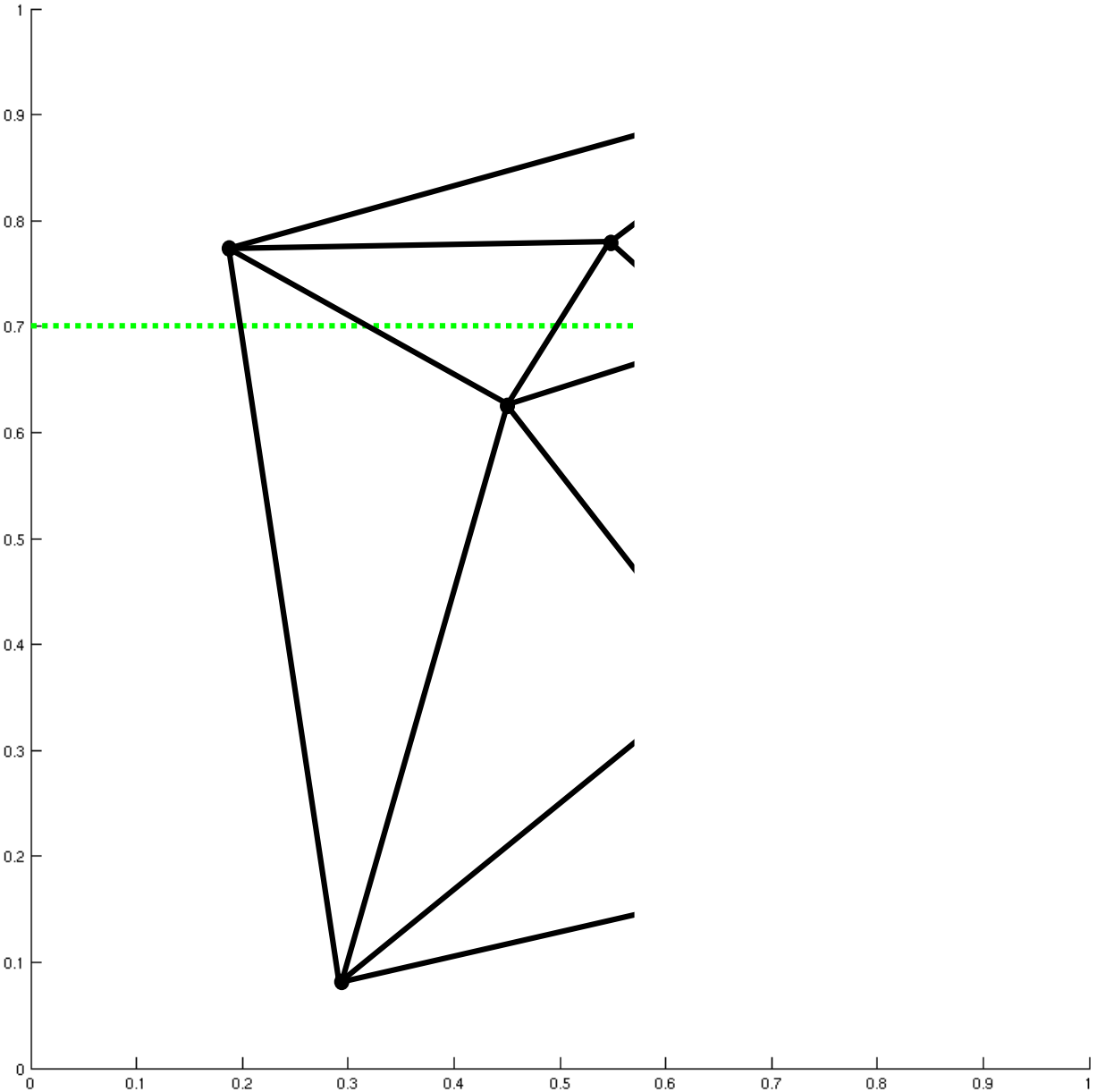
Zpracované hrany

Hrany v AFL $_\alpha$

Hrany v AFL $_1$

Hrany v AFL $_2$

Zpracovávaná hrana



Příklad

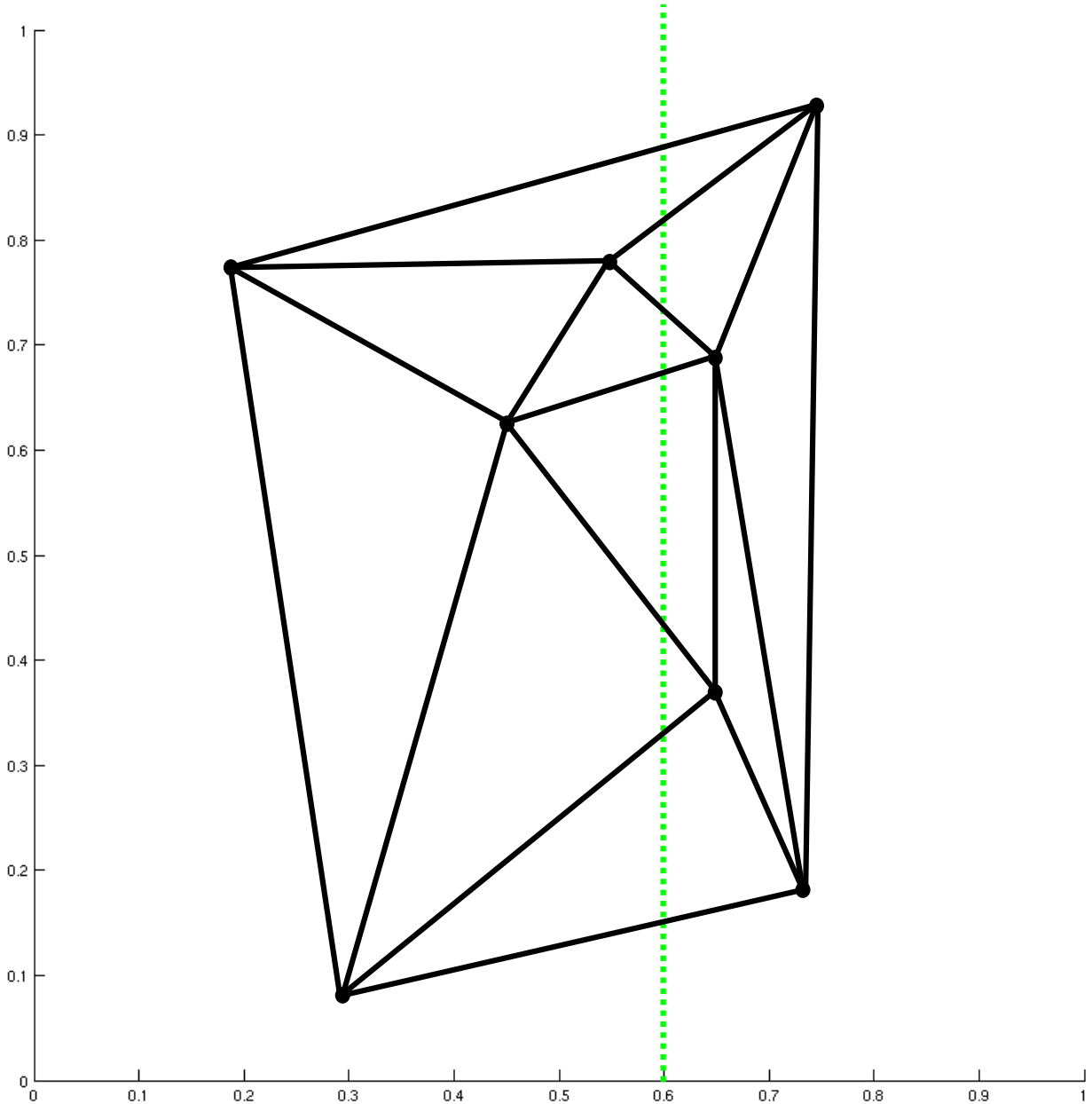
Zpracované hrany

Hrany v AFL $_\alpha$

Hrany v AFL $_1$

Hrany v AFL $_2$

Zpracovávaná hrana



Příklad

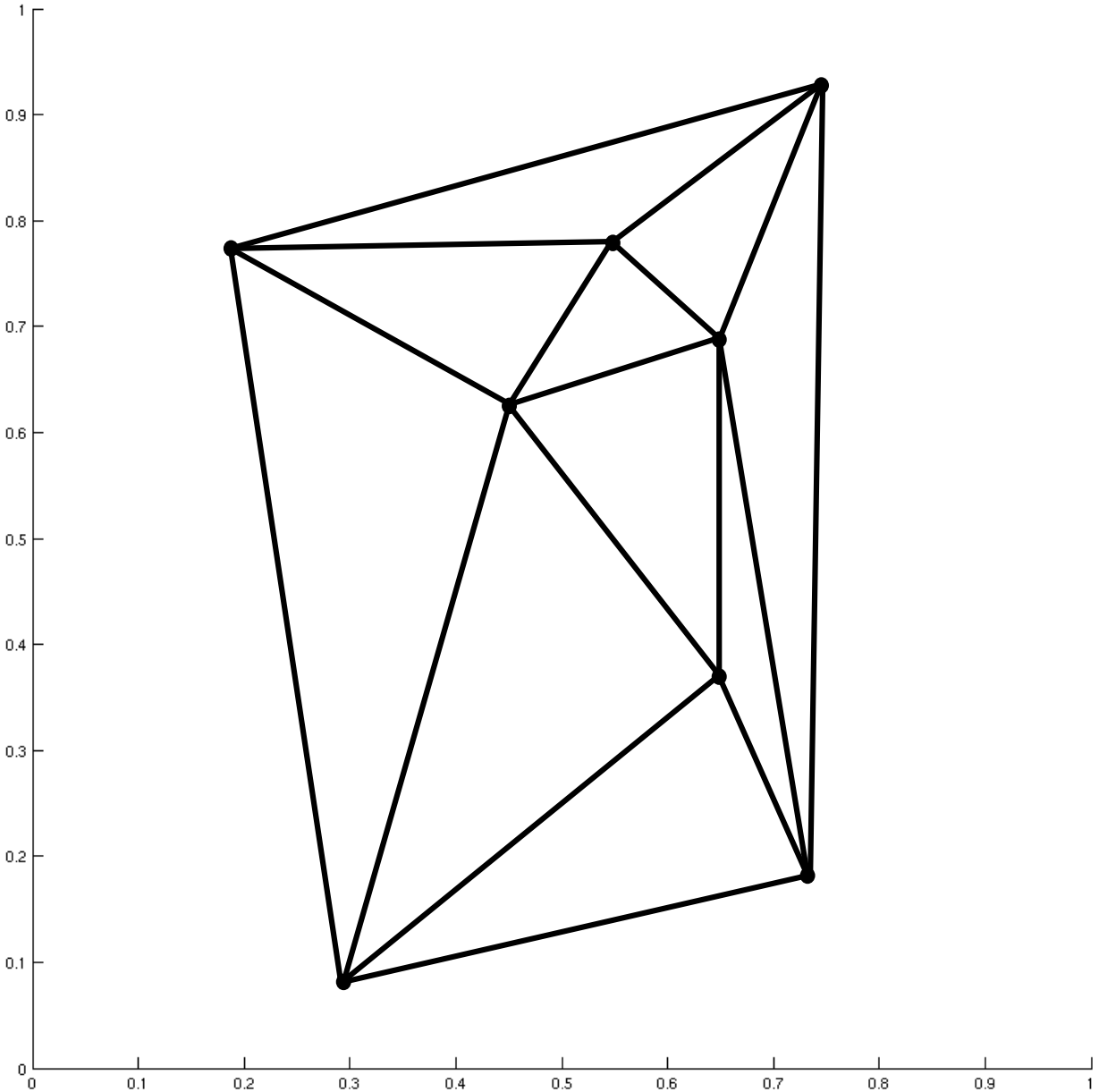
Zpracované hrany

Hrany v AFL $_\alpha$

Hrany v AFL $_1$

Hrany v AFL $_2$

Zpracovávaná hrana



Časová složitost

Největší problém je nalezení bodu pro nějakou hranu, který minimalizuje Delaunay vzdálenost, složitost této operace je $\mathcal{O}(n)$. Tuto operaci musíme provést pro každý trojúhelník triangulace. Protože počet trojúhelníků roste lineárně s počtem bodů, tak výsledná složitost pro dvoudimenzionální prostor je $\mathcal{O}(n^2)$. Pro obecný, d-dimenzionální prostor, je složitost nejhoršího případu $\mathcal{O}(n^{\lceil \frac{d}{2} \rceil + 1})$.

Zdroje

[cignoni] P. Cignoni, M. Montani, R. Scopigno. *DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in E^d*

M. Berg, O. Cheong, M. Kreveld, M. Overmars. *Computational Geometry: Algorithms and Applications*

Filip Leifer. *Delaunayho triangulace a její aplikace*

OI-OPPA. European Social Fund
Prague & EU: We invest in your future.