Zobrazování 2D křivek

5. cvičení IZG

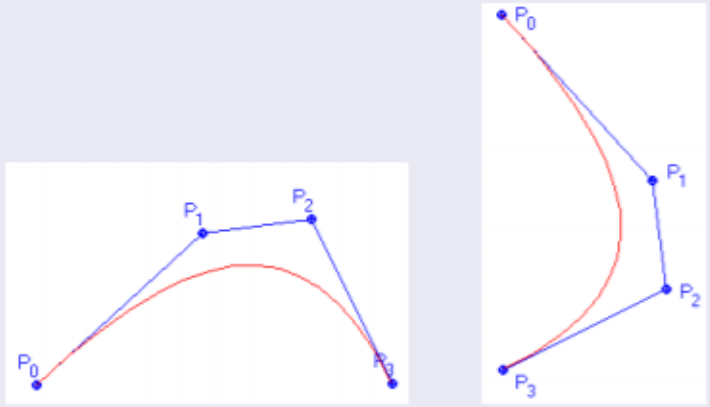
Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
isvec@fit.vutbr.cz

19.04.2021
Autor: Ing. Tomáš Švec

Požadované vlastnosti křivek

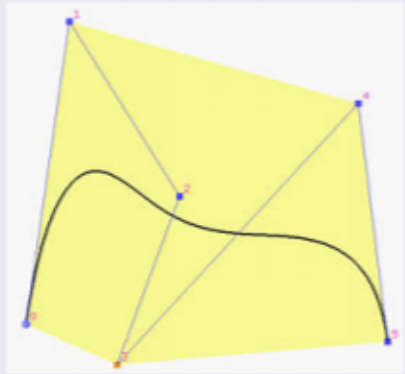
Invariance k affiním transformacím a projekci

- Rotace řídících bodů nemá vliv na tvar křivky.



Konvexní obálka

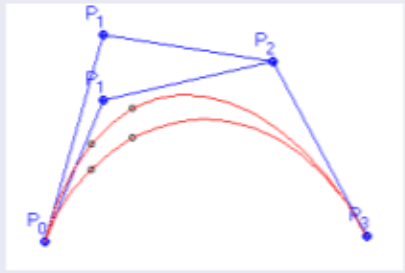
- Křivka leží v konvexní obálce svých řídících bodů.



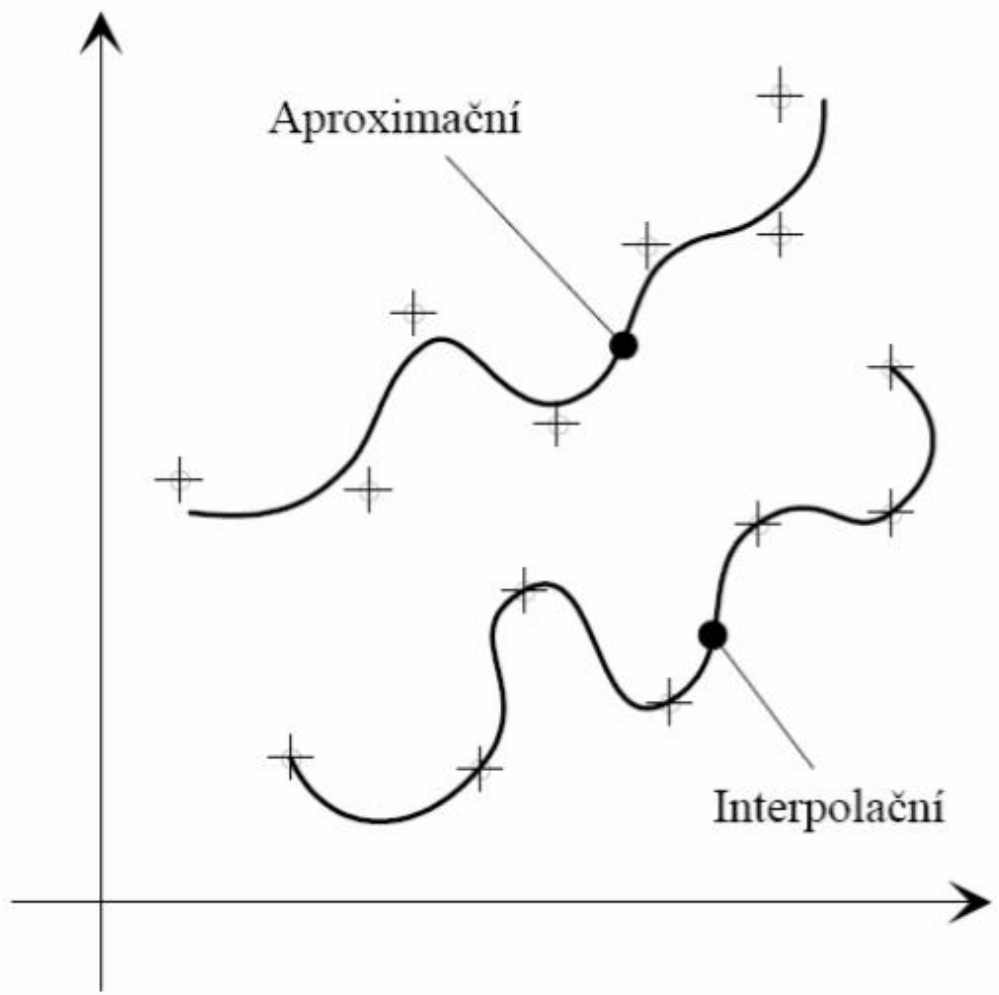
Interpolace krajních bodů

- Křivka prochází krajními body.

Lokalita změn



Základní druhy křivek 1/2



Interpolační křivka

- Křivka přímo prochází body (*"proložení"bodů křivkou*).

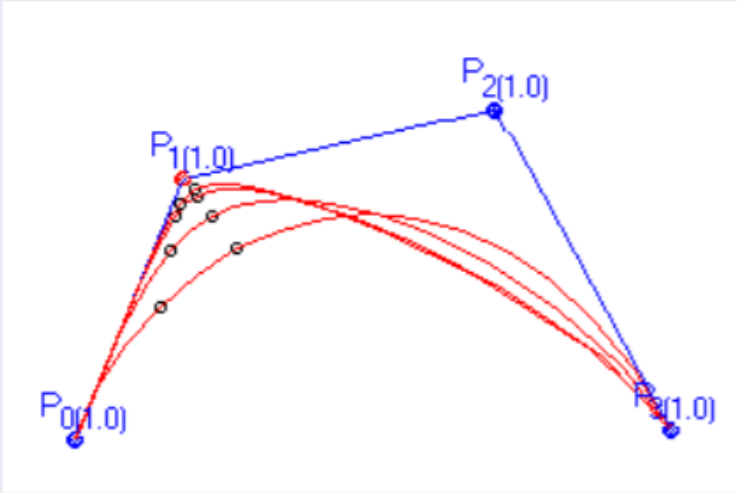
Aproximační křivka

- Neprochází body.
- Tzv. *řídící body*).

Základní druhy křivek 2/2

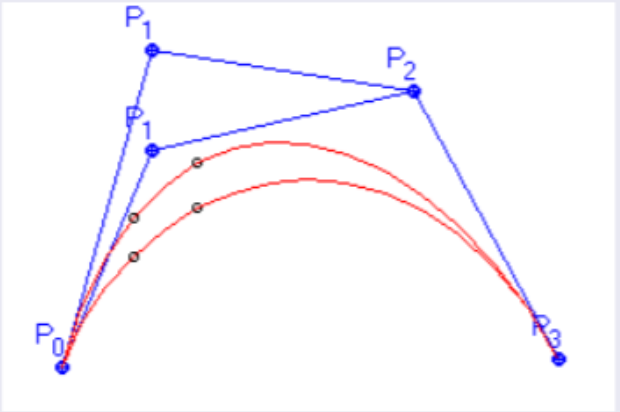
Racionální křivka

- *Váhové koeficienty* w_i řídících bodů.
- Invariantní vůči perspektivní projekci.



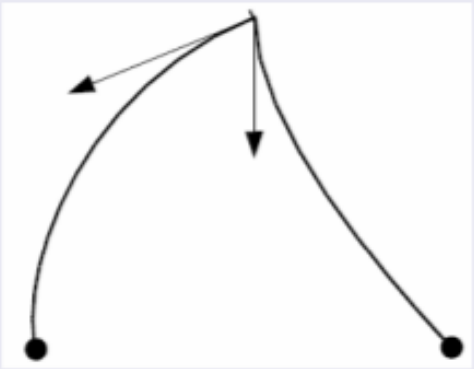
Neracionální křivka

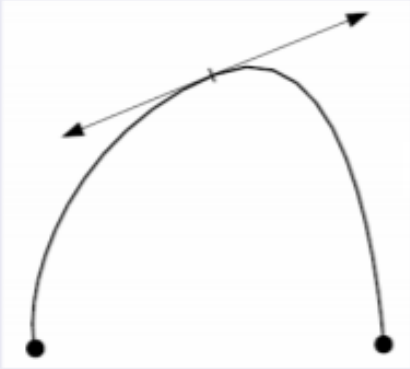
- Tvar křivky ovlivňujeme pouze *změnou polohy řídících bodů*.
- Speciální případ racionální křivky, kdy $w_i = 1$.
- Nejsou invariantní vůči perspektivní projekci!



Možnosti *parametrické spojitosti* C^n

- C^0 – totožnost navazujících koncových bodů.
- C^1 – totožnost tečných vektorů v navazujících bodech.
- C^2 – totožnost vektorů 2. derivace v navazujících bodech.

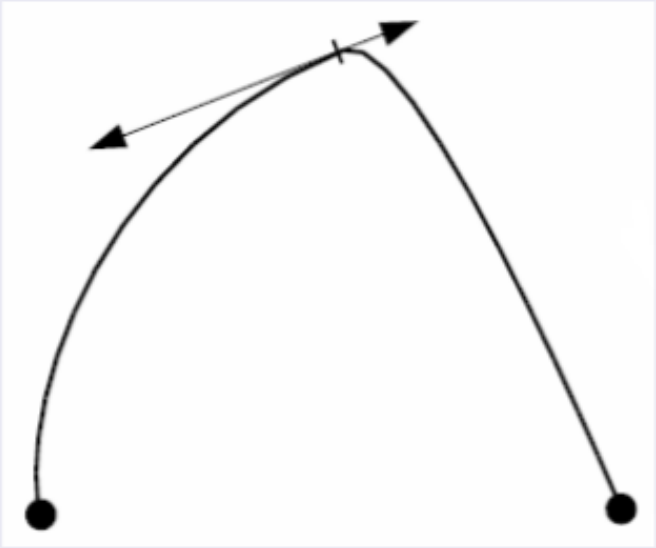




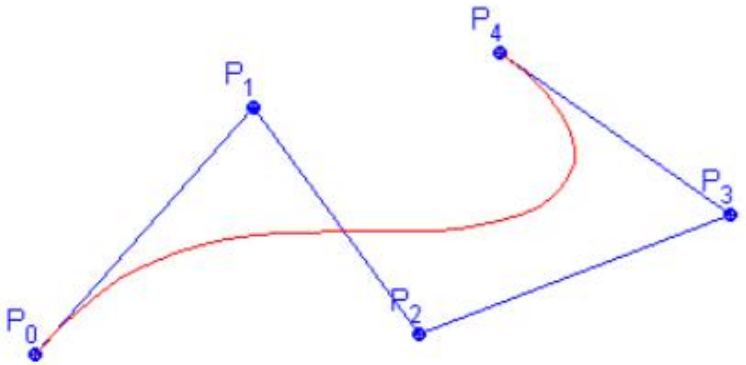
- Čím vyšší spojitost, tím déle se oba segmenty k sobě přimykají.

Oslabená podmínka spojitosti (též. *geometrická spojitost* G^n)

- G^0 – totožnost navazujících koncových bodů.
- G^1 – tečné vektory v navazujících bodech jsou lineárně závislé.
- G^2 – shoda první křivosti v navazujících bodech.



Beziérovy křivky 1/2



- Aproximační křivky (2D grafika, fonty, šablonování) 1960.
- Polynomiální křivka s použitím Bernsteinových polynomů B_i^n.
- Křivka stupně n určena $n+1$ body.
- Prochází koncovými body.

Definice křivky

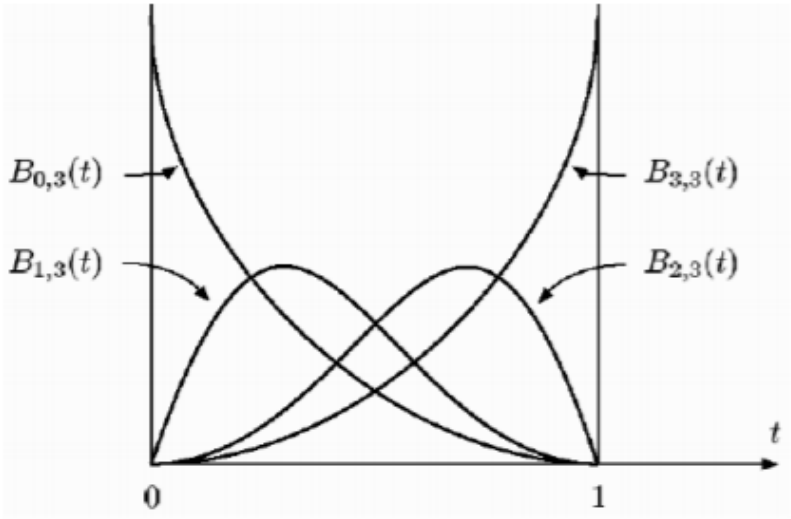
$$Q(t) = \sum_{i=0}^{n} P_i . B_i^n(t);\ i = 0, 1, \ldots, n$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i};\ t \in < 0, 1 >$$

Tečné vektory v koncových bodech

$$\vec{P'(0)} = 3(P_1 - P_0)$$
$$\vec{P'(1)} = 3(P_3 - P_2)$$

Beziérovy křivky 2/2



Rekurentní definice Bernsteinových polynomů

$$B_i^n(t) = (1-t).B_i^{n-1}(t) + t.B_{i-1}^{n-1}(t)$$

- Využívá ji algoritmus *de Casteljau* pro vykreslení křivky (viz. dále).

Další vlastnosti polynomů

- Mají nezápornou hodnotu.
- Mají jednotkový součet – křivka leží v konvexní obálce.
- *Rekurentní definice.*

- Beziérova křivka 3. řádu popsaná 4 řídicími body
- Segment, který se dá spojovat s dalšími Beziérovými kubikami

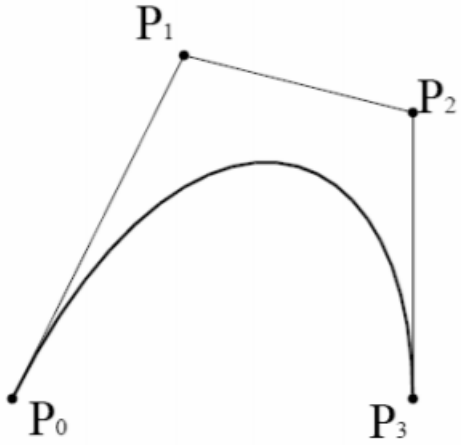
$$Q(t) = P_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + P_3 B_3^3(t) = \sum_{i=0}^{3} P_i B_i^3(t)$$

$$B_0^3(t) = (1-t)^3$$

$$B_1^3(t) = 3t(1-t)^2$$

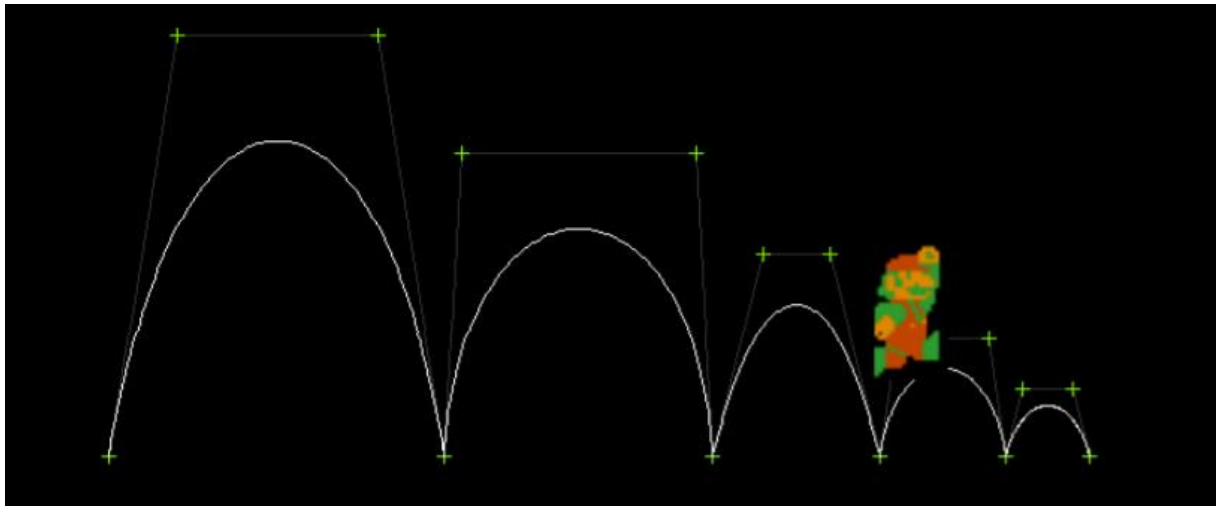
$$B_2^3(t) = 3t^2(1-t)$$

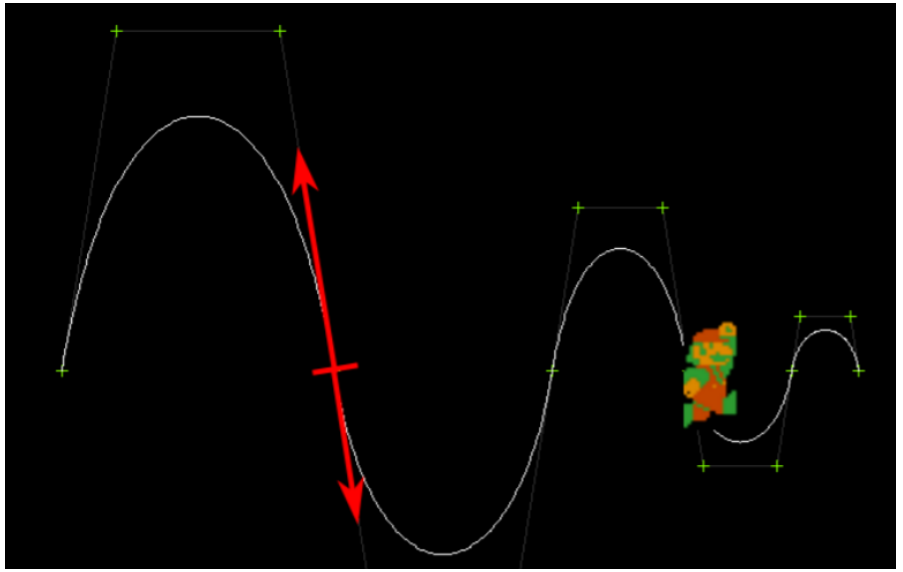
$$B_3^3(t) = t^3$$



Obsah cvičení

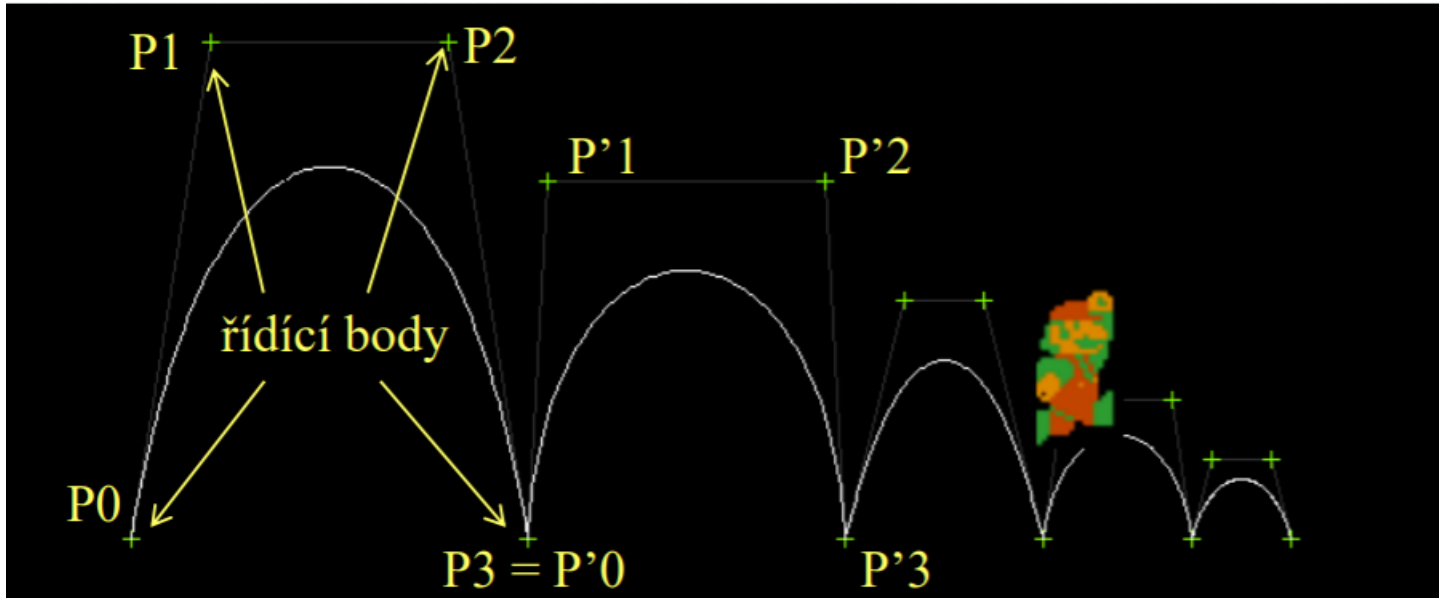
- Soubor student.cpp
- **Úloha 1** (2b) – Výpočet trajektorie pomocí Beziérových kubik
- **Úloha 2** (1b) – Úprava řídicích bodů pro G1 spojitost křivky





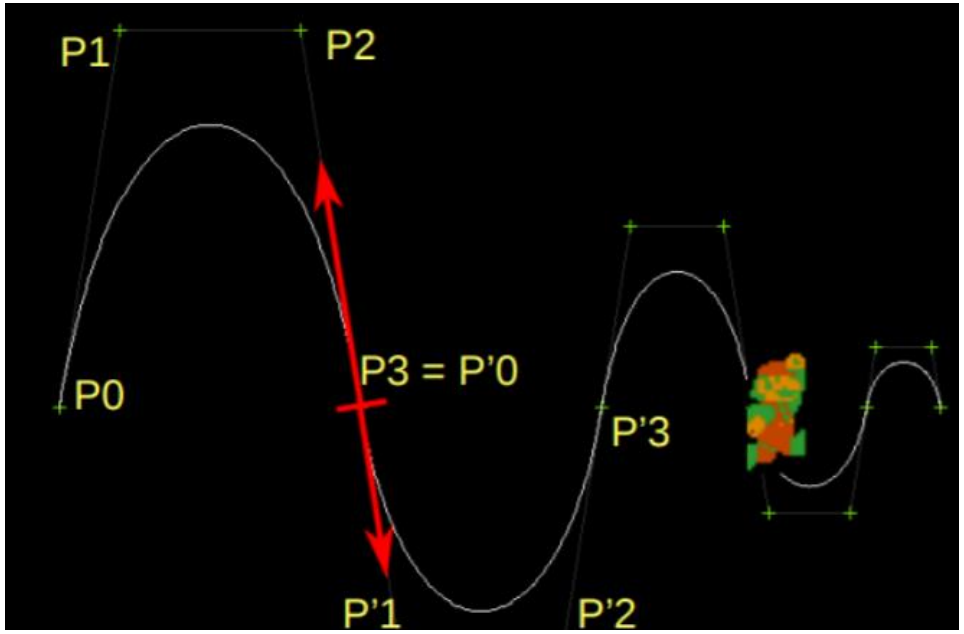
Úloha č. 1 – Zakřivení trajektorie

- Ve funkci **`bezierCubicsTrajectory()`** získávání řídících bodů a následné volání funkce **`bezierCubic()`**
- Ve funkci **`bezierCubic()`** výpočet bodů trajektorie za pomocí Beziérovy kubiky ze 4 řídících bodů
- C0/G0 spojitost stačí – poslední a první bod následujících kubik stejný



Úloha č. 2 – spojitost trajektorie křivky (G_1)

- Potřeba upravit polohu řídicích bodů ve funkci **`initControlPointsDown()`** tak, aby byla zachována spojitost G1 (tečné přímky v bodech)
- Zkopírujte obsah funkce **`initControlPointsUp()`** a ručně spočítejte nové souřadnice potřebných bodů



Pomocné funkce – vector.h

- **`S_Vector (vektor prvků Point2d)`**
  - `struct Point2d { double x, y, weight};`
- **`point2d_vecGet(pVec, i)`**
  - získání i-tého prvku z vektoru pVec
- **`point2d_vecGetPtr(pVec, i)`**
  - ukazatel na i-tý prvek vektoru pVec
- **`point2d_vecSize(pVec)`**
  - vrátí velikost vektoru
- `S_Vector* vector = point2d_vecCreateEmpty()`
  - vytvoří prázdný vektor pro prvky typu Point2d
- `point2d_vecPushBack(pVec, p)`
  - vloží na konec vektoru pVec prvek p
- `point2d_vecSet(pVec, i, p)`
  - vloží prvek p do vektoru pVec na index i
- `point2d_vecRelease(&pVec)`
  - zruší kompletně celý vektor pVec – už s ním nelze pracovat
- `point2d_vecClean(pVec)`
  - smaže prvky vektoru – vektor je prázdný o velikosti 0

Další funkce viz vector.h

Dotazy?