11. rasterizace,....

Rasterizace je proces převodu vektorové reprezentace dat na jejich rastrovou formu s cílem dosáhnout maximální možnou kvalitu a zároveň rychlost výsledného zobrazení.

Úsečka

Úsečka je základní geometrická vektorová entita definovaná souřadnicemi dvou koncových bodu. Geometrie úsečky je matematicky popsána rovnicí přímky, která muže být podle potřeby v tvaru:

- obecném $Ax + By + C = 0, A = (y_2 - y_1), B = (x_2 - x_1)$
- parametrickém $x = x_1 + t\,(x_2 - x_1)\,,\ y = y_1 + t\,(y_2 - y_1)$
- směrnicovém $y = kx + q,\ k = \Delta y/\Delta x = (y_2 - y_1)/(x_2 - x_1)$

Vykreslení úsečky pomocí níže popsaných algoritmů je realizováno pouze pokud je úsečka v prvním kvadrantu je rostoucí od počátečního bodu a největší přírůstek je ve směru osy X. Pokud úsečka tyto podmínky nesplňuje je potřeba ji převést.

DDA algoritmus

Jeden z nejjednodušších algoritmu pro rasterizaci úsečky. Používá floating-point aritmetiku. Není proto tak efektivní a je těžší jej implementovat pomocí HW. Při výpočtu používám k výpočtu následujícího kroku výsledek z kroku předchozí.

Úsečku vykreslujeme od počátečního ke koncovému bodu. V ose X je přírůstek 1. V ose Y je přírůstek určen velikostí směrnice **k** úsečky. Jelikož potřebujeme celočíselné souřadnice je hodnota v ose Y zaokrouhlována.

$$\mathbf{k} = (y_2\text{-}y_1) / (x_2 - x_1)$$

```
LineDDA(int x1, int y1, int x2, int y2)
{
        double k = (y2-y1) / (x2-x1);
        double y = y1;

        for (int x = x1; x <= x2; x++)
        {
                draw_pixel( x, round(y));
                y += k;
        }
}
```

Vykreslování úsečky s hlídáním chyby

Varianta DDA algoritmu. Nepočítá konkrétní hodnotu v ose y, ale její odchylkou od skutečné hodnoty. V ose X je opět přírustek v každém kroku 1. V ose Y je přírustek založen na chybě. Pokud je hodnota chyby větší než 0.5, zvětšíme Y o 1 a snížíme chybu o 1. Pokud je chyba menší, tak pouze k aktuální chybě přičtu **k**. To je zde opět získáno jako směrnice u DDA algritmu. Tato varianta tak ušetří potřebu zaokrouhlování.

```
LineEC(int x1, int y1, int x2, int y2)
{
      double    k = (y2-y1) / (x2-x1);
      double    E = 0;
      int       y = y1;

      for (int x = x1; x <= x2; x++)
      {
            draw_pixel( x, y);
            E += k;
            if (E >= 0.5) { y++; E -= 1; }
      }
}
```

Bressenhamův algoritmus

Patří k dnes nejpoužívanějším a nerozšířenějším algoritmům. Na rozdíl od předchozích variant pracuje pouze s celočíselnou aritmetikou. Je tak mnohem efektivnější a jeho implementace v HW je jednodušší. Algoritmus stejně jako v předchozích případech vykresluje od počátečního ke koncovému bodu. V ose X je přírůstech 1. V ose Y je přírůstek určen znaménkem prediktoru.

Při odvození prediktoru vycházíme ze směrnicového tvaru úsečky. Cílem je zjistit, zda-li vůči aktuální pozici $[x_i,y_i]$ leží následující pixel $[x_{i+1},y_{i+1}]$ na pozici $[x_{i+1}+1,y_{i+1}]$, nebo $[x_{i+1}+1,y_{i+1}+1]$. Vyjádříme si proto vzdálenost každého řešení(4.7.,4.8.) od skutečné pozice přímky(4.6.). Abychom nemuseli používat floating-point aritmetiku, tak pracujeme s rozdílem těchto vzdáleností, konkrétně jeho znaménkem.(4.9.).

Hodnota Δd je však stále reálné číslo, kvůli hodnotě směrnice přímky úsečky k. Reálné hodnoty k se proto zbavíme vynásobením celého vztahu pro Δd hodnotou Δx $(k = \Delta y/\Delta x)$
(viz. rovnice 4.10).

$$
\begin{aligned}
y_{i+1} &= kx_{i+1} + q = k(x_i + 1) + q && (4.6)\\
d_1 &= y_{i+1} - y_i = k(x_i + 1) + q - y_i && (4.7)\\
d_2 &= y_i + 1 - y_{i+1} = y_i + 1 - k(x_i + 1) - q && (4.8)\\
\Delta d &= d_1 - d_2 = 2k(x_i + 1) - 2y_i + 2q - 1 && (4.9)\\
p_i &= \Delta d \Delta x = 2\Delta y x_i - 2\Delta x y_i + 2\Delta y + \Delta x(2q - 1) && (4.10)
\end{aligned}
$$

$$
\begin{aligned}
p_i < 0 &\implies y_{i+1} = y_i && (4.11)\\
p_i \geq 0 &\implies y_{i+1} = y_i + 1 && (4.12)
\end{aligned}
$$

Rekurentní vztah pro vyjádření prediktoru (závisí na předchízejícím prediktoru a jeho znaménku).

$$
\begin{aligned}
p_{i+1} &= 2\Delta y x_{i+1} - 2\Delta x y_{i+1} + 2\Delta y + \Delta x(2q - 1) && (4.13)\\
p_{i+1} &= p_i + 2\Delta y \iff p_i < 0 && (4.14)\\
p_{i+1} &= p_i + 2\Delta y - 2\Delta x \iff p_i \geq 0 && (4.15)
\end{aligned}
$$

Startovací hodnota prediktoru je $pi = 2\Delta y - \Delta x$

```
LineBres(int x1, int y1, int x2, int y2)
{
      int   dx = x2-x1, dy = y2-y1;
      int   P = 2*dy – dx;
      int   P1 = 2*dy, P2 = P1 - 2*dx;
      int   y = y1;

      for (int x = x1; x <= x2; x++)
      {
            draw_pixel( x, y);
            if (P >= 0)
                  { P += P2; y++; }
            else
                  P += P1;
      }
}
```

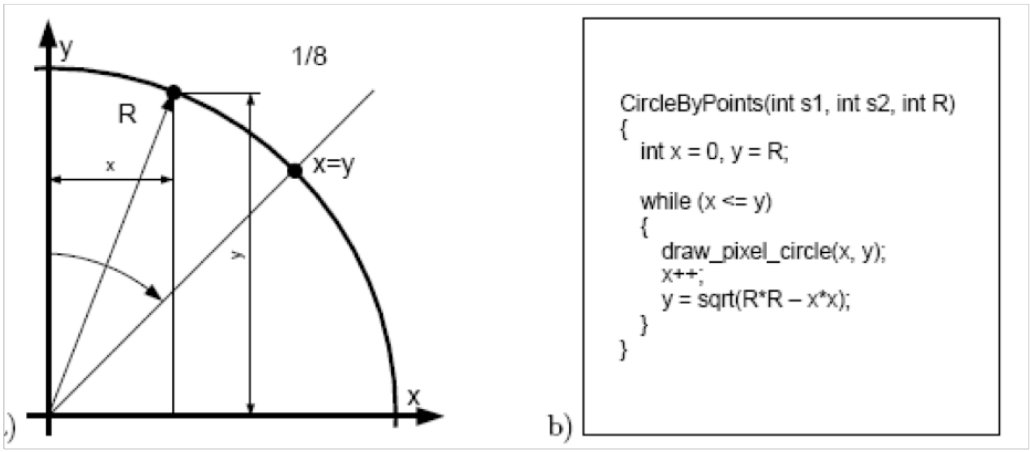
Kružnice

Kružnice je osminásobně symetrická vektorová entita, která je určena souřadnicemi středu a hodnotou poloměru. Pro její vykreslení stačí vypočítat pouze její 1/8 bodu v 1/2 prvního kvadrantu a zbývající body získat záměnou souřadnic. Pro výpočet je nevhodnější mít kružnici v počátku souřadného systému. V následujících příkladech proto uvažujeme pouze takovéto kružnice.

Rovnice kružnice: $F(x, y) : (x - s_1)^2 + (y - s_2)^2 = R^2$

Vykreslení po bodech

Jedná se o nejjednodušší způsob vykreslení. Pracuje s desetinnými čísly a proto je výpočetně náročný. V ose X se pohybujeme ve směru hodinových ručiček s přírůstkem 1. V ose X začínáme v hodnotě 0 a končíme v místě kde X == Y. Od tohoto místa dále už je kružnice symetrická. V ose Y získáme přírůstek dosazením do rovnice.

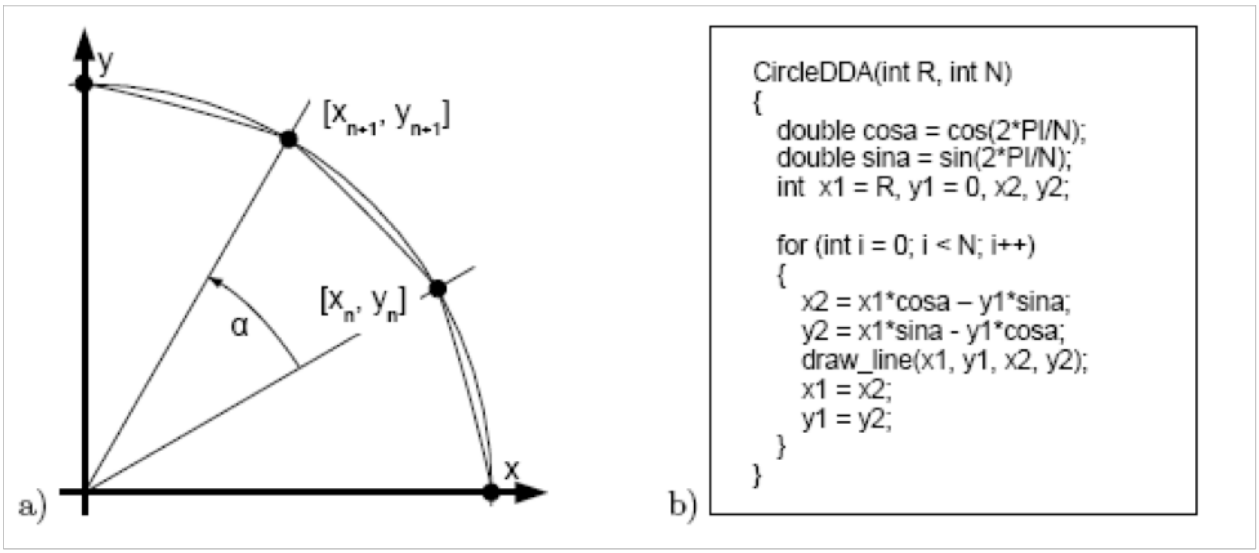


Vykreslení kružnice jako n-úhelník

Ekvivalent algoritmu DDA pro kružnici. Pracuje s floating-point aritmetikou a je proto neefektivní a v HW těžko implementovatelný. Využívá rekurentní vztahy pro výpočet souřadnic. Úhel α má určitý přírůstek a hodnoty jeho *sin* a *cos* můžou být vypočítány dopředu.

$$x_{n+1} = x_n \cos \alpha - y_n \sin \alpha$$
$$y_{n+1} = x_n \sin \alpha + y_n \cos \alpha$$

Tento algoritmus, tak jak je popsán, je citlivý na kumulované numerické chyby výpočtu funkci sin a cos. Pro odstranění tohoto vlivu je možné počítat pouze 1/8 kružnice a vykreslovat s využitím symetrie nebo použít "High precision" variantu algoritmu, která pro výpočet sin a cos využívá Taylorova rozvoje.



Midpoint algoritmus

Princip stejná jako u Bressenhemova algoritmu. Pracuje s celými čísly a pouze s jednoduchými operacemi. Rozdílné je pouze stanovení kritéria pro určení prediktoru. Vykreslení probíhá pouze pro 1/8. Zbytek je dopočítán díky symetrii. V ose X začínáme v 0 s přírůstkem 1 a pokračujeme dokud X == Y. V ose Y klesáme od R. O posuvu v osy Y rozhodujeme podle znaménka prediktoru. Jako prediktor použijeme rovnici kružnice v jejím homogenním tvaru $F(x, y) : x^2 + y^2 - R^2 = 0$.

Předpokládejme, že jsme v celočíselné pozici $[x_i, y_i]$ a rozhodujeme, jestli následující pixel leží na pozici $[x_{i+1}+1, y_{i+1}]$ nebo $[x_{i+1}+1, y_{i+1}-1]$. Toto rozhodnutí provedeme podle polohy středního bodu $[x_i + 1, y_i - 1/2]$ (midpointu) vůči kružnici (uvnitř nebo vně). Midpoint dosadíme do vztahu pro prediktor. Pokud vyjde prediktor záporný Y zůstává stejné, pokud je prediktor kladný, znamená to, že bod leží vně. Y proto snížíme o 1.

$$\begin{aligned} p_i &= F(x_i + 1, y_i - \frac{1}{2}) \\ p_i &= (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - R^2 \end{aligned}$$

$$\begin{aligned} p_i < 0 &\implies y_{i+1} = y_i \\ p_i \geq 0 &\implies y_{i+1} = y_i - 1 \end{aligned}$$

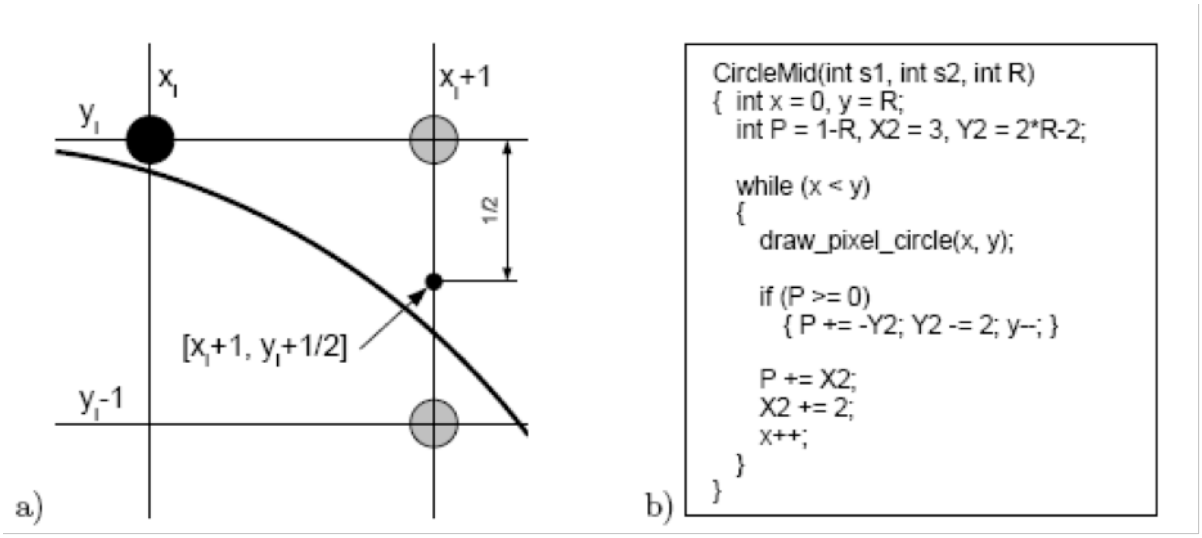
Pro průběžné výpočty během vykreslování kružnice bude opět výhodnější vyjádřit prediktor rekurentně, tedy na základe hodnoty z předchozího kroku (viz. rovnice 4.25). Výsledný rekurentní vztah pro prediktor bude opět samozřejmě záležet nejen na hodnotě předchozího prediktoru, ale také na jeho znaménku (viz. rovnice 4.26 a 4.27).

$$p_{i+1} = (x_{i+1}+1)^2 + (y_{i+1}-\frac{1}{2})^2 - R^2 \quad (4.25)$$

$$p_{i+1} = p_i + 2x_i + 3 \iff p_i < 0 \quad (4.26)$$

$$p_{i+1} = p_i + 2x_i - 2y_i + 5 \iff p_i \geq 0 \quad (4.27)$$

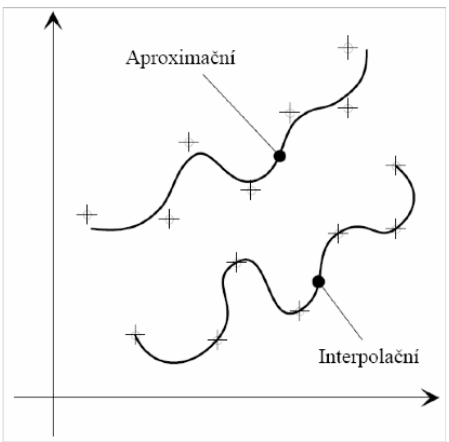
Startovací hodnota prediktoru je: *pi = 1 – R*



Křivky

Požadované vlastnosti křivky:

- invariance k lineárním transformacím – rotace řídících bodů nemá vliv na tvar
- konvexní obálka – křivka leží v konvexní obálce svých bodů
- lokalita změn – poloha 1 řídícího bodu mění jenom část křivky, ne celou
- interpolace krajních bodů – křivka nimi prochází



Základní druhy:

- interpolační – prochází body
- aproximační – aproximují k bodům

- racionální – řídící body mají váhové koeficienty, jsou invariantní vůči perspektivní projekci
- neracionální – tvar můžeme ovlivnit pouze změnou řídících bodů, invariantní vůči projekci nejsou, speciální případ, kdy váha je *wi =1*.

Vyjádření křivky

- klasické y = f(x) není vhodné
- parametrizovaná křivka *Q(t) = [x(t), y(t)]; t je z <0,1>*
- polynomiální křivka - kubické polynomy

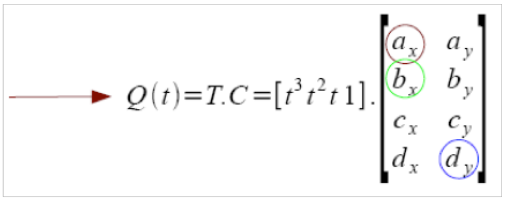
$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$

Maticový zápis

$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$

$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$



Spojitost křivek

Možnosti parametrické spojitosti C^n

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

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.

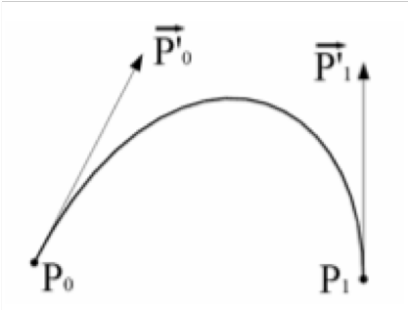
Spline křivky

Po částech polynomiální křivka. Minimalizuje křivost křivky. Umožňuje efektivní řízení tvaru. Přirozený spline je spline, který interpoluje své řídící body.

Fergusonova kubika – interpolační

Patří mezi interpolační křivky. Je určena dvěma koncovými body a dvěma tečnými vektory. Neinteraktivní a neintuitivní řízení tvaru. Nelokální změna tvaru posunem jednoho bodu. Umožňuě navazováni segmentů, má

totožné koncové body a shodné tečné vektory.

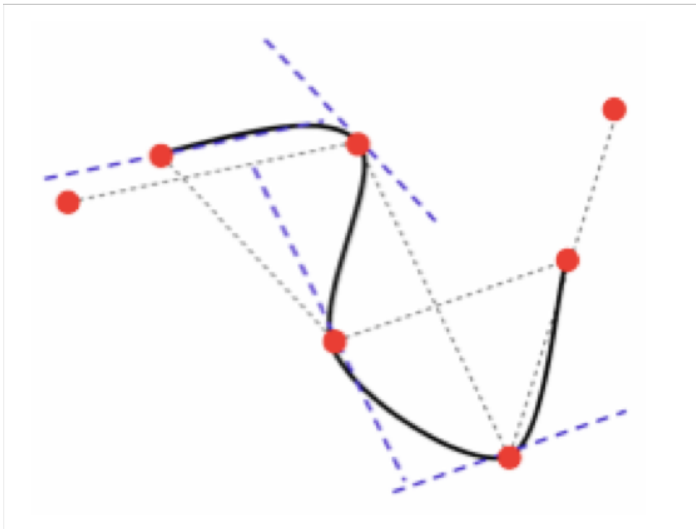


Kochanek-Bartels spline

Interpolační spline křivka. Pro interpolaci využívá Fergusonových kubik. Určeno množinou N interpolovaných bodů P_i a odpovídajícími koeficienty a_i(tenze), b_i(šikmost) a c_i(spojitost). Koeficienty určují chování křivky v daném bodě.

Catmull-Rom spline

Interpolační spline křivka. Určen množinou N bodů. Tečný vektor v P_i je rovnoběžný s vektorem P_{i-1} a P_{i+1}. Jedná se o Kochanek-Bartels s nulovými koeficienty. Vychází z bodu P_1 a končí v bodě P_{N-1}. Interpolací koncových bodů zajistíme jejich opakování. Výsledný spline neleží v konvexní obálce.



Beziérovy křivky

Aproximační křivka. Využívá Bernsteinových polynomů. Křivka stupně N je určena N+1 body. Interpoluje koncové body.

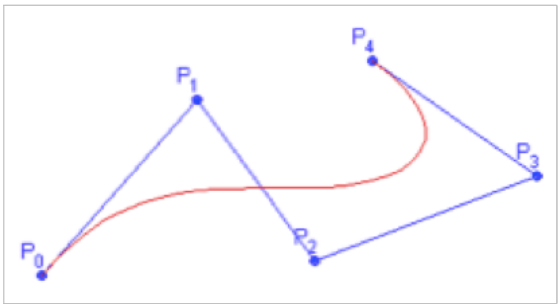
Definice křivky

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

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

Rekurentní definice Bernsteinových polynomů

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

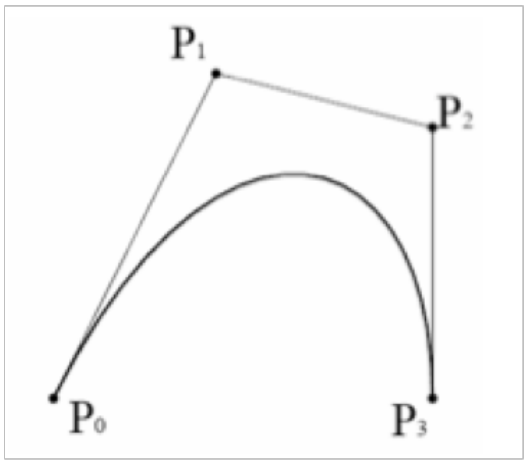


Algoritmus de Casteljau

Rekurzivní algoritmus vykreslování Beziérových křivek. Opakovaný výpočet pro různé hodnoty t se zvolený krokem. Spojování vypočtených bodů úsečkami. Problémem je zvolení optimální velikosti kroku a rovnoměrné rozprostření bodů na rovné i křivé části.

Beziérovy kubiky

Aproximační křivka. Segment popsán čtyřmi řídícími body. Nelokální změna tvaru posune, jednoho bodu. Invariantní k lineárním transformacím. Využívá Bernsteinových polynomů.

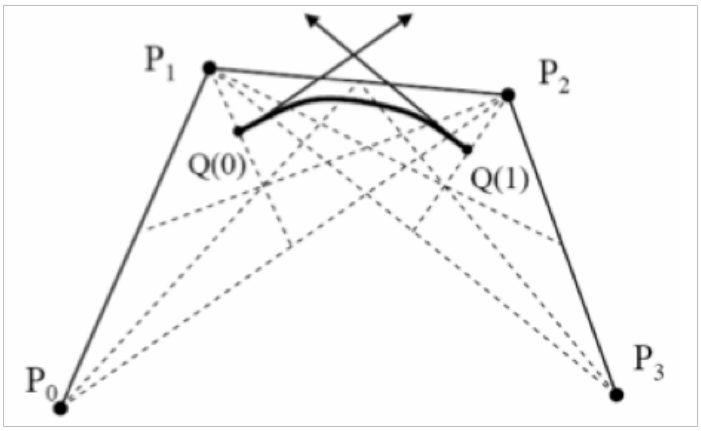


Racionální Beziérovy křivky

Místo neracionálních Bernsteinových polynomů jsou použity racionální polynomy. Pro $w_i = 1$ jsou racionální polynomy rovny Bernsteinovým polynomům. Mají nezápornou hodnotu. Mají jednotkový součet a leží v konvexní obálce. Nemají rekurentní definici. Pro vykreslení tedy nejde použít Algoritmus de Casteljau.

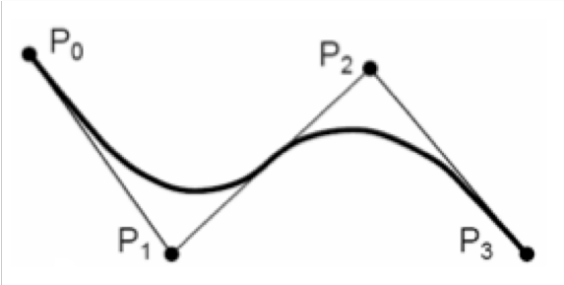
Coonsovy křivky

Aproximační křivky. Křivka stupně N je určena N+1 řídícími body. Neinterpoluje koncové řídící body. S křivkou pracujeme po segmentech. Nelze tedy přidat jeden bod, ale pouze celý segment. Navazování segmentů splňuje spojitost C2.



B-spline křivky

Aproximační křivky. Jedná se o zobecnění Coonsových křivek. Křivka určena N+1 body. Křivka stupně k má spojitost k+1. Nejde o obyčejné navazování segmentů. Polynomy mají nezápornou hodnotu. Jednotkový součet – křivka leží v konvexní obálce. Vykreslování algoritmem de Boor.



Definice křivky

$$Q(t) = \sum_{i=0}^{n} P_i.N_i^k(t);\ t \in < 0, t_m >$$

NURBS

Zobecnění B-spline křivek. Racionální křivka. Umožňuje vkládání řídícího bodu při zachování tvaru. Přesné vyjádření kuželoseček. Invariantní vůči lineárním transformacím.

Definice křivky

$$Q(t) = \sum_{i=0}^{n} P_i.R_i^k(t) = \frac{\sum_{i=0}^{n} w_i.P_i.N_i^k(t)}{\sum_{i=0}^{n} w_i.N_i^k(t)}$$

$$R_i^k(t) = \frac{w_i.N_i^k(t)}{\sum_{i=0}^{n} w_i.N_i^k(t)}$$