Vykresľovanie Bézierovej krivky pomocou algoritmu de Casteljau

Bézierové krivky sú výrazne používané v počítačovej grafike, animácii a robotike, kedy sa vytvára súvislá krivka pohybu montážnych robotov aby sa predišlo opotrebúvaniu. Matematický základ Bézierovych kriviek bol definovaný už v roku 1912 - Bernsteinove polynómy, avšak efektívny spôsob vykresľovania kriviek bol objavený až v roku 1959 Paul de Casteljauom, ktorý vymyslel výpočtovo jednoduchý a stabilný algoritmus - de Casteljauov algoritmus.

Krivka je definovaná štyrmi kontrolnými bodmi, ktoré definujú kreslenú krivku. Dá sa použiť aj vyšší počet kontrolných bodov, ale to zvyšuje výpočtovú zložitosť a všetky krivky je možné vykresliť pomocou kriviek definovaných štyrmi bodmi, ktoré sa spájajú. Zvyčajne sa to robí tak, že posledný bod jednej krivky je prvým bodom nasledujúcej (taktiež známe aj ako C0 spojitosť - dve krivky sú spojené), ale ak je potrebné aj súvislé zakrivenie, zdieľajú sa dva (C1 spojitosť - smer a "rýchlosť" pohybu sa nemení skokom) alebo až tri (C2 spojitosť - krivka nemení "zrýchenie") kontrolné body medzi dvoma krivkami.

Pre demoštratívne účely nám stačí jedna krivka definovaná štyrmi bodmi:

In[1]:=
```
(* controlPoints = {{0.636, 0.22}, {0.465, 0.94}, {-0.465, 0.94}, {-0.636, 0.22}}; *)
controlPoints = {{0.6, 0.5},
{0.2, 0.8},
{-0.2, 0.2},
{-0.6, 0.5}};
```

Následne je definovaná funkcia na interpoláciu bodu medzi dvomi predom vytvorenými bodmi. Funkcia vezme *x* a *y* súradnice bodov *p0* a *p1* a nájde bod posunutý od bodu *p0* o *t* krokov smerom k bodu *p1*:

In[2]:=
```
interpolate[p0_List, p1_List, t_] := (
intx = t * p0[[1]] + (1 - t) * p1[[1]];
inty = t * p0[[2]] + (1 - t) * p1[[2]];
{intx, inty})
```

Funkcia *bezierPoint* vezme štyri kontrolné body a vypočíta bod Bézierovej krivky v konkrétnom kroku *t*. Na výpočet sa využíva predom definovaná funkcia interpolate, ktorá najprv vezme prvé dva body, potom druhý s tretím, a nakoniec posledné dva body. Tieto tri body sa nachádzajú na úsečkách definovaných kontrolnými bodmi (jedna úsečka je medzi prvým a druhým bodom, ďalšia medzi druhým a tretím, a tretia medzi tretím a štvrtým kontrolným bodom).

Následne medzi interpolovanými bodmi vzniknú dve úsečky, pričom na každej sa opäť vypočíta bod. Vzniknuté dve body definujú poslednú úsečku, na ktorej sa interpoluje bod kresliaci samotnú krivku.

Krok *t* určuje hustotu krivky. Malé *t* definuje nepresnú a "hranatú" krivku, naopak veľké *t* je výpočtovo náročné. Napríklad ak *t* = 100, *bezierPoint* je potrebné zavolať 100-krát. Algoritmus však nepracuje so samotnou hodnotou kroku, ale s pomerom voči celkovému počtu krokov, takže v prípade, že počet krokov je 100 a *t* = 2, do funkcie sa posiela 2/100. Takto sa zabezpečí, aby *t* bolo z intervalu <0, 1>.

In[3]:=
```
bezierPoint[p0_List, p1_List, p2_List, p3_List, t_] := (
(*Interpolácia prvých troch bodov*)
lin1 = interpolate[p0, p1, t];
lin2 = interpolate[p1, p2, t];
lin3 = interpolate[p2, p3, t];

(*Interpolácia dvoch bodov na interpolovaných úsečkách*)
quad1 = interpolate[lin1, lin2, t];
quad2 = interpolate[lin2, lin3, t];

(*Interpolácia kresliaceho bodu na interpolovanej úsečke*)
cub1 = interpolate[quad1, quad2, t];
cub1)
```

Následne sa vytvorí tabuľka hodnôt. Prvé tri stĺpce definujú prvé dve úsečky, respektíve interpolované body v danom kroku na úsečkách definovaných kontrolnými bodmi. Nasledujúce dva stĺpce popisujú body na interpolovaných úsečkách a posledný stĺpec určuje kadiaľ ide samotná Bézierova krivka.

In[4]:=
```
table = Table[{
(*Pozície prvých troch bodov*)
interpolate[controlPoints[[1]], controlPoints[[2]], t/100],
interpolate[controlPoints[[2]], controlPoints[[3]], t/100],
interpolate[controlPoints[[3]], controlPoints[[4]], t/100],

(*Pozície bodov na interpolovaných úsečkách*)
interpolate[interpolate[controlPoints[[1]], controlPoints[[2]], t/100],
      interpolate[controlPoints[[2]], controlPoints[[3]], t/100], t/100],
interpolate[interpolate[controlPoints[[2]], controlPoints[[3]], t/100],
      interpolate[controlPoints[[3]], controlPoints[[4]], t/100], t/100],

(*Pozícia Bézierovej krivky*)
bezierPoint[controlPoints[[1]],
      controlPoints[[2]], controlPoints[[3]], controlPoints[[4]], t/100]},
{t, 1, 100}];
```

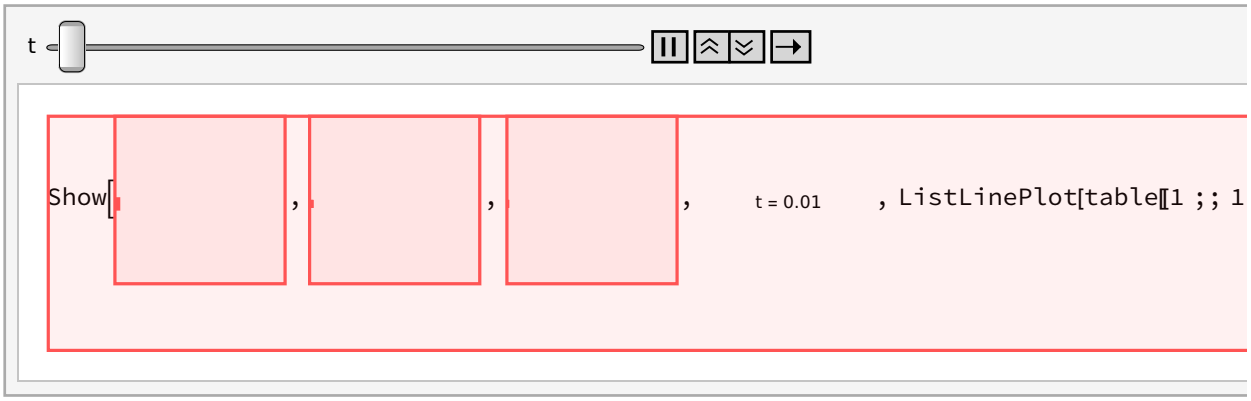
Najprv je potrebné vykresliť kontrolné body, ktoré sú pevne definované, ostatné časti sa menia podľa kroku. Po kontrolných bodoch a úsečkách medzi nimi sa vykreslia body na týchto úsečkách a interpolo-

vané úsečky. Následne sa vykreslia ďalšie dva body a úsečka medzi nimi, a nakoniec sa kreslí samotná Bézierova krivka.

Môžeme si všimnúť, že úsečky sa pohybujú podľa kroku t, ktorý je taktiež zobrazený pod grafom. Krivka sleduje polohu interpolovaných úsečiek a bodov na nich, až kým sa nevykreslí celá a animácia začne odznova.

In[5]:=
```
Pane[Animate[Show[
(*Kontrolné body a úsečky medzi nimi*)
Graphics[{Black, Dashed,
      Line[controlPoints], PointSize[Large], Red, Point[controlPoints]}],

(*Prvé interpolované body a úsečky medzi nimi*)
Graphics[{Blue, Line[{table[[t, 1]], table[[t, 2]], table[[t, 3]]}],
Black, Point[table[[t, 1]]],
Point[table[[t, 2]]],
Point[table[[t, 3]]]
}],

(*Body na interpolovaných úsečkách a ďalšia úsečka medzi nimi*)
Graphics[{Green, Line[{table[[t, 4]], table[[t, 5]]}],
Black, Point[table[[t, 4]]],
Point[table[[t, 5]]],
Point[table[[t, 6]]]
}],

(*Text zobrazujúci momentálny krok / 100*)
Graphics[{Text["t = " <> ToString[t / 100 // N]]}],

(*Bézierova krivka*)
ListLinePlot[table[[1 ;; t, 6]], PlotStyle → {Red, Thick}]],
{t, 1, 100, 1}, AnimationRate → 4
]]
```

Out[5]=



Rozšírenie programu

Ako bolo predtým spomínané, algoritmus je možné pomerne jednoducho upraviť, aby namiesto kriviek vytváral plochy. Najprv si musíme vytvoriť nové kontrolné body, no v tomto prípade ich potrebujeme aspoň 16 (4 kontrolné do jednej strany x 4 kontrolné body do druhej strany). Keďže pracujeme v troch dimenziách, musíme pridať aj z-ovú súradnicu ku každému bodu.

Interpolačná funkcia je takmer rovnaká s tým, že bola pridaná interpolácia bodu v z-ovej osi. bezierPoint je úplne rovnaký, no musíme pridať ďalšiu funkciu *bezierPatch*, ktorá vygeneruje samotnú plochu. Pri *bezierPatch* musíme nastaviť dodatočné argumenty u a v, čo sú v podstate argument t, ale v dvoch dimenziách, nakoľko okrem šírky ako pri krivkách, má plocha aj nejakú dĺžku. Veľkosť *u* a *v* taktiež určí, z koľkých štvorcov sa bude plocha skladať v danom smere, čo znamená, že čím viac štvorcov, tým je plocha samozrejme presnejšia. To je možné si aj vyskúšať pomocou samostatných sliderov pre u a v.

bezierPatch najprv vytvorí krivku v jednom smere a následne interpoluje bod krivky kolmej na ňu podľa kroku *v*. Každý bod je uložený do zoznamu, podľa ktorého je následne vykreslená vygenerovaná plocha.

In[6]:=
```
controlPoints3D = {
{{-1, 1, 0}, {-0.5, 1, 0}, {0.5, 1, 0}, {1, 1, -1}},
{{-1, 0.5, 0}, {-0.5, 0.5, 1}, {0.5, 0.5, -1}, {1, 0.5, 0}},
{{-1, -0.5, 0}, {-0.5, 0.5, 1}, {0.5, -0.5, -1}, {1, -0.5, 0}},
{{-1, -1, -1}, {-0.5, -1, 0}, {0.5, -1, 0}, {1, -1, 0}}
};
```

```
(*Interpolačná funkcia s dodatočným výpočtom pre z-súradnicu*)
interpolate3D[p0_List, p1_List, t_] := (
intx = t * p0[[1]] + (1 - t) * p1[[1]];
```

```
inty = t * p0[[2]] + (1 - t) * p1[[2]];
intz = t * p0[[3]] + (1 - t) * p1[[3]];
{intx, inty, intz})



bezierPoint3D[p0_List, p1_List, p2_List, p3_List, t_] := (
lin1 = interpolate3D[p0, p1, t];
lin2 = interpolate3D[p1, p2, t];
lin3 = interpolate3D[p2, p3, t];

quad1 = interpolate3D[lin1, lin2, t];
quad2 = interpolate3D[lin2, lin3, t];

cub1 = interpolate3D[quad1, quad2, t];
cub1)



(*Výpočet Bézierovej plochy pre zadané kontrolné body*)
bezierPatch3D[imax_, jmax_] := (
vert = {};
For[i = 0, i < imax, i++ ×
For[j = 0, j < jmax, j++,
(*Predelenie krokov celkovým počtom, rovnako ako pri t*)
u = i / imax;
v = j / jmax;

(*Výpočet kriviek pre každý rad kontrolných bodov*)
pointsU = {bezierPoint3D[controlPoints3D[[1, 1]],
          controlPoints3D[[1, 2]], controlPoints3D[[1, 3]], controlPoints3D[[1, 4]], u],
bezierPoint3D[controlPoints3D[[2, 1]], controlPoints3D[[2, 2]],
          controlPoints3D[[2, 3]], controlPoints3D[[2, 4]], u],
bezierPoint3D[controlPoints3D[[3, 1]], controlPoints3D[[3, 2]],
          controlPoints3D[[3, 3]], controlPoints3D[[3, 4]], u],
bezierPoint3D[controlPoints3D[[4, 1]], controlPoints3D[[4, 2]],
          controlPoints3D[[4, 3]], controlPoints3D[[4, 4]], u]};

(*Výpočet krivky kolmej na prvú*)
pointV = bezierPoint3D[pointsU[[1]], pointsU[[2]], pointsU[[3]], pointsU[[4]], v];

(*Uloženie vypočítaných súradníc kriviek*)
AppendTo[vert, pointV];
```

```
];];
vert)


(*Možnosť nastaviť počet štvorcov*)
Manipulate[
ListPlot3D[bezierPatch3D[u, v], Mesh → All],
{u, 5, 100, 1, Appearance → "Labeled"},
{v, 5, 100, 1, Appearance → "Labeled"}
]
```

Out[10]=