Софийски университет „Св. Климент Охридски“

Факултет по информатика и математика

Специалност: Приложна математика

Degree Elevation Algorithm

Изготвил: Иво Банков

Факултетен номер: 31631, II курс

Град София

2021 година

1. Математическо описание на алгоритъма

Основната идея на алгоритъма е, че можем да увеличим гъвкавостта на една крива, като добавим още един възел към контролния ѝ полигон без да променяме вида на кривата. Това означава, че ще представим кривата от степен **n** като крива от степен **n + 1.**

Нека $b_0, \ldots, b_n$ са контролните точки на първоначалния полигон на кривата $b^n(t)$. След увеличаването на степента точките са $b_0^{(1)}, \ldots, b_n^{(1)}, b_{n+1}^{(1)}$. Сега ще изразим новите контролни чрез старите.

$$b^n(t) = (1 - t)\, b^n(t) + t\, b^n(t)$$

$$= \sum_{i=0}^{n} (1-t) b_i B_i^n(t) + \sum_{i=0}^{n} t b_i B_i^n(t)$$

$$= \sum_{i=0}^{n} b_i \binom{n}{i} t^i (1-t)^{n+1-i} + \sum_{i=0}^{n} b_i \binom{n}{i} t^{i+1} (1-t)^{(n+1)-(i+1)}$$

$$= \sum_{i=0}^{n} b_i \binom{n}{i} \frac{B_i^{n+1}(t)}{\binom{n+1}{i}} + \sum_{i=0}^{n} b_i \binom{n}{i} \frac{B_{i+1}^{n+1}(t)}{\binom{n+1}{i+1}}$$

$$= \sum_{i=0}^{n} b_i \frac{n+1-i}{n+1} B_i^{n+1}(t) + \sum_{i=0}^{n} b_i \frac{1+i}{n+1} B_{i+1}^{n+1}(t)$$

$$= \sum_{i=0}^{n} b_i \frac{n+1-i}{n+1} B_i^{n+1}(t) + \sum_{i=1}^{n+1} b_{i-1} \frac{i}{n+1} B_i^{n+1}(t) \; *^1$$

$$= \sum_{i=0}^{n+1} b_i \frac{n+1-i}{n+1} B_i^{n+1}(t) + \sum_{i=0}^{n+1} b_{i-1} \frac{i}{n+1} B_i^{n+1}(t) \; *^2$$

$$= \sum_{i=0}^{n+1} \left(b_i \frac{n+1-i}{n+1} + b_{i-1} \frac{i}{n+1} \right) B_i^{n+1}(t)$$

$*^1$ „избутваме“ границите на втората сума с 1.

$*^2$ за горна граница на първата сума си позволяваме да напишем n + 1, защото това не променя стойността ѝ, понеже при i = n + 1 числителят се нулира.

Откъдето новите точки ще получаваме по следния начин:

$$b_i^{(1)} = \frac{i}{n+1} b_{i-1} + \left(1 - \frac{i}{n+1}\right) b_i \text{ , за i = 0, . . . ,n+1}$$

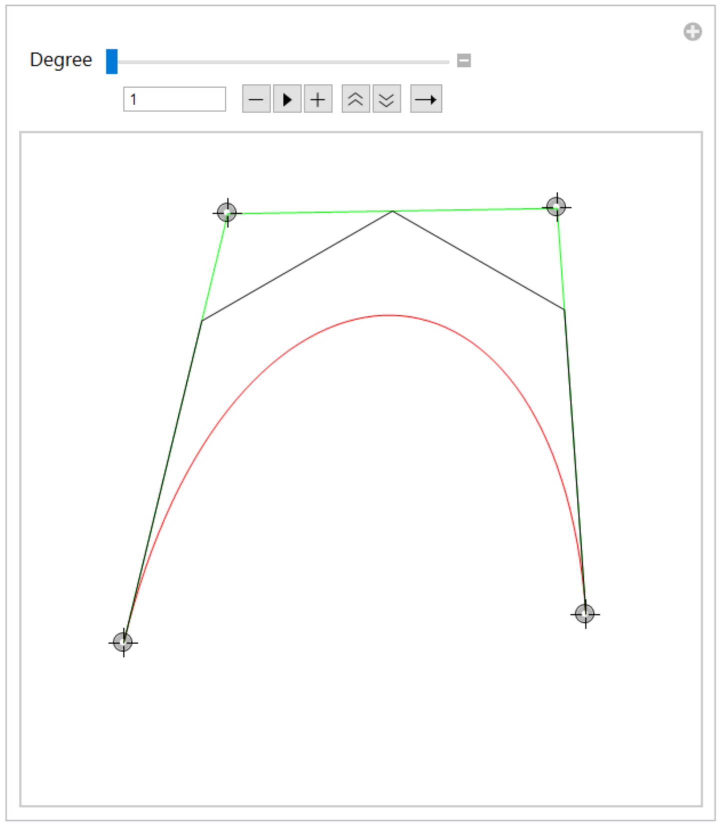
Новият полигон се получава от точките на стария чрез на части линейна интерполация при стойност на параметъра $\frac{i}{n+1}$. Следователно новият полигон лежи в изпъкналата обвивка на стария.

След много голям брой увеличавания на степента, за полигона получаваме следната

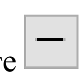
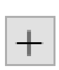
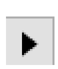
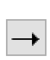
Теорема: $\lim_{n \to \infty} P^{(r)}(t) = b^{(n)}(t)$, където P са полигоните.

(тоест с увеличаването на броя на възлите на полигоните, постепенно се доближаваме до кривата)

2. Потребителски интерфейс



В активния прозорец са възможни следните действия:

- С натискане на **левия бутон на мишката + alt** добавяме нови точки.
- Със задържане на левия бутон върху точка можем да я местим.
- С натискане на **левия бутон на мишката + alt** върху вече съществуваща точка, можем да я премахнем.
- Чрез плъзгача „Degree“ можем постепенно да виждаме всеки следващ полигон.
- Действието на плъзгача може да бъде заместено с кликването на бутоните  и .
- Бутонът  пуска анимация на принтираща всички полигони.
- С бутоните  контролираме скоростта на анимацията.
- С бутонът  контролираме посоката на анимацията.

3. Реализация на алгоритъма

Двигателя на алгоритъма ще ни бъде функцията *elevation*, която приема за параметър списъка *initPTS* от точките на контролния полигон на създадената от на крива на Безие, след което записваме точките на новия полигон като елемент на списъка *newPol*, чиито елементи са списъци от точките на всеки новополучен полигон.

Използвам вградената функция *Module* ***3** , за да мога да третирам списъка от начални точки като локален за функцията, понеже те не са фиксирани, а се създават динамично от потребителя.

Имаме два вложени *For*-цикъла:

- Чрез **външния** създаваме с точна, съответстваща на степента, дължина списъка, в който ще запишем точките на всеки нов полигон (като за начало ги запълваме с нули), с помощта на функцията *Table* ***4**. След това чрез функцията AppendTo ***5** добавяме списъка от точки на всеки следващ полигон, като първият е създаден от потребителя интерактивно.
- Чрез **вътрешния** създаваме точките на текущия полигон. С първия If правим първата точка да съвпада с първата на първоначалния полигон, а с втория аналогично за последната точка. С третия If създаваме останалите точки като директно използваме формулата от математическото доказателство $b_i^{(1)} = \frac{i}{n+1} b_{i-1} + \left(1 - \frac{i}{n+1}\right) b_i$ (в случая съм положил i – 1 = k)

Накрая връщаме готовия списък от точките на новите полигони (сложил съм ги с горна граница 100 понеже разликата между следващите не е особено видима).

```
elevation[initPTS_List] := Module[{PTS}, PTS = initPTS;
  newPol = {};
  For[n = 3, n < 100, n++,
   AppendTo[newPol, Table[0, {x, 1, n + 2}]];
   If[n == 3, a = PTS, a = newPol[[n - 3]] ];

   For[i = 1, i <= Length[a] + 1, i++,
    k = i - 1;

    If[i == 1, newPol[[n - 2, i]] = a[[1]]];
    If[i == Length[a] + 1, newPol[[n - 2, i]] = a[[Length[a]]] ];
    If[i > 1 && i <= Length[a], newPol[[n - 2, i]] = (k / (n + 1)) * a[[i - 1]] + (1 - (k / (n + 1))) * a[[i]]]

   ]

  ]; newPol]
```

Втората главна функция е вградената Manipulate, която отговаря за интерфейса на програмата. Тя се състои от три компонента:

- Graphics, която ще комбинира графиките на кривата на Безие*[6], графиката на първия полигон и графиките на всички останали полигони (чрез Line, която просто приема за параметри точки и според тях рисува крива) (PlotRange определя големината на екрана).
- Списък, който съдържа началните точки (поставени от потребителя) (за да направя точките интерактивни съм използвал вградения елемент Locator) и вградена функция, която създава Locator директно, където сме кликнали.
- Списък съдържащ настройките на плъзгача.

```
Manipulate[
 Graphics[{{Red, BezierCurve[t]}, {Green, Line[t]}, {Line[func1[t][[u]]]}}, PlotRange → 2],
 {{t, {}}, Locator, LocatorAutoCreate → True},
 {{u, 1, "Degree"}, 1, 97, 1}]
```

*[3] Module приема локален параметър и извършва зададена от нас операция с него (и връща някаква стойност).

*[4] Table генерира списък от стойности на израза подаден като параметър (другият подаден параметър е списък съдържащ долната и горната граница на броя новосъздадени стойности).

*[5] AppendTo приема за параметри два елемента и към първия конкатенира втория.

*[6] BezierCurve е вградена функция която взима за параметри контролните точки на кривата на Безие и я чертае.

4. Използвана литература

- Лекции към курса „Компютърно геометрично моделиране“ на доц. Красимира Влъчкова
- Официалната документация на WolframMathematica https://reference.wolfram.com/language/