СОФИЙСКИ УНИВЕРСИТЕТ
„СВ. КЛИМЕНТ ОХРИДСКИ"

SOFIA UNIVERSITY
ST. KLIMENT OHRIDSKI

Курсова работа

на тема:

Реализиране на алгоритъма за намиране на cross plot на крива на Bezier

За курса:

Компютърно геометрично моделиране (CAGD)

Лектор: доц. Красимира Влъчкова

Изготвена от:

Светлин Иванов Попиванов

II курс, Софтуерно инженерство

София, 2020

Съдържание

Условие на задачата

При зададени точки от потребителя, първо да се начертае кривата на Bézier с тях като контролни точки, след това при желание на потребителя да се начертае cross plot-а на тази крива.

Работа с програмата

След стартирането на програмата, на прозореца ще излезе Декартова координатна система. Потребителят може да си взаимодейства с програмата само и единствено чрез първи квадрант. Там той може да задава/премахва контролните точки на кривата. Във втори квадрант ще бъде начертана y(t) функцията на Bezier, а в четвърти квадрант – x(t) функцията. Трети квадрант винаги стои празен. По-надолу са описани различните възможности, предоставени на потребителя:

1. ***При натискане на левия бутон на мишката върху първи квадрант*** потребителят създава нова контролна точка на мястото на курсора в момента на натискане. След всяка нова точка се чертае кривата на Bézier с обновените контролни точки (новата точка е добавена към предишните и така получаваме нов брой контролни точки и съответно различна крива на Bézier).

2. ***При натискане на десния бутон на мишката върху първи квадрант*** потребителят изтрива последната добавена контролна точка (ако такава съществува). След всяко такова изтриване, се чертае новата крива на Bézier с обновените контролни точки (точката се премахва от контейнера с контролни точки и така ще получим нова крива на Bézier).

3. ***При натискане на клавиша 'c'*** се начертава/скрива контролния полигон.

4. ***При натискане на клавиша 'p'*** се начертават/скриват контролните точки.

5. ***При натискане на клавиша 'd'*** се начертава/скрива cross plot-а на кривата на Bézier. Във втори квадрант ще бъде нарисувана/скрита y(t) функцията, а в четвърти квадрант ще бъде нарисувана/скрита x(t) функцията.

6. ***При натискане на клавиша '1'*** се намалява стойността на червения цвят с 0.1 в цвета на кривата на Bézier.

7. ***При натискане на клавиша ‘2’*** се намалява стойността на синия цвят с 0.1 в цвета на кривата на Bézier.

8. ***При натискане на клавиша ‘3’*** се намалява стойността на зеления цвят с 0.1 в цвета на кривата на Bézier.

9. ***При натискане на клавиша ‘4’*** се намалява стойността на червения цвят с 0.1 в цвета на контролния полигон.

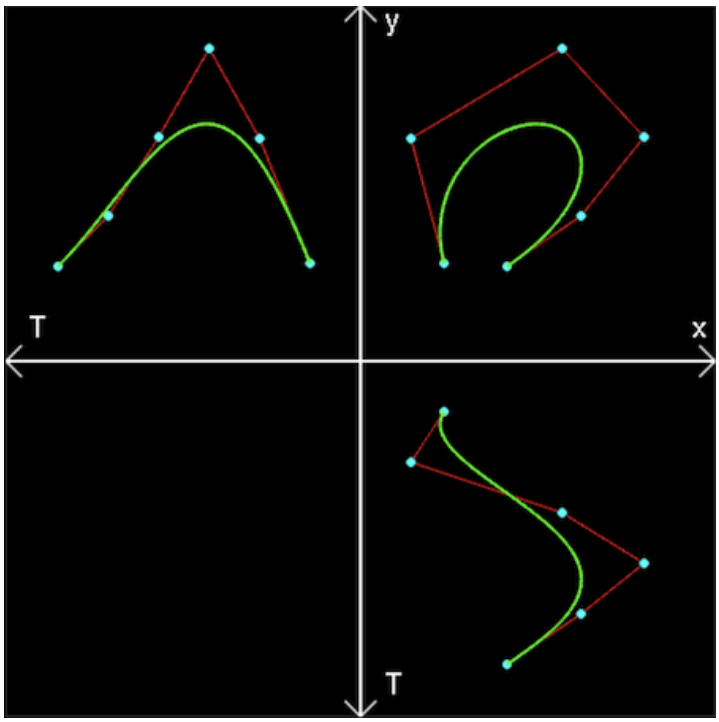
10. ***При натискане на клавиша ‘5’*** се намалява стойността на синия цвят с 0.1 в цвета на контролния полигон.

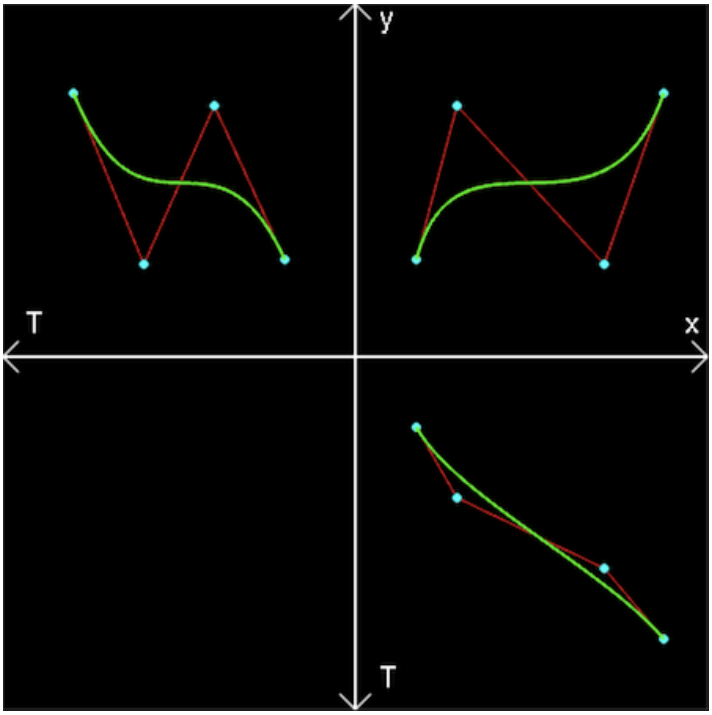
11. ***При натискане на клавиша ‘6’*** се намалява стойността на зеления цвят с 0.1 в цвета на контролния полигон.

Ще отбележим, че всички стойности на цветовете са измежду интервала [0,1], като ако някое от тях е 0 и се подаде заявка да се намали, той става 1.

Примерно изпълнение на програмата

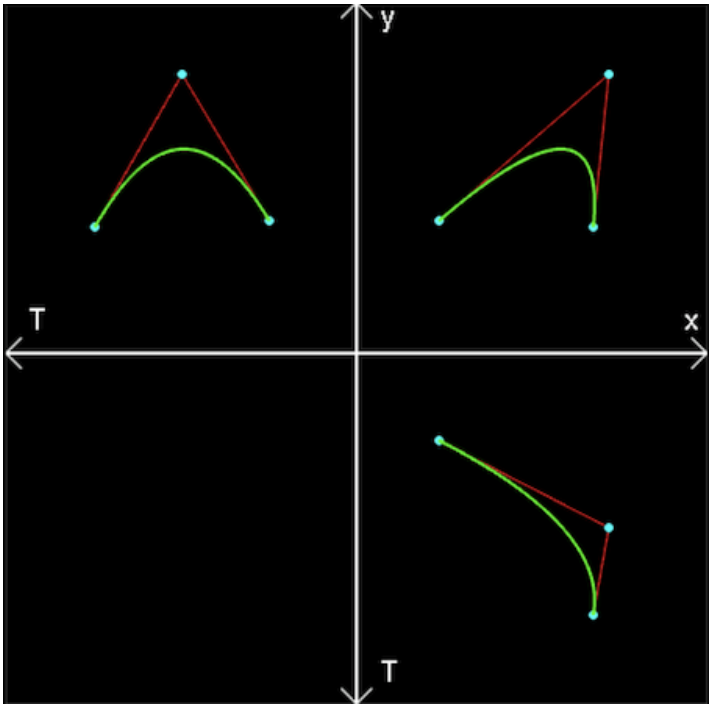
На следните картини са показани примерни криви на Bézier и техния cross plot:



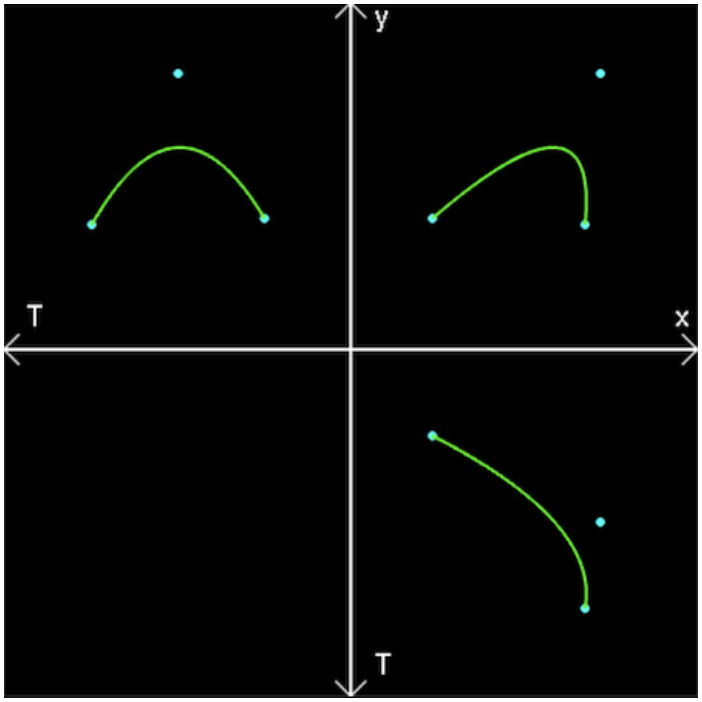


По-долните картини представят една крива на Bézier и нейния cross plot, но всяка картинка представя отделна функционалност:

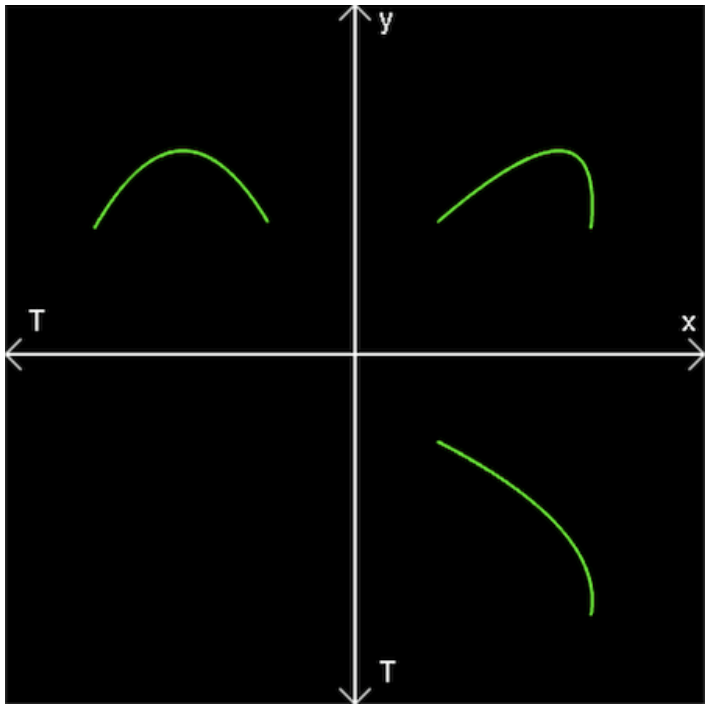
1. Потребителят е въвел три точки и е натиснал клавиша ‘d’, за да се начертае cross plot-а на кривата



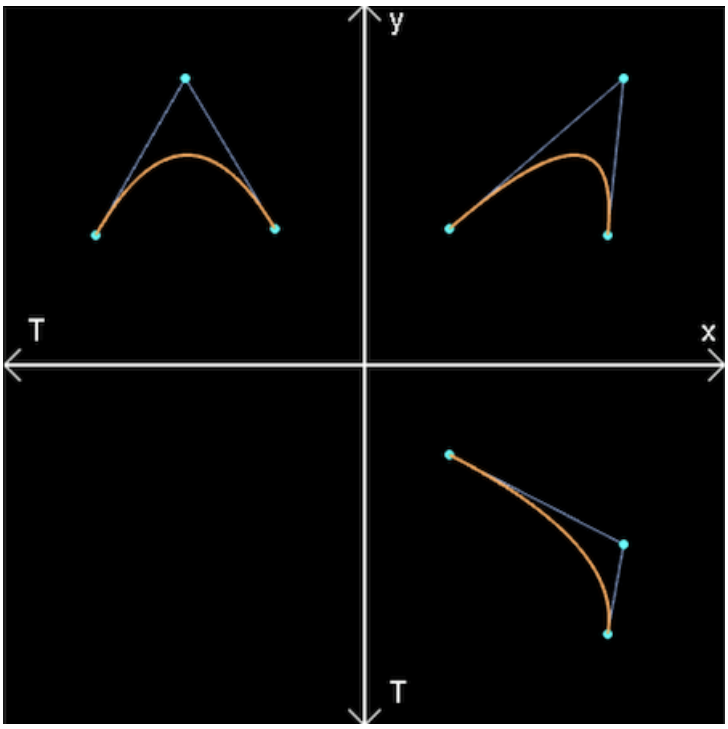
2. Потребителят е натиснал клавиша ‘c’, за да скрие контролните полигони



3. Потребителят е натиснал клавиша ‘p’, за да скрие контролните точки



4. Потребителят е натиснал клавишите ‘c’ и ‘p’ отново, за да се начертаят съответно контролните полигони и контролните точки на кривите. След това потребителят е променил цвета на контролните полигони и на кривите, използвайки клавишите от 1 до 6

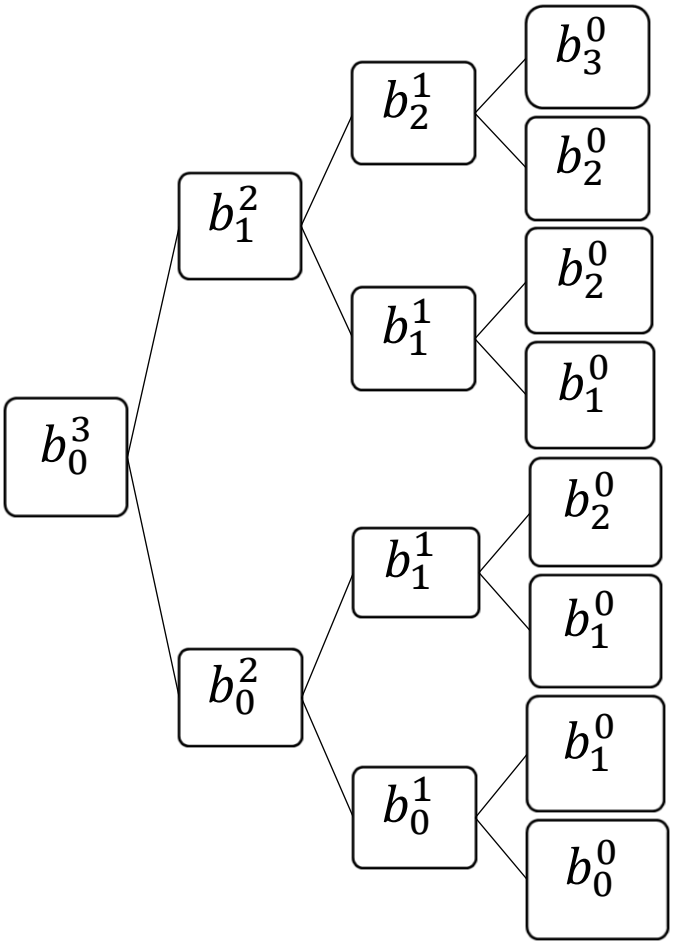


Основни функции:

❖ **<u>computeBezPt</u> –** това е една от основните функции в тази програма, защото чрез нея изчисляваме точка от кривата на Bézier при подадени контролни точки и дадена стойност **t**. Алгоритъмът, който е използван тук, е рекурсивният алгоритъм на *de Casteljau:*

$$b_i^r(t) = (1 - t)\, b_i^{r-1}(t) + t b_{i+1}^{r-1}(t),$$
$$r = 1,\ldots, n;\ i = 0,\ldots, n - r$$
$$b_i^0 = b_i$$

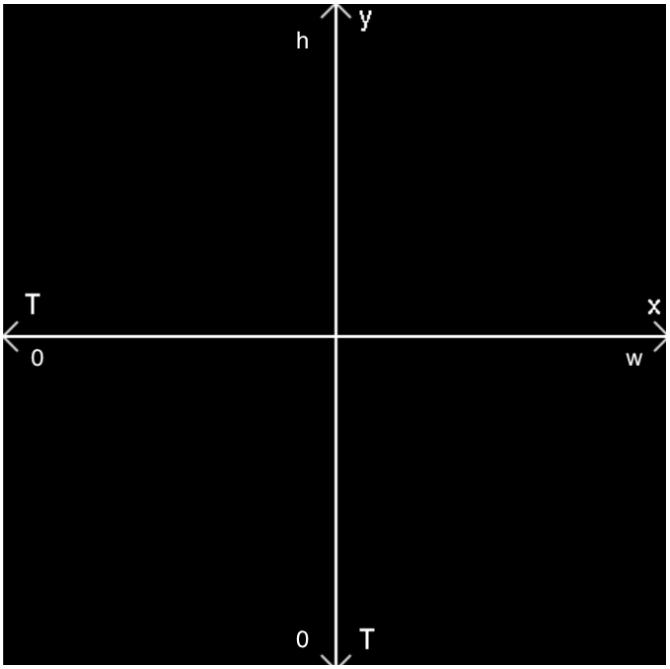
Във вектора pts съхраняваме контролните точки до даден момент. Нека векторът pts съхранява четири точки: b_0, b_1, b_2, b_3. На картинката долу е показано как рекурсията изчислява търсената точка b_0^3 при тези контролни точки и за някое **t**.



Първоначално функцията започва от b_0^3, като за да я изчисли са ѝ нужни точките b_1^2 и b_0^2. Затова функцията започва да изчислява b_0^2 и след това b_1^2. Този процес се повтаря за всички междинни точки, докато не се стигне до контролна точка (тоест r = 0). След като се стигне до последното ниво на горната йерархия (тоест се достигнат контролните точки), рекурсията започва да се връща и по този начин да смята стойностите на междинните точки. Така, когато функцията получи стойностите на точките b_1^2 и b_0^2, изчислява b_0^3 по гореописаната формула за някаква стойност на **t** и връща тази точка.

❖ **<u>computeBezier</u> –** Тази функция също е една от главните за тази програма, тъй като чрез нея се чертае кривата на Bézier. За тази цел се създава цикъл, в който параметърът t ще пробягва интервала [0,1], като на всяка итерация след първата t = t + 0.02 и този параметър се подава във функцията **computeBezPt**. По този начин на всяка итерация получаваме точка от кривата, която се намира съвсем близо до предишната, като я свързваме с нея и функцията начертава правата между тези две точки. Ясно е че при t = 0 точката от кривата на Bézier ще е именно b_0, а при t = 1 точката ще е именно b_n. Така след 50 на брой итерации, ще сме получили 50 на брой прави, които представляват

кривата на Bézier. Поради това, че самата крива се представя от много на брой прави, тя изглежда гладка. Именно затова параметърът t се увеличава с малка стойност.



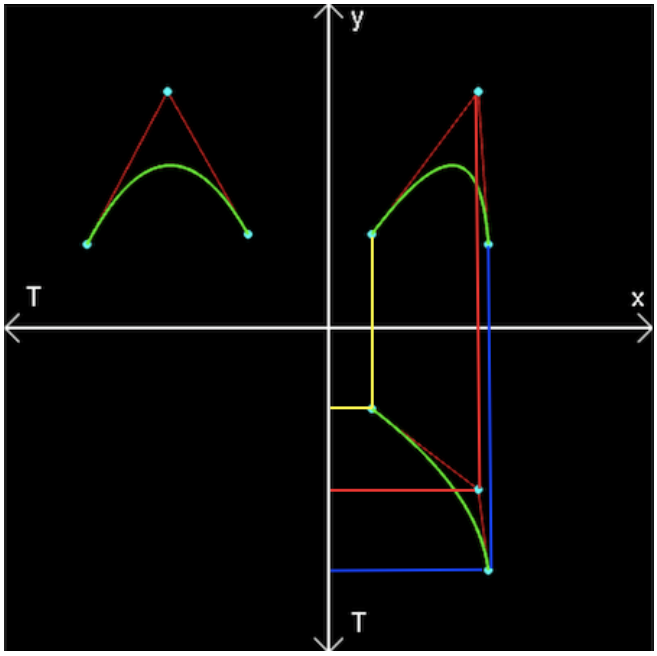
❖ **<u>computeXFuncPts</u> –** Тази функция изчислява координатите на контролните точки на функционалната крива x(t). Алгоритъмът е:

За всяка контролна точка на функционалната крива:

- стойността на x координатата е същата като съответната от i-тата контролна точка (i-та по ред на поява) на кривата на Bézier.
- y координатата се изчислява по следния начин: Разбива се интервала на четвърти квадрант (за y), който е [h / 2, 0], където h е дължината на прозореца (виж картинката горе), чрез броя на контролните точки на кривата. По този начин получаваме n + 1 равни подинтервала, където n е броят контролни точки. Краят на всеки един интервал без последния съответства на y-координатата на i-тата контролна точка на функционалната крива. Математически, това може да се изрази чрез следната формула:

$$y_i = (h / 2) – i * ((h / 2) / (n + 1)), \text{ за } i = 1, . . ., n$$

На картинката по-долу е показано как се получават контролните точки на функционалната крива x(t) при дадена крива на Bézier. С жълт цвят се означава как се получава първата контролна точка, с червен – следващата и със син – последната:



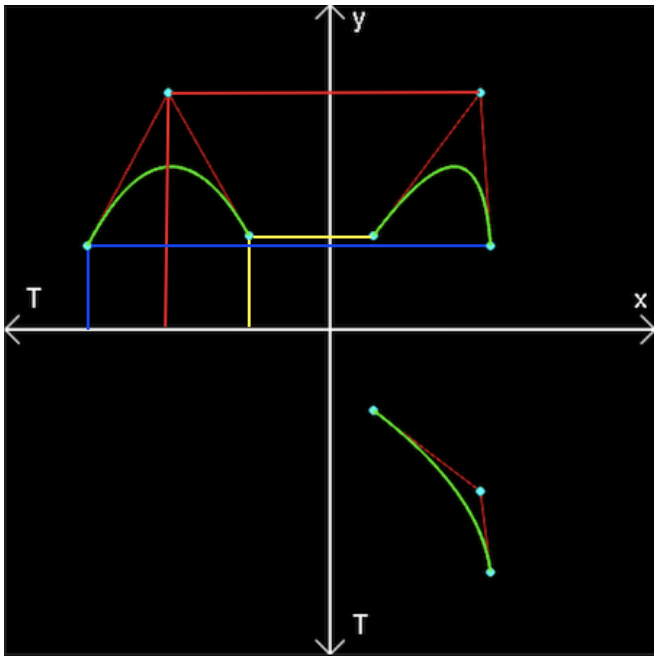
❖ **<u>computeYFuncPts</u> –** Тази функция изчислява координатите на контролните точки на функционалната крива y(t). Алгоритъмът е:

За всяка контролна точка на функционалната крива:

- стойността на y координатата е същата като съответната от i-тата контролна точка (i-та по ред на поява) на кривата на Bézier.
- x координатата се изчислява по следния начин: Разбива се интервала на втори квадрант (за x), който в случая е [0, w / 2], където w е ширината на прозореца (виж картинката на миналата страница), чрез броя на контролните точки на кривата. По този начин получаваме n + 1 равни подинтервала, където n е броят контролни точки. Краят на всеки един интервал без последния съответства на x-координатата на i-тата контролна точка на функционалната крива. Математически, това може да се изрази чрез следната формула:

$$x_i = (w / 2) – i * ((w / 2) / n + 1), \text{ за } i = 1, . . ., n$$

На картинката по-долу е показано как се получават контролните точки на функционалната крива y(t) при дадена крива на Bézier. С жълт цвят се означава първата получена точка, с червен - следващата и със син – последната:



❖ **drawCoordinateSystem –** Тази функция чертае координатната система, като първо чертае правите, а след това стрелките. Координатните оси винаги се позиционират в средата на прозореца по ширина и дължина, независимо какъв е размера на прозореца.

❖ **drawBezier –** Тази функция чертае текущата крива на Bézier (използвайки **computeBezier**), координатната система (използвайки **drawCoordinateSystem**) и ако са поискани от потребителя, се чертаят cross plot-а, контролния полигон и контролните точки на кривата/кривите. Тази функция се извиква винаги, когато има промяна в броя на контролните точки или промяна в начина на показване (различен цвят, скриване на детайли, промяна на размера на прозореца и т.н.). Ако потребителят е избрал да се нарисува cross plot-а на кривата, то тази функция използва **computeXPts** и **computeYPts**, за да се сдобие с контролните точки на функционалните криви и изпраща тези точки на функцията **computeBezier**, която ще начертае двете криви.

Използвана литература

Лекции към курса за бакалаври ,,Компютърно геометрично моделиране”

Използван софтуер

Microsoft Visual Studio - https://visualstudio.microsoft.com/vs/

The OpenGL Utility Toolkit - http://www.opengl.org/resources/libraries/glut/