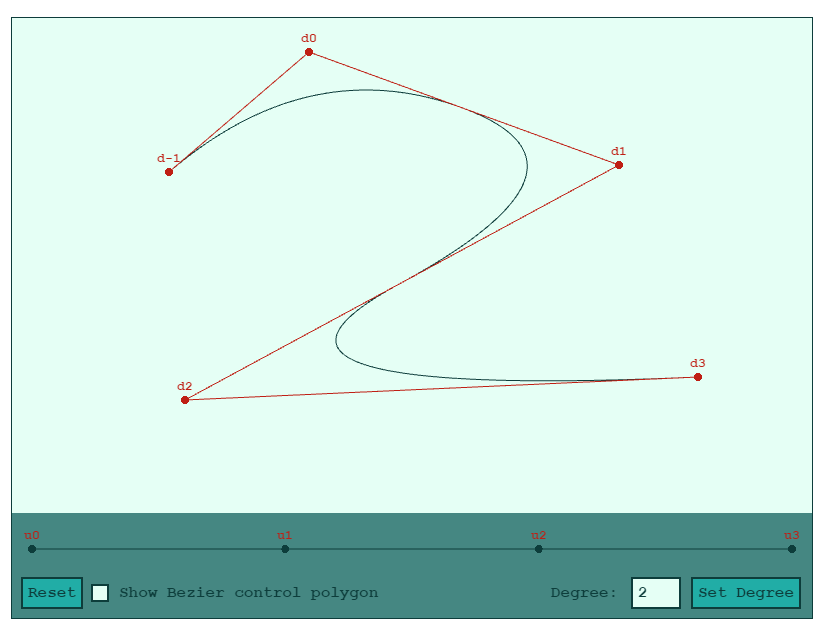
**Квадратичен сплайн в 2D**

*Изабела Недялкова*

*Курсов проект по CAGD*

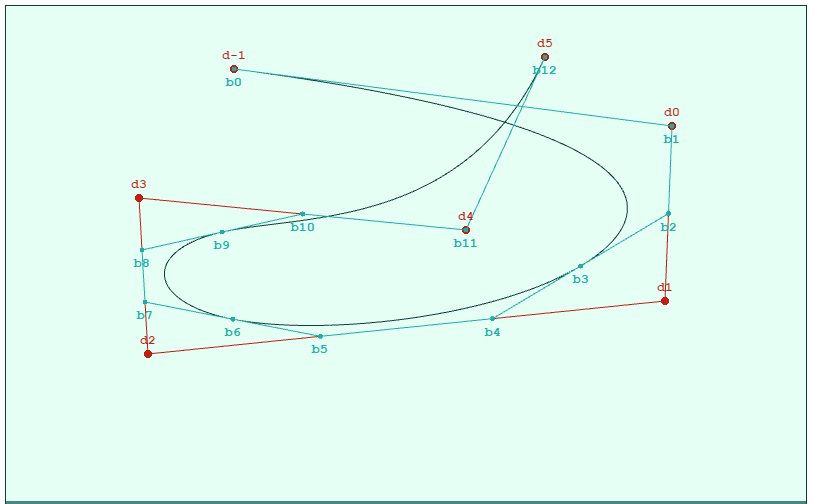
**Основна цел**

Целта на приложението е по зададени контролни точки на Де Бор в двумерното пространство и зададени възли да бъде изчертана в работната площ сплайн крива от 2-ра степен (като допълнителна функционалност е предоставено поле за промяна на степента на кривата, тъй като реализацията позволява изчертаването на крива от коя да е степен n).



**Функционалности**

*Работна площ* - Могат да бъдат добавяни неограничен брой нови контролни точки на Де Бор, както и вече съществуващи точки да бъдат премествани (и в реално време да бъде пречертана сплайн кривата). Също така е налична функционалност за изтриване на селектирана контролна точка на Де Бор (**di**).



*Контролен панел -* В долната част на екрана има контролен панел, в който е показано отношението на интервалите между възлите, като стойност по подразбиране е 1:1. С влачене на възлите тези отношения могат да бъдат променяни. Под плъзгача има поле за задаване степента на кривата със стойност по подразбиране 2 (спрямо заданието) и бутон, който потвърждава въвеждането на степента. Наличен е и бутон Reset за зачистване на работната площ. Възможно е да се сложи отметка в полето Show Bezier control polygon (за квадратични и кубични криви), в резултат на което, да се визуализира контролния полигон на Безие с неговите контролни точки.



**Технологии**

За реализацията на приложението са използвани следните технологии:

* JavaScript
* HTML5
* CSS3
* WebGL

**Работа с програмата**

* Приложението се статрира в браузъра с двоен клик на index.html файла
* С клик на ляв бутон на мишката върху работната площ се добавя контролна точка на Де Бор
* С клик на ляв бутон, задържане и влачене на мишката върху избрана контролна точка на Де Бор се извършва преместване на точката
* С клик на десен бутон на мишката върху избрана контролна точка на Де Бор, тя се изтрива
* С клик на ляв бутон, задържане и влачене на мишката върху избран възел (knot) ui се извършва преместване на възела (като позицията на първия и последния възел е фиксирана), което от своя страна променя отношенията между интервалите
* С клик на бутон Reset се зачиства цялата работна площ
* С въвеждане на стойност в полето и клик на бутон Set Degree се променя степента на кривата
* С маркиране/отмаркиране на полето Show Bezier control polygon се показва/скрива контролния полигон на Безие с неговите контролни точки (за квадратични и кубични криви)

**Структура на приложението**

Приложението се състои от един JavaScript файл, с име app.js, в който се намира цялата логика, един HTML файл на име index.html, съдържащ два canvas-а (един за изчертаване на кривата с WebGL и още един насложен върху него като втори слой, в който се визуализират етикетите с имената на контролните точки и възлите), бутон Reset и поле за въвеждане на степен на кривата и един CSS файл с име styles.css, който съдържа стиловете.

Интеракцията на потребителя се реализира чрез следенето за няколко потребителски действия:

* Движение на мишката – ако няма натиснат бутон на мишката, се изчислява дали има контролна точка на Де Бор или възел в достатъчно малък радиус и ако това е така, съответната контролна точка или възел се маркират като селектирани. Ако е натиснат левия бутон и има вече селектирана контролна точка или възел, то те се транслират според новите координати на курсора.

textCanvas.addEventListener("mousemove", function(e) {

var x = 2 \* e.offsetX / canvas.width - 1

var y = 2 \* (canvas.height - e.offsetY) / canvas.height - 1

if (e.buttons === 0) {

selectNearControlPoint(x, y)

selectNearKnot(x, y)

} else if (e.buttons === 1) {

translateSelectedControlPoint(x, y)

slideKnot(x)

}

})

* Натискане на бутон на мишката
  + С ляв бутон – ако няма селектирана точка, се извиква метод за добавяне на нова контролна точка на Де Бор на съответните координати.
  + С десен бутон – ако има селектирана точка, се извиква метод за изтриване на съответната контролна точка на Де Бор.

textCanvas.addEventListener("mousedown", function(e) {

var x = 2 \* e.offsetX / canvas.width - 1

var y = 2 \* (canvas.height - e.offsetY) / canvas.height - 1

if (e.buttons === 1) {

addControlPoint(x, y)

} else if (e.buttons === 2) {

deleteSelectedControlPoint()

}

})

* Отпускане на бутон на мишката – зачистват се селектираната контролна точка на Де Бор или възел

textCanvas.addEventListener("mouseup", function(e) {

selectedControlPointIndex = null

selectedKnotIndex = null

})

**Математическа обосновка**

За реализация е използван алгоритъма на Де Бор. Той е бърз и стабилен алгоритъм за изчисление на сплайн криви в B-Spline форма. Той е генерализация на алгоритъма на Кастелжо за криви на Безие. Алгоритъмът е създаден от Карл Р. Де Бор. Опростени и потенциално по-бързи варианти на алгоритъма на Де Бор са създавани, но те имат сравнително по-малка стабилност.

Чрез алгоритъма на Де Бор се конструира крива, чиято форма се описва от поредица от **p** точки **d0, d1, … , dp-1**, които иамт ролята на контролен полигон. Кривата може да бъде описана чрез функция **s(x)** на параметър **x**. За да мине кривата през редицата от точки, тя трябва да удовлетворява **s(u0) = d0, … , s(up-1) = dp-1,** но в общияслучай е достатъчно кривата да апроксимира контролния полигон на Де Бор. Предполага се, че **u0, … , up-1**са дадени заедно с **d0, d1, … , dp-1**.

Сплайнът е крива, която се състои от множество полиноми от **n**-та степен. Това означава, че за всеки интервал **[ui, ui+1)**, кривата трябва да е равна на полином от степен най-много **n**. Когато полиноми от интервали **[ui-1, ui)** и **[ui, ui+1)** се срещат в точка **ui**, те трябва да има една и съща стойност в тази точка и производните им да бъдат еднакви (за да се осигури гладкостта на кривата).

*Алгоритъм*

*,*

Където базисната функция (basis function) **N** се изчислява рекурсивно

,

а дъното на рекурсията е при **n = 0**

*Реализация в кода*

Итеративно за всички контролни точки на Де Бор **di­** се изчислява рекурсивно базисната функция **,** умножава се по **di­** и се сумира към досегашните стойности на **x** и **y** координатите на резултата.

var s = function(x) {

var result = {

x: 0.0,

y: 0.0

}

for (var i = 0; i < d.length; i++) {

var b = N(x, i, degree)

result.x += d[i].x \* b

result.y += d[i].y \* b

}

return result

}

Самата базисна функция я изчисляваме като първо винаги проверяваме дали е изпълнено условието за дъно на рекурсията и ако не е използваме рекурсивната формула, описана по-горе, като тук за яснота на кода е разделена в две променливи **first** и **second,** които може да имат стойност NaN (not a number) при деление на 0, поради спецификата на JavaScript и затова, преди да бъдат сумирани, се проверява, ако имат такава стойност, да бъдат занулени, за да останат коректни изчисленията.

var N = function(x, i, n) {

if (n === 0) {

if (x >= u[i] && x < u[i + 1]) {

return 1

} else {

return 0

}

}

var first = ((x - u[i]) / (u[i + n] - u[i])) \* N(x, i, n - 1)

var second = ((u[i + n + 1] - x) / (u[i + n + 1] - u[i + 1])) \* N(x, i + 1, n - 1)

if (isNaN(first)) {

first = 0

}

if (isNaN(second)) {

second = 0

}

return first + second

}