Кривата на Безие и нейната първа поляра

курсов проект по Компютърно геометрично моделиране (CAGD)

Лектор: доц. Красимира Влъчкова

Цветомир Стайков - 1MI0800469

2 курс, Компютърни науки

12 февруари 2025 година

Съдържание

1. Въведение

Настоящият проект представя двуизмерна крива на Безие и нейната първа поляра. За генерирането на кривите се използва алгоритъмът на de Casteljau. Реализирани са стандартни функции за създаване на криви. Програмата работи в уеб браузър и е тествана на настолни компютри.

2. Структура и технологии

Проектът представлява уеб приложение, разработено с HTML, CSS и JavaScript, като използва панел за рисуване, известен като Canvas, за визуализация на кривите. Изборът на тези технологии позволява лесна адаптация и използване на програмата на различни устройства.

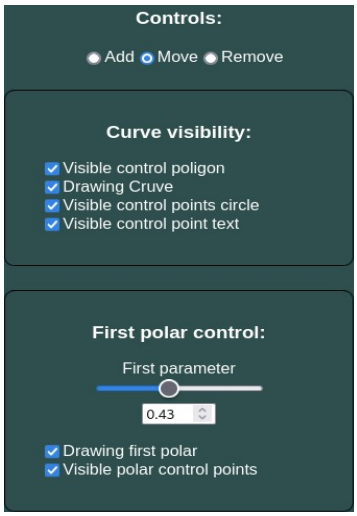
Компресирана структура на проекта

- *CAGD-1MI0800469.html* – Целият проект, събран в един файл.

Оригинална структура на проекта

- *index.html* – Начална страница на сайта.
- *styles.css* – Дизайн на интерфейса.
- *Interface.js* – Функции за създаване, управление и изтриване на контролните точки, както и визуализация на крайната крива.
- *Algorithms.js* – Алгоритми за изчисление на кривата и нейната първа поляра.
- *Debug.js* – Помощни функции за извеждане на информация и грешки.

3. Интерфейс



Потребителският интерфейс се състои от две основни части:

- Ляв панел за управление - Панелът има настройки за създаване на крива. Позволява добавяне, преместване и премахване на контролни точки. Разполага с отметки за промяна на видимостта на контролния полигон, точките, кривата, надписите, както на полярата и нейните точки. Предвиден е слайдер за управление на първия параметър на полярата. До него има текстово поле, което приема числови

стойности за по-прецизен контрол на параметъра с до два знака след десетичната запетая. Въведено е ограничение в интервала [0,1].

За управление на контролните точки в горната част на панела има три режима – Add, Move и Remove. При стартиране на програмата по подразбиране е активиран режимът Add.

Ако екранът не е достатъчно голям, чрез използване на скрол колелото на мишката, панела може да бъде разгледан изцяло.

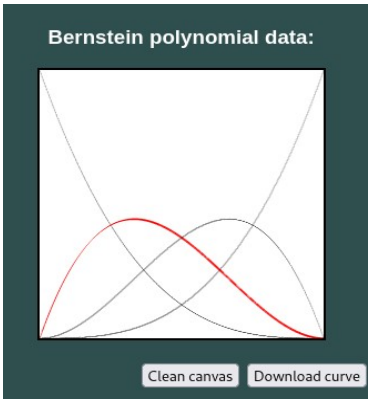
- Десен панел за рисуване (Canvas) – След избиране на една от настройките за контролните точки чрез натискане на десен бутон на мишката може да се манипулират точките върху този панел. Когато курсорът е върху контролна точка в режим Move или Remove, тя се огражда в червено, сигнализирайки, че може да бъде избрана. Панелът се преразмерява при отваряне или презареждане на страницата.

Основни функции върху контролните точки

- Add – Позволява добавяне на нови точки, като новодобавената става следващата в реда. Точка се добавя чрез натискане на левия бутон върху панела за рисуване.
- Move – Като курсорът се намира върху контролна точка, тя се огражда с червен кръг. При натискане и задържане на левия бутон този кръг изчезва, а контролната точка ще следва движението на мишката.
- Remove – Работи аналогично на Move, но вместо да премества контролната точка, я изтрива при натискане на левия бутон.

Допълнително

- Програмата позволява изтеглянето на нарисуваната крива без фон, чрез бутон „Download curve“, най-долу в контролния панел.
- В долната част на контролния панел са представени полиномите на Бернщайн за настоящата крива. Когато курсорът е върху контролна точка в режим Move или Remove, съответният коефицент върху графа се маркира в червено.



4. Алгоритъм

За пресмятане на кривата е използван алгоритъм на de Casteljau. След добавянето на контролни точки, програмата автоматично генерира кривата. Има две функции за чертане на кривата и полярата, като има реализирани три подфункции за изчисленията.

Главните функции са извикани от интерфейса и те проверяват дали има достатъчно контролни точки, след което чертаят кривите със стъпка 0.001.

Две от помощните подфункции използват алгоритъма на de Casteljau за изчисление на междинните точки. Те връщат като резултат нужната точка за чертаене на кривата. Подадените аргументи са променливата t и контролни точки.

Алгоритъм за чертане на кривата:

```
function drawCurve(){
    if(controlPoints.length <=1){
        return;
    }
    ctx.lineWidth = 2;
    ctx.strokeStyle = "grey";

    for(let i =0;i<=1;i+=0.001){
        pos =calculateAtPos(i,controlPoints);

        ctx.beginPath();
        ctx.moveTo(pos.x, pos.y);

        pos =calculateAtPos(i+0.001,controlPoints);

        ctx.lineTo(pos.x, pos.y);
        ctx.stroke();
    }

    ctx.strokeStyle = "black";
}
```

```
function calculateAtPos(t,LocalControlPoints){
    let count = LocalControlPoints.length;

    let intermediatePoints = [LocalControlPoints]

    for(let i = 0;i<count-1;++i){
        let prev = intermediatePoints[i];
        let newPoints = new Array(prev.length-1);

        for(let j =0;j<prev.length-1;++j){
            let first = prev[j];
            let second = prev[j+1];
            newPoints[j] = {
                x: (1-t)* first.x + t * second.x,
                y: (1-t)* first.y + t * second.y
            }
        }
        intermediatePoints.push(newPoints);

    }

    return intermediatePoints[intermediatePoints.length-1][0];
}
```

Въпреки че помощната функция на полярата използва същите изчисления както на обикновена крива, тя допълнително получава променливата t_1. Тази променлива е подадена от интерфейса към главната функция и после към подфункцията, която изчислява полярната крива. Тя замества t при първоначалното изчисляване, след което продължава по стандартния алгоритъм.

Алгоритъм за чертане на полярата:

```
function drawFirstPolar(t0){

    if(t0 < 0 || t0 > 1){
        DebugError("Polar t0 is out of [0,1] bound");
        return;
    }

    if(controlPoints.length <=2){
        return;
    }

    for(let i =0;i<=1;i+=0.001){
        let pos =calculateAtPosFirstPolar(t0,i,controlPoints);

        ctx.beginPath();
        ctx.moveTo(pos.x, pos.y);

        pos =calculateAtPosFirstPolar(t0,i+0.001,controlPoints);

        ctx.lineTo(pos.x, pos.y);
        ctx.lineWidth = 2;
        ctx.strokeStyle = "green";
        ctx.stroke();
    }


    ctx.strokeStyle = "black";
}
```

```
function calculateAtPosFirstPolar(t0,t,LocalControlPoints){
    let count = LocalControlPoints.length;

    let intermediatePoints = [LocalControlPoints]

    for(let i = 0;i<count-1;++i){
        let prev = intermediatePoints[i];
        let newPoints = new Array(prev.length-1);
        for(let j =0;j<prev.length-1;++j){

            let first = prev[j];
            let second = prev[j+1];

            if(i == 0){
                newPoints[j] = {
                    x: (1-t0)* first.x + t0 * second.x,
                    y: (1-t0)* first.y + t0 * second.y
                }
            }else{
                newPoints[j] = {
                    x: (1-t)* first.x + t * second.x,
                    y: (1-t)* first.y + t * second.y
                }
            }
        }
        intermediatePoints.push(newPoints);
    }
    return intermediatePoints[intermediatePoints.length-1][0];
}
```

Има отделна функция, която също приема t_1 и изчислява веднъж алгоритъма на de Casteljau и връща точките { $b_0^1(t_1)$, ... $b_{n-1}^1(t_1)$ }, където n е броят на контролните точки. Резултатът се изобразява на контролния полигон и може да се разгледа като контролните точки на полярната крива.

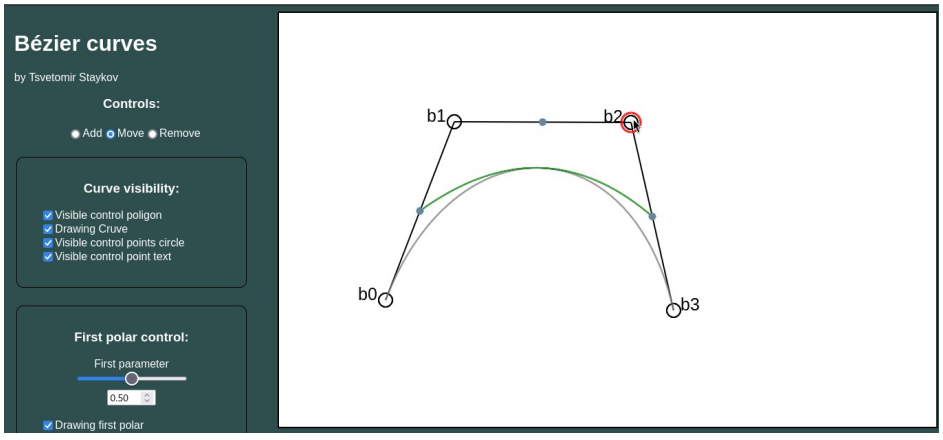
```
function PolarControlPoints(t0,LocalControlPoints){

    let count = LocalControlPoints.length;
    let PolarControlPoints = new Array(count-1);

    for(let j =0;j<count-1;++j){

        let first = LocalControlPoints[j];
        let second = LocalControlPoints[j+1];

        PolarControlPoints[j] = {
            x: (1-t0)* first.x + t0 * second.x,
            y: (1-t0)* first.y + t0 * second.y
        }

    }

    return PolarControlPoints;
}
```

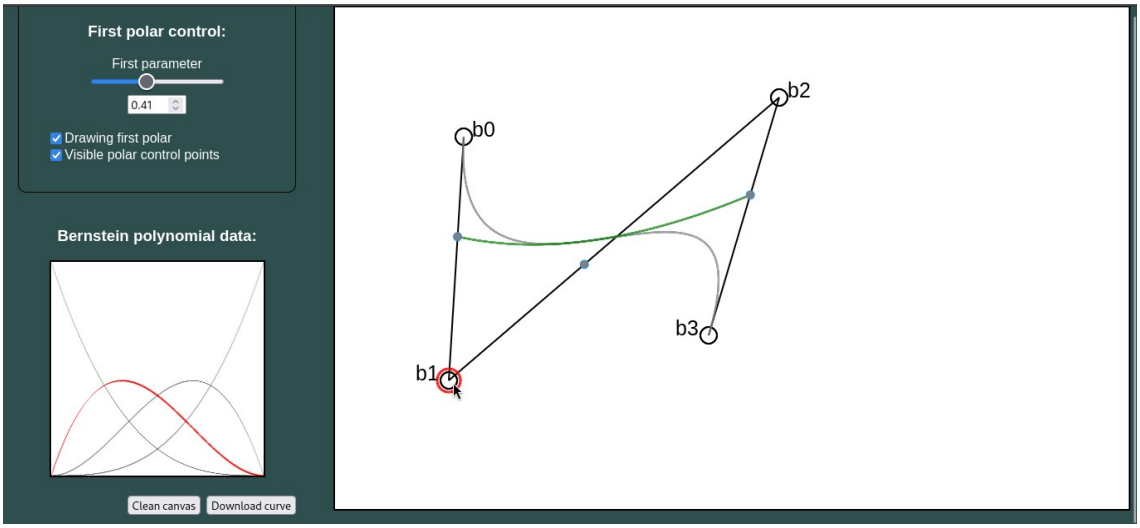
```
if(document.getElementById("DrawPolarControlPointsCheckbox").checked){
  let polarControlPoints = PolarControlPoints(t0,controlPoints);
  for(let i = 0; i<polarControlPoints.length; ++i){

      ctx.beginPath();
      ctx.arc(polarControlPoints[i].x,
              polarControlPoints[i].y,
              5, 0, 2 * Math.PI);
      ctx.fillStyle = "gray";
      ctx.fill();
      ctx.lineWidth = 1;
      ctx.strokeStyle = "DodgerBlue";
      ctx.stroke();
  }
```

```
function RedrawBernsteinGraph(){
  let count = controlPoints.length;
  for(let i = 0;i<count; ++i){

    if(currentSelectedPoint == i){
      graphCtx.lineWidth = 2;
      graphCtx.strokeStyle = "red";
    }else{
      graphCtx.lineWidth = 1;
      graphCtx.strokeStyle = "black";
    }
    let translate = 256;

    let width =  BernsteinPolynomialGraph.width;
    let height = BernsteinPolynomialGraph.height;
    let step = 0.2;
    graphCtx.beginPath();
    for(let t= 0;t<= 1;t+=0.001){
      let data =  binomial(count-1,i)
                  * Math.pow(1-t,i)
                  * Math.pow(t,count-1-i);

      let x = width-(t*translate);
      let y = height-(data*translate);
      graphCtx.moveTo(x,      y);
      graphCtx.lineTo(x+step, y+step);
    }
    graphCtx.stroke();
  }
}
```

Въпреки че полиномите на Бернщайн не се използват директно в алгоритмите, те са представени в малък граф като допълнителна информация в контролния панел.

5. Снимки на проекта





6. Краен продукт

Крайният продукт е един HTML файл, който съдържа всичко необходимо за функционирането на проекта без нужда от интернет и може да бъде пуснат през стандартните браузъри, които поддържат функционалността Canvas.

Работеща версия на проекта е качена на сайта: eguardian.dev/CAGD

Кодът е също качен в Github.

7. Използвани ресурси

- Записки по курса Компютърно геометрично моделиране (CAGD) на лектор доц. Красимира Влъчкова, София, 2024 г.
- Mozilla’s Javascript canvas tutorial/documentation - https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial

Допълнителни връзки

- https://www.eguardian.dev/CAGD/
- GitHub https://github.com/eGuardianDev/University-Work/tree/main/(%202%20)%20second%20Year/(%203%20)%20%20Third%20semester/CAGD/