Ходограф и крива на Безие

Курсов проект по Компютърно геометрично моделиране
Лектор: доц. Красимира Александрова

Александър Девинизов - 2MI0800684
Компютърни науки, 2 курс

Съдържание

1 Въведение

Настоящия проект реализира интерактивна визуализация на кривата на Безие и нейния ходограф (производна) с помощта на WebGL. Потребителят може да добавя, премества и изтрива контролни точки, като в реално време се наблюдава влиянието върху формата на кривата и нейния ходограф. Реализирана е визуализация на алгоритъма на de Casteljau за изчисляването на точки върху кривата.

1.1 Единен HTML файл

- **CAGD-2MI0800684.html** – основният файл на проекта

2 Технологии

Проектът представлява уеб приложение, разработено с HTML, CSS, JavaScript и WebGL. За визуализацията се използват HTML5 Canvas елементи, в които се инициализира WebGL контекст за рисуване. Всички графики се извеждат чрез WebGL рендиране, което осигурява висока производителност чрез директно използване на графичния процесор.

3 Интерфейс

Потребителският интерфейс е изчистен и интуитивен, като основно се състои от меню с бутони и панел за визуализация на изменението на променливата t в интервала $[0, 1]$.

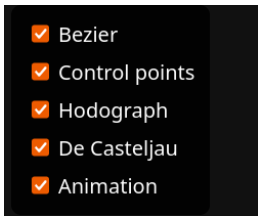
3.1 Основни функции върху контролните точки

- **Add** – Позволява добавяне на контролни точки в лявото платно чрез натискане на левия бутон на мишката.
- **Move/Drag** – Позволява преместване на контролни точки чрез задържане на левия бутон на мишката върху съответната точка.
- **Remove** – Позволява премахване на контролна точка чрез натискане на десния бутон на мишката.

Добавянето и премахването на контролни точки е свързано: следи се кой бутон ще бъде натиснат. Ако бъде натиснат левия бутон, ще бъде добавена нова точка на мястото на курсора. Ако се натисне десния бутон на мишката върху някоя точка, тя ще бъде премахната от полето.

3.2 Функционалности

Бутоните до лявото платно предоставят възможност на потребителя да включва или изключва визуализацията на описания обект. След презареждане на страницата, бутоните се връщат към началното включено състояние.




```
let show = {bezier:true, control:true, hodo:true, deCasteljau:
    true, animation:true};
```

Долният панел съдържа три основни възможности:

- **Слайдер** – позволява визуализация на алгоритъма на de Casteljau за дадена стойност на t в интервала $[0, 1]$;
- **Анимация** – бутоните *Play* и *Reset* стартират анимация на алгоритъма на de Casteljau или го връщат в началното му състояние;
- **Clear** – изтрива както кривата, така и ходографа от платната.

4 Работа с WebGL

WebGL е инструментът, който позволява визуализация на точки, линии и криви в проекта. За улеснение са реализирани две основни помощни функции, чрез които чертаенето на обектите става бързо и ефективно.

4.1 initializeGL

Функцията `initializeGL` създава платното, на което се изчертават кривата на Безие и ходографа. Тя инициализира WebGL контекст и създава шейдърна програма, състояща се от vertex и fragment шейдър.

- **Vertex шейдър:** определя позицията и размера на върховете.
- **Fragment шейдър:** отговаря за оцветяването на примитивите.

Получената програма се използва за изчертаване на всички обекти в приложението.

```
function initializeGL(canvas) {
  const gl = canvas.getContext("webgl");
  gl.clearColor(1,1,1,1); // RGBA

  const vertexShader = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(vertexShader, `
    attribute vec2 p; // 2D coordinates
    void main() {
      gl_Position = vec4(p, 0.0, 1.0); // set vertex position
      gl_PointSize = 8.5; // point size when rendering
    }
  `);
  gl.compileShader(vertexShader);

  const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
  gl.shaderSource(fragmentShader, `
    precision mediump float;  // floating point
    uniform vec3 c;  // fragment color
    void main(){
      gl_FragColor = vec4(c,1.0); // set fragment color will full
          opacity
    }
  `);
  gl.compileShader(fragmentShader);

  const prog = gl.createProgram();
  gl.attachShader(prog, vertexShader);
  gl.attachShader(prog, fragmentShader);
  gl.linkProgram(prog);

  return {gl, prog};
}
```

4.2 drawGL

Функцията `drawGL` отговаря за визуализацията на геометрични примитиви чрез WebGL. Тя създава буфер с координати на върховете, подава данните към шейдърната програма и извършва изчертаване в зависимост от избрания режим (например `gl.LINE_STRIP`) и зададения цвят.

```
function drawGL(ctx, points, mode, color) {
  if(points.length == 0) return;
  const {gl, prog} = ctx;

  const buffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(points.flat()),
       gl.STATIC_DRAW);

  gl.useProgram(prog);
  const position = gl.getAttribLocation(prog, "p");
  gl.enableVertexAttribArray(position);
  gl.vertexAttribPointer(position, 2, gl.FLOAT, false, 0, 0);

  gl.uniform3fv(gl.getUniformLocation(prog, "c"), color);
  gl.drawArrays(mode, 0, points.length);
}
```

5 Алгоритми

5.1 de Casteljau

Функцията `getDeCasteljauPoints` изчислява всички междинни точки на кривата на Безие за даден параметър $t \in [0, 1]$. Процесът започва с оригиналните контролни точки, които формират първото ниво. За всяко следващо ниво се извършва линейна интерполация между съседни точки от предишното ниво според формулата:

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

Този процес се повтаря, докато на последното ниво не остане единствена точка, която съответства на точка от кривата за зададеното t. Функцията връща масив, съдържащ всички изчислени нива на кривата.

```
function getDeCasteljauPoints(points, t){
  const levels = [];
  const temp = points.map(p => [...p]);

  // First level is the original points
  levels.push(temp.map(p => [...p]));

  // Calculate intermediate points
  for(let level = 1; level < points.length; level++) {
    const prevLevel = levels[level-1];
    const newLevel = [];

    for(let i = 0; i < prevLevel.length - 1; i++){
      newLevel.push([
      (1-t) * prevLevel[i][0] + t * prevLevel[i+1][0],
      (1-t) * prevLevel[i][1] + t * prevLevel[i+1][1]
      ]);
    }
    levels.push(newLevel);
  }

  return levels;
}
```

5.2 bezierPoints

Функцията използва алгоритъма на De Casteljau: започва се с всички контролни точки и се извършват последователни линейни интерполации между съседни точки, докато не остане една единствена точка за даден параметър $t \in [0, 1]$. Тази точка представлява точка от кривата $\mathcal{B}$, съответстваща на стойността t.

```
function bezierPoint(points, t){
  let p = points.map(v => [...v]);
  for (let k = p.length - 1; k > 0; k--){
    for (let i = 0; i < k; i++) {
      p[i] = [(1-t) * p[i][0] + t * p[i+1][0], (1-t) * p[i][1] +
          t * p[i+1][1]];
    }
  }
  return p[0];
}
```

5.3 sampleBezier

Функцията `sampleBezier` се използва за представяне на кривата на Безие. За всяка стъпка се изчислява текущата стойност на параметъра t като $\frac{i}{steps}$, така че t варира равномерно в интервала $[0, 1]$. След това за всяка стойност на t се извиква функцията `bezierPoint`, която прилага алгоритъма на de Casteljau, за да определи точката на кривата, съответстваща на текущия параметър. Всички изчислени точки се съхраняват в масив, който се връща като резултат на функцията.

```
function sampleBezier(points, steps = 100){
  const out = [];
  for(let i = 0; i <= steps; i++) {
    // Calculate point at current t value
    out.push(bezierPoint(points, i/steps));
  }
  return out; // array of points along the curve
}
```

5.4 Ходограф на крива на Безие

Ходографът на кривата на Безие представлява нейната производна спрямо параметъра t. Производната на крива на Безие от степен n се задава с формулата:

$$\frac{d}{dt}B^n(t) = n\sum_{i=0}^{n-1}(b_{i+1} - b_i)\, B_i^{n-1}(t),$$

където b_i са контролните точки на кривата, а $B_i^{n-1}(t)$ са полиномите на Бернщайн от степен $n - 1$.

В теоретичната формула разликите между съседните контролни точки се умножават по коефициента n. В практическата реализация този коефициент не се използва директно. Причината е, че умножението по n води до значително увеличаване на дължините на векторите на ходографа, което би довело до излизане на визуализацията извън границите на платното.

Вместо това, ходографът се конструира чрез разликите между съседни контролни точки, след което получените вектори се мащабират с подходящ коефициент. По

този начин се запазва формата и посоката на ходографа, като същевременно се гарантира, че той остава в рамките на видимата област.

Полученият масив с разлики между контролни точки се подава към функцията `sampleBezier` за генериране на точки по ходографа. Кривата се мащабира с подходящ коефициент, за да се запази видимостта ѝ на платното, като се използва същата процедура, както при оригиналната крива на Безие.

```
  const n = controlPoints.length - 1;
  const deriv = [];

  for(let i = 0; i < n; i++) {
    deriv.push([
    controlPoints[i+1][0] - controlPoints[i][0],
    controlPoints[i+1][1] - controlPoints[i][1]
    ]);
  }

  // Scale vectors for visualization
  let maxMag = 0;
  for(const p of deriv)
  maxMag = Math.max(maxMag, Math.hypot(p[0], p[1]));

  const scale = maxMag > 0 ? 0.7 / maxMag : 1;

  ...

  // Draw hodograph curve
  const sampled = sampleBezier(deriv).map(p => [p[0]*scale, p[1]*
      scale]);
  drawGL(ctxH, sampled, ctxH.gl.LINE_STRIP, [0.8,0.2,0.2]);
```

6 Допълнителен код

6.1 Функция за реализация (`render`)

Функцията `render` отговаря за цялостната визуализация на сцената. В нея се извършват последователни проверки за активирани опции чрез състоянието на съответните бутони (контролни точки, крива на Безие, алгоритъм на de Casteljau, анимация и ходограф).

В зависимост от избраните настройки функцията условно извиква различни процедури за изчертаване, като всяка от тях се активира само при наличие на достатъчен брой контролни точки. По този начин се избягват невалидни операции и се гарантира коректна визуализация.

Примерни проверки за кривата на Безие:

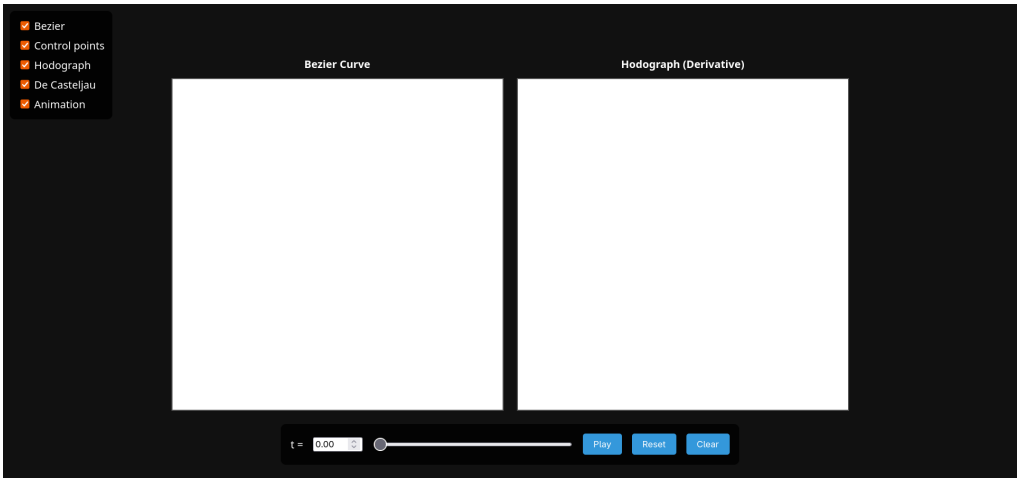
```
function render(){
  ...
  // Draw control polygon and control points
  if(show.control && controlPoints.length) {
    drawGL(ctxB, controlPoints, ctxB.gl.LINE_STRIP, [0.1,0.1,0.1]);
    drawGL(ctxB, controlPoints, ctxB.gl.POINTS, [0,0.5,0.8]);
  }
  ...
  // Draw animation for Bezier curve
```

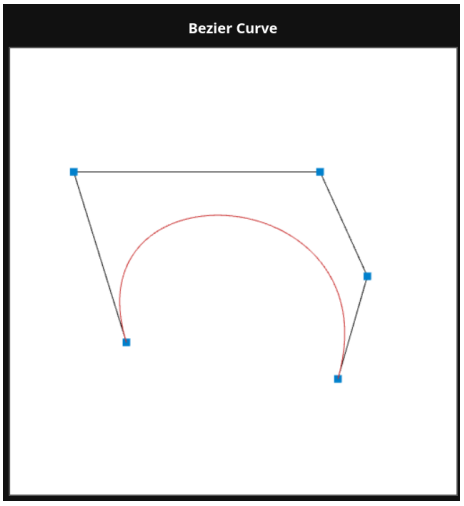
```
  if(show.animation && t > 0 && controlPoints.length > 1) {
    const tracedPoints = [];
    for(let i = 0; i <= Math.floor(t * 100); i++) {
      tracedPoints.push(bezierPoint(controlPoints, i / 100));
    }
    drawGL(ctxB, tracedPoints, ctxB.gl.LINE_STRIP, [0,0,0]);
  }
  ...
}
```

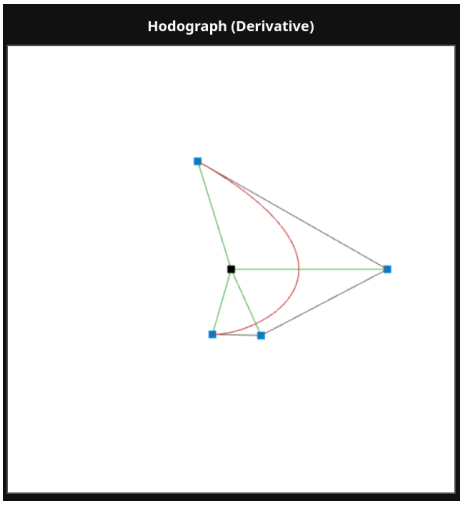
За ходографа:

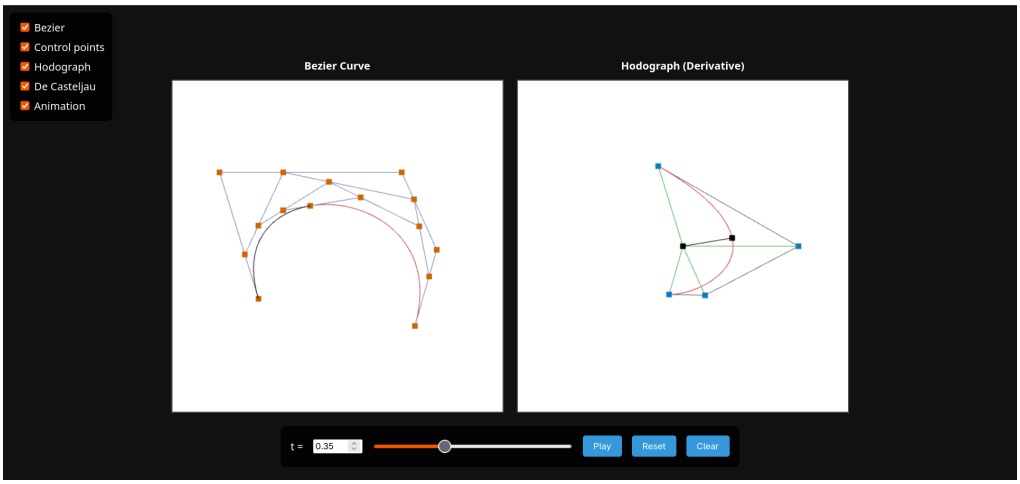
```
function render(){
    ...

  if(show.hodo && controlPoints.length >= 2) {

    ...

    // Draw vectors from central point
    for(const p of scaled) {
      drawGL(ctxH, [[0,0], p], ctxH.gl.LINES, [0.3,0.7,0.3]);
    }

    ...

    // Draw the animation vector for t > 0
    if(t > 0) {
      const pt = bezierPoint(deriv, t);
      drawGL(ctxH, [[0,0],[pt[0] * scale, pt[1] * scale]], ctxH.gl.
          LINES, [0,0,0]);
      drawGL(ctxH, [[pt[0] * scale, pt[1] * scale]], ctxH.gl.POINTS
          , [0,0,0]);
    }
  }
  ...
}
```

7 Снимки на проекта









8 Заключение

Крайният продукт представлява самостоятелен HTML файл, който включва целия необходим код и ресурси за функционирането на проекта, без да изисква интернет връзка. Файлът може да се стартира директно в стандартните уеб браузъри. Кодът и документацията на проекта са публикувани в GitHub.

9 Използвани материали

В реализирането на проекта бяха използвани следните ресурси за усвояване на концепции и техники:

- Записки по курса Компютърно геометрично моделиране (CAGD) на лектор доц. Красимира Александрова
- Video tutorial: **Learn WebGL #3 - The First Triangle**:
  www.youtube.com - Learn WebGL #3 - "The First Triangle" (GL API Tutorial)
- Mozilla Developer Network (MDN) documentation: **WebGL Model-View-Projection**:
  `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/WebGL_model_view_projection`

- WebGL Fundamentals: **WebGL Fundamentals Lessons**:
  https://webglfundamentals.org/webgl/lessons/webgl-fundamentals.html
- Detecting mouse position on the canvas:
  riptutorial.com - Mouse Position on Canvas
- За реализацията на частта за скалиране на ходографа беше използвана помощ от `DeepSeek AI`:
  https://chat.deepseek.com/

Допълнителни връзки:

- Github: https://github.com/AdamS839/Hodograph-BezierCurve