Mathematical Description of the Algorithm

De Casteljau's Algorithm

Let n be a natural number, and let $b_0, b_1, ..., b_n$ be $n + 1$ distinct points in $\mathbb{R}^2$, and t from the interval [0, 1].

The de Casteljau algorithm uses successive linear interpolations and, after n steps, constructs a point $b_0^n(t)$ on a polynomial curve B of degree n. This curve is called a **Bézier curve**. The points $b_0, b_1, ..., b_n$ are called **control points**, and the polygon formed by these points is called the **control polygon**.

In the context of the program, the algorithm is used to visualize the Bézier curve defined by user-provided control points, as well as to visualize the polar curve selected by the user.

Step-by-step explanation of the algorithm:

```
35  std::vector<Point> deCasteljau(const std::vector<Point>& controlPoints, float t) {
36      std::vector<Point> newPoints;
37
38      for (size_t i = 0; i < controlPoints.size() - 1; ++i) {
39          float x = (1 - t) * controlPoints[i].x + t * controlPoints[i + 1].x;
40          float y = (1 - t) * controlPoints[i].y + t * controlPoints[i + 1].y;
41          newPoints.push_back({ x, y });
42      }
43
44      return newPoints;
45  }
```

The algorithm is executed on the given control points and the parameter t, generating new intermediate points according to the following formula:

Let the control points be: $b_0, b_1, ..., b_n \in \mathbb{R}^2$, and the parameter $t \in [0, 1]$

$$\mathbf{b_i^{new}(t) = (1 - t) * b_i(t) + t * b_{i+1}(t)}$$

This operation is performed for all n control points, resulting in an array of $n - 1$ intermediate points.

To obtain the final point lying on the Bézier curve, the algorithm must be applied n times (or in this case, until a single point is obtained).

```
Point bezierPoint(const std::vector<Point>& controlPoints, float t) {
    std::vector<Point> tempPoints = controlPoints;

    while (tempPoints.size() > 1) {
        tempPoints = deCasteljau(tempPoints, t);
    }

    return tempPoints[0];
}
```

Thus, we have described the de Casteljau algorithm for computing a single point on the Bézier curve, in the context of the program.

Algorithm 1 *de Casteljau*

Вход: $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^3,\ t \in \mathbb{R}$

$$\begin{aligned} &\mathbf{b}_i^r(t) = (1-t)\mathbf{b}_i^{r-1}(t) + t\mathbf{b}_{i+1}^{r-1}(t), \\ &r = 1, \ldots, n;\ i = 0, \ldots, n-r \\ &\mathbf{b}_i^0 = \mathbf{b}_i \end{aligned} \tag{9}$$

Изход: $\mathbf{b}_0^n(t)$ е точка от кривата $\mathcal{B}$, съответстваща на параметъра t.

After applying the de Casteljau algorithm for all values of the parameter $t \in [0, 1]$ with a certain step size, the Bézier curve will be visualized. The following describes the function for visualizing the Bézier curve:

```
void renderBezier(const std::vector<Point>& controlPoints, const float color[3]) {
    const int numSegments = 100;
    glColor3fv(color);
    glBegin(GL_LINE_STRIP);

    for (int i = 0; i <= numSegments; ++i) {
        float t = i / (float)numSegments;
        Point bezierPt = bezierPoint(controlPoints, t);
        glVertex2f(bezierPt.x, bezierPt.y); // Plot the point
    }

    glEnd();
}
```

Algorithm for the Polar of a Bézier Curve

The **first polar** of a Bézier curve is the curve that is formed by taking the first intermediate points obtained after one execution of the de Casteljau algorithm for a parameter t_1, and then treating these points as control points of another Bézier curve, which in turn depends on another parameter $t_2 \in [0, 1]$.

This resulting curve will be of one degree lower than the original Bézier curve.