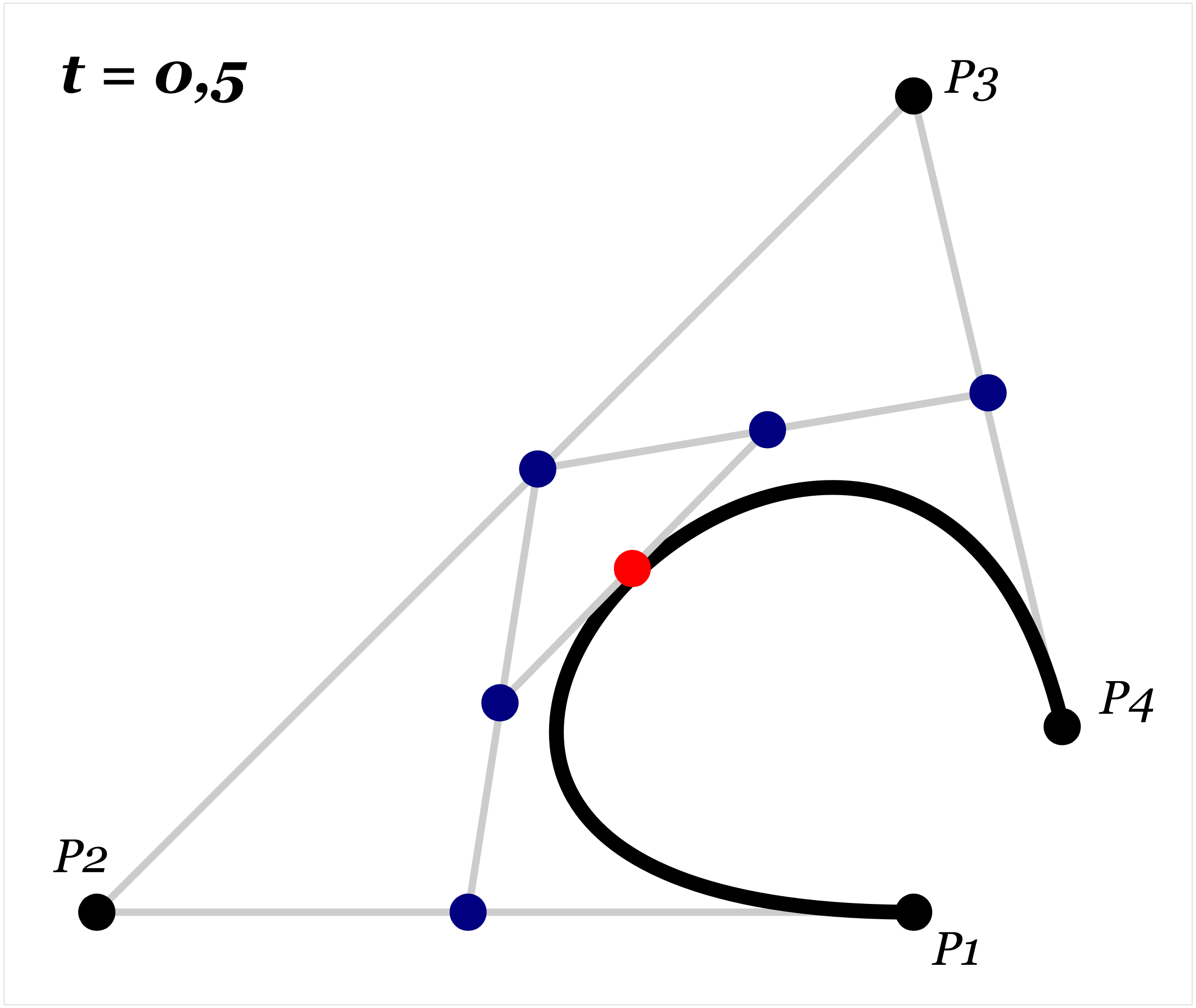
Algorytm de Casteljau

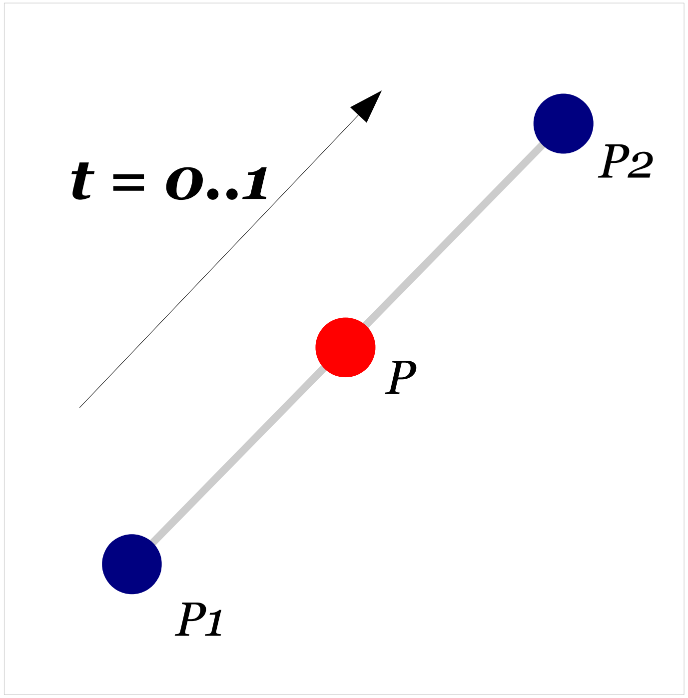
- Dowolna łamana zdefiniowana jest przez ***n+1*** wierzchołków ***p0, p1 … pn***.
- Każdy odcinek łamanej dzielony jest na części w stosunku ***t : 1-t***, dla ***0*** ≤ ***t*** ≤ ***1***, co daje ***n*** wierzchołków wyznaczających nową łamaną.
- Proces powtarzany jest do momentu aż zostanie jeden punkt (po ***n*** krokach).

Krok	Ciąg punktów
0	*p0, p1, p2, p3, … pn*
1	*p0, p1, p2, … pn-1*
...	*...*
n-1	*p0, p1*
n	*p0*

Algorytm de Casteljau

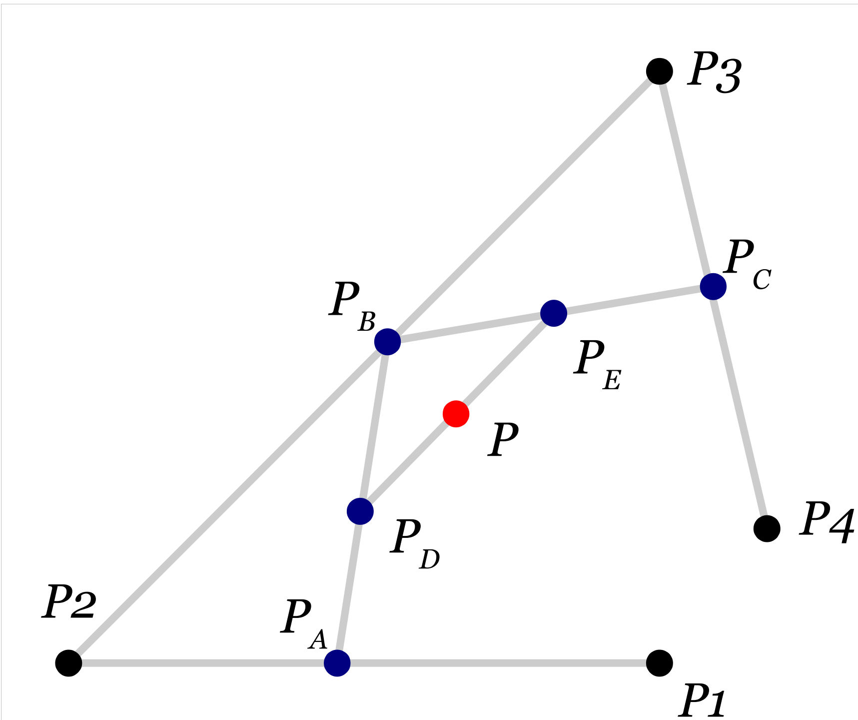


Interpolacja liniowa



$$\begin{cases} X = X_1 + (X_2 - X_1) * t \\ Y = Y_1 + (Y_2 - Y_1) * t \end{cases}$$

Interpolacja w algorytmie de Casteljau



$$\begin{cases} X = X_D + (X_E - X_D) * t \\ Y = Y_D + (Y_E - Y_D) * t \end{cases}$$

$$\begin{cases} X_E = X_B + (X_C - X_B) * t \\ Y_E = Y_B + (Y_C - Y_B) * t \end{cases}$$

$$\begin{cases} X_D = X_A + (X_B - X_A) * t \\ Y_D = Y_A + (Y_B - Y_A) * t \end{cases}$$

$$\begin{cases} X_A = X_1 + (X_2 - X_1) * t \\ Y_A = Y_1 + (Y_2 - Y_1) * t \end{cases} \quad \begin{cases} X_B = X_2 + (X_3 - X_2) * t \\ Y_B = Y_2 + (Y_3 - Y_2) * t \end{cases} \quad \begin{cases} X_C = X_3 + (X_4 - X_3) * t \\ Y_C = Y_3 + (Y_4 - Y_3) * t \end{cases}$$

Algorytm de Casteljau dla 4 punktów

Po przekształceniach...

$$X = X_1\left(1-3t+3t^2-t^3\right)+X_2\left(3t-6t^2+3t^3\right)+X_3\left(3t^2-3t^3\right)+X_4t^3$$
$$Y = Y_1\left(1-3t+3t^2-t^3\right)+Y_2\left(3t-6t^2+3t^3\right)+Y_3\left(3t^2-3t^3\right)+Y_4t^3$$

Wzory skróconego mnożenia itp....

$$X = X_1\left(1-t\right)^3+3t\,X_2\left(1-t\right)^2+3\,X_3\left(1-t\right)t^2+X_4t^3$$
$$Y = Y_1\left(1-t\right)^3+3t\,Y_2\left(1-t\right)^2+3\,Y_3\left(1-t\right)t^2+Y_4t^3$$

Algorytm de Casteljau dla 4 punktów

```
// Algorytm de Casteljau dla 4 punktów ograniczających:
void deCastel(float curve[], float t, float res[3]) {
    res[0] = pow((1 - t), 3) * curve[0]
            + 3 * t * pow((1 -t), 2) * curve[3+0]
            + 3 * (1-t) * pow(t, 2)* curve[6+0]
            + pow(t, 3)* curve[9+0];
    res[1] = pow((1 - t), 3) * curve[1]
            + 3 * t * pow((1 -t), 2) * curve[3+1]
            + 3 * (1-t) * pow(t, 2)* curve[6+1]
            + pow(t, 3)* curve[9+1];
    res[2] = pow((1 - t), 3) * curve[2]
            + 3 * t * pow((1 -t), 2) * curve[3+2]
            + 3 * (1-t) * pow(t, 2)* curve[6+2]
            + pow(t, 3)* curve[9+2];
}
```

Pozycja na krzywej Béziera

```
bool getBezierPos(float kr[], float t, float *x, float *y) {
      float res[3];
      deCastel(kr, t, res);
      *x = res[0];
      *y = res[1];
      return(true);
}
```

Rysowanie krzywej Béziera

```
void drawBezier(float kr[]) {
   // Narysowanie wszystkich punktów:
   drawPoints(kr, 4);

   glColor3f(0.0, 0.0, 0.0);
   glLineWidth(3);

   glBegin(GL_LINE_STRIP);
      for(int i=0;i <= 64;i++) {
         float pt[3];
         deCastel(kr, (float)i/64.0, pt);
         glVertex3fv(pt);
      }
   glEnd();
}
```