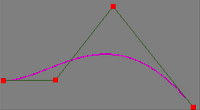
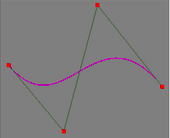
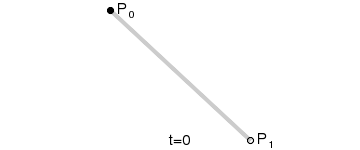
Bezier曲线中的每个控制点都会影响整个曲线的形状，而B样条中的控制点只会影响整个曲线的一部分，显然B样条提供了更多的灵活性；Bezier曲线只是B样条的一个特例而已，而B样条又是NURBS的一个特例。



1. **一阶贝塞尔曲线**

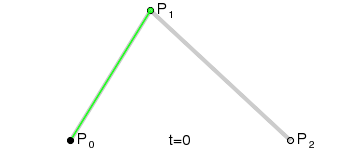


<动态图/一阶贝塞尔曲线.gif>

对应的公式为：

2020-11-26 15-33-21 的屏幕截图

1. **二阶贝塞尔曲线**

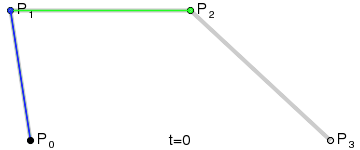
****

**<动态图/二阶贝塞尔曲线.gif>**

对应的公式为：

2020-11-26 15-43-31 的屏幕截图

1. **三阶贝塞尔曲线**

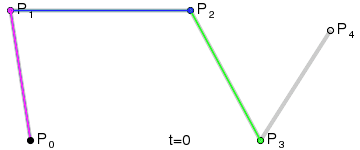


<动态图/三阶贝塞尔曲线.gif>

对应的公式为：

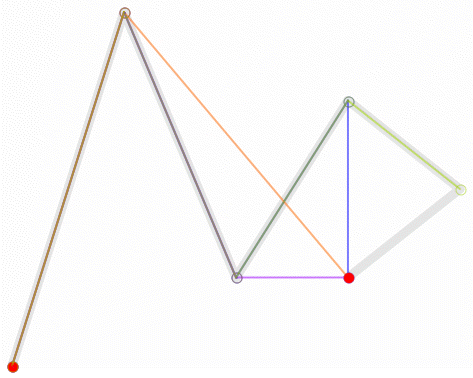
2020-11-26 15-46-10 的屏幕截图

1. **四阶贝塞尔曲线**



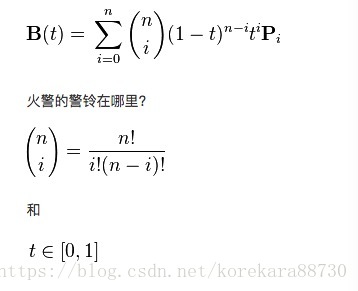
<动态图/四阶贝塞尔曲线.gif>

1. **五阶贝塞尔曲线**



<动态图/五阶贝塞尔曲线.gif>

1. **通用公式：**



1. **三阶贝塞尔曲线的C++实现**

void createCurvecal\_angle 求分辨率为 \_resolution 的时候每一点的切线方向。

(point2D \*originPoint,int originCount,std::vector<point2D> &curvePoint){

//控制点收缩系数 ，经调试0.6较好，CvPoint是opencv的，可自行定义结构体(x,y)

float scale = 0.6;

point2D midpoints[originCount];

//生成中点

for(int i = 0 ;i < originCount ; i++){

int nexti = (i + 1) % originCount;

midpoints[i].x = (originPoint[i].x + originPoint[nexti].x)/2.0;

midpoints[i].y = (originPoint[i].y + originPoint[nexti].y)/2.0;

}

//平移中点

CvPoint extrapoints[2 \* originCount];

for(int i = 0 ;i < originCount ; i++){

int nexti = (i + 1) % originCount;

int backi = (i + originCount - 1) % originCount;

CvPoint midinmid;

midinmid.x = (midpoints[i].x + midpoints[backi].x)/2.0;

midinmid.y = (midpoints[i].y + midpoints[backi].y)/2.0;

int offsetx = originPoint[i].x - midinmid.x;

int offsety = originPoint[i].y - midinmid.y;

int extraindex = 2 \* i;

extrapoints[extraindex].x = midpoints[backi].x + offsetx;

extrapoints[extraindex].y = midpoints[backi].y + offsety;

//朝 originPoint[i]方向收缩

int addx = (extrapoints[extraindex].x - originPoint[i].x) \* scale;

int addy = (extrapoints[extraindex].y - originPoint[i].y) \* scale;

extrapoints[extraindex].x = originPoint[i].x + addx;

extrapoints[extraindex].y = originPoint[i].y + addy;

int extranexti = (extraindex + 1)%(2 \* originCount);

extrapoints[extranexti].x = midpoints[i].x + offsetx;

extrapoints[extranexti].y = midpoints[i].y + offsety;

//朝 originPoint[i]方向收缩

addx = (extrapoints[extranexti].x - originPoint[i].x) \* scale;

addy = (extrapoints[extranexti].y - originPoint[i].y) \* scale;

extrapoints[extranexti].x = originPoint[i].x + addx;

extrapoints[extranexti].y = originPoint[i].y + addy;

}

CvPoint controlPoint[4];

//生成4控制点，产生贝塞尔曲线

for(int i = 0 ;i < originCount ; i++){

controlPoint[0] = originPoint[i];

int extraindex = 2 \* i;

controlPoint[1] = extrapoints[extraindex + 1];

int extranexti = (extraindex + 2) % (2 \* originCount);

controlPoint[2] = extrapoints[extranexti];

int nexti = (i + 1) % originCount;

controlPoint[3] = originPoint[nexti];

float u = 1;

while(u >= 0){

int px = bezier3funcX(u,controlPoint);

int py = bezier3funcY(u,controlPoint);

//u的步长决定曲线的疏密

u -= 0.005;

CvPoint tempP = cvPoint(px,py);

//存入曲线点

curvePoint.push\_back(tempP);

}

}

}//三次贝塞尔曲线double CurvePlot::bezier3funcX(double \_t,point2D \*controlP){ // cal x coor

double part0 = controlP[0].x \* \_t \* \_t \* \_t;

double part1 = 3 \* controlP[1].x \* \_t \* \_t \* (1 - \_t);

double part2 = 3 \* controlP[2].x \* \_t \* (1 - \_t) \* (1 - \_t);

double part3 = controlP[3].x \* (1 - \_t) \* (1 - \_t) \* (1 - \_t);

return part0 + part1 + part2 + part3;

}double CurvePlot::bezier3funcY(double \_t,point2D \*controlP){ // cal y coor

double part0 = controlP[0].y \* \_t \* \_t \* \_t;

double part1 = 3 \* controlP[1].y \* \_t \* \_t \* (1 - \_t);

double part2 = 3 \* controlP[2].y \* \_t \* (1 - \_t) \* (1 - \_t);

double part3 = controlP[3].y \* (1 - \_t) \* (1 - \_t) \* (1 - \_t);

return part0 + part1 + part2 + part3;

}double CurvePlot::cal\_angle(double \_t, point2D \*controlP){ // cal dy / dx angle

double \_dx\_1= 3 \* controlP[0].x \* \_t \* \_t;

double \_dx\_2= 3 \* controlP[1].x \* (\_t \* 2 - 3 \* \_t \* \_t);

double \_dx\_3= 3 \* controlP[2].x \* (1 - 4 \* \_t + 3 \* \_t \* \_t);

double \_dx\_4= - 3 \* controlP[3].x \* (1 - \_t) \* (1 - \_t);

double \_dy\_1= 3 \* controlP[0].y \* \_t \* \_t;

double \_dy\_2= 3 \* controlP[1].y \* (\_t \* 2 - 3 \* \_t \* \_t);

double \_dy\_3= 3 \* controlP[2].y \* (1 - 4 \* \_t + 3 \* \_t \* \_t);

double \_dy\_4= - 3 \* controlP[3].y \* (1 - \_t) \* (1 - \_t);

return std::atan2(\_dy\_1 + \_dy\_2 + \_dy\_3 + \_dy\_4 , \_dx\_1 + \_dx\_2 + \_dx\_3 + \_dx\_4);

}