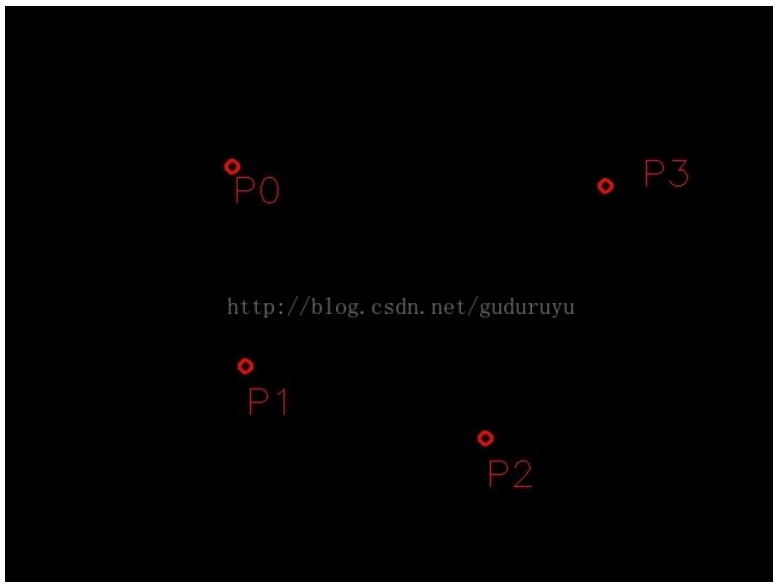
转 基于三次Bezier原理的曲线拟合算法C++与OpenCV实现

2018年03月07日 15:09:12 eric_e 阅读数：482

近期，因为要实现经过多个控制点的曲线拟合，研究起了曲线拟合算法。综合搜索到的资料，发现Bezier曲线拟合算法是一种相对较容拟合的效果较好的算法。关于Bezier曲线原理，请参照（Bezier曲线原理），这里就不再做具体介绍了，我们使用的是Besier三次曲线下面主要介绍算法的实现过程。

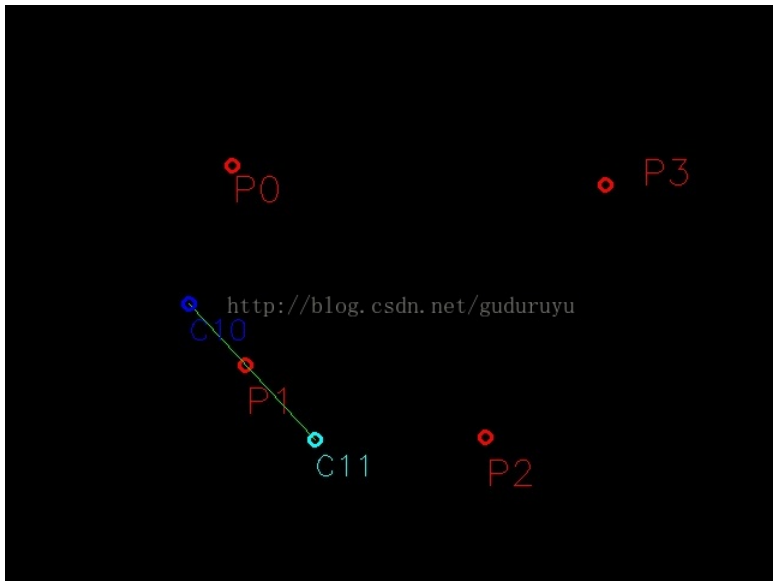
如下图中，P0、P1、P2、P3四个点，我们最终是想获取过这四个点的封闭平滑曲线。



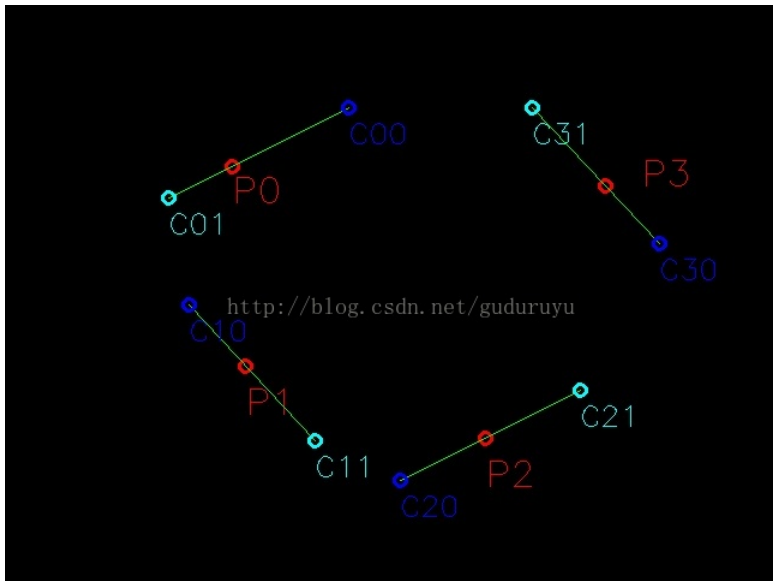
根据Bezier三次曲线拟合的原理，我们可以分别拟合P0P1、P1P2、P2P3、P3P0四段曲线，进而连接成一个封闭的曲线。但是，Bezie拟合需要在两点之间找到两个控制点。每个点的控制点可以根据其前后相邻的两点获得，具体实现如下：

```cpp
void get_control_points(double x0, double y0, double x1, double y1, double x2, double y2,
    double& p1x, double& p1y, double& p2x, double& p2y, double t)
{
    double d01 = sqrt(pow(x1 - x0, 2) + pow(y1 - y0, 2));
    double d12 = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

    double fa = t * d01 / (d01 + d12);
    double fb = t * d12 / (d01 + d12);

    p1x = x1 - fa * (x2 - x0);
    p1y = y1 - fa * (y2 - y0);
    p2x = x1 + fb * (x2 - x0);
    p2y = y1 + fb * (y2 - y0);

    return;
}
```

其中，（x0，y0）、（x1，y1）、（x2，y2）分别为P0、P1、P2三点的坐标；t为曲率因子（取值范围0-1.0），影响的是拟合曲线的面将对其作进一步介绍。根据三点的坐标和t，即可求得P1点的两个控制点C10（p1x，p1y）、C11(p2x，p2y)。



以此类推，我们可分别求得P2、P3、P0等各点的控制点，如下图所示。

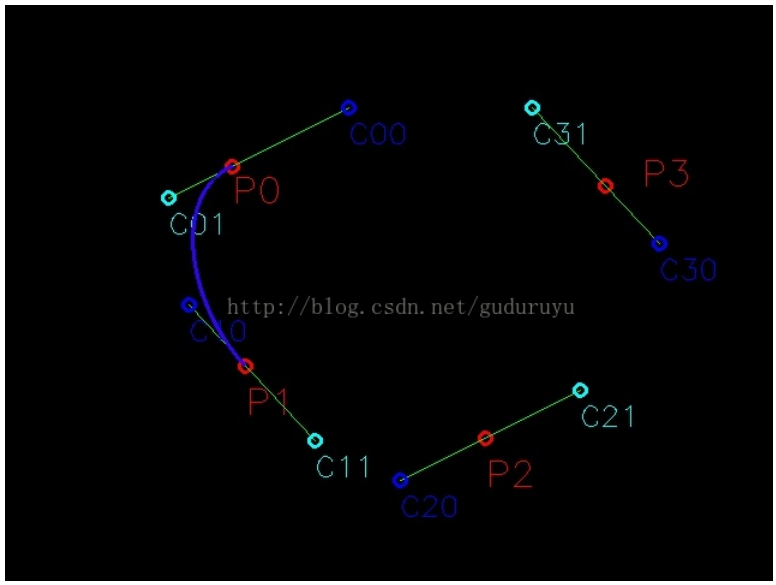


接下来，我们我们逐段绘制Besier曲线。通过两个顶点P0、P1和两个控制点C01、C10，根据Bezier曲线拟合原理，即可获得连接P0、曲线。

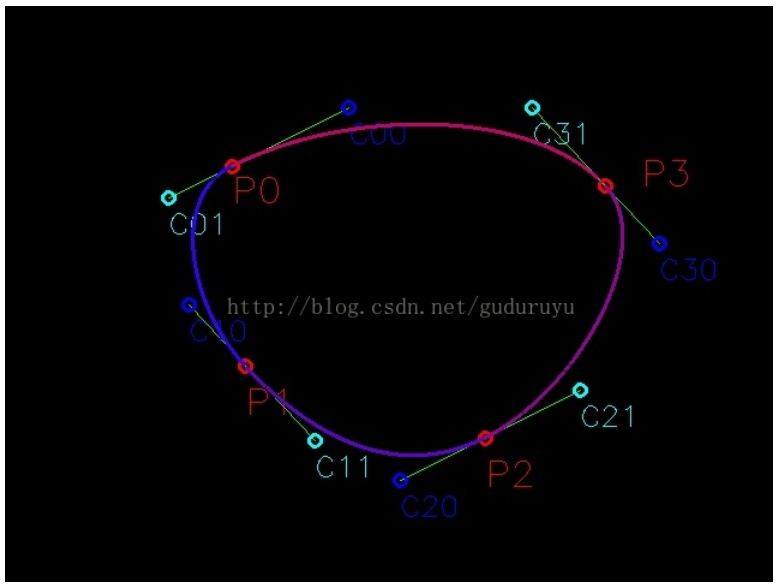
```cpp
void get_bezier(double x1, double y1, double x2, double y2, double p12x, double p12y,
    double p21x, double p21y, std::vector<int>& vec_x, std::vector<int> & vec_y)
{
    int prev_x = (int)round(x1);
    int prev_y = (int)round(y1);
    int last_x = (int)round(x2);
    int last_y = (int)round(y2);

    for (double s = 0.0; s < (1.0 + 0.00001); s += DELTA_S)
    {
        double J0 = pow(1 - s, 3);
        double J1 = pow(1 - s, 2) * s * 3;
        double J2 = pow(s, 2) * (1 - s) * 3;
        double J3 = pow(s, 3);
```

```
        double ptx = x1 * J0 + p12x * J1 + p21x * J2 + x2 * J3;
        double pty = y1 * J0 + p12y * J1 + p21y * J2 + y2 * J3;

        int iptx = (int)round(ptx);
        int ipty = (int)round(pty);

        vec_x.push_back(iptx);
        vec_y.push_back(ipty);
    }
    return;
}
```

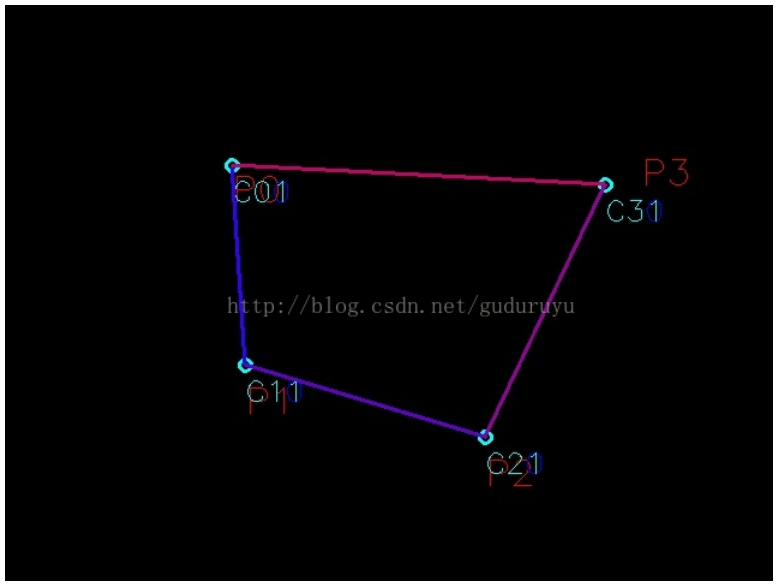
其中，DELTA_S是拟合的步长。再通过轮廓查找算法和插值算法，即可得到一段完整的Besier曲线，如下图所示。

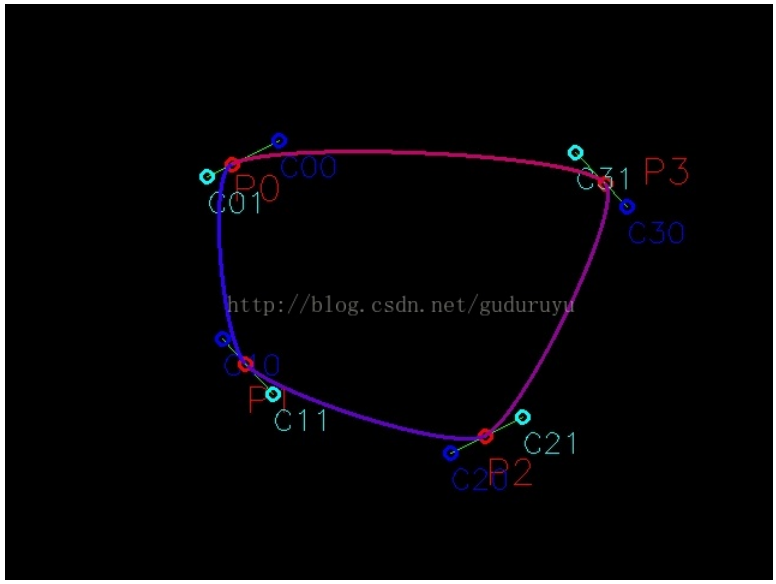


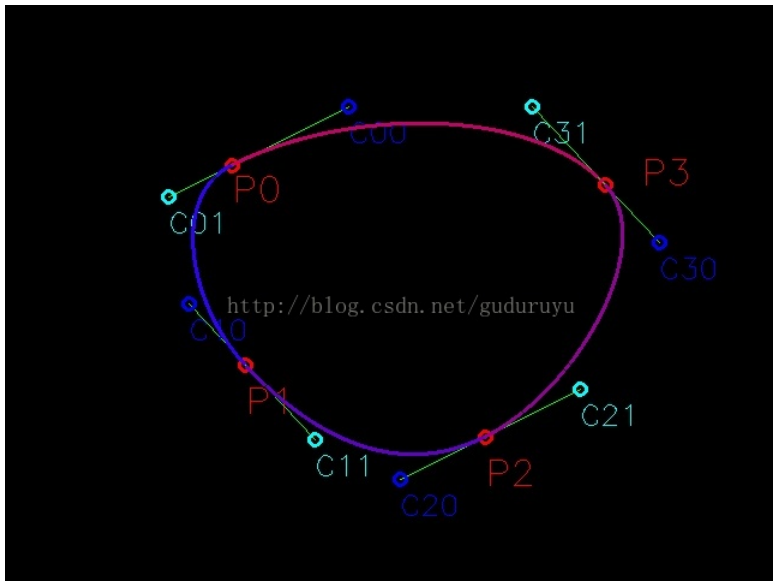
同样，以此类推，我们可以分别得到P1P2、P2P3、P3P0之间的曲线，将这几段曲线连接在一起，即可得到一条完整的封闭曲线。

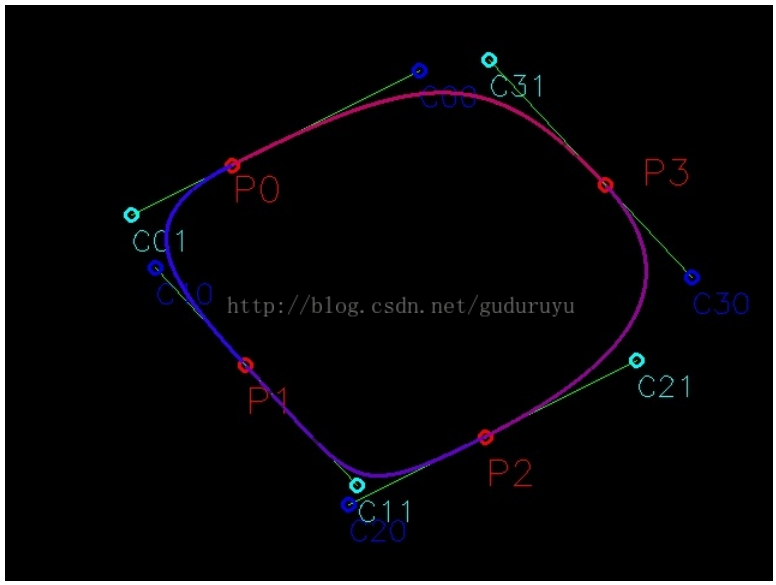


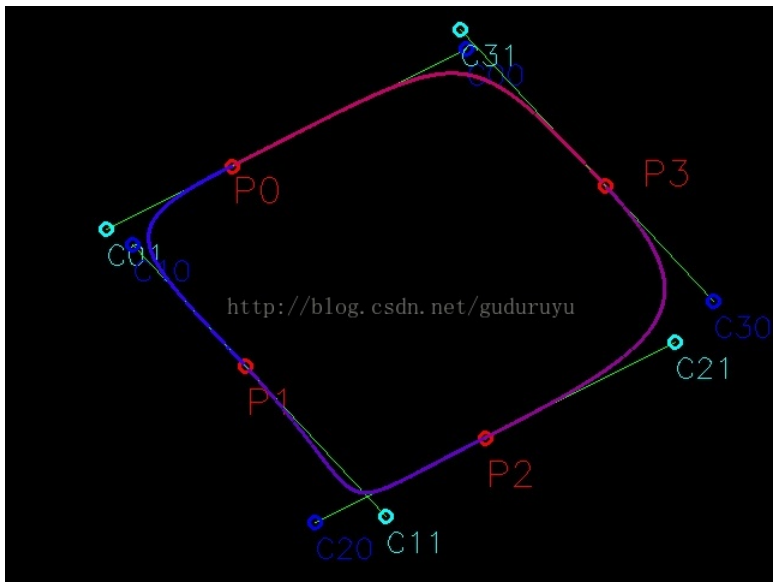
最后，我们再看一看前文提到的曲率因子t对拟合出来的曲线的影响。分别令 t = 0.0、0.2、0.5、0.8、1.0，得到的曲线分别如下图所示











下面是基于OpenCV的完整实现代码：

```cpp
#include "spline_curve.h"
#include <vector>

#define DELTA_S		0.01
#define T		0.5


void get_control_point(cv::Point2d& point0, cv::Point2d& point1, cv::Point2d& point2,
	cv::Point2d& c01, cv::Point2d& c12, double t)
{
	double d01 = sqrt(pow(point1.x - point0.x, 2) + pow(point1.y - point0.y, 2));
	double d12 = sqrt(pow(point2.x - point1.x, 2) + pow(point2.y - point1.y, 2));


	double fa = t * d01 / (d01 + d12);
	double fb = t * d12 / (d01 + d12);

	c01.x = point1.x - fa * (point2.x - point0.x);
	c01.y = point1.y - fa * (point2.y - point0.y);
	c12.x = point1.x + fb * (point2.x - point0.x);
	c12.y = point1.y + fb * (point2.y - point0.y);

}
```

```
25.
26. void get_control_points_array
    (std::vector<cv::Point2d>& key_points, std::vector<cv::Point2d>& vec_c01,
27.     std::vector<cv::Point2d>& vec_c02, double t)
28. {
29.     int N = key_points.size();
30.
31.     for (int i = 0; i < N; i++)
32.     {
33.         cv::Point2d c01, c02;
34.
35.         if (i == 0)
36.         {
37.             get_control_point(key_points[N - 1], key_points[i], key_points[i + 1], c01, c02, t);
38.             vec_c01.push_back(c01);
39.             vec_c02.push_back(c02);
40.         }
41.         else if (i < (N - 1))
42.         {
43.             get_control_point(key_points[i - 1], key_points[i], key_points[i + 1], c01, c02, t);
44.             vec_c01.push_back(c01);
45.             vec_c02.push_back(c02);
46.         }
47.         else
48.         {
49.             get_control_point(key_points[i - 1], key_points[i], key_points[0], c01, c02, t);
50.             vec_c01.push_back(c01);
51.             vec_c02.push_back(c02);
52.         }
53.     }
54. }
55.
56.
57. bool is_adjcent_point(cv::Point2i& point1, cv::Point2i& point2)
58. {
59.     if (((point1.x == point2.x) && (point1.y == point2.y)) ||
60.         (std::abs(point1.x - point2.x) > 1) || (std::abs(point1.y - point2.y) > 1))
61.     {
62.         return false;
63.     }
64.
65.     return true;
66. }
67.
68. bool is_same_point(cv::Point2i& point1, cv::Point2i& point2)
69. {
70.     if ((point1.x == point2.x) && (point1.y == point2.y))
71.     {
72.         return true;
73.     }
74.
75.     return false;
76. }
77.
78. // interpolation between not adjacent points
79. void get_line_points(cv::Point2i& point1, cv::Point2i& point2, std::vector<cv::Point2i>& line_points)
80. {
81.     line_points.push_back(point1);
82.
83.     int dx = abs(point1.x - point2.x);
84.     int dy = abs(point1.y - point2.y);
85.
86.     if (dx == 0 && dy == 0)
87.     {
88.         return;
89.     }
90.
91.     if (dx > dy)
92.     {
93.         if (point1.x < point2.x)
94.         {
95.             for (int i = point1.x + 1; i < point2.x; i++)
96.             {
97.                 int y = (int)(((point1.y - point2.y + 0.0) / (point1.x - point2.x)) * (i - point1.x) + point1.y);
98.                 line_points.push_back(cv::Point2i(i, y));
99.             }
100.        }
101.        else
102.        {
103.            for (int i = point1.x - 1; i > point2.x; i--)
104.            {
105.                int y = (int)(((point1.y - point2.y + 0.0) / (point1.x - point2.x)) * (i - point1.x) + point1.y);
106.                line_points.push_back(cv::Point2i(i, y));
107.            }
108.        }
109.    }
110.    else
111.    {
112.        if (point1.y < point2.y)
```

```
        {
            for (int i = point1.y + 1; i < point2.y; i++)
            {
                int x = (int)(((point1.x - point2.x + 0.0) / (point1.y - point2.y)) * (i - point1.y) + point1.x);
                line_points.push_back(cv::Point2i(x, i));
            }
        }
        else
        {
            for (int i = point1.y - 1; i > point2.y; i--)
            {
                int x = (int)(((point1.x - point2.x + 0.0) / (point1.y - point2.y)) * (i - point1.y) + point1.x);
                line_points.push_back(cv::Point2i(x, i));
            }
        }

    }

    line_points.push_back(point2);

    return;
}


bool get_spline(cv::Point2d& point1, cv::Point2d& point2, cv::Point2d& c01,
    cv::Point2d& c12, std::vector<cv::Point2i>& spline_points, double delta_s)
{
    cv::Point2i point_prev = (cv::Point2i)point1;
    cv::Point2i point_last = (cv::Point2i)point2;

    spline_points.push_back(point_prev);

    for (double s = 0.0; s < (1.0 + 0.0001); s += delta_s)
    {
        double J0 = pow(1 - s, 3);
        double J1 = pow(1 - s, 2) * s * 3;
        double J2 = pow(s, 2) * (1 - s) * 3;
        double J3 = pow(s, 3);

        double ptx = point1.x * J0 + c01.x * J1 + c12.x * J2 + point2.x * J3;
        double pty = point1.y * J0 + c01.y * J1 + c12.y * J2 + point2.y * J3;

        cv::Point2i ipoint;

        ipoint.x = (int)round(ptx);
        ipoint.y = (int)round(pty);

        if (is_same_point(ipoint, point_last))
        {
            get_line_points(point_prev, point_last, spline_points);
            break;
        }

        if (is_adjcent_point(point_prev, ipoint))
        {

            spline_points.push_back(ipoint);
            point_prev = ipoint;
        }
        else if (is_same_point(point_prev, ipoint))
        {
            continue;
        }
        else
        {
            get_line_points(point_prev, ipoint, spline_points);
            point_prev = ipoint;
        }
    }
    return true;
}


void smooth_curve
(std::vector<cv::Point2i>& curve_in, std::vector<cv::Point2i>& curve_out, bool is_closed)
{
    int vec_size = curve_in.size();

    for (int i = 0; i < (vec_size - 2); i += 2)
    {
        if (i == 0 && is_closed)
        {
            if (is_adjcent_point(curve_in[vec_size - 1], curve_in[1]))
            {
                curve_out.push_back(curve_in[1]);
            }
            else
            {
                curve_out.push_back(curve_in[0]);
```

```
            curve_out.push_back(curve_in[1]);
        }
    }

    if (is_adjcent_point(curve_in[i], curve_in[i + 2]))
    {
        curve_out.push_back(curve_in[i + 2]);
    }
    else
    {
        curve_out.push_back(curve_in[i + 1]);
        curve_out.push_back(curve_in[i + 2]);
    }
}

return;
}


bool get_spline_curve
(std::vector<cv::Point2d>& key_points, std::vector<cv::Point2i>& spline_curve, double t, bool is_closed
)
{
    if (key_points.size() < 2)
    {
        std::cout << "Key points is less than two!!!" << std::endl;
        return false;
    }

    if (key_points.size() == 2)
    {
        cv::Point2i point1 = (cv::Point2i)key_points[0];
        cv::Point2i point2 = (cv::Point2i)key_points[1];
        get_line_points(point1, point2, spline_curve);
        return true;
    }

    std::vector<cv::Point2d> vec_c01, vec_c12;
    get_control_points_array(key_points, vec_c01, vec_c12, t);

    std::vector<cv::Point2i> temp_spline;

    for (int i = 0; i < key_points.size(); i++)
    {
        if (i < (key_points.size() - 1))
        {
            get_spline(key_points[i], key_points[i + 1], vec_c12[i], vec_c01[i + 1], temp_spline, DELTA_S);
            continue;
        }

        if (is_closed)
        {
            get_spline(key_points[i], key_points[0], vec_c12[i], vec_c01[0], temp_spline, DELTA_S);
        }
    }

    smooth_curve(temp_spline, spline_curve, is_closed);

    return true;
}
```

转载：http://blog.csdn.net/guduruyu/article/details/60877086

曲线拟合/平滑算法实现及优化[基于C语言] 阅读数 1771
用CVI编写个小东西时，发现曲线的拟合已经到了1秒多才能完成一次曲线的拟合。代码如下：staticvoi... 博文 来自： fzh2712的专栏

 # 回归、拟合算法心得 阅读数 6001
根据斯坦福机器学习公开课整理的一点体会。记录了概念性的一些理解，具体的定义和证明还需要参阅... 博文 来自： jayandchuxu...

一个拟合曲线趋势的算法 阅读数 9564
笔者有一个这样的需求首先，有一套模板数据，横坐标是时间，做坐标是温度，用图形来看，大概是这... 博文 来自： 程序猿的成长...

二维曲线拟合 阅读数 565
二维曲线拟合相关基础理论知识最小二乘法广义逆（伪逆矩阵）矩阵分解特征值分解（EigenValueDec... 博文 来自： luppys

曲线平滑算法 阅读数 4906
由于项目开发需要对等值线进行平滑处理，所以研究了线条的平滑算法，经研究查阅资料，可以使用三... 博文 来自： 宋炜杰的博客

算法系列之二十一：实验数据与曲线拟合 阅读数 7万+
曲线拟合(CurveFitting)的数学定义是指用连续曲线近似地刻画或比拟平面上一组离散点所表示的坐标... 博文 来自： oRbIt 的专栏

【算法+OpenCV】基于opencv的直线和曲线拟合与绘制（最小二乘法） 阅读数 2万+
最小二乘法多项式曲线拟合，是常见的曲线拟合方法，有着广泛的应用，这里在借鉴最小二乘多项式曲... 博文 来自： guduruyu的专栏

曲线拟合与绘制 阅读数 745
2017-07-27 在学习图形学课程中，一个很重要的部分就是绘制曲线、曲面。其实，这部分需要的基础... 博文 来自： know yourself

线性拟合算法 03-03
线性拟合算法，可以计算一系列点的最小二乘拟合曲线，内涵使用方法说明，希望大家能用到 下载

多种方法实现的曲线拟合 04-15
VC6.0开发，曲线拟合，包括GDI+、贝塞尔曲线 下载

最小二乘法曲线拟合（C++） 10-23
用C++编写的程序，用最小二乘法实现对曲线的拟合，拟合的多项式达到六阶。 下载

用matlab实现，如何对一幅图片不同像素点进行多条曲线拟合？相同的像素点用一条曲线拟...
- 问答

基于贝塞尔曲线路径平滑算法 阅读数 643
tobecontinued... 博文 来自： 少年的技术积...

Bezier曲线原理及实现代码（c++） 阅读数 1482
Bezier曲线原理及实现代码（c++） 一、原理： 贝塞尔曲线于1962年，由法国工程师皮埃尔·... 博文 来自： 熊小屁的专栏

bezier曲线拟合，opencv，车道线拟合 12-26
在visual studio上新建项目，将本程序添加到源文件目录下，直接运行即可，你可以用鼠标在改变控制点的位置观察探究bezier拟合出曲线的变... 下载

基于C++的三次B样条曲线拟合代码 06-08

代码是基于C++的三次B样条曲线拟合代码，包含插值拟合，近似拟合就不放代码了，较简单，我的博客中有相关论文链接。http://blog.csdn.... 下载

图形算法：贝塞尔曲线 阅读数 1万+

图形算法：贝塞尔曲线标签（空格分隔）：算法版本：0作者：陈小默声明：禁止商用，禁止转载发布... 博文 来自：陈小默的博客

德卡斯特里奥算法——找到Bezier曲线上的一个点 阅读数 5120

随着Bezier曲线的构造，接下来最重要的任务就是通过给定的u值查找Bezier曲线上的点C(u)。... 博文 来自：venshine专栏

贝塞尔曲线生成算法 阅读数 4万+

这里先介绍另一个经典的曲线逼近方法，称作Bezier曲线。想必学过图形图像的都应该知道啦，所以概... 博文 来自：高兴的地方

bezier曲线的插值模拟算法 阅读数 5462

/********** REVISIONLOGENTRY Revi... 博文 来自：Robin's Log

曲线拟合 阅读数 224

polyfit函数基于最小二乘法，使用的基本格式为：p=polyfit(x,y,n)[p,S]=polyfit(x,y,n)[p,S,mu]=polyfi... 博文 来自：qq_40516725...

一种简单的贝塞尔拟合算法 阅读数 495

C#一种简单的贝塞尔拟合算法前两天实现了一项功能，在一端进行书写，在另一端还原笔迹。由于两端... 博文 来自：Iron 的博客

数值分析（拟合、插值和逼近）之数据插值方法（线性插值、二次插值、Cubic插值... 阅读数 7247

插值、拟合和逼近的区别据维基百科，科学和工程问题可以通过诸如采样、实验等方法获得若干离散的... 博文 来自：eric_e的博客

实验6 Bezier曲线生成 阅读数 2142

1．实验目的：了解曲线的生成原理，掌握几种常见的曲线生成算法，利用VC+OpenGL实现Bezier曲... 博文 来自：图形学与可视化

OpenGL画bezier曲线 阅读数 399

BezierCurve算法是根据参数曲线方程来得到光滑曲线的一种算法，曲线方程的参数由控制点决定。其... 博文 来自：unirrrrr的博客

C++ Bezier曲线拟合算法 12-20

这是用c++平台开发的一个Bezier曲线拟合的Demo例程，代码清晰。可以拓展应用性强。 下载

三阶贝塞尔曲线拟合圆弧的一般公式 阅读数 7684

针对三阶贝塞尔曲线拟合圆弧，进行一般性的公式求解，可以表达如下图所示：通过圆心O作出半径为1... 博文 来自：选择的专栏

三阶贝塞尔曲线拟合1/2正弦 阅读数 4109

三阶贝塞尔曲线拟合1/2正弦根据贝塞尔曲线的知识，我们知道三阶贝塞尔曲线的参数方程如下，其中A... 博文 来自：选择的专栏

2次三次4次Bezier曲线实践 阅读数 2001

程序实现此算法截图　2次三次4次Bezier曲线演示程序(无源码)QT4.6VS2008编译 下载... 博文 来自：dreamcs的专栏

光滑曲线拟合算法 阅读数 1703

/*二次抛物线法绘制曲线函数*/ voidpaowuxian(int*x,int*y,intn,unsignedintk){unsignedinti,j;floatt1... 博文 来自：运行Bug(RunB...

贝塞尔曲线拟合原理 阅读数 7221

1.什么是贝塞尔曲线？贝塞尔曲线所依据的最原始的数学公式，是早在1912年就广为人知的伯恩斯坦多... 博文 来自：沈春旭的博客

二维数据拟合算法 06-19

适用于测试采集到的离散数据的曲线拟合。 下载

求实时曲线拟合的算法

我一个小软件，要用到实时采集数据，并根据实时数据画出曲线（见下图），可是因为数据采集有误差，导... 论坛

proe齿轮渐近线曲线方程 阅读数 1878

直齿轮关系式ha=(hax+x)*mhf=(hax+cx-x)*md=m*zda=d+2*hadf=d-2*hfdb=d*cos(alpha)直齿... 博文 来自：Sday0202的专栏

GAN(拟合sin函数) 阅读数 220

pytorch之GAN的实现(拟合sin函数)importtorchimporttorch.nnasnnimportnumpyasnpimportmat... 博文 来自：朴素.无恙的博客

曲线渐开线方程 阅读数 874

看到有人问曲线渐开线方程推导。搜了下维基百科，发现，中文的翻译版本（中文链接https://zh.wikip... 博文 来自：stereohomolo...

渐开线python 阅读数 599

渐开线还是本科的时候学机械原理学过，当时是齿轮的渐开线，做实验需要数据集，昨天画了阿基米德... 博文 来自：ASD99193615...

Opencv 曲线拟合圆 阅读数 3987

此函数用于拟合一段弧线，求出其圆心和半径。具体原理请参考http://blog.csdn.net/liyuanbhu/artic... 博文 来自：u013351270...

C++ 过控制点的三次B样条曲线拟合 12-20

主要采用C++编程实现，过控制点的三次B样条曲线拟合，可以用于各种高级的曲线拟合方面。 下载

Bezier曲线原理 阅读数 2万+

一、原理： 贝塞尔曲线于1962年，由法国工程师皮埃尔·贝塞尔（PierreBézier）所广泛发表，他运... 博文 来自：I'm Hero

opencv拟合抛物线 阅读数 1844

简介 本篇主要是对opnecv函数cvSolve的熟悉笔记。这里只是简单用它来拟合抛物线。大致内容为... 博文 来自：羽凌寒

Bezier曲线曲面算法实现代码 01-29

文档包括bezier曲线曲面生成算法 的原理公式说明 以及编程实现。 文档的项目“Bezier”是使用 Microsoft VC++ 6.0 实现的，当然只要是配... 下载

Bezier曲线的递推(de Casteljau)算法 阅读数 2597

最近复习计算机图形学的曲线时,重新看了一下Bezier曲线,计算Bezier曲线上的点,可用Bezier曲线方程直... 博文 来自：honeymc的游...

圆弧与贝塞尔曲线互相转换 06-19

主要讲述了圆弧与贝塞尔曲线互相转换 贝塞尔曲线拆分多边形处理中，曲线与圆弧几乎拟合，可以不用分解为线段 下载

自己用c++写的3阶贝塞尔曲线 10-22

用了两种算法，公式法和递归算法 界面用OpenGL做的 刚学，里面还有B样条没完成，哪个高手做了在把它传上来给我学习一下 谢谢 O(∩_∩)O~ 下载

B样条曲线绘制、bezier曲线绘制（c语言实现） 05-21

B样条和Bezier曲线的动态绘制，采用opengl绘制曲线，实现曲线的交互绘制 下载

Bezier曲线的原理 及 二次Bezier曲线的实现 阅读数 4017

Bezier曲线的原理Bezier曲线是应用于二维图形的曲线。曲线由顶点和控制点组成，通过改变控制点坐... 博文 来自：无形的风专栏

Java设计模式学习06——静态代理与动态代理 阅读数 1万+

一、代理模式为某个对象提供一个代理，从而控制这个代理的访问。代理类和委托类具有共同的父类或... 博文 来自：小小本科生成...

centos 查看命令源码 阅读数 6万+

yum install yum-utils 设置源: [base-src] name=CentOS-5.4 - Base src - baseurl=http://vault.ce... 博文 来自：linux/unix

强连通分量及缩点tarjan算法解析 阅读数 55万+

强连通分量：简言之 就是找环（每条边只走一次，两两可达）孤立的一个点也是一个连通分量 使用t... 博文 来自：九野的博客

Matlab并行编程方法 阅读数 9万+

本文讲一下matlab中的并行方法与技巧。分为以下几个板块： 1. 什么东西好并行？ 2. 怎么并行？ 3. p... 博文 来自： Rachel Zhang...

魔兽争霸3冰封王座1.24e 多开联机补丁 信息发布与收集点 阅读数 1万+

畅所欲言！ 博文 来自： Smile_qiqi的专...

VirtualBox COM获取对象失败 阅读数 3万+

错误详情 1. 先来看看错误详情 获取 VirtualBox COM 对象失败.应用程序将被中断.Failed to instantiat... 博文 来自： 多点折腾少点...

jquery/js实现一个网页同时调用多个倒计时(最新的) 阅读数 41万+

jquery/js实现一个网页同时调用多个倒计时(最新的) 最近需要网页添加多个倒计时. 查阅网络,基本上都... 博文 来自： Websites

人脸检测工具face_recognition的安装与应用 阅读数 4万+

人脸检测工具face_recognition的安装与应用 博文 来自： roguesir的博客

Java设计模式14——中介者(Mediator)模式 阅读数 3767

一、定义用一个中介对象封装一系列对象的交互，中介者是多个对象不需要显示的相互作用，而且可以... 博文 来自： 小小本科生成...

R语言逻辑回归、ROC曲线和十折交叉验证 阅读数 4万+

自己整理编写的逻辑回归模板，作为学习笔记记录分享。数据集用的是14个自变量Xi，一个因变量Y的a... 博文 来自： Tiaaaaa的博客

Particle Filter Tutorial 粒子滤波：从推导到应用（一） 阅读数 5万+

前言： 博主在自主学习粒子滤波的过程中，看了很多文献或博客，不知道是看文献时粗心大意还是... 博文 来自： 知行合一

关于SpringBoot bean无法注入的问题（与文件包位置有关） 阅读数 14万+

问题场景描述整个项目通过Maven构建，大致结构如下： 核心Spring框架一个module spring-boot-b... 博文 来自： 开发随笔

eric_e 关注

原创	粉丝	喜欢	评论
113	234	130	143

等级： 博客 5 访问： 62万+

积分： 8115 排名： 4166

你尚未连接

Web 因你而不同。

我们会让你重新联机

- 请检查网络电缆是否已插好。

最新文章

CUDA的cublas 和 Intel的MKL 矩阵运算对比

cuda二维数组内存分配和数据拷贝（good）

CUDA使用二级指针表示二维数组

CUDA：并行计算实现矩阵相加

cuda教程（good）

个人分类

Python	2篇
逆向工程	1篇
可视化	4篇
破解	1篇
统计学	1篇

展开

归档

2019年3月	83篇
2019年2月	56篇
2019年1月	67篇
2018年12月	85篇
2018年11月	45篇

展开

热门文章

3个方法解决百度网盘限速 (2018-07-10)
阅读数 58809

C语言中的struct用法
阅读数 49893

图像去噪算法简介
阅读数 20799

常见的几种矩阵分解方式（LU分解、QR分解）
阅读数 20707

Chrome插件 - 突破百度网盘下载限制（大文件直接下载、使用迅雷下载）
阅读数 16524

最新评论

C语言中的struct用法
l_newbie：大佬！ ...

3个方法解决百度网盘限速 (201...
qq_40411735：https://share.weiyun.com/5DPwksb 最新的破解版 也能不限速下载 ...

使用遗传算法整定PID参数
weixin_42719268：请问博主你的系统函数是什么？还有给定的遗传算法搜索范围是多大 ...

occ+vtk显示igs模型
jiashou2471：严重性 代码 说明 项目 文件 行 禁止显示状态 错误(活动) namespace "BRepMesh."

详细了解加密狗的加密原理
stevenliu888：如何破译emmc上的加密锁，求教，感谢！ ...


CSDN学院


CSDN企业招聘