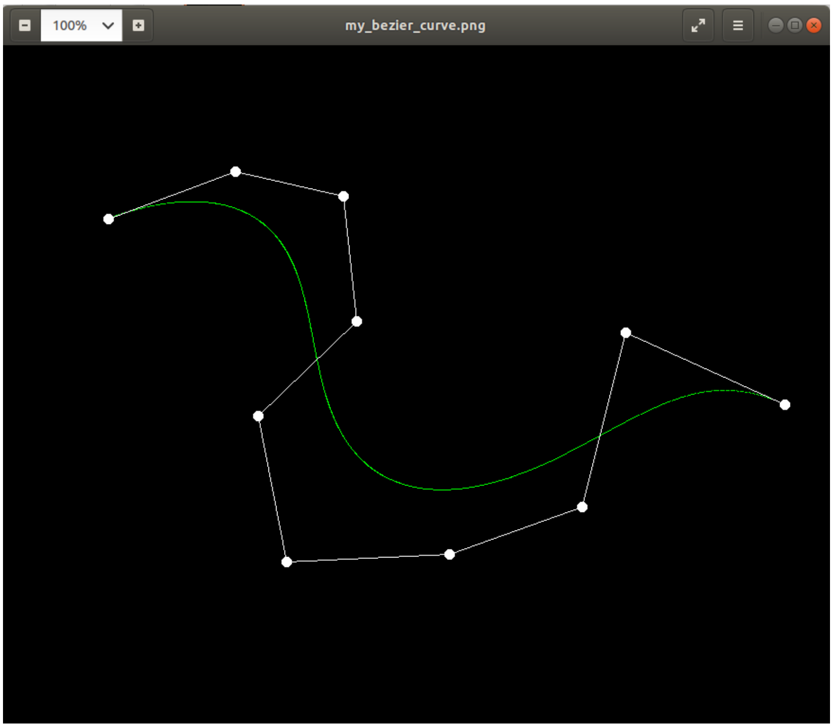
Assignment 6: Bézier曲线

Computer Graphics Teaching Stuff, Sun Yat-Sen University

- Due Date11月29号晚上12点之前，提交到zhangzk3@mail2.sysu.edu.cn邮箱。

> 在前面的作业中我们都是围绕渲染这一主题展开，但光学渲染并非图形学的全部，建模和动画亦是图形学的核心研究内容。其中的建模对当前的工业设计有着极其重要的意义，**小到螺丝、大到飞机火箭等这些物品外观形状的精准设计，都离不开图形学中的曲线曲面建模方法**。本次作业你将初步了解曲线建模的内容，实现de Casteljau算法来绘制Beizer曲线，绘制**任意**形状的平滑曲线。本次作业是本课程的最后一次作业。



1、总览

我们知道，对于某些特定的曲线（例如椭圆曲线、抛物线），它们有显式的数学表达式（即参数方程），因此可以很容易地利用它们的数学表达式来绘制相应形状的曲线。但在日常的实践中，我们更希望能够绘制任意形状的平滑曲线，因为极大部分的物体都是不规则的曲线形状（不规则但平滑，甚至有一定的曲率要求），没有一个固定的数学表达式来描述它。由此，Bézier曲线应运而生。Bézier曲线是计算机图形图像造型的基本工具，是图形造型运用得最多的基本线条之一，它通过控制顶点序列来创造、编辑图形，设计师们可以通过操控控制顶点的位置来调整曲线的形状，实现设计师与计算机交互式的图形设计。

Bézier曲线本质上是由调和函数根据控制点插值生成，其参数方程如下：

$$Q(t)=\sum_{i=0}^{n}P_iB_{i,n}(t),t\in[0,1] \tag{1}$$

其中Bézier曲线的控制点记为$P_i,i=0,\dots,n$，一共有$n+1$个控制顶点。上式是一个n次多项式，一共$n+1$项，每个控制点对应一项。多项式系数$B_{i,n}(t)$是Bernstein基函数，其数学公式为：

$$B_{i,n}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}, i=0,1,\ldots,n \tag{2}$$

公式(1)和公式(2)就是Bézier曲线的核心。可以看到，Bézier曲线的计算复杂度与控制顶点的数量息息相关，控制点越多，则多项式系数的计算（即公式(2)）需要越多的计算资源开销。因此一般不会直接暴力求解公式(2)，而是采用de Casteljau算法来高效地求解。

de Casteljau算法说明如下：

- 1、考虑一个$p_0,p_1,\ldots,p_n$为控制点序列的Bézier曲线，首先将相邻的点连接起来形成线段；
- 2、用$t:(1-t)$的比例划分每个线段，用线性插值法找到分割点；
- 3、对所有的线段执行上述操作，得到的分割点作为新的控制点序列，新序列的数量会减少一个；
- 4、如果新的序列只包含一个点，则返回该点，终止递归过程。否则，使用新的控制点序列并转到步骤1，如此递归下去。

de Casteljau算法的递归实现并不复杂，请仔细体会其中的奥妙。**在本次作业中，一开始你需要实现de Casteljau算法来绘制由4个控制点表示的Bézier曲线。然后正确实现该算法后，你需要修改一下代码以支持更多的控制点来绘制更复杂的Bézier曲线。**作业难度不会太大，请同学们放心。

2、代码框架

我们提供的代码框架非常简单，仅有一个`main.cpp`文件。你需要修改的函数是下面的两个：

- `bezier(const std::vector<cv::Point2f> &control_points, cv::Mat &window)`：

  在输入的参数中，`control_points`是控制点序列，`window`是绘制窗口。在该函数中，你需要实现使`t`在`0.0`到`1.0`的范围内进行迭代，并在每次迭代中使`t`增加一个微小值（`0.001`）。对于每一个需要计算的`t`，利用另一个函数`de_Casteljau`，得到Bézier曲线上的点。最后，将该点绘制在`window`上。

- `de_Casteljau(const std::vector<cv::Point2f> &control_points, float t)`：

  `control_points`是控制点序列，而`t`是插值参数。在该函数中，你需要实现de Casteljau算法，这是一个递归算法，所以你需要递归调用该函数，并设置恰当的递归终止条件。

一开始，你需要实现4个控制点的Bézier曲线。因此，我们在`main.cpp`实现了一个`naive_bezier`函数，它本质上就是直接计算公式(2)，4个控制点的曲线是三次Bézier曲线，它的表达式可以写成下面的形式：

$$Q(t)=(1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3 \tag{3}$$

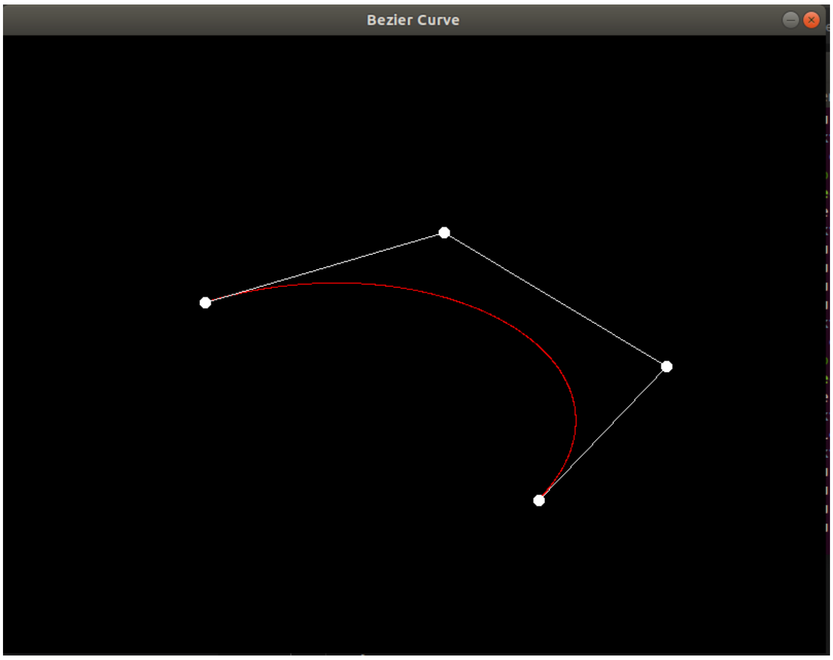
`naive_bezier`函数就是直接利用上面的公式直接计算Bézier曲线上的点，同学们可以仔细对比看看，体会一下Bézier曲线的求解和绘制过程。`main.cpp`的代码很少，其他的一些细节请同学们直接看代码框架。

3、编译

如果使用自己的系统而不是之前提供的虚拟机，本次程序作业中使用到的库为`OpenCV`，请确保该库的配置正确。如果使用我们提供的虚拟机，请在终端 (命令行) 以此输入以下内容：

```
mkdir build
cd build
cmake ..
make
```

执行`make`命令之后，就会调用相关的编译程序对当前的代码进行编译，编译出错会把信息输出到终端中，如有报错请仔细查看提供的报错信息并调试修正。编译成功之后，在终端输入`./BezierCurve`即可运行编译好的程序。运行时，程序将打开一个黑色窗口。**你可以用鼠标点击屏幕来选择一个控制点，选择了四个控制点之后，默认的代码框架将调用`naive_bezier`函数绘制Beizer曲线**。不出意外，你应该得到下面的结果：



其中，白色的点是控制顶点，白色的折线是控制点之间的线段，红色的曲线就是Beizer曲线。

4、作业描述与提交

（1）、作业提交

将PDF报告文件和代码压缩打包提交到zhangzk3@mail2.sysu.edu.cn邮箱，邮件及压缩包命名格式为：hw6_姓名_学号。

（2）、作业描述

本次作业要求同学们完成的工作如下所示：

Task 1、实现de Casteljau算法，并用它来绘制Beizer曲线。（60分）

```
cv::Point2f de_Casteljau(const std::vector<cv::Point2f> &control_points, float t)
{
    // TODO: Implement de Casteljau's algorithm
    return cv::Point2f();

}

void bezier(const std::vector<cv::Point2f> &control_points, cv::Mat &window)
{
    // TODO: Iterate through all t = 0 to t = 1 with small steps, and call de Casteljau's
    // recursive Bezier algorithm.


```

```
}
```

与`naive_bezier`不同，这里我们要求你绘制的Beizer曲线是**绿色**的。实现好之后，你需要在`main`函数中改一下调用的函数，将`naive_bezier`的调用注释掉，转而调用`bezier`函数，如下所示：

```
        if (control_points.size() == 4)
        {
            //naive_bezier(control_points, window);
            bezier(control_points, window);

            cv::imshow("Bezier Curve", window);
            cv::imwrite("my_bezier_curve.png", window);
            key = cv::waitKey(0);

            return 0;
        }
```

简述你是怎么做的，并贴上结果。

Task 2、在Task 1的基础上，调整一下代码以支持更多的控制点。（30分）

只需稍微修改一下代码框架即可，默认只支持4个顶点，请你调整成支持10个顶点，并贴上结果。

Task 3、谈谈你对Beizer曲线的理解。（10分）