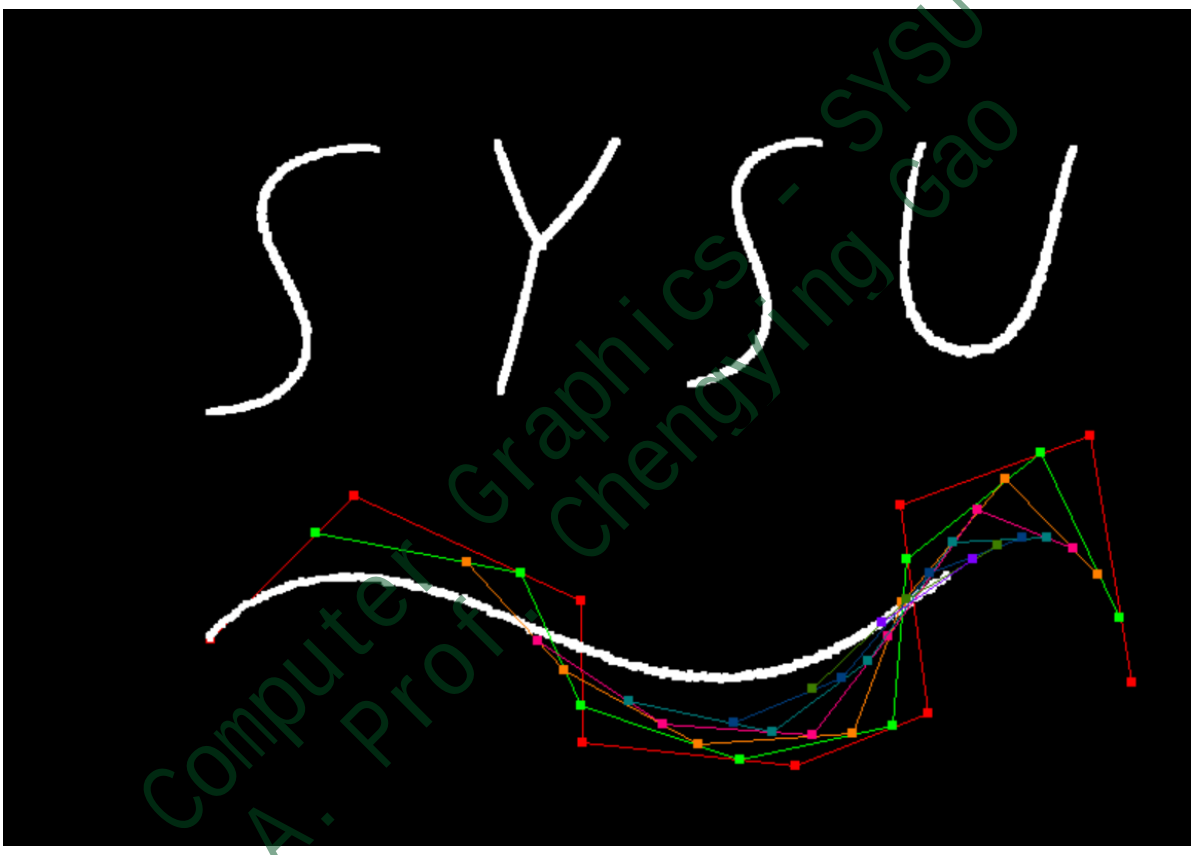
Assignment 5: Bézier Curve

Computer Graphics Teaching Stuff, Sun Yat-Sen University

Due Date: 具体截止日期见群公告

*Submission: Send the report (In **PDF** Format) to mailbox* (邮箱地址见群公告)

> 在计算机图形学领域，几何建模也是核心的研究主题。几何建模对工业制造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。在本次的作业框架下，你将手动实现Bezier曲线的生成算法，对曲线曲面的建模做一个初步的了解。



本次作业你们将实现的效果实例图

1、作业概述

本次作业的主题为Bézier曲线建模，要求你们上机动手实现Bézier曲线的生成算法。本次提供的作业框架已经帮你们搭好了绝大部分的工作，你只需实现核心的Bézier算法即可。下面对Bézier曲线做一个简述。

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：

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

其中Bézier曲线的控制点记为$P_i, i = 0, \ldots, n$，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。

Bézier曲线上顶点计算可以用两种方式计算，一种是直接求解上述的公式(1)和(2)得到给定t下的曲线顶点值；另一种是采用de Casteljau算法。本次作业要求你均实现两种算法，这两种算法的实现均不难，同学们请放心。

de Casteljau算法的核心伪代码可以描述如下：

- 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，终止迭代过程。否则，使用新的控制点序列并转到步骤1，如此迭代下去。

2、代码框架

关于本次作业的框架代码部署和构建，请仔细阅读`CGAssignment5/readme.pdf`文档，基本上与`CGAssignment4`没有太大的差别。本次作业依赖的第三方库为：

- **SDL2**：窗口界面库，主要用于创建窗口给并显示渲染的图片结果，本作业不需要你对这个库深入了解

这些第三方库同学们无需太过关注，我们的框架代码已经构建好了相应的功能模块。目录`CGAssignment5/src`存放本次作业框架的所有代码：

- **main.cpp**：程序入口代码，负责执行主要的渲染循环逻辑，无需修改；
- **WindowsApp(.h/.cpp)**：窗口类，负责创建窗口、显示结果、处理鼠标交互事件，无需修改；
- **Utils(.h/.cpp)**：算法核心代码，负责生成Bézier曲线，此文件绝大部分代码你无需修改。

在本作业框架的`Utils.h`文件中，我们用如下的`Point2D`来表示一个二维空间的点。`Point2D`的`color`用于控制绘制点的颜色，你无需关注和修改。在本次作业中，你会用到下面的`*`、`+`或`+=`重载的操作符函数，这些操作符的意义不言自明。`lerp`用于线性插值，在`Task2`你将会用到这个函数。`Line2D`你无需关注。

```
class Point2D
{
public:

    double x, y;
    std::array<Uint8, 3> color;

    Point2D(int _x, int _y) : x(_x), y(_y) {}

    Point2D operator*(const double &w) const
```

```
    {
        Point2D ans(x*w, y*w);
        return ans;
    }

    Point2D operator+(const Point2D &rhs) const
    {
        Point2D ans(x + rhs.x, y + rhs.y);
        return ans;
    }

    Point2D& operator+=(const Point2D &rhs)
    {
        x += rhs.x;
        y += rhs.y;
        return *this;
    }

    static Point2D lerp(const Point2D &p0, const Point2D &p1, const double &t)
    {
        // Linear interpolation
        double x = p0.x * (1.0 - t) + p1.x * t;
        double y = p0.y * (1.0 - t) + p1.y * t;
        return Point2D(x, y);
    }
};
```

`Utils.h`文件中的`BezierCurve`就是Bézier曲线核心代码，它存储当前的控制顶点，然后在相应的键盘操作时启动生成Bézier曲线。`main.cpp`的代码负责从键盘、鼠标获取相应的输入。这些代码你如果不太理解，那也没关系，你只负责在需要填充代码的部分编写代码即可。**编译运行本次作业的代码框架，你应该会得到一个黑色窗口，你可以做如下的鼠标、键盘操作**：

- 点击鼠标，窗口屏幕上会留下一个红点，并与前一个点击的点相连，这表示添加了一个控制顶点；
- 按下键盘的`c`键，会清除当前的控制顶点和已经生成的Bézier点；
- 按下键盘的`g`键，如果当前的控制顶点数目大于3个，那么就会进入Bézier曲线生成模式，在屏幕上显示生成的过程和相应的Bézier点（即白色的点），直到结束。

3、作业描述

请你按照下列顺序完成本次做的作业任务。

Task 1、根据Bézier曲线的定义来计算给定t值（$t \in [0,1]$）下的Bézier曲线上的点。

正如前面提到，Bézier曲线的定义公式如下所示，其中P_i为第i个控制顶点：

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

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

你需要填充代码的地方在`Utils.cpp`文件的`implementTask1`函数，如下所示：

```
Point2D BezierCurve::implementTask1(const std::vector<Point2D> &points, const
double &t) const
{
    //Task1: implement Bezier curve generation algorithm accroding to the
definition

    return Point2D(0, 0);
}
```

该函数的输入参数有两个，分别是控制顶点序列`points`和`t`，请根据上面的公式计算在给定`t`下的Bézier点，返回一个`Point2D`对象（你无需设置`Point2D`点的颜色）。实现好之后，尝试在屏幕上点击几个点，然后按下`g`键，看看是否生成了一条平滑的Bézier曲线，贴出你的效果，并简述你是怎么做的。

特别注意：

- 上述公式中的n是控制顶点数量减1，即n等于`points.size() - 1`；
- 在i从0到n的迭代过程中，每一次计算$B_{i,n}(t)$时，你没有必要每次都从头开始计算$i!$、$(n-i)!$、t^i和$(1-t)^{n-i}$，这是一个小优化；
- 你应该会用到的`Point2D`类的`*`、`+`或者`+=`操作符，这些操作符的含义详见`Point2D`类。

Task 2、实现更为简单、直观的de Casteljau算法来生成给定t值（$t \in [0, 1]$）下的Bézier曲线上的点。

de Casteljau算法在本文档的前面已经描述过，这里不再赘述。这个算法非常简单、直观。你需要填充代码的地方在`Utils.cpp`文件的`implementTask2`函数。该函数的输入参数和返回值与Task1一样。实现好之后，尝试在屏幕上点击几个点，然后按下`g`键，看看是否生成了一条平滑的Bézier曲线，贴出你的效果，并简述你是怎么做的。

```
Point2D BezierCurve::implementTask2(const std::vector<Point2D> &points, const
double &t) const
{
    //Task2: implement de Casteljau algorithm for Bezier curve
    // Note: you should use Point2D::lerp().

    return Point2D(0, 0);
}
```

特别注意：为了完成Task2，如下所示，你需要把`evaluate`函数中的`implementTask1`这一行删掉（或注释掉），把`implementTask2`这一行反注释，这样实现的de Casteljau算法才会生效。

```
void BezierCurve::evaluate(const double &t)
{
    // Visualization of auxiliary lines
    generateAuxiliaryLines(t);

    // Task1
    Point2D bp = implementTask1(m_controlPoints, t);

    // Task2
    //Point2D bp = implementTask2(m_controlPoints, t);

    m_beizerPoints.back().push_back(bp);
}
```

Task3、谈谈你对Bézier曲线的理解，Bézier曲线的缺点是什么？

注意事项:

- 将作业报告、源代码一起压缩打包，文件命名格式为：学号+姓名+HW5，例如19214044+张三+HW5.zip。
- **提交的文档请提交编译生成的pdf文件，请勿提交markdown、docx以及图片资源等源文件！**
- **提交代码只需提交源代码文件即可，请勿提交教程文件、作业描述文件、工程文件、中间文件和二进制文件（即删掉build目录下的所有文件！）。**
- 禁止作业文档抄袭，我们鼓励同学之间相互讨论（或者群里咨询助教），但最后每个人应该独立完成。