Bezier曲线与多项式曲线的转化、求值方法介绍与对比

软博21 沈冠霖 2021312593

1.理论基础

1.1 Bezier曲线转多项式曲线

$$公式推导：Bezier曲线方程C(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)，其中P_i为第i个控制顶点坐标。$$

$$多项式曲线方程D(t) = \sum_{j=0}^{n} Q_j t^j，其中Q_j为第j次项系数。$$

$$Bernstein基函数B_{i,n}(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}$$

$$设其第j项系数为B_{i,n,j}, 根据二项式定理，B_{i,n,j} = \begin{cases} 0 & j < i \\ (-1)^{j-i} \frac{n!}{i!(n-i)!} \binom{n-i}{j-i} & j \geq i \end{cases}$$

$$回到多项式曲线方程，有第j次项系数Q_j = \sum_{i=0}^{n} P_i B_{i,n,j} = \sum_{i=0}^{j} P_i (-1)^{j-i} \binom{n-i}{j-i} \frac{n!}{i!(n-i)!}$$

$$根据组合数性质，\binom{n-i}{j-i} = \frac{(n-i)!}{(n-j)!(j-i)!}，代入上个式子，有Q_j = \sum_{i=0}^{j} (-1)^{j-i} P_i \frac{n!}{i!(n-j)!(j-i)!}$$

算法设计：求解这个问题需要频繁进行从$0!$到$n!$的阶乘运算，因此考虑进行预处理，提前计算0到n的阶乘并且存储。

之后每次计算Q_j需要计算$j+1$项，每项的计算复杂度为$O(1)$，总共需要计算$\sum_{j=0}^{n} j+1$项。综上所述，算法复杂度为$O(n^2)$。

1.2 多项式曲线转Bezier曲线

$$公式推导：设Bernstein基函数第j项系数为B_{i,n,j}, 根据二项式定理，B_{i,n,j} = \begin{cases} 0 & j < i \\ (-1)^{j-i} \frac{n!}{i!(n-j)!(j-i)!} & j \geq i \end{cases}$$

将所有的$B_{i,n,j}$组成一个矩阵B, 使得$B[j][i] = B_{i,n,j}$，此时$B[i][j]$代表第j个$Bernstein$基函数在t^i的系数。

显然有B为$(n+1)*(n+1)$的下三角矩阵，且其对角线元素均不为0，其行列式也不为0，为可逆矩阵。

设$n+1$维向量p_x, p_y的第i维为$Bezier$曲线第i个控制顶点的x，y坐标，$n+1$维向量q_x, q_y的第i维为多项式曲线x，y坐标在t^i项的系数。

则有$Bp_x = q_x$，$Bp_y = q_y$。因为B为可逆矩阵，所以理论上有$p_x = B^{-1}q_x$，$p_y = B^{-1}q_y$。

算法设计：实际上，因为B为下三角矩阵，所以没有必要使用矩阵求逆算法，可以使用高斯消元算法来求解，具体步骤如下：

对于p向量第i维的求解，首先将前$i-1$维的解代入，求解其与B矩阵前$i-1$行的乘积之和，然后根据q向量的值反推p向量第i维的解。

这样每一维的求解花费$O(n)$的时间，求解每个方程组花费$O(n^2)$的时间，一共两个方程组，算法复杂度为$O(n^2)$。

1.3 多项式曲线求值

可以通过一遍循环完成求值：使用一个变量记录t^j的值，每次循环先更新t^j的值再求$Q_j t^j$，再加上去即可。算法复杂度$O(n)$。

1.4 Bezier曲线直接求值

首先，可以事先存储$0!$到$n!$的数值、t^0到t^n的数值、还有$(1-t)^0$到$(1-t)^n$的数值，这三个预先计算每个复杂度都是$O(n)$。

之后，可以通过一次循环得到最终结果：每次先根据存储好的数值计算出$B_{i,n}(t)$的数值，再求得$P_i B_{i,n}(t)$，再加上去。

综上所述，算法复杂度为$O(n)$。

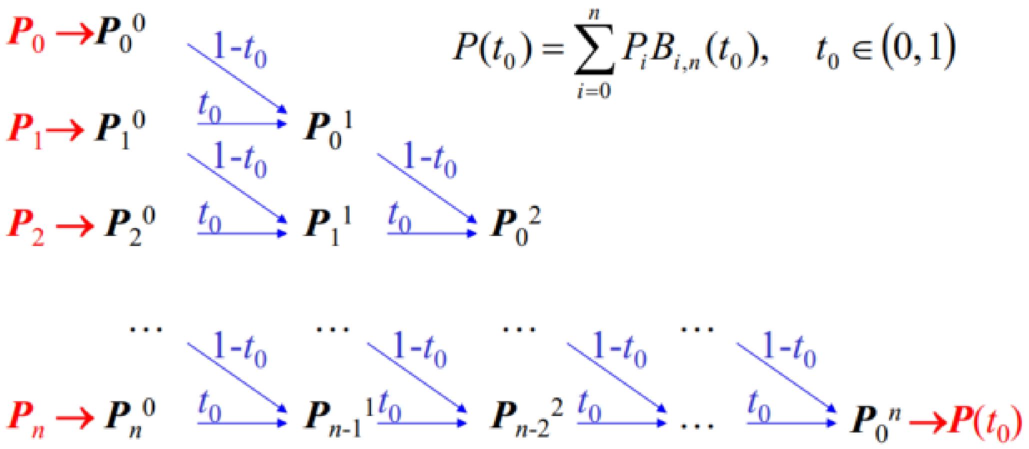
1.5 de Casteljau算法

$deCasteljau$算法是$Bezier$曲线求值的重要算法，推导和证明过程这里不再赘述，这里只陈述其表达式和计算复杂度。

$$P_i^k = \begin{cases} P_i & k = 0 \\ (1-t)P_i^{k-1} + tP_{i+1}^{k-1} & k = 1, 2 \ldots, n \quad i = 0, 1, \ldots, n-k \end{cases}$$

其中P_0^n为待求的曲线值。算法图解见下图。

可以看出，一共需要记录$n(n+1)/2$个中间状态，每个中间状态都需要运算，因此算法复杂度为$O(n)$。



2.实验设计和结果

2.1 实验环境

系统：Windows 10

CPU：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz   2.30 GHz

内存：32GB

编程语言：C++ 11

编程环境：vs code

2.2 多项式曲线和Bezier曲线互相转换

我首先设计了五组简单数据，进行对拍测试，数据如下，代码位于main.cpp文件的TestExamples()函数下。

阶数	贝塞尔曲线控制顶点	多项式曲线方程	是否正确
1	(0, 0), (1, 1)	x = t, y = t	是
1	(1, 3), (4, 5)	x = 3t + 1, y = 2t + 3	是
2	(0, 0), (1, 1), (2, 0)	x = 2t, y = -2t^2 + 2t	是
2	(1, 2), (3, 4), (5, 6)	x = 4t + 1, y = 4t + 2	是
3	(0, 0), (1, 1), (2, 1), (3, 0)	x = 3t, y = -3t^2 + 3t	是

之后我采用了随机对拍的方法进行测试。对于阶数n=1到20的20种情况，每种我都生成了10条随机贝塞尔曲线，保证控制顶点坐标在[-1e6,1e6]之间。我使用我实现的算法将贝塞尔曲线转化成多项式曲线，再转化为贝塞尔曲线，再转化为多项式曲线，如果两条贝塞尔曲线和多项式曲线一致，则说明转化正确。考虑到浮点运算的误差，我设定正确的标准为对应顶点欧几里得距离不超过1e-5。

经过测试，对于1到20的阶数，我的互相转化方法都正确。当阶数进一步提升，阶乘运算会导致浮点数溢出，转化错误。

综上所述，在阶数不是特别大（1到20）的时候，我的互相转化算法是正确的。

2.3 曲线求值精度

我仍然采用随机化的方法进行测试。

首先，要比较求值算法的精度，就应该有一个标准。我设置按照定义求曲线的值是完全准确的，其他方法可能有误差。度量误差的标准我选择是结果坐标的欧几里得距离，度量平均距离和最大距离两个指标。

其次，我们需要考虑多个维度的因素，一是曲线阶数n，二是顶点坐标范围，三是t的取值范围，我采用控制变量法，保证其他两个因素不变，只调整一个因素。

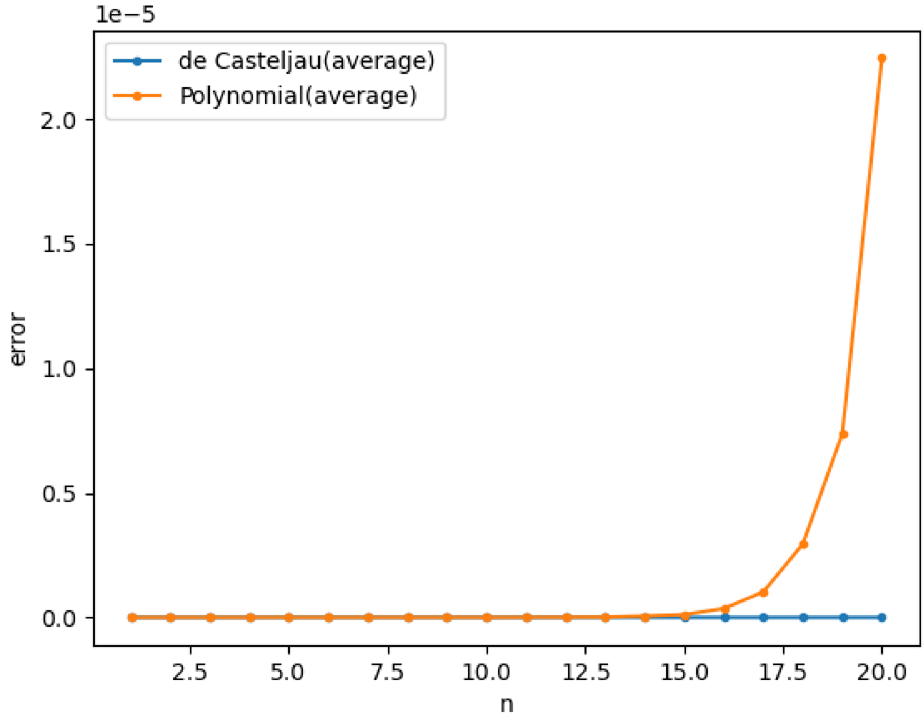
最终，为了更好地度量精度，我采用多次重复试验的方法，每种情况重复实验1000次。

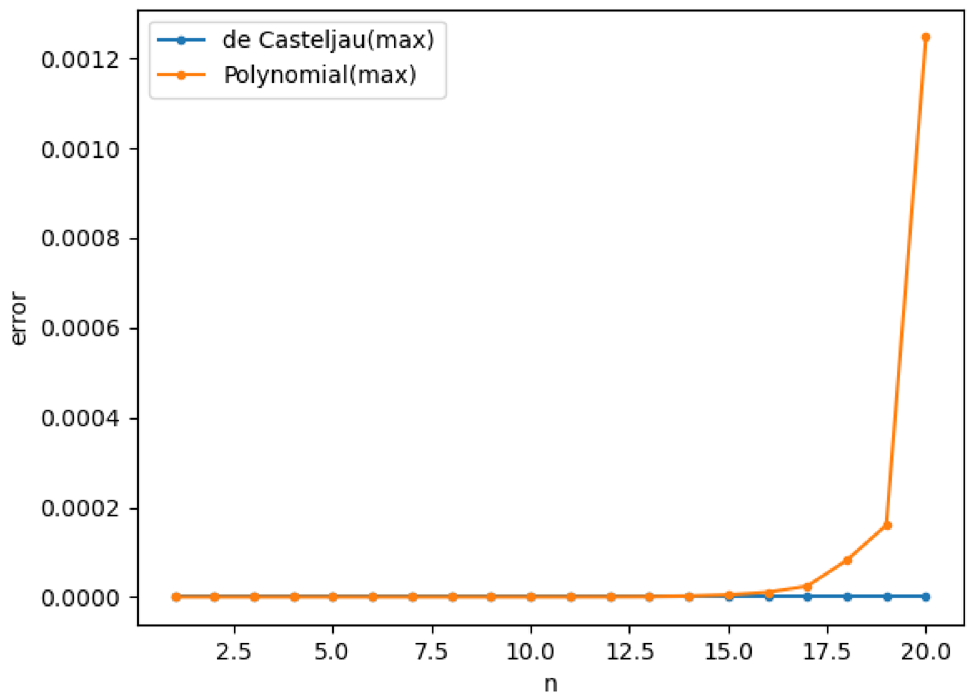
2.3.1 不同阶数下，求值精度对比

这里，我固定t的范围为[0, 1]，顶点坐标范围为[-1e6, 1e6]。

贝塞尔曲线求值

阶数	4	8	12	16	20
de Casteljau（平均误差）	1.53e-10	2.05e-10	2.48e-10	2.80e-10	3.12e-10
转化为多项式（平均误差）	1.85e-10	4.08e-10	6.47e-9	3.56e-7	2.25e-5
de Casteljau（最大误差）	5.21e-10	7.36e-10	8.87e-10	1.00e-9	1.15e-9
转化为多项式（最大误差）	6.79e-10	3.91e-9	2.18e-7	1.05e-5	1.25e-3

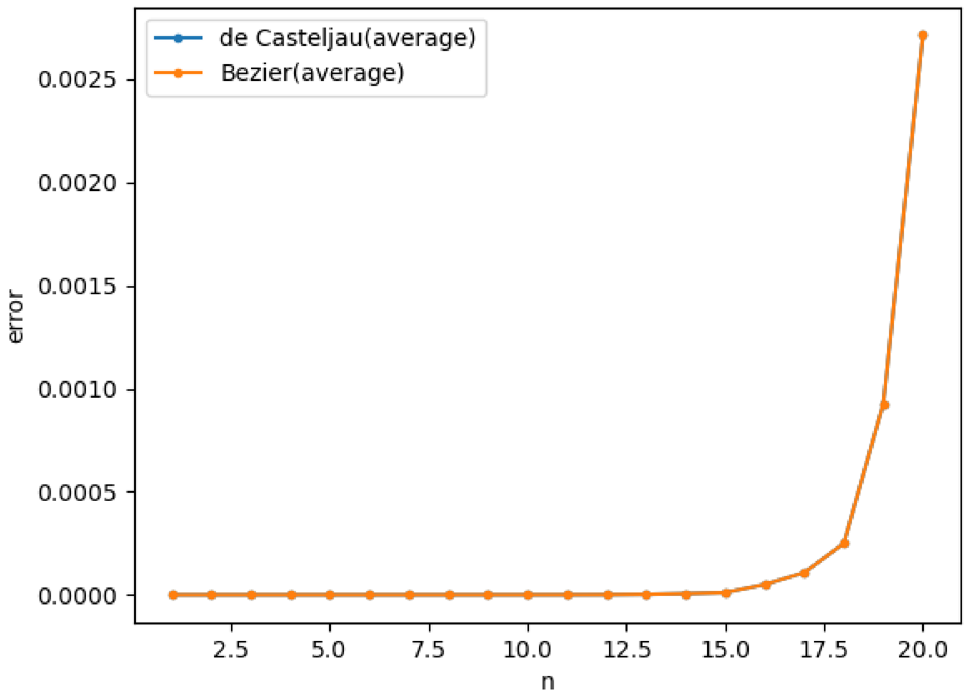


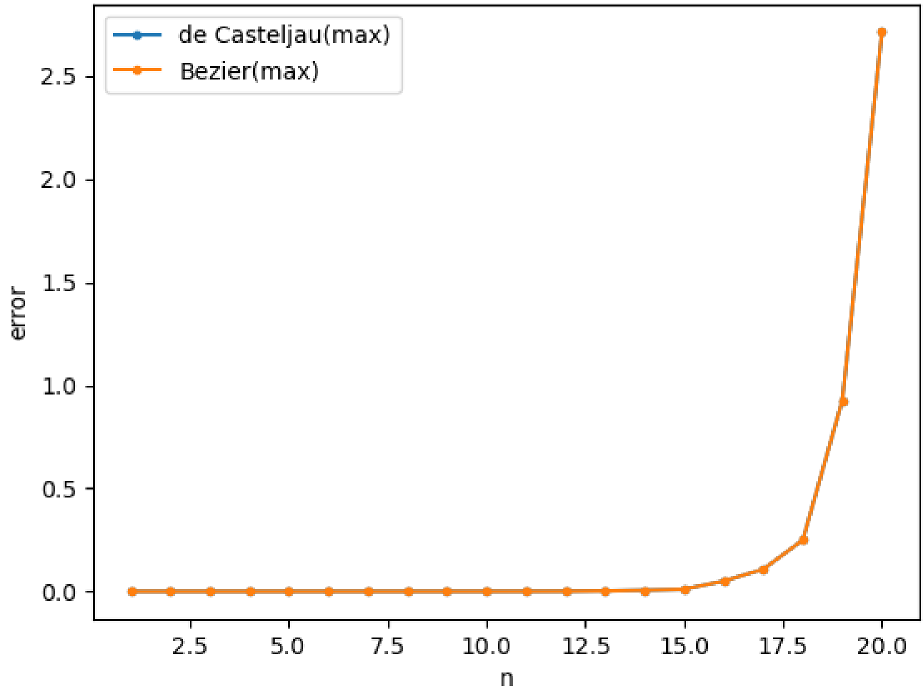


可以看出，在阶数n较小时，两种方法的误差均较小。在阶数n较大（大于15）时，无论是平均误差还是最大误差，de Casteljau算法的精度都远高于转化为多项式求解。随着阶数增大，转化为多项式求解的误差迅速上升，而de Casteljau算法的误差基本不变。

多项式曲线求值

阶数	4	8	12	16	20
转化为Bezier，然后de Casteljau（平均误差）	4.18e-10	7.85e-9	2.23e-7	5.03e-5	2.71e-3
转化为Bezier，然后按照定义（平均误差）	3.50e-10	7.81e-9	2.23e-7	5.03e-5	2.71e-3
转化为Bezier，然后de Casteljau（最大误差）	4.18e-7	7.85e-6	2.23e-4	5.03e-2	2.71
转化为Bezier，然后按照定义（最大误差）	3.50e-7	7.81e-6	2.23e-4	5.03e-2	2.71





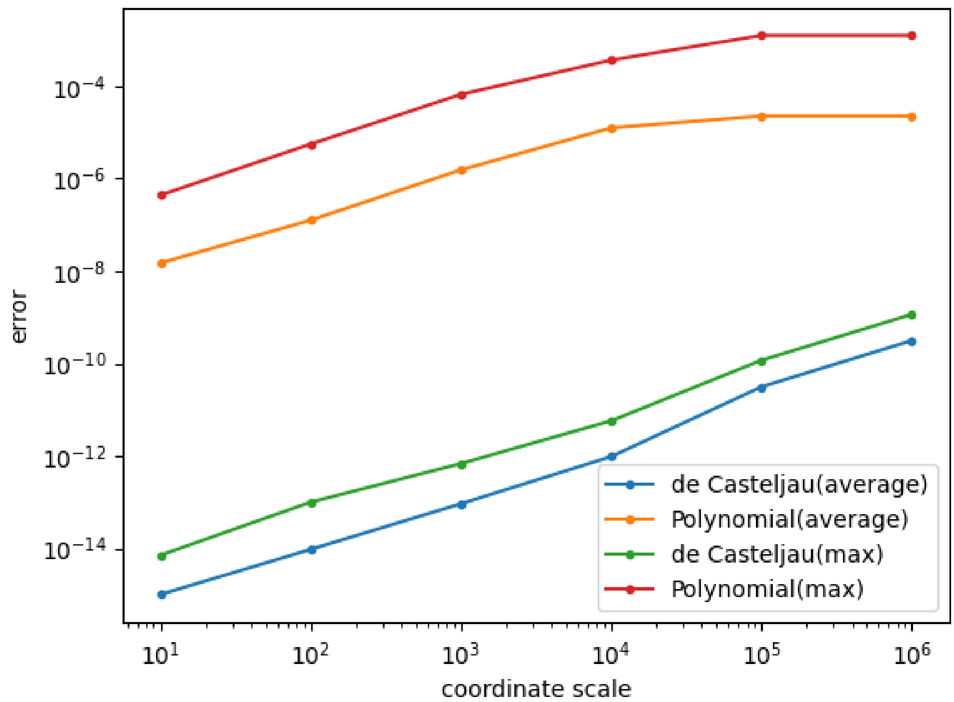
可以看出，两种方法的误差基本上差不多，而且在阶数较小的时候较小，随着阶数增加，两种方法的误差迅速增加。

2.3.2 不同顶点坐标范围下，求值精度对比

这里，我固定阶数为20，t范围为[0, 1]。

贝塞尔曲线求值

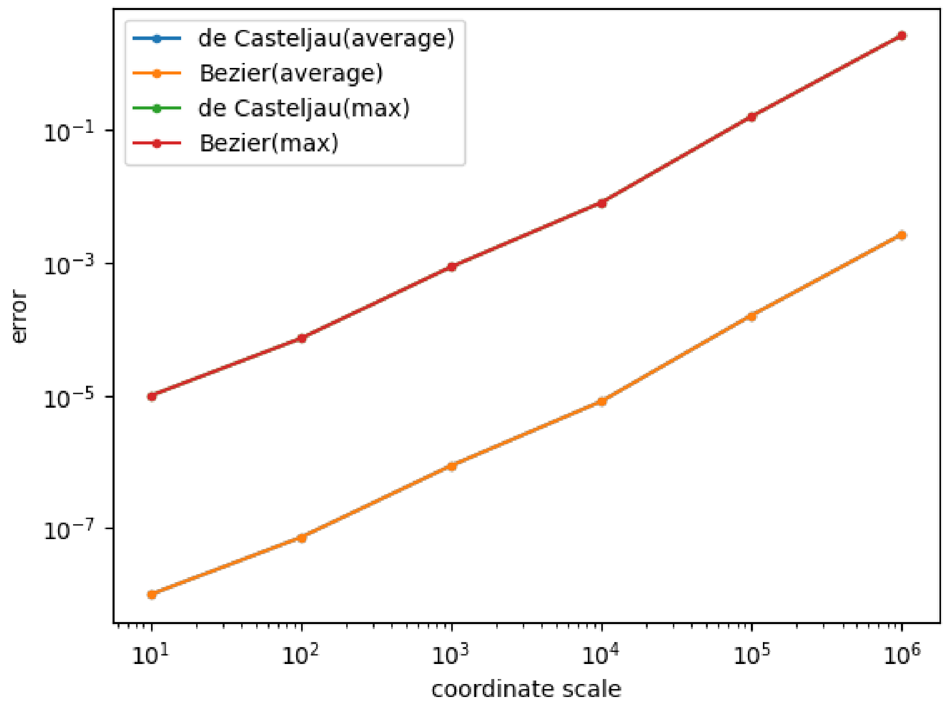
顶点坐标范围	[-10,10]	[-1e2,1e2]	[-1e3,1e3]	[-1e4,1e4]	[-1e5,1e5]	[-1e6,1e6]
de Casteljau（平均误差）	9.97e-16	9.55e-15	9.24e-14	9.70e-13	3.12e-11	3.12e-10
转化为多项式（平均误差）	1.49e-8	1.26e-7	1.54e-6	1.25e-5	2.25e-5	2.25e-5
de Casteljau（最大误差）	7.12e-15	1.00e-13	6.83e-13	5.75e-12	1.17e-10	1.15e-9
转化为多项式（最大误差）	4.37e-7	5.56e-6	6.60e-5	3.64e-4	1.25e-3	1.25e-3



可以看出，随着坐标范围的增大，误差也在逐渐增大。但是de Casteljau算法的精度一致高于转化为多项式再求解的精度。

多项式曲线求值

顶点坐标范围	[-10,10]	[-1e2,1e2]	[-1e3,1e3]	[-1e4,1e4]	[-1e5,1e5]	[-1e6,1e6]
转化为Bezier，然后de Casteljau（平均误差）	9.90e-9	7.25e-8	8.73e-7	8.16e-6	1.62e-4	2.71e-3
转化为Bezier，然后按照定义（平均误差）	9.90e-9	7.25e-8	8.73e-7	8.16e-6	1.62e-4	2.71e-3
转化为Bezier，然后de Casteljau（最大误差）	9.90e-6	7.25e-5	8.73e-3	8.16e-3	1.62e-1	2.71
转化为Bezier，然后按照定义（最大误差）	9.90e-6	7.25e-5	8.73e-3	8.16e-3	1.62e-1	2.71



可以看出，随着坐标范围的增大，误差也在逐渐增大。但是两种算法精度差不多。

2.3.3 不同顶点坐标范围下，求值精度对比

这里，我固定阶数为20，顶点坐标范围为[1000, 1000]。

贝塞尔曲线求值

t范围	[0,1]	[0,0.2]	[0.8,1]	[0.4,0.6]	[0.2,0.8]
de Casteljau（平均误差）	9.24e-14	9.55e-15	1.16e-13	7.23e-14	7.63e-14
转化为多项式（平均误差）	1.54e-6	1.26e-7	5.98e-6	9.38e-9	1.11e-7
de Casteljau（最大误差）	6.83e-13	1.00e-13	7.03e-13	4.55e-13	4.55e-13
转化为多项式（最大误差）	6.60e-5	5.56e-6	8.18e-5	1.02e-7	3.30e-6

可以看出，当t越小，求值精度就越高。而无论t如何取值，de Casteljau算法都比转化成多项式更精确。

多项式曲线求值

t范围	[0,1]	[0,0.2]	[0.8,1]	[0.4,0.6]	[0.2,0.8]
转化为Bezier，然后de Casteljau（平均误差）	8.73e-7	7.25e-8	3.90e-6	5.50e-9	6.86e-8
转化为Bezier，然后按照定义（平均误差）	8.73e-7	7.25e-8	3.90e-6	5.50e-9	6.86e-8
转化为Bezier，然后de Casteljau（最大误差）	8.73e-3	7.25e-5	3.90e-3	5.50e-6	6.86e-5
转化为Bezier，然后de Casteljau（最大误差）	8.73e-3	7.25e-5	3.90e-3	5.50e-6	6.86e-5

可以看出，当t越小，求值精度就越高。而无论t如何取值， 两种算法精度都大致相同。

2.4 求值速度

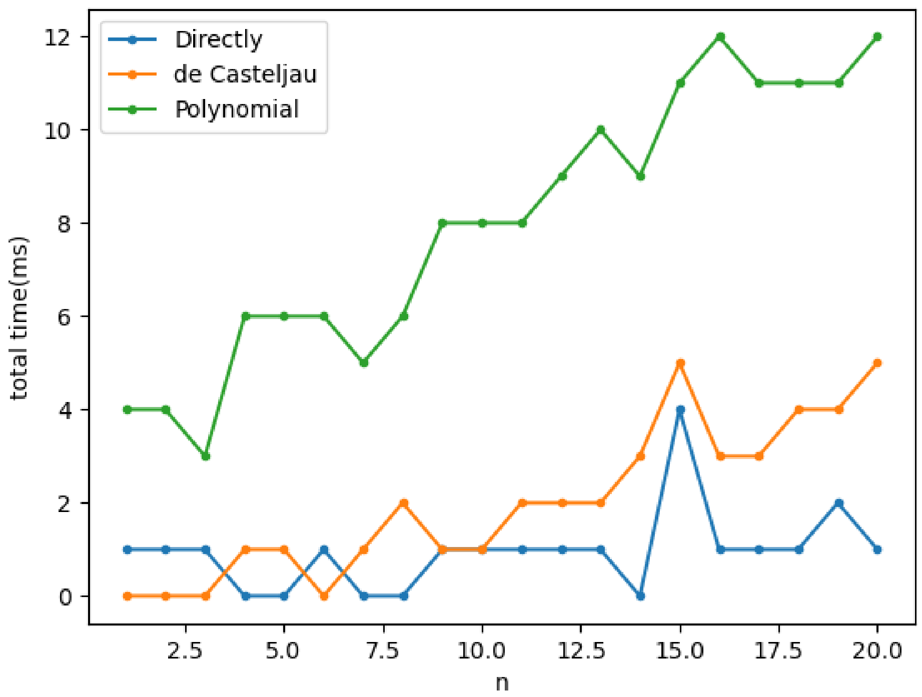
我仍然采用随机化的方法进行测试。

我固定t的范围为[0, 1]，顶点坐标范围为[-1e6, 1e6]。

我采用多次重复试验的方法，每种情况重复实验1000次，计算总时间。

贝塞尔曲线求值

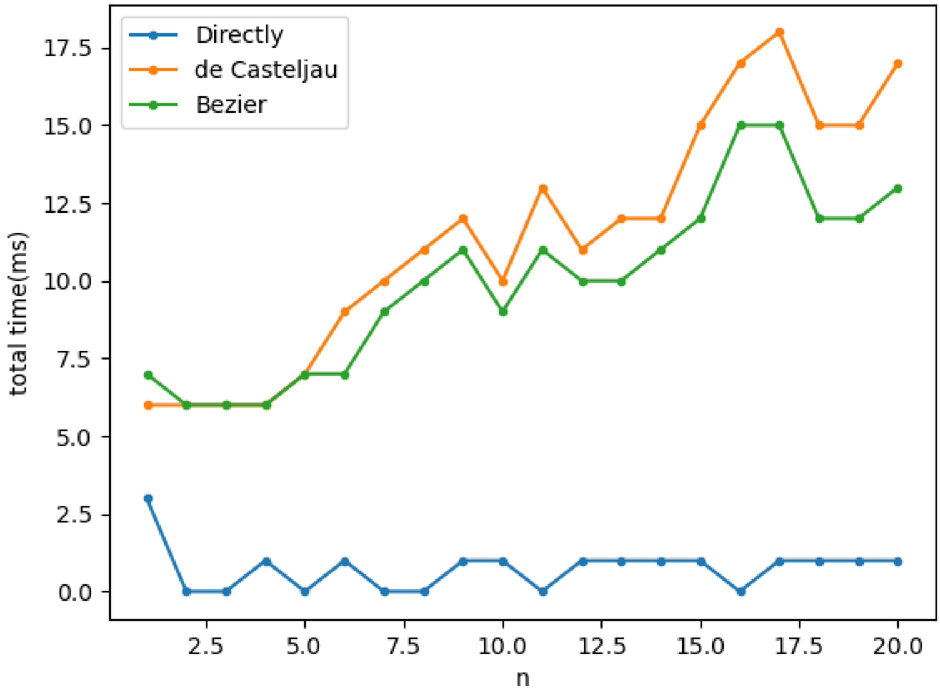
阶数	4	8	12	16	20
直接求解（总时间，单位：毫秒）	0	0	1	1	1
de Casteljau（总时间，单位：毫秒）	1	2	2	3	5
转化为多项式（总时间，单位：毫秒）	6	6	9	12	12



可以看出，时间效率上，直接求解>de Casteljau算法>转化为多项式

多项式曲线求值

阶数	4	8	12	16	20
直接求解（总时间，单位：毫秒）	1	1	1	1	1
转化为Bezier，然后de Casteljau（总时间，单位：毫秒）	6	10	11	17	17
转化为Bezier，然后按照定义（总时间，单位：毫秒）	6	10	10	14	13



可以看出，时间效率上，直接求解>转化为Bezier曲线再直接求解>转化为贝塞尔曲线再de Casteljau算法

3.分析和总结

3.1 正确性

首先，从互相转化算法的正确性上，无论是我设计的数据还是随机生成的数据，我都能保证对应点误差在1e-5以内，说明互相转化算法正确。

之后，从求值算法的正确性上，以直接求值为基准，在绝大多数情况下，可以保证求值的精度在1e-3以内，相对坐标点的范围来说并不是很大，可以说明我的求值算法都实现正确。

3.2 求值效率和准确率

对于Bezier曲线，直接按照定义求解是速度最快、准确率最高的，毕竟复杂度只有O(n)。使用de Casteljau算法速度和准确率其次，因为是O(n^2)的算法，且不比直接按照定义求解差特别多。而先转化为多项式再求值则在速度和准确率上都比较差，不仅有O(n^2)的运算开销，还有两种数据结构相关的其他系统开销。

对于多项式曲线，直接求解无论是速度还是准确率都是很高的。转化为Bezier曲线，无论是用哪种方法求解，速度和准确率都差不多，毕竟都需要O(n^2)的运算开销和数据结构相关的系统开销。

在实际生产中，Bezier曲线只需要定义控制点即可，而多项式曲线的定义更为抽象，Bezier曲线更为实用。而Bezier曲线的求解上，转化为多项式无论是速度还是精确度都不够；按照定义求解虽然速度和准确率都略高，但是要求阶乘，而阶乘随着n的增加会产生溢出；de Casteljau算法则是一个递推算法，不会有溢出情况，而且其速度和准确率都不逊色直接求解太多，因此是首选。综上所述，在实际生产中，应当采用Bezier曲线，用de Casteljau算法进行求值。