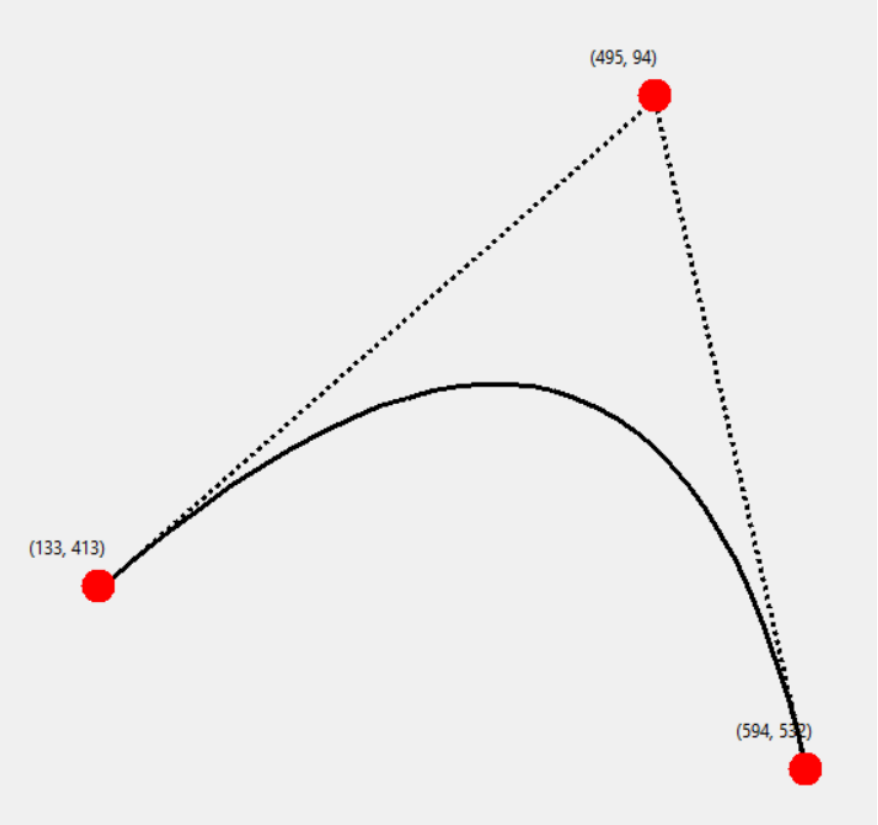
Laporan Tugas Kecil 2

IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer



Rafiki Prawhira Harianto	13522065
M Hanief Fatkhan Nashrullah	13522100

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

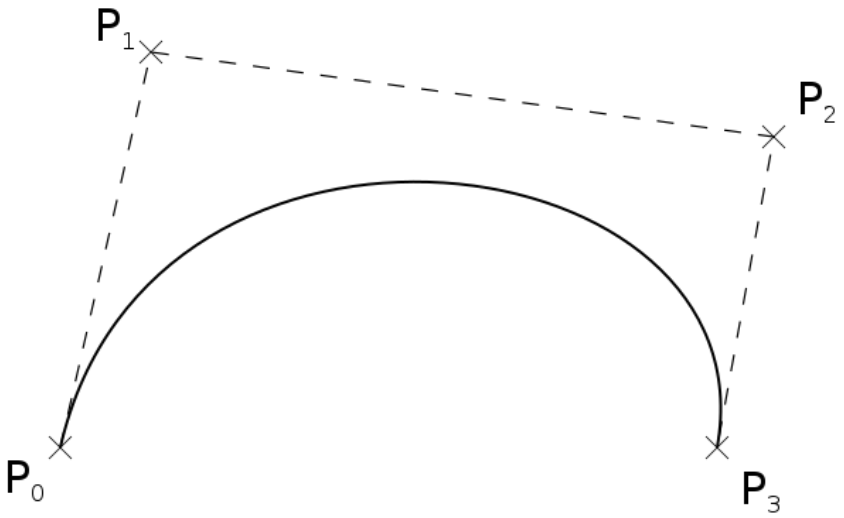
2024

Daftar Isi

Deskripsi Persoalan

Kurva Bézier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva. Cara membuatnya cukup mudah, yaitu dengan menentukan titik-titik kontrol dan menghubungkannya dengan kurva. Kurva Bézier memiliki banyak kegunaan dalam kehidupan nyata, seperti pen tool, animasi yang halus dan realistis, membuat desain produk yang kompleks dan presisi, dan membuat font yang indah dan unik. Keuntungan menggunakan kurva Bézier adalah kurva ini mudah diubah dan dimanipulasi, sehingga dapat menghasilkan desain yang presisi dan sesuai dengan kebutuhan.

Sebuah kurva Bézier didefinisikan oleh satu set titik kontrol P0 sampai Pn , dengan n disebut order (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva. Pada gambar 1 diatas, titik kontrol pertama adalah P0, sedangkan titik kontrol terakhir adalah P3. Titik kontrol P1 dan P2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk



Gambar 1. Kurva Bézier dengan 4 Poin Kontrol

Sumber : https://en.wikipedia.org/wiki/Bézier_curve

Secara eksplisit, kurva Bézier didefinisikan dengan persamaan

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^{i} P_i \qquad (1)$$

Dengan t bernilai [0,1] sebagai parameter yang menggambarkan besar bagian dari titik kontrol awal dan n adalah banyak dari titik kontrol dikurangi dengan satu.

Contohnya untuk $n = 3$ atau 4 Titik kontrol:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3 \qquad t \in [0,1]$$

Untuk $n = 4$ atau 5 Titik kontrol:

$$B(t) = (1-t)^4 P_0 + 4(1-t)^3 t P_1 + 6(1-t)^2 t^2 P_2 + 4(1-t)t^3 P_3 + t^4 P_4 \qquad t \in [0,1]$$

Pembentukan persamaan akan semakin rumit seiring dengan bertambahnya titik. Maka dari itu, diperlukan suatu algoritma lain untuk meningkatkan efisiensi dalam pembuatan kurva Bézier. Dalam pengimplementasian kali ini, akan digunakan algoritma titik tengah berbasis *Divide and Conquer*.

Algoritma Brute Force

Sebelum menggunakan algoritma *Divide and Conquer* dalam membentuk Kurva Bezier, perlu adanya algoritma untuk membanding efektivitas algoritma tersebut. Pembanding yang paling mudah adalah menggunakan algoritma *Brute Force*.

Algoritma *Brute Force* merupakan salah satu pendekatan paling sederhana dalam menyelesaikan suatu persoalan. Algoritma ini menyelesaikan persoalan dengan cara yang langsung, jelas, dan intuitif. Namun, kekurangan utama dari algoritma *Brute Force* adalah cenderung lambat dan kurang kreatif dibandingkan dengan algoritma lainnya, sehingga sering kali disebut sebagai algoritma naif. Meskipun begitu, algoritma *Brute Force* tetap menjadi pilihan yang handal sebagai alat perbandingan dalam mengevaluasi kecepatan dan efisiensi algoritma lainnya.

Pada program ini, algoritma *Brute Force* memanfaatkan definisi eksplisit Kurva Bezier yang telah dijelaskan pada persamaan (1). Implementasi terdiri dari dua fungsi utama. Fungsi pembantu, binomialCoeff, yaitu koefisien binomial antara suatu bilangan n dan k. Fungsi main, BezierBruteForce, menambahkan poin-poin sesuai interval t pada kurva. Implementasi dari BezierBruteForce adalah sebagai berikut:

1. Masukkan poin-poin kontrol pada variabel sementara.
2. Inisialisasi n sebagai jumlah poin kontrol.
3. Inisialisasi tNew = 0.
4. Cari poin pada tNew menggunakan definisi eksplisit Kurva Bezier, yaitu poin memiliki nilai $\sum_{i=0}^{n} binomialCoeff(n,i)(1 - tNew)^{n-i} tNew^{i}$.
5. Tambahkan poin ke hasil kurva.
6. Tambahkan tNew dengan t, kemudian ulangi langkah 4-5 sampai tNew >= 1.

Untuk fungsi pembantu binomialCoeff(n,k) menggunakan pendekatan rekursif dan berupa formula standar untuk mencari koefisien binomial. Implementasinya yaitu,

1. Basis: Jika k = 0 atau n = k, keluarkan 1.
2. Rekurens: Keluakarkan binomialCoeff(n-1,k-1) + binomialCoeff(n-1,k).

Algoritma Divide and Conquer

Algoritma *Divide and Conquer* merupakan sebuah algoritma yang membagi sebuah persoalan menjadi beberapa upa-persoalan sebelum menyelesaikan seluruh upa-persoalan. Sesuai dengan namanya, *divide*, membagi sebuah persoalan yang besar menjadi persoalan yang lebih kecil dari ukuran semula, kemudian *conquer*, menaklukan (menyelesaikan) masing masing upa-persoalan. Setelah di-*divide* dan di-*conquer*, hasil solusi setiap upa-persoalan digabungkan kembali *(combine)* yang membentuk solusi dari persoalan semula.

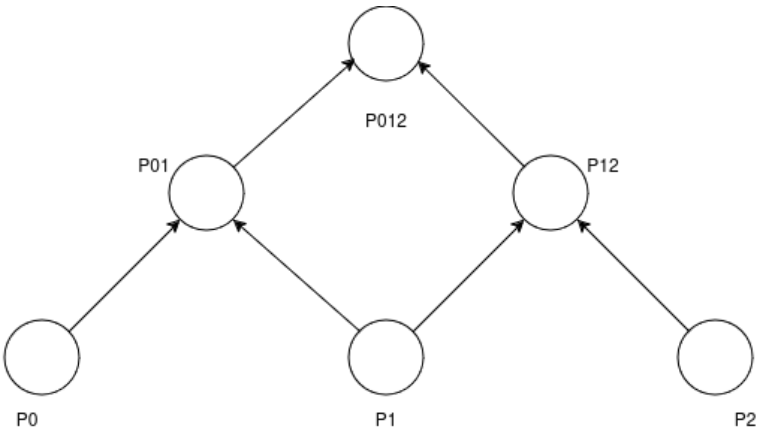
Berikut adalah skema umum dari Algoritma *Divide and Conquer*

```
procedure DIVIDEandCONQUER (input P : problem, n : integer)
{ Menyelesaikan persoalan P dengan algoritma Divide and Conquer
Masukan: masukan persoalan P berukuran n
Luaran: solusi dari persoalan semula }
Deklarasi
    r : integer
Algoritma
    if n ≤n0 then {ukuran persoalan P sudah cukup kecil }
        SOLVE persoalan P yang berukuran n ini
    else
        DIVIDE menjadi r upa-persoalan, P1,P2,...,Pr yang masing-masing
        berukuran n1, n2, ..., nr
        for masing-masing P1, P2, …, Pr , do
            DIVIDEandCONQUER(Pi, ni)
        endfor
        COMBINE solusi dari P1, P2, ..., Pr menjadi solusi persoalan semula
    endif
```

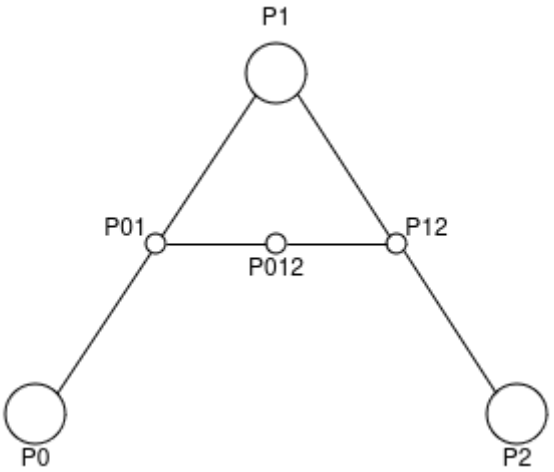
Sumber : Algoritma Divide and Conquer (Bagian 1) Oleh Rinaldi Munir

Dalam persoalan pembentukan kurva Bézier, dapat diterapkan algoritma *Divide and Conquer*. Algoritma *Divide and Conquer* dalam persoalan ini dapat diterapkan dengan cara membagi persoalan menjadi tiga bagian, yaitu titik tengah, titik sebelum titik tengah, dan titik setelah titik tengah. Kedua bagian setelah dan sebelum titik tengah dapat dibagi lagi menjadi tiga bagian dengan konsep yang sama. Persoalan akan akan diselesaikan ketika sudah cukup kecil atau dalam konteks persoalan ini jumlah iterasi pembentukan kurva sudah sampai batas yang diinginkan. Ketika sudah mencapai batas, titik tengah dari setiap upa-persoalan digabungkan menjadi satu solusi. Setiap titik tengah dicari dengan menggunakan algoritma De Casteljau.

Algoritma De Casteljau merupakan salah satu algoritma yang dapat digunakan untuk membentuk kurva Bézier. Dalam persoalan ini, hanya sebagian dari algoritma De Casteljau yang digunakan untuk membentuk kurva Bézier, yaitu *Bézier Curve Subdivision*. Pembagian ini dapat dilakukan untuk menemukan titik kontrol dan titik tengah dari sembarang titik kontrol. Berikut adalah graf yang dapat menjelaskan bagaimana cara mencari *subdivision of Bézier curve.*



Gambar 2. Graf *Subdivision of Bézier curve*
Sumber : Dokumentasi penulis



Gambar 3. Garis representasi dari graf
Sumber : Dokumentasi penulis

Setiap titik kontrol, yaitu P0, P1, dan P2 memiliki titik tengah. Titik tengah dari P0 dengan P1 adalah P01, dan titik tengah dari P1 dengan P2 adalah P12. Titik tengah dari ketiga titik kontrol tersebut adalah titik tengah P01 dengan P02, yaitu P012. Algoritma ini dapat di generalisasi untuk sembarang jumlah titik kontrol. Setelah didapatkan titik tengah, titik kontrol baru bisa didapatkan dengan mengambil simpul (titik) terluar dari graf seperti pada Gambar 2. Maka, titik kontrol baru sebelum titik tengah P012 adalah P0, P01, dan P012. Titik kontrol baru setelah titik tengah adalah P012, P12, dan P2.

Berikut adalah langkah implementasi algoritma *Divide and Conquer* dalam persoalan pembentukan kurva Bézier, langkah langkah ini dilakukan secara rekursif:

1. Periksa apakah level iterasi sudah melewati batas iterasi
2. Jika sudah, hentikan rekursi, lakukan langkah 8
3. Jika belum, buat graf *subdivision of Bézier curve*
4. Buat titik kontrol untuk titik sebelum titik tengah dan untuk setelah titik tengah

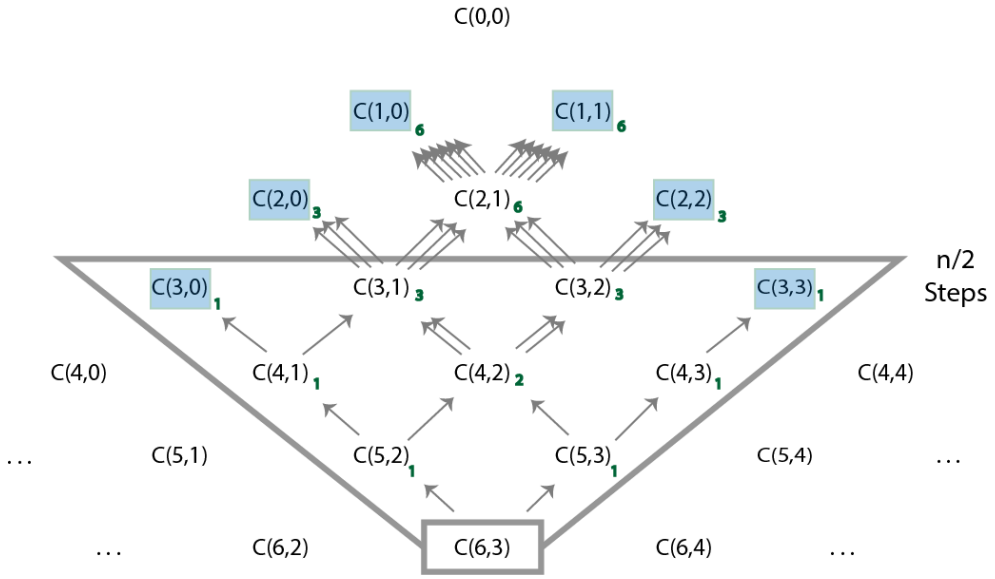
5. Lakukan langkah 2 dengan titik kontrol sebelum tengah dengan level iterasi bertambah satu
6. Masukkan titik tengah pada kumpulan solusi
7. Lakukan langkah 2 dengan titik kontrol setelah titik tengah dengan level iterasi bertambah satu
8. Masukkan titik kontrol semula pertama dan terakhir pada solusi

Perbandingan Algoritma

Algoritma *Brute Force* memiliki dua bagian algoritma dalam implementasinya. Bagian pertama adalah algoritma koefisien binomial, dan bagian kedua adalah algoritma utama *Brute Force*. Algoritma koefisien binomial diperlukan untuk membentuk suatu persamaan eksplisit dari kurva Bézier. Algoritma utama *Brute Force* merupakan implementasi dari persamaan (1).

Algoritma koefisien binomial mencari hasilnya secara rekursif. Algoritma ini akan memanggil dirinya sendiri sampai mencapai basis yaitu n=k atau k=0. Pemanggilan ini akan terjadi setidaknya sebanyak n/2 kali. Untuk satu kali pemanggilan, algoritma ini akan memanggil dirinya sendiri sebanyak dua kali. Maka setiap pemanggilan akan berkembang menjadi dua kali lipat. Maka kompleksitas waktu algoritmanya adalah $O(2^n)$.

Algoritma *Brute Force* melakukan iterasi pada kalang pertama sebanyak 1 dibagi *increment* kali dan melakukan iterasi pada kalang kedua sebanyak n (jumlah titik kontrol) kali. Maka kompleksitas waktunya adalah O(mn). Jika dalam kalang kedua algoritma koefisien binomial dipanggil, maka kompleksitas waktunya juga akan meningkat. Kompleksitas waktu dari keseluruhan pembentukan kurva Bézier melalui algoritma *Brute Force* adalah $O(2^n mn)$.



Gambar 4. Langkah pemanggilan algoritma koefisien binomial

Sumber: *Time-complexity of recursive algorithm for calculating binomial coefficient, Answer by AbcAeffchen*

Algoritma *Divide and Conquer* memiliki empat bagian dalam implementasinya. Bagian pertama adalah algoritma pencarian titik tengah. Algoritma ini menambahkan dua titik koordinat lalu membaginya dengan dua. Algoritma ini memiliki kompleksitas waktu O(1). Bagian kedua adalah algoritma *splitter*. Algoritma ini membentuk titik tengah iterasi pertama seperti pada Gambar 2. Algoritma ini memiliki kompleksitas waktu O(n). Selanjutnya ada algoritma *lineMidPoint*. Algoritma ini memanggil algoritma *splitter* sebanyak n-1 kali untuk membentuk graf *subdivision of Bézier curve*. Algoritma ini memiliki

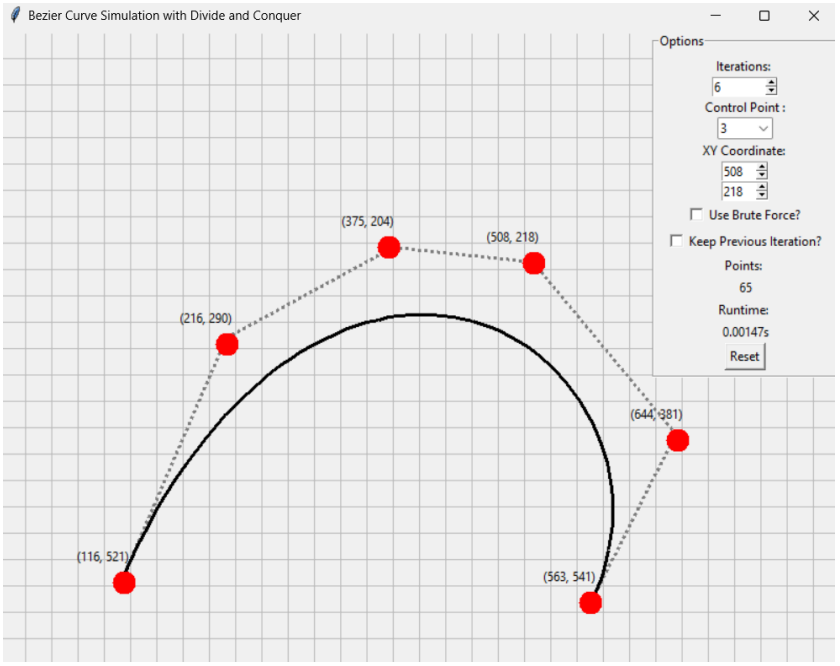
kompleksitas waktu $O(n^2)$ (sudah memperhitungkan pemanggilan *splitter*). Terakhir, terdapat algoritma *Divide and Conquer*. Algoritma ini memanggil *lineMidPoint* dan juga bekerja secara rekursif. Algoritma ini memanggil dirinya sendiri sebanyak dua kali, yaitu untuk pembentukan kurva sebelum titik tengah dan pembentukan kurva setelah titik tengah. Pemanggilan rekursi ini akan memiliki kompleksitas waktu $O(2^n)$. Jika memperhitungkan pemanggilan *lineMidPoint* di dalamnya, algoritma *Divide and Conquer* akan memiliki kompleksitas waktu $O(2^n n^2)$

Jika kedua algoritma dibandingkan dari kompleksitas waktunya, waktu yang diperlukan tidak terlalu berbeda. Algoritma *Brute Force* memiliki kompleksitas waktu $O(2^n mn)$, dan algoritma *Divide and Conquer* memiliki kompleksitas waktu $O(2^n n^2)$. Efisiensi dari algoritma *Brute Force* menjadi lebih efisien ketika m (1 dibagi *increment,* dengan $increment \leq 1$) lebih kecil dibandingkan n. Maka, untuk jumlah jumlah titik Bézier yang memerlukan jumlah *increment* yang kecil, algoritma *Brute Force* akan menjadi lebih efisien. Namun, untuk jumlah titik yang besar dan memerlukan *increment* yang lebih besar dibandingkan jumlah titik kontrol, algoritma *Divide and Conquer* akan menjadi lebih efisien.

Implementasi Bonus

A. Kurva untuk n Titik Kontrol

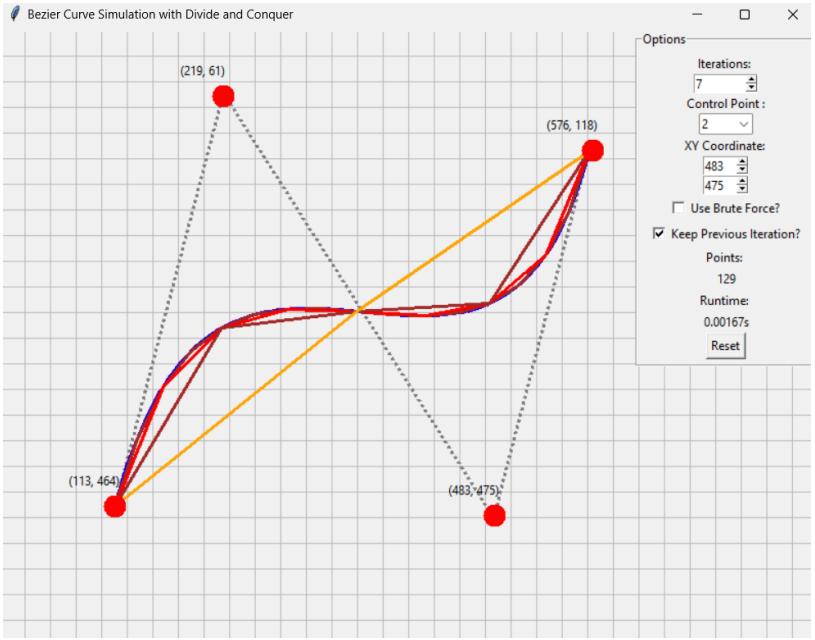
Menggunakan algoritma *Brute Force* maupun *Divide and Conquer,* program dapat membuat kurva dengan jumlah poin kontrol berapapun. Generalisasi *Brute Force* dilakukan dengan definisi eksplisit Kurva Bezier, sedangkan Generalisasi *Divide and Conquer* dilakukan menggunakan *subdivision* pada kurva sehingga bisa dipakai untuk sebanyak n titik. Cara menambahkan poin pada program yaitu dengan klik dua kali pada kanvas program.



Gambar 5. Kurva Bezier dengan 6 Poin Kontrol
Sumber : Dokumentasi Penulis

B. Visualisasi Pembuatan Kurva

Program dapat membuat visualisasi pembuatan kurva dengan tidak menghilangkan garis pada iterasi-iterasi sebelumnya. Klik checkbox “Keep Previous Iteration?” pada Options, kemudian turunkan atau naikkan opsi iterations pada Options.



Gambar 5. Visualisasi Pembuatan Kurva Bezier
Sumber : Dokumentasi Penulis

Lampiran

A. Tabel checklist

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✓	
2. Program dapat melakukan visualisasi kurva Bézier.	✓	
3. Solusi yang diberikan program optimal.	✓	
4. **[Bonus]** Program dapat membuat kurva untuk n titik kontrol.	✓	
5. **[Bonus]** Program dapat melakukan visualisasi proses pembuatan kurva.	✓	

B. Source program python

BezierBruteForce.py

```
from PointListClass import *

def binomialCoeff(n,k):
    if k==0:
        return 1
    if n==k:
        return 1
    else:
        return binomialCoeff(n-1,k-1)+binomialCoeff(n-1,k)


def BezierBruteForce(t:float,result:Line,line:Line):
    temp = line.head
    n = line.length
    tNew = 0;
    while tNew<=1:
        px = 0
        py = 0
        for i in range(n):
            bez = ((1-tNew)**(n-i-1))*(tNew**i)*binomialCoeff(n-1,i)
            px += temp.x*bez
            py += temp.y*bez
            temp = temp.next
        temp = line.head
        result.pushback(Point(px,py))
        tNew += t
    result.pushback(line.tail)
```

BezierDivideAndConquer.py

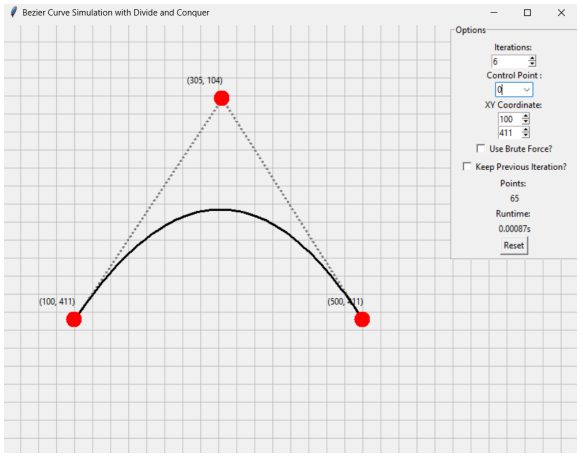
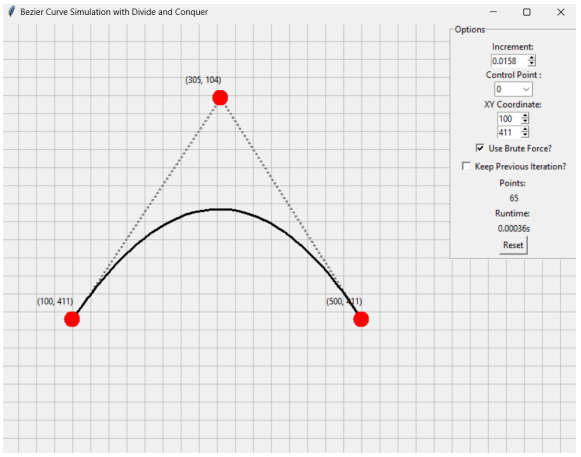
```
from PointListClass import *

def binomialCoeff(n,k):
    if k==0:
        return 1
    if n==k:
        return 1
    else:
        return binomialCoeff(n-1,k-1)+binomialCoeff(n-1,k)


def BezierBruteForce(t:float,result:Line,line:Line):
    temp = line.head
    n = line.length
    tNew = 0;
    while tNew<=1:
        px = 0
        py = 0
        for i in range(n):
            bez = ((1-tNew)**(n-i-1))*(tNew**i)*binomialCoeff(n-1,i)
            px += temp.x*bez
            py += temp.y*bez
            temp = temp.next
        temp = line.head
        result.pushback(Point(px,py))
        tNew += t
    result.pushback(line.tail)
```

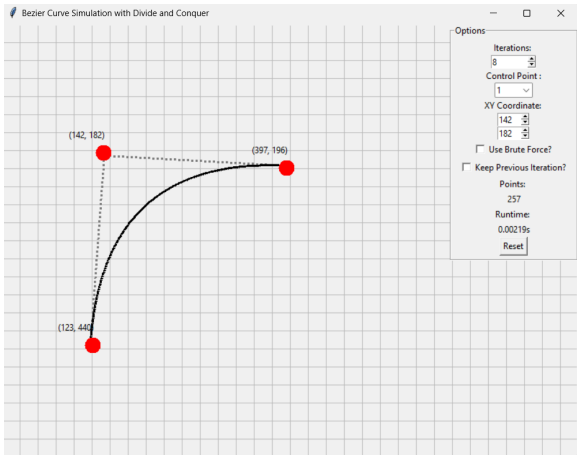
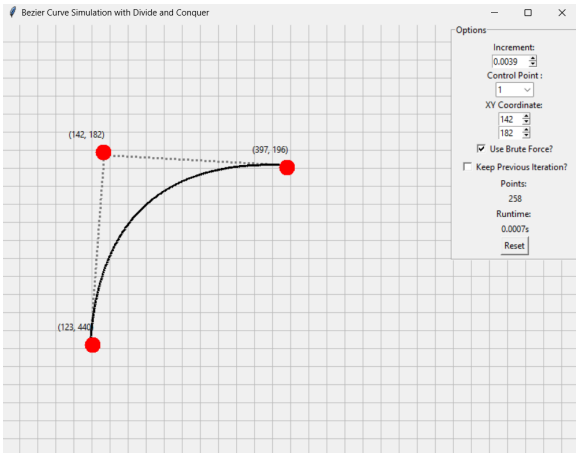
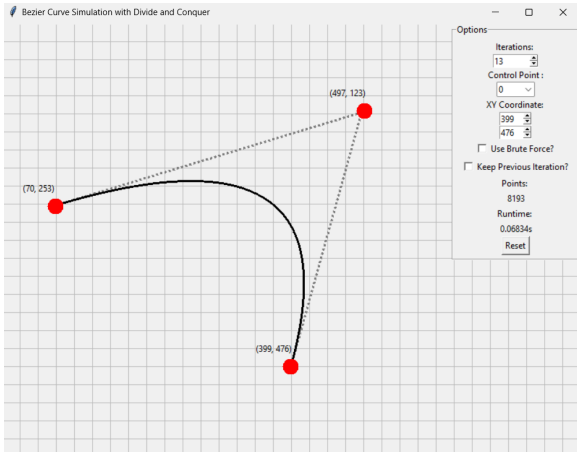
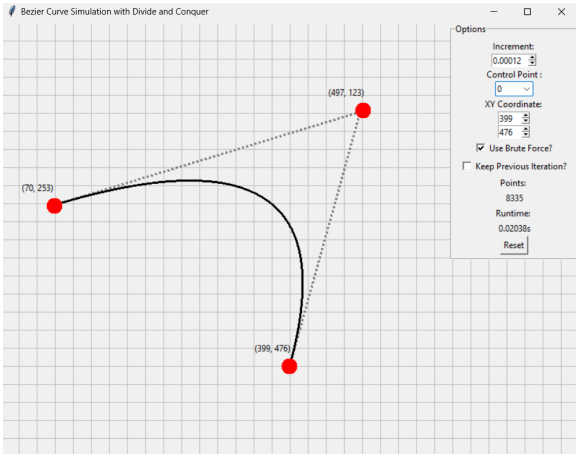
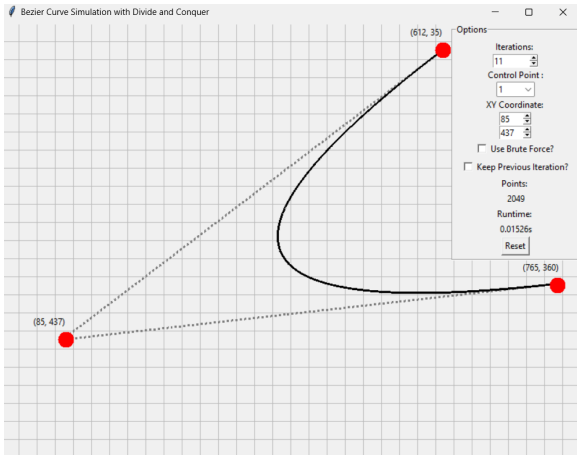
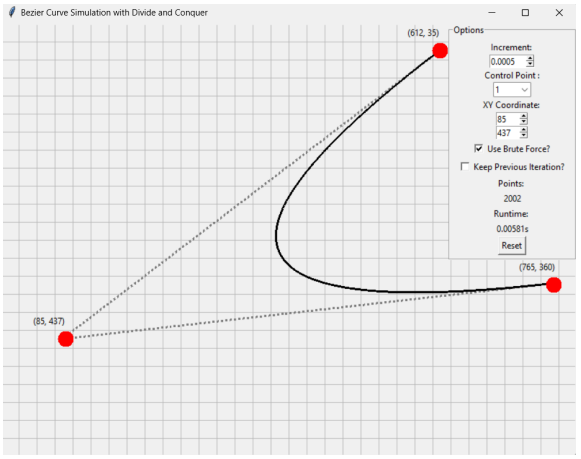
PointListClass.py

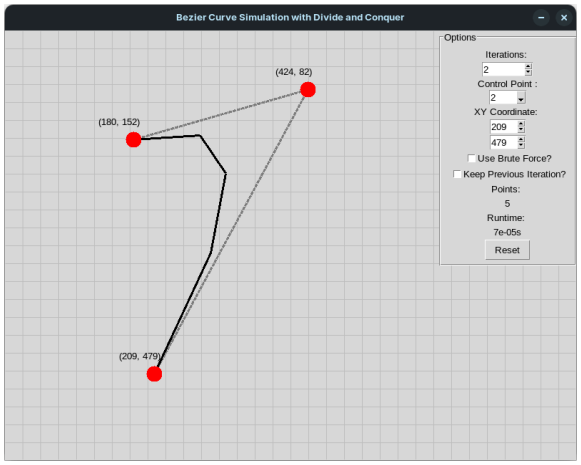
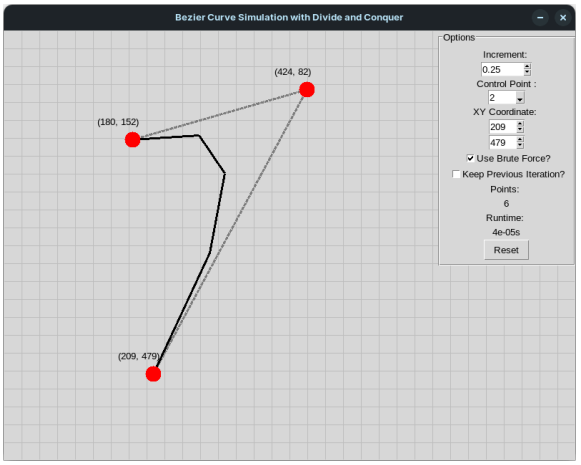
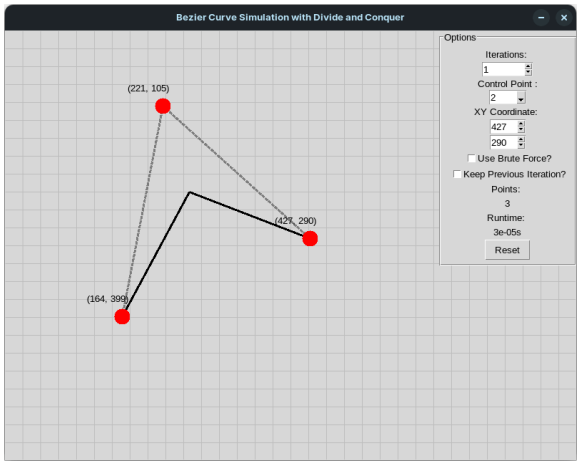
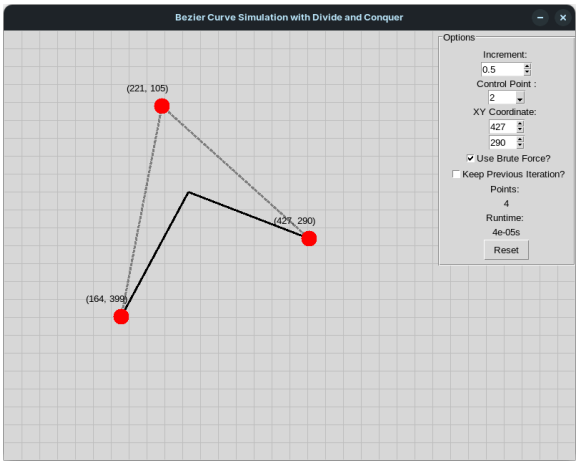
```
class Point:
    def __init__(self,x = 0,y = 0, prev = None, next = None, id = None):
        self.x = x if x is not None else 0
        self.y = y if y is not None else 0
        self.prev = prev if prev is not None else None
        self.next = next if next is not None else None
        self.id = id if id is not None else None

    def copy(self):
        copy = Point(self.x,self.y)
        copy.next = self.next
        copy.prev = self.prev
        copy.id = self.id
        return copy


class Line:
    def __init__(self):
```

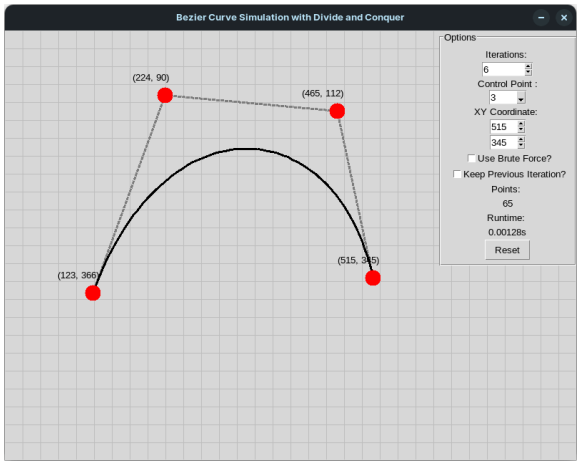
```
        self.head = None
        self.child = None
        self.tail = None
        self.length = 0

    def pushback(self, a:Point):
        p = a.copy()
        if not self.head:
            self.head = p
            self.tail = p
        else:
            p.prev = self.tail
            self.tail.next = p
            self.tail = p
        self.length += 1
        return self

    def pushfront(self, a:Point):
        p = a.copy()
        if not self.head:
            self.head = p
            self.tail = p
        else:
            p.next = self.head
            self.head.prev = p
            self.head = p
        self.length += 1
        return self
```

C. Test Case

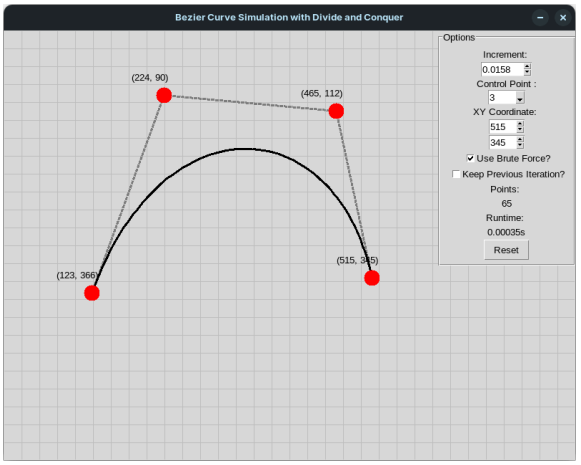
Algoritma *Divide and Conquer*	Algoritma *Brute Force*
	
Iterasi : 6 Control Point : 3 Point : 65	Increment : 0.0158 Control Point : 3 Point : 65

Runtime : 0.00087s	Runtime : 0.00036s
 Iterasi : 8 Control Point : 3 Point : 257 Runtime : 0.00219s	 Increment : 0.0039 Control Point : 3 Point : 258 Runtime : 0.0007s
 Iterasi : 13 Control Point : 3 Point : 8193 Runtime : 0.06834s	 Increment : 0.00012 Control Point : 3 Point : 8335 Runtime : 0.02038s
 Iterasi : 11	 Increment : 0.0005

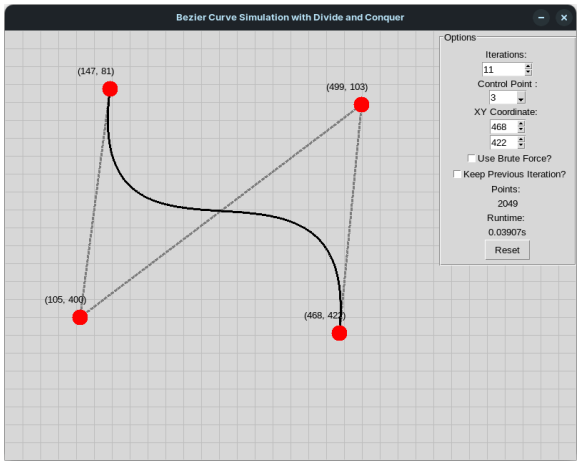
Control Point : 3 Point : 2049 Runtime : 0.01526s	Control Point : 3 Point : 2002 Runtime : 0.00581s
 Iterasi : 2 Control Point : 3 Point : 5 Runtime : 7e-05s	 Increment : 0.25 Control Point : 3 Point : 6 Runtime : 4e-05s
 Iterasi : 1 Control Point : 3 Point : 3 Runtime : 3e-05s	 Increment : 0.5 Control Point : 3 Point : 4 Runtime : 4e-05s



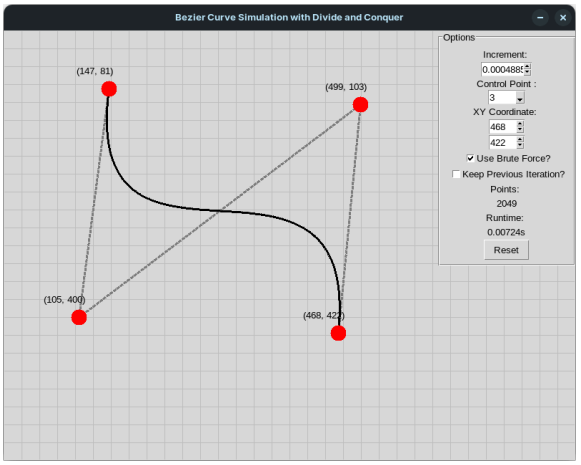
Iterasi : 6
Control Point : 4
Point : 65
Runtime : 0.00128s



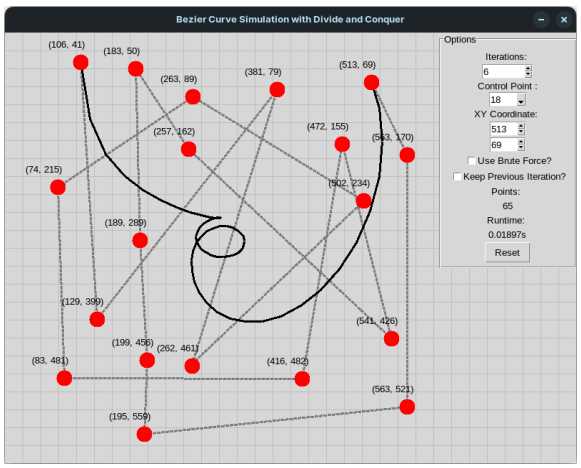
Increment : 0.0158
Control Point : 4
Point : 65
Runtime : 0.00035s



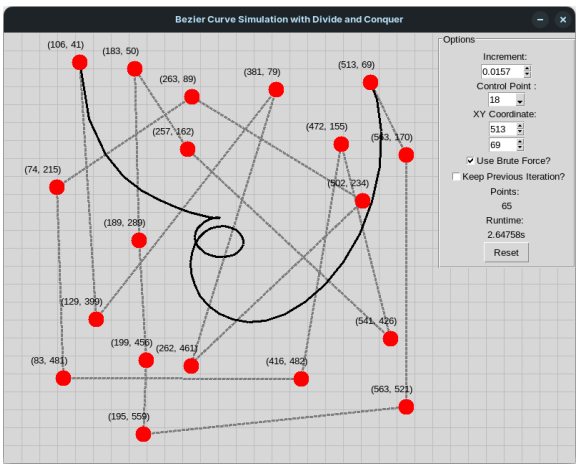
Iterasi : 11
Control Point : 4
Point : 2049
Runtime : 0.03907s



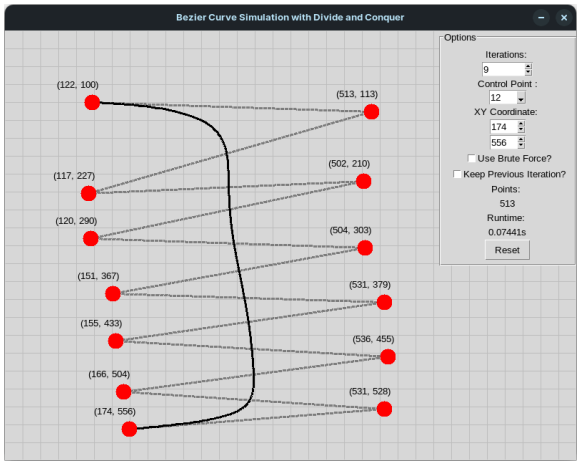
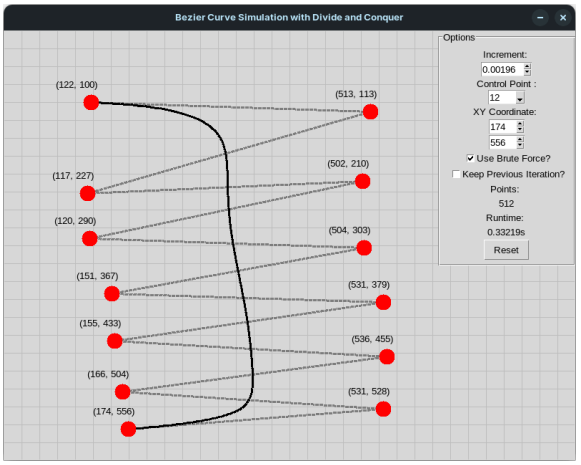
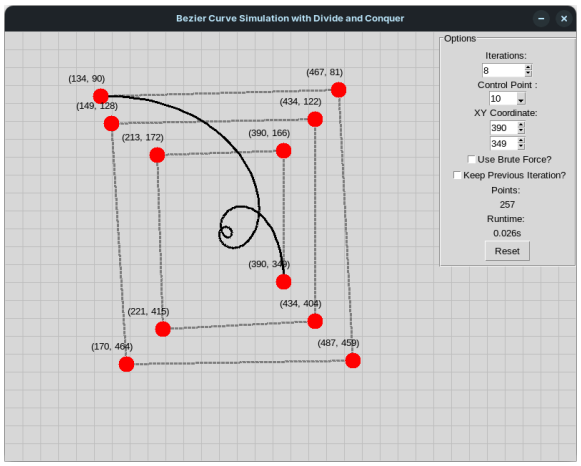
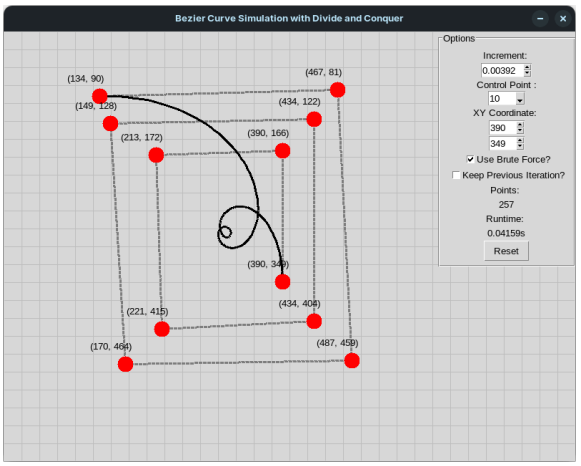
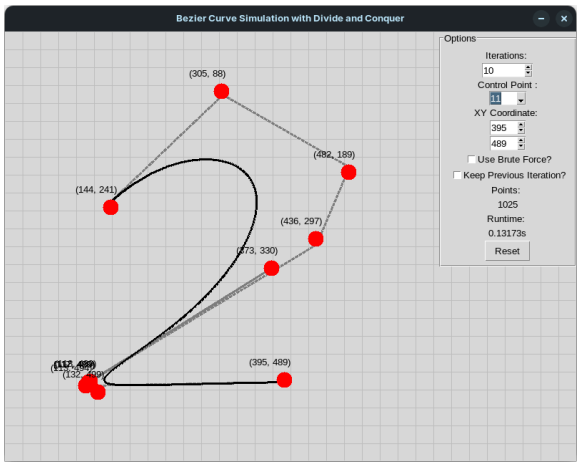
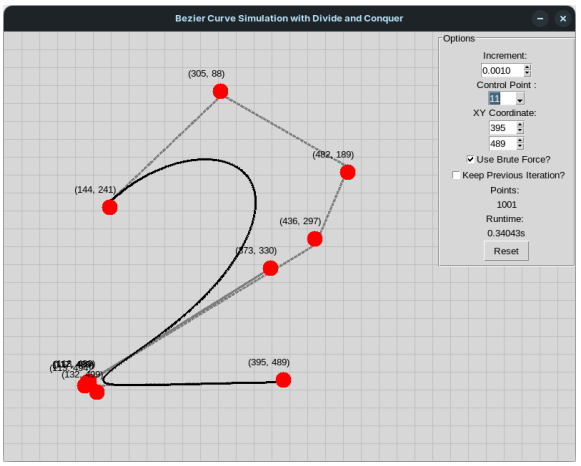
Increment : 0.0004885
Control Point : 4
Point : 2049
Runtime : 0.00724s



Iterasi : 6
Control Point : 19



Increment : 0.0157
Control Point : 19

Point : 65 Runtime : 0.01897s	Point : 65 Runtime : 2.64758s
 Iterasi : 9 Control Point : 13 Point : 513 Runtime : 0.07441s	 Increment : 0.00196 Control Point : 13 Point : 512 Runtime : 0.33219s
 Iterasi : 8 Control Point : 11 Point :257 Runtime : 0.026s	 Increment : 0.00392 Control Point : 11 Point : 257 Runtime : 0.04159s
	

Iterasi : 10 Control Point : 11 Point : 1025 Runtime : 0.13173s	Increment : 0.0010 Control Point : 11 Point : 1001 Runtime : 0.34043s

D. Link Repository: https://github.com/hannoobz/Tucil2_13522065_13522100