Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:

1. Aurelius Justin Philo Fanjaya 13522020
2. Steven Tjhia 13522103

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

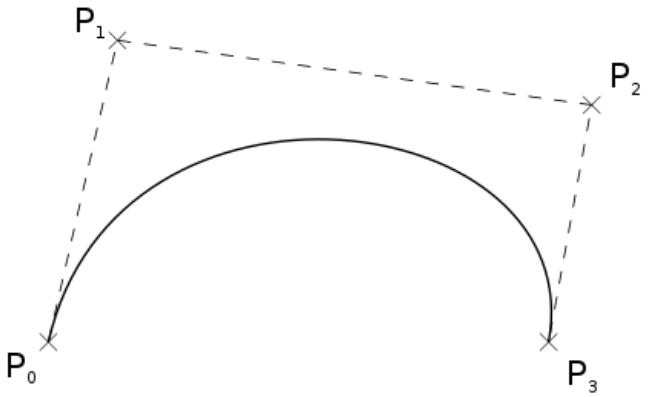
2024

Daftar Isi

Daftar Isi....2
BAB 1
Deskripsi Tugas....3
BAB 2
Algoritma Brute Force sebagai Algoritma Pembanding....6
2.1. Analisis....6
2.2. Implementasi....6
BAB 3
Algoritma Divide and Conquer....7
3.1. Analisis....7
3.2. Implementasi....7
BAB 4
Source Code....8
4.1. Algoritma....8
4.2. Visualisasi....9
BAB 5
Test Case....46
Tabel 5.1. Tabel Test Case 3 titik....46
BAB 6
Perbandingan Brute Force dan Divide and Conquer....49
BAB 7
Implementasi Bonus....50
7.1. Generalisasi Algoritma (n-titik)....50
7.2. GUI Visualisasi Bézier Curve....52
Link Repository....55

BAB 1

Deskripsi Tugas



Gambar 1. Kurva Bézier Kubik
(Sumber: https://id.wikipedia.org/wiki/Kurva_B%C3%A9zier)

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 P_0 sampai P_n , 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 P_0, sedangkan titik kontrol terakhir adalah P_3. Titik kontrol P_1 dan P_2 disebut sebagai titik kontrol antara yang tidak terletak dalam kurva yang terbentuk.

Mengulas lebih jauh mengenai bagaimana sebuah kurva Bézier bisa terbentuk, misalkan diberikan dua buah titik P_0 dan P_1 yang menjadi titik kontrol, maka kurva Bézier yang terbentuk adalah sebuah garis lurus antara dua titik. Kurva ini disebut dengan **kurva Bézier linier**. Misalkan terdapat sebuah titik Q_0 yang berada pada garis yang dibentuk oleh P_0 dan P_1, maka posisinya dapat dinyatakan dengan persamaan parametrik berikut.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

dengan t dalam fungsi kurva Bézier linier menggambarkan seberapa jauh $B(t)$ dari P_0 ke P_1. Misalnya ketika t = 0.25, maka $B(t)$ adalah seperempat jalan dari titik P_0 ke P_1. sehingga seluruh rentang variasi nilai t dari 0 hingga 1 akan membuat persamaan $B(t)$ membentuk sebuah garis lurus dari P_0 ke P_1.

Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda

dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

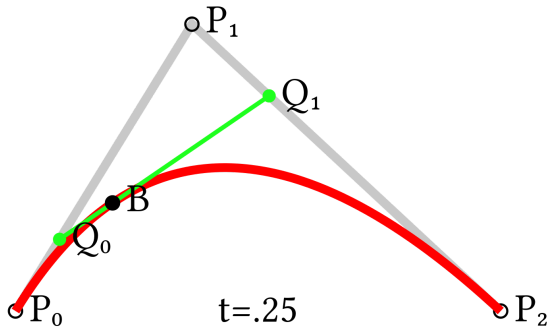
$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

dengan melakukan substitusi nilai Q_0 dan Q_1, maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1-t)^2P_0 + (1-t)tP_1 \; + \; t^2P_2, \qquad t \in [0,1]$$

Berikut adalah ilustrasi dari kasus diatas.



Gambar 2. Pembentukan Kurva Bézier Kuadratik.
(Sumber: https://simonhalliday.com/2017/02/15/quadratic-bezier-curve-demo/)

Proses ini dapat juga diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan **kurva Bézier kubik**, lima titik akan menghasilkan **kurva Bézier kuartik**, dan seterusnya. Berikut adalah persamaan kurva Bézier kubik dan kuartik dengan menggunakan prosedur yang sama dengan yang sebelumnya.

$$S_0 = B(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + \; t^3P_3, \qquad t \in [0,1]$$

$$T_0 = B(t) = (1-t)^4P_0 + 4(1-t)^3tP_1 + 6(1-t)^2t^2P_2 + 4(1-t)t^3P_3 + \; t^4P_4, \qquad t \in [0,1]$$

Tentu saja persamaan yang terbentuk sangat panjang dan akan semakin rumit seiring bertambahnya titik. Oleh sebab itu, dalam rangka melakukan efisiensi pembuatan kurva Bézier yang sangat berguna ini, maka Anda diminta untuk mengimplementasikan **pembuatan kurva Bézier** dengan algoritma titik tengah berbasis ***divide and conquer*.**

Ilustrasi kasus

Idenya cukup sederhana, relatif mirip dengan pembahasan sebelumnya, dan dilakukan secara iteratif. Misalkan terdapat tiga buah titik, P_0, P_1, dan P_2, dengan titik P_1 menjadi titik kontrol antara, maka:

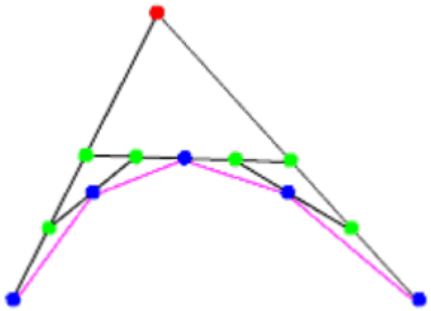
a. Buatlah sebuah titik baru Q_0 yang berada di tengah garis yang menghubungkan P_0 dan P_1, serta titik Q_1 yang berada di tengah garis yang menghubunhkan P_1 dan P_2.

b. Hubungkan Q_0 dan Q_1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru R_0 yang berada di tengah Q_0 dan Q_1.
d. Buatlah sebuah garis yang menghubungkan P_0 - R_0 - P_2.

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah "kurva" yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedurnya.

a. Buatlah beberapa titik baru, yaitu S_0 yang berada di tengah P_0 dan Q_0, S_1 yang berada di tengah Q_0 dan R_0, S_2 yang berada di tengah R_0 dan Q_1, dan S_3 yang berada di tengah Q_1 dan P_2.
b. Hubungkan S_0 dengan S_1 dan S_2 dengan S_3 sehingga terbentuk garis baru.
c. Buatlah dua buah titik baru, yaitu T_0 yang berada di tengah S_0 dan S_1, serta T_1 yang berada di tengah S_2 dan S_3.
d. Buatlah sebuah garis yang menghubungkan P_0 - T_0 - R_0 - T_1 - P_2.

Melalui iterasi kedua akan tampak semakin mendekati sebuah kurva, dengan aproksimasi 5 buah titik. Anda dapat membuat visualisasi atau gambaran secara mandiri terkait hal ini sehingga dapat diamati dan diterka dengan jelas bahwa semakin banyak iterasi yang dilakukan, maka akan membentuk sebuah kurva yang tidak lain adalah kurva Bézier.



Gambar 3. Hasil pembentukan Kurva Bézier Kuadratik dengan *divide and conquer* setelah iterasi ke-2.

BAB 2

Algoritma Brute Force sebagai Algoritma Pembanding

2.1. Analisis

Algoritma Brute Force Kurva Bezier Kuadratik yang kami implementasikan menggunakan persamaan sebagai berikut.

$$B(t) = (1-t)^2 P_0 + (1-t)tP_1 + t^2 P_2$$

dengan $t \in [0, 1]$ dan P_0, P_1, P_2 adalah titik kontrol.

Cara kami menentukan t yang dijadikan sampel atau titik uji adalah dengan menggunakan jumlah iterasi yang menjadi input. Misalkan jumlah iterasi yang menjadi input adalah n, maka akan dihitung 2^n buah titik dari persamaan tersebut dengan selisih tiap titik sebesar $\frac{1}{2^n}$. Dengan demikian, akan didapatkan 2^n+1 buah titik dengan titik pertama adalah P_0 dan titik terakhir adalah P_2.

2.2. Implementasi

Implementasi Algoritma Brute Force kami buat dengan melakukan langkah-langkah sebagai berikut.

1. Menghitung nilai 2^n, dengan n adalah jumlah iterasi.
2. Menghitung selisih nilai t dari tiap titik dengan rumus $\Delta t = \frac{1}{2^n}$.
3. Menginisialisasi nilai t = 0.
4. Melakukan *looping* terhadap nilai t dengan memasukkannya ke rumus

   $$B(t) = (1-t)^2 P_0 + (1-t)tP_1 + t^2 P_2$$

   dan menambahkan nilai t dengan Δt pada akhir setiap iterasi *loop* hingga nilai t = 1.

BAB 3

Algoritma Divide and Conquer

3.1. Analisis

Algoritma Divide and Conquer Kurva Bezier Kuadratik yang kami implementasikan terbagi menjadi 2 bagian yaitu bagian *Divide* dan bagian *Conquer*. Pada bagian *Conquer*, dicari titik tengah antara titik kontrol pertama (p0) dengan kedua (p1) sebagai q0, dan titik kontrol kedua (p1) dengan ketiga (p2) sebagai q1. Titik tengah dari q0 dan q1 tersebut (r0) merupakan salah satu solusi dari iterasi tersebut. Kemudian, pada bagian *Divide*, persoalan akan dibagi untuk iterasi selanjutnya. Akan dilakukan kembali rekursi algoritma ini dengan (p0, q0, r0) sebagai titik kontrol dan (r0, q1, p2) sebagai titik kontrol lainnya. Kemudian, solusi dari kedua rekursi tersebut akan di konkatenasi dengan r0.

3.2. Implementasi

Implementasi Algoritma Divide and Conquer untuk Kurva Bezier Kuadratik (3 titik kontrol) kami buat dengan melakukan langkah-langkah sebagai berikut.

1. Cari titik tengah antara titik kontrol pertama (p0) dan titik kontrol kedua (p1) sebagai q0.
2. Cari titik tengah antara titik kontrol kedua (p1) dan titik kontrol ketiga (p2) sebagai q1.
3. Cari titik tengah antara titik q0 dan q1 sebagai r0.
4. Lakukan rekursi dengan titik kontrol (p0, q0, r0) sebagai s0.
5. Lakukan rekursi dengan titik kontrol (r0, q1, p2) sebagai s1.
6. Jika iterasi merupakan iterasi pertama, *return* konkatenasi p0 + s0 + r0 + s1 + p2.
7. Jika iterasi bukan iterasi pertama, return s0 + r0 + s1.

BAB 4

Source Code

4.1. Algoritma

```
def getMidPoint(p1,p2):
  x = float((p1[0]+p2[0])/2 )
  y = float((p1[1]+p2[1])/2)
  pout = (x,y)
  return pout

def BezierN(points, iterasi, iterasiMax):
  if (iterasi >= iterasiMax):
     return []
  else:
     q = points
     left = [q[0]]
     right = [q[-1]]
     while len(q) > 1:
        temp = q
        q = [getMidPoint(temp[i], temp[i+1]) for i in range(len(temp)-1)]
        left.append(q[0])
        right.append(q[-1])
     right.reverse()
     if iterasi == 0:
        iterasi += 1
        return [points[0]] + BezierN(left, iterasi, iterasiMax) + [right[0]] + BezierN(right, iterasi, iterasiMax) + [points[-1]]
     else:
        iterasi += 1
        return BezierN(left, iterasi, iterasiMax) + [right[0]] + BezierN(right, iterasi, iterasiMax)

def BezierBruteForce(p0, p1, p2, iterasi):
  n = pow(2, iterasi)
  delta = 1 / n
  t = 0
  b = []
  for i in range(n):
     p = [(pow(1-t, 2)*p0[i] + 2*(1-t)*t*p1[i] + pow(t, 2) * p2[i]) for i in range(2)]
     b.append((p[0], p[1]))
     t += delta
```

```
    p = [pow(t, 2) * p2[i] for i in range(2)]
    b.append((p[0], p[1]))
    return b
```

4.2. Visualisasi

```
import flet as ft
import flet.canvas as cv
import math
import time

def main(page: ft.Page):
    page.title = "Bézier Curve"
    app_h = 700
    app_w = 1300
    page.window_height = app_h
    page.window_width = app_w
    page.window_resizable = False

    # Fonts
    page.fonts = {
        "Railinc" : "/fonts/Raillinc.otf",
    }

    # Initiate Variable
    num_of_iteration = [0]

    process_time = [0]

    range_x = [0]
    range_y = [0]

    canvas_path = []
    paint_canvas = ft.Paint(
        style=ft.PaintingStyle.STROKE,
        stroke_width=0.5/(num_of_iteration[0]+1),
        color="#6B728E"
    )

    result_path = []
```

```
paint_result_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=1/(num_of_iteration[0]+1),
    color="#000000"
)

initial_circle = []
paint_initial_circle = ft.Paint(
    style=ft.PaintingStyle.FILL,
    stroke_width=1,
    color="#6B728E"
)

result_circle = []
paint_result_circle = ft.Paint(
    style=ft.PaintingStyle.FILL,
    stroke_width=1,
    color="#000000"
)

cartesian_line_canvas = []
cartesian_line_result_canvas = []
paint_cartesian_line_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=0.3,
    color="#45474B"
)

cartesian_text_number_canvas = []
cartesian_text_number_result_canvas = []
paint_cartesian_text_number_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=1,
    color="#45474B"
)

initial_canvas_axis = []
initial_canvas_ordinat = []
```

```
coordinate_points = []
initial_points = []
visualization_speed = [0]

listview_controls = [
    ft.Container(
        margin=ft.margin.only(top=10),
        content=ft.Text("Result Point",text_align="CENTER",size=12, font_family="Railinc")
    ),
    ft.Divider(color="#000000")
]

# Implementation
def resetPath():
    makeListEmpty(canvas_path)
    makeListEmpty(result_path)
    makeListEmpty(result_circle)
    process_time[0] = 0
    makeListEmpty(cartesian_line_result_canvas)
    makeListEmpty(cartesian_text_number_result_canvas)
    makeListEmpty(listview_controls)
    listview_controls.append(
        ft.Container(
            margin=ft.margin.only(top=10),
            content=ft.Text("Result Point",text_align="CENTER",size=15, font_family="Railinc")
        )
    )
    listview_controls.append(
        ft.Divider(color="#000000")
    )

def resetInitial():
    makeListEmpty(initial_points)
    makeListEmpty(coordinate_points)
    makeListEmpty(initial_circle)
    makeListEmpty(cartesian_line_canvas)
    makeListEmpty(cartesian_text_number_canvas)
    makeListEmpty(initial_canvas_axis)
    makeListEmpty(initial_canvas_ordinat)
```

```
    process_time[0] = 0
    range_x[0] = 0
    range_y[0] = 0

def makeListEmpty(List):
    List.clear()

def insertToPath(ps1):
    if(ps1[1] == "m"):
        canvas_path.append(cv.Path.MoveTo(ps1[0][0],ps1[0][1]))
    else:
        canvas_path.append(cv.Path.LineTo(ps1[0][0],ps1[0][1]))
    page.update()
    time.sleep(1-visualization_speed[0])

def insertToPathResult(ps1):
    if(ps1[1] == "m"):
        result_path.append(cv.Path.MoveTo(ps1[0][0],ps1[0][1]))
    else:
        result_path.append(cv.Path.LineTo(ps1[0][0],ps1[0][1]))
    page.update()
    time.sleep(1-visualization_speed[0])

def insertToPathCircle(p):
    s = 5
    initial_circle.append(cv.Path.Oval(p[0]-(0.5*s),p[1]-(0.5*s),s,s))
    page.update()

def insertToPathResultCircle(p):
    s = 5
    result_circle.append(cv.Path.Oval(p[0]-(0.5*s),p[1]-(0.5*s),s,s))
    page.update()

def insertToPathCartesianLineCanvas(ps):
    if(ps[1] == "m"):
        cartesian_line_canvas.append(cv.Path.MoveTo(ps[0][0],ps[0][1]))
    else:
        cartesian_line_canvas.append(cv.Path.LineTo(ps[0][0],ps[0][1]))
    page.update()
```

```
  def insertToPathCartesianLineResultCanvas(ps):
      if(ps[1] == "m"):
          cartesian_line_result_canvas.append(cv.Path.MoveTo(ps[0][0],ps[0][1]))
      else:
          cartesian_line_result_canvas.append(cv.Path.LineTo(ps[0][0],ps[0][1]))
      page.update()

  def insertToPathCartesianTextNumberCanvas(ps):
      cartesian_text_number_canvas.append(
          cv.Text(
              ps[0][0]-5,
              ps[0][1]-5,
              ps[1],
              ft.TextStyle(size=ps[2]),
          )
      )
      page.update()

  def insertToPathCartesianTextNumberResultCanvas(ps):
      t = cv.Text(
              ps[0][0]-5,
              ps[0][1]-5,
              ps[1],
              ft.TextStyle(size=ps[2]),
          )
      if(t not in cartesian_text_number_result_canvas and t not in cartesian_text_number_canvas):
          cartesian_text_number_result_canvas.append(t)
      page.update()

  def insertToListView(p):
      if(str(p[0]) == "process time"):
          listview_controls.append(
              ft.Container(
                  margin=ft.margin.only(left=15),
                  content=ft.Column(
                      controls=[
                          ft.Text("Process time (Netto): " + str(p[1]*1000) + " ms",size=12,color="#000000",
weight=800,font_family="Railinc"),
```

```
                    ft.Text(" ",size=5,color="#000000"),
                ]
            )
        )
    )
else:
    listview_controls.append(
        ft.Container(
            margin=ft.margin.only(left=15),
            content=ft.Column(
                controls=[
                    ft.Text(str(len(listview_controls)-1)+ ". " + str(p),size=12,color="#000000"),
                    ft.Text(" ",size=5,color="#000000"),
                ]
            )
        )
    )
page.update()

def initCoordinate_to_canvasCoordinate(p):
    min_y = min([point[1] for point in initial_points])
    min_x = min([point[0] for point in initial_points])
    if(range_y[0] > range_x[0]):
        coordinate_multiplier = float((app_h*0.85)/range_y[0])
        y = (app_h*0.85) - ((p[1]-min_y)*coordinate_multiplier)
        x = p[0]*coordinate_multiplier +
(float(((app_h*0.85)-(range_x[0]*coordinate_multiplier))/2)-(min_x*coordinate_multiplier))
    else:
        coordinate_multiplier = float((app_h*0.85)/range_x[0])
        x = (p[0]-min_x)*coordinate_multiplier
        y = (app_h*0.85)-(p[1]*coordinate_multiplier +
(float(((app_h*0.85)-(range_y[0]*coordinate_multiplier))/2)-(min_y*coordinate_multiplier)))
    return (x,y)

def getMidPoint(p1,p2):
    x = float((p1[0]+p2[0])/2 )
    y = float((p1[1]+p2[1])/2)
    pout = (x,y)
    return pout
```

```
def BezierN_process_time(points, iterasi, iterasiMax):
    if (iterasi >= iterasiMax):
        return []
    else:
        q = points
        left = [q[0]]
        right = [q[-1]]
        while len(q) > 1:
            temp = q
            q = [getMidPoint(temp[i], temp[i+1]) for i in range(len(temp)-1)]

            left.append(q[0])
            right.append(q[-1])
        right.reverse()
        if iterasi == 0:
            iterasi += 1
            output = [points[0]] + BezierN_process_time(left, iterasi, iterasiMax)
            return  output + [left[-1]] + BezierN_process_time(right, iterasi, iterasiMax) + [points[-1]]
        else:
            iterasi += 1
            output = BezierN_process_time(left, iterasi, iterasiMax)
            return output + [left[-1]] + BezierN_process_time(right, iterasi, iterasiMax)

def BezierN(points, iterasi, iterasiMax):
    if(iterasi == 0):
        insertToPathResultCircle(initCoordinate_to_canvasCoordinate(points[0]))
        insertToListView(points[0])
    offset=5
    if (iterasi >= iterasiMax):
        return []
    else:
        q = points
        left = [q[0]]
        right = [q[-1]]
        while len(q) > 1:
            temp = q
            q = [getMidPoint(temp[i], temp[i+1]) for i in range(len(temp)-1)]
            left.append(q[0])
```

```
            right.append(q[-1])
            for i in range(len(temp)):
                if(i == 0):
                    insertToPath((initCoordinate_to_canvasCoordinate(temp[0]),"m"))
                else:
                    insertToPath((initCoordinate_to_canvasCoordinate(temp[i]),"l"))
        right.reverse()
        canvas_point = initCoordinate_to_canvasCoordinate(q[0])
        if(iterasi == iterasiMax-1):

            insertToPathResultCircle(canvas_point)

            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(q[0][0
],2)),8))

            insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
            insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))

insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(q[0][1],2)),8))

            insertToPathResult((initCoordinate_to_canvasCoordinate(points[0]),"m"))
            insertToPathResult((canvas_point,"l"))
            insertToPathResult((initCoordinate_to_canvasCoordinate(points[-1]),"l"))
        elif(iterasi < iterasiMax-1):
            insertToPath((initCoordinate_to_canvasCoordinate(points[0]),"m"))
            insertToPath((canvas_point,"l"))
            insertToPath((initCoordinate_to_canvasCoordinate(points[-1]),"l"))

        if iterasi == 0:
            iterasi += 1

            output = [points[0]]

            output += BezierN(left, iterasi, iterasiMax)

            insertToListView(left[-1])
```

```
            canvas_point = initCoordinate_to_canvasCoordinate(left[-1])
            insertToPathResultCircle(canvas_point)
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(left[-1
][0],2)),8))

            insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
            insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))

insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(left[-1][1],2)),8))

            output += [left[-1]]

            output += BezierN(right, iterasi, iterasiMax)

            output += [points[-1]]

            insertToPathResultCircle(initCoordinate_to_canvasCoordinate(points[-1]))
            insertToListView(points[-1])
            return output

        else:
            iterasi += 1
            output = BezierN(left, iterasi, iterasiMax)

            insertToListView(left[-1])

            canvas_point = initCoordinate_to_canvasCoordinate(left[-1])
            insertToPathResultCircle(canvas_point)
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(left[-1
][0],2)),8))

            insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
```

```
            insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))

insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(left[-1][1],2)),8))

            output += [left[-1]]

            output += BezierN(right, iterasi, iterasiMax)

            return output

    def buttonVisualizeClicked(e):
        resetPath()
        e.control.visible = False
        BezierN(initial_points,0,num_of_iteration[0])
        startT = time.time()
        BezierN_process_time(initial_points,0,num_of_iteration[0])
        endT = time.time()
        process_time[0] = endT-startT
        insertToListView(("process time",endT-startT))
        e.control.visible = True
        page.update()

    buttonVisualize = ft.ElevatedButton(
        style=ft.ButtonStyle(
            color={
                ft.MaterialState.DEFAULT: "#FFFFFF",
                ft.MaterialState.HOVERED: "#000000",
                ft.MaterialState.DISABLED: "#000000",
            },
            bgcolor={ft.MaterialState.DEFAULT: "#000000",
                    ft.MaterialState.HOVERED: "#FFFFFF",
                    ft.MaterialState.DISABLED: "#FFFFFF"
            },
            padding={ft.MaterialState.DEFAULT: 28},
            overlay_color=ft.colors.TRANSPARENT,
            elevation={"pressed": 0, "": 1},
            side={
                ft.MaterialState.DEFAULT: ft.BorderSide(1, "#000000"),
            },
```

```
        ),
    width=0.15*app_w,
    on_click=buttonVisualizeClicked,
    content=ft.Text("Visualize!",size=15,font_family="Railinc")
)

def inputChanged(e):
    resetInitial()
    resetPath()
    temp_range_y = 0
    max_y = 0
    min_y = 0
    temp_range_x = 0
    max_x = 0
    min_x = 0
    coordinate_multiplier = 1
    isValid = True
    points_temp = e.control.value.split("_")
    n_points = len(points_temp)
    if(n_points < 3):
        e.control.error_text = "A curve consist of minimum 3 control points"
    else:
        for i in range(n_points):
            point = points_temp[i].split(",")
            if(len(point) == 2):
                try:
                    x = int(point[0])
                    y = int(point[1])
                except:
                    try:
                        x = float(point[0])
                        y = float(point[1])
                    except:
                        e.control.error_text = "x and y must be integer"
                        resetInitial()
                        isValid = False
                        break
                    else:
                        if(i==0):
```

```
                    max_y = y
                    min_y = y
                    max_x = x
                    min_x = x
                else:
                    if(y > max_y):
                        max_y = y
                    elif(y < min_y):
                        min_y = y
                    if(x > max_x):
                        max_x = x
                    elif(x < min_x):
                        min_x = x
                e.control.error_text = ""
                initial_points.append((x,y))
            else:
                if(i==0):
                    max_y = y
                    min_y = y
                    max_x = x
                    min_x = x
                else:
                    if(y > max_y):
                        max_y = y
                    elif(y < min_y):
                        min_y = y
                    if(x > max_x):
                        max_x = x
                    elif(x < min_x):
                        min_x = x
                e.control.error_text = ""
                initial_points.append((x,y))
        else:
            e.control.error_text = "Invalid format"
            resetInitial()
            isValid = False
            break

    if isValid:
```

```
            temp_range_y = max_y-min_y
            temp_range_x = max_x-min_x
            if(temp_range_y > temp_range_x):
                coordinate_multiplier = float((app_h*0.85)/temp_range_y)
                for point in initial_points:
                    y_coor = (app_h*0.85) - ((point[1]-min_y)*coordinate_multiplier)
                    x_coor = point[0]*coordinate_multiplier +
(float(((app_h*0.85)-(temp_range_x*coordinate_multiplier))/2)-(min_x*coordinate_multiplier))
                    coordinate_points.append((x_coor,y_coor))
            else:
                coordinate_multiplier = float((app_h*0.85)/temp_range_x)
                for point in initial_points:
                    x_coor = (point[0]-min_x)*coordinate_multiplier
                    y_coor = (app_h*0.85)-(point[1]*coordinate_multiplier +
(float(((app_h*0.85)-(temp_range_y*coordinate_multiplier))/2)-(min_y*coordinate_multiplier)))
                    coordinate_points.append((x_coor,y_coor))
            range_y[0] = temp_range_y
            range_x[0] = temp_range_x
            # Draw Axis and Ordinat line
            insertToPathCartesianLineCanvas(((0,0),"m"))
            insertToPathCartesianLineCanvas(((0,0+app_h*0.85),"l"))
            insertToPathCartesianLineCanvas(((0+app_h*0.85,0+app_h*0.85),"l"))
            # Draw Circle (Dot)
            for i in range(len(coordinate_points)):
                point = coordinate_points[i]
                insertToPathCircle(point)
                offset = 5
                if(point[0] not in initial_canvas_axis):
                    insertToPathCartesianLineCanvas(((point[0],0+app_h*0.85-offset),"m"))
                    insertToPathCartesianLineCanvas(((point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberCanvas(((point[0],0+app_h*0.85+offset+15),str(round(initial_points[i][0],2)
),8))
                    initial_canvas_axis.append(point[0])
                if(point[1] not in initial_canvas_ordinat):
                    insertToPathCartesianLineCanvas(((0-offset,point[1]),"m"))
                    insertToPathCartesianLineCanvas(((0+offset,point[1]),"l"))

insertToPathCartesianTextNumberCanvas(((0-offset-15,point[1]),str(round(initial_points[i][1],2)),8))
```

```
                initial_canvas_ordinat.append(point[1])
    page.update()

def iterationChanged(e):
    try:
        num_of_iteration[0] = int(e.control.value)
    except:
        num_of_iteration[0] = 0
        e.control.error_text = "Value must be integer"
    else:
        e.control.error_text = ""
    page.update()


def slider_change(e):
    visualization_speed[0] = e.control.value
    page.update()

container = ft.Container(
    height=app_h*0.92,
    width=app_w,
    bgcolor="#C7C8CC",
    content=ft.Column(
        controls=[
            ft.Row(
                alignment="center",
                vertical_alignment="center",
                controls=[
                    ft.Container(
                        content=ft.Column(
                            height=app_h-49,
                            alignment="center",
                            controls=[
                                ft.Container(
                                    height=app_h*0.85,
                                    width=app_h*0.85,
                                    alignment=ft.alignment.center,
                                    margin=ft.margin.only(left=20,bottom=20),
                                    content=ft.Column(
```

```
                controls=[
                    ft.Stack(
                        controls=[
                            cv.Canvas(
                                [
                                    cv.Path(
                                        cartesian_line_canvas,
                                        paint=paint_cartesian_line_canvas
                                    )
                                ],
                                width=float("inf"),
                                expand=True,
                            ),
                            cv.Canvas(
                                [
                                    cv.Path(
                                        cartesian_line_result_canvas,
                                        paint=paint_cartesian_line_canvas
                                    )
                                ],
                                width=float("inf"),
                                expand=True,
                            ),
                            cv.Canvas(
                                [
                                    cv.Path(
                                        canvas_path,
                                        paint=paint_canvas
                                    )
                                ],
                                width=float("inf"),
                                expand=True,
                            ),
                            cv.Canvas(
                                [
                                    cv.Path(
                                        result_path,
                                        paint=paint_result_canvas
                                    )
```

```
                        ],
                        width=float("inf"),
                        expand=True,
                    ),
                    cv.Canvas(
                        [
                            cv.Path(
                                initial_circle,
                                paint=paint_initial_circle,
                            )
                        ],
                        width=float("inf"),
                        expand=True,
                    ),
                    cv.Canvas(
                        [
                            cv.Path(
                                result_circle,
                                paint=paint_result_circle,
                            )
                        ],
                        width=float("inf"),
                        expand=True,
                    ),
                    cv.Canvas(
                        cartesian_text_number_canvas
                    ),
                    cv.Canvas(
                        cartesian_text_number_result_canvas
                    ),
                ]
            )
        ]
    )
),
ft.Container(
```

```
                height=app_h*0.8545,
                width=app_h*0.855,
                # bgcolor="#000000",
                content=ft.Column(
                    horizontal_alignment="center",
                    alignment="center",
                    controls=[
                        # TITLE
                        ft.Row(
                            alignment="center",
                            controls=[
                                ft.Text("Bézier Curve",size=40,font_family="Railinc")
                            ]
                        ),
                        # INPUT
                        ft.Container(
                            margin=ft.margin.only(top=0.03*app_h),
                            alignment=ft.alignment.center,
                            width=app_w*0.45,
                            content=ft.Row(
                                alignment="center",
                                controls=[
                                    ft.Container(
                                        width=app_w*0.18,
                                        content=ft.TextField(
                                            label="Insert Points",
                                            hint_text="x0,y0_x1_y1_x2,y2_...",
                                            on_change=inputChanged,
                                        )
                                    ),
                                    ft.Container(
                                        width=app_w*0.1,
                                        alignment=ft.alignment.center,
                                        content=ft.TextField(
                                            label="Iteration(s)",
                                            on_change=iterationChanged,
                                        )
                                    )
                                ],
```

```
            )
        ),
        # Button "Visualize!"
        ft.Container(
            margin=ft.margin.only(top=10,bottom=10),
            content=buttonVisualize
        ),
        # SPEED SLIDER
        ft.Container(
            width=0.3 * app_w,
            content=ft.Column(
                controls=[
                    ft.Text("Visualization Speed",size=15,font_family="Railinc"),
                    ft.Slider(
                        min=0,
                        max =1,
                        divisions=100,
                        active_color= "#000000",
                        on_change=slider_change
                    ),
                    ft.Row(
                        alignment=ft.MainAxisAlignment.SPACE_BETWEEN,
                        controls=[
                            ft.Container(
                                content=ft.Text("Slow",size=10,font_family="Railinc")
                            ),
                            ft.Container(
                                content=ft.Text("Fast",size=10,font_family="Railinc")
                            )
                        ]
                    )
                ]
            )
        ),
        # ListView
        ft.Container(
            width=0.25*app_w,
            height=app_h*0.3,
            border=ft.border.all(2, "#000000"),
```

```
                            border_radius=15,
                            content=ft.ListView(
                                auto_scroll=True,
                                controls=listview_controls,
                            )
                        )

                    ]
                )
            ),
        ]
    ),
  ]
 )
 )

  page.add(container)
  page.update()

ft.app(target=main)




import flet as ft
import flet.canvas as cv
import math
import time

def main(page: ft.Page):
  page.title = "Bézier Curve"
  app_h = 700
  app_w = 1300
  page.window_height = app_h
  page.window_width = app_w
  page.window_resizable = False

  # Fonts
  page.fonts = {
```

```
    "Railinc" : "/fonts/Raillinc.otf",
}

# Initiate Variable
num_of_iteration = [0]

process_time = [0]

range_x = [0]
range_y = [0]

canvas_path = []
paint_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=0.5/(num_of_iteration[0]+1),
    color="#6B728E"
)

result_path = []
paint_result_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=1/(num_of_iteration[0]+1),
    color="#000000"
)

initial_circle = []
paint_initial_circle = ft.Paint(
    style=ft.PaintingStyle.FILL,
    stroke_width=1,
    color="#6B728E"
)

result_circle = []
paint_result_circle = ft.Paint(
    style=ft.PaintingStyle.FILL,
    stroke_width=1,
    color="#000000"
)
```

```
cartesian_line_canvas = []
cartesian_line_result_canvas = []
paint_cartesian_line_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=0.3,
    color="#45474B"
)

cartesian_text_number_canvas = []
cartesian_text_number_result_canvas = []
paint_cartesian_text_number_canvas = ft.Paint(
    style=ft.PaintingStyle.STROKE,
    stroke_width=1,
    color="#45474B"
)

initial_canvas_axis = []
initial_canvas_ordinat = []

coordinate_points = []
initial_points = []
visualization_speed = [0]

listview_controls = [
    ft.Container(
        margin=ft.margin.only(top=10),
        content=ft.Text("Result Point",text_align="CENTER",size=12, font_family="Railinc")
    ),
    ft.Divider(color="#000000")
]

# Implementation
def resetPath():
    makeListEmpty(canvas_path)
    makeListEmpty(result_path)
    makeListEmpty(result_circle)
    process_time[0] = 0
    makeListEmpty(cartesian_line_result_canvas)
    makeListEmpty(cartesian_text_number_result_canvas)
```

```
    makeListEmpty(listview_controls)
    listview_controls.append(
        ft.Container(
            margin=ft.margin.only(top=10),
            content=ft.Text("Result Point",text_align="CENTER",size=15, font_family="Railinc")
        )
    )
    listview_controls.append(
        ft.Divider(color="#000000")
    )

def resetInitial():
    makeListEmpty(initial_points)
    makeListEmpty(coordinate_points)
    makeListEmpty(initial_circle)
    makeListEmpty(cartesian_line_canvas)
    makeListEmpty(cartesian_text_number_canvas)
    makeListEmpty(initial_canvas_axis)
    makeListEmpty(initial_canvas_ordinat)
    process_time[0] = 0
    range_x[0] = 0
    range_y[0] = 0

def makeListEmpty(List):
    List.clear()

def insertToPath(ps1):
    if(ps1[1] == "m"):
        canvas_path.append(cv.Path.MoveTo(ps1[0][0],ps1[0][1]))
    else:
        canvas_path.append(cv.Path.LineTo(ps1[0][0],ps1[0][1]))
    page.update()
    time.sleep(1-visualization_speed[0])

def insertToPathResult(ps1):
    if(ps1[1] == "m"):
        result_path.append(cv.Path.MoveTo(ps1[0][0],ps1[0][1]))
    else:
        result_path.append(cv.Path.LineTo(ps1[0][0],ps1[0][1]))
```

```
    page.update()
    time.sleep(1-visualization_speed[0])

def insertToPathCircle(p):
    s = 5
    initial_circle.append(cv.Path.Oval(p[0]-(0.5*s),p[1]-(0.5*s),s,s))
    page.update()

def insertToPathResultCircle(p):
    s = 5
    result_circle.append(cv.Path.Oval(p[0]-(0.5*s),p[1]-(0.5*s),s,s))
    page.update()

def insertToPathCartesianLineCanvas(ps):
    if(ps[1] == "m"):
        cartesian_line_canvas.append(cv.Path.MoveTo(ps[0][0],ps[0][1]))
    else:
        cartesian_line_canvas.append(cv.Path.LineTo(ps[0][0],ps[0][1]))
    page.update()

def insertToPathCartesianLineResultCanvas(ps):
    if(ps[1] == "m"):
        cartesian_line_result_canvas.append(cv.Path.MoveTo(ps[0][0],ps[0][1]))
    else:
        cartesian_line_result_canvas.append(cv.Path.LineTo(ps[0][0],ps[0][1]))
    page.update()

def insertToPathCartesianTextNumberCanvas(ps):
    cartesian_text_number_canvas.append(
        cv.Text(
            ps[0][0]-5,
            ps[0][1]-5,
            ps[1],
            ft.TextStyle(size=ps[2]),
        )
    )
    page.update()

def insertToPathCartesianTextNumberResultCanvas(ps):
```

```
        t = cv.Text(
                ps[0][0]-5,
                ps[0][1]-5,
                ps[1],
                ft.TextStyle(size=ps[2]),
            )
        if(t not in cartesian_text_number_result_canvas and t not in cartesian_text_number_canvas):
            cartesian_text_number_result_canvas.append(t)
        page.update()

    def insertToListView(p):
        if(str(p[0]) == "process time"):
            listview_controls.append(
                ft.Container(
                    margin=ft.margin.only(left=15),
                    content=ft.Column(
                        controls=[
                            ft.Text("Process time (Netto): " + str(p[1]*1000) + " ms",size=12,color="#000000",
weight=800,font_family="Railinc"),
                            ft.Text(" ",size=5,color="#000000"),
                        ]
                    )
                )
            )
        else:
            listview_controls.append(
                ft.Container(
                    margin=ft.margin.only(left=15),
                    content=ft.Column(
                        controls=[
                            ft.Text(str(len(listview_controls)-1)+ ". " + str(p),size=12,color="#000000"),
                            ft.Text(" ",size=5,color="#000000"),
                        ]
                    )
                )
            )
        page.update()

    def initCoordinate_to_canvasCoordinate(p):
```

```
        min_y = min([point[1] for point in initial_points])
        min_x = min([point[0] for point in initial_points])
        if(range_y[0] > range_x[0]):
            coordinate_multiplier = float((app_h*0.85)/range_y[0])
            y = (app_h*0.85) - ((p[1]-min_y)*coordinate_multiplier)
            x = p[0]*coordinate_multiplier +
(float(((app_h*0.85)-(range_x[0]*coordinate_multiplier))/2)-(min_x*coordinate_multiplier))
        else:
            coordinate_multiplier = float((app_h*0.85)/range_x[0])
            x = (p[0]-min_x)*coordinate_multiplier
            y = (app_h*0.85)-(p[1]*coordinate_multiplier +
(float(((app_h*0.85)-(range_y[0]*coordinate_multiplier))/2)-(min_y*coordinate_multiplier)))
        return (x,y)

    def getMidPoint(p1,p2):
        x = float((p1[0]+p2[0])/2 )
        y = float((p1[1]+p2[1])/2)
        pout = (x,y)
        return pout

    def BezierN_process_time(points, iterasi, iterasiMax):
        if (iterasi >= iterasiMax):
            return []
        else:
            q = points
            left = [q[0]]
            right = [q[-1]]
            while len(q) > 1:
                temp = q
                q = [getMidPoint(temp[i], temp[i+1]) for i in range(len(temp)-1)]

                left.append(q[0])
                right.append(q[-1])
            right.reverse()
            if iterasi == 0:
                iterasi += 1
                output = [points[0]] + BezierN_process_time(left, iterasi, iterasiMax)
                return  output + [left[-1]] + BezierN_process_time(right, iterasi, iterasiMax) + [points[-1]]
            else:
```

```
            iterasi += 1
            output = BezierN_process_time(left, iterasi, iterasiMax)
            return output + [left[-1]] + BezierN_process_time(right, iterasi, iterasiMax)

    def BezierN(points, iterasi, iterasiMax):
        if(iterasi == 0):
            insertToPathResultCircle(initCoordinate_to_canvasCoordinate(points[0]))
            insertToListView(points[0])
        offset=5
        if (iterasi >= iterasiMax):
            return []
        else:
            q = points
            left = [q[0]]
            right = [q[-1]]
            while len(q) > 1:
                temp = q
                q = [getMidPoint(temp[i], temp[i+1]) for i in range(len(temp)-1)]
                left.append(q[0])
                right.append(q[-1])
                for i in range(len(temp)):
                    if(i == 0):
                        insertToPath((initCoordinate_to_canvasCoordinate(temp[0]),"m"))
                    else:
                        insertToPath((initCoordinate_to_canvasCoordinate(temp[i]),"l"))
            right.reverse()
            canvas_point = initCoordinate_to_canvasCoordinate(q[0])
            if(iterasi == iterasiMax-1):

                insertToPathResultCircle(canvas_point)

                insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
                insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(q[0][0
],2)),8))

                insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
                insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))
```

```
insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(q[0][1],2)),8))

            insertToPathResult((initCoordinate_to_canvasCoordinate(points[0]),"m"))
            insertToPathResult((canvas_point,"l"))
            insertToPathResult((initCoordinate_to_canvasCoordinate(points[-1]),"l"))
        elif(iterasi < iterasiMax-1):
            insertToPath((initCoordinate_to_canvasCoordinate(points[0]),"m"))
            insertToPath((canvas_point,"l"))
            insertToPath((initCoordinate_to_canvasCoordinate(points[-1]),"l"))

        if iterasi == 0:
            iterasi += 1

            output = [points[0]]

            output += BezierN(left, iterasi, iterasiMax)

            insertToListView(left[-1])

            canvas_point = initCoordinate_to_canvasCoordinate(left[-1])
            insertToPathResultCircle(canvas_point)
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(left[-1
][0],2)),8))

            insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
            insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))

insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(left[-1][1],2)),8))

            output += [left[-1]]

            output += BezierN(right, iterasi, iterasiMax)

            output += [points[-1]]
```

```
            insertToPathResultCircle(initCoordinate_to_canvasCoordinate(points[-1]))
            insertToListView(points[-1])
            return output

        else:
            iterasi += 1
            output = BezierN(left, iterasi, iterasiMax)

            insertToListView(left[-1])

            canvas_point = initCoordinate_to_canvasCoordinate(left[-1])
            insertToPathResultCircle(canvas_point)
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85-offset),"m"))
            insertToPathCartesianLineResultCanvas(((canvas_point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberResultCanvas(((canvas_point[0],0+app_h*0.85+offset+15),str(round(left[-1
][0],2)),8))

            insertToPathCartesianLineResultCanvas(((0-offset,canvas_point[1]),"m"))
            insertToPathCartesianLineResultCanvas(((0+offset,canvas_point[1]),"l"))

insertToPathCartesianTextNumberResultCanvas(((0-offset-15,canvas_point[1]),str(round(left[-1][1],2)),8))

            output += [left[-1]]

            output += BezierN(right, iterasi, iterasiMax)

            return output

    def buttonVisualizeClicked(e):
        resetPath()
        e.control.visible = False
        BezierN(initial_points,0,num_of_iteration[0])
        startT = time.time()
        BezierN_process_time(initial_points,0,num_of_iteration[0])
        endT = time.time()
        process_time[0] = endT-startT
        insertToListView(("process time",endT-startT))
        e.control.visible = True
```

```
    page.update()

buttonVisualize = ft.ElevatedButton(
    style=ft.ButtonStyle(
            color={
                ft.MaterialState.DEFAULT: "#FFFFFF",
                ft.MaterialState.HOVERED: "#000000",
                ft.MaterialState.DISABLED: "#000000",
            },
            bgcolor={ft.MaterialState.DEFAULT: "#000000",
                    ft.MaterialState.HOVERED: "#FFFFFF",
                    ft.MaterialState.DISABLED: "#FFFFFF"
            },
            padding={ft.MaterialState.DEFAULT: 28},
            overlay_color=ft.colors.TRANSPARENT,
            elevation={"pressed": 0, "": 1},
            side={
                ft.MaterialState.DEFAULT: ft.BorderSide(1, "#000000"),
            },
        ),
    width=0.15*app_w,
    on_click=buttonVisualizeClicked,
    content=ft.Text("Visualize!",size=15,font_family="Railinc")
)

def inputChanged(e):
    resetInitial()
    resetPath()
    temp_range_y = 0
    max_y = 0
    min_y = 0
    temp_range_x = 0
    max_x = 0
    min_x = 0
    coordinate_multiplier = 1
    isValid = True
    points_temp = e.control.value.split("_")
    n_points = len(points_temp)
    if(n_points < 3):
```

```
        e.control.error_text = "A curve consist of minimum 3 control points"
    else:
        for i in range(n_points):
            point = points_temp[i].split(",")
            if(len(point) == 2):
                try:
                    x = int(point[0])
                    y = int(point[1])
                except:
                    try:
                        x = float(point[0])
                        y = float(point[1])
                    except:
                        e.control.error_text = "x and y must be integer"
                        resetInitial()
                        isValid = False
                        break
                    else:
                        if(i==0):
                            max_y = y
                            min_y = y
                            max_x = x
                            min_x = x
                        else:
                            if(y > max_y):
                                max_y = y
                            elif(y < min_y):
                                min_y = y
                            if(x > max_x):
                                max_x = x
                            elif(x < min_x):
                                min_x = x
                        e.control.error_text = ""
                        initial_points.append((x,y))
                else:
                    if(i==0):
                        max_y = y
                        min_y = y
                        max_x = x
```

```
                    min_x = x
                else:
                    if(y > max_y):
                        max_y = y
                    elif(y < min_y):
                        min_y = y
                    if(x > max_x):
                        max_x = x
                    elif(x < min_x):
                        min_x = x
                e.control.error_text = ""
                initial_points.append((x,y))
            else:
                e.control.error_text = "Invalid format"
                resetInitial()
                isValid = False
                break

        if isValid:
            temp_range_y = max_y-min_y
            temp_range_x = max_x-min_x
            if(temp_range_y > temp_range_x):
                coordinate_multiplier = float((app_h*0.85)/temp_range_y)
                for point in initial_points:
                    y_coor = (app_h*0.85) - ((point[1]-min_y)*coordinate_multiplier)
                    x_coor = point[0]*coordinate_multiplier +
(float(((app_h*0.85)-(temp_range_x*coordinate_multiplier))/2)-(min_x*coordinate_multiplier))
                    coordinate_points.append((x_coor,y_coor))
            else:
                coordinate_multiplier = float((app_h*0.85)/temp_range_x)
                for point in initial_points:
                    x_coor = (point[0]-min_x)*coordinate_multiplier
                    y_coor = (app_h*0.85)-(point[1]*coordinate_multiplier +
(float(((app_h*0.85)-(temp_range_y*coordinate_multiplier))/2)-(min_y*coordinate_multiplier)))
                    coordinate_points.append((x_coor,y_coor))
            range_y[0] = temp_range_y
            range_x[0] = temp_range_x
            # Draw Axis and Ordinat line
            insertToPathCartesianLineCanvas(((0,0),"m"))
```

```
            insertToPathCartesianLineCanvas(((0,0+app_h*0.85),"l"))
            insertToPathCartesianLineCanvas(((0+app_h*0.85,0+app_h*0.85),"l"))
            # Draw Circle (Dot)
            for i in range(len(coordinate_points)):
                point = coordinate_points[i]
                insertToPathCircle(point)
                offset = 5
                if(point[0] not in initial_canvas_axis):
                    insertToPathCartesianLineCanvas(((point[0],0+app_h*0.85-offset),"m"))
                    insertToPathCartesianLineCanvas(((point[0],0+app_h*0.85+offset),"l"))

insertToPathCartesianTextNumberCanvas(((point[0],0+app_h*0.85+offset+15),str(round(initial_points[i][0],2)
),8))
                    initial_canvas_axis.append(point[0])
                if(point[1] not in initial_canvas_ordinat):
                    insertToPathCartesianLineCanvas(((0-offset,point[1]),"m"))
                    insertToPathCartesianLineCanvas(((0+offset,point[1]),"l"))

insertToPathCartesianTextNumberCanvas(((0-offset-15,point[1]),str(round(initial_points[i][1],2)),8))
                    initial_canvas_ordinat.append(point[1])
        page.update()

    def iterationChanged(e):
        try:
            num_of_iteration[0] = int(e.control.value)
        except:
            num_of_iteration[0] = 0
            e.control.error_text = "Value must be integer"
        else:
            e.control.error_text = ""
        page.update()


    def slider_change(e):
        visualization_speed[0] = e.control.value
        page.update()

    container = ft.Container(
        height=app_h*0.92,
```

```
    width=app_w,
    bgcolor="#C7C8CC",
    content=ft.Column(
        controls=[
            ft.Row(
                alignment="center",
                vertical_alignment="center",
                controls=[
                    ft.Container(
                        content=ft.Column(
                            height=app_h-49,
                            alignment="center",
                            controls=[
                                ft.Container(
                                    height=app_h*0.85,
                                    width=app_h*0.85,
                                    alignment=ft.alignment.center,
                                    margin=ft.margin.only(left=20,bottom=20),
                                    content=ft.Column(
                                        controls=[
                                            ft.Stack(
                                                controls=[
                                                    cv.Canvas(
                                                        [
                                                            cv.Path(
                                                                cartesian_line_canvas,
                                                                paint=paint_cartesian_line_canvas
                                                            )
                                                        ],
                                                        width=float("inf"),
                                                        expand=True,
                                                    ),
                                                    cv.Canvas(
                                                        [
                                                            cv.Path(
                                                                cartesian_line_result_canvas,
                                                                paint=paint_cartesian_line_canvas
                                                            )
                                                        ],
```

```
            width=float("inf"),
            expand=True,
        ),
        cv.Canvas(
            [
                cv.Path(
                    canvas_path,
                    paint=paint_canvas
                )
            ],
            width=float("inf"),
            expand=True,
        ),
        cv.Canvas(
            [
                cv.Path(
                    result_path,
                    paint=paint_result_canvas
                )
            ],
            width=float("inf"),
            expand=True,
        ),
        cv.Canvas(
            [
                cv.Path(
                    initial_circle,
                    paint=paint_initial_circle,
                )
            ],
            width=float("inf"),
            expand=True,
        ),
        cv.Canvas(
            [
                cv.Path(
                    result_circle,
                    paint=paint_result_circle,
                )
```

```
                                ],
                                width=float("inf"),
                                expand=True,
                            ),
                            cv.Canvas(
                                cartesian_text_number_canvas
                            ),
                            cv.Canvas(
                                cartesian_text_number_result_canvas
                            ),
                        ]
                    )
                ]
            )
        )
    ]
  )
),
ft.Container(
  height=app_h*0.8545,
  width=app_h*0.855,
  # bgcolor="#000000",
  content=ft.Column(
    horizontal_alignment="center",
    alignment="center",
    controls=[
      # TITLE
      ft.Row(
        alignment="center",
        controls=[
          ft.Text("Bézier Curve",size=40,font_family="Railinc")
        ]
      ),
      # INPUT
      ft.Container(
        margin=ft.margin.only(top=0.03*app_h),
        alignment=ft.alignment.center,
        width=app_w*0.45,
        content=ft.Row(
```

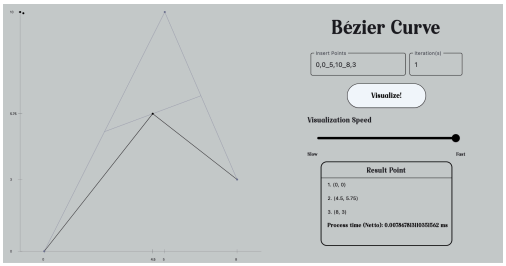
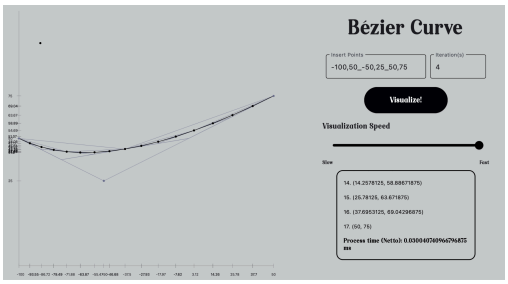
```
                alignment="center",
                controls=[
                    ft.Container(
                        width=app_w*0.18,
                        content=ft.TextField(
                            label="Insert Points",
                            hint_text="x0,y0_x1_y1_x2,y2_...",
                            on_change=inputChanged,
                        )
                    ),
                    ft.Container(
                        width=app_w*0.1,
                        alignment=ft.alignment.center,
                        content=ft.TextField(
                            label="Iteration(s)",
                            on_change=iterationChanged,
                        )
                    )
                ],
            )
        ),
        # Button "Visualize!"
        ft.Container(
            margin=ft.margin.only(top=10,bottom=10),
            content=buttonVisualize
        ),
        # SPEED SLIDER
        ft.Container(
            width=0.3 * app_w,
            content=ft.Column(
                controls=[
                    ft.Text("Visualization Speed",size=15,font_family="Railinc"),
                    ft.Slider(
                        min=0,
                        max =1,
                        divisions=100,
                        active_color= "#000000",
                        on_change=slider_change
                    ),
```

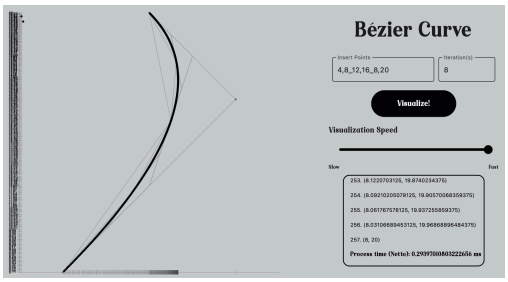
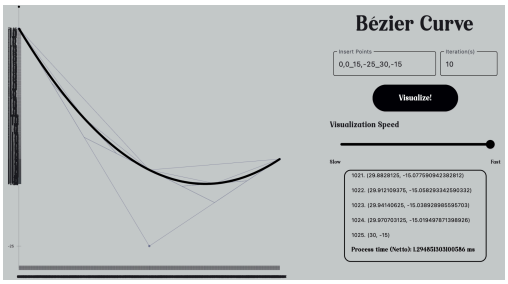
```
                                ft.Row(
                                    alignment=ft.MainAxisAlignment.SPACE_BETWEEN,
                                    controls=[
                                        ft.Container(
                                            content=ft.Text("Slow",size=10,font_family="Railinc")
                                        ),
                                        ft.Container(
                                            content=ft.Text("Fast",size=10,font_family="Railinc")
                                        )
                                    ]
                                )
                            ]
                        )
                    ),
                    # ListView
                    ft.Container(
                        width=0.25*app_w,
                        height=app_h*0.3,
                        border=ft.border.all(2, "#000000"),
                        border_radius=15,
                        content=ft.ListView(
                            auto_scroll=True,
                            controls=listview_controls,
                        )
                    )

                ]
            )
        ),
    ]
  )
)

page.add(container)
page.update()

ft.app(target=main)
```

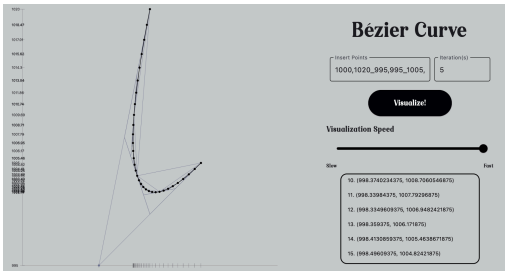
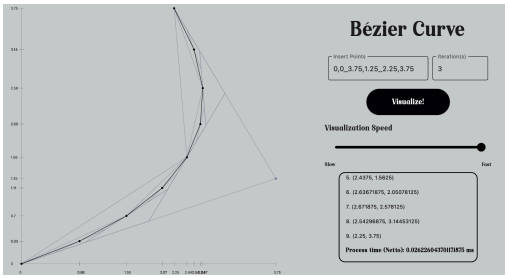
BAB 5

Test Case

Tabel 5.1. Tabel Test Case 3 titik

No	Input	Brute Force	Divide and Conquer
1	Points = [(0,0), (5,10), (8,3)] Iterasi = 1	x1 = 0 y1 = 0 x2 = 5 y2 = 10 x3 = 8 y3 = 3 Iterasi: 1 Brute Force: 0.11801719665527344 [(0, 0), (4.5, 5.75), (8.0, 3.0)]	 1. (0, 0) 2. (4.5, 5.75) 3. (8, 3) **Process time (Netto): 0.007867813110351562 ms**
2	Points = [(100,-50), (-50,25), (50,75)] Iterasi = 4	x1 = -100 y1 = 50 x2 = -50 y2 = 25 x3 = 50 y3 = 75 Iterasi: 4 Brute Force: 0.19502639770507812 [(-100, 50), (-93.5546875, 47.16796875), (-86.71875, 44.921875), (-79.4921875, 43.26171875), (-71.875, 42.1875), (-63.8671875, 41.69921875), (-55.46875, 41.796875), (-46.796875, 42.48046875), (-37.5, 43.75), (-27.9296875, 45.60546875), (-17.96875, 48.046875), (-7.6171875, 51.07421875), (3.125, 54.6875), (14.2578125, 58.88671875), (25.78125, 63.671875), (37.6953125, 69.04296875), (50.0, 75.0)]	 1. (-100, 50) 2. (-93.5546875, 47.16796875) 3. (-86.71875, 44.921875) 4. (-79.4921875, 43.26171875) 5. (-71.875, 42.1875) 6. (-63.8671875, 41.69921875)

			7. (-55.46875, 41.796875) 8. (-46.6796875, 42.48046875) 9. (-37.5, 43.75) 10. (-27.9296875, 45.60546875) 11. (-17.96875, 48.046875) 12. (-7.6171875, 51.07421875) 13. (3.125, 54.6875) 14. (14.2578125, 58.88671875) 15. (25.78125, 63.671875) 16. (37.6953125, 69.04296875) 17. (50, 75) **Process time (Netto): 0.030040740966796875 ms**
3	Points = [(4,8), (12,16), (8,20)] Iterasi = 8	x1 = 4 y1 = 8 x2 = 12 y2 = 16 x3 = 8 y3 = 20 Iterasi: 8 Brute Force: 0.2999305725097656 [(4, 8), (4.06231689453125, 8.06243896484375), (4.124267578125, 8.124755859375), (4.18585205078125, 8.186950683593775), (4.2470703125, 8.2490234375), (4.30792236328125, 8.31097412109375), (4.368408203125, 8.372802734375), (4.428527832031250, 8.43450927734375), (4.48828125, 8.49609375), (4.54766845703125, 8.55755615234375), (4.606689453125, 8.618896484375), (4.66534423828125, 8.68011474609375), (4.7	 253. (8.1220703125, 19.8740234375) 254. (8.09210205078125, 19.90570068359375) 255. (8.061767578125, 19.937255859375) 256. (8.03106689453125, 19.96868896484375) 257. (8, 20) **Process time (Netto): 0.29397010803222656 ms**
4	Points = [(0,0), (15,-25), (30,-15)] Iterasi = 10	x1 = 0 y1 = 0 x2 = 15 y2 = -25 x3 = 30 y3 = -15 Iterasi: 10 Brute Force: 1.024007797241211 [(0, 0), (0.029296875, -0.04879474639892578), (0.05859375, -0.09752273559570312), (0.087890625, -0.14618396759033203), (0.1171875, -0.1947784423828125), (0.146484375, -0.24330615997314453), (0.17578125, -0.2917671203613281), (0.205078125, -0.3401613235473633), (0.234375, -0.3884887695315125), (0.263671875, -0.4367494583129883), (0.29296875, -0.48494338989257781), (0.322265625, -0.5330705642700195), (0.3515625, -0.5811309814453125), (0.380859375, -0.629124641418457), (0.41015625, -0.6770515441894531), (0.4394531	

			1021. (29.8828125, -15.077590942382812) 1022. (29.912109375, -15.058293342590332) 1023. (29.94140625, -15.038928985595703) 1024. (29.970703125, -15.019497871398926) 1025. (30, -15) **Process time (Netto): 1.294851303100586 ms**
5	Points = [(1000,1020), (995,995), 1005, 1005)] Iterasi = 5	x1 = 1000 y1 = 1020 x2 = 995 y2 = 995 x3 = 1005 y3 = 1005 Iterasi: 5 Brute Force: 0.16164779663085938 [(1000, 1020), (999.7021484375, 1018.4716796875), (999.43359375, 1017.01171875), (999.1943359375, 1015.6201171875), (998.984375, 1014.296875), (998.8037109375, 1013.0419921875), (998.65234375, 1011.85546875), (998.5302734375, 1010.7373046875), (998.4375, 1009.6875), (998.3740234375, 1008.7060546875), (998.33984375, 1007.79296875), (998.3349609375, 1006.9482421875), (998.359375, 1006.171875), (998.4130859375, 1005.4638671875), (998.49609375, 1004.8242187 5), (998.6083984375, 1004.2529296875), (998.75, 1003.75)	 29. (1002.734375, 1003.046875) 30. (1003.2568359375, 1003.4326171875) 31. (1003.80859375, 1003.88671875) 32. (1004.3896484375, 1004.4091796875) 33. (1005, 1005) **Process time (Netto): 0.05817413330078125 ms**
6	Points = [(0,0), (3.75, 1.25), (2.25, 3.75)] Iterasi = 3	x1 = 0 y1 = 0 x2 = 3.75 y2 = 1.25 x3 = 2.25 y3 = 3.75 Iterasi: 3 Brute Force: 0.17595291137695312 [(0.0, 0.0), (0.85546875, 0.33203125), (1.546875, 0.703125), (2.07421875, 1.11328125), (2.4375, 1.5625), (2.63671875, 2.05078125), (2.671875, 2.578125), (2.54296875, 3.14453125), (2.25, 3.75)]	 5. (2.4375, 1.5625) 6. (2.63671875, 2.05078125) 7. (2.671875, 2.578125) 8. (2.54296875, 3.14453125) 9. (2.25, 3.75) **Process time (Netto): 0.026226043701171875 ms**

BAB 6

Perbandingan Brute Force dan Divide and Conquer

6.1. Kompleksitas Brute Force

Pada algoritma Brute Force untuk Kurva Bezier Kuadratik (3 titik kontrol), dilakukan perhitungan terhadap persamaan kurva bezier kuadratik untuk $2^n - 1$ buah titik dengan n adalah jumlah iterasi. Kompleksitas akan dihitung berdasarkan operasi pertambahan, pengurangan, perkalian, pembagian, perpangkatan, dan konkatenasi. Pada awal algoritma, dihitung nilai 2^n dengan 1 operasi perpangkatan dan nilai $\frac{1}{2^n}$ dengan 1 operasi pembagian. Pada setiap evaluasi persamaan, dilakukan 4 operasi pengurangan, 5 operasi pertambahan, 4 operasi perpangkatan, dan 10 operasi perkalian. Dengan demikian, kompleksitas waktu dari algoritma Brute Force adalah sebagai berikut.

$$T(n) = 23(2^n - 1) + 2 = 23(2^n) - 21$$
$$O(n) = 2^n$$

6.2. Kompleksitas Divide and Conquer

Pada algoritma Divide and Conquer untuk Kurva Bezier Kuadratik (3 titik kontrol), dilakukan rekursi hingga iterasi telah dilakukan sebanyak n kali. Kompleksitas akan dihitung berdasarkan operasi pertambahan, pengurangan, perkalian, pembagian, perpangkatan, dan konkatenasi. Pada setiap rekursi dilakukan 3 kali perhitungan titik tengah, dengan melakukan 2 operasi pertambahan dan 2 operasi pembagian pada tiap iterasi, sehingga dilakukan 6 kali operasi pertambahan dan 6 kali operasi pembagian pada tiap rekursi. Kemudian setiap fungsi memanggil dirinya secara rekursif 2 kali dengan iterasi yang harus dilakukan berkurang satu. Kemudian, dilakukan konkatenasi sebanyak 2 kali (kecuali untuk kasus iterasi paling pertama sebanyak 4 kali, tapi tidak terlalu berpengaruh sehingga diabaikan) untuk menggabungkan hasil. Dengan demikian, didapatkan kompleksitas waktu sebagai berikut.

$$T(n) = 2T(n - 1) + 8 \; dan \; T(1) = 8$$
$$T(n) = 8(2^n) - 8$$
$$O(n) = 2^n$$

6.3. Perbandingan dan Analisis

Berdasarkan kompleksitas Big O, kedua algoritma tersebut memiliki kompleksitas O(2^n). Hal tersebut menjelaskan mengapa kedua algoritma tersebut memiliki waktu proses yang tidak terlalu berbeda dalam melakukan perhitungan titik-titik dalam kurva bezier. Tapi, berdasarkan kompleksitas T(n), algoritma Divide and Conquer memiliki T(n) yang lebih kecil. Hal ini menjelaskan mengapa pada iterasi yang tidak terlalu besar, algoritma Divide and Conquer memiliki waktu proses yang lebih cepat dibandingkan brute force. Tetapi, ketika jumlah iterasi dibuat besar, algoritma Brute Force akan menjadi lebih cepat daripada Divide and Conquer. Hal ini disebabkan karena operasi konkatenasi list pada python memiliki kompleksitas O(n) dengan n adalah jumlah elemen pada list yang dikonkatenasi. Konkatenasi hanya dilakukan pada algoritma Divide and Conquer dan ketika jumlah iterasi semakin besar, maka konkatenasi akan menjadi semakin lama karena jumlah elemen list semakin banyak dan membuat algoritma Divide and Conquer menjadi lebih lambat daripada Brute Force.

BAB 7

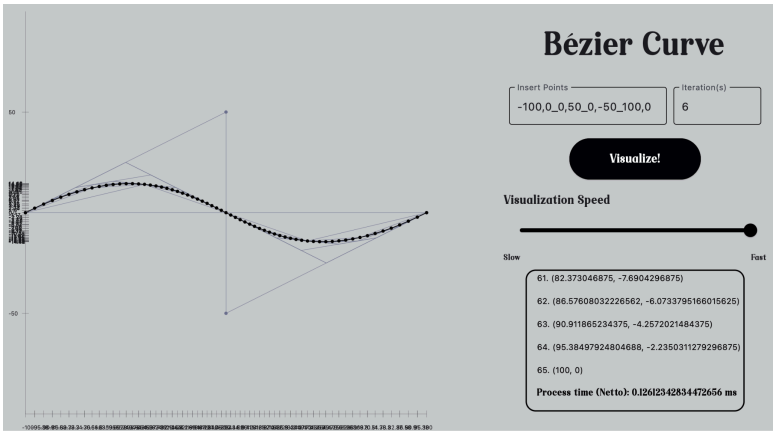
Implementasi Bonus

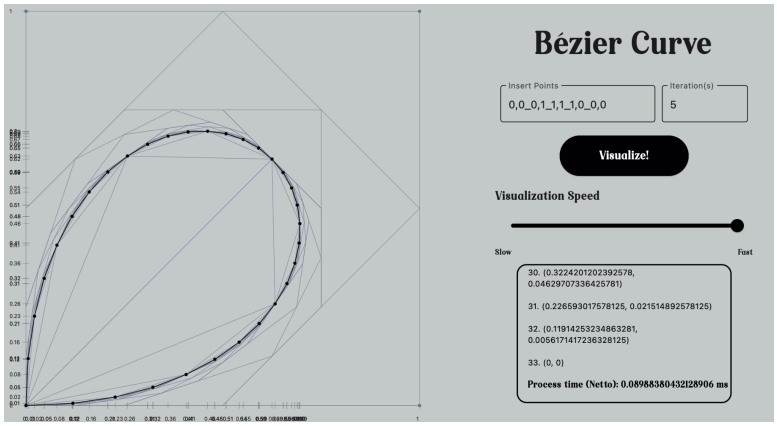
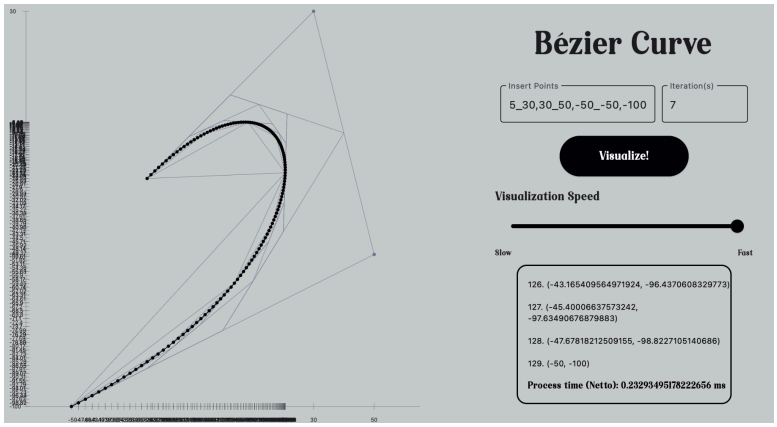
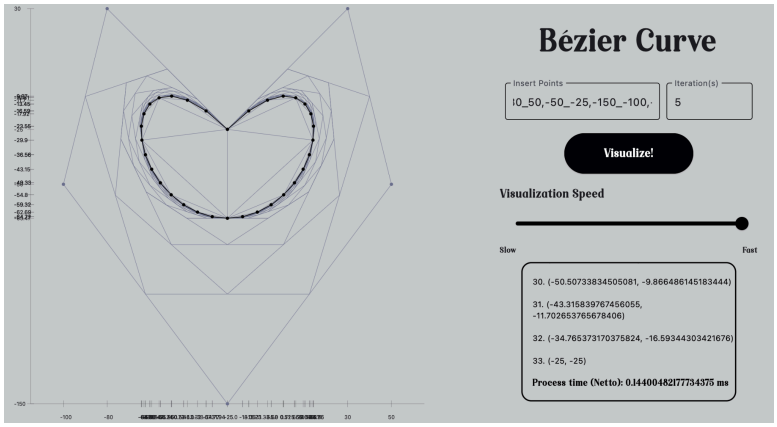
7.1. Generalisasi Algoritma (n-titik)

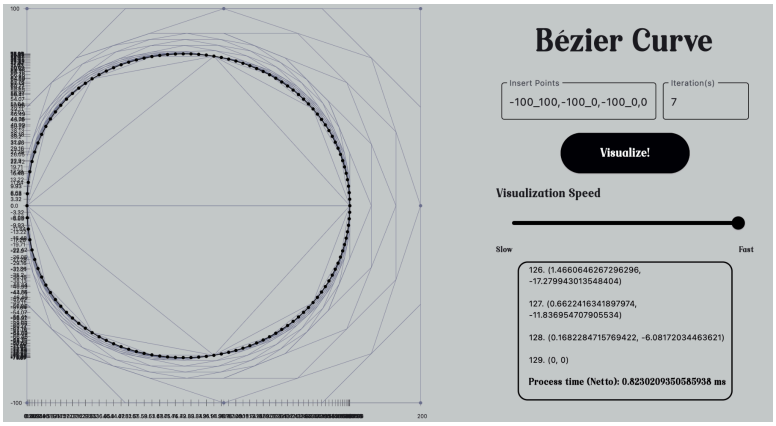
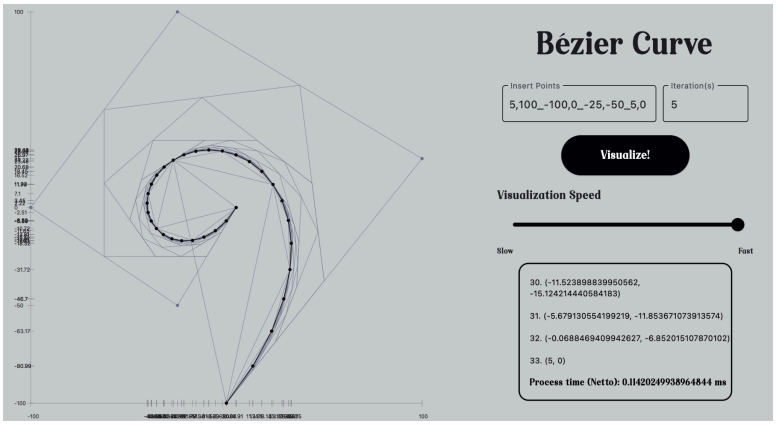
Generalisasi algoritma Kurva Bezier dilakukan dengan menerima input list of titik dengan panjang list n dan jumlah iterasi yang ingin dilakukan. Implementasi Algoritma Divide and Conquer untuk Kurva Bezier dengan input n buah titik kami buat dengan melakukan langkah-langkah sebagai berikut.

1. Append koordinat titik paling kiri dari array input (p) ke array Left.
2. Append koordinat titik paling kanan dari array input ke array Right.
3. Cari koordinat titik tengah dari tiap titik dengan index bersebelahan dari array input, kemudian masukkan ke suatu array.
4. Append koordinat titik paling kiri dari array ke array Left.
5. Append koordinat titik paling kanan dari array ke array Right.
6. Ulangi step 3-5 hingga tersisa 1 buah titik, yang merupakan titik solusi dari iterasi tersebut.
7. Reverse list Right.
8. Lakukan rekursi dengan titik input Left dan jumlah rekursi yang dilakukan berkurang 1 sebagai s0.
9. Lakukan rekursi dengan titik input Right dan jumlah rekursi yang dilakukan berkurang 1 sebagai s1.
10. Jika merupakan iterasi pertama, return p[0] + s0 + right[0] + s1 + p[1].
11. Jika bukan merupakan iterasi pertama, return s0 + right[0] + s1.

Tabel 7.1.1 Tabel Test Case Bonus (n-titik)

No	Input	Output
1	Points = [(-100,0), (0,50), (0,-50), (100,0)] Iterasi = 6	

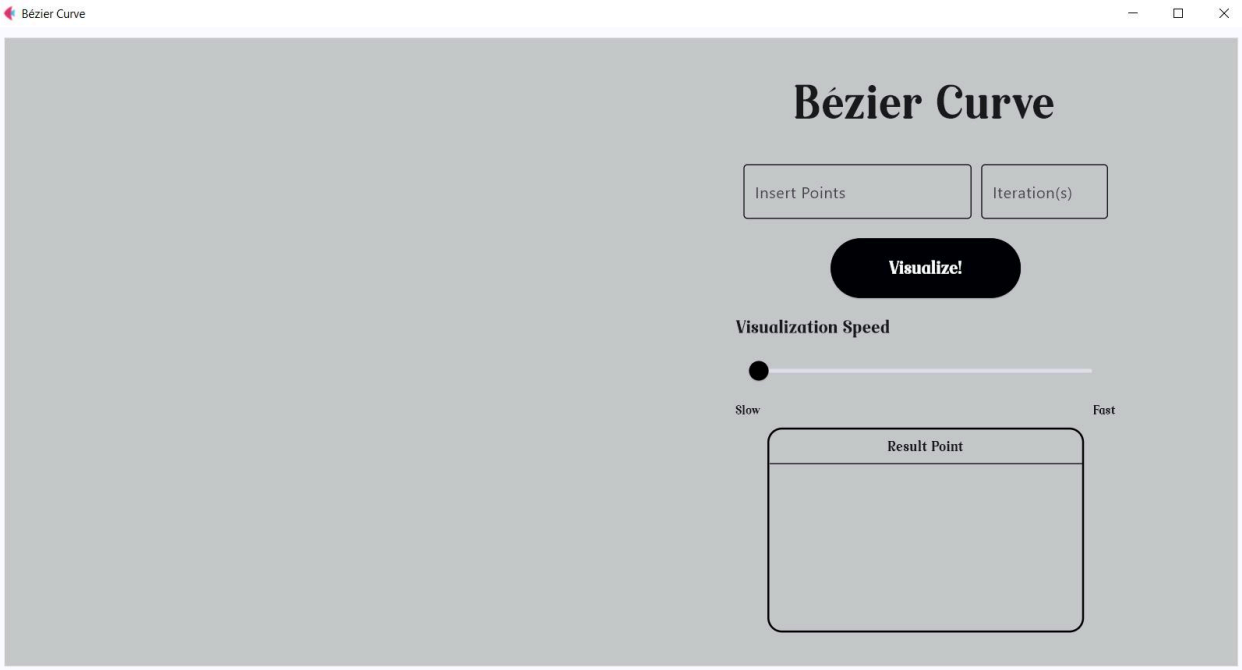
2	Points = [(0,0), (0,1), (1,1), (1,0), (0,0)] Iterasi = 5	
3	Points = [(-25,-25), (30,30), (50,-50), (-50,-100)] Iterasi = 7	
4	Points = [(-25,-25), (30,30), (50,-50), (-25,-150), (-100,-50), (-80,30), (-25,-25)] Iterasi = 5	

5	Points = [(0,0), (0,100), (100,100), (200,100), (200,0), (200,-100), (100,-100), (0,-100), (0,0)] Iterasi = 6	
6	Points = [(0,-100), (100,25), (-25,100), (-100,0), (-25,-50), (5,0)] Iterasi = 5	

7.2. GUI Visualisasi Bézier Curve

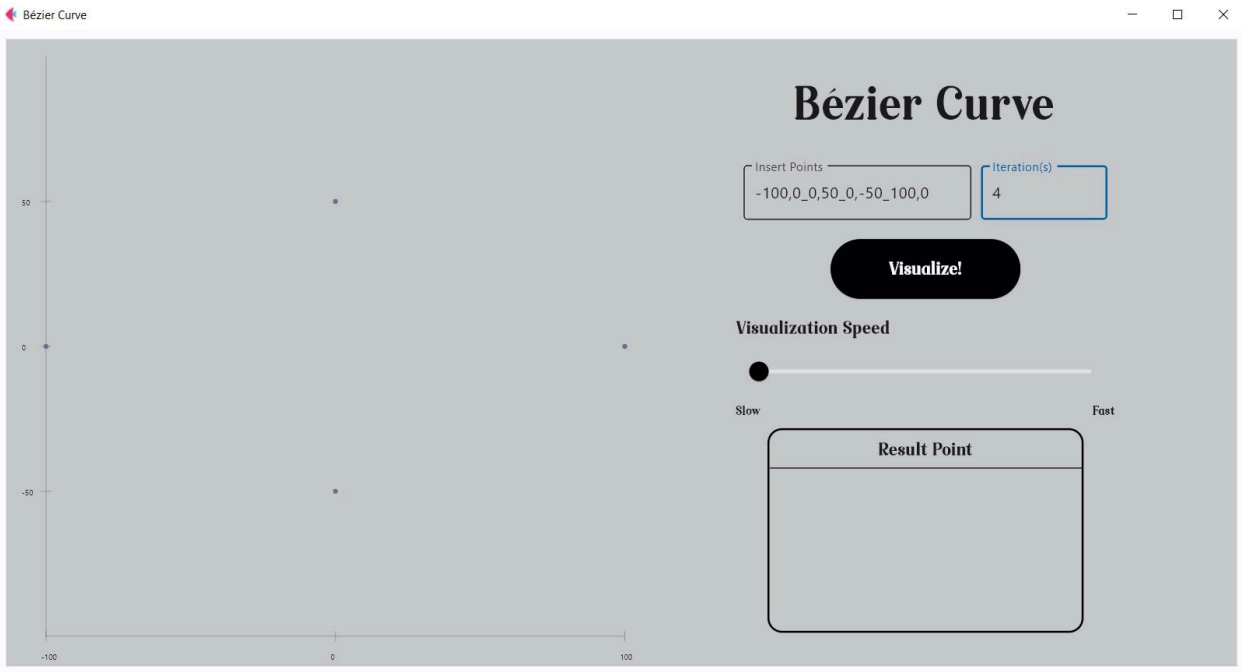
7.2.1 Tampilan Utama

Tampilan utama GUI terdiri dari 6 komponen: *textfield* untuk masukan titik koordinat, *textfield* untuk masukan jumlah iterasi yang ingin dilakukan, *button* untuk menunjukkan proses pembentukan kurva, *slider* untuk mengatur kecepatan dari proses pembentukan kurva, *listview* untuk menampilkan hasil akhir titik koordinat dari kurva yang terbentuk, dan *canvas* untuk membentuk kurva. Tujuan penggunaan *listview* disini adalah untuk memperjelas titik koordinat dari kurva yang dihasilkan karena semakin banyak iterasi maka akan semakin banyak titik yang terbentuk sehingga informasi koordinat (angka pada axis dan ordinat) akan tumpang tindih sehingga sulit dilihat.



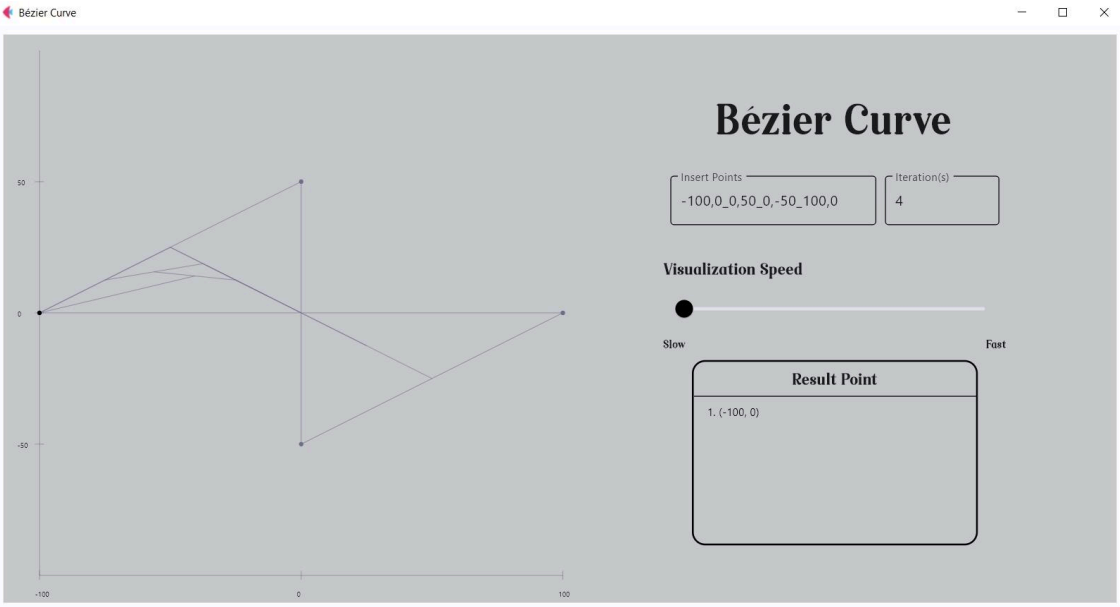
7.2.2 Memasukkan Input

GUI dapat menerima input titik koordinat dan jumlah iterasi yang ingin dilakukan. Masukan titik koordinat memiliki format yang harus diikuti, yaitu setiap pasangan x,y harus dipisahkan dengan '_' dan nilai x dan y harus berupa bilangan bulat atau real (terdapat validasi). Masukan jumlah iterasi tidak memiliki format tertentu, tetapi jumlah iterasi haruslah berupa bilangan bulat (terdapat validasi).



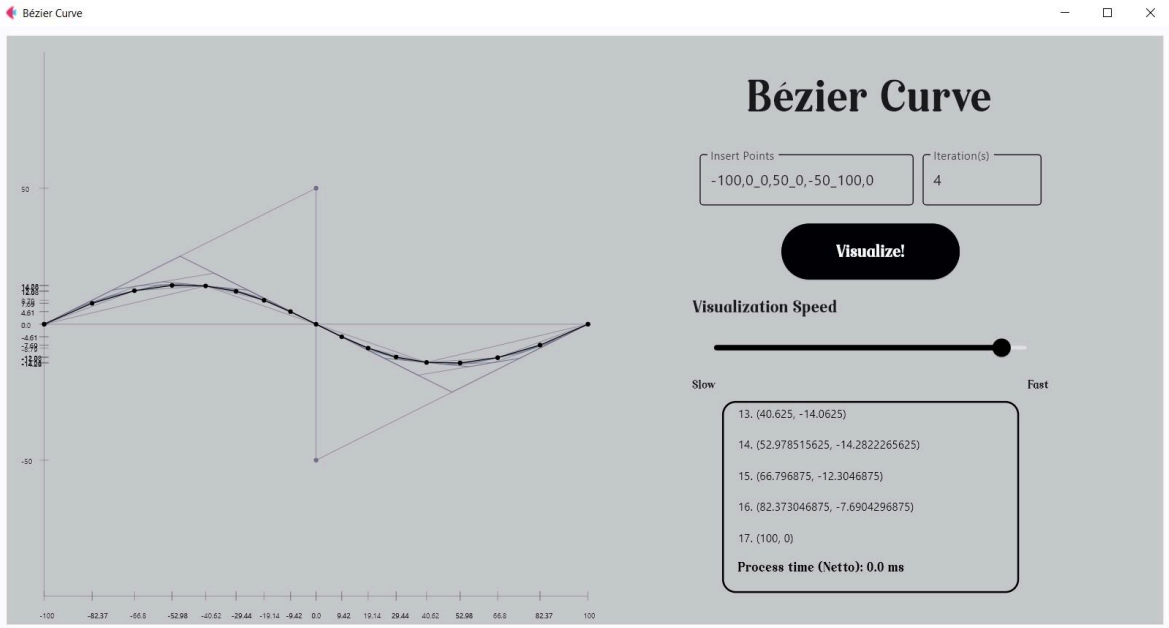
7.2.3 Proses Pembentukan Kurva

Proses pembentukan kurva dimulai setelah *button* “Visualize!” ditekan, setelah itu *button* akan menghilang dan akan muncul kembali setelah proses pembentukan kurva telah berakhir. Selama proses pembentukan kurva berlangsung, *slider* “Visualization Speed” dapat digunakan untuk mengatur kecepatan pembentukan kurva, semakin ke kanan akan semakin cepat.Selain itu juga terdapat *listview* untuk melihat titik koordinat kurva yang terbentuk seiring berjalannya waktu.



7.2.4 Hasil Akhir

Ketika proses telah berakhir, total waktu pemrosesan akan ditampilkan. Perlu diperhatikan bahwa waktu pemrosesan yang ditampilkan disini tidak termasuk proses visualisasi pada *canvas*, jadi benar-benar murni waktu eksekusi dari algoritma DnC.



Link Repository

https://github.com/Zechtro/Tucil2_13522020_13522103