

Laporan Tugas Kecil 2 Strategi Algoritma
Membangun Kurva Bézier dengan Algoritma Titik Tengah
berbasis Divide and Conquer
Semester II Tahun 2023/2024



oleh

Muhammad Atpur Rafif	13522086
Andhika Tantyo Anugrah	13522094

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

BAB 1	
Strategi Algoritma dengan Brute Force	3
BAB 2	
Strategi Algoritma dengan Divide and Conquer	5
BAB 3	
Source Code Program	6
BAB 4	
Eksperimen	8
BAB 5	
Hasil Analisis	11
BAB 6	
Penjelasan Bonus	12
BAB 7	
Pranala ke Repository	13
Daftar Pustaka	14
Lampiran	15

BAB 1

Strategi Algoritma dengan *Brute Force*

Algoritma *brute force* merupakan algoritma yang sifatnya mengandalkan kekuatan komputasi untuk mencari semua kemungkinan solusi. Algoritma ini dijamin selalu menghasilkan solusi yang optimal karena karakteristik pencarian solusinya yang mencoba semua kemungkinan, sehingga sering disebut *exhaustive search*. Sebuah titik $P_{i,n}$ merupakan titik yang dihasilkan pada urutan i , iterasi n . Pada sebuah kurva berderajat n yang dibuat menggunakan $n + 1$ titik. Kita perlu membuat sebuah titik $P_{0,n}$ yang akan dimasukan parameter t dengan domain dari 0 sampai 1, dengan definisi:

$$P_{i,n} = P_{i,n-1}(1 - t) + P_{i+1,n-1}t$$

Penjumlahan titik di atas berupa penjumlahan vektor. Nantinya, seluruh titik yang dihasilkan oleh fungsi $P_{0,n}$ dengan memasukan t akan ditarik garis sehingga membentuk kurva. Apabila kita mengaplikasikannya pada kurva berderajat dua dengan *control points* berupa $P_{0,0}$, $P_{1,0}$, $P_{2,0}$, akan didapatkan:

$$P_{0,1} = P_{0,0}(1 - t) + P_{1,0}t$$

$$P_{1,1} = P_{1,0}(1 - t) + P_{2,0}t$$

$$P_{0,2} = P_{0,1}(1 - t) + P_{1,1}t = P_{0,0}(1 - t)^2 + 2P_{0,1}(1 - t)t + P_{2,0}t^2$$

Perhatikan bahwa kita koefisien dari setiap titik pada iterasi n selalu sama, namun hanya titik yang dihasilkan pada iterasi sebelumnya yang digunakan berbeda. Sehingga apabila kita menambahkan $P_{3,0}$ untuk membuat kurva kubik, kita bisa menggunakan hasil pada kurva kuadratik.

Selain itu, pada urutan iterasi ke- i , kedua titik yang dibutuhkan untuk membuat derajat tiga memiliki derajat t dan $(t - 1)$ yang selalu sama. Sehingga kita bisa menambahkannya dengan cara sebagai berikut:

$P_{0,2}(1 - t) =$	$P_{0,0}(1 - t)^3$	$2P_{0,1}(1 - t)^2t$	$P_{0,2}(1 - t)t^2$	
$P_{1,2}t =$		$P_{0,1}(1 - t)^2t$	$2P_{0,2}(1 - t)t^2$	$P_{0,3}t^3$
$P_{0,3} =$	$P_{0,0}(1 - t)^3$	$3P_{0,1}(1 - t)^2t$	$3P_{0,2}(1 - t)t^2$	$P_{0,3}t^3$

Kemudian kita bisa meneruskan pola diatas untuk derajat n , dan mendapatkan koefisien dari setiap titik $P_{i,0}$ memiliki bentuk seperti pola Segitiga Pascal. Terdapat argumen lain untuk membuktikan hal ini. Titik $P_{i,0}$ selalu memiliki suku $(1 - t)^{n-i}t^i$ pada ekspansi $P_{0,n}$. Pada setiap iterasinya, $P_{i,0}$ dapat memilih suku $(1 - t)$ dan t , sehingga koefisien yang didapatkan berupa banyaknya kombinasi dari $(1 - t)$ dan t yang akan membentuk $(1 - t)^{n-i}t^i$. Singkatnya, koefisiennya berupa $\binom{i}{n} = \frac{n!}{i!(n-i)!}$.

Generalisasi di atas akan menghasilkan rumus sebagai berikut:

$$P_{0,n}(t) = \sum_{i=0}^n \binom{i}{n} P_{i,0} (1-t)^{n-i} t^i$$

Bentuk tersebut biasa dikenal dengan nama polinomial Bernstein. Kurva yang dihasilkan akan melalui titik pertama dan terakhir dari *control points*. Namun tidak menutup kemungkinan titik lain akan dilewati. Implementasi dalam algoritma *brute force* mengikuti langkah-langkah berikut:

1. Bangkitkan Segitiga Pascal yang merupakan identitas kombinatorial dari koefisien binomial (Hal ini dilakukan untuk menghindari *integer overflow* ketika mencari kurva dengan *control points* yang banyak)
2. Iterasi nilai t dari 0 sampai 1, dalam iterasi lakukan:
 - a. Jumlahkan setiap titik yang telah dikalikan dengan polinomial Bernstein
 - b. Simpan titik hasil penjumlahan pada senarai penampung
3. Gambarkan kurva Bézier ke layar

BAB 2

Strategi Algoritma dengan *Divide and Conquer*

Algoritma *divide and conquer* merupakan sebuah strategi pemecahan masalah dengan membaginya menjadi permasalahan identikal yang lebih kecil (*divide*), kemudian mencari hasil dari masing-masing masalah, dan menggabungkannya (*conquer*). Pembuatan kurva Bézier dapat dilakukan dengan cara *divide and conquer* ini. Pembagian masalah dilakukan dengan menggunakan *midpoint* dari hasil yang sudah didapat. Hal ini dilakukan sampai kedalaman pembagian telah mencapai titik tertentu. Pada saat pengimplementasian, langsung digunakan cara yang telah digeneralisasi untuk n titik untuk membuat kurva berderajat $n - 1$. Langkah-langkah yang digunakan sebagai berikut:

1. Program akan menerima poin sebanyak n titik sebagai *control points*
2. Pada iterasi pertama, kita mencari sebuah titik *midpoint* pertama yang akan didapatkan
3. Dilakukan sub-iterasi dengan cara sebagai berikut:
 - a. Dengan n titik, dapat dibuat garis menyambung yang terdiri dari $n - 1$ segmen
 - b. Pada setiap segmen, kita mencari titik tengah dan menyimpannya. Sehingga kita mendapatkan $n - 1$ titik
 - c. Selain itu, kita juga menyimpan titik pertama dan terakhir pada poin b secara terpisah untuk digunakan pada iterasi selanjutnya
 - d. Dari $n - 1$, didapatkan $n - 2$ segmen dan diambil titik tengah dari setiap segmen sehingga menjadi $n - 2$
 - e. Hal ini terus dilakukan, yaitu mengulang langkah b hingga d sampai tersisa hanya 1 titik
 - f. Titik terakhir yang didapat pada poin e yang akan digunakan untuk menggambar kurva Bézier
4. Setelah sub-iterasi selesai, kita menggunakan kumpulan titik pertama (dinamakan *left*) beserta kumpulan titik terakhir (dinamakan *right*). Kedalaman dari pembuatan titik ditambah satu, dan dilakukan hal yang sama pada *left* dan *right* dengan mengandaikan mereka menjadi *control points* baru.

Detail implementasi menggunakan struktur data *tree* yang memiliki sifat *lazy*. Sifat ini memiliki arti bahwa nilai dari data tidak akan dikomputasi sampai benar-benar dibutuhkan. Setelah melakukan komputasi, maka nilai ini dijadikan sebagai *cache* agar dapat digunakan kembali. Struktur data *tree* didefinisikan sebagai berikut:

```
LazyPath = [LazyPoint, Point, LazyPoint]
LazyPoint = {
    control: Point[],
    result: LazyPath
}
```

Nilai dari *control points* dapat dituliskan dengan tipe *LazyPoint*. Hal ini berlaku pada penerapan iterasi kedalaman pertama. Pada iterasi selanjutnya, yang disebutkan pada poin diatas, kita menggunakan tipe data yang sama untuk melakukan komputasi.

BAB 3

Source Code Program

1. *Brute Force*

```
function bezierBruteForce(controlPoints: Point[], iteration: number) {
    const length = controlPoints.length
    const coefficient = []
    for (let i = 0; i < length; ++i) {
        const tmp = [...coefficient]
        for (let j = 1; j < i; ++j) {
            coefficient[j] = tmp[j - 1] + tmp[j]
        }
        coefficient.push(1)
    }

    const degree = length - 1
    const points: Point[] = []
    points.push(controlPoints[0])
    for (let i = 1; i <= iteration; ++i) {
        const t = i / (iteration + 1)
        let point = new Point(0, 0)
        for (let j = 0; j < coefficient.length; ++j) {
            const scale = coefficient[j] * (t ** j) * ((1 - t) ** (degree - j))
            point = Point.add(point, Point.scale(controlPoints[j], scale));
        }
        points.push(point)
    }
    points.push(controlPoints[degree])
    return points
}
```

2. *Divide and Conquer*

```
export class BezierDnC {
    lazyPoint: LazyPoint;
    point: Point[]
    constructor(point: Point[]) {
        this.point = point;
        this.lazyPoint = new LazyPoint(this.point);
    }

    static traverse(lazy: LazyPoint, depth: number, accumulator: Point[]) {
        if (depth === 0) return;
        const [left, mid, right] = lazy.get();
        BezierDnC.traverse(left, depth - 1, accumulator);
        accumulator.push(mid);
        BezierDnC.traverse(right, depth - 1, accumulator);
    }

    generate(iteration: number) {
        const accumulator: Point[] = [];
        accumulator.push(this.point[0]);
        BezierDnC.traverse(this.lazyPoint, iteration, accumulator);
        accumulator.push(this.point[this.point.length - 1]);
        return accumulator;
    }
}
```

```
type LazyPath = [LazyPoint, Point, LazyPoint];
export class LazyPoint {
    private computed: boolean;
    control: Point[];
    result: LazyPath;

    constructor(control: Point[]) {
        this.computed = false;
        this.control = control;
    }

    intermediatePoint: Point[][] = [];
    getIntermediatePoint() {
        if (!this.computed) this.get();
        return this.intermediatePoint;
    }

    get(): LazyPath {
        if (!this.computed) return this.result;
        const count = this.control.length;

        let left: Point[] = [];
        let right: Point[] = [];
        let current: Point[] = [...this.control];
        let next: Point[] = [];

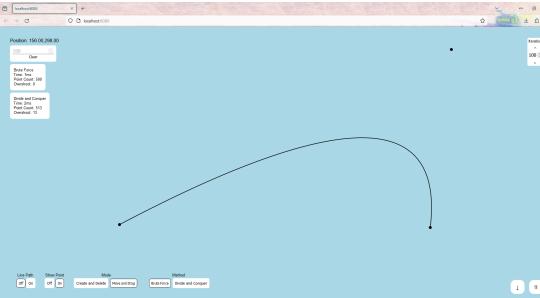
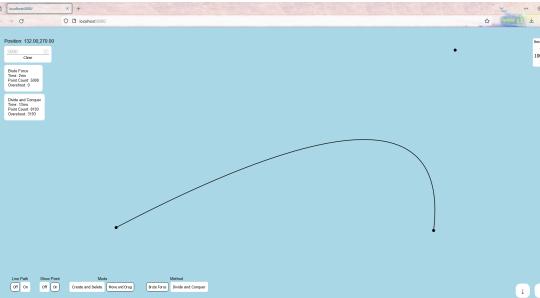
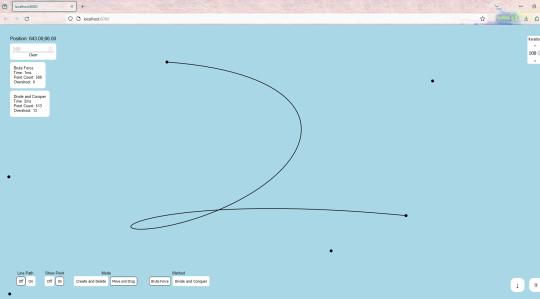
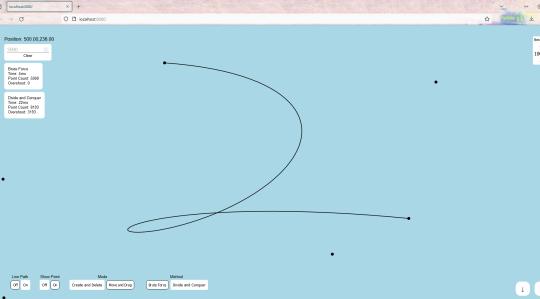
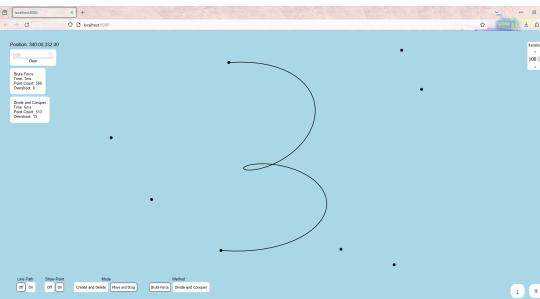
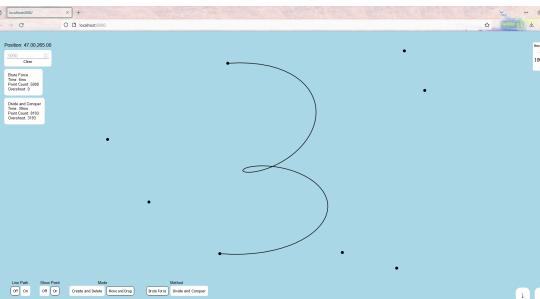
        left.push(current[0]);
        right.push(current[count - 1]);
        for (let i = count - 1; i > 0; --i) {
            for (let j = 0; j < i; ++j) {
                const mid = Point.midPoint(current[j], current[j + 1]);
                if (j == 0) left.push(mid);
                if (j == i - 1) right.push(mid);
                next.push(mid);
            }
            this.intermediatePoint.push(current);
            current = [...next];
            next = [];
        }
        right = right.reverse();
        this.intermediatePoint.push([current[0]]);

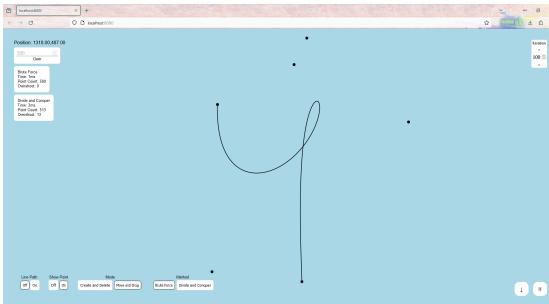
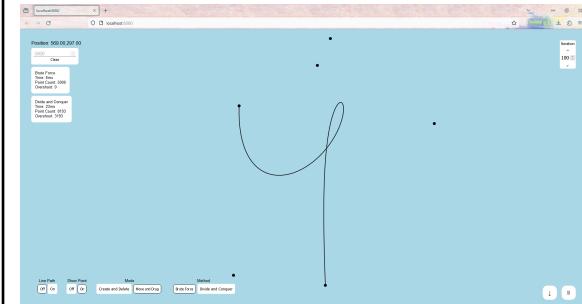
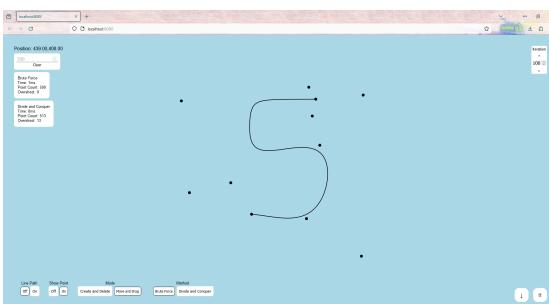
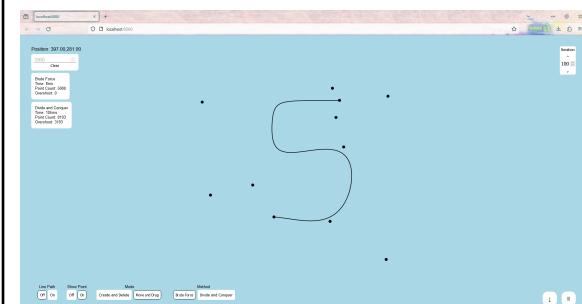
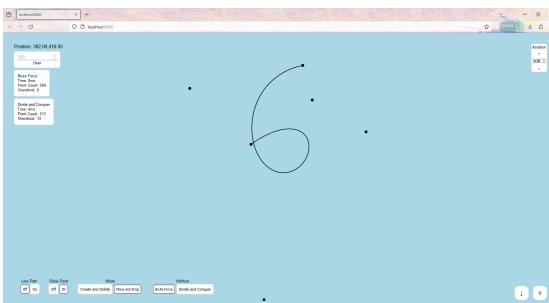
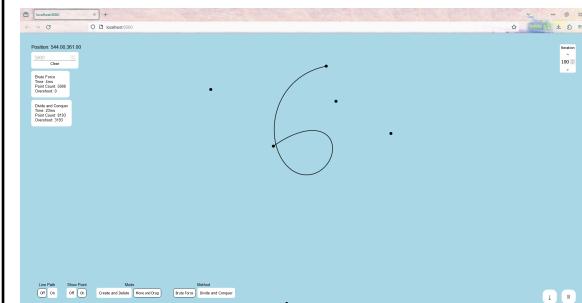
        this.computed = true;
        this.result = [new LazyPoint(left), current[0], new LazyPoint(right)];
        return this.result;
    }
}
```

BAB 4

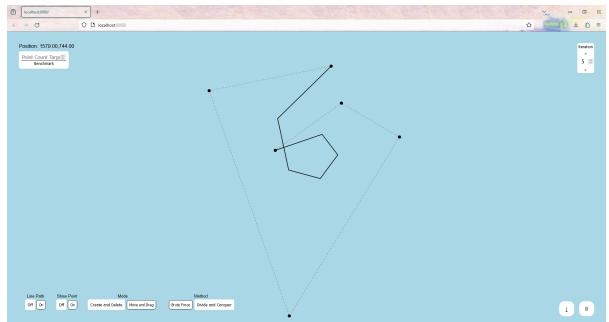
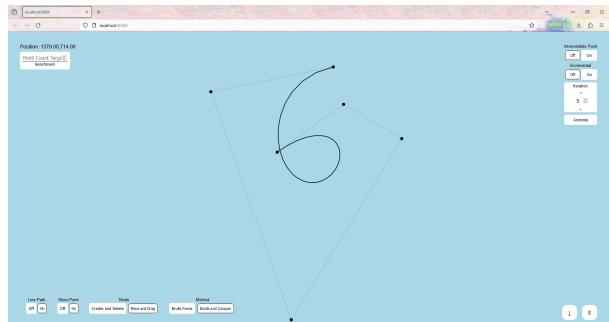
Eksperimen

Tabel 4.1. Eksperimen Berdasarkan Banyak Titik yang Dibuat

No.	Hasil (500 titik yang harus dibuat oleh masing-masing algoritma)	Hasil (5000 titik yang harus dibuat oleh masing-masing algoritma)
1.	 <p><i>Brute force:</i> 1ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 2ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 2ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 13ms, 8193 points, 3193 overshoot</p>
2.	 <p><i>Brute force:</i> 1ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 2ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 4ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 22ms, 8193 points, 3193 overshoot</p>
3.	 <p><i>Brute force:</i> 1ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 6ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 6ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 39ms, 8193 points, 3193 overshoot</p>

4.	 <p><i>Brute force:</i> 1ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 3ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 6ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 23ms, 8193 points, 3193 overshoot</p>
5.	 <p><i>Brute force:</i> 1ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 8ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 8ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 105ms, 8193 points, 3193 overshoot</p>
6.	 <p><i>Brute force:</i> 0ms, 500 points, 0 overshoot <i>Divide and conquer:</i> 4ms, 513 points, 13 overshoot</p>	 <p><i>Brute force:</i> 4ms, 5000 points, 0 overshoot <i>Divide and conquer:</i> 23ms, 8193 points, 3193 overshoot</p>

Tabel 4.2. Perbandingan Hasil Visualisasi Kurva Bézier dengan Lima Iterasi

<i>Brute Force</i>	<i>Divide and Conquer</i>
	

BAB 5

Hasil Analisis

Perhitungan kompleksitas waktu membutuhkan nilai berupa banyaknya input yang dimasukan. Pada kasus ini, kita menuliskan banyaknya *control points* adalah n . Membandingkan dua algoritma berdasarkan iterasi bukanlah hal yang empiris, karena definisi iterasi pada setiap algoritma adalah berbeda. Perbedaan ini terlihat jelas pada Tabel 4.2. yang membandingkan visualisasi dari hasil lima iterasi yang dilakukan oleh masing-masing algoritma. Oleh karena itu kita menggunakan nilai m yang merupakan titik yang setidaknya harus dibuat oleh setiap strategi algoritma, seperti hasil eksperimen pada Tabel 4.1. dengan 500 dan 5000 titik. Berikut merupakan pembahasan dari kompleksitas masing-masing strategi:

1. Brute Force

Pada langkah pertama, dibuat baris Segitiga Pascal ke- n . Bagian ini terdiri dari dua loop, yang apabila dijumlahkan akan menghasilkan kompleksitas algoritma sebesar $T(n(n + 1)/2)$. Kemudian pada tahap selanjutnya kita mencari seluruh titik yang dibutuhkan, yaitu sebesar m . Pada setiap titiknya, dilakukan penjumlahan sebanyak n suku. Sehingga kompleksitas pada tahap ini adalah $T(nm)$. Sehingga algoritma secara keseluruhan memiliki kompleksitas $T(n(n + 1)/2 + nm)$. Dikarenakan banyaknya titik yang dihasilkan sering lebih banyak dibandingkan dengan banyaknya control points, sehingga kita bisa menuliskan Big O sebesar $O(mn)$.

2. Divide and Conquer

Pada setiap iterasi terdapat tahap sub-iterasi yang mereduksi n titik menjadi 1 titik. Kompleksitas pada tahap ini sebesar $T(n(n + 1)/2)$. Dikarenakan titik yang setidaknya harus dihasilkan sebanyak m , maka program harus melakukan sub-iterasi sebanyak m dengan menggunakan strategi *divide and conquer*. Kemudian terdapat relasi rekurens untuk kompleksitas waktu yang digunakan sebagai berikut:

$$T(1) = 1$$

$$T(m) = 2T(m/2) + [n(n + 1)/2] \approx 2T(m/2) + n^2$$

Kita tidak bisa menggunakan teorema master, dikarenakan nilai m dan n dapat berbeda jauh. Kita asumsikan nilai dari $m = 2^p$. Lalu dilakukan ekspansi sebagai berikut:

$$\begin{aligned} T(2^p) &\approx 2T(2^{p-1}) + n^2 \\ &= 2(2T(2^{p-2}) + n^2) + n^2 \\ &= 4T(2^{p-2}) + 2n^2 + n^2 \\ &\dots \\ &= 2^p T(1) + 2^{p-1} n^2 + 2^{p-2} n^2 + \dots + n^2 \\ &= 2^p T(1) + 2^p n^2 - n^2 \end{aligned}$$

Karena nilai dari $2^p T(1)$ adalah konstan dan n^2 relatif kecil, kita bisa abaikan. Selain itu, karena $m = 2^p$, nilai dari kompleksitas waktu algoritma ini adalah $O(mn^2)$.

Berdasarkan analisis yang dilakukan, beserta hasil eksperimen yang didapat. Terlihat bahwa penggunaan strategi *divide and conquer* tidak menghasilkan algoritma yang lebih sangkil. Salah satu kemungkinan dari penyebab hal ini adalah adanya tahap pembuatan *intermediate point* pada strategi *divide and conquer*. Sedangkan pada strategi *brute force*, algoritma langsung menghasilkan titik akhir dari fungsi polinomial yang dibuat.

BAB 6

Penjelasan Bonus

Program dibuat menggunakan bahasa pemrograman *typescript* yang nantinya dikompilasi menjadi *javascript*. Teknologi yang digunakan untuk menggambar kurva adalah HTMLCanvas. Berikut merupakan beberapa fitur bonus yang kami kembangkan:

1. *Realtime Curve Editing*

Titik *control points* dapat diubah secara langsung tanpa mengulang jalannya program. Hal ini termasuk mengubah banyaknya iterasi dalam pembentukan kurva.

2. *Intermediate Point Visualization*

Seluruh titik yang tidak menjadi bagian pada kurva, namun dibuat sebagai titik perantara dapat digambarkan pada tampilan layar

3. *Incremental Generation*

Manfaatkan struktur data dan sifat *lazy* dari implementasi, kita bisa memberikan sebuah target jarak maksimal antara dua buah titik yang dihasilkan pada kurva. Hal ini dilakukan dengan melakukan traversal, sampai terdapat dua titik yang memiliki jarak kurang dari nilai tertentu.

4. Animation

Pada strategi *divide and conquer*, fitur animasi menambahkan *in between frame* yang menggambarkan perubahan kurva secara halus seiring bertambahnya titik yang dihasilkan.

5. *Infinite Canvas Size*

Tampilan dapat dilakukan perbesaran dan pergeseran, sehingga tampilan visualisasi yang dibuat seakan-akan memiliki ukuran tak-hingga.

Kemudian untuk titik koordinat yang dibuat dapat didapatkan dengan memencet tombol panah ke bawah (*download*) pada bagian bawah kanan tampilan. Selain itu, tombol kanan pada *mouse* dapat digunakan untuk mengubah *mode* operasi secara cepat. Terakhir, di sebelah kanan tombol panah ke bawah (*download*) ada tombol berbentuk seperti tong sampah yang digunakan untuk membersihkan semua titik yang telah ditambahkan ke *canvas*.

BAB 7

Pranala ke *Repository*

Repository: https://github.com/atpur-rafir/Tucil2_13522086_13522094

Rilis: https://github.com/atpur-rafir/Tucil2_13522086_13522094/releases/tag/1.1

Daftar Pustaka

1. [Spesifikasi Tugas Kecil 2 Stima 2023/2024 - Google Docs](#)
2. [CS184 Lecture 21 summary \(berkeley.edu\)](#)
3. [ch_bezier.pdf \(umd.edu\)](#)

Lampiran

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	✓	