LAPORAN TEKNIS

TUGAS KELOMPOK ANALISIS NUMERIK

Studi Penggunaan Bézier Curves dalam Pembuatan Ilustrasi Gambar yang Sudah Ada

Disusun oleh Kelompok C10:

Argya Farel Kasyara	2306152424
Alexander William Lim	2306207505
Jason Kent Winata	2206081313
Daffa Abhipraya Putra	2306245131

Fakultas Ilmu Komputer
Universitas Indonesia

2025

"Dengan ini, kami menyatakan bahwa tugas ini adalah hasil pekerjaan kelompok sendiri."

Argya Farel Kasyara

NPM: 2306152424

Alexander William Lim

NPM: 2306207505

Jason Kent Winata

NPM: 2206081313

Daffa Abhipraya Putra

NPM: 2306245131

Rangkuman

Laporan ini membahas implementasi metode numerik untuk merepresentasikan gambar raster menjadi ilustrasi vektor menggunakan kurva Bézier kubik. Pendekatan yang digunakan adalah *Least Squares Fitting* untuk mencocokkan kurva dengan kontur gambar yang diekstraksi menggunakan deteksi tepi Canny. Hasil akhir berupa berkas keluaran standar PDF yang di render dari kurva Bézier tersebut.

Daftar Isi

1. Pendahuluan

1.1 Latar Belakang

Kurva Bézier merupakan salah satu primitif grafis terpenting dalam *Computer Assisted Design* (CAD) dan grafis vektor. Kemampuannya untuk memodelkan bentuk lengkung yang halus dengan hanya beberapa titik kontrol menjadikannya sangat efisien dibandingkan dengan representasi poligon atau raster. Dalam proyek ini, kami mengeksplorasi bagaimana mengonversi citra raster statis menjadi representasi matematis menggunakan kurva Bézier, sebuah proses yang dikenal sebagai *vectorization* atau *image tracing*.

1.2 Rumusan Masalah

Permasalahan utama yang dikaji adalah:

1. Bagaimana mengekstraksi titik-titik data yang merepresentasikan bentuk dari sebuah citra raster?
2. Bagaimana mencocokkan (*fit*) kurva Bézier kubik pada sekumpulan titik data tersebut dengan galat yang minimal?
3. Bagaimana cara menghasilkan berkas keluaran standar (PDF) yang dapat memvisualisasikan hasil kurva tersebut?

1.3 Tujuan Penulisan

1. Bagaimana mengekstraksi titik-titik data yang merepresentasikan bentuk dari sebuah citra raster?
2. Bagaimana mencocokkan (*fit*) kurva Bézier kubik pada sekumpulan titik data tersebut dengan galat yang minimal?
3. Bagaimana menghasilkan berkas keluaran standar (PDF) yang dapat memvisualisasikan hasil kurva tersebut?

1.4 Batasan Masalah

- Kurva yang digunakan adalah Bézier kubik (derajat 3).
- Masukan berupa citra raster (PNG/JPG).
- Keluaran berupa berkas PDF.

2. Dasar Teori

2.1 Dasar Teori

Kurva Bézier derajat n didefinisikan oleh persamaan parametrik menggunakan polinomial Bernstein:

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

Untuk kasus Cubic Bézier ($n = 3$), persamaannya adalah:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

Representasi ini menjamin bahwa kurva dimulai di P_0 (saat $t = 0$) dan berakhir di P_3 (saat $t = 1$). Garis singgung di P_0 searah dengan vektor $\vec{P_0 P_1}$, dan di P_3 searah dengan $\vec{P_2 P_3}$.

2.2 Least Square Fitting

Untuk mencocokkan kurva Bézier pada sekumpulan titik data $D_0, D_1, \ldots, D_k$, kita perlu mencari titik kontrol P_1 dan P_2 (dengan P_0 dan P_3 tetap di ujung segmen) yang meminimalkan jumlah kuadrat galat:

$$E = \sum_{j=0}^{k} |B(t_j) - D_j|^2$$

Masalah ini dapat diselesaikan dengan menurunkan fungsi E terhadap komponen P_1 dan P_2, yang menghasilkan sistem persamaan linear.

3. Metodologi Eksperimen

3.1 Pengolahan Citra (Image Processing)

Kami menggunakan pustaka `cv2` (OpenCV) untuk tahap pra-pemrosesan. Awalnya, kami menggunakan *binary thresholding*, namun metode ini kurang *robust* terhadap gambar dengan kontras rendah atau latar belakang putih. Oleh karena itu, kami beralih menggunakan *Canny Edge Detection*. Metode Canny mendeteksi tepi berdasarkan gradien intensitas, sehingga mampu menangkap kontur logo dengan lebih akurat terlepas dari warna latar belakangnya.

3.2 Fitting Kurva Rekursif

Algoritma utama kami (`curve_fitter.py`) bekerja secara rekursif:

1. Mencoba mencocokkan satu kurva Bézier pada sekumpulan titik kontur.
2. Menghitung galat maksimum antara kurva hasil fitting dengan titik asli.
3. Jika galat melebihi ambang batas (*threshold*), segmen titik dibagi dua pada titik dengan galat terbesar, dan proses diulang untuk kedua bagian tersebut.

Pseudocode:

```
Algorithm: Recursive Curve Fitting
Input: List of points, Error threshold
Output: List of Bezier curves

1. Check Base Case:
   If the number of points is less than 2:
       Return an empty list (cannot fit a curve).

2. Initial Fit:
   Attempt to fit a single cubic Bezier curve to the current set of
   points using Least Squares.

3. Error Calculation:
   Calculate the maximum distance (error) between the fitted curve and
   the original points.
   Identify the point index where this maximum error occurs.

4. Decision:
   If the maximum error is within the threshold:
       Return the single fitted curve.
   Else (error is too high):
       Split the points into two subsets at the index of maximum error:
           Left Subset: Points from start to split index.
           Right Subset: Points from split index to end.
       Recursively call this algorithm for the Left Subset.
       Recursively call this algorithm for the Right Subset.
```

```
    Return the combined results (Left Curves + Right Curves).
```

3.3 Pembuatan PDF

Kami mengembangkan generator PDF (`pdf_generator.py`) yang meng-render kurva Bézier ke dalam citra raster menggunakan pustaka OpenCV. Hal ini dilakukan untuk mempermudah visualisasi hasil langsung. Kami menggunakan fungsi `cv2.polylines` dengan antialiasing untuk memastikan kualitas garis yang halus.

Pseudocode:

```
Algorithm: Generate PDF
Input: List of Bezier curves, Output filename

1. Initialize PDF Structure:
   Create a list to hold PDF content lines.
   Write the PDF Header ("%PDF-1.4").

2. Create PDF Objects:
   Create the Catalog Object (referencing the Pages object).
   Create the Pages Object (referencing the Page object).
   Create the Page Object (defining media box and referencing content
 ↪  stream).

3. Construct Content Stream:
   Initialize an empty string for the stream data.
   For each curve in the list:
       Get the start point (P0).
       Append "Move To" command (m) for P0 to the stream.
       Get control points (P1, P2, P3).
       Append "Curve To" command (c) using P1, P2, P3 to the stream.
       Append "Stroke" command (S) to draw the path.

4. Finalize PDF:
   Write the Content Stream Object (containing the stream data).
   Write the Cross-Reference Table (xref) calculating byte offsets for
 ↪  each object.
   Write the Trailer (pointing to the Root object).
   Write the End-of-File marker ("%%EOF").

5. Output:
   Write all PDF content to the specified filename.
```

4. Hasil dan Analisis

I. Masukan dan Keluaran Bézier Curves

- **Masukan:** Sekumpulan titik kontrol P_0, P_1, P_2, P_3 dan parameter $t \in [0, 1]$.
- **Keluaran:** Koordinat titik (x, y) pada kurva.

II. Representasi Matematis

Seperti dijelaskan pada Dasar Teori, kurva direpresentasikan oleh kombinasi linear dari basis Bernstein. Sifat *convex hull* menjamin kurva selalu berada dalam poligon yang dibentuk oleh titik-titik kontrolnya.

III. Implementasi Program "Freehand Draw"

Program 3.7 pada buku Timothy Sauer menggunakan pendekatan interpolasi. Implementasi kami memperluas konsep ini dengan menggunakan fitting. Dimana Sauer menggunakan interpolasi spline yang melewati setiap titik, kami menggunakan aproksimasi *Least Squares* yang lebih efisien untuk data yang rapat (seperti piksel kontur), sehingga kami dapat menghasilkan kurva yang halus tanpa osilasi berlebih.

IV. Contoh Perhitungan Analitik

Misalkan $P_0 = (0, 0)$, $P_1 = (2, 5)$, $P_2 = (8, 5)$, $P_3 = (10, 0)$. Untuk $t = 0.5$:

$$B(0.5) = 0.125(0, 0) + 0.375(2, 5) + 0.375(8, 5) + 0.125(10, 0)$$

$x = 0.75 + 3 + 1.25 = 5$ $y = 1.875 + 1.875 = 3.75$

Hasil: $(5, 3.75)$.

V. Perbandingan Kualitas, Akurasi, Efisiensi Keluaran

A. Gambar Non Gradien

Kami melakukan percobaan pada dua gambar logo berwarna polos yang berbeda: Makara UI (kompleks) dan Logo Superbank (sederhana). Berikut adalah masukan, keluaran, dan data hasil pemrosesan:

Input	Output
[IMAGE: Logo Makara UI Kuning Original]	[IMAGE: Logo Makara UI Outline/Vector Result]

Input	Output
[IMAGE: Logo Superbank Hijau Original]	[IMAGE: Logo Superbank Outline/Vector Result]

Gambar 4.1 Perbandingan Masukkan dan Keluaran Makara UI Gambar 4.2 Perbandingan Masukkan dan Keluaran Logo Superbank

Tabel 4.1 Perbandingan Spesifikasi Makara UI dan Logo Superbank

Metrik	Makara UI (makara.png)	Logo Superbank (superbank.png)
Dimensi Gambar	1890×2065 piksel	512×512 piksel
Total Kontur	46	3
Total Titik Asli	74,615 titik	2,546 titik
Total Kurva Bézier	797 kurva	37 kurva
Rasio Kompresi	1 kurva per 93.62 titik	1 kurva per 68.81 titik

Analisis:

- **Efisiensi Representasi:** Pada logo Makara yang sangat kompleks dengan lebih dari 74 ribu titik kontur, algoritma kami mampu merepresentasikan sebagian besar gambar hanya dengan 797 kurva Bézier. Ini menunjukkan rasio kompresi yang sangat tinggi (1:93), di mana satu kurva halus dapat menggantikan hampir 100 titik diskrit.
- **Kompleksitas vs. Jumlah Kurva:** Jumlah kurva yang dihasilkan berbanding lurus dengan kompleksitas visual dan jumlah kontur objek. Logo Superbank yang lebih geometris dan sederhana membutuhkan jauh lebih sedikit kurva dibandingkan logo Makara yang memiliki banyak lekukan detail.
- **Kualitas Visual:** Secara keseluruhan, hasil visualisasi PDF menunjukkan bahwa kurva yang dihasilkan mampu mengaproksimasi kontur asli dengan sangat halus, memvalidasi efektivitas metode *Least Squares Fitting* yang diterapkan. Hal ini terbukti pada rekonstruksi logo Superbank yang memiliki tingkat kesesuaian tinggi dengan citra aslinya. Namun, keterbatasan algoritma terlihat pada geometri yang lebih kompleks seperti logo Makara UI, di mana terdapat detail yang gagal terdeteksi sehingga menghasilkan rekonstruksi gambar yang tidak utuh.

B. Gambar dengan Gradien

Selanjutnya, kami juga melakukan percobaan pada dua gambar logo berwarna gradien yang berbeda: Firefox (kompleks) dan Messenger (sederhana). Berikut adalah masukan, keluaran, dan data hasil pemrosesan:

Input	Output
[IMAGE: Logo Firefox Original]	[IMAGE: Logo Firefox Outline/Vector Result]

Input	Output
[IMAGE: Logo Messenger Original]	[IMAGE: Logo Messenger Outline/Vector Result]

Gambar 4.4 Perbandingan Masukkan dan Keluaran Logo Firefox Gambar 4.5 Perbandingan Masukkan dan Keluaran Logo Messenger App

Tabel 4.2 Perbandingan Spesifikasi Logo Firefox dan Logo Messenger

Metrik	Logo Firefox (firefox.png)	Logo Messenger (messenger.png)
Dimensi Gambar	1200×1200 piksel	979×980 piksel
Total Kontur	7	4
Total Titik Asli	15,370 titik	5,132 titik
Total Kurva Bézier	88 kurva	55 kurva
Rasio Kompresi	1 kurva per 174.66 titik	1 kurva per 93.31 titik

Analisis:

- **Efisiensi Representasi:** Efisiensi kompresi yang signifikan tercapai pada pengujian logo Firefox. Dari total lebih dari 15.000 titik kontur, algoritma mampu merekonstruksi citra hanya dengan 88 kurva Bézier. Hal ini mengindikasikan bahwa variasi gradien warna memiliki dampak minimal terhadap efisiensi, di mana satu kurva Bézier rata-rata merepresentasikan sekitar 175 titik kontur.
- **Kompleksitas vs. Jumlah Kurva:** Jumlah kurva yang dihasilkan berbanding lurus dengan kompleksitas geometri objek. Hal ini terlihat dari perbandingan antara logo Messenger yang bersifat geometris sederhana dan membutuhkan sedikit kurva, dengan logo Firefox yang memiliki detail lekukan tinggi sehingga membutuhkan jumlah kurva yang lebih banyak.
- **Kualitas Visual:** Secara umum, visualisasi PDF mampu mengaproksimasi gambar asli dengan akurasi tinggi. Namun, keterbatasan terlihat pada logo Firefox, di mana kombinasi gradien warna dan geometri kompleks menyebabkan kegagalan deteksi pada sebagian titik kontur. Akibatnya, hasil rekonstruksi tampak tidak utuh, menandakan perlunya peningkatan algoritma dalam menangani citra dengan kompleksitas warna dan bentuk yang tinggi.

C. Kurva yang Berbeda

Kami juga melakukan perbandingan kualitas gambar yang dihasilkan dengan menggunakan banyak kurva yang berbeda pada gambar yang sama. Berikut adalah masukkan, keluaran, dan hasil pemrosesan:

Gambar 4.6 Perbandingan Keluaran Makara Terhadap Banyak Kurva

Output (100 Kurva)	Output (500 Kurva)	Output (800 Kurva)
[IMAGE: Hasil Makara 100 Kurva]	[IMAGE: Hasil Makara 500 Kurva]	[IMAGE: Hasil Makara 800 Kurva]

Tabel 4.2 Perbandingan Spesifikasi Perbedaan Banyak Kurva

Metrik	100 Kurva	500 Kurva	800 Kurva
Dimensi Gambar	1890×2065 piksel	1890×2065 piksel	1890×2065 piksel
Total Kontur	46	46	46
Total Titik Asli	74,615 titik	74,615 titik	74,615 titik
Rasio Kompresi	1 kurva per 746.15 titik	1 kurva per 149.23 titik	1 kurva per 93.27 titik

Analisis:

- **Rasio Kompresi vs. Total Kurva:** Terdapat korelasi berbanding terbalik antara jumlah kurva Bézier yang digunakan dengan rasio kompresi titik. Pada konfigurasi 100 kurva, algoritma mencapai efisiensi representasi tertinggi, di mana satu kurva Bézier mampu mengakomodasi rata-rata 746,15 titik data asli. Sebaliknya, ketika jumlah kurva ditingkatkan menjadi 500 dan 800, densitas representasi menurun drastis menjadi masing-masing 149,23 dan 93,27 titik per kurva. Hal ini menunjukkan bahwa penambahan jumlah kurva secara signifikan mengurangi efisiensi penyimpanan data per segmennya.
- **Kualitas Visual:** Terdapat hubungan berbanding lurus antara jumlah kurva Bézier dengan tingkat akurasi visual yang dihasilkan, namun memiliki batas optimal. Pada visualisasi dengan 100 kurva, algoritma cenderung melakukan generalisasi bentuk (*smoothing*), di mana detail-detail kecil pada ornamen logo Makara terabaikan demi mencapai rasio kompresi tinggi. Peningkatan ke 500 kurva memperbaiki detail tersebut secara signifikan. Namun, pada penggunaan 800 kurva, meskipun alokasi titik per kurva lebih sedikit (93,27 titik), fleksibilitas yang lebih tinggi ini tidak memberikan perbedaan visual yang signifikan dibandingkan hasil 500 kurva, mengindikasikan terjadinya saturasi kualitas. Hasil eksperimen membuktikan bahwa kualitas visual yang setara dapat dipertahankan dengan jumlah kurva Bézier yang lebih sedikit. Hal ini terkonfirmasi dari hasil rekonstruksi menggunakan 800 kurva yang tidak menunjukkan perbedaan visual signifikan dibandingkan dengan penggunaan 500 kurva. Temuan ini mengindikasikan bahwa kualitas gambar mencapai titik jenuh (*saturation point*), di mana penambahan kurva melampaui batas optimal tidak lagi memberikan peningkatan kualitas yang nyata.

VI. Langkah Pembuatan PDF

Program menulis header `%PDF-1.4`, mendefinisikan objek Catalog dan Pages, lalu menulis stream yang berisi perintah-perintah grafis. Sesuai spesifikasi, kami menggunakan operator `m` untuk berpindah ke titik awal kurva dan operator `c` untuk menggambar kurva Bézier kubik.

Contoh potongan *content stream*:

```
1148.00 188.00 m
1148.47 189.61 1149.94 188.80 1151.00 189.00 c
1195.54 152.40 1241.87 120.11 1292.00 97.00 c
1293.89 95.98 1292.64 93.66 1293.00 92.00 c
1259.48 35.92 1182.23 18.27 1123.00 38.00 c
```

Ini menggambar kurva dari (1148,188) ke (1151,189) dengan titik kontrol (1148.47,189.61) dan (1149.94,188.80), dan seterusnya.

5. Kesimpulan

Proyek ini berhasil mengimplementasikan sistem vektorisasi citra menggunakan kurva Bézier. Penggunaan metode *Least Squares Fitting* dengan pembagian rekursif terbukti efektif untuk menyeimbangkan antara akurasi (galat rendah) dan efisiensi (jumlah kurva minimal). Implementasi *Canny Edge Detection* meningkatkan ketahanan sistem terhadap variasi input gambar.

Daftar Referensi

Sauer, T. (2012). *Numerical Analysis* (2nd ed.). Pearson.

Lampiran

Complete Source Code: https://github.com/absolutepraya/bezier-curves-python

src/main.py

```
import os
from image_processor import get_contours
from curve_fitter import fit_curve_recursive
from pdf_generator import generate_pdf_from_curves

def main():
    input_dir = "input"
    output_dir = "output"

    # Ensure output directory exists
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # 1. List images in input directory
    if not os.path.exists(input_dir):
        print(f"Error: Directory '{input_dir}' not found.")
        return

    files = [f for f in os.listdir(input_dir) if
↪   f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp'))]

    if not files:
        print(f"No image files found in '{input_dir}'.")
        return

    print("Available images:")
    for i, f in enumerate(files):
        print(f"{i+1}. {f}")

    # 2. User selection
    while True:
        try:
            choice = input("\nSelect an image number to process: ")
            idx = int(choice) - 1
            if 0 <= idx < len(files):
                filename = files[idx]
                break
            else:
                print("Invalid selection. Please try again.")
        except ValueError:
            print("Please enter a number.")

    input_path = os.path.join(input_dir, filename)
```

```
    # 3. Construct output filename
    name_without_ext = os.path.splitext(filename)[0]
    print(f"\nProcessing {input_path}...")

    try:
        # Get contours
        contours, (height, width, _) = get_contours(input_path)
        all_curves = []
        total_curves_count = 0
        total_points_count = 0

        # Fit curves
        for i, contour in enumerate(contours):
            total_points_count += len(contour)
            curves = fit_curve_recursive(contour, error_threshold=2.0)
            all_curves.append(curves)
            total_curves_count += len(curves)

        # Data Logging
        print("-" * 40)
        print(f"DATA LOGGING SUMMARY")
        print("-" * 40)
        print(f"Image Dimensions : {width}x{height}")
        print(f"Total Contours Found : {len(contours)}")
        print(f"Total Original Points : {total_points_count}")

        if total_curves_count > 0:
            print(f"Total Bezier Curves : {total_curves_count}")
            ratio = total_points_count / total_curves_count
            print(f"Compression Ratio : 1 curve per {ratio:.2f} points")
        print("-" * 40)

        # Generate PDF
        pdf_output_filename = f"{name_without_ext}-output.pdf"
        pdf_output_path = os.path.join(output_dir, pdf_output_filename)
        print(f"Generating PDF: {pdf_output_path}")

        stream_snippet = generate_pdf_from_curves(all_curves,
↪   pdf_output_path, width, height)

        print(f"Content Stream Snippet (first 5 lines):")
        for line in stream_snippet:
            print(f" {line}")
        print("-" * 40)
        print("Done!")

    except Exception as e:
        print(f"Error: {e}")
```

```
if __name__ == "__main__":
    main()
```

src/bezier_math.py

```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Point(self.x - other.x, self.y - other.y)

    def __mul__(self, scalar):
        return Point(self.x * scalar, self.y * scalar)

    def __rmul__(self, scalar):
        return self.__mul__(scalar)

    def __truediv__(self, scalar):
        return Point(self.x / scalar, self.y / scalar)

    def __repr__(self):
        return f"Point({self.x:.2f}, {self.y:.2f})"

    def dist(self, other):
        return math.sqrt((self.x - other.x)**2 + (self.y - other.y)**2)

def bernstein_poly(n, i, t):
    # Calculates the Bernstein polynomial B_{i,n}(t).
    # B_{i,n}(t) = C(n,i) * t^i * (1-t)^(n-i)
    if i < 0 or i > n:
        return 0

    binomial_coeff = math.factorial(n) / (math.factorial(i) *
↪   math.factorial(n - i))
    return binomial_coeff * (t**i) * ((1-t)**(n-i))

def cubic_bezier(t, p0, p1, p2, p3):
    # Evaluates a cubic Bezier curve at parameter t.
    # Returns a Point.
    # B(t) = (1-t)^3 P0 + 3(1-t)^2 t P1 + 3(1-t) t^2 P2 + t^3 P3
```

```
    b0 = bernstein_poly(3, 0, t)
    b1 = bernstein_poly(3, 1, t)
    b2 = bernstein_poly(3, 2, t)
    b3 = bernstein_poly(3, 3, t)

    return p0*b0 + p1*b1 + p2*b2 + p3*b3

def evaluate_curve_length(p0, p1, p2, p3, steps=100):
    # Estimates the length of the cubic Bezier curve using numerical
    ↪ integration
    # (chord summation).
    length = 0.0
    prev_point = p0
    for i in range(1, steps + 1):
        t = i / steps
        curr_point = cubic_bezier(t, p0, p1, p2, p3)
        length += prev_point.dist(curr_point)
        prev_point = curr_point
    return length
```

src/curve_fitter.py

```
import math
from bezier_math import Point, cubic_bezier, bernstein_poly

def chord_length_parameterize(points):
    # Parameterizes points based on chord length.
    # Returns a list of t values corresponding to each point.
    u = [0.0]
    for i in range(1, len(points)):
        dist = points[i].dist(points[i-1])
        u.append(u[-1] + dist)

    total_length = u[-1]
    if total_length == 0:
        return [0.0] * len(points)

    return [x / total_length for x in u]

def fit_cubic_bezier(points):
    # Fits a single cubic Bezier curve to a set of points using Least
    ↪ Squares.
    # P0 and P3 are fixed as the first and last points.
    # We need to solve for P1 and P2.

    n = len(points)
    if n < 2:
        return None # Not enough points
```

```
p0 = points[0]
p3 = points[-1]

if n == 2:
    # Straight Line, place control points at 1/3 and 2/3
    p1 = p0*(2/3) + p3*(1/3)
    p2 = p0*(1/3) + p3*(2/3)
    return [p0, p1, p2, p3]

# Parameterize points
u = chord_length_parameterize(points)

# We want to minimize sum || B(u_i) - P_i ||^2
# B(u) = (1-u)^3 P0 + 3(1-u)^2 u P1 + 3(1-u) u^2 P2 + u^3 P3
# Let A1(u) = 3(1-u)^2 u
# Let A2(u) = 3(1-u) u^2
# We want to find P1, P2 such that:
# A1(u_i) P1 + A2(u_i) P2 = Pi - (1-u_i)^3 P0 - u_i^3 P3

# Let X = [P1, P2]^T (conceptually, solving for x and y separately)
# System of equations:
# [ sum(A1^2) sum(A1A2) ] [ P1 ] = [ sum(A1*RHS) ]
# [ sum(A1A2) sum(A2^2) ] [ P2 ]   [ sum(A2*RHS) ]

c11 = 0.0
c12 = 0.0
c22 = 0.0
x1 = 0.0
x2 = 0.0
y1 = 0.0
y2 = 0.0

for i in range(n):
    t = u[i]
    a1 = 3 * (1-t)**2 * t
    a2 = 3 * (1-t) * t**2

    # RHS vector Pi - (1-t)^3 P0 - t^3 P3
    b0 = (1-t)**3
    b3 = t**3
    rhs = points[i] - (p0 * b0) - (p3 * b3)

    c11 += a1 * a1
    c12 += a1 * a2
    c22 += a2 * a2

    x1 += a1 * rhs.x
    x2 += a2 * rhs.x
    y1 += a1 * rhs.y
```

```
        y2 += a2 * rhs.y

    # Determinant of the matrix
    det = c11 * c22 - c12 * c12

    if abs(det) < 1e-9:
        # Singular matrix, fallback to straight Line approximation
        p1 = p0*(2/3) + p3*(1/3)
        p2 = p0*(1/3) + p3*(2/3)
        return [p0, p1, p2, p3]

    # Inverse matrix multiplication
    # [P1] = (1/det) [ c22 -c12 ] [X1]
    # [P2]           [ -c12 c11 ] [X2]

    p1_x = (c22 * x1 - c12 * x2) / det
    p2_x = (-c12 * x1 + c11 * x2) / det
    p1_y = (c22 * y1 - c12 * y2) / det
    p2_y = (-c12 * y1 + c11 * y2) / det

    p1 = Point(p1_x, p1_y)
    p2 = Point(p2_x, p2_y)

    return [p0, p1, p2, p3]

def calculate_max_error(points, curve):
    # Calculates the maximum distance between the points and the fitted
     ↪  curve.
    u = chord_length_parameterize(points)
    max_dist = 0.0
    for i in range(len(points)):
        p_curve = cubic_bezier(u[i], curve[0], curve[1], curve[2],
 ↪  curve[3])
        dist = points[i].dist(p_curve)
        if dist > max_dist:
            max_dist = dist
    return max_dist

def fit_curve_recursive(points, error_threshold=2.0):
    # Recursively fits Bezier curves to the points.
    # If the error is too large, splits the points and fits again.

    if len(points) < 2:
        return []

    curve = fit_cubic_bezier(points)
    if curve is None:
        return []
```

```
    max_err = calculate_max_error(points, curve)

    if max_err < error_threshold or len(points) < 4:
        return [curve]

    # Split points at the point of maximum error
    # To find the split index, we re-evaluate distances
    u = chord_length_parameterize(points)
    max_dist = 0.0
    split_idx = len(points) // 2 # Default split

    for i in range(len(points)):
        p_curve = cubic_bezier(u[i], curve[0], curve[1], curve[2],
↪   curve[3])
        dist = points[i].dist(p_curve)
        if dist > max_dist:
            max_dist = dist
            split_idx = i

    # Ensure we don't split at endpoints
    if split_idx == 0: split_idx = 1
    if split_idx == len(points) - 1: split_idx = len(points) - 2

    left_points = points[:split_idx+1]
    right_points = points[split_idx:]

    left_curves = fit_curve_recursive(left_points, error_threshold)
    right_curves = fit_curve_recursive(right_points, error_threshold)

    return left_curves + right_curves
```

src/image_processor.py

```
import cv2
import numpy as np
from bezier_math import Point

def get_contours(image_path, min_area=100):
    # Loads an image, processes it, and extracts contours.
    # Returns a list of contours, where each contour is a list of Point
↪    objects.

    # 1. Load image
    img = cv2.imread(image_path)
    if img is None:
        raise FileNotFoundError(f"Could not load image at {image_path}")

    # 2. Convert to grayscale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    # 3. Edge Detection (Canny)
    # Canny is more robust for finding edges regardless of brightness
     ↪  direction.
    # We use standard thresholds 100 and 200.
    edges = cv2.Canny(img, 100, 200)

    # 4. Find contours
    # RETR_EXTERNAL retrieves only the extreme outer contours.
    # RETR_LIST retrieves all contours.
    # CHAIN_APPROX_NONE stores all the contour points (no
     ↪  approximation).
    contours, _ = cv2.findContours(edges, cv2.RETR_LIST,
↪   cv2.CHAIN_APPROX_NONE)

    processed_contours = []
    for cnt in contours:
        # Filter small noise
        if cv2.contourArea(cnt) < min_area:
            continue

        # Convert numpy array (N, 1, 2) to List of Points
        # Note: cv2 points are (x, y), and image coordinates have y
         ↪  growing downwards.
        # We keep this coordinate system for now (PDF also has y, but
         ↪  usually growing upwards,
        # we might need to flip y later or in the PDF generator).
        points = []
        for p in cnt:
            x, y = p[0]
            points.append(Point(float(x), float(y)))
        processed_contours.append(points)

    return processed_contours, img.shape
```

src/pdf_generator.py

```
class PDFGenerator:
    def __init__(self, filename, width, height):
        self.filename = filename
        self.width = width
        self.height = height
        self.objects = []
        self.content_stream = []

    def add_object(self, content):
        obj_id = len(self.objects) + 1
        self.objects.append(content)
        return obj_id
```

```
    def add_curve(self, p0, p1, p2, p3):
        # PDF coordinates usually start from bottom-Left.
        # If our image coordinates are top-left, we might need to flip
        ↪  Y.
        # For now, we assume the points are already in the desired
        ↪  coordinate system
        # or we flip them here. Let's flip them here assuming input is
        ↪  image coords (y down).

        h = self.height

        # Move to P0
        self.content_stream.append(f"{p0.x:.2f} {h - p0.y:.2f} m")

        # Curve to P3 via P1, P2
        self.content_stream.append(f"{p1.x:.2f} {h - p1.y:.2f} {p2.x:.2f}
        ↪  {h - p2.y:.2f} {p3.x:.2f} {h - p3.y:.2f} c")

    def add_stroke(self):
        self.content_stream.append("S")

    def add_fill(self):
        self.content_stream.append("f")

    def generate(self):
        # 1. Header
        pdf_content = "%PDF-1.4\n"

        # 2. Objects
        offsets = []

        # Object 1: Catalog
        offsets.append(len(pdf_content))
        pdf_content += "1 0 obj\n<< /Type /Catalog /Pages 2 0 R
↪  >>\nendobj\n"

        # Object 2: Pages
        offsets.append(len(pdf_content))
        pdf_content += "2 0 obj\n<< /Type /Pages /Kids [3 0 R] /Count 1
↪  >>\nendobj\n"

        # Object 3: Page
        offsets.append(len(pdf_content))
        pdf_content += f"3 0 obj\n<< /Type /Page /Parent 2 0 R /MediaBox
↪  [0 0 {self.width} {self.height}] /Contents 4 0 R >>\nendobj\n"

        # Object 4: Content Stream
        stream_data = "\n".join(self.content_stream)
```

```
        stream_len = len(stream_data)
        offsets.append(len(pdf_content))
        pdf_content += f"4 0 obj\n<< /Length {stream_len}
↪   >>\nstream\n{stream_data}\nendstream\nendobj\n"

        # 3. Xref
        xref_start = len(pdf_content)
        pdf_content += "xref\n"
        pdf_content += f"0 {len(offsets) + 1}\n"
        pdf_content += "0000000000 65535 f \n"
        for offset in offsets:
            pdf_content += f"{offset:010d} 00000 n \n"

        # 4. Trailer
        pdf_content += "trailer\n"
        pdf_content += f"<< /Size {len(offsets) + 1} /Root 1 0 R >>\n"
        pdf_content += "startxref\n"
        pdf_content += f"{xref_start}\n"
        pdf_content += "%%EOF"

        with open(self.filename, 'w') as f:
            f.write(pdf_content)

def generate_pdf_from_curves(curves, output_filename, width, height):
    pdf = PDFGenerator(output_filename, width, height)

    # We can group curves that are connected, but for simplicity
    # we can just treat each curve as a sub-path.
    # Ideally, we should detect closed loops for filling.
    # The 'curves' input is a list of lists of curves (one list per
     ↪  contour).

    for contour_curves in curves:
        if not contour_curves:
            continue

        # Start path
        p0 = contour_curves[0][0]
        h = height
        pdf.content_stream.append(f"{p0.x:.2f} {h - p0.y:.2f} m")

        for curve in contour_curves:
            # curve is [p0, p1, p2, p3]
            # We assume p0 of this curve is p3 of previous, so we just
             ↪  issue 'c'
            p1, p2, p3 = curve[1], curve[2], curve[3]
            pdf.content_stream.append(f"{p1.x:.2f} {h - p1.y:.2f}
↪   {p2.x:.2f} {h - p2.y:.2f} {p3.x:.2f} {h - p3.y:.2f} c")
```

```
        # Close and stroke/fill
        # For now, Let's just stroke.
        pdf.add_stroke()

pdf.generate()

# Return a snippet of the content stream for Logging
return pdf.content_stream[:5]
```