

**PENDETEKSIAN KELILING LUCA MENGGUNAKAN
METODE BORDER FOLLOWING DENGAN BANTUAN
INTERPOLASI SPLINE**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



*Mencerdaskan dan
Memartabatkan Bangsa*

**Oleh:
Pramudio
1313619013**

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA**

2024

LEMBAR PERSETUJUAN HASIL SIDANG SKRIPSI

**PENDETEKSIAN KELILING LUCA MENGGUNAKAN METODE BORDER
FOLLOWING DENGAN BANTUAN INTERPOLASI SPLINE**

Nama : Pramudio
Nomor Registrasi : 1313619013

Penanggung Jawab

Dekan : Prof. Dr. Muktiningsih N., M.Si.
NIP. 196405111989032001

Wakil Penanggung Jawab

Wakil Dekan I : Dr. Esmar Budi, S.Si., M.T.
NIP. 197207281999031002

Ketua : Dr. Ria Arafiyah, M.Si
NIP. 197511212005012004

Sekretaris : Ir. Fariani Hermin Indiyah, MT.
NIP. 196002111987032001

Pengaji : Ari Hendarno, S.Pd., M.Kom
NIP. 198811022022031002

Pembimbing I : Muhammad Eka Suryana, M.Kom
NIP. 198512232012121002

Pembimbing II : Drs. Mulyono, M.Kom.
NIP. 196605171994031003

| Nama | Tanda Tangan | Tanggal |
|------|--------------|---------|
|------|--------------|---------|



31/07
2024

29/07
2024

24/07
2024

23/07
2024

22/07
2024

22/07
2024

22/07
2024

22/07
2024

22/07
2024

Dinyatakan lulus ujian skripsi pada tanggal : 18 Juli 2024

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul "**Pendeteksian Keliling Luka Menggunakan Metode Border Following dengan Bantuan Interpolasi Spline**" yang disusun sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Sumber informasi yang diperoleh dari peneliti lain yang telah dipublikasikan yang disebutkan dalam teks Skripsi ini, telah dicantumkan dalam Daftar Pustaka sesuai dengan norma, kaidah dan etika penulisan ilmiah.

Jika dikemudian hari ditemukan sebagian besar skripsi ini bukan hasil karya saya sendiri dalam bagian-bagian tertentu, saya bersedia menerima sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundang-undangan yang berlaku.

Jakarta, 26 Juli 2024



Pramudio

KATA PENGANTAR

Puji syukur atas kehadirat Allah Yang Maha Esa, karena atas rahmat dan karunia-Nya yang melimpah penulis berhasil menyelesaikan skripsi ini dengan baik. Adapun jenis penelitian dengan judul *Pendeteksian Keliling Luka Menggunakan Metode Border Following dengan Bantuan Interpolasi Spline*.

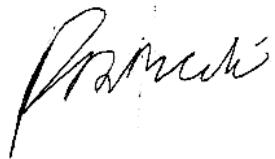
Dalam menyelesaikan skripsi ini, penulis selalu mendapat bantuan dari orang di sekitar penulis baik dalam bentuk bimbingan dalam mengerjakan skripsi ini maupun dorongan semangat dalam penggerjaan. Oleh maka dari itu, penulis ingin menyampaikan terima kasih yang sebesar-besarnya kepada:

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta
2. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana,M.Kom selaku Dosen Pembimbing I yang telah membimbing dalam penggerjaan skripsi ini
4. Yth. Bapak Drs. Mulyono, M.Kom selaku Dosen Pembimbing II yang telah membimbing dalam penggerjaan skripsi ini.
5. Kedua orangtua penulis yang telah senantiasa mendukung, memberi semangat, serta mendoakan penulis
6. Teman-teman yang telah memberikan dukungan dan bantuan dalam penggerjaan skripsi ini.

Dalam penulisan skripsi ini penulis menyadari keterbatasan ilmu pengetahuan dan kemampuan penulis yang menyebabkan skripsi ini jauh dari sempurna, baik dari segi penulisan, penyajian materi, dan juga bahasa. Oleh karena itu penulis meminta kritik dan saran yang dapat dijadikan sebagai pembelajaran serta dapat membangun penulis agar lebih baik lagi dalam mengerjakan tugas-tugas dan permasalahan yang ada kedepannya.

Akhir kata, penulis berharap skripsi ini dapat bermanfaat bagi semua pihak baik itu bagi Ilmu Komputer Universitas Negeri Jakarta, teman-teman serta para pembaca sekalian. Semoga Allah SWT senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan skripsi ini.

Jakarta, 5 Juli 2024



Pramudio

ABSTRAK

PRAMUDIO, Pendekripsi Keliling Luka Menggunakan Metode *Border Following* dengan Bantuan Interpolasi Spline. Skripsi, Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. juli 2024.

Luka kronis merupakan salah satu penyakit yang kompleks, khususnya bagi penyandang penyakit Diabetes Melitus (DM). Proses penyembuhan luka diawasi oleh pekerja medis dengan asesmen, yaitu mengukur keliling luka dan melihat warna luka. Setelah asesmen, pekerja medis baru bisa memberi keputusan dalam penanganan luka. Proses penyembuhan luka perlu melakukan asesmen yang tepat dan pengelolaan yang efektif, hanya saja asesmen manual dalam pengukuran luka sangat memakan waktu dan berpotensi mengganggu penyembuhan. Skripsi ini bertujuan untuk membantu pekerja medis dalam asesmen luka kronis penggunaan metode *border following* dibantu dengan interpolasi *spline* dalam pemrosesan citra untuk mengurangi asesmen luka manual. Penelitian ini memiliki potensi untuk memberikan analisis yang objektif dan reliabel dalam penggunaan pengolahan citra untuk asesmen luka kronis. Dengan memanfaatkan teknologi pengolahan citra, peneliti mencoba mengatasi keterbatasan metode asesmen luka manual dengan menggunakan algoritma *border following*. Yang pertama dilakukan dalam metode ini adalah mengambil foto luka dengan menggunakan perangkat seluler, dilanjutkan dengan pemotongan citra untuk meningkatkan akurasi, lalu metode *border following* dijalankan pada foto yang sudah dipotong untuk mendapatkan daerah kurva sekitar luka, selanjutnya dihaluskan menggunakan interpolasi *spline*. Metode ini dilakukan pada ketiga kategori luka; merah, kuning, dan hitam yang sebanyak 69 data citra. Eksperimen ini menunjukkan *border following* hanya dapat mendekripsi 14 luka dengan rata-rata akurasi masing-masing; merah 70.9% dengan 4 luka, kuning 73.4% pada 2 luka, dan hitam 92.6% pada 8 luka

Kata kunci: luka kronis, *border following*, interpolasi *spline*, asesmen luka, pemrosesan citra, asesmen luka, penyembuhan luka

ABSTRACT

PRAMUDIO, Detection of around Wound Perimeter Using the Border Following Algorithm with the Help of Spline Interpolation. Undergraduate Thesis, Computer Science Program, Faculty of Mathematics and Natural Sciences, State University of Jakarta. July 2024.

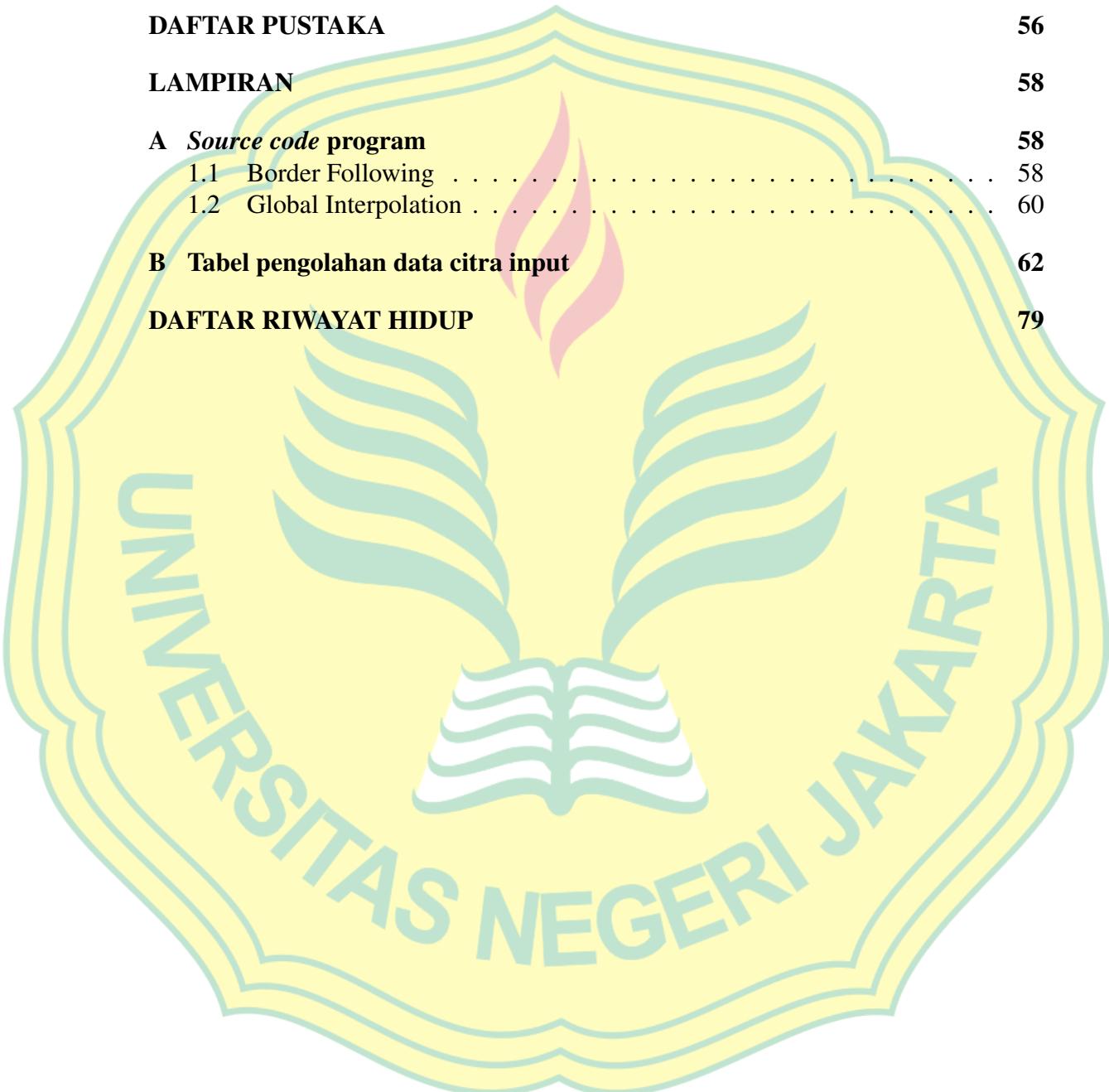
Chronic wounds one of complex illness, particularly for whose suffer Diabetes Mellitus (DM). The process of wound healing need to be monitored by medical worker with assessment, which measure area of wound and looking at the color of the wound. After the assessment, medical worker will make a decision for the wound treatment. The wound healing process involves appropriate assessment and effective management, yet manual methods in wound measurement often time-consuming and potentially interfere with healing. This thesis aims to help the medical worker to chronic wound assessment using the method of border following with the help of spline interpolation in image processing to lessen manual chronic wound assessment. This research holds the potential to bring analysis that are objective and reliable on the uses of image processing for chronic wound assessment. By utilizing technology of image processing, researcher tried to overcome the limitations of manual methods of chronic wound assessment with border following algorithm. The first to do with this method is to use phone to capture a photo, followed by cropping the photo to increase accuracy, then border following method run on photo that are cropped to get curve area around the wound, next smoothed by using spline interpolation. This method used on three kind of wound; red, yellow and black with the quantity of 69 image data. This experiment bring forth border following can only detect 14 wound with the average of accuracy; red 70.9% with 4 wounds, yellow 73.4% from 2 wounds, and black 92.6% on 8 wounds.

Keywords: chronic wounds, border following, spline interpolation, image processing, wound assessment, wound healing.

DAFTAR ISI

| | |
|--|------|
| LEMBAR PERSETUJUAN | iii |
| LEMBAR PERNYATAAN | iv |
| ABSTRAK | vii |
| ABSTRACT | viii |
| DAFTAR ISI | ix |
| DAFTAR GAMBAR | xii |
| DAFTAR TABEL | xii |
| | |
| I PENDAHULUAN | 1 |
| 1.1 Latar Belakang Masalah | 1 |
| 1.2 Rumusan Masalah | 5 |
| 1.3 Pembatasan Masalah | 5 |
| 1.4 Tujuan Penelitian | 6 |
| 1.5 Manfaat Penelitian | 6 |
| | |
| II KAJIAN PUSTAKA | 7 |
| 2.1 <i>Border Following</i> | 7 |
| 2.2 <i>Splines interpolation imaging</i> | 14 |
| 2.2.1 <i>Bézier Spline</i> | 14 |
| 2.2.2 <i>B-Spline</i> | 15 |
| 2.2.3 <i>Global Interpolation</i> | 19 |
| 2.2.4 <i>Local Interpolation</i> | 25 |
| | |
| III METODOLOGI PENELITIAN | 32 |
| 3.1 Perancangan Sistem | 32 |
| 3.2 <i>Border Following</i> | 33 |
| 3.3 Interpolasi | 36 |
| 3.4 Perancangan Eksperimen | 41 |
| 3.4.1 Sumber Data | 41 |
| 3.4.2 Validasi | 41 |
| | |
| IV HASIL DAN PEMBAHASAN | 43 |
| 4.1 Pengolahan data citra input | 43 |
| 4.2 Deteksi Keliling Menggunakan <i>Border Following</i> | 43 |
| 4.3 Penghalusan menggunakan Interpolasi Global | 45 |
| 4.4 Validasi | 48 |

| | |
|-------------------------------------|-----------|
| V KESIMPULAN DAN SARAN | 55 |
| 5.1 Kesimpulan | 55 |
| 5.2 Saran | 55 |
| DAFTAR PUSTAKA | 56 |
| LAMPIRAN | 58 |
| A <i>Source code</i> program | 58 |
| 1.1 Border Following | 58 |
| 1.2 Global Interpolation | 60 |
| B Tabel pengolahan data citra input | 62 |
| DAFTAR RIWAYAT HIDUP | 79 |



DAFTAR GAMBAR

| | | |
|-------------|--|----|
| Gambar 2.1 | <i>surroundness</i> antar komponen yang terhubung (b) dan antar tepi(<i>borders</i>) (c). | 8 |
| Gambar 2.2 | kondisi <i>starting point</i> dari algoritma <i>border following</i> untuk tepi luar (a) dan tepi lubang (b) | 9 |
| Gambar 2.3 | Ilustrasi proses pemberian hierarki terhadap piksel citra; lingkaran pada piksel menunjukkan titik mula pindai <i>border following</i> | 11 |
| Gambar 2.4 | Struktur topologi antara tepi satu dengan tepi yang lain jika menggunakan Algoritma pertama. Hasil citra (a) dan Struktur yang diekstrak (b) | 12 |
| Gambar 2.5 | Hasil pencuitan blob dengan Algoritma kedua. “#” merepresentasikan <i>starting point</i> dari tepi terluar, “:” menunjukkan nilai -2. | 13 |
| Gambar 2.6 | Kurva Bézier berderajat-2. | 15 |
| Gambar 2.7 | Kurva polinomial kubik dipotong tiga segmen. | 16 |
| Gambar 2.8 | Kurva Gambar (2.7) ditunjukkan dengan segmen polinomial yang direpresentasikan dalam bentuk Bézier. | 16 |
| Gambar 2.9 | definisi rekursif dari rumus dasar B-spline. | 18 |
| Gambar 2.10 | Contoh interpolasi kurva menggunakan parameterisasi <i>chord length</i> dan <i>knot vector</i> yang diperoleh dengan rata-rata parameter. | 22 |
| Gambar 2.11 | Contoh interpolasi kurva dengan parameterisasi dan <i>knot vector</i> berbeda. | 23 |
| Gambar 2.12 | Contoh interpolasi kurva dengan parameterisasi dan <i>knot vector</i> berbeda. | 23 |
| Gambar 2.13 | Interpolasi kurva dengan derajat berbeda menggunakan parameterisasi chord length dan knot yang diperoleh dengan rata-rata. | 24 |
| Gambar 2.14 | Interpolasi kurva kubik global ke data yang berisi titik-titik kolinear (lihat garis putus-putus). | 24 |
| Gambar 2.15 | Perhitungan vektor tangen (\mathbf{V}_k) dan turunan (\mathbf{D}_k) untuk interpolasi kurva lokal. | 26 |
| Gambar 2.16 | $C^{(1,1)}$ Interpolasi permukaan bikubik lokal | 29 |
| Gambar 3.1 | Diagram alir penelitian | 32 |
| Gambar 3.2 | Hasil Interpolasi citra | 36 |
| Gambar 3.3 | Data citra luka | 41 |
| Gambar 3.4 | Validasi alat deteksi | 42 |
| Gambar 4.1 | Kode untuk membaca citra | 43 |
| Gambar 4.2 | Kode border following | 44 |

| | | |
|-------------|--|----|
| Gambar 4.3 | Hasil pindai <i>border following</i> | 45 |
| Gambar 4.4 | Kode Global Interpolation | 45 |
| Gambar 4.5 | Kode <i>chord length</i> | 46 |
| Gambar 4.6 | Kode knot vector | 46 |
| Gambar 4.7 | Kode rumus dasar (N) | 47 |
| Gambar 4.8 | Hasil Interpolasi dari deteksi <i>border following</i> | 47 |
| Gambar 4.9 | Kode Bézier | 48 |
| Gambar 4.10 | Penghalusan kurva menggunakan bázier | 48 |
| Gambar 4.11 | Alat validasi | 49 |
| Gambar 4.12 | Alat validasi | 49 |



DAFTAR TABEL

| | | |
|-----------|--|----|
| Tabel 2.1 | Aturan penetapan <i>parent border</i> | 9 |
| Tabel 4.1 | Percobaan pada luka merah | 49 |
| Tabel 4.1 | Percobaan pada luka merah | 50 |
| Tabel 4.2 | Similiaritas deteksi luka merah <i>border following</i> dan yang dibantu dengan interpolasi | 50 |
| Tabel 4.3 | Percobaan pada luka kuning | 51 |
| Tabel 4.4 | Similiaritas deteksi luka kuning <i>border following</i> dan yang dibantu dengan interpolasi | 51 |
| Tabel 4.5 | Percobaan pada luka hitam | 52 |
| Tabel 4.6 | Similiaritas deteksi luka hitam <i>border following</i> dan yang dibantu dengan interpolasi | 52 |
| Tabel 4.6 | Similiaritas deteksi luka hitam <i>border following</i> dan yang dibantu dengan interpolasi | 53 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 62 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 63 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 64 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 65 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 66 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 67 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 68 |
| Tabel 2.1 | Hasil <i>crop</i> sumber data | 69 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 69 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 70 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 71 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 72 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 73 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 74 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 75 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 76 |
| Tabel 2.2 | Pemeriksaan <i>Border Following</i> | 77 |
| Tabel 2.3 | Similiaritas deteksi luka merah <i>border following</i> dan yang dibantu dengan interpolasi | 77 |
| Tabel 2.4 | Similiaritas deteksi luka kuning <i>border following</i> dan yang dibantu dengan interpolasi | 78 |
| Tabel 2.5 | Similiaritas deteksi luka hitam <i>border following</i> dan yang dibantu dengan interpolasi | 78 |

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Kulit merupakan perlindungan pertama manusia dalam menjaga tubuhnya dari berbagai macam substansi. Apabila kulit terluka, maka diperlukan penanganan yang baik agar tidak terjadi infeksi. Luka adalah keadaan di mana fungsi anatomis kulit normal mengalami kerusakan akibat proses patologis yang berasal dari internal maupun eksternal dan mengenai organ tertentu. Ketika luka tidak terinfeksi, maka pada normalnya luka tersebut akan melakukan penyembuhan. Proses penyembuhan terdapat beberapa fase, yaitu: hemostasis (beberapa jam pasca-terjadinya luka), inflamasi (1 - 3 hari), proliferasi (4 - 21 hari), dan remodelling (21 hari - 1 tahun). Fase-fase penyembuhan luka terjadi secara bertahap, namun dapat terjadi secara bersamaan (*overlap*) (Simon, 2018).

Dari beberapa kondisi luka, terdapat luka yang proses penyembuhannya tidak normal dengan durasi fase-fase yang sesuai. Kondisi tersebut disebut dengan luka kronis (Landén et al., 2016), kondisi ini dapat memiliki kaitan dengan berbagai faktor yang memperlambat penyembuhan luka seperti adanya penyakit kronis, insufisiensi vaskuler, diabetes, gangguan nutrisi, penuaan, dan berbagai faktor lokal pada luka (tekanan, infeksi, dan edema). Secara umum, luka kronis dapat terjadi akibat ulkus vena, ulkus arteri, ulkus dekubitus, dan ulkus diabetik (Zhao et al., 2016).

Hal paling pertama yang dilakukan untuk menangani suatu masalah adalah mengidentifikasi masalah tersebut, luka tidak menjadi pengecualian. Pengidentifikasi luka yang akurat dapat membantu memberikan diagnosa yang akurat, penanganan luka yang tepat, memantau perbaikan luka, menghindari terjadinya komplikasi, serta dapat mengurangi biaya perawatan luka. Pengidentifikasi luka kronis umumnya didasarkan pada dua jenis tinjauan klinis, yaitu: tinjauan visual terhadap warna dominan luka dengan mengidentifikasi jaringan luka, dan tinjauan manual terhadap bentuk luka (area, perimeter, kedalaman, dan lain-lain) dengan pemeriksaan luka. Saat ini, tersedia dua teknik untuk pemeriksaan ini, yang pertama adalah metode manual langsung yang digunakan oleh dokter dan perawat untuk mengukur secara berkala dimensi luka menggunakan penggaris, dan teknik yang kedua adalah melacak batas luka pada

kertas kalkir transparan yang ditempatkan pada kotak metrik. Kertas kalkir merupakan sebuah kertas yang memiliki permukaan yang tembus pandang dan sering digunakan oleh desainer untuk merancang desain atau gambar. Kertas ini memiliki struktur seperti kaca yang dapat dilihat secara tembus pandang ke bagian belakang kertas kalkir tersebut. Permukaan luka kemudian ditentukan dengan menghitung jumlah kotak secara manual setelah memindai gambar. Namun, kedua metode tersebut tidak dapat mengukur dimensi luka secara akurat dan kertas kalkir yang digunakan dapat menyebabkan infeksi pada luka (Gupta, 2017). Salah satu metode yang diusulkan untuk mengatasi masalah ini adalah dengan menggunakan metode yang berbasis *digital image processing*.

Metode berbasis *digital image processing* merupakan alternatif untuk penilaian luka karena dapat memberikan langkah-langkah yang objektif, lebih akurat, dan dapat direproduksi. Salah satu keuntungan besarnya adalah metode ini memiliki resiko yang lebih rendah karena tidak ada kontak antara luka dan sistem pengukuran. Segmentasi citra merupakan proses mempartisi gambar digital menjadi beberapa segmen (set piksel, disebut juga *superpixel*) yang memiliki fitur atau atribut yang sama. Segmentasi bertujuan untuk menyederhanakan atau mengubah representasi suatu citra menjadi sesuatu yang lebih bermakna dan lebih mudah untuk dianalisis. Segmentasi citra biasa digunakan untuk menemukan objek dan batas (seperti garis, kurva, dan lain-lain) dalam gambar (Faten Abu Shmmala, 2013). Biasanya segmentasi menggunakan informasi lokal dalam gambar digital untuk menghitung segmentasi terbaik, seperti informasi warna yang digunakan untuk membuat histogram atau informasi yang mengindikasikan tepi, batas atau informasi tekstur (Khattab et al., 2014).

Segmentasi warna citra (*color image segmentation*) didasarkan pada fitur warna piksel gambar yang mengasumsikan bahwa warna-warna homogen pada gambar bersesuaian dengan kelompok yang terpisah. Dengan kata lain, setiap kelompok mendefinisikan kelas piksel yang memiliki properti warna yang sama. Karena hasil segmentasi bergantung pada ruang warna (*color space*) yang digunakan, tidak ada ruang warna tunggal yang dapat memberikan hasil yang dapat diterima untuk semua jenis gambar. Karena alasan ini, banyak penulis yang mencoba menentukan ruang warna yang sesuai dengan masalah segmentasi warna citra spesifik mereka (*ibid.*). Beberapa macam dari ruang warna (*color space*), yaitu *RGB*, *CMY(K)*, *HSV*, *CIE*, *L*a*b*, *L*u*v*, dan *YCrCb*. Setiap ruang warna (*color space*) mempunyai sekurang-kurangnya 3 elemen warna dasar.

Salah satu cara mengidentifikasi luka adalah dengan tinjauan visual untuk identifikasi jaringan luka dari warna dominan. Warna luka akan memberikan banyak informasi penting berkaitan dengan perkiraan waktu penyembuhan luka, kondisi umum luka apakah dalam keadaan baik atau memburuk, dan risiko komplikasi. Warna tersebut dapat dipisahkan menjadi tiga warna, yaitu merah, kuning, dan hitam. Tinjauan visual ini, bisa dilakukan dengan metode segmentasi warna citra. Salah satu metode segmentasi warna citra yang telah dilakukan Aprilia Khairunnisa adalah dengan melihat ruang warna LAB terhadap segmentasi warna *red, yellow, and black*. Proses segmentasinya bisa dilakukan dengan menggunakan dua metode yaitu *k-means* atau *mean shift*. *K-means* merupakan metode pengelompokan dari sejumlah *cluster* yang terpisah. "K" mengacu pada jumlah *cluster* yang ditentukan (Manoj Kumar Yadav, 2013). Metode *k-means* mempartisi *n-set* input data menjadi *k-cluster* di mana setiap set input data termasuk ke dalam cluster dengan mean terdekat (Xingming Zheng, 2012). Metode ini akan mempartisi data yang berkarakteristik sama ke dalam suatu kelompok yang sama dan data yang lainnya ke kelompok yang berbeda (Gustientiedina Gustientiedina, 2019). *Mean shift* adalah teknik analisis *non-parametric feature space* untuk mencari nilai maksimum dari fungsi kerapatan atau kepadatan yang diberikan dari data diskrit yang ada di fungsi tersebut. *Mean shift* merupakan prosedur berulang (iteratif) sederhana yang menggeser setiap titik data ke rata-rata (*mean*) titik data di daerahnya. Algoritma *mean shift* disebut juga sebagai algoritma pencarian mode (Cheng, 1995).

Dari metode yang dilakukan oleh Aprilia Khairunnisa terdapat kekurangan, di mana hasil dari penelitiannya belum dapat memperlihatkan pengaruh dari penggunaan model warna LAB pada proses segmentasi (Aprilia, 2021). ada satu hal yang masih dilakukan secara manual dalam proses metode tersebut, yaitu memasukkan hanya gambar luka saja, tanpa sekitar lukanya. Hal ini bisa dikembangkan dengan memasukkan metode segmentasi yang mendeteksi tepi luka yaitu *active contour*.

Model *active contour*, juga disebut *Snake*, adalah *framework* dalam pengolahan citra yang diperkenalkan oleh Michael Kass, Andrew Witkin, dan Demetri Terzopoulos untuk menggambarkan garis objek dari gambar 2D yang mungkin *noisy*. Model *active contour* populer dalam pengolahan citra, dan *active contour* banyak digunakan dalam aplikasi seperti *object tracking, shape recognition, segmentasi, deteksi tepi, dan pencocokan stereo*. *Active contour* merupakan peminimalisir energi, dapat dideformasi yang terpengaruh oleh *constraint* dan *image*

forces yang menarik ke arah objek kontur dan gaya di dalam yang menahan deformasi. *Active contour* bisa diartikan sebagai model yang bisa dideformasi kepada citra dengan meminimalisir energi. Dalam dua dimensi, model bentuk aktif mewakili versi diskrit dari pendekatan ini, mengambil keuntungan dari model distribusi titik untuk membatasi rentang bentuk ke domain eksplisit yang dipelajari dari set pelatihan (Kass et al., 1988).

Active contour merupakan metode yang biasanya tidak digunakan sendiri, dikarenakan metode ini perlu mengetahui bentuk suatu benda dan interaksi pengguna untuk mendapatkan hasil yang diinginkan. Pada Teknik yang dilakukan oleh Muhammad Rizki setelah memasukan citra dimulai dengan mengkonversi data *RGB* menjadi *grayscale*. Lalu gambar *grayscale* tersebut mulai dideteksi menggunakan active contour di mana inisialisasi kurva awalnya yang seharusnya integer diubah dengan *float*. Untuk rumusan energi internal, tidak diubah dari *active contour* yang asli, tapi ketika masuk ke energi eksternal, persamaan yang dipakai berbeda sedikit. Setelah mendapatkan energi dari citra tersebut, dimulai proses update intersi kurva. Proses tersebut menggunakan memakai metode gradien arah direction untuk mendapatkan turunan pertama citra yang akan digunakan ke dalam rumus, lalu dilakukan iterasi yang dihitung melalui proses interpolasi (Rizki, 2022).

Interpolasi dalam matematika umum adalah proses membuat suku-suku peralihan dari titik yang diketahui. Dalam pemrosesan gambar, biasanya digunakan untuk *image scaling*, *image resampling*, dan *image resize*. Ada banyak algoritma yang saat ini digunakan untuk mengubah gambar digital. Kebanyakan dari mereka berupaya mereproduksi replika aslinya yang menarik secara visual. Sekarang seiring dengan teknologi untuk area tampilan yang lebih kecil untuk dilihat pada berbagai perangkat, ukuran gambar umumnya diambil sampelnya (atau disubsampel atau dikurangi) untuk menghasilkan thumbnail. Pengambilan sampel gambar (atau pembesaran atau interpolasi) paling umum dilakukan pada monitor atau televisi berukuran layar besar (Parsania et al., 2016). Interpolasi yang akan dipakai penulis akan berdasarkan pada metode **NURBS** (*Non-Uniform Rational B-Splines*)(Piegl et al., 1996) yang merupakan interpolasi penghalusan suatu kurva, ini bertujuan untuk menambahkan akurasi dalam pendekripsi luka, di mana pendekripsinya hanya menangkap tepi secara kasar.

seperti yang sudah tertuliskan dalam penilitian Muhammad Rizki, sebaiknya *active contour* digantikan untuk meningkatkan akurasi deteksi luka (Rizki, 2022). Dari metode yang dilakukan Muhammad Rizki, terdapat keunggulan di mana arah

tepinya lebih jelas dibanding dengan citra asli sehingga membantu *active contour* untuk melihat tepi suatu citra, tetapi ketika dilihat baik-baik, hasil dari gradien citra arah yang menggunakan interpolasi menambahkan arah tepi yang tidak diinginkan sehingga mengganggu pendektsian *active contour*. Untuk mengatasi masalah tersebut, penulis akan menggunakan algoritma *Border Following* Suzuki.

Algoritma *Border Following* suzuki merupakan salah satu algoritma topologi gambar biner digital yang pertama mendefinisikan hubungan hirarki antar pembatasan dan membedakan antara batas luar atau batas lubang. Algoritma ini memindai gambar dari kiri ke kanan, mengecek ada nya objek piksel pada piksel yang dipindai. algoritma ini akan memindai sekitar piksel yang sedang dipindai untuk menentukan apakah piksel ini akan naik derajatnya. apabila sudah selesai pemindaian piksel ini, akan pindah pemindaian ke arah jarum jam, prioritas kanan. proses ini akan diulang sampai tidak ada piksel yang bisa digantikan derajatnya. setelah sudah mendapatkan dari derajat dari semua piksel, maka bisa dilihat batas(*border*) pada gambar yang dipindai algoritma *Border Following*(Satoshi Suzuki, 1985).

Penulis menginginkan untuk mengembangkan metode yang sudah pernah dilakukan oleh Muhammad Rizki(Rizki, 2022). Penelitian ini bertujuan untuk meningkatkan akurasi dalam pendektsian luka menggunakan *active contour* dengan mengubah metode deteksi citra dengan *border following*, lalu melakukan interpolasi pada hasil pendektsian lukanya. Penelitian ini bertujuan untuk membantu kalangan dokter dan perawat terkait penilaian luka kronis agar dapat memberikan hasil aproksimasi yang lebih dekat dengan *ground truth*(data asli yang diambil dari menggambar manual tepi luka).

1.2 Rumusan Masalah

Berdasarkan Latar belakang yang telah dikemukakan di atas, Fokus permasalahan pada penelitian ini adalah “Bagaimana mengembangkan pendektsian keliling luka kronis menggunakan metode *border following* yang dibantu dengan interpolasi spline”.

1.3 Pembatasan Masalah

Pembatasan masalah pada penelitian ini antara lain:

1. Pendekripsi keliling luka kronis menggunakan *border following* yang dibantu dengan interpolasi terhadap data citra luka yang didapat dari penelitian luka Ns. Ratna Aryani, M.Kep, tahun 2018 (Aryani et al., 2018).
2. Penelitian dilakukan sampai mendapatkan hasil, yaitu nilai akurasi dari selisih area kurva *border following* terhadap area *ground truth*(data asli yang diambil dari menggambar manual tepi luka).

1.4 Tujuan Penelitian

Tujuan penelitian ini adalah untuk melanjutkan penelitian Rizki(Rizki, 2022) dalam mendekripsi citra keliling luka dengan menggunakan metode *Border Following* yang ditambahkan interpolasi spline pada pendekripsi keliling luka kronis.

1.5 Manfaat Penelitian

1. Bagi peneliti

Penelitian ini merupakan media penerapan ilmu pengetahuan, khususnya dalam pengembangan metode *border following* dalam pengkajian luka kronis serta membantu penulis untuk menyelesaikan perkuliahan.

2. Instansi Terkait

Metode yang diajukan diharapkan dapat membuka peluang untuk diajukan ke instansi kesehatan terkait dalam proses pengkajian luka kronis.

3. Bagi ilmu pengetahuan

- Mahasiswa

Diharapkan penelitian ini dapat membantu dalam penulisan paper yang bersangkutan dengan pendekripsi luka.

- Bagi peneliti selanjutnya

Diharapkan penelitian ini dapat dikembangkan oleh peneliti selanjutnya untuk mengembangkan metode dari penelitian ini.

BAB II

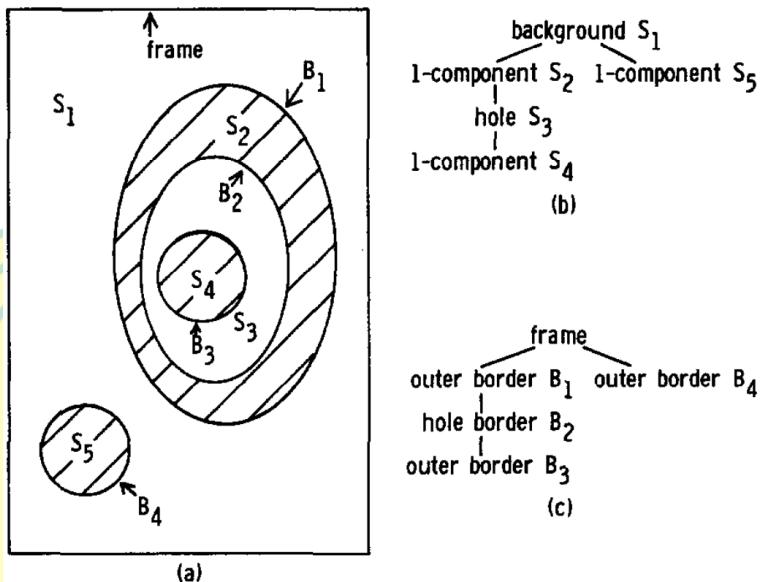
KAJIAN PUSTAKA

2.1 *Border Following*

Salah satu teknik yang sering dipakai dalam pengolahan citra adalah *Border Following*, terutama citra biner. *Border following* telah diteliti oleh beberapa orang karena teknik ini memiliki pengaplikasian yang cukup luas, seperti pengenalan objek, analisis citra, deteksi objek, dan kompresi data citra (Satoshi Suzuki, 1985)

Dalam penelitian yang dibuat oleh Suzuki, ia menawarkan dua algoritma *border following* yang sampai saat ini masih digunakan oleh *library* pengolahan citra yang umum diketahui yaitu **OpenCV**. Algoritma ini merupakan salah satu algoritma yang mengenalkan pendefinisian hubungan hierarki antar tepi (*border*). Tidak hanya itu, algoritma ini juga membedakan tepi lubang (*hole*) dan tepi luar (*outer border*) di sebuah objek. Suzuki memberikan definisi pada algoritma ini, yaitu:

- **Definisi 1 (*border point*).** Sebuah piksel bernilai 1 yang dikelilingi oleh piksel bernilai 0 dapat disebut sebagai *border point*.
- **Definisi 2 (komponen yang mengelilingi komponen lain.).** Dari gambar (2.1) dapat dikatakan bahwa komponen S_2 mengelilingi komponen S_4
- **Definisi 3(*outer border* dan *hole border*).** Sebuah rangkaian *border point* antar komponen yang mempunyai nilai piksel 1 (objek) dengan komponen yang mempunyai nilai piksel 0 (latar belakang)
- **Definisi 4(*parent border*).** Berdasarkan gambar (2.1) dapat dikatakan bahwa tepi B_2 adalah *parent border* dari tepi B_3
- **Definisi 5(tepi yang mengelilingi tepi lain).** Dari gambar (2.1) dapat dikatakan bahwa tepi B_2 mengelilingi tepi B_3 .



Gambar 2.1: *surroundness* antar komponen yang terhubung (b) dan antar tepi(borders) (c).

Diasumsikan citra yang dimasukan algoritma ini berupa citra biner. Piksel dengan nilai 0 dan 1 disebut sebagai 0-piksel dan 1-piksel. Nilai sebuah piksel dalam koordinat (i, j) yang masing-masing melambangkan baris dan kolom sebuah citra digital didefinisikan sebagai $f_{i,j}$. Baris paling pinggir yang berada di atas, kanan, bawah, dan kiri adalah *frame*(bingkai) dari citra tersebut. Dalam hal ini, setiap tepi baru yang ditemukan akan diberi angka unik dan disebut sebagai **NBD**. Asumsikan NBD dari *frame* adalah 1, sementara tepi lain diberi angka NBD secara berurutan. Nilai NBD dari *parent* setiap tepi di dalam variabel **LNBD** kemudian disimpan. Berikut merupakan langkah-langkah algoritma pertama *border following*:

Dimulai dengan pemindaiannya piksel citra dari kiri ke kanan hingga *scanner* menemukan piksel objek ($f_{i,j} \neq 0$). Dilanjutkan dengan menentukan apakah piksel tersebut merupakan tepi luar atau tepi lubang. Untuk setiap baris baru yang dipindai, kembalikan nilai LNBD ke 1.

1. Langkah 1

- jika $f_{i,j} = 1$ dan $f_{i,j-1} = 0$, maka piksel (i, j) adalah *starting point* dari operasi *border following* untuk tepi luar(*outer border*). inkrement NBD dan set $(i_2, j_2) \leftarrow (i, j - 1)$.
- jika $f_{i,j} \geq 1$ dan $f_{i,j+1} = 0$, maka piksel (i, j) adalah *starting point* dari

operasi *border following* untuk tepi lubang(*hole border*). inkremen NBD, set $(i_2, j_2) \leftarrow (i, j + 1)$, dan $\text{LNBD} \leftarrow f_{i,j}$ jika $f_{i,j} > 1$

- (c) jika kedua kondisi tidak terpenuhi, maka lanjut ke langkah 4.



Gambar 2.2: kondisi *starting point* dari algoritma *border following* untuk tepi luar (a) dan tepi lubang (b)

2. Langkah 2

Tentukan *parent border* untuk tepi piksel (i, j) berdasarkan jenis tepi piksel dari piksel (i, j) dan LNBD (tepi terakhir yang ditemukan), mengikuti aturan pada tabel (2.1)

Tabel 2.1: Aturan penetapan *parent border*

| | <i>Type of border B' with the sequential number of LNBD</i> | |
|---------------------|---|---|
| <i>Type of B</i> | <i>Outer border</i> | <i>Hole border</i> |
| <i>Outer border</i> | <i>The parent border of the border B'</i> | <i>The border B'</i> |
| <i>Hole border</i> | <i>The border B'</i> | <i>The parent border of the border B'</i> |

3. langkah 3

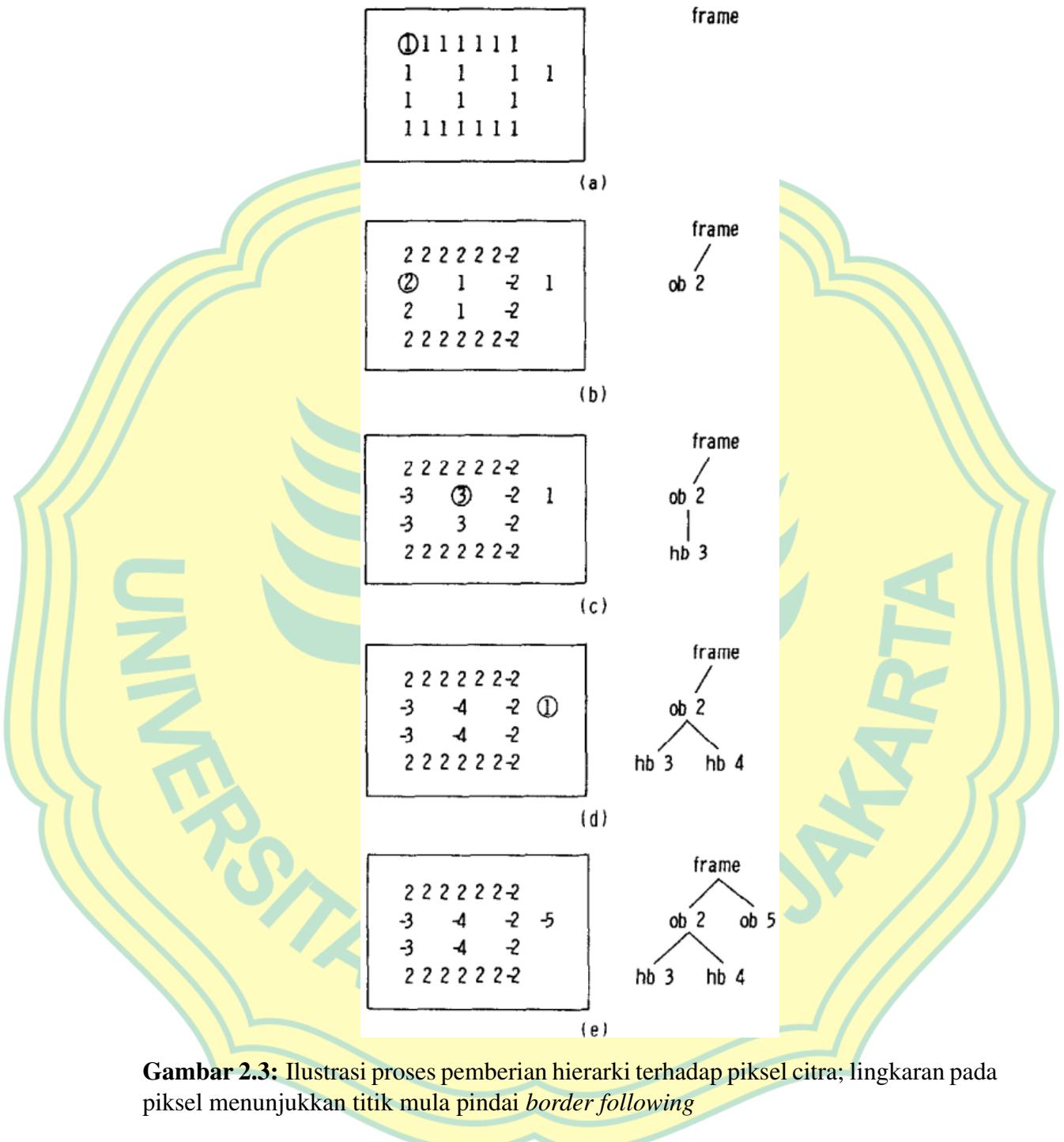
Proses *border following* dimulai dari *starting point* piksel (i, j) pada langkah ini

- (a) Dimulai dari piksel (i_2, j_2) , periksa piksel tetangga searah jarum jam dari piksel (i, j) sampai menemukan piksel *non-zero*. Piksel *non-zero* yang pertama ditemukan didefinisikan sebagai (i_1, j_1) . jika piksel *non-zero* tidak ditemukan, set $f_{i,j} = -\text{NBD}$ dan lanjutkan ke langkah keempat
 - (b) $(i_2, j_2) \leftarrow (i_1, j_1)$ dan $(i_3, j_3) \leftarrow (i, j)$
 - (c) Mulai dari elemen setelah piksel (i_2, j_2) , mencari piksel *non-zero* di sekitar piksel (i_3, j_3) dengan arah berlawanan jarum jam dan set piksel *non-zero* pertama yang ditemukan ke dalam variabel (i_4, j_4) .

- (d) Ubah nilai f_{i_3,j_3} dari piksel (i_3, j_3) dengan aturan menandai(*marking policy*):
- Jika piksel $(i_3, j_3 + 1)$ adalah 0-piksel (piksel bernilai 0), set $(i_3, j_3 + 1) \leftarrow -\text{NBD}$
 - Jika piksel $(i_3, j_3 + 1)$ bukan 0-piksel (piksel bernilai 0), set $(i_3, j_3 + 1) \leftarrow \text{NBD}$
 - Jika kedua kondisi di atas tidak terpenuhi, jangan ubah nilai f_{i_3,j_3}
- (e) Jika $(i_4, j_4) = (i, j)$ dan $(i_3, j_3) = (i_1, j_1)$ (kembali ke *starting point*), maka lanjut ke langkah 4, jika tidak set $(i_2, j_2) \leftarrow (i_3, j_3)$, $(i_3, j_3) \leftarrow (i_4, j_4)$ dan kembali ke langkah (3.c).

4. Langkah 4

jika $f_{i,j} \neq 1$, maka $\text{LNBD} \leftarrow |f_{i,j}|$ dan lanjutkan pemindaian dari mulai piksel $(i, j + 1)$ untuk kembali mencari nilai piksel objek $f_{i,j} \neq 0$. Algoritma dinyatakan selesai jika pemindai sudah sampai sudut kanan bawah dari citra(piksel terakhir)



Gambar 2.3: Ilustrasi proses pemberian hierarki terhadap piksel citra; lingkaran pada piksel menunjukkan titik mula pindai *border following*

(a)

| | 10 | 20 | 30 | 40 |
|----|----------------------------|-------------------|----------|----|
| 1 | | | | |
| 2 | 22: 333- | 4444444444444444+ | 555= 66# | |
| 3 | 21: 31- 417* | 771+ 51= 61# | | |
| 4 | 2: 3- 447* | 774+ 5= 6# | | |
| 5 | : | | | |
| 6 | : 3- 41* 888888888? | 71+ 5= # | | |
| 7 | : 31- 41* 88199999999918? | 71+ 51= * | | |
| 8 | 3- 4* 818 | 991? 71+ 51= | | |
| 9 | 31- 4* 818 AAAAAAA@ | 91? 7+ 51= | | |
| 10 | 3- 41* 818 A1BBBBBBB1e | 91? 71+ 5= | | |
| 11 | 3- 4* 818 AAB | 88Ae 91? 7+ 5= | | |
| 12 | 3- 4* 8e A* | 8e 9? 7+ 5= | | |
| 13 | 3- 4* 818 A* CCCCC* Be | 91? 7+ 5= | | |
| 14 | 3- 4* 8e A1* C1DDDD01* | 81e 9? 7+ 5= | | |
| 15 | 3- 4* 8e A* C1 D1 | Be 9? 7+ 5= | | |
| 16 | 3- 4* 8e A* C* D | Be 9? 7+ 5= | | |
| 17 | 3- 4* 8e A* C* EE* D* | Be 9? 7+ 5= | | |
| 18 | 3- 4* 8e A* C* E1* D* | Be 9? 7+ 5= | | |
| 19 | 3- 4* 8e A* C* EE* D* | Be 9? 7+ 5= | | |
| 20 | 3- 4* 8e A* C* D* | Be 9? 7+ 5= | | |
| 21 | 3- 4* 8e A* C1* D1* | Be 9? 7+ 5= | | |
| 22 | 3- 4* 8e A1* C1DDDD01* | 81e 9? 7+ 5= | | |
| 23 | 3- 4* 818 A* CCCCC* Be | 91? 7+ 5= | | |
| 24 | 3- 4* 8e A* Be | 9? 7+ 5= | | |
| 25 | 3- 4* 818 AAB* | 88Ae 91? 7+ 5= | | |
| 26 | 3- 41* 818 A1BBBBBBB1e | 91? 71+ 5= | | |
| 27 | 31- 4* 818 AAAAAAA@ | 91? 7+ 51= * | | |
| 28 | 3- 4* 818 | 91? 7+ 5= | | |
| 29 | 31- 41* 8198 | 991? 71+ 51= | | |
| 30 | * 31- 41* 88199999999918? | 71+ 51= * | | |
| 31 | * 3- 41* 888888888? | 71+ 5= * | | |
| 32 | F* 3- 447* | 774+ 5= G* | | |
| 33 | F1* 31- 417* | 771+ 51= G1* | | |
| 34 | FF* 333- 4444444444444444+ | 555= GG* | | |
| 35 | | | | |

(b) Structured borders

| Border no. | Outer border (1) /Hole border (0) | Coordinates of the border following starting point | Parent border no. |
|------------|--------------------------------------|--|----------------------|
| 1 | 0 | (1, 1) | 0 |
| 2 | 1 | (2, 3) | 1 |
| 3 | 1 | (2, 8) | 1 |
| 4 | 1 | (2, 15) | 1 |
| 5 | 1 | (2, 35) | 1 |
| 6 | 1 | (2, 41) | 1 |
| 7 | 0 | (3, 17) | 4 |
| 8 | 1 | (5, 18) | 7 |
| 9 | 0 | (7, 18) | 8 |
| 10 | 1 | (9, 19) | 9 |
| 11 | 0 | (11, 19) | 10 |
| 12 | 1 | (13, 20) | 11 |
| 13 | 0 | (15, 20) | 12 |
| 14 | 1 | (17, 22) | 13 |
| 15 | 1 | (30, 2) | 1 |
| 16 | 1 | (30, 44) | 1 |

Gambar 2.4: Struktur topologi antara tepi satu dengan tepi yang lain jika menggunakan Algoritma pertama. Hasil citra (a) dan Struktur yang diekstrak (b)

Selain algoritma pertama Suzuki, ia juga mengusulkan algoritma kedua *border following* di mana hasilnya hanya tepi terluarnya saja. Berikut perbedaan algoritma pertama dan algoritma kedua yang diusulkan Suzuki:

1. Kondisi *starting point* untuk melakukan operasi *border following* hanya berlaku untuk tepi terluar (gambar 2.2) dan jika $\text{LNBD} \leq 0$.
 2. *Marking policy* tetap sama dengan algoritma pertama (langkah 3.d), tetapi nilai NBD dan -NBD akan diganti masing-masing dengan nilai 2 dan -2
 3. Nilai LNBD juga tetap disimpan dari piksel *non-zero* yang ditemukan sebelumnya. Akan tetapi, LNBD akan di-*reset* ke nilai 0 untuk setiap bari baru yang dipindai

Gambar 2.5: Hasil pencuitan blob dengan Algoritma kedua. “#” merepresentasikan *starting point* dari tepi terluar, “:” menunjukkan nilai -2.

2.2 Splines interpolation imaging

Pada konteks pengolahan citra, Interpolasi merupakan proses pembangunan kurva atau *spline*, atau permukaan(*surface*) terdefinisi bebas yang melalui sekumpulan titik data(*data points*) secara tepat. Terkadang diberikan pembatas tambahan untuk mendapatkan interpolasi yang dibatasi agar mendapatkan hasil kurva atau *spline* yang diinginkan(Lockyer, 2006). *Spline* yang sering dirujuk pada pengolahan citra adalah kurva yang dibentuk dengan gabungan polinominal. Contoh *Spline* yang sudah di-interpolasikan dapat dilihat pada gambar (2.10)

2.2.1 Bézier Spline

Bézier *Spline* merupakan salah satu kurva polinominal menggunakan kumpulan diskrit titik kontrol untuk membuat kurva yang halus dan kontinu.

Bézier *spline* dengan derajat ke- n terdefinisikan sebagai berikut

$$\mathbf{C}(u) = \sum_{i=0}^n B_{i,n}(u) \mathbf{P}_i \quad 0 \leq u \leq 1 \quad (2.1)$$

Fungsi dasar (*blending*), $\{B_{i,n}(u)\}$, merupakan polinomial Bernstein yang didefinisikan sebagai

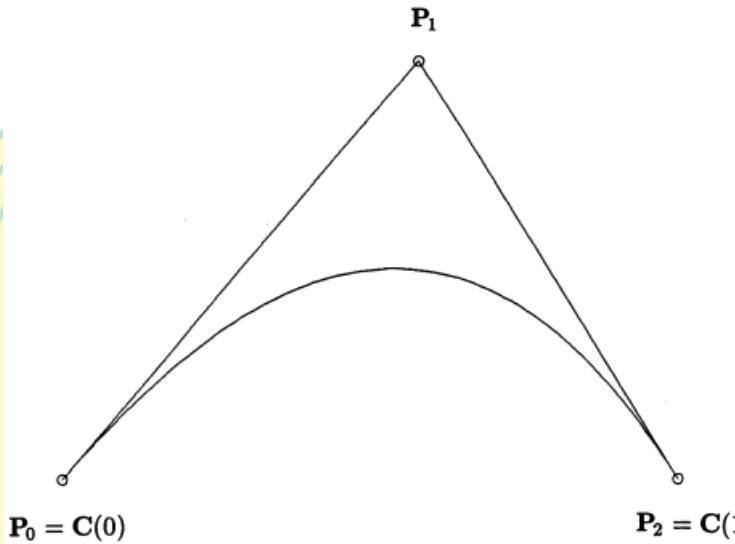
$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-1} \quad (2.2)$$

Koefisien geometri dalam bentuk ini, $\{\mathbf{P}_i\}$, disebut sebagai titik kontrol(*control points*), dan pada persamaan (2.1), u nya harus diantara 0 sampai 1 [$u \in [0, 1]$]

Bisa dicontohkan hasil Bézier sebagai berikut. Bézier memiliki derajat 2, $n = 2$, tarik dari persamaan (2.1) dan (2.2), bisa didapat $\mathbf{C}(u) = (1-u)^2 \mathbf{P}_0 + 2u(1-u) \mathbf{P}_1 + u^2 \mathbf{P}_2$ akan menghasilkan busur parabola dari \mathbf{P}_0 ke \mathbf{P}_2 . Dapat kita lihat pada gambar(2.6)

- poligon yang dibentuk oleh $\{\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2\}$, disebut sebagai *control polygon*, mendekati bentuk kurva yang cukup baik.
- $\mathbf{P}_0 = \mathbf{C}(0)$ dan $\mathbf{P}_2 = \mathbf{C}(1)$.
- arah tangen kepada kurva yang berada di titik akhir paralel terhadap $\mathbf{P}_1 - \mathbf{P}_0$ dan $\mathbf{P}_2 - \mathbf{P}_1$.

- kurva berada di dalam segitiga yang dibentuk $\mathbf{P}_0\mathbf{P}_1\mathbf{P}_2$



Gambar 2.6: Kurva Bézier berderajat-2.

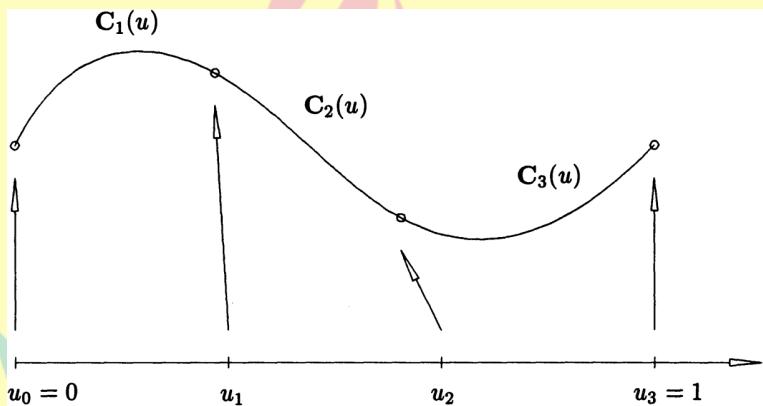
2.2.2 *B-Spline*

pada analisis numerik dalam matematika, *B-Spline* atau bisa disebut dengan *basis spline* merupakan fungsi spline dengan tunjangan yang sedikit dengan diberikan derajat, kehalusan, dan partisi domain. Kurva yang hanya berisikan satu polinomial atau segmentasi rasional yang sering tidak memadai, kekurangannya adalah:

- memerlukan *high degree*(interpolasi dengan derajat tinggi) untuk kompensasi banyak keterbatasan; misalnya, $(n - 1)$ -derajat diperlukan untuk melewatkkan kurva Bézier polinomial melalui n titik data. Namun, kurva tingkat tinggi tidak efisien untuk diproses dan secara numerik tidak stabil;
- Suatu *high degree* diperlukan untuk *fitting* bentuk(formula) kompleks secara akurat
- kurva segment tunggal (*surface* atau disebut permukaan) tidak cocok untuk desain bentuk(formula) interaktif; meskipun kurva Bézier dapat dibentuk melalui titik kontrolnya (dan bobot), kontrolnya tidak cukup lokal.

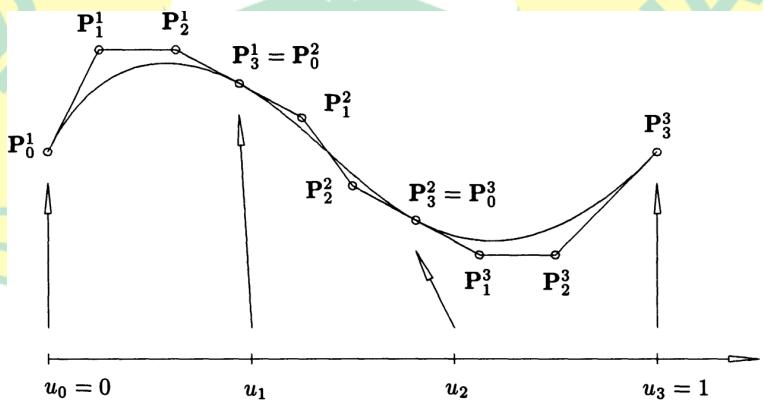
Solusinya adalah dengan menggunakan kurva (atau permukaan) yang *piecewise polynomial*(polinomial yang dipotong-potong), atau *piecewise*

rational(rasional yang dipotong-potong). Gambar (2.7) menunjukkan kurva $\mathbf{C}(u)$, yang terdiri dari $m (= 3)$ segmen polinomial berderajat- n . $\mathbf{C}(u)$ didefinisikan pada $u \in [0, 1]$. Nilai parameter $u_0 = 0 < u_1 < u_2 < u_3 = 1$ disebut *breakpoints*. Mereka memetakan ke titik akhir dari tiga segmen polinomial. ditunjukkan segmen sebagai $\mathbf{C}_i(u)$, $1 \leq i \leq m$. Segmen-segmen tersebut dibangun sedemikian rupa sehingga bergabung dengan tingkat kontinuitas tertentu (tidak harus sama di setiap *breakpoint*). Misalkan $\mathbf{C}_i^{(j)}$ menyatakan turunan ke- j dari \mathbf{C}_i . $\mathbf{C}(u)$ dikatakan \mathbf{C}^k kontinu pada *breakpoint* u_i jika $\mathbf{C}_i^{(j)}(u_i) = \mathbf{C}_{i+1}^{(j)}(u_i)$ untuk semua $0 \leq j \leq k$.



Gambar 2.7: Kurva polinomial kubik dipotong tiga segmen.

Setiap bentuk polinomial standar dapat digunakan untuk menyatakan bentuk $\mathbf{C}_i(u)$. Gambar (2.8) menunjukkan kurva Gambar (2.7) dengan tiga segmen dalam bentuk kubik Bézier. \mathbf{P}_i^j menunjukkan titik kontrol ke- i dari segmen ke- j .



Gambar 2.8: Kurva Gambar (2.7) ditunjukkan dengan segmen polinomial yang direpresentasikan dalam bentuk Bézier.

Jika derajatnya sama dengan tiga dan *breakpoints* $U = u_0, u_1, u_2, u_3$ tetap,

dan jika dua belas titik kontrol, \mathbf{P}_i^j , bervariasi secara sembarang, akan memperoleh ruang vektor, ν , yang terdiri dari semua potongan kurva polinomial kubik di U . ν memiliki dimensi dua belas, dan kurva di ν mungkin terputus-putus di u_1 atau u_2 . Sekarang misalkan ditetapkan (seperti pada Gambar (2.8)) bahwa $\mathbf{P}_3^1 = \mathbf{P}_0^2$ dan $\mathbf{P}_3^2 = \mathbf{P}_0^3$. Hal ini akan menghasilkan ν^0 , ruang vektor dari semua potongan kurva polinomial kubik pada U yang setidaknya C^0 kontinu di semua titik. ν^0 memiliki dimensi sepuluh, dan $\nu^0 \subset \nu$

Dari paragraf di atas, bisa dibuat kurva yang representatif dalam bentuk:

$$\mathbf{C}(u) = \sum_{i=0}^n f_i(u) \mathbf{P}_i \quad (2.3)$$

- \mathbf{P}_i adalah titik kontrol
- $f_i(u), i = 0, \dots, n$ adalah potongan fungsi polinomial yang membentuk basis untuk ruang vektor semua fungsi potongan polinomial dengan derajat dan kontinuitas yang diinginkan (untuk sebuah urutan breakpoint tetap, $U = u_i$, $0 \leq i \leq m$)

Perhatikan bahwa kontinuitas ditentukan oleh fungsi basis *basis function*, sehingga titik kontrol dapat dimodifikasi tanpa mengubah kontinuitas kurva. Selain itu, f_i harus memiliki sifat analitik bagus yang 'biasa'. Hal ini memastikan bahwa kurva yang ditentukan oleh Persamaan (2.3) memiliki sifat geometri yang mirip dengan kurva Bézier, misalnya *convex hull*, *variation diminishing*, *transformation invariance*. Properti penting lainnya yang dicari dalam fungsi basis ini adalah *local support*; ini menyiratkan bahwa setiap $f_i(u)$ bukan nol hanya pada sejumlah subinterval yang terbatas, bukan seluruh domain, $[u_0, u_m]$. Karena \mathbf{P}_i dikalikan dengan $f_i(u)$, pergerakan \mathbf{P}_i mempengaruhi bentuk kurva hanya pada subinterval di mana $f_i(u)$ bukan nol.

Ada banyak cara untuk mendefinisikan $f_i(u)$ pada rumus (2.3), di sini akan memakai cara *recurrence formula*. Misalkan $U = u_0, \dots, u_m$ adalah barisan bilangan real tak menurun, yaitu $u_i \leq u_i + 1$, $i = 0, \dots, m - 1$. u_i disebut *knots*, dan U adalah *knots vector*. Fungsi basis *B-spline* ke- i merupakan p -derajat (ordo

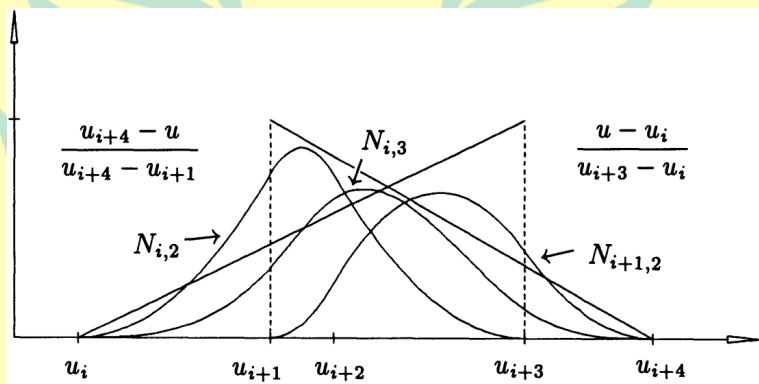
$p + l$), dilambangkan dengan $N_{i,p}(u)$, didefinisikan sebagai

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (2.4)$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

Dengan catatan :

- $N_{i,0}(u)$ merupakan fungsi langkah(*step function*) yang hampir semua hasilnya merupakan nol kecuali di dalam interval setengah terbuka $u \in [u_i, u_{i+1})$
- Untuk $p > 0$, $N_{i,p}(u)$ merupakan kombinasi linear dari dua fungsi dasar $(p - 1)$ -derajat. (gambar 2.9)



Gambar 2.9: definisi rekursif dari rumus dasar B-spline.

- Komputasi dari kumpulan fungsi dasar memerlukan spesifikasi dari sebuah *knot vector*, U , dan derajatnya, p .
- Rumus (2.4) bisa menghasilkan 0/0; nanti hasilnya diubah menjadi nol.
- $N_{i,p}(u)$ adalah potongan polinomial, yang didefinisikan pada seluruh garis ril; umumnya hanya pada interval $[u_0, u_m]$ yang diperhatikan.
- interval setengah terbuka, $[u_i, u_{i+1})$, disebut *knot span* ke- i ; panjangnya bisa nol, karena *knot* tidak perlu dibedakan;

- perhitungan fungsi derajat ke- p menghasilkan tabel segitiga terpotong

| | | | |
|-----------|-----------|-----------|-----------|
| | | $N_{0,1}$ | |
| $N_{1,0}$ | | $N_{0,2}$ | |
| | $N_{1,1}$ | | $N_{0,3}$ |
| $N_{2,0}$ | | $N_{1,2}$ | |
| | $N_{2,1}$ | | $N_{1,3}$ |
| $N_{3,0}$ | | $N_{2,2}$ | \vdots |
| | $N_{3,1}$ | | \vdots |
| $N_{4,0}$ | \vdots | | |

Dengan seluruh hal di atas, penulis mendapatkan rumus dasar untuk merubah suatu kurva acak menjadi kurva yang memiliki kontinuitas. Bahasan berikut akan menyempurnakan rumus dasar untuk memiliki kontinuitas yang lebih lanjut.

2.2.3 Global Interpolation

Kebanyakan algoritma interpolasi terbagi dalam salah satu dari dua kategori: global atau lokal. Dengan algoritma global, sistem masalah pada persamaan atau optimasi sudah diatur dan diselesaikan. Jika data yang diberikan hanya terdiri dari titik dan turunan, dan jika yang tidak diketahui hanya titik kontrol (derajat, *knot*, dan bobot telah dipilih sebelumnya), maka sistem tersebut linier dan karenanya akan mudah diselesaikan. Jika data yang lebih esoterik, seperti kelengkungan diketahui, atau *knot* dan/atau bobot juga tidak diketahui sistemnya, maka sistem yang dihasilkan adalah nonlinier. Secara teoritis, gangguan pada *input item data* siapa pun dapat mengubah bentuk keseluruhan kurva atau permukaan; namun, besarnya perubahan menurun seiring bertambahnya jarak dari *item data* yang terpengaruh. Algoritma lokal lebih bersifat geometris, membangun kurva atau segmen permukaan, hanya menggunakan data lokal untuk setiap langkah. Gangguan pada item data hanya mengubah kurva atau permukaan secara lokal. Algoritme ini biasanya lebih murah secara komputasi dibandingkan metode global. Mereka juga dapat menangani titik puncak, segmen garis lurus, dan anomali data lokal lainnya dengan lebih baik. Namun, mencapai tingkat kontinuitas yang diinginkan pada batas segmen lebih menyusahkan, dan metode lokal sering kali menghasilkan banyak *interior knot*.

Berikut merupakan pembentukan rumus interpolasi global. Misalkan penulis

diberikan sekumpulan titik \mathbf{Q}_k , $k = 0, \dots, n$, dan penulis ingin menginterpolasi titik-titik ini dengan kurva *B-spline* nonrasional p -derajat. Jika ditetapkan nilai parameter, \bar{u}_k , pada masing-masing \mathbf{Q}_k , dan memilih *knot vector* yang sesuai $U = u_0, \dots, u_m$, dapat disiapkan sistem persamaan linier $(n + 1)$ variabel dan $(n + 1)$ persamaan:

$$\mathbf{Q}_k = \mathbf{C}(u) = \sum_{i=0}^n N_{i,p}(\bar{u}_k) \mathbf{P}_i \quad (2.5)$$

Titik kontrol, \mathbf{P}_i , adalah variabel $n + 1$ yang tidak diketahui. r merupakan jumlah koordinat dalam \mathbf{Q}_k (biasanya 2, 3, atau 4). Perhatikan bahwa metode ini tidak bergantung pada r ; Persamaan (2.5) mempunyai satu koefisien matriks, dengan r sisi kanan dan, dengan demikian, r merupakan himpunan solusi untuk r koordinat dari \mathbf{P}_i .

Masalah berada dalam pemilihan \bar{u}_k dan U , dan pilihannya mempengaruhi bentuk dan parameterisasi kurva. Sepanjang bagian ini diasumsikan bahwa parameterinya terletak pada rentang $U \in [0, 1]$. Tiga metode umum dalam memilih \bar{u}_k adalah:

- *equally spaced*:

$$\begin{aligned} \bar{u}_0 &= 0 & \bar{u}_n &= 1 \\ \bar{u}_k &= \frac{k}{n} & k &= 1, \dots, n - 1 \end{aligned} \quad (2.6)$$

Metode ini tidak direkomendasikan dikarenakan bisa membuat bentuk yang tak menentu (seperti *loops*) ketika datanya tidak berjarak sama

- *chord length*: diumpamakan d merupakan jumlah chord length

$$d = \sum_{k=1}^n |\mathbf{Q}_k - \mathbf{Q}_{k-1}| \quad (2.7)$$

lalu

$$\bar{u}_0 = 0 \quad \bar{u}_n = 1$$

$$\bar{u}_k = \bar{u}_{k-1} + \frac{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}{d} \quad k = 1, \dots, n - 1 \quad (2.8)$$

Ini adalah metode yang paling banyak digunakan, dan secara umum sudah memadai. Hal ini juga memberikan parameterisasi yang "bagus" pada kurva,

dalam arti mendekati parameterisasi yang seragam.

- metode *centripetal*: umpamakan

$$d = \sum_{k=1}^n \sqrt{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}$$

$$\bar{u}_0 = 0 \quad \bar{u}_n = 1$$

$$\bar{u}_k = \bar{u}_{k-1} + \frac{\sqrt{|\mathbf{Q}_k - \mathbf{Q}_{k-1}|}}{d} \quad k = 1, \dots, n-1 \quad (2.9)$$

Knots bisa berjarak sama, yaitu:

$$u_0 = \dots = u_p = 0 \quad u_{m-p} = \dots = u_m = 1$$

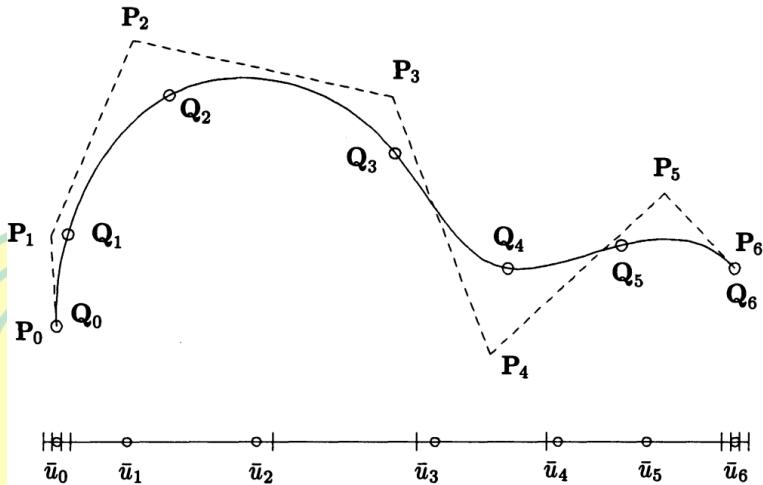
$$u_{j+p} = \frac{j}{n-p+1} \quad j = 1, \dots, n-p \quad (2.10)$$

Namun, cara ini tidak disarankan, jika digunakan bersama dengan Persamaan (2.8) atau (2.9) dapat menghasilkan sistem persamaan tunggal pada Persamaan 2.5. untuk mengatasinya ada teknik rata-rata sebagai berikut

$$u_0 = \dots = u_p = 0 \quad u_{m-p} = \dots = u_m = 1$$

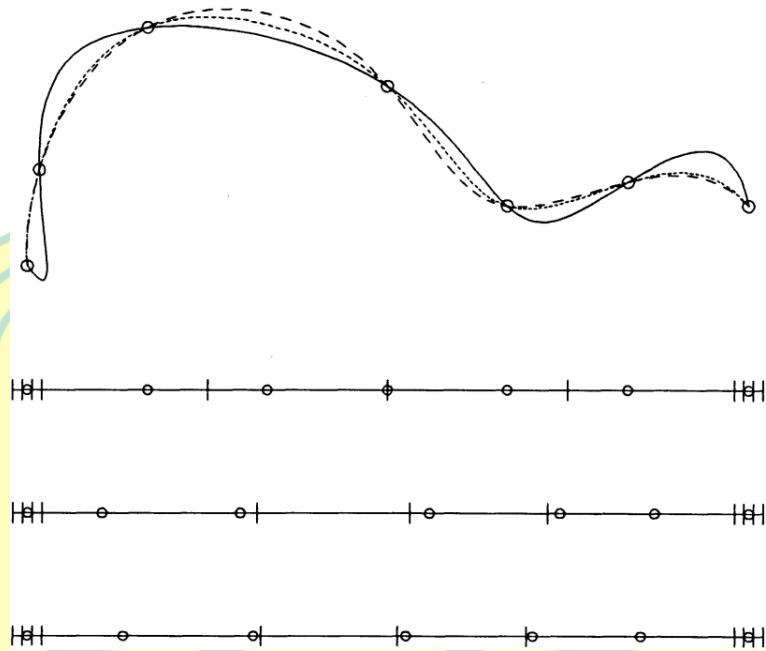
$$u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p-1} \bar{u}_i \quad j = 1, \dots, n-p \quad (2.11)$$

Dengan metode ini knots mencerminkan distribusi \bar{u}_k . Selanjutnya menggunakan Persamaan (2.11) dikombinasikan dengan Persamaan (2.8) atau (2.9) untuk menghitung \bar{u}_k mengarah ke sistem (Persamaan 2.5) yang benar-benar positif dan terikat dengan *semibandwidth* kurang dari p , yaitu, $N_{i,p}(\bar{u}_k) = 0$ jika $|i - k| \geq p$. Oleh karena itu, penyelesaiannya dapat dilakukan dengan eliminasi Gaussian tanpa melakukan *pivoting*.

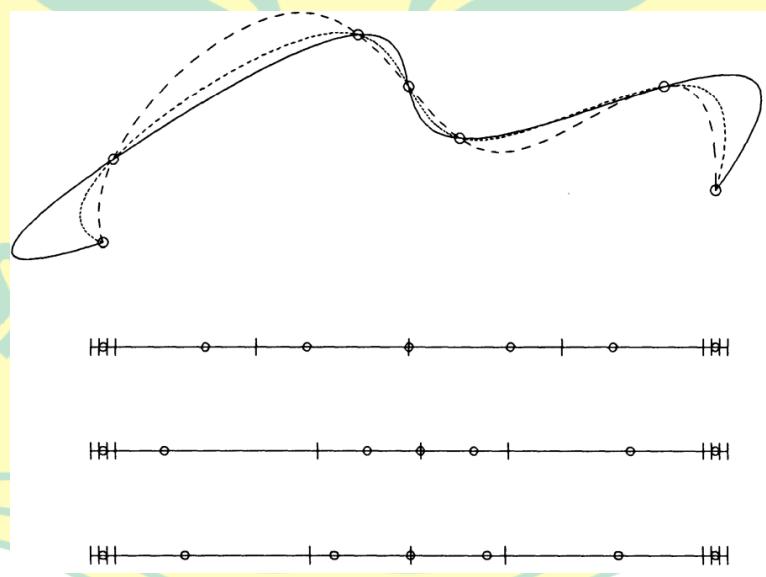


Gambar 2.10: Contoh interpolasi kurva menggunakan parameterisasi *chord length* dan *knot vector* yang diperoleh dengan rata-rata parameter.

Gambar (2.10) menunjukkan titik kontrol, parameter, dan *knot vector* dari kurva kubik yang menginterpolasi tujuh titik. Parameter ditentukan dengan metode *chord length*, dan *knot* diperoleh dengan merata-ratakan parameternya (Persamaan 2.11). Pada Gambar (2.11) diilustrasikan perbandingan parameterisasi yang berbeda. Gambar (2.12) menunjukkan perbandingan yang sama dengan menggunakan lebih banyak titik data yang tersebar secara "liar". Dalam kedua kasus tersebut, kurva kubik dilewatkan melalui tujuh titik, menggunakan parameter seragam dan knot seragam (kurva padat dan knot vector atas - lihat Persamaan [2.6] dan [2.10]); parameter *chord length* dan *knot* yang diperoleh dengan rata-rata (kurva putus-putus dan *knot vector* tengah - lihat Persamaan [2.8] dan [2.11]); dan parameter sentripetal serta knot yang diperoleh dengan rata-rata (kurva titik-titik dan *knot vector* bawah - lihat Persamaan [2.9] dan [2.11]). Perhatikan akan Gambar (2.12) bagaimana *chord length* dan kurva parameter sentripetal beradaptasi dengan perubahan jarak titik.



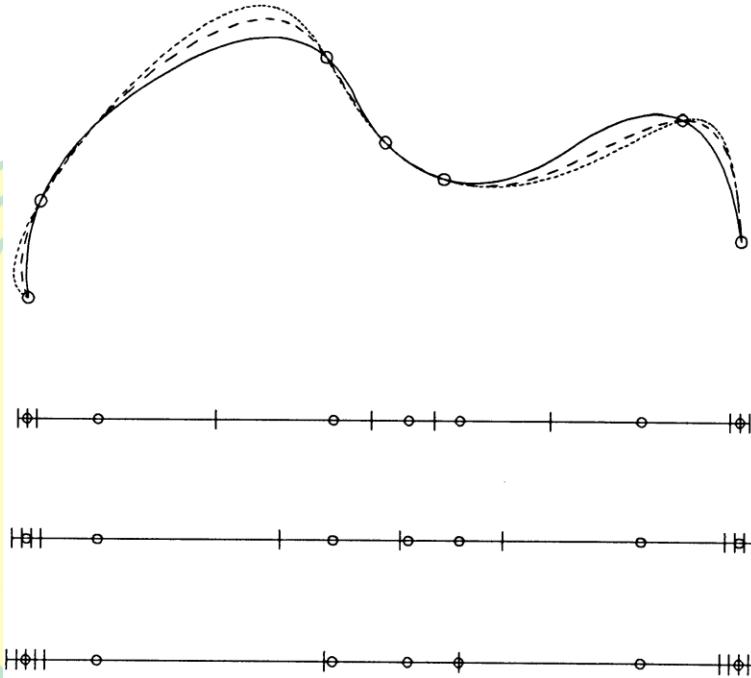
Gambar 2.11: Contoh interpolasi kurva dengan parameterisasi dan *knot vector* berbeda.



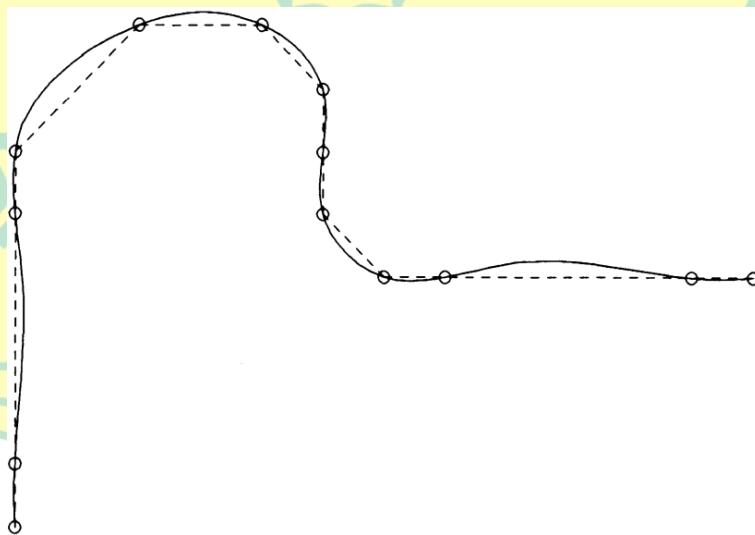
Gambar 2.12: Contoh interpolasi kurva dengan parameterisasi dan *knot vector* berbeda.

Gambar (2.13) mengilustrasikan interpolasi dengan derajat yang berbeda-beda; kurva padat, putus-putus, dan putus-putus masing-masing memiliki derajat 2, 3, dan 4. Gambar (2.14) menunjukkan ketidakmampuan interpolasi global

untuk menangani kumpulan titik kolinear.



Gambar 2.13: Interpolasi kurva dengan derajat berbeda menggunakan parameterisasi chord length dan knot yang diperoleh dengan rata-rata.



Gambar 2.14: Interpolasi kurva kubik global ke data yang berisi titik-titik kolinear (lihat garis putus-putus).

Setelah \bar{u}_k dan *knot* dihitung, matriks koefisien $(n + 1) \times (n + 1)$ dari sistem

(Persamaan 2.5) dibuat dengan mengevaluasi fungsi basis bukan nol pada setiap $\bar{u}_k, k = 0, \dots, n$.

Contoh:

Misalkan $\mathbf{Q}_k = (0, 0), (3, 4), (-1, 4), (-4, 0), (-4, -3)$, dan asumsikan penulis ingin menginterpolasi \mathbf{Q}_k dengan kurva kubik. Digunakan persamaan (2.8) dan (2.11) untuk menghitung \bar{u}_k dan u_j , dan kemudian membuat sistem persamaan linier, Persamaan (2.5). *chord lengths* yang terpisah adalah

$$|\mathbf{Q}_1 - \mathbf{Q}_0| = 5 \quad |\mathbf{Q}_2 - \mathbf{Q}_1| = 4 \quad |\mathbf{Q}_3 - \mathbf{Q}_2| = 5 \quad |\mathbf{Q}_4 - \mathbf{Q}_3| = 3$$

Dan jumlah *chord length* menjadi $d = 17$, maka

$$\bar{u}_0 = 0 \quad \bar{u}_1 = \frac{5}{17} \quad \bar{u}_2 = \frac{9}{17} \quad \bar{u}_3 = \frac{14}{17} \quad \bar{u}_4 = 1$$

Gunakan rumus (2.11)

$$u_4 = \frac{1}{3} \left(\frac{5}{17} + \frac{9}{17} + \frac{14}{17} \right) = \frac{28}{51}$$

maka

$$U = \{0, 0, 0, 0, \frac{28}{51}, 1, 1, 1, 1\}$$

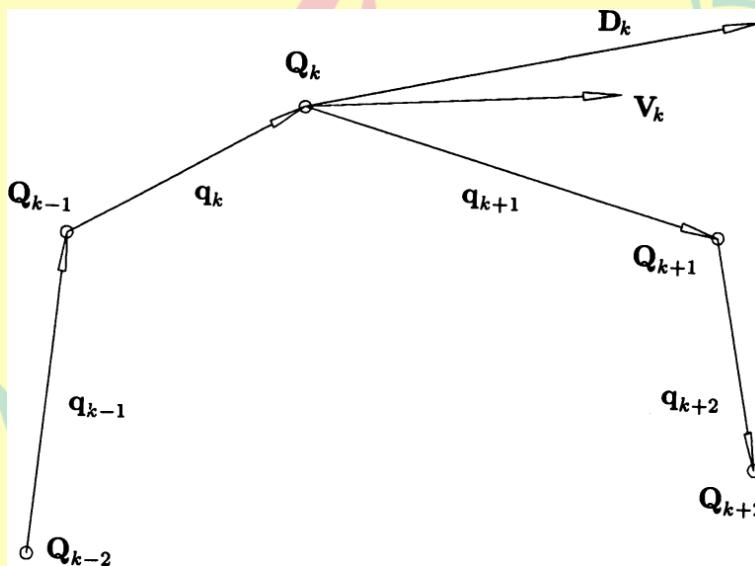
Sistem persamaan linearnya adalah

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ N_{0,3}\left(\frac{5}{17}\right) & N_{1,3}\left(\frac{5}{17}\right) & N_{2,3}\left(\frac{5}{17}\right) & N_{3,3}\left(\frac{5}{17}\right) & 0 \\ N_{0,3}\left(\frac{9}{17}\right) & N_{1,3}\left(\frac{9}{17}\right) & N_{2,3}\left(\frac{9}{17}\right) & N_{3,3}\left(\frac{9}{17}\right) & 0 \\ 0 & N_{1,3}\left(\frac{14}{17}\right) & N_{2,3}\left(\frac{14}{17}\right) & N_{3,3}\left(\frac{14}{17}\right) & N_{4,3}\left(\frac{14}{17}\right) \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \\ \mathbf{P}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_0 \\ \mathbf{Q}_1 \\ \mathbf{Q}_2 \\ \mathbf{Q}_3 \\ \mathbf{Q}_4 \end{bmatrix}$$

2.2.4 Local Interpolation

Seperti yang dikatakan sebelumnya, interpolasi global tidak memberikan hasil yang sempurna, karena rumus nya terlalu mengeneralisir pemindahan *knot vector*. Untuk mengatasi hal itu, digunakan interpolasi lokal. Misalkan Q_k , $k = 0, \dots, n$, diberikan. Yang dimaksud dengan interpolasi kurva lokal adalah metode yang membentuk n segmen kurva polinomial atau rasional,

$C_i(u), i = 0, \dots, n - 1$, sehingga Q_i dan Q_{i+1} adalah titik akhir dari $C_i(u)$. Segmen-segmen yang berdekatan digabungkan dengan tingkat kontinuitas tertentu, dan konstruksi berlangsung berdasarkan segmen, umumnya dari kiri ke kanan. Persamaan apa pun yang muncul bersifat lokal hanya pada beberapa segmen yang berdekatan. Dalam *framework* NURBS, mereka membuat segmen menggunakan kurva Bézier polinomial atau rasional, kemudian memperoleh kurva NURBS dengan memilih *knot vector* yang sesuai.



Gambar 2.15: Perhitungan vektor tangen (\mathbf{V}_k) dan turunan (\mathbf{D}_k) untuk interpolasi kurva lokal.

Sekarang misalkan \bar{u}_i menyatakan parameter awal $\mathbf{C}_i(u)$ dan parameter akhir $\mathbf{C}_{i-1}(u)$ dan $\mathbf{C}_i(u)$ bertemu di \bar{u}_i dengan kontinuitas G^1 (G untuk geometri) jika arah singgungnya berimpit di sana, yaitu jika $\mathbf{C}'_i(\bar{u}_i)$ dan $\mathbf{C}'_{i-1}(\bar{u}_i)$ menunjuk ke arah yang sama. Namun, besarnya mungkin berbeda. Kontinuitas G^1 menyiratkan bahwa kurva tersebut secara visual kontinu (halus) tetapi mungkin memiliki diskontinuitas dalam parameterisasinya. Algoritma interpolasi lokal dirancang untuk memberikan tingkat kontinuitas G atau C tertentu; dalam sub bab ini hanya menyajikan algoritma yang menyediakan kontinuitas G^1 atau C^1 . Metode lokal yang menghasilkan kontinuitas lebih tinggi tidak banyak digunakan. Meskipun G^2 dan C^2 dimungkinkan dengan kubik, derajat yang lebih tinggi harus digunakan jika fleksibilitas yang wajar ingin dipertahankan.

Untuk mendapatkan segmen Bézier, $\mathbf{C}_i(u)$, memerlukan komputasi titik kontrol Bézier bagian dalam, satu titik untuk kuadrat, dua untuk kubik. Titik kendali

ini terletak pada garis singgung(*tangent*) kurva di \mathbf{Q}_k ; oleh karena itu, diperlukan vektor singgung(*tangent vector*) \mathbf{T}_k pada setiap \mathbf{Q}_k . Dalam beberapa kasus, vektor tersebut dapat dimasukkan bersama dengan \mathbf{Q}_k ; misalnya, persamaan tersebut mudah diperoleh saat menghitung titik potong antara dua permukaan. Namun, jika tidak dimasukkan maka harus dihitung sebagai bagian dari algoritma interpolasi. Ada sejumlah metode; Boehm [Böhm et al., 1984] memberikan survei tentang berbagai metode. Misalkan

$$\Delta \bar{u}_k = \bar{u}_k - \bar{u}_{k-1} \quad \mathbf{q}_k = \mathbf{Q}_k - \mathbf{Q}_{k-1} \quad \mathbf{d}_k = \frac{\mathbf{q}_k}{\Delta \bar{u}_k}$$

Semua metode memiliki satu dari dua bentuk

$$\mathbf{D}_k = (1 - \alpha_k) \mathbf{d}_k = \alpha_k \mathbf{d}_{k+1} \quad (2.12)$$

atau

$$\mathbf{T}_k = \frac{\mathbf{V}_k}{|\mathbf{V}_k|} \quad \mathbf{V}_k = (1 - \alpha_k) \mathbf{q}_k + \alpha_k \mathbf{q}_{k+1} \quad (2.13)$$

(lihat Gambar 2.15). Perhatikan bahwa Persamaan (2.12) mengasumsikan bahwa nilai \bar{u}_k telah ditetapkan. Vektor \mathbf{D}_k dapat dipandang sebagai perkiraan turunannya. Persamaan (2.13) tidak menggunakan parameter \bar{u}_k dan vektor yang dihasilkan harus dipandang sebagai arah singgung saja. Penetapan besaran dan parameter harus dilakukan bersamaan satu sama lain karena tidak independen. Perhatikan juga bahwa ini menggunakan notasi \mathbf{T} hanya untuk vektor singgung satuan panjang. Persamaan (2.12) dan (2.13) merupakan interpolasi linier. Berbagai skema berbeda dalam cara mereka mengkomputasi parameter interpolasi α_k yang biasanya bergantung pada tiga atau lima titik tetangga. Misalnya, metode Bessel [Boor, 1978] adalah metode tiga titik menggunakan

$$\alpha_k = \frac{\Delta \bar{u}_k}{\Delta \bar{u}_k + \Delta \bar{u}_{k+1}} \quad k = 1, \dots, n-1 \quad (2.14)$$

Bersama dengan persamaan (2.12) digabungkan menjadi

$$\alpha_k = \frac{|\mathbf{q}_{k-1} * \mathbf{q}_k|}{|\mathbf{q}_{k-1} * \mathbf{q}_k| + |\mathbf{q}_{k+1} * \mathbf{q}_{k+2}|} \quad k = 2, \dots, n-2 \quad (2.15)$$

bersama dengan Persamaan (2.13) menghasilkan metode lima poin untuk memperoleh \mathbf{T}_k . Keuntungannya adalah tiga titik yang segaris, $\mathbf{Q}_{k-1}, \mathbf{Q}_k, \mathbf{Q}_{k+1}$,

menghasilkan \mathbf{T}_k yang sejajar dengan ruas garis. Penyebut Persamaan (2.15) hilang jika $\mathbf{Q}_{k-2}, \mathbf{Q}_{k-1}, \mathbf{Q}_k$ segaris dan $\mathbf{Q}_k, \mathbf{Q}_{k+1}, \mathbf{Q}_{k+2}$ segaris. Ini menyiratkan antara adanya sudut di \mathbf{Q}_k atau ada segmen garis lurus dari \mathbf{Q}_{k-2} sampai \mathbf{Q}_{k+2} . Dalam kasus ini α_k dapat didefinisikan dalam beberapa cara; yaitu

- $\alpha_k = 1$, yang berarti $\mathbf{V}_k = \mathbf{q}_{k+1}$, ini menghasilkan sudut di \mathbf{Q}_k jika tersirat dalam data
- $\alpha_k = 1/2$, yang berarti $\mathbf{V}_k = \frac{1}{2}(\mathbf{q}_k + \mathbf{q}_{k+1})$; pilihan ini menghaluskan suatu sudut jika tersirat.

Berdasarkan cara di atas, rutinitas interpolasi kurva lokal dapat diterima dengan tanda masukan(*flag*), yang menunjukkan apakah akan dipertahankan sudutnya atau tidak. Semua metode memerlukan perlakuan khusus pada ujungnya. Untuk skema tiga titik bisa dibuat

$$\mathbf{D}_0 = 2\mathbf{d}_1 - \mathbf{D}_1 \quad \mathbf{D}_n = 2\mathbf{d}_n - \mathbf{D}_{n-1} \quad (2.16)$$

Dan untuk skema lima titik bisa dipakai

$$\begin{aligned} \mathbf{q}_0 &= 2\mathbf{q}_1 - \mathbf{q}_2 & \mathbf{q}_{-1} &= 2\mathbf{q}_0 - \mathbf{q}_1 \\ \mathbf{q}_{n+1} &= 2\mathbf{q}_n - \mathbf{q}_{n-1} & \mathbf{q}_{n+2} &= 2\mathbf{q}_{n+1} - \mathbf{q}_n \end{aligned} \quad (2.17)$$

untuk dimasukkan ke dalam Persamaan (2.15) dan (2.13) sehingga diperoleh $\mathbf{T}_0, \mathbf{T}_1$ dan $\mathbf{T}_{n-1}, \mathbf{T}_n$. selanjutnya akan menjelaskan interpolasi permukaan lokal bikubik

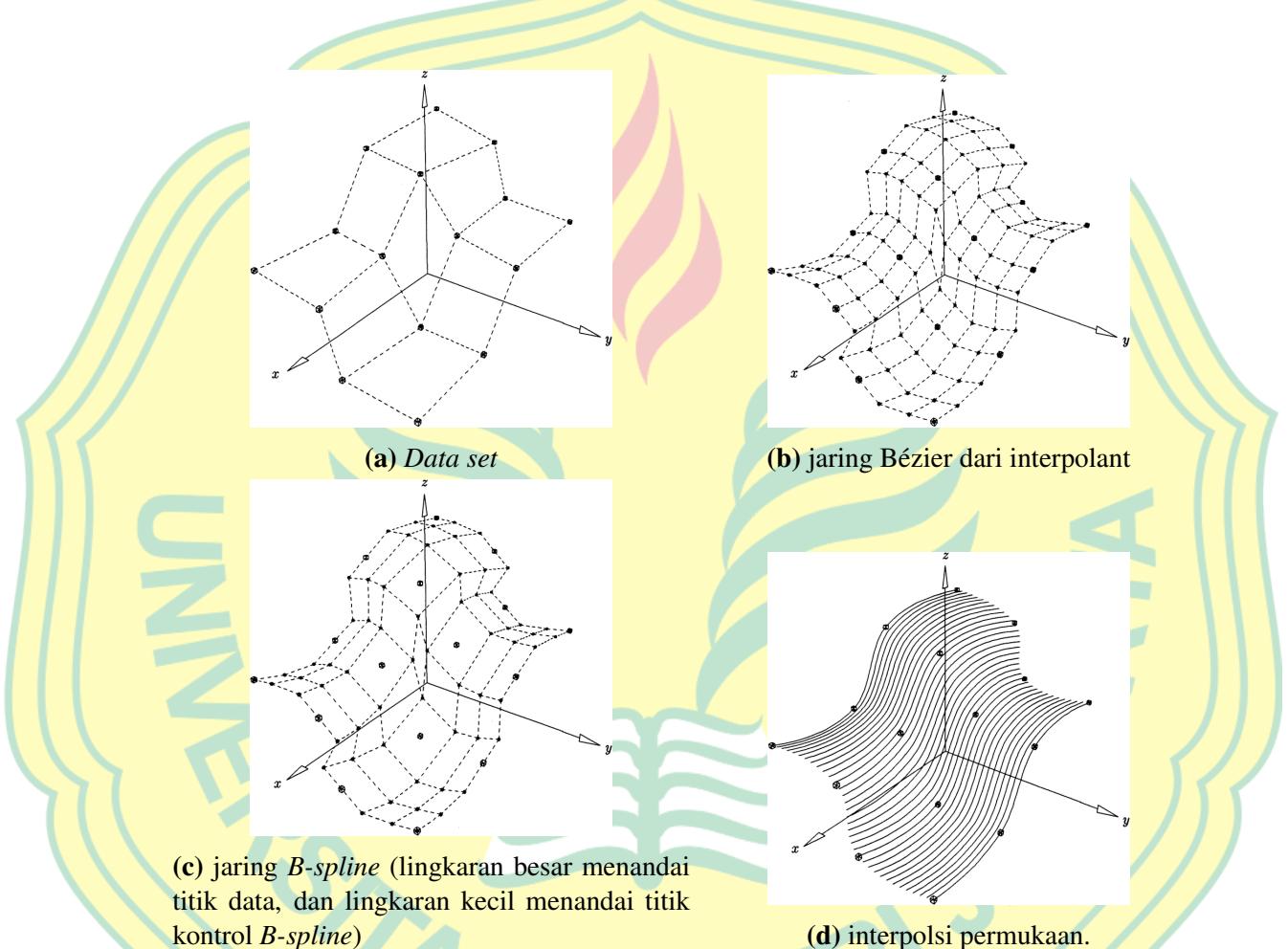
Misalkan $\mathbf{Q}_{k,l}$, $k = 0, \dots, n$ dan $l = 0, \dots, m$, adalah himpunan titik data, dan misalkan (\bar{u}_k, \bar{v}_l) adalah pasangan parameter yang bersesuaian, dihitung dengan rata-rata chord length (seperti pada Algoritma (2.8)). Metode berikut menghasilkan permukaan bikubik, $\mathbf{S}(u, v)$, yaitu

$$\mathbf{S}(\bar{u}_k, \bar{v}_l) = \sum_{i=0}^{2n+1} \sum_{j=0}^{2m+1} N_{i,3}(\bar{u}_k) N_{j,3}(\bar{v}_l) \mathbf{P}_{i,j} \quad (2.18)$$

diperoleh permukaan dengan membuat tambalan Bézier bikubik nm , $\{\mathbf{B}_{k,l}(u, v)\}$, $k = 0, \dots, n-1$, $l = 0, \dots, m-1$, di mana $\mathbf{Q}_{k,l}, \mathbf{Q}_{k+1,l}, \mathbf{Q}_{k,l+1}, \mathbf{Q}_{k+1,l+1}$ adalah titik sudut tambalan, dan tambalan bergabung dengan kontinuitas $C^{1,1}$ melintasi batasnya. Kecuali batas permukaan, semua baris dan kolom titik kontrol yang berisi $\mathbf{Q}_{k,l}$ asli dihilangkan (lihat Gambar (2.16b) dan (2.16c)), menyisakan

$(2n + 2)(2m + 2)$ titik kontrol di *B-spline* permukaan *spline*. *Knot vector*-nya adalah

$$\begin{aligned} U &= \{0, 0, 0, 0, \bar{u}_1, \bar{u}_1, \bar{u}_2, \bar{u}_2, \dots, \bar{u}_{n-1}, \bar{u}_{n-1}, 1, 1, 1, 1\} \\ V &= \{0, 0, 0, 0, \bar{v}_1, \bar{v}_1, \bar{v}_2, \bar{v}_2, \dots, \bar{v}_{m-1}, \bar{v}_{m-1}, 1, 1, 1, 1\} \end{aligned} \quad (2.19)$$



Gambar 2.16: $C^{(1,1)}$ Interpolasi permukaan bikubik lokal

Tambalan bikubik Bézier memiliki 16 titik kontrol. 12 titik kontrol perbatasan diperoleh dengan awalnya melakukan looping melalui $m + 1$ baris dan $n + 1$ kolom data dan menggunakan skema interpolasi kurva kubik. Skemanya sedikit berbeda dengan detail sebelumnya, karena sudah mempunyai parameterinya (\bar{u}_k, \bar{v}_k). Perhatikan bahwa ini harus dikomputasi terlebih dahulu, karena semua baris (kolom) harus memiliki parameterisasi yang sama pada permukaan produk tensor. Dengan demikian, masih dapat memaksakan kontinuitas C^1 pada titik akhir segmen, namun tidak dapat dipaksakan kecepatan yang sama pada titik tengah

segmen kurva Bézier. Lebih spesifik lagi, misalkan $l = l_0$ tetap, dan perhatikan kurva kubik yang menginterpolasi titik-titik $\mathbf{Q}_{0,l_0}, \dots, \mathbf{Q}_{n,l_0}$. Misalkan r_{l_0} menyatakan jumlah chord length pada baris ke- l_0 . Pada setiap titik, \mathbf{Q}_{k,l_0} , hitung \mathbf{T}_{k,l_0}^u , (satuan tangen pada arah u) seperti sebelumnya (Persamaan (2.13), (2.15), dan (2.17)). Kemudian titik-titik *interior* Bézier pada baris ini dikomputasi dengan

$$\mathbf{P}_{1,0}^{k,l_0} = \mathbf{Q}_{k,l_0} + a\mathbf{T}_{k,l_0}^u \quad \mathbf{P}_{2,0}^{k,l_0} = \mathbf{Q}_{k+1,l_0} + a\mathbf{T}_{k+1,l_0}^u \quad (2.20)$$

dimana

$$a = \frac{r_{l_0}(\bar{u}_{k+1} - \bar{u}_k)}{3} = \frac{r_{l_0}\Delta\bar{u}_{k+1}}{3}$$

Kurva yang dihasilkan adalah kontinu C^1 , dengan besaran turunan sama dengan r_{l_0} pada semua \mathbf{Q}_{k,l_0} . Teknik yang sama diterapkan pada $n + 1$ kolom data.

ini kembali pada menghitung *interior* empat titik kontrol dari setiap patch Bézier. Hal ini memerlukan estimasi untuk turunan parsial campuran, $\mathbf{D}_{k,l}^{uv}$, pada setiap $\mathbf{Q}_{k,l}$. diperoleh rumus untuk $\mathbf{Q}_{k,l}$ berdasarkan metode tiga titik Bessel (Persamaan (2.12), (2.14), dan (2.16)). Misalkan r_l dan s_k menyatakan jumlah *chord length* pada baris ke- l (kolom ke- k). Kemudian

$$\mathbf{D}_{k,l}^u = r_l \mathbf{T}_{k,l}^u \quad \mathbf{D}_{k,l}^v = s_k \mathbf{T}_{k,l}^v \quad (2.21)$$

lalu bentuk

$$\mathbf{d}_{k,l}^{vu} = (1 - \alpha_k) \frac{\mathbf{D}_{k,l}^v - \mathbf{D}_{k-1,l}^v}{\Delta\bar{u}_k} + \alpha_k \frac{\mathbf{D}_{k+1,l}^v - \mathbf{D}_{k,l}^v}{\Delta\bar{u}_{k+1}}$$

dan

$$\mathbf{d}_{k,l}^{uv} = (1 - \beta_l) \frac{\mathbf{D}_{k,l}^u - \mathbf{D}_{k,l-1}^u}{\Delta\bar{v}_l} + \beta_l \frac{\mathbf{D}_{k,l+1}^u - \mathbf{D}_{k,l}^u}{\Delta\bar{v}_{l+1}}$$

dengan

$$\alpha_k = \frac{\Delta\bar{u}_k}{\Delta\bar{u}_k + \Delta\bar{u}_{k+1}} \quad \beta_l = \frac{\Delta\bar{v}_l}{\Delta\bar{v}_l + \Delta\bar{v}_{l+1}}$$

menjadi

$$\mathbf{D}_{k,l}^{uv} = \frac{\alpha_k \mathbf{d}_{k,l}^{uv} + \beta_l \mathbf{d}_{k,l}^{vu}}{\alpha_k + \beta_l} \quad (2.22)$$

Rumus akhir yang sesuai (Persamaan (2.16)) harus digunakan pada pembatasnya. Empat *interior* titik kontrol pada tambalan ke- (k, l) sekarang dihitung

menggunakan Persamaan. (2.22)

$$\begin{aligned}\mathbf{P}_{1,1}^{k,l} &= \gamma \mathbf{D}_{k,l}^{uv} + \mathbf{P}_{0,1}^{k,l} + \mathbf{P}_{1,0}^{k,l} - \mathbf{P}_{0,0}^{k,l} \\ \mathbf{P}_{2,1}^{k,l} &= -\gamma \mathbf{D}_{k+1,l}^{uv} + \mathbf{P}_{3,1}^{k,l} - \mathbf{P}_{3,0}^{k,l} + \mathbf{P}_{2,0}^{k,l} \\ \mathbf{P}_{1,2}^{k,l} &= -\gamma \mathbf{D}_{k,l+1}^{uv} + \mathbf{P}_{1,3}^{k,l} - \mathbf{P}_{0,3}^{k,l} + \mathbf{P}_{0,2}^{k,l} \\ \mathbf{P}_{2,2}^{k,l} &= \gamma \mathbf{D}_{k+1,l+1}^{uv} + \mathbf{P}_{2,3}^{k,l} + \mathbf{P}_{3,2}^{k,l} - \mathbf{P}_{3,3}^{k,l}\end{aligned}\tag{2.23}$$

dimana

$$\gamma = \frac{\Delta \bar{u}_k + 1 \Delta \bar{v}_l + 1}{9}$$

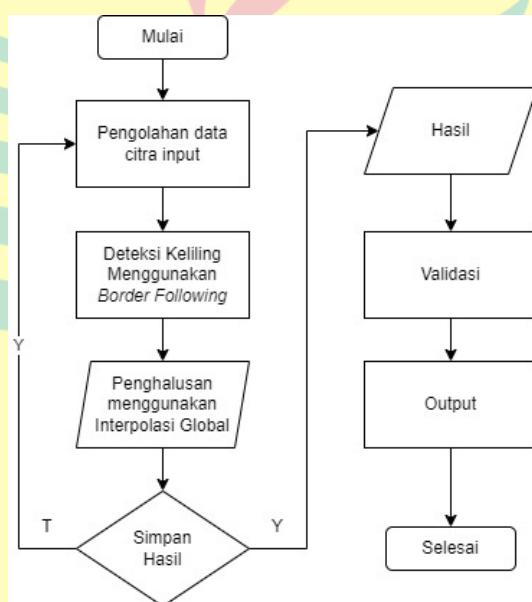


BAB III

METODOLOGI PENELITIAN

3.1 Perancangan Sistem

Penulis merancang sistem ini berdasarkan dari penelitian Rizki, yaitu untuk pendekripsi luka. Yang membuat penelitian ini berbeda dengan penelitian Rizki adalah interpolasi yang dipakai adalah buatan penulis dengan referensi NURBS, lalu menggantikan *active contour* dengan *border following* yang ditulis oleh Hafizun Alim(Alim, 2023). berikut merupakan diagram alir penelitian yang akan dilakukan.



Gambar 3.1: Diagram alir penelitian

Algorithm 1 Main

Require: Image, p **Ensure:** Curve Points

```

1: function MAIN(Image,  $p$ )
2:   image  $\leftarrow$  imread(Image)                                 $\triangleright$  read image as value
3:   points  $\leftarrow$  BorderFollowing(image)                       $\triangleright$  algorithm(2)
4:   curve  $\leftarrow$  GlobalInterpolation(points,  $p$ )            $\triangleright$  algorithm(6)
5:   Results  $\leftarrow$  Image + curve                           $\triangleright$  put curve to image
6:   return Results
7: end function

```

3.2 Border Following

Setelah citra diolah

Setelah merubah gambarnya dengan Interpolasi, penulis akan menggunakan teknik *border following* untuk membuat garis pada gambar. *Border following* merupakan teknik yang diusulkan oleh Satoshi Suzuki dan Keichi Abe di mana teknik ini menelusuri batas suatu objek dengan mengikuti transisi antara piksel latar depan (objek) dan latar belakang pada gambar lalu memberikan definisi derajat pada piksel. Algoritma *border following* bekerja dengan memulai pada titik tertentu pada batas suatu objek dan kemudian mengikuti batas tersebut searah atau berlawanan arah jarum jam dengan mengidentifikasi titik berikutnya di sepanjang batas tersebut.

Algoritma ini dimulai dengan mengasumsikan citra yang dimasukkan merupakan citra biner dengan latar belakang merupakan piksel bernilai 0 dan latar depan (objek) sebagai piksel bernilai 1. Lalu dilanjutkan dengan memberi masukan berupa citra biner dari proses operasi morfologi yang dilanjutkan dengan proses memindai(scan) dimulai dari piksel paling kiri atas. Apabila sudah memindai sampai piksel (i, j) bernilai $f_{i,j} \neq 0$ (bukan latar belakang), ditentukan piksel tersebut apakah merupakan *starting point* dari operasi *border following* untuk tepi luar atau tepi dalam (gambar 2.2). Lalu menentukan *parent border* untuk piksel (i, j) berdasarkan gambar (2.5). Dilanjutkan dengan melakukan *border following* dimulai dari *starting point* piksel (i, j) sampai *pointer* kembali ke posisi *starting point*. Algoritma yang dipakai adalah algoritma kedua Suzuki(Satoshi Suzuki, 1985) maka nilai NBD dan $-NBD$ akan di-set masing-masing 2 dan -2. lalu pemindai akan melanjutkan pemindaian untuk mencari objek ($f_{i,j} \neq 0$). Algoritma ini selesai

apabila pemindai sudah sampai sudut kanan bawah.

Algorithm 2 Main Border Following

Require: image, start, previous

Ensure: contour

```

1: function BORDERFOLLOWING(self, image, start, previous)
2:   pointer_one ← previous
3:   pointer_three ← start
4:   contour[]
5:   # clockwise movement
6:   count ← 0
7:   while image[pointer_one[0]][pointer_one[1]] = 0 do
8:     if count > 7 then           ▷ If the starting pixel is a single pixel dot
9:       image[pointer_one[0]][pointer_one[1]] = 4
10:      contour ← image[pointer_one[1]-1][pointer_one[1]-1]
11:      return contour
12:    end if
13:    position, next pointer ← next pointer position clockwise
14:    pointer_one ← next pointer
15:    count ← count + 1
16:  end while
17:  pointer_two ← copy of pointer_one
18:  counter ← 0

```

```

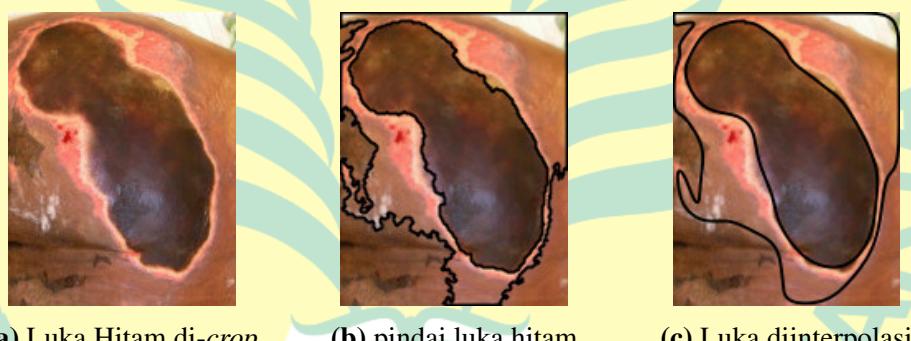
19: while True do
20:     # Counter clockwise movement
21:     position, next pointer ← next pointer position counter clockwise
22:     pointer_two ← next pointer
23:     while image[pointer_two[0]][pointer_two[1]] = 0 do
24:         position, next pointer ← next pointer position counter clockwise
25:         pointer_two ← next pointer
26:     end while
27:     pointer_four ← pointer_two
28:     # Assign NBD
29:     NBD_coordinate ← copy pointer_three
30:     if image[NBD_coordinate[0]][NBD_coordinate[1]+1] = 0 then
31:         image[NBD_coordinate[0]][NBD_coordinate[1]] = 4
32:     else if image[NBD_coordinate[0]][NBD_coordinate[1]+1] ≠ 0 and
33:         image[NBD_coordinate[0]][NBD_coordinate[1]] = 1 then
34:             image[NBD_coordinate[0]][NBD_coordinate[1]] = 2
35:     end if
36:     contour ← image[NBD_coordinate[1]-1][NBD_coordinate[0]-1]
37:     # Determine new pointer or break
38:     if pointer_four[0] = start[0] and pointer_four[1] = start[1] then
39:         if pointer_three[0] = pointer_one[0] and pointer_three[1] =
40:             pointer_one[1] then
41:             break
42:         end if
43:     end if
44:     pointer_two ← copy of pointer_three
45:     pointer_three ← copy of pointer_four
46:     counter ← counter + 1
47: end while
48: return contour
49: end function

```

3.3 Interpolasi

Setelah mendapatkan titik koordinat dari hasil pemindaian dari border following, maka akan dilanjutkan dengan penghalusan kurva oleh interpolasi. Interpolasi yang digunakan oleh penulis merupakan *surface fitting interpolation* di mana teknik ini digunakan untuk membuat permukaan halus yang melewati titik dari poin data yang diberikan. Dalam konteks gambar, *surface fitting interpolation* juga bisa digunakan untuk membuat permukaan halus dan kontinu yang merepresentasikan nilai intensitas gambar pada setiap lokasi piksel.

Berdasarkan hasil penelitian Rizki, interpolasi ini meningkatkan akurasi dari alat deteksinya, dari yang normalnya mendapatkan 77.18% menjadi 86.1%. Penulis menginginkan untuk mengembangkan angka itu dengan meneliti cara kerja algoritma interpolasi rizki, menjadi buatan sendiri yang bisa diatur algoritmanya.



Gambar 3.2: Hasil Interpolasi citra

Seperti di penelitian Rizki, untuk mendapatkan hasil yang lebih akurat, dilakukan anotasi terlebih dahulu sehingga mengurangi kemungkinan untuk mendapatkan *image noise* atau penggarisan tidak perlu pada hasil pendeksi luka.

Untuk membuat program interpolasi ini, dibuat dulu variabel yang diperlukan untuk menjalankan rumus utama. Variabel yang pertama dicari adalah \bar{u} atau disebut *knot* menggunakan rumus *chord length* (2.8), yang dihitung dengan memasukkan sejumlah koordinat, lalu dihitung setiap panjang antara koordinat. Setelah mendapatkan semua panjangnya, lalu ditotalkan untuk menjadi pembagi sehingga didapatkan \bar{u} sesuai dengan jumlah koordinatnya.

Algorithm 3 Chord Length

Require: $\{\mathbf{Q}_k\}, k = 0, \dots, n$ **Ensure:** $\bar{u}[n]$

```

1: function CHORDLENGTH( $\mathbf{Q}_k[n]$ )
2:    $\bar{u}[n]$ 
3:    $\bar{u} \leftarrow 0$ 
4:    $d \leftarrow 0$ 
5:   for  $i \leftarrow 0$  to  $n$  do
6:     length  $\leftarrow |\mathbf{Q}_k - \mathbf{Q}_{k-1}|$ 
7:      $d \leftarrow d + \text{length}$ 
8:      $\bar{u}[i] \leftarrow d$ 
9:   end for
10:  for  $i \leftarrow 0$  to  $n$  do
11:     $\bar{u}[i] \leftarrow \bar{u}[i] / d$ 
12:  end for
13:  return  $\bar{u}[n]$ 
14: end function

```

- $\{\mathbf{Q}_k\}$ merupakan input koordinat
- k adalah indeks dari koordinat
- n jumlah koordinat yang dimasukkan
- \bar{u} knot, hasil dari algoritma

Dari algoritma (3), didapatkan array yang berisikan \bar{u} . Variabel selanjutnya adalah U yang disebut juga *knot vector* menggunakan rumus rata-rata knot (2.11). Dimulai dengan mengambil hasil dari algoritma 3 lalu isi array dengan indeks dari 0 sampai sederajatnya menjadi 0 dan mengisi array dengan indeks setelah jumlah indeks dikurangi derajat dengan angka 1. Diantara dari 0 dan 1, diisi dengan menghitung jumlah knot didalam indeks yang tidak diubah, lalu dibagi dengan derajatnya.

Algorithm 4 Rata-rata Knot

Require: $\bar{u}[n]$ and p **Ensure:** $U[n + p + 1]$

```

1: function KNOTVECTOR( $\bar{u}[n]$ ,  $p$ )
2:    $U[n + p + 1]$ 
3:   for  $i \leftarrow 0$  to  $n$  do
4:     if  $i \leq p$  then
5:        $U[i] \leftarrow 0$ 
6:     else if  $i \geq n - p - 1$  then
7:        $U[i] \leftarrow 1$ 
8:     else
9:        $U[i] \leftarrow 0$ 
10:      for  $ii \leftarrow i - p$  to  $i$  do
11:         $U[i] \leftarrow U[i] + \bar{u}[ii]$ 
12:      end for
13:       $U[i] = U[i] / p$ 
14:    end if
15:   end for
16:   return  $U[n + p + 1]$ 
17: end function

```

- \bar{u} knot yang didapat dari algoritma (3)
- n jumlah knot
- p merupakan derajat interpolasi
- U knot vector, hasil dari algoritma(4)

Setelah mendapat dua variabel penting yaitu \bar{u} dan U , sudah bisa menjalankan rumus dasar (2.4). Algoritma dimasukkan empat variabel, yaitu knot yang didapatkan dari algoritma (3), indeks, derajat, dan knot vector dari algoritma (4). Yang pertama diperiksa adalah apakah derajatnya merupakan nol. apabila derajatnya nol maka akan diperiksa knotnya melebihi atau sama dengan knot vector pada indeksnya dan kurang dari knot vector dengan indeks selanjutnya. apabila iya akan diberikan angka satu, apabila tidak akan diberikan nol. Selain dari derajat nol, maka algoritma ini akan menjadi rekursif di mana akan selalu memanggil hasil dari derajat sebelumnya.

Algorithm 5 Rumus Dasar Spline (*B-Spline*)

Require: $\bar{u}, i, p, U[]$

Ensure: spline

```

1: function SPLINE( $\bar{u}, i, p, U[ ]$ )
2:   if  $p$  is 0 then
3:     if  $U[i] \leq \bar{u} < U[i+1]$  then
4:       return 1
5:     else
6:       return 0
7:     end if
8:   else                                ▷ separate left and right equation
9:     left  $\leftarrow (\bar{u} - U[i]) / (U[i + p] - U[i])$ 
10:    right  $\leftarrow (U[i + p + 1] - \bar{u}) / (U[i + p + 1] - U[i + 1])$ 
11:    left  $\leftarrow$  left * Spline( $\bar{u}, i, p - 1, U[ ]$ )           ▷ recursive
12:    left  $\leftarrow$  right * Spline( $\bar{u}, i + 1, p - 1, U[ ]$ )
13:    return left + right
14:  end if
15: end function

```

- \bar{u} knot yang didapat dari algoritma (3)
- i indeks yang akan dipakai pada algoritma(6)
- p merupakan derajat interpolasi
- U knot vector, didapat dari algoritma(4)

Sebelum langkah terakhir adalah membuat matriks dari rumus (2.5) memanggil algoritma (3), (4), dan (5) yang lalu di *inverse*-nya dikalikan dengan koordinat input.

Algorithm 6 Global Interpolation

Require: $\{\mathbf{Q}_k\}, k = 0, \dots, n, p$ **Ensure:** Curve Points

```

1: function GLOBALINTERPOLATION( $\mathbf{Q}_k[n], p$ )
2:    $\bar{u} \leftarrow \text{ChordLength}(\mathbf{Q}_k[n])$ 
3:    $U \leftarrow \text{KnotVector}(\bar{u}, p)$ 
4:    $C[n_x, n_y]$ 
5:   for  $y \leftarrow 0$  to  $n_y$  do
6:     for  $x \leftarrow 0$  to  $n_x$  do
7:        $C[x, y] = \text{Spline}(\bar{u}[x], y, p, U)$ 
8:     end for
9:   end for
10:   $C \leftarrow \text{Inverse\_of\_}C * \mathbf{Q}$ 
11:  return  $C$ 
12: end function

```

Langkah terakhir yang akan dilakukan dengan proses interpolasi adalah dengan membuat garis halus sesuai dengan titik-titik yang sudah dibuat oleh *global interpolation*(6) menggunakan algoritma Bézier(2.1)

Algorithm 7 Bézier

Require: CurvePoints, TotalPoints

▷ TotalPoints is how smooth the curve is

Ensure: Curve**function** BERNSTEIN(n, k, t) **return** $\text{binom}(n, k) * t^k * (1-t)^{n-k}$

▷ binom function from scipy

end function**function** BEZIER(CurvePoints, TotalPoints) $N \leftarrow \text{Length of CurvePoints}$ $t \leftarrow \text{sequence of numbers from 0 to 1 with the length of TotalPoints}$ Curve $\leftarrow []$ **for** $i \leftarrow 0$ to N **do** Curve $\leftarrow \text{Bernstein}(N - 1, i, t) * \text{CurvePoints}[i]$ **end for** **return** Curve**end function**

3.4 Perancangan Eksperimen

Pada subbab ini akan membahas bagaimana rancangan eksperimen yang akan dilakukan dalam penelitian ini. Eksperimen dimulai dengan pengolahan data, kemudian deteksi luka oleh sistem yang diakhiri dengan validasi data.

3.4.1 Sumber Data

Dikarenakan penulis melanjutkan penelitian dari Rizki, maka penulis akan menggunakan sumber data yang sama, yaitu bersumber dari penelitian luka Ns. Ratna Aryani, M.Kep, tahun 2018 (Aryani et al., 2018) diambil dari data klinik yang tersedia di <https://github.com/mekas/InjuryDetection>, berjumlahkan 108 luka. dari sumber tersebut, terdapat 37 data yang tidak dapat dipakai karena data tersebut merupakan duplikat dari sumber yang sama sehingga data yang dipakai berjumlah 69 buah citra. Dari citra tersebut, terdapat 24 luka hitam, 15 luka kuning, dan 30 luka merah.



(a) Luka Hitam



(b) Luka Kuning



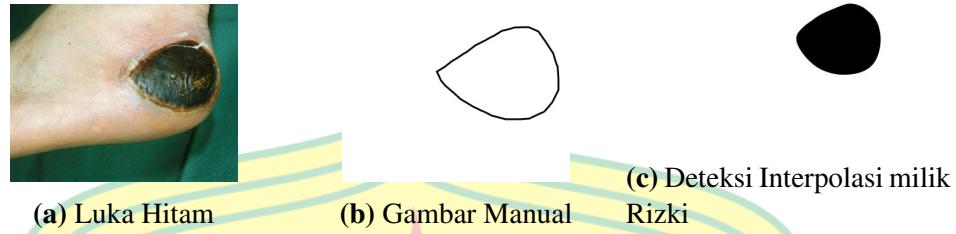
(c) Luka Merah

Gambar 3.3: Data citra luka

3.4.2 Validasi

Seperti pada penelitian Rizki, akan diperiksa dahulu apakah hasil pindai berhasil atau tidak. Setelah mendapatkan hasil deteksi keliling luka, harus divalidasi ketepatan alat deteksi tersebut. penulis akan memvalidasi data deteksi dengan menghitung selisih piksel dari hasil alat deteksi, dengan *ground truth* yang diambil dari gambar manual menggunakan program GIMP. similaritas akan dihitung sebagai berikut

$$\text{similaritas}(\%) = 100 - \left| \frac{\text{luas ground truth} - \text{luas hasil deteksi}}{\text{luas ground truth}} * 100 \right| \quad (3.1)$$



Gambar 3.4: Validasi alat deteksi



BAB IV

HASIL DAN PEMBAHASAN

4.1 Pengolahan data citra input

Untuk optimal nya pendeksi luka, maka piksel yang dipindai dikurangi dengan memotong citra yang dipindai. Pemotongan ini dilakukan dengan menggunakan program GIMP(*GNU Image Manipulation Program*) dan memakai fungsi *crop*

Langkah selanjutnya dalam pengolahan citra ini dimulai dengan membaca citra nya dalam bentuk hitam putih(*grayscale*). Proses pengolahan data citra input menggunakan *imread* yang merupakan fungsi yang di-*import* dari CV2

```
import cv2 as cv2
gambar = cv2.imread(image, cv2.IMREADGRAYSCALE)
```

Gambar 4.1: Kode untuk membaca citra

4.2 Deteksi Keliling Menggunakan *Border Following*

Citra yang sudah dimasukkan akan diolah oleh program *border following*. Metode ini akan membaca citra dalam bentuk citra biner yang hanya terisi 0 dan 1. Dibentuklah kode sebagai berikut yang mengikuti kode dibuat oleh Hafizhun Alim(Alim, 2023) pada skripsi-nya yang berjudul "*Fish Movement Tracking Menggunakan Metode Gaussian Mixture Models (GMM) dan Kalman Filter*".

```

def border_following(self, img, start, previous):

    pointer_one = previous
    pointer_three = start
    contour = []

    # Step 3.1 Move clockwise
    count = 0
    while img[pointer_one[0]][pointer_one[1]] == 0:
        # If the starting pixel is a single pixel dot
        if count > 7:
            img[pointer_three[0]][pointer_three[1]] = 4
            contour.append(np.array([[pointer_three[1] - 1, pointer_three[0] - 1]]))
            return np.array(contour)

        position, next_pointer = self.next_pointer_position(pointer_one, pointer_three, 1)
        pointer_one = next_pointer
        count += 1

```



```

# Step 3.2
pointer_two = copy.copy(pointer_one)

counter = 0
while True:
    # Step 3.3 Move counter clockwise
    # First, move pointer one time in counter-clockwise direction
    position, next_pointer = self.next_pointer_position(pointer_two, pointer_three, 2)
    pointer_two = next_pointer
    while img[pointer_two[0]][pointer_two[1]] == 0:
        position, next_pointer = self.next_pointer_position(pointer_two, pointer_three, 2)
        pointer_two = next_pointer
        pointer_four = pointer_two

    # Step 3.4 Assign NBD
    # rows or i represent y-axis
    # cols or j represent x-axis
    # the coordinate are inverted because we wanted to return a set of (x, y) points, not (y, x)
    # we use 2 and 4 (-2) since we only extract outer border
    nbd_coordinate = copy.copy(pointer_three)
    if img[nbd_coordinate[0]][nbd_coordinate[1] + 1] == 0:
        img[nbd_coordinate[0]][nbd_coordinate[1]] = 4
    elif img[nbd_coordinate[0]][nbd_coordinate[1] + 1] != 0 and img[nbd_coordinate[0]][nbd_coordinate[1]] == 1:
        img[nbd_coordinate[0]][nbd_coordinate[1]] = 2
        contour.append(np.array([[nbd_coordinate[1] - 1, nbd_coordinate[0] - 1]]))

    # Step 3.5 Determine new pointer or break
    if pointer_four[0] == start[0] and pointer_four[1] == start[1]:
        if pointer_three[0] == pointer_one[0] and pointer_three[1] == pointer_one[1]:
            break
        pointer_two = copy.copy(pointer_three)
        pointer_three = copy.copy(pointer_four)

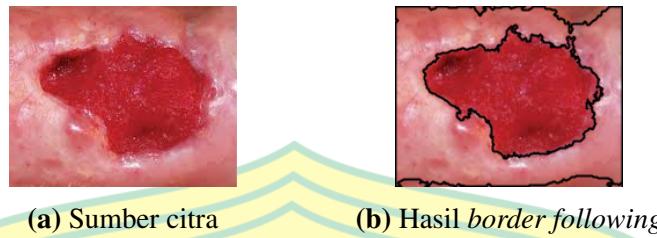
    counter += 1

return np.array(contour)

```

Gambar 4.2: Kode border following

setelah jalannya program diatas, didapatkan hasil contour sebagai berikut



Gambar 4.3: Hasil pindai *border following*

Dalam pendeksiannya, metode ini akan menghasilkan lebih dari daerah dan terkadang mendekksi daerah yang hanya memiliki satu titik koordinat. Untuk membuang daerah yang pasti bukan daerah luka, maka penulis hanya menggambar daerah yang memiliki 50 titik.

4.3 Penghalusan menggunakan Interpolasi Global

Dari hasil *border following*, akurasi dari pindaian citra masih kasar dan belum 100% kepada *ground truth*. Untuk membantu ini, penulis menggunakan *Global Interpolation* yang ada di buku NURBS(Piegl et al., 1996).

Penulis tidak menggunakan interpolasi lokal dikarenakan kurang mampunya penulis untuk mengerti cara penghitungan interpolasi lokal. Banya variabel yang penulis kurang tangkap seperti pada rumus(2.18), di mana memiliki dua jenis *knot vector* yang penulis tidak tahu perbedaannya.

Dimulai dengan rumus utama akan menggunakan algoritma(6). Algoritma ini akan membentuk kode sebagai berikut

```

def GlobalCurveInterpolation(Point, degree):
    Knot = GlobalInterpolation.ChordLength(Point)
    KnotVector = GlobalInterpolation.knotGlobalCurve(Knot, degree)
    nurb = np.empty([len(Point), len(Point)])
    for x in range(len(Point)):
        for y in range(len(Point)):
            nurb[x,y] = GlobalInterpolation.N(Knot[x], y, degree, KnotVector)

    #making sure the last array
    nurb[len(Point)-1,len(Point)-1] = 1.0

    hasil = np.dot(np.linalg.pinv(nurb), Point)

    return hasil

```

Gambar 4.4: Kode Global Interpolation

Dari kode di atas, terdapat beberapa fungsi dari *class GlobalInterpolation* yang akan dijelaskan sebagai berikut

Dalam interpolasi, menentukan titik-titik kurva membutuhkan *knot*. *Knot* dapat ditentukan dengan perata-rataan memakai fungsi *numpy* yaitu *np.linspace(0,1,len(Point))*. Ada juga metode lain yang memastikan kurva lebih stabil yaitu menggunakan *chord length* menggunakan algoritma(3).

```
def ChordLength(koordinat = np.array([0,0])):
    knot = np.empty([len(koordinat)],dtype=float)
    knot[0] = 0
    totalJarak = 0

    #d / total jarak
    for i in range(len(koordinat)):
        if(i>0):
            jarak = np.linalg.norm(koordinat[i-1] - koordinat[i])
            totalJarak += jarak
            knot[i] = totalJarak

    #knotkord
    for i in range(len(koordinat)):
        knot[i] = knot[i]/totalJarak

    #return as KnotVector, group of knots, 1d array
    return np.array(knot)
```

Gambar 4.5: Kode *chord length*

Setelah mendapatkan *knot*, langkah selanjutnya adalah membuat *knot vector*. Kode untuk *knot vector* akan dibentuk mengikuti algoritma(4)

```
def knotGlobalCurve(knots = np.array([0]), degree=0):
    KnotVector = np.empty([len(knots) + degree + 1])
    for i, val in enumerate(KnotVector):
        if(i <= degree): KnotVector[i] = 0
        elif(i >= len(KnotVector) - degree-1): KnotVector[i] = 1
        else:
            knotvector = 0.0
            for ii in range(i-degree, i):
                knotvector += knots[ii]
            KnotVector[i] = knotvector/degree

    #return as 1d array
    return KnotVector
```

Gambar 4.6: Kode *knot vector*

Dan metode utama yang dijalankan secara rekursif, yaitu rumus dasar interpolasi yang dilambangkan sebagai *N* di buku *NURBS*. Dalam pembuatannya, penulis agak kesulitan dalam mendapatkan hasil yang ditujukan oleh buku. Masalah

yang pertama dihadapi adalah membuat $0/0$ menjadi 0. yang penulis pertama kali lakukan adalah mengecek penyebut merupakan 0 maka hasilnya akan 0. Metode ini menghasilkan nilai yang salah pada baris awal dan terakhir dan membuat kolom tengah menjadi 1 semua. Lalu, karena penyebut terjadi operasi pengurangan, pengecek nya diubah dengan apabila variabel kiri sama dengan yang kanan. Setelah pengetesan hampir semua nya benar kecuali angka terakhir, yang seharusnya 1 menjadi 0, di sini penulis memaksa angka matriks terakhir menjadi 1. Berikut adalah kode untuk rumus dasar mengikuti algoritma(5).

```
def N(knot = 0.0, i = 0 , degree = 0, KnotVector = np.array([0])):
    #Basis function that used to count N in interpolation

    if (degree == 0):
        #
        return 1.0 if (KnotVector[i] <= knot < KnotVector[i + 1]) else 0.0
    else:

        #count the left side and right side of equation
        kiri = 0.0 if(KnotVector[i + degree] == KnotVector[i]) else (knot - KnotVector[i]) / (KnotVector[i + degree] - KnotVector[i]) * GlobalInterpolation.N(knot, i, degree - 1, KnotVector)
        kanan = 0.0 if(KnotVector[i + degree + 1] == KnotVector[i + 1]) else (KnotVector[i + degree + 1] - knot) / (KnotVector[i + degree + 1] - KnotVector[i + 1]) * GlobalInterpolation.N(knot, i + 1, degree - 1, KnotVector)

    return kiri + kanan
```

Gambar 4.7: Kode rumus dasar (N)

Dari semua kode di atas, sudah bisa dijalankan sebagai satu program yang akan menghasilkan citra dengan input dari *border following*



(a) *Border following*



(b) Hasil interpolasi

Gambar 4.8: Hasil Interpolasi dari deteksi *border following*

Dari metode yang dipakai di atas, hanya didapatkan titik kontrol dari suatu kurva. Untuk mendapatkan kurva halus seperti yang dicanangkan dalam buku yang akan menghasilkan gambar(4.10) *NURBS* dipakai algoritma dari Bézier(7)

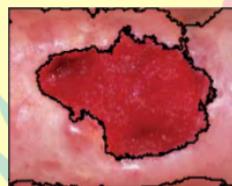
```

berNSTein = lambda n, k, t: binom(n,k)* t**k * (1.-t)**(n-k)

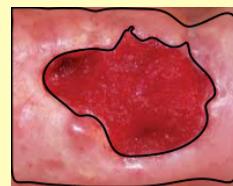
def bezier(points, num=400):
    N = len(points)
    t = np.linspace(0, 1, num=num)
    curve = np.zeros((num, 2))
    for i in range(N):
        curve += np.outer(GlobalInterpolation.berNSTein(N - 1, i, t), points[i])
    return curve

```

Gambar 4.9: Kode Bézier



(a) Interpolasi global



(b) Kurva bézier

Gambar 4.10: Penghalusan kurva menggunakan bézier

Kekurangan dari kode ini adalah kode ini tidak bisa diberi titik data yang banyak, sehingga penulis membataskan titik data menjadi 1000 dengan memotong titik data secara rata.

4.4 Validasi

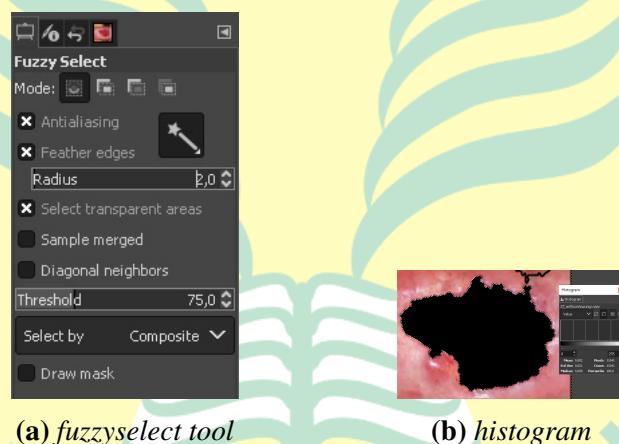
Source image eksperimen diambil dari repository yang dapat diakses di <https://github.com/mekas/InjuryDetection>. Dari semua metode yang sudah ditunjukkan dalam bab ini, maka akan dijalankan ke semua citra yang berada dalam repository.

Dalam pengolahan citra, tidak semua citra dapat dipindai dengan baik. Apabila suatu pindaian dekat dengan bentuk *ground truth* maka akan dikategorikan sebagai berhasil dan yang sebaliknya akan dikategorikan sebagai gagal seperti yang ditunjukkan pada gambar(4.11). Hasil dari pindaian akan dimasukkan kedalam lampiran.



Gambar 4.11: Alat validasi

Dalam memvalidasi citra, akan digunakan *GIMP* dengan menggunakan *fuzzy select tool* dengan treshold 75.- dan 2px feather edges lalu dilihat *pixel count* nya dengan menggunakan *histogram*. Karena program yang penulis pakai hanya memberi garis luar, maka daerah yang terdeteksi akan dihitamkan manual agar dapat dihitung luasnya. Setelah mendapatkan luas dari sumber data, maka akan dibandingkan dengan *ground truth* menggunakan rumus(3.1)



Gambar 4.12: Alat validasi

Percobaan dimulai dengan luka merah, didapatkan hasil sebagai berikut.

Tabel 4.1: Percobaan pada luka merah

| Indeks | Sumber | Border Following | Interpolasi | Ground Truth |
|------------|--------|------------------|-------------|--------------|
| Luka Merah | | | | |
| 12 | | | | |

Tabel 4.1: Percobaan pada luka merah

| Indeks | Sumber | Border Following | Interpolasi | Ground Truth |
|--------|---|---|--|---|
| 18 |  |  |  |  |
| 33 |  |  |  |  |
| 44 |  |  |  |  |

Tabel 4.2: Similiaritas deteksi luka merah *border following* dan yang dibantu dengan interpolasi

| Luka Merah | | | | | |
|------------|----------------|--------|-----------------|----------------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 12 | 29834 | 26565 | 89.04270296 | 26182 | 87.75893276 |
| 18 | 2788 | 3596 | 71.01865136 | 3337 | 80.30846485 |
| 33 | 10951 | 11176 | 97.94539311 | 10732 | 98.00018263 |
| 44 | 2395 | 4172 | 25.80375783 | 3966 | 34.40501044 |
| | <i>Average</i> | | 70.95262632 | <i>Average</i> | 75.11814767 |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka merah = 30
- Jumlah luka yang terdeteksi contour tracing = 4

Dari percobaan deteksi luka merah, tingkat pendekstian dari *border following* lumayan rendah, terdapat 4 pindaian yang berhasil dari 30 citra, namun memiliki similiaritas yang bagus terhadap *ground truth*, dan Bantuan dari Interpolasi meningkatkan similiaritas sebesar 4.16%.

Percobaan akan dilanjut ke luka kuning.

Tabel 4.3: Percobaan pada luka kuning

| Indeks | Sumber | Border Following | Interpolasi | Ground Truth |
|-------------|--------|------------------|-------------|--------------|
| Luka Kuning | | | | |
| 17 | | | | |
| 25 | | | | |

Tabel 4.4: Similiaritas deteksi luka kuning *border following* dan yang dibantu dengan interpolasi

| Luka Kuning | | | | | |
|----------------|--------|--------|-----------------|----------------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 17 | 319 | 317 | 99.37304075 | 282 | 88.40125392 |
| 25 | 2831 | 1345 | 47.50971388 | 1291 | 45.60226069 |
| <i>Average</i> | | | 73.44137732 | <i>Average</i> | 67.0017573 |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka kuning = 15
- Jumlah luka yang terdeteksi contour tracing = 2

Setelah melakukan percobaan di luka kuning, dapat dilihat tingkat pendektsian masih sama seperti pada luka merah, yaitu 13.3%, namun mendapat peningkatan pada pengecekan similiaritas di *border following* pada angka 73.4%, tetapi terjadi penurunan 6.43% di bantuan interpolasi terhadap *border following*

Kategori terakhir untuk dipercobakan adalah luka hitam.

Tabel 4.5: Percobaan pada luka hitam

| Indeks | Sumber | <i>Border Following</i> | Interpolasi | <i>Ground Truth</i> |
|-------------------|--------|-------------------------|-------------|---------------------|
| Luka Hitam | | | | |
| 7 | | | | |
| 18 | | | | |
| 26 | | | | |
| 28 | | | | |
| 29 | | | | |
| 33 | | | | |
| 40 | | | | |
| 41 | | | | |

Tabel 4.6: Similiaritas deteksi luka hitam *border following* dan yang dibantu dengan interpolasi

| Luka Hitam | | | | | |
|------------|--------|--------|-----------------|----------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 7 | 2146 | 2061 | 96.03914259 | 1928 | 89.8415657 |
| 18 | 7328 | 6430 | 87.74563319 | 6009 | 82.00054585 |

Tabel 4.6: Similiaritas deteksi luka hitam *border following* dan yang dibantu dengan interpolasi

| Luka Hitam | | | | | |
|------------|--------|---------|-----------------|----------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 26 | 60411 | 60923 | 99.15247223 | 59143 | 97.90104451 |
| 28 | 5408 | 5608 | 96.30177515 | 5902 | 90.86538462 |
| 29 | 43763 | 44984 | 97.20997189 | 43368 | 99.09741106 |
| 33 | 12899 | 9792 | 75.91286146 | 9417 | 73.00565935 |
| 40 | 3040 | 3069 | 99.04605263 | 2948 | 96.97368421 |
| 41 | 9644 | 9466 | 98.15429282 | 9365 | 97.10700954 |
| | | Average | 92.5807947 | Average | 91.54594104 |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka hitam = 24
- Jumlah luka yang terdeteksi contour tracing = 8

Percobaan terakhir terhadap luka hitam terdapat peningkatan pada tingkat deteksi, berada di angka 33.3%. Similiaritas juga terjadi peningkatan yaitu menjadi 92.6% dan terjadi penurunan yang similiar pada deteksi dengan bantuan interpolasi sebesar 2.85%.

Dari hasil tes, didapatkan hasil merupakan berikut:

- Hasil deteksi pada luka merah menghasilkan 4 citra yang terdeteksi dari 30 citra akurasi daerah luas kurva 70.9% pada *border following* dan 75.1% setelah di-interpolasi
- Hasil deteksi pada luka kuning menghasilkan 2 citra yang terdeteksi dari 15 citra akurasi daerah luas kurva 73.4% pada *border following* dan 67% setelah di-interpolasi
- Hasil deteksi pada luka hitam menghasilkan 8 citra yang terdeteksi dari 24 citra akurasi daerah luas kurva 92.6% pada *border following* dan 91.5% setelah di-interpolasi

Dari hasil ini, dapat dilihat bahwa interpolasi spline yang dilakukan sering sekali menurunkan angka similiaritas di hasil tes pendekripsi luka. Penulis menspekulasi penyebab dari penurunan ini dikarenakan interpolasi memindahkan titik data yang diberikan oleh alat deteksi, *border following* yang sudah berada dekat di garis *ground truth* dipindah secara sembarang dimana lebih sering menjauh daripada mendekat.



BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil eksperimen pengolahan citra, maka diperoleh kesimpulan sebagai berikut:

1. Hasil deteksi luka dengan menggunakan *border Following* berhasil memindai 14 luka dari 69 luka di mana luka hitam yang paling banyak terpindai. Hasil luka yang berhasil terpindai menggunakan *border following* menghasilkan rata-rata akurasi 84.3%, dan setelah menggunakan interpolasi rata-rata akurasi menjadi 82.9%.
2. Apabila dibandingkan dengan *active contour* yang digunakan Rizki, hasil pindai *border Following* lebih banyak gagal dibanding berhasil tetapi memiliki akurasi yang lebih tinggi dengan rata-rata perbedaan 3.6% untuk yang hanya menggunakan *border Following* dan 1.3% untuk yang dibantu dengan interpolasi.

5.2 Saran

Adapun saran untuk penelitian selanjutnya adalah mengembangkan atau mengubah metode pendekripsi luka untuk meningkatkan keberhasilan deteksi luka dan meningkatkan akurasi deteksi luka

DAFTAR PUSTAKA

- Alim, H. (2023). *FISH MOVEMENT TRACKING MENGGUNAKAN METODE GAUSSIAN MIXTURE MODELS (GMM) DAN KALMAN FILTER*.
- Aprilia (2021). *PENGARUH PENGGUNAAN COLOR MODEL LAB DALAM KALIBRASI WARNA LUCA MENGGUNAKAN METODE SEGMENTASI K-MEANS DAN MEAN SHIFT*.
- Aryani, R. et al., (2018). “Buku Panduan: Rancang Bangun Aplikasi Mobile Android Sebagai Alat Deteksi Warna Dasar Luka Dalam Membantu Proses Pengkajian Luka Kronis dengan Nekrosis”.
- Böhm, W., G. Farin, and J. Kahmann (1984). “A survey of curve and surface methods in CAGD”. *Computer Aided Geometric Design* 1.1, pp. 1–60.
- Boor, C. de (Jan. 1978). *A Practical Guide to Spline*. Vol. Volume 27.
- Cheng, Y. (1995). “Mean shift, mode seeking, and clustering”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8, pp. 790–799.
- Faten Abu Shmmala, W. A. (2013). “Color Based Image Segmentation using Different Versions of K-Means in two Spaces”. *Global Advanced Research Journal of Engineering, Technology and Innovation* 1.9, pp. 30–41.
- Gupta, A. (2017). “Real time wound segmentation/management using image processing on handheld devices”. *Journal of Computational Methods in Sciences and Engineering* 12.2, pp. 321–329.
- Gustientiedina Gustientiedina M. Hasmil Adiya, Y. D. (2019). “Penerapan algoritma k-means untuk clustering data obat-obatan pada rsud pekanbaru”. *Jurnal Nasional Teknologi dan Sistem Informasi* 5.1, pp. 17–24.
- Kass, M., A. Witkin, and D. Terzopoulos (1988). “Snakes: Active contour models”. *International journal of computer vision* 1.4, pp. 321–331.
- Khattab, D. et al., (2014). “Color image segmentation based on different color space models using automatic GrabCut”. *The Scientific World Journal* 2014, pp. 1–10.
- Landén, N. X., D. Li, and M. Ståhle (2016). “Transition from inflammation to proliferation: a critical step during wound healing”. *Cellular and Molecular Life Sciences* 73.20, pp. 3861–3885.
- Lockyer, P. (2006). *Controlling the Interpolation of NURBS Curves and Surfaces*. University of Birmingham.

- Manoj Kumar Yadav Dhiraj Dhane, g. M. (2013). "Segmentation of chronic wound areas by clustering techniques using selected color space". *Journal of Medical Imaging and Health Informatics* 3.1, pp. 22–29.
- Parsania, P. and P. Virparia (Jan. 2016). "A Comparative Analysis of Image Interpolation Algorithms". *IJARCCE* 5, pp. 29–34.
- Piegl, L. and W. Tiller (1996). *The NURBS Book*. second. New York, NY, USA: Springer-Verlag.
- Rizki, M. (2022). *DETEKSI KELILING LUCA KRONIS MENGGUNAKAN ACTIVE CONTOUR (SNAKE) DAN ACTIVE CONTOUR YANG DITAMBAHKAN INTERPOLASI*.
- Satoshi Suzuki, K. A. (1985). "Topological structural analysis of digitized binary images by border following". *Computer Vision, Graphics, and Image Processing* 30.1, pp. 32–46.
- Simon, P. E. (2018). *Skin Wound Healing: Overview, Hemostasis, Inflammatory Phase*. <https://emedicine.medscape.com/article/884594-overview>.
- Xingming Zheng, N. L. (2012). "Color recognition of clothes based on k-means and mean shift". *2012 IEEE International Conference on Intelligent Control, Automatic Detection and High-End Equipment (ICADE)*, pp. 22–29.
- Zhao, R. et al., (2016). "Inflammation in chronic wounds". *International journal of molecular sciences* 17.12, p. 2085.

LAMPIRAN A

Source code program

1.1 Border Following

```
import numpy as np
import copy as copy
import cv2 as cv2

class ContourTracing(object):
    def __init__(self):
        pass

    def move_pointer(self, direction, i, j):
        """
        u
        |
        |
        l ----- r
        |
        |
        d

        Parameters
        -----
        direction : string
            DESCRIPTION.
        i : number
            DESCRIPTION.
        j : number
            DESCRIPTION.

        Returns
        -----
        None.
        """
        if direction == "u":
            i -= 1
            j = j
        elif direction == "r":
            i = i
            j += 1
        elif direction == "d":
            i += 1
            j = j
        elif direction == "l":
            i = i
            j -= 1

        return i, j

    def next_pointer_position(self, current, pivot, direction):
        """
        Parameters
        -----
        current : number[][]
            Current pixel coordinate.
        pivot : number[]
            Pivot.
        direction: number
            1 for clockwise , 2 for counter-clockwise

        Returns
        -----
        position: string
            Position char.
        next_pointer: number[][]
            Next Pointer.
        """
        position = ""
        if current[0] == pivot[0] and current[1] == pivot[1] - 1:
            position = "1"
            if direction == 1:
                current[0], current[1] = self.move_pointer("u", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("d", current[0], current[1])
        elif current[0] == pivot[0] - 1 and current[1] == pivot[1] - 1:
            position = "ul"
            if direction == 1:
                current[0], current[1] = self.move_pointer("r", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("d", current[0], current[1])
```

```

        elif current[0] == pivot[0] - 1 and current[1] == pivot[1]:
            position = "u"
            if direction == 1:
                current[0], current[1] = self.move_pointer("r", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("l", current[0], current[1])
        elif current[0] == pivot[0] - 1 and current[1] == pivot[1] + 1:
            position = "ur"
            if direction == 1:
                current[0], current[1] = self.move_pointer("d", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("u", current[0], current[1])
        elif current[0] == pivot[0] and current[1] == pivot[1] + 1:
            position = "r"
            if direction == 1:
                current[0], current[1] = self.move_pointer("d", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("u", current[0], current[1])
        elif current[0] == pivot[0] + 1 and current[1] == pivot[1] + 1:
            position = "dr"
            if direction == 1:
                current[0], current[1] = self.move_pointer("l", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("u", current[0], current[1])
        elif current[0] == pivot[0] + 1 and current[1] == pivot[1]:
            position = "d"
            if direction == 1:
                current[0], current[1] = self.move_pointer("l", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("r", current[0], current[1])
        elif current[0] == pivot[0] + 1 and current[1] == pivot[1] - 1:
            position = "dl"
            if direction == 1:
                current[0], current[1] = self.move_pointer("u", current[0], current[1])
            else:
                current[0], current[1] = self.move_pointer("r", current[0], current[1])

    next_pointer = current
    return position, next_pointer

def border_following(self, img, start, previous):
    """
    Tracing border of an object.

    Parameters
    -----
    img : number[][]
        Image Object.
    start : number[]
        A pixel coordinate which border following starts from.
    previous : number[]
        Previous pixel coordinate of starting point.

    Returns
    -----
    List of coordinate for border points
    """
    pointer_one = previous
    pointer_three = start
    contour = []

    # Step 3.1 Move clockwise
    count = 0
    while img[pointer_one[0]][pointer_one[1]] == 0:
        # If the starting pixel is a single pixel dot
        if count > 7:
            img[pointer_three[0]][pointer_three[1]] = 4
            contour.append(np.array([[pointer_three[1] - 1, pointer_three[0] - 1]]))
            return np.array(contour)

        position, next_pointer = self.next_pointer_position(pointer_one, pointer_three, 1)
        pointer_one = next_pointer
        count += 1

    # Step 3.2
    pointer_two = copy.copy(pointer_one)

    counter = 0
    while True:
        # Step 3.3 Move counter clockwise
        # First, move pointer one time in counter-clockwise direction
        position, next_pointer = self.next_pointer_position(pointer_two, pointer_three, 2)
        pointer_two = next_pointer
        while img[pointer_two[0]][pointer_two[1]] == 0:
            position, next_pointer = self.next_pointer_position(pointer_two, pointer_three, 2)
            pointer_two = next_pointer
            pointer_four = pointer_two

```

```

# Step 3.4 Assign NBD
# rows or i represent y-axis
# cols or j represent x-axis
# the coordinate are inverted because we wanted to return a set of (x, y) points, not (y, x)
# we use 2 and 4 (-2) since we only extract outer border
nbd_coordinate = copy.copy(pointer_three)
if img[nbd_coordinate[0]][nbd_coordinate[1] + 1] == 0:
    img[nbd_coordinate[0]][nbd_coordinate[1]] = 4
elif img[nbd_coordinate[0]][nbd_coordinate[1] + 1] != 0 and img[nbd_coordinate[0]][nbd_coordinate[1]] == 1:
    img[nbd_coordinate[0]][nbd_coordinate[1]] = 2
contour.append(np.array([[nbd_coordinate[1] - 1, nbd_coordinate[0] - 1]]))

# Step 3.5 Determine new pointer or break
if pointer_four[0] == start[0] and pointer_four[1] == start[1]:
    if pointer_three[0] == pointer_one[0] and pointer_three[1] == pointer_one[1]:
        break
pointer_two = copy.copy(pointer_three)
pointer_three = copy.copy(pointer_four)

counter += 1

return np.array(contour)

def raster_scan(self, img):
    """
    Take 2d binary image (bw) as an input. Raster scan will create an additional 0 padded
    border around the original image. This will handle all true pixel (255) that sit at the edge
    of the image

    Parameters
    -----
    img : number [][] 
        Binary image.

    Returns
    -----
    List of contours
    """
    (ret, thresh2) = cv2.threshold(img, 127, 1, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    padded_image = np.pad(thresh2, pad_width=[(1, 1), (1, 1)], mode="constant")
    rows, cols = padded_image.shape
    contours = []
    LNB2 = 0
    for i in range(1, rows - 1):
        for j in range(1, cols - 1):
            # Check if pixel is a starting point or not
            if padded_image[i][j] == 1 and padded_image[i][j - 1] == 0:
                if LNB2 == 4 or LNB2 == 0:
                    contour = self.border_following(padded_image, [i, j], [i, j - 1])
                    contours.append(contour)
            # Check LNB2
            if padded_image[i][j] != 1:
                LNB2 = padded_image[i][j]
    LNB2 = 0

    return np.array(contours)

def findCountourCustom(self, img):
    return self.raster_scan(copy.copy(img))

```

1.2 Global Interpolation

```

import numpy as np
from scipy.special import binom

class GlobalInterpolation(object):
    def __init__(self, point):
        self.point = point
        self.GlobalCurveInterpolation = GlobalInterpolation.GlobalCurveInterpolation(point)

    def ChordLength(koordinat = np.array([0,0])):
        knot = np.empty([len(koordinat)], dtype=float)
        knot[0] = 0
        totalJarak = 0

        #d / total jarak
        for i in range(len(koordinat)):

```

```

if(i>0):
    jarak = np.linalg.norm(koordinat[i-1] - koordinat[i])
    totalJarak += jarak
    knot[i] = totalJarak

#knotkord
for i in range(len(koordinat)):
    knot[i] = knot[i]/totalJarak

#return as KnotVector, group of knots, 1d array
return np.array(knot)

def knotGlobalCurve(knots = np.array([0]), degree=0):
    #knot Vector that used in global interpolation
    KnotVector = np.empty([len(knots) + degree + 1])
    for i, val in enumerate(KnotVector):
        if(i <= degree): KnotVector[i] = 0
        elif(i >= len(KnotVector) - degree-1): KnotVector[i] = 1
        else:
            knotvector = 0.0
            for ii in range(i-degree, i):
                knotvector += knots[ii]
            KnotVector[i] = knotvector/degree

    #return as 1d array
    return KnotVector

```

```

def N(knot = 0.0, i = 0 , degree = 0, KnotVector = np.array([0])):
    #Basis function that used to count N in interpolation
    if (degree == 0):
        #
        return 1.0 if (KnotVector[i] <= knot < KnotVector[i + 1]) else 0.0
    else:
        #count the left side and right side of equation
        kiri = 0.0 if(KnotVector[i + degree] == KnotVector[i]) else (knot - KnotVector[i]) / (KnotVector[i + degree] - KnotVector[i]) * GlobalInterpolation.N(knot, i, degree - 1, KnotVector)
        kanan = 0.0 if(KnotVector[i + degree + 1] == KnotVector[i + 1]) else (KnotVector[i + degree + 1] - knot) / (KnotVector[i + degree + 1] - KnotVector[i + 1]) * GlobalInterpolation.N(knot, i + 1, degree - 1, KnotVector)

        return kiri + kanan

def GlobalCurveInterpolation(Point = np.array([0,0]), degree = 0):
    Knot = GlobalInterpolation.ChordLength(Point)
    KnotVector = GlobalInterpolation.knotGlobalCurve(Knot, degree)
    nurb = np.empty([len(Point), len(Point)])
    for x in range(len(Point)):
        for y in range(len(Point)):
            nurb[x,y] = GlobalInterpolation.N(Knot[x], y, degree , KnotVector)

    #making sure the last array
    nurb[len(Point)-1,len(Point)-1] = 1.0

    hasil = np.dot(np.linalg.pinv(nurb), Point)
    return hasil

berNSTein = lambda n, k, t: binom(n,k)* t**k * (1.-t)**(n-k)

def bezier(points , num=400):
    N = len(points)
    t = np.linspace(0, 1, num=num)
    curve = np.zeros((num, 2))
    for i in range(N):
        curve += np.outer(GlobalInterpolation.berNSTein(N - 1, i, t), points[i])
    return curve

```

LAMPIRAN B

Tabel pengolahan data citra input

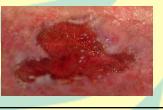
Tabel 2.1: Hasil *crop* sumber data

| Indeks | File | Citra | Crop | Resolusi |
|------------|------------------|---|--|-----------|
| Luka Merah | | | | |
| 2 | luka_merah/2.jpg |  |  | 93 x 183 |
| 3 | luka_merah/3.jpg |  |  | 148 x 250 |
| 4 | luka_merah/4.jpg |  |  | 183 x 264 |
| 6 | luka_merah/6.jpg |  |  | 264 x 86 |
| 7 | luka_merah/7.jpg |  |  | 182 x 139 |

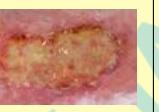
Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|--------|-------------------|---|--|-----------|
| 8 | luka_merah/8.jpg |  |  | 139 x 257 |
| 9 | luka_merah/9.jpg |  |  | 125 x 120 |
| 10 | luka_merah/10.jpg |  |  | 222 x 86 |
| 11 | luka_merah/11.jpg |  |  | 364 x 210 |
| 12 | luka_merah/10.jpg |  |  | 287 x 183 |
| 14 | luka_merah/14.jpg |  |  | 371 x 260 |
| 16 | luka_merah/16.jpg |  |  | 189 x 115 |
| 17 | luka_merah/17.jpg |  |  | 154 x 118 |
| 18 | luka_merah/18.jpg |  |  | 161 x 101 |

Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|--------|-------------------|---|--|-----------|
| 19 | luka_merah/19.jpg |  |  | 427 x 201 |
| 20 | luka_merah/20.jpg |  |  | 343 x 327 |
| 22 | luka_merah/22.jpg |  |  | 264 x 346 |
| 23 | luka_merah/23.jpg |  |  | 428 x 251 |
| 24 | luka_merah/24.jpg |  |  | 272 x 147 |
| 25 | luka_merah/25.jpg |  |  | 290 x 214 |
| 30 | luka_merah/30.jpg |  |  | 78 x 56 |
| 31 | luka_merah/31.jpg |  |  | 192 x 127 |
| 32 | luka_merah/32.jpg |  |  | 187 x 149 |
| 33 | luka_merah/33.jpg |  |  | 197 x 156 |

Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|-------------|--------------------|---|--|-----------|
| 35 | luka_merah/35.jpg |  |  | 218 x 58 |
| 36 | luka_merah/36.jpg |  |  | 108 x 75 |
| 37 | luka_merah/37.jpg |  |  | 180 x 117 |
| 38 | luka_merah/38.jpg |  |  | 290 x 259 |
| 39 | luka_merah/39.jpg |  |  | 57 x 47 |
| 44 | luka_merah/44.jpg |  |  | 170 x 149 |
| Luka Kuning | | | | |
| 3 | luka_kuning/3.jpg |  |  | 107 x 67 |
| 10 | luka_kuning/10.jpg |  |  | 138 x 234 |
| 12 | luka_kuning/12.jpg |  |  | 130 x 72 |
| 13 | luka_kuning/12.jpg |  |  | 130 x 72 |

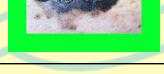
Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|--------|--------------------|---|--|-----------|
| 16 | luka_kuning/16.jpg |  |  | 266 x 190 |
| 17 | luka_kuning/17.jpg |  |  | 57 x 44 |
| 18 | luka_kuning/18.jpg |  |  | 114 x 136 |
| 19 | luka_kuning/19.jpg |  |  | 69 x 89 |
| 21 | luka_kuning/21.jpg |  |  | 164 x 176 |
| 23 | luka_kuning/23.jpg |  |  | 310 x 320 |
| 25 | luka_kuning/25.jpg |  |  | 78 x 95 |
| 34 | luka_kuning/34.jpg |  |  | 269 x 196 |
| 35 | luka_kuning/35.jpg |  |  | 174 x 168 |

Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|------------|--------------------|---|--|-----------|
| 38 | luka_kuning/38.jpg |  |  | 264 x 138 |
| 42 | luka_kuning/42.jpg |  |  | 247 x 209 |
| Luka Hitam | | | | |
| 2 | luka_hitam/2.jpg |  |  | 182 x 152 |
| 4 | luka_hitam/4.jpg |  |  | 173 x 152 |
| 5 | luka_hitam/5.jpg |  |  | 203 x 193 |
| 6 | luka_hitam/6.jpg |  |  | 134 x 105 |
| 7 | luka_hitam/7.jpg |  |  | 70 x 72 |
| 8 | luka_hitam/8.jpg |  |  | 70 x 67 |
| 14 | luka_hitam/14.jpg |  |  | 118 x 109 |

Tabel 2.1: Hasil crop sumber data

| Indeks | File | Citra | Crop | Resolusi |
|--------|-------------------|---|--|-----------|
| 15 | luka_hitam/15.jpg |  |  | 235 x 278 |
| 16 | luka_hitam/16.jpg |  |  | 336 x 143 |
| 17 | luka_hitam/17.jpg |  |  | 219 x 198 |
| 18 | luka_hitam/18.jpg |  |  | 120 x 122 |
| 19 | luka_hitam/19.jpg |  |  | 248 x 171 |
| 20 | luka_hitam/20.jpg |  |  | 291 x 169 |
| 22 | luka_hitam/20.jpg |  |  | 243 x 120 |
| 26 | luka_hitam/26.jpg |  |  | 340 x 321 |
| 27 | luka_hitam/27.jpg |  |  | 379 x 265 |
| 28 | luka_hitam/28.jpg |  |  | 120 x 125 |

Tabel 2.1: Hasil *crop* sumber data

| Indeks | File | Citra | Crop | Resolusi |
|--------|-------------------|---|--|-----------|
| 29 | luka_hitam/29.jpg |  |  | 385 x 278 |
| 31 | luka_hitam/31.jpg |  |  | 326 x 345 |
| 33 | luka_hitam/33.jpg |  |  | 187 x 138 |
| 37 | luka_hitam/37.jpg |  |  | 178 x 138 |
| 39 | luka_hitam/39.jpg |  |  | 351 x 183 |
| 40 | luka_hitam/40.jpg |  |  | 171 x 78 |
| 41 | luka_hitam/41.jpg |  |  | 138 x 178 |

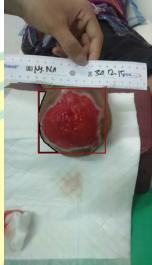
Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | Border Following | Ground Truth | Kategori |
|------------|--------|------------------|--------------|----------|
| Luka Merah | | | | |

Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | <i>Border Following</i> | Ground Truth | Kategori |
|--------|---|---|--|----------|
| 2 |  |  |  | Gagal |
| 3 |  |  |  | Gagal |
| 4 |  |  |  | Gagal |
| 6 |  |  |  | Gagal |
| 7 |  |  |  | Gagal |
| 8 |  |  |  | Gagal |

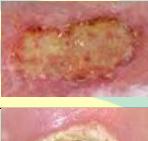
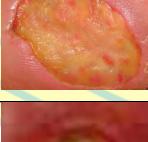
Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | <i>Border Following</i> | Ground Truth | Kategori |
|--------|---|---|---|----------|
| 9 |  |  |  | Gagal |
| 10 |  |  |  | Gagal |
| 11 |  |  |  | Gagal |
| 12 |  |  |  | Berhasil |
| 14 |  |  |  | Gagal |
| 16 |  |  |  | Gagal |
| 17 |  |  |  | Gagal |
| 18 |  |  |  | Berhasil |
| 19 |  |  |  | Gagal |

Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | <i>Border Following</i> | Ground Truth | Kategori |
|--------|--------|-------------------------|--------------|----------|
| 20 | | | | Gagal |
| 23 | | | | Gagal |
| 24 | | | | Gagal |
| 25 | | | | Gagal |
| 30 | | | | Gagal |
| 31 | | | | Gagal |
| 33 | | | | Berhasil |
| 35 | | | | Gagal |
| 36 | | | | Gagal |
| 37 | | | | Gagal |

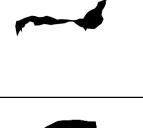
Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | Border Following | Ground Truth | Kategori |
|--------------------|---|---|---|----------|
| 38 |  |  |  | Gagal |
| 39 |  |  |  | Gagal |
| 44 |  |  |  | Berhasil |
| Luka Kuning | | | | |
| 3 |  |  |  | Gagal |
| 10 |  |  |  | Gagal |
| 12 |  |  |  | Gagal |
| 13 |  |  |  | Gagal |
| 16 |  |  |  | Gagal |
| 17 |  |  |  | Berhasil |

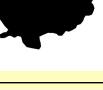
Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | Border Following | Ground Truth | Kategori |
|------------|--------|------------------|--------------|----------|
| 18 | | | | Gagal |
| 19 | | | | Gagal |
| 21 | | | | Gagal |
| 23 | | | | Gagal |
| 25 | | | | Berhasil |
| 34 | | | | Gagal |
| 35 | | | | Gagal |
| 38 | | | | Gagal |
| 42 | | | | Gagal |
| Luka Hitam | | | | |

Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | Border Following | Ground Truth | Kategori |
|--------|---|---|--|----------|
| 2 |  |  |  | Gagal |
| 4 |  |  |  | Gagal |
| 5 |  |  |  | Gagal |
| 6 |  |  |  | Gagal |
| 7 |  |  |  | Berhasil |
| 8 |  |  |  | Gagal |
| 14 |  |  |  | Gagal |
| 15 |  |  |  | Gagal |
| 16 |  |  |  | Gagal |
| 17 |  |  |  | Gagal |

Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | Border Following | Ground Truth | Kategori |
|--------|---|---|---|----------|
| 18 |  |  |  | Berhasil |
| 19 |  |  |  | Gagal |
| 20 |  |  |  | Gagal |
| 22 |  |  |  | Gagal |
| 26 |  |  |  | Berhasil |
| 27 |  |  |  | Gagal |
| 28 |  |  |  | Berhasil |
| 29 |  |  |  | Berhasil |
| 31 |  |  |  | Gagal |
| 33 |  |  |  | Berhasil |

Tabel 2.2: Pemeriksaan *Border Following*

| Indeks | Sumber | <i>Border Following</i> | <i>Ground Truth</i> | Kategori |
|--------|--------|-------------------------|---------------------|----------|
| 37 | | | | Gagal |
| 39 | | | | Gagal |
| 40 | | | | Berhasil |
| 41 | | | | Berhasil |

Tabel 2.3: Similiaritas deteksi luka merah *border following* dan yang dibantu dengan interpolasi

| Luka Merah | | | | | |
|------------|---------|-------------|-----------------|-------------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 12 | 29834 | 26565 | 89.04270296 | 26182 | 87.75893276 |
| 18 | 2788 | 3596 | 71.01865136 | 3337 | 80.30846485 |
| 33 | 10951 | 11176 | 97.94539311 | 10732 | 98.00018263 |
| 44 | 2395 | 4172 | 25.80375783 | 3966 | 34.40501044 |
| | Average | 70.95262632 | Average | 75.11814767 | |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka merah = 30
- Jumlah luka yang terdeteksi contour tracing = 4

Tabel 2.4: Similiaritas deteksi luka kuning *border following* dan yang dibantu dengan interpolasi

| Luka Kuning | | | | | |
|-------------|----------------|--------|-----------------|----------------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 17 | 319 | 317 | 99.37304075 | 282 | 88.40125392 |
| 25 | 2831 | 1345 | 47.50971388 | 1291 | 45.60226069 |
| | <i>Average</i> | | 73.44137732 | <i>Average</i> | 67.0017573 |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka kuning = 15
- Jumlah luka yang terdeteksi contour tracing = 2

Tabel 2.5: Similiaritas deteksi luka hitam *border following* dan yang dibantu dengan interpolasi

| Luka Hitam | | | | | |
|------------|----------------|--------|-----------------|----------------|-----------------|
| Indeks | GT(px) | BF(px) | Similiaritas(%) | Intp(px) | Similiaritas(%) |
| 7 | 2146 | 2061 | 96.03914259 | 1928 | 89.8415657 |
| 18 | 7328 | 6430 | 87.74563319 | 6009 | 82.00054585 |
| 26 | 60411 | 60923 | 99.15247223 | 59143 | 97.90104451 |
| 28 | 5408 | 5608 | 96.30177515 | 5902 | 90.86538462 |
| 29 | 43763 | 44984 | 97.20997189 | 43368 | 99.09741106 |
| 33 | 12899 | 9792 | 75.91286146 | 9417 | 73.00565935 |
| 40 | 3040 | 3069 | 99.04605263 | 2948 | 96.97368421 |
| 41 | 9644 | 9466 | 98.15429282 | 9365 | 97.10700954 |
| | <i>Average</i> | | 92.5807947 | <i>Average</i> | 91.54594104 |

- GT = *Ground Truth*
- BF = *Border Following*
- Intp = Interpolasi
- Jumlah luka hitam = 24
- Jumlah luka yang terdeteksi contour tracing = 8

DAFTAR RIWAYAT HIDUP



Nama : Pramudio
Tempat, Tanggal Lahir : Jakarta, 27 Agustus 2000
Orang tua : Ronny Hanggoro & Wenita Indrasari
Email : pramudiohanggoro@gmail.com

Pendidikan

1. SD Negeri Cipinang Muara 05 Pagi (2006-2012)
2. SMP Negeri 52 Jakarta (2012-2013)
3. SMP Negeri 209 Jakarta (2013-2015)
4. SMA Negeri 93 Jakarta, Matematika dan Ilmu Alam (2015-2019)
5. Universitas Negeri Jakarta, Ilmu Komputer (2019 - 2024)

Pengalaman Kerja

1. Game Programmer Intern, **Dragon Emperors** (Juni - November 2022)