Optimasi Penempatan Pola Poligon Konveks Menggunakan Algoritma Genetik

Hadi Saloko - 13504157 [1)]

1) Jurusan Teknik Informatika STEI, ITB, Bandung 40132, email: hadi_saloko@yahoo.com

Abstract – *Dari berbagai macam industri, industri manufaktur merupakan industri yang sangat dipengaruhi oleh bahan. Untuk menghasilkan produk dengan harga minimal, diperlukan optimasi pemakaian bahan. Dengan kuantitas bahan baku produk yang minimal, diusahakan dapat menghasilkan kuantitas produk yang maksimal. Atau dengan kata lain, diusahakan dapat memperkecil kuantitas bahan baku produk yang terbuang dalam proses produksi. Sebagai contoh, peletakan pola baju pada bahan baku produk (kain) akan sangat mempengaruhi seberapa optimal pemakaian bahan bakunya.*

Salah satu algoritma yang relatif mudah dengan ruang pencarian solusi yang dibatasi adalah algoritma genetik*. Dengan algoritma* genetik*, diharapkan dapat ditemukan solusi yang cukup baik, dengan algoritma yang cukup sederhana, dan tidak perlu menjelajahi seluruh ruang pencarian solusi.*

Walaupun dibatasi, tetapi solusi yang diproses / dicoba relatif cukup banyak. Untuk tiap solusi yang dicoba, diperlukan perhitungan geometri (misal: menghitung luas area overlapped) dengan algoritma yang rumit. Kerumitan algoritma untuk perhitungan geometri ini dibatasi untuk poligon konveks mengingat Tugas Akhir ini lebih difokuskan pada algoritma genetik *ketimbang geometri.*

Perangkat lunak yang dihasilkan bernama Polygon Layout Optimizer (PLO). PLO dikembangkan dengan C++ dengan beberapa graphic library bantuan: OpenGL, GLUT, GLUI. PLO mampu menerima berkas poligon dan mencari solusi pola peletakan poligon yang cukup baik. PLO juga dapat menyimpan solusinya ke dalam berkas kromosom untuk dapat dilihat sewaktu-waktu (PLO dapat menampilkan kromosom tertentu). Pengujian telah dilakukan pada berbagai bentuk dan ukuran poligon konveks dan dapat disimpulkan bahwa algoritma genetik *dapat digunakan untuk penempatan optimal pola poligon konveks.*

Kata Kunci. genetik, *penempatan pola.*

1. PENDAHULUAN

Tujuan dari Tugas Akhir ini sebagai berikut:

1. Memodelkan masalah optimasi penempatan poligon konveks ke dalam algoritma genetik.
2. Mengetahui fungsi *fitness*, metode reproduksi, metode *crossover*, metode mutasi yang dapat digunakan untuk masalah tersebut.
3. Membuat perangkat lunak untuk mengoptimasi penempatan pola poligon konveks menggunakan algoritma genetik.
4. Mencari parameter-parameter terkait algoritma genetik yang terbaik (proses pencarian solusi yang sesingkat mungkin dan menemukan solusi yang seoptimal mungkin).

Batasan Tugas Akhir ini sebagai berikut:

1. Pola poligon yang dimaksud adalah pola poligon konveks, tidak menangani pola poligon konkaf.
2. Pola poligon tidak dapat di-*flip*. Hal ini dikarenakan, bahan baku yang digunakan umumnya berbeda sisi yang satu dengan sisi yang lainnya (misal: satu sisi kayu lebih halus daripada sisi lainnya). *Flip* dapat digambarkan sebagai berikut :
   pola panah  tidak dapat di-*flip* menjadi 
   catatan: pola panah di atas adalah pola poligon konkaf dan sekedar digunakan menjelaskan batasan masalah untuk *flip*
3. Tekstur dan pola pada satu sisi bahan baku diasumsikan tidak mempengaruhi peletakan pola poligon (misal: pemotongan pola pada kain bergaris-garis sedemikian sehingga pakaian memiliki garis yang seluruhnya vertikal).
4. Pola poligon tidak dapat diskalakan. Diasumsikan bahwa produk hanya diproduksi dalam satu ukuran (tidak ada pembagian ukuran seperti S, M, L, XL).
5. Seluruh pola poligon hanya akan diletakan pada satu bahan baku yang berbentuk persegi panjang.
6. Mesin produksi tidak memiliki keterbatasan kontinuitas. Misalnya dua pola poligon sedekat apapun tetap dapat dipisahkan (dipotong).

2. DASAR TEORI

Teori yang digunakan dikelompokan ke dalam teori terkait algoritma genetik, teori terkait geometri, dan teori pendukung lainnya.

2.1. Algoritma Genetik

Algoritma genetik terinspirasi oleh teori evolusi. Algoritma genetik dimulai dengan membentuk sekumpulan calon solusi. Pemilihan calon solusi mana

yang akan digunakan untuk membentuk populasi yang baru, bergantung pada nilai *fitness*. Pencarian solusi dan pembentukan populasi baru ini akan terus dilakukan sampai memenuhi suatu kondisi berhenti tertentu.

Gambaran umum cara kerja algoritma genetik sebagai berikut [6]:

1. [*Start*] Pilih secara acak sekumpulan kromosom sebagai populasi awal. Kromosom ini harus sesuai dengan masalah yang hendak dicari solusinya.
2. [*Fitness*] Evaluasi nilai *fitness* dari tiap kromosom pada populasi.
3. [*New population*] Buat populasi baru dari populasi lama dengan cara:
    a. [*Selection*] Pilih dua kromosom induk dari populasi lama berdasarkan nilai *fitness*-nya (semakin baik suatu kromosom, semakin besar kemungkinan terpilihnya).
    b. [*Crossover*] Pertimbangkan kemungkinan pindah silang untuk memutuskan apakah terjadi pindah silang. Jika ya, bentuk kromosom baru dengan melakukan pindah silang pada kedua kromosom induk. Jika tidak, salin ulang kedua kromosom induk.
    c. [*Mutation*] Pertimbangkan kemungkinan mutasi untuk memutuskan apakah terjadi mutasi. Jika ya, mutasikan kromosom yang dihasilkan proses 3.b [*Crossover*]. Jika tidak, salin ulang kromosom yang dihasilkan proses 3.b [*Crossover*].
    d. [*Accepting*] Masukkan kromosom, yang dihasilkan proses 3.c [*Mutation*] ke dalam populasi *offspring*.
4. [*Replace*] Buang populasi lama, gunakan populasi *offspring* untuk proses selanjutnya.
5. [*Test*] Cek kondisi berhenti (pertimbangkan banyak populasi baru yang sudah terbentuk dan/atau pertimbangkan isi populasi baru). Jika terpenuhi, proses pencarian solusi dapat diakhiri. Beberapa kromosom terbaik dari populasi terakhir dapat dianggap sebagai solusi.
6. [*Loop*] Jika kondisi berhenti tidak terpenuhi, lanjutkan proses ke langkah 2 [*Fitness*].

Sekumpulan solusi direpresentasikan dengan sekumpulan kromosom [6]. Suatu kromosom mengandung informasi mengenai solusi yang direpresentasikannya. Representasi kromosom akan menentukan bagaimana cara kromosom menyimpan informasi mengenai solusi.

Dalam *value encoding*, setiap kromosom berupa rangkaian dari beberapa nilai [6]. *Domain* nilai yang dimaksud sangat beragam (bilangan bulat, bilangan real, karakter, simbol, dan sebagainya), dan disesuaikan masalah yang hendak dicari solusinya.

Contoh kromosom yang menggunakan *value encoding* encoding dapat dilihat pada Gambar 1.

Kromosom 1	1.2324 5.3243 0.4556 2.3293 2.4545
Kromosom 2	ABDJEIFJDHDIERJFDLDFLFEGT
Kromosom 3	(belakang), (belakang), (kanan)

Gambar 1:Kromosom *value encoding* [6]

Pindah silang ditujukan untuk memilih *gen* dari kromosom induk yang selanjutnya disusun ulang untuk membentuk kromosom anak [6]. Dengan pindah silang, akan diperoleh kromosom anak yang mewarisi bagian kromosom dari kedua kromosom induknya. Bagian kromosom yang diwariskan, diharapkan merupakan bagian yang unggul. Kromosom anak diharapkan lebih unggul dibandingkan kedua kromosom induknya.

Untuk *uniform crossover* (dapat digunakan untuk *binary encoding* maupun *value encoding*), dipilih secara acak beberapa *crossing point* pada kromosom, kemudian menyilangkan bagian kromosom induk sebelum dan setelah posisi tersebut secara bergantian antara kedua kromosom induk.

Uniform crossover dengan tiga *crossing point* encoding dapat dilihat pada Gambar 2. (*crossing point* pertama di posisi antara lokus pertama dan lokus kedua, *crossing point* kedua di posisi antara lokus ketiga dan lokus keempat, *crossing point* ketiga di posisi antara lokus keenam dan lokus ketujuh; ditandai dengan '|')

Kromosom induk 1	**1** | **10** | **010** | **11**
Kromosom induk 2	1 | 10 | 111 | 11
Kromosom anak 1	**1** | 10 | **010** | 11
Kromosom anak 2	1 | **10** | 111 | **11**

Gambar 2: Pindah silang *binary encoding*: *uniform crossover* [6]

Setelah pindah silang, kromosom hasil kemungkinan akan mengalami mutasi [6]. Mutasi sendiri ditujukan untuk menghindari proses pencarian solusi yang terperangkap solusi optimal lokal. Mutasi mengubah bagian kecil dari kromosom hasil pindah silang.

Mutasi kromosom pada *value encoding* dilakukan dengan cara memilih secara acak satu *gen* dan mengubah nilainya. Pengubahan nilai dilakukan dengan menambahkan atau mengurangkan dengan nilai tertentu (yang juga dibangkitkan secara acak).

Contoh mutasi pada *value encoding* dapat dilihat pada Gambar 3.

Kromosom anak	1.29 5.68 **2.86 4.11**
Kromosom anak termutasi	1.29 5.68 **2.73 4.22**

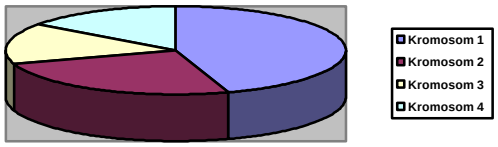
Gambar 3: Mutasi *value encoding* [6]

Terdapat dua parameter dasar pada algoritma genetik: peluang pindah silang dan peluang mutasi. Selain kedua parameter dasar itu, ada juga parameter tambahan seperti besar populasi.

Dalam algoritma genetik, fungsi *fitness* merupakan fungsi objektif masalah yang akan dicari solusinya [9]. Fungsi ini menghasilkan nilai yang digunakan sebagai ukuran: seberapa optimal suatu solusi. Misalnya saja keuntungan (yang hendak dimaksimalkan) atau biaya (yang hendak diminimalkan). Terdapat beberapa cara pendefinisian fungsi *fitness* terlepas dari masalah apa yang hendak dicari solusinya.

Raw fitness merupakan pendefinisian fungsi *fitness* yang paling natural, langsung didefinisikan dari masalah yang hendak dicari solusinya [9]. Selain natural, *raw fitness* merupakan fungsi *fitness* yang paling sederhana, seperti yang telah dijelaskan di bagian sebelumnya (fungsi yang menghitung keuntungan atau biaya). Karenanya, *raw fitness* memiliki rentang dan arti yang berbeda untuk masalah yang berbeda.

Pada tahap 3.a. [*Selection*] dilakukan pemilihan kromosom dari populasi lama untuk menjadi calon kromosom induk. Pada bagian tersebut, dijelaskan bahwa pemilihan ini berdasarkan nilai *fitness* masing-masing kromosom. Akan tetapi bagaimana detail pemilihan kromosom itu sendiri? Terdapat banyak metode untuk memilih kromosom (yang dapat bertahan hidup dan akan menciptakan populasi *offspring*). Diantaranya saja seleksi *roulette wheel* (dapat dilihat pada Gambar 4), seleksi ranking, seleksi *steady state*, *elitism*, dan seleksi turnamen.



Gambar 4: Mutasi *value encoding* [6]

2.2. Kalkulasi Geometri

Diperlukan dasar teori mengenai vektor (*Tupple n*-dimensi **V** = (*v*1, *v*2, ..., *vn*) dimana tiap elemennya (*v*i, $1 \leq i \leq n$) merupakan suatu besaran skalar) antara lain: Panjang vektor [14], penjumlahan vektor [14], perkalian dengan skalar [14], perkalian titik [14], dan perkalian silang [14].

Diperlukan dasar teori mengenai poligon (Beberapa segmen garis yang semuanya terletak pada satu bidang datar. Tiap segmen garis saling sambung-menyambung dan membentuk bentuk tertutup) antara lain: definisi poligon konveks dan poligon sederhana [4], area poligon [1][11], *centroid* poligon [1][11], dan titik ekstrim poligon [13].

Diperlukan dasar teori untuk menghitung interseksi dua poligon konveks antara lain: Separating *axis theorem* [2][12] dan *rotating calipers* [7][8][15].

2.3. Dasar Teori Lainnya

Dasar teori lainnya yang diperlukan antara lain: *Spatial Relation* [10] dan Teori Graf [3].

3. ANALISIS MASALAH

Representasi kromosomnya sebagai berikut: Untuk tiap-tiap poligon diperlukan empat *gen* untuk menjelaskannya, kromosom untuk peletakan tiga poligon sebagai berikut:

Z1, X1, Y1, A1, Z2, X2, Y2, A2, Z3, X3, Y3, A3

dimana :

Z1 adalah poligon acuan untuk poligon pertama
X1 dan Y1 adalah posisi poligon pertama
A1 adalah sudut putar poligon pertama
Z2 adalah poligon acuan untuk poligon kedua
X2 dan Y2 adalah posisi poligon kedua
A2 adalah sudut putar poligon kedua
Z3 adalah poligon acuan untuk poligon ketiga
X3 dan Y3 adalah posisi poligon ketiga
A3 adalah sudut putar poligon ketiga

Nilai *fitness* yang digunakan adalah *raw fitness* dengan nilai yang semakin kecil untuk peletakan poligon yang lebih optimal (berusaha meminimalisasi nilai *fitness*). Nilai *fitness* ini diperoleh dari luas area bounding box, yang kemudian ditambah dengan penalti yaitu *overlapped area* (setelah dikalikan dengan faktor tertentu).

fitness = luas_bounding_box + (faktor * luas_overlapped_area)

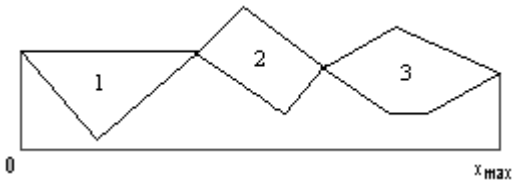
Setelah pindah silang diperlukan pengecekan terhadap kromosom hasil (apakah tetap merupakan kromosom yang valid). Pengecekan terhadap nilai *gen* dilakukan cukup sederhana (pengecekan *domain*), kecuali *gen* Z.

Jika *gen* Z direpresentasikan sebagai graf, untuk tiap-tiap graf terhubung bagian dari hutan tersebut haruslah merupakan pohon (sehingga dijamin tidak ada sirkuit penyebab ketidakvalidan kromosom). Misalnya saja kromosom (2, 2, 2, 0, 1, 3, 3, 0, 0, 4, 4, 0)
Cara untuk memperbaikinya

1. dengan menyambungkan dua graf terhubung menjadi satu graf terhubung (mengubah nilai *gen* kelima menjadi 2), atau
2. dengan mengubah graf terhubung pertama menjadi pohon (mengubah nilai *gen* kelima menjadi 0)

Untuk *domain* poligon acuan (*gen* Z) tergantung dari banyaknya poligon yang ingin disusun. *Domain*nya 0 – *n* (*n* = jumlah poligon).

Untuk *domain* posisi (*gen* X dan Y) dapat dibuat bergantung pada jumlah, bentuk, dan ukuran poligon-poligon yang ingin disusun. *Domain*-nya bilangan real antara –xmax sampai dengan xmax (xmax = jumlah dimensi terpanjang tiap-tiap poligon). Untuk mencari dimensi terpanjang suatu poligon dapat digunakan *rotating calipers*. Contoh xmax dapat dilihat pada Gambar 5. Akan tetapi, dengan alasan kemudahan, *domain* posisi dapat disederhanakan dan di-*hardcode*. Penjelasan lebih lanjut dapat dilihat pada bagian Implementasi.



Gambar 5: Xmax, jumlah dimensi terpanjang

Domain yang tidak tergantung poligon yang ingin disusun hanyalah sudut rotasi (*gen* A). *Domain*-nya bilangan real antara 0 ^{o} - 360 ^{o}.

Di bagian fungsi *fitness*, terlihat bahwa *overlapped area* dapat dianggap sebagai *constraint*. Yang menentukan *hard* / *soft*-nya *constraint overlapped area* adalah faktor pengali tersebut. Dengan mengambil faktor pengali yang besar, maka *overlapped area* dianggap sebagai *hard constraint*, dan sebaliknya.

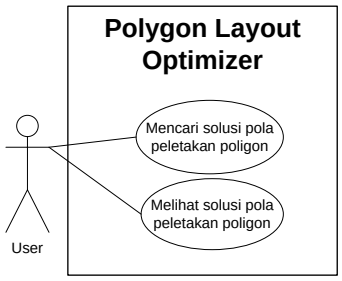
4. ANALISIS DAN PERANCANGAN PERANGKAT LUNAK

Yang diperlukan pada analisis berorientasi objek sebagai berikut: analisis kebutuhan, analisis *usecase*, analisis skenario, dan analisis diagram kelas. Kemudian dilanjutkan dengan perancangan.

4.1. Analisis Perangkat Lunak

Perangkat lunak harus dapat:
1. Menerima data poligon berbentuk berkas.
2. Mencari solusi peletakan optimal untuk poligon masukan.
3. Menerima parameter pencarian solusi dan menyesuaikan proses pencarian solusi.
4. Menghasilkan solusi berbentuk berkas.
5. Menampilkan solusi dalam bentuk yang lebih mudah dimengerti user.

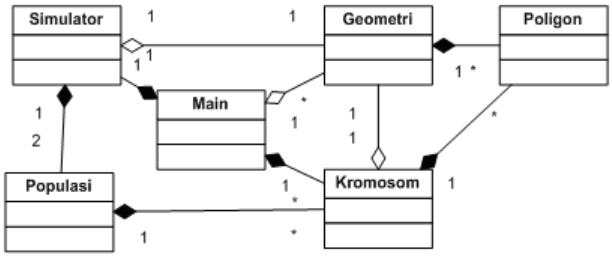
Polygon Layout Optimizer digunakan oleh satu user. Perangkat lunak ini memenuhi kelima kebutuhan melalui kedua *usecase*-nya (lihat Gambar).
1. Mencari solusi
2. Melihat solusi



Gambar 6: Diagram *Usecase*

Dengan terpisahnya dua *usecase* tersebut, user dapat melihat ulang solusi yang pernah dihasilkan tanpa harus melakukan proses pencarian solusi. Tiap *usecase* tersebut memiliki skenarionya masing-masing.

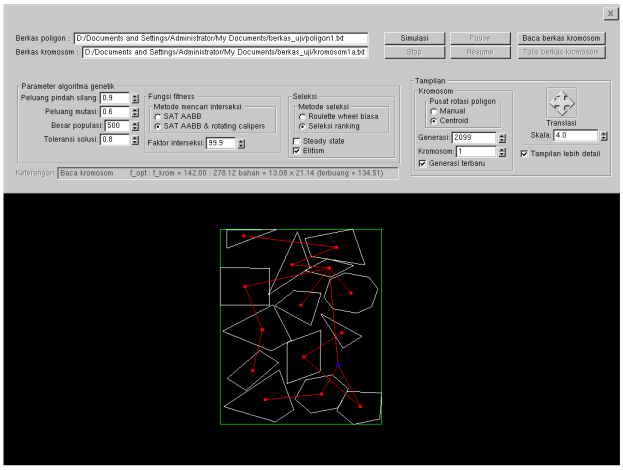
Terdapat enam kelas untuk membangun perangkat lunak *Polygon Layout Optimizer*, seperti yang dapat dilihat pada Gambar.



Gambar 7: Kelas Analisis

4.2. Perancangan Perangkat Lunak

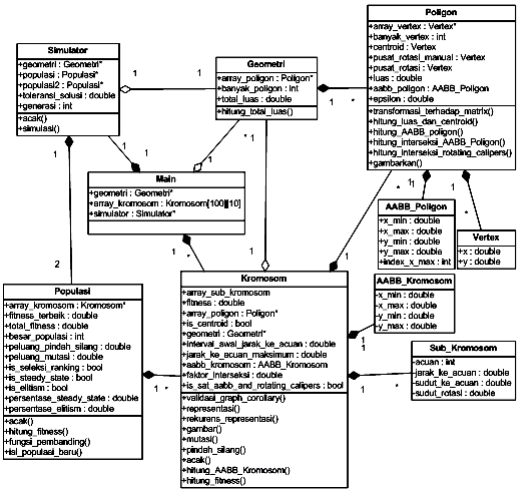
Antar muka dirancang sedemikian rupa sehingga (dapat dilihat pada Gambar 8):
1. Pengguna dapat melakukan kedua *usecase* tanpa mengganti tampilan (misal mengganti *form*).
2. Bersifat *Console Application* (bukan *Windows Form Aplication*).
3. Memanfaatkan *library* tambahan yaitu *OpenGL* / *Graphic Library*



Gambar 8: Rancangan Antar Muka

Enam kelas dirancang sebagai berikut (dapat dilihat pada Gambar 9):

1. Kelas *Main* adalah kelas yang berinteraksi langsung dengan user.
2. Kelas Simulator adalah kelas yang mengatur jalannya proses pencarian solusi dalam algoritma genetik.
3. Kelas Populasi adalah kelas yang mengatur hubungan kromosom-kromosom dan menampung kromosom-kromosom.
4. Kelas Kromosom adalah kelas yang merepresentasikan solusi.
5. Kelas Geometri adalah kelas yang fungsinya untuk melakukan komputasi geometri terlepas dari algoritma genetik.
6. Kelas Poligon adalah kelas yang fungsinya untuk melakukan komputasi geometri dalam konteks poligon.



Gambar 9: Kelas Rancangan

Empat struktur data dirancang sebagai berikut:

1. *Vertex*, adalah struktur data yang menyimpan data *vertex* yang digunakan oleh kelas Poligon.
2. AABB_Poligon, adalah struktur data yang menyimpan data bounding box, atau lebih tepatnya *axis aligned bounding box* ditambah data mengenai index *vertex* terkanan.
3. AABB_Kromosom, adalah struktur data yang menyimpan data *axis aligned bounding box* sekumpulan poligon.
4. Sub_Kromosom, adalah struktur data yang menyimpan data empat *gen* (representasikan suatu poligon) pada kromosom.

5. IMPLEMENTASI DAN PENGUJIAN

Bagian ini berisi kesimpulan mengenai keseluruhan Tugas Akhir ini. Bagian ini juga berisi saran untuk pengembangan lebih lanjut topik sejenis

5.1. Implementasi

Perangkat lunak harus dapat:
Desktop dengan spesifikasi perangkat keras:

1. Prosesor *Intel Pentium* 4 CPU 2.00 GHz
2. Memori RAM 512 MB
3. *Harddisk space* 33.1 GB

Spesifikasi perangkat lunak:

1. *Windows XP Service Pack* 2.0
2. *.Net framework* 2.0
3. memiliki glu32.dll, glut32.dll, dan opengl32.dll pada *folder system32*-nya
4. *Microsoft Visual* C++ 2005/2008 dengan:
   a. GL.h, GLU.h, glui.h, dan glut.h yang dibungkus *folder* GL pada *include path*
   b. OPENGL32.lib, GLU32.lib, glui32.lib, dan glut32.lib pada *library path*

Seluruh kelas pada bagian Perancangan Kelas, diimplementasikan ke dalam dua berkas:

1. Berkas pertama (dengan ekstensi .h) berisi prototipe (*header*) kelas dan *struct*
2. Berkas kedua (dengan ekstensi .cpp) berisi implementasi dari prototipe.

Seluruh struktur data pada bagian Perancangan Kelas, diimplementasikan ke dalam berkas (dengan ekstensi .h) berisi prototipe (*header*) kelas yang terkait.

Antarmuka diimplementasikan dengan bantuan *library* tambahan: *OpenGL* (Open *Graphic Library*), GLUT (*Graphic Library Utility Tools*), dan GLUI (GLUT-*Based User Interface*).

Diperlukan beberapa adaptasi (dari dasar teori, analisis, dan perancangan) agar dapat diimplementasikan. Berikut ini adaptasi yang dilakukan pada implementasi.

1. Titik Ekstrim Poligon
   Jika pada dasar teori dikatakan bahwa pencarian titik ekstrim poligon dapat dilakukan binary search dengan kompleksitas O(log *n*), pada implementasinya pencarian titik ekstrim poligon dilakukan pencarian linier dengan kompleksitas O(*n*).
2. Representasi Kromosom
   Jika pada analisis dipakai representasi absis ordinat untuk menentukan posisi poligon, pada implementasinya dipakai representasi polar.
3. Analisis *Domain*
   Jika pada analisis dipakai *domain* posisi yang bergantung pada (jumlah, bentuk, dan ukuran) poligon-poligon yang ingin disusun, pada implementasinya digunakan *hard*code untuk menentukan *domain* posisi.

5.2. Pengujian

Spesifikasi perangkat lunak :

1. *Windows XP Service Pack* 2.0
2. *.Net framework* 2.0
3. memiliki glu32.dll, glut32.dll, dan opengl32.dll pada *folder system32*-nya atau pada *folder* tempat *executable* berada

Tujuan pengujian yaitu:
1. Mencari tahu apakah perangkat lunak dapat berjalan dengan benar.
2. Mencari tahu seberapa baik proses dan hasilnya (bergantung pada parameter-parameter terkait algoritma genetik).

Data pengujian yaitu:
1. Enam berkas poligon dengan format tertentu:
   a. poligon1.txt : bangun dasar, sedikit *vertex*, berukuran kecil.
   b. poligon2.txt : banyak *vertex*, berukuran besar.
   c. poligon3.txt : sebangun dengan poligon2.txt, ukuran yang lebih kecil, posisi *vertex* tidak bulat dan / atau negatif.
   d. poligon4.txt : segitiga kecil, cukup mudah didapatkan solusi optimalnya.
   e. poligon5.txt : segitiga dan persegi kecil, masih cukup mudah didapatkan solusi optimalnya.
   f. poligon6.txt : beragam poligon, sulit didapatkan solusi optimalnya.
2. Sembilan puluh enam berkas kromosom dengan format tertentu, merupakan solusi dengan berbagai parameter algoritma genetik.

Kasus pengujian yaitu:
1. Berbagai berkas poligon.
2. Berbagai parameter algoritma genetik.
   Hasil pengujian disimpulkan secara general, mengingat banyaknya kombinasi parameter yang mungkin dan sangat acaknya proses dan hasil.

Hasil dan evaluasi pengujian:
1. Berbagai berkas poligon:
   Perangkat lunak berjalan dengan benar. Hal ini terlihat dari pola poligon yang masuk akal dan cukup optimal.
2. Berbagai parameter algoritma genetik:
   a. Peluang pindah silang yang tinggi akan mempercepat ditemukannya solusi.
   b. Peluang mutasi yang tinggi akan mempercepat ditemukannya solusi.
   c. Besar populasi yang tinggi akan mempercepat ditemukannya solusi.
   d. Toleransi solusi yang tinggi akan mempercepat ditemukannya solusi.
   e. Dengan SAT AABB, pergantian generasi sangat cepat terjadi.
   f. Faktor interseksi yang tinggi akan memperlambat ditemukannya solusi.
   g. Dengan seleksi ranking / *Steady state* / *Elitism*, konvergensi suatu populasi sangat cepat terjadi dan mengakibatkan hasil yang buruk.
   h. *Centroid* ataupun dengan pusat rotasi manual, hasil yang diberikan relatif sama.

6. PENUTUP

6.1. Kesimpulan

Menjawab tujuan Tugas Akhir ini, dapat disimpulkan:
1. Masalah optimasi penempatan poligon konveks dengan algoritma genetik dapat dimodelkan dengan *value encoding*.
2. Digunakan dua fungsi *fitness*: SAT AABB dengan ataupun tanpa *rotating calipers* (dikombinasikan dengan faktor interseksi).
   Digunakan dua metode reproduksi: *roulette wheel* biasa ataupun dengan seleksi ranking.
   Digunakan metode *crossover*: multi *crossover*.
   Digunakan metode mutasi: adding.
3. Berhasil dibuat perangkat lunak *Polygon Layout Optimizer*.
4. Parameter-parameter terkait algoritma genetik yang terbaik:
   a. Peluang pindah silang yang cukup besar, yaitu 0.9.
   b. Peluang mutasi yang cukup besar (walau tidak sebesar peluang pindah siang), yaitu 0.6.
   c. Besar populasi yang cukup besar, yaitu 500.
   d. Toleransi solusi yang cukup baik, yaitu mulai 0.6 sampai dengan 1.0.
   e. Metode mencari interseksi yaitu SAT AABB & *rotating calipers*.
   f. Faktor interseksi yang sebesar mungkin, yaitu 99.9.
   g. Metode seleksi yaitu *roulette wheel* biasa (walaupun masalah algoritma genetik dengan seleksi ranking pada umumnya memberikan hasil yang lebih baik).
   h. Seleksi tanpa *steady state*.
   i. Seleksi dengan *elitism*.
   j. Pusat rotasi poligon dengan pusat rotasi manual ataupun *centroid*.

6.2. Saran

Untuk pengembangan lebih lanjut topik sejenis, disarankan untuk:
1. Dapat mengisi populasi dengan kromosom-kromosom tertentu.
2. Memperbaiki algoritma *rotating calipers*.
3. Menggunakan *graphic library* yang lebih baru.
4. Menangani alokasi memori yang gagal.
5. Memperluas topik dengan membuang batasan poligon konveks.

DAFTAR REFERENSI

[1] Bourke, Paul. *Calculating The Area and Centroid of A Poligon*.1988.

[2] Burns, Raigan. *N-Tutorials – Collision Detection and Response*: 21 Juni 2008. <http://www.harveycartel.org/metanet/tutorials/>

[3] Heidelberg, Verlag. *Graph Theory* . 2005.

[4] Page, John. *Math Open Reference*: 21 Juni 2008. <http://www.mathopenref.com/>
[5] Milenkovic, Victor J. *Rotational Polygon Overlap Minimization*, Department of Mathematics and Computer Science University of Miami. 1997.
[6] Obitko, Marek. *Genetic Algorithms*: 20 Juni 2008. <http://www.obitko.com/tutorials/genetic-algorithms/>
[7] Pirzante, Hormoz. *Rotating Calipers*: 25 Juni 2008. <http://www-cgrl.cs.mcgill.ca/~godfried/research/*calipers*.html>
[8] Plante, Eric. *Intersecting Convex Polygon*: 25 Juni 2008. <http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/97/Plante/CompGeomProject-EPlante/>
[9] Saputro, Nico. Diktat Kuliah Algoritma Genetik, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Katolik Parahyangan, Bandung. 1994.
[10] Stockdale, Carol. *Spatial Relation and Learning*: 20 Juni 2008. <http://www.newhorizons.org/spneeds/inclusion/teaching/stockdale.html>
[11] Sunday, Dan. *Area of Triangles and Poligon (2D&3D)*: 25 Juni 2008. <http:// *soft*surfer.com>
[12] Sunday, Dan. *Bounding Containers for Polygon, Polyhedra, and Point Sets (2D&3D)*: 25 Juni 2008. <http:// *soft*surfer.com>
[13] Sunday, Dan. *Extreme Points of Convex Polygons and Distance of Polygon to a Line*: 25 Juni 2008. <http://*soft*surfer.com>
[14]Sunday, Dan. *Basic Linear Algebra*: 21 Juni 2008. <http://*soft*surfer.com>
[15] Toussaint, Godfried T. *A Simple Linear Algorithm for Intersecting Convex Polygons*. 2008.