Pengembangan algoritma cepat penentuan titik-kontak pahat (*cutter contact point*) pada sistem-CAM berbasis model faset 3D untuk pemesinan awal (*roughing*) dan akhir (*finishing*)

Gandjar Kiswanto, A Mujahid
Laboratorium Teknologi Manufaktur
Departemen Teknik Mesin, Universitas Indonesia
Kampus Baru UI Depok 16424
E-mail : gandjar_kiswanto@eng.ui.ac.id

Abstrak

Dengan meningkatnya kompleksitas produk akhir yang diinginkan, kehandalan sistem-CAM untuk pemesinan multi-axis menjadi hal yang utama untuk menjamin keakurasian produk akhir. Laboratorium Teknologi Manufaktur, Departemen Teknik Mesin – Universitas Indonesia, mengembangkan suatu sistem-CAM untuk pemesinan milling multi-axis (3-5 axis) berbasis model faset 3D yang handal dan memiliki kecerdasan dalam otomasi perancanaan strategi pembuatan lintasan pahat. Salah satu kebutuhan utama suatu sistem-CAM adalah pembuatan lintasan pahat (tool path generation). Pada makalah ini disampaikan hasil penelitian mengenai pengembangan algoritma untuk menentukan titik-kontak pahat (cutter contact potint, cc-point) baik untuk pemesinan awal (roughing) maupun akhir (finishing). Secara sederhana, titik-kontak pahat merupakan perpotongan antara bidang potong dengan segitiga model faset 3D. Pada pembuatan lintasan pahat berbasis model faset 3D, jumlah segitiga sangat mempengaruhi kecepatan proses perhitungan/penentuan cc-point. Oleh karena itu dibutuhkan algoritma yang dapat dengan cepat melakukan perhitungan walaupun dengan jumlah segitiga yang banyak. Algoritma atau metode yang dikembangkan menggunakan prinsip yang disebut 'satu antara dua pilihan' (one between two choices). Algoritma ini terbukti lebih cepat, terstruktur dan converge dibandingkan dengan algoritma yang mencari cc-point secara random.

.

Kata kunci: titik kontak pahat (cc-point), pembuatan lintasan pahat, model faset 3D

Pendahuluan

Dengan perkembangan teknologi manufaktur yang semakin pesat dan semakin tingginya kompetisi antara produsen produk-produk manufaktur, kebutuhan akan kualitas produk yang tinggi (*high quality product*) yang dihasilkan dengan kecepatan produksi yang tinggi (*high speed manufacturing*) dengan efisiensi biaya produksi yang tinggi (*low cost production*) menjadi suatu prasyarat. Kesemuanya itu membutuhkan sistem pendukung proses manufaktur yang handal. Salah satu pendukung tersebut adalah sistem-CAM (*Computer-Aided Manufacturing*).

Departemen Teknik Mesin – Universitas Indonesia sedang mengembangkan sistem-CAM berbasis model faset 3D untuk pembuatan lintasan pahat proses pemesinan *milling multi-axis* yang, bila dibandingkan dengan pembuatan lintasan pahat konvensional berbasis model parametrik dan/atau *solid*, dapat menghasilkan lintasan pahat yang cepat dan lebih handal yaitu yang bebas dari *gouging* (*overcut*) dan *collision* dan berada pada akurasi yang dispesifikasikan. Model faset 3D sendiri adalah model produk yang direpresentasikan dengan serangkaian segitiga (*mesh of triangles*) dengan proses triangulasi terhadap model CAD tiga dimensi dari produk. *Gouging*, *overcut* atau *collision* adalah terjadinya interferensi antara *tools* (pahat potong) dengan produk yang dalam proses pemesinan atau antar komponen yang terlibat dalam proses pemesinan (a. l. pahat, produk, sistem clamping, dan bagian mesin lainnya) yang menyebabkan cacat produk hasil pemesinan.

Pada makalah ini disampaikan salah satu hasil penelitian dalam penentuan (pembuatan) *cc-point* yang digunakan sebagai dasar dari pembuatan lintasan pahat. Dikembangkan dua metode yang menghasilkan cc-point dengan cepat. Namun masing-masing metode memiliki kekurangan dan kelebihannya bila saling dibandingkan satu dengan yang lainnya.

Pembuatan Cc-point pada Model Faset

Hal yang pertama harus dilakukan adalah mendapat *roughing points*. *Roughing* point dapat dianalogikan sebagai cc-point pada pemesinan akhir (*finishing*). Secara sederhana dapat

dikatakan bahwa untuk mendapatkan *roughing points* dilakukan dengan mengiriskan bidang z = c dengan model. Irisan ini akan menghasilkan titik-titik perpotongan antara bidang z dengan model. Karena modelnya berbasis facet segitiga, maka perpotongan terjadi antara bidang z dengan sisi-sisi segitiga. Bidang-bidang yang digunakan untuk mencari perpotongan ini tentunya bukan satu bidang saja melainkan bidang-bidang $z = c_i$ untuk i= 1, 2 , … , k, dimana nilai c_i berkisar dari koordinat z paling rendah sampai koordinat z paling tinggi, atau $z_{min} \leq c_i \leq z_{max.}$ Untuk mendapatkan nilai-nilai c_i cukup diberikan berapa interval nilai antara c_i dan c_{i+1} yang dinginkan. Jadi, masukan (*input*) pada proses menentukan *roughing points* ini adalah interval antar bidang potong. Dalam pencarian *roughing points* dalam riset ini digunakan nilai interval sama dengan 2.0.

Ada dua algoritma yang bisa digunakan untuk mendapatkan *roughing points*. **Pertama, melalui *brute searching***. Metode ini dilakukan dengan mengecek setiap segitiga yang ada apakah berpotongan dengan bidang z atau tidak. Jika berpotongan, indeks segitiga tersebut disimpan dalam sebuah struktur data *Vector*. Pengecekan ini dilakukan untuk setiap bidang z yang akan diiriskan dengan model facet.

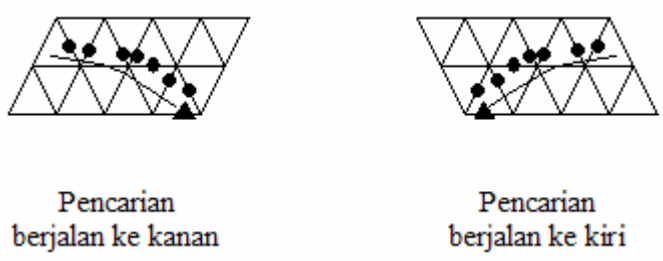
Cara yang **kedua adalah menggunakan *adjacent searching***. Metode ini diawali dengan mencari secara acak sebuah segitiga yang berpotongan dengan bidang z (sebut saja segitiga *u*) Begitu didapatkan satu segitiga tersebut, maka algoritma berjalan sebagai berikut:

1. ambil satu sisi segitiga, misal sisi *s*, yang berpotongan dengan bidang z, kemudian tentukan titik perpotongannya. Simpan titik ini pada sebuah struktur data *Vector* misalnya.
2. cari segitiga yang memiliki sisi *s* sebagai salah satu dari tiga sisi-sisnya selain segitiga yang sudah disebutkan pada no.1 di atas.
3. lakukan kembali langkah no. 1 di atas.

Algoritma akan berhenti jika menemui salah satu dari kondisi-kondisi berikut:

1. Segitiga terakhir yang ditemukan memiliki indeks yang sama dengan segitiga yang pertama kali ditemukan secara acak, yaitu segitiga *u*. Kondisi ini dapat terjadi manakala titik-titik perpotongan membentuk kurva tertutup.
2. Sisi *s* hanya dimiliki oleh satu segitiga saja, artinya tidak ada segitiga lain yang bertetangga dengan segitiga tersebut. Ini dapat terjadi pada titik-titik perpotongan yang membentuk kurva terbuka dan sisi *s* adalah sisi tepi dari model faset.
3. Tidak ada lagi segitiga yang berpotongan dengan bidang z.

Sama halnya dengan metode *brute searching*, metode ini dilakukan untuk setiap kali pengirisan bidang z dengan model faset. Perhatikan gambar berikut sebagai ilustrasinya.



Gambar 1 : Arah pencarian perpotongan antara bidang potong dengan model faset

Ada satu kelebihan utama metode *adjacent searching* dibandingkan metode *brute searching* dalam hal keterurutan. Jika menggunakan *brute,* setelah pencarian titik-titik potong selesai dilakukan, sulit untuk melakukan *tracking* (perjalanan) dari titik satu ke titik lainnya. Sebab, pencarian segitiga dimulai dari indeks terkecil hingga indeks terakhir di mana belum – bahkan dipastikan- belum terurut. Istilah terurut merujuk pada kondisi jika suatu titik potong d_i berada dalam jarak paling dekat dengan d_{i+1} dibandingkan titik-titik lainnya (ada perkecualian pada kondisi tertentu, misalnya jika model berbentuk cekungan).

Namun, berbeda jika metode *adjacent searching* yang digunakan. Sejak awal pencarian hingga akhir, metode ini justru mempertahankan keurutan ini. Apa sebenarnya manfaat dari

keterurutan ini? Dalam implementasi riil proses *milling*, pahat akan bekerja jauh lebih efisien jika titik-titik yang akan dipotong berada dalam posisi terurut berdasarkan kedekatannya. Oleh karena itu, dalam riset ini digunakan metode *adjacent searching* dalam menentukan *roughing points*.

Memotong (*slicing*) segitiga-segitiga pada model faset

Yang dimaksud adalah mengambil segitiga-segitiga yang berada di sekitar bidang z = c. Sekitar bidang ini berarti segitiga-segitiga yang berada antara c-ε dan c+ε, dengan ε adalah bilangan real tertentu. Ini dilakukan untuk meningkatkan efisiensi pada saat pencarian segitiga secara acak yang berpotongan dengan bidang z. Dibandingkan mencari semua segitiga –termasuk yang berjarak jauh dari c, lebih baik dan lebih cepat mencari segitiga di sekitar c.

Algoritma *slicing* ini dijelaskan melalui langkah-langkah berikut ini:

1. cari semua vertek yang berada dalam rentang c-ε dan c+ε
2. cari indeks dari masing-masing vertek hasil no.1
3. temukan segitiga-segitiga yang memiliki indeks hasil no.2

Keluaran (*output*) dari algoritma ini adalah sederetan indeks segitiga yang berada dekat dengan z = c.

Selain digunakan sebagai tempat pencarian segitiga awal secara acak, segitiga-segitiga ini dapat pula digunakan untuk memastikan bahwa tidak ada lagi segitiga yang dipotong oleh bidang z = c.

Fragment code dari algoritma ini dapat dilihat di bawah ini.

Langkah pertama:

```
        // cari semua vertex yang sama dengan z, dengan rentang toleransi
        VertexVector vertexes= new VertexVector(100, 10);
        float tolerance= 2;
        for(int i=0; i<vertexVector.size(); i++){
            Vertex vertex= (Vertex) vertexVector.get(i);
            if(Math.abs( Math.abs(vertex.getZ()) - Math.abs(z)) < tolerance){
                vertexes.addVertex(vertex);
            }
        }
```

Langkah kedua:

```
        // cari index dari setiap vertex dari hasil di atas
        Vector indexes= new Vector(100, 10);
        for(int i=0; i<vertexes.size(); i++){
            Vertex v= (Vertex) vertexes.get(i);
            int found= vertexVector.indexOf(v);
            if (found != -1) indexes.add(new Integer(found));
        }
```

Langkah ketiga:

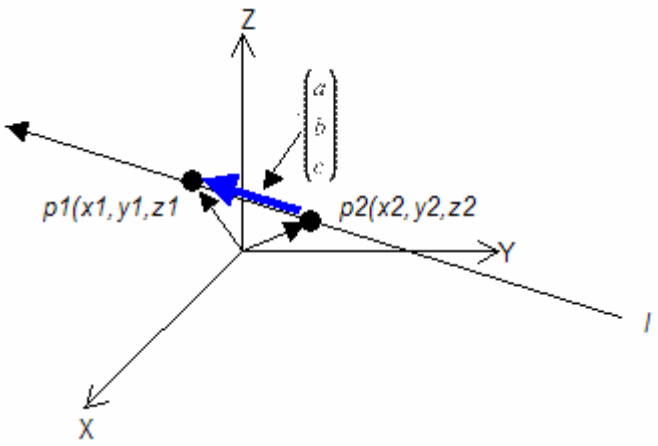
```
        // temukan index segitiga-segitiga yang memiliki indeks vertex di atas
        Vector trianglesIndex= new TriangleVector(100, 10);
        for(int i=0; i<indexes.size(); i++){
            int counter= 0;
            Integer index= (Integer) indexes.get(i);
            for(int j=0; j<triangleVector.size(); j++){
                Triangle t= (Triangle) triangleVector.get(j);
                int ind[]= new int[3];
                ind[0]= t.getFirstVertexIndex();
                ind[1]= t.getSecondVertexIndex();
                ind[2]= t.getThirdVertexIndex();

                if(ind[0]==index || ind[1]==index || ind[2]==index){
                    if(!trianglesIndex.contains(new Integer(j))){
                        trianglesIndex.add(new Integer(j));
                    }
                    // satu vertex dimiliki bersama oleh maksimal kira-kira 6 segitiga
                    if(++counter > 7) break;
                }
            }
        }
```

Menentukan perpotongan sisi segitiga dengan bidang z

Tahap ini maksudnya menentukan formulasi untuk mencari titik potong bidang z = c dengan sisi dari sebuah segitiga. Sisi segitiga dibentuk oleh dua buah titik *p1(x1,y1,z1)* dan *p2(x2,y2,z2)*. Langkah pertama adalah mengecek apakah *c* berada di dalam rentang *z1* dan *z2*. Jika iya, maka berarti sisi segitiga tersebut berpotongan dengan bidang *z = c*. Langkah berikutnya adalah mencari titik di mana perpotongan terjadi.

Secara matematis, persamaan garis dalam ruang R3 dirumuskan sebagai berikut.



Berdasarkan gambar di atas, persamaan garis *l* yang dibentuk oleh 2 titik adalah

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix} t + \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} \qquad (1)$$

Karena $p1 = p2 + \begin{pmatrix} a \\ b \\ c \end{pmatrix}$, maka $\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} x1 \\ y1 \\ z1 \end{pmatrix} - \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix}$ (2)

Dengan demikian persamaan 1) dapat disubstitusi menjadi

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \left(\begin{pmatrix} x1 \\ y1 \\ z1 \end{pmatrix} - \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} \right) t + \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} \qquad (3)$$

Karena *z = c*, maka nilai t dapat dihitung, yaitu $t = \dfrac{z - z2}{z1 - z2}$.

Maka, dari persamaan 3) dapat diperoleh titik *(x,y,z)* sebagai titik perpotongan antara sisi segitiga yang dibentuk oleh titik *p1(x1,y1,z1)* dan *p2(x2,y2,z2)* dengan bidang *z = c*.

Berikut ini adalah *fragment code* untuk mencari perpotongan ini.

```
235         // menghitung t=(c-z2)/(z1-z2)
236         double t= (c - p2[2])/(p1[2] - p2[2]);
237         
238         // x=(x1-x2)t+x2
239         double x= (p1[0] - p2[0])*t + p2[0];
240         // y=(y1-y2)t+y2
241         double y= (p1[1] - p2[1])*t + p2[1];
242         // z=(z1-z2)t+z2
243         double z= (p1[2] - p2[2])*t + p2[2];
244         
245         if(z>=p1[2] && z<=p2[2]) return new double[]{x,y,z};
246         else if(z>=p2[2] && z<=p1[2]) return new double[]{x,y,z};
247         else return null;
```

Implementasi algoritma *adjacent search* dan Hasil

Algoritma ini dapat dijelaskan melalui *pseudocode* berikut.

```
1 for z=z_max to z_min {
2   while (sliced triangles exists){
3     do {search a cutted-triangle t randomly}
4     while (t not found & sliced triangles exists)
5   }
6   do {
7    intersect z with t, get a vertex v
8    add(v)
9    search an adjacent triangle of t
10  } while (found v)
11}
```

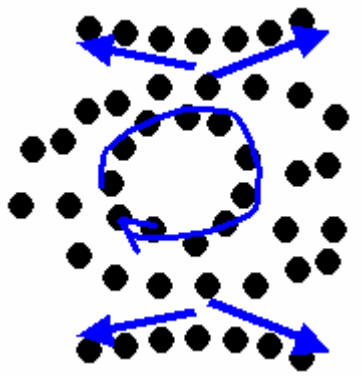
Baris 2 sampai 5 menunjukkan pencarian sebuah segitiga secara acak yang berpotongan dengan bidang z. Rentang segitiga yang dicari tidak lagi seluruh segitiga melainkan segitiga yang sudah dilakukan *slicing*. Sedangkan baris 6 sampai 10 menunjukkan pencarian segitiga secara *adjacent*.

Sebagaimana telah dijelaskan diatas, tiga kondisi di mana perulangan (*looping*) while berhenti dapat digambarkan sebagai berikut.

1. jika titik-titik membentuk kurva tertutup

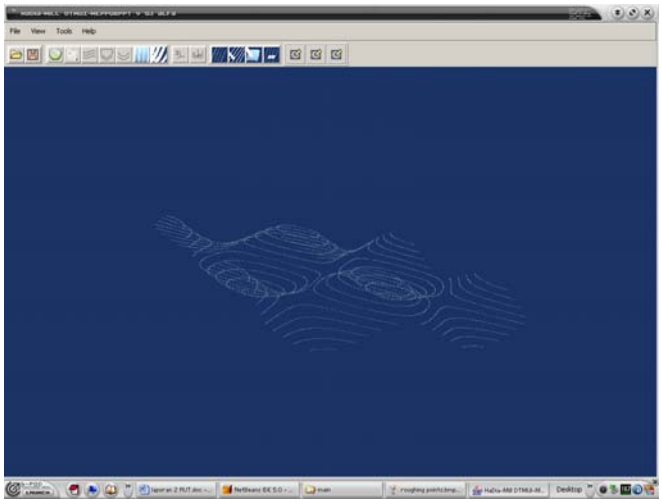
2. jika titik-titik membentuk kurva terbuka

3. jika tidak ada lagi segitiga yang berpotongan

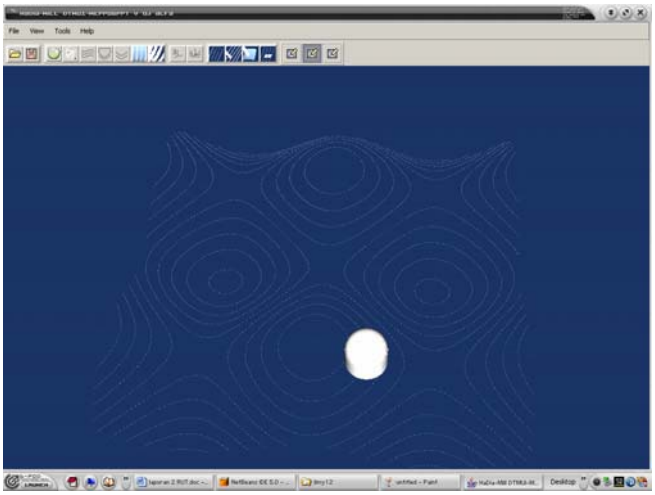


Titik-titik yang diperoleh ini kemudian disimpan dalam sebuah struktur data *Vector* misalnya. Berikutnya adalah dengan menggunakan titik-titik ini, dapat dibuat visualisasi *roughing points* dan simulasi pergerakan model pahat untuk proses *roughing.* Hal ini dapat dilakukan menggunakan cara yang sama dengan visualisasi dan simulasi pergerakan model pahat pada CC-point di atas.

Berikut ini adalah gambar yang diambil dari visualisasi *roughing points* dan simulasi pergerakan model pahat dengan menggunakan sistem-CAM yang dibangun, Hadia-Mill.



Gambar 3: Roughing points pada tampak samping



Gambar 4: Simulasi pergerakan model pahat pada proses roughing

Kesimpulan

Metode brute-search menghasilkan cc-point dalam waktu yang lebih cepat ketimbang adjacent search, namun keterhubungan posisi antara satu cc-point dengan cc-point lainnya tidak dapat diketahui secara langsung. Sedangkan dengan adjacent search, dihasilkan cc-point yang keterhubungan posisi antara satu cc-point dengan cc-point lainnya, urutan, langsung diketahui.

Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Kementrian Negara Riset dan Teknologi (KNRT) yang telah membiaya sebagian dari penelitian ini melalui Riset Unggulan Terpadu XII 2005-2006.

Daftar Acuan

[1]. Byoung K.C, Robert B.J, *Sculptured Surface Machining.* Kluwer Academic Publishers, 1998.
[2]. Dejonghe P., An integrated approach for tool path planning and generation for multi-axis milling, ISBN 90-5682-315-9, PhD-thesis, K.U. Leuven, Leuven 2001.
[3]. Kiswanto G., Tool path generation for multi-axis milling based on faceted models, ISBN : 90-5682-449-X, K. U. Leuven, Leuven 2003.
[4]. Lauwers B., Kiswanto G., Kruth J. -P., Development of five-axis milling tool path generation algorithm based on faceted models, *Annals of the CIRP*, vol. 52, no. 1, pp. 85-88, 2003.
[5]. Hwang J.S., Interference-*free tool-path generation in the NC machining of parametric compound surfaces*, Computer Aided Design, 1992, vol. 24, no. 12, pp. 675-676.
[6]. Hwang J.S., Chang T.-C., *Three-axis machining of compound surfaces using flat and filleted endmills*, Computer Aided Design, 1998, vol. 30, no. 8, pp. 641-647.
[7]. Jensen C.G., Mullins S.H., Anderson D.C., *Scallop elimination based on precise 5-axis tool placement,* orientation *and step-over calculations*, ASME-Advances in Design Automation, 1993, vol. 65-2, pp. 535-544.
[8]. Kiswanto G., Kruth J.-P., Lauwers B., *Tool path generation for 5-axis milling based on faceted models*, Journal of Engineering, Vol.1., 2002.
[9]. Kruth J.-P., Klewais P., *Optimization and dynamic adaptation of the cutter inclination during 5-axis milling of sculptured surfaces*, Annals of CIRP, 1994, vol. 43/1, pp. 443-448.
[10]. Lai J.-Y., Wang D.-J., *A strategy for finish cutting path generation of compound surfaces*, Computer in Industry, 1994, no. 25, pp. 189-209.
[11]. http://www.vtk.org