Pengembangan Modul Pembuat Lintasan Pahat Pemesinan Awal (*Roughing*) 3-Axis dengan Metode Paralel pada Sistem CAM (*Computer Aided Manufacturing*) berbasis Model Faset 3D dengan Simulasi Pergerakan Pahat

Gandjar Kiswanto, Abdurrasyid Mujahid
Departemen Teknik Mesin Universitas Indonesia,
Kampus Baru UI – Depok 16424
gandjar_kiswanto@eng.ui.ac.id

Abstrak

Salam proses pemesinan salah satu tahap yang harus dilakukan adalah proses pemesinan awal atau biasa disebut 'roughing'. Tahap ini bertujuan mengilangkan material lebih dari benda kerja secepat mungkin mendekati bentuk akhir yang dikehendaki sesuai dengan yang dispesifikasikan. Laboratorium Teknologi Manufaktur dan Otomasi Departemen Teknik Mesin – Universitas Indonesia mengembangkan sistem-CAM berbasis model faset 3D. Untuk melengkapi modul pembuat lintasan pahat akhir (finishing) 5-axis yang telah diimplementasi terlebih dahulu maka dikembangkan modul untuk tahap pemesinan awal. Salah satu teknik pembuat lintasan pahat pemesinan awal adalah dengan metode/teknik Paralel. Teknik ini secara bertahap memotong material lebih benda kerja dalam setiap layer (lapis) pada interval vertikal tertentu, dengan arah pola pemotongan paralel antara satu lintasan pahat dengan lintasan pahat berikutnya. Untuk memastikan titik-titik potong yang dihasilkan, maka perlu visualisasi titik-titik tersebut yang menjadi lintasan pahat roughing, serta simulasi pergerakan pahatnya di atas model produk.

Kata kunci: *sistem CAM, roughing, faset 3D, Java, VTK.*

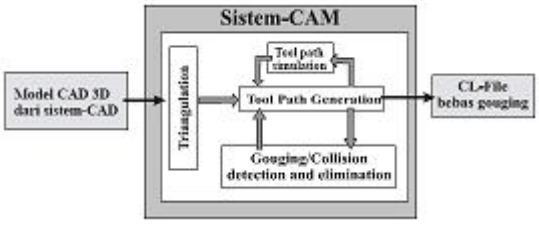
1. PENDAHULUAN

Proses *roughing* adalah tahap awal sebelum masuk ke tahap *finishing* dalam proses pemesinan. Jika *finishing* mengehendaki hasil akhir yang halus, sempurna, dan akurat, maka *roughing* bertujuan membentuk materi mentah dalam waktu yang singkat. Selain itu, *roughing* memakan materi mentah dalam cakupan wilayah yang luas. Dengan demikian, teknik-teknik yang digunakan pada tahap *roughing* dioptimalkan agar pahat bisa memakan material dalam wilayah yang luas dan dalam waktu singkat. Materi hasil bentukan *roughing* memang masih kasar, masih berupa bentuk mentah. Proses penyempurnaan ini dilanjutkan dalam *finishing*.

Tulisan dalam paper ini diawali dengan pengenalan arsitektur sistem CAM yang sedang dikembangkan dimana modul *roughing* diintegrasikan. Kemudian dilanjutkan dengan penjelasan algoritma dalam teknik parallel. Langkah-langkah teknis pengerjaan teknik ini kemudian dijelaskan secara singkat dalam impelementasi program menggunakan bahasa Java (J2SDK) beserta visualisasi dan simulasinya. Bagian terakhir ditutup dengan kesimpulan dan rencana kerja berikutnya (*further works*).

2. ARSITEKTUR SISTEM

Dalam [1] dijelaskan bahwa sistem CAM yang dikembangkan membaca model faset yang telah jadi dalam bentuk format STL. Kemudian dilakukan proses pembuatan lintasan pahat diikuti oleh simulasi pergerakan pahat, identifikasi interferensi model pahat dan model faset. Setelah lintasan pahat dinyatakan bebas interferensi (*gouging*) maka *cutter-location file* (CL-file) dapat dibuat. Alur kerja ini dapat secara singkat dapat dilihat pada gambar 1 di bawah ini.



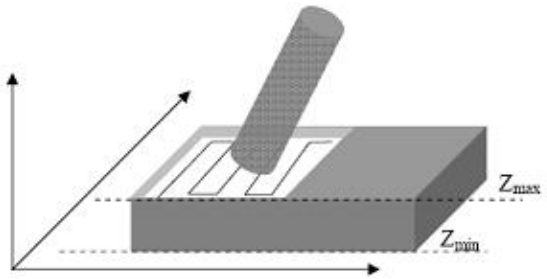
Gambar 1 Alur proses pembuatan lintasan pahat dari sistem-CAD ke sistem-CAM berbasis model faset 3D yang dikembangkan

Untuk itu, modul-modul yang dikembangkan terdiri dari empat buah modul, ditambah kernel (*database*) dan sistem interaksi mesin-manusia atau *graphical user interface* (GUI). Keempat modul tersebut adalah modul simulasi (*simulation module*), modul perencana lintasan pahat (*tool path planning module*), modul pembuat lintasan pahat dan pendeteksi dan penghilang gouging (*tool path generation module*), serta modul pembacaan triangulasi.

Implementasi *roughing* sendiri dalam tulisan ini termasuk ke dalam modul perencana dan pembuat lintasan pahat dan modul simulasi. Untuk itu, keluaran yang diharapkan adalah titik-titik koordinat yang menjadi lintasan pahat menggunakan teknik parallel, beserta visualisasi titik-titik tersebut dan pergerakan pahat di atas model produk untuk mensimulasikan pekerjaan *roughing* pada mesin NC.

3. ANALISIS TEKNIK PARALLEL ROUGHING

Teknik ini disebut dengan *parallel* karena pahat akan bekerja menggerus material dengan arah yang sejajar (paralel) dengan sumbu X dan sumbu Y. Lintasan pahat yang dihasilkan mirip dengan gerakan zig-zag seperti pada gambar di bawah.



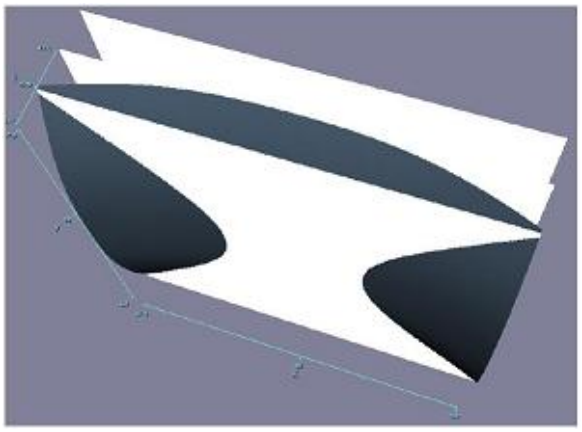
Gambar 2 Ilustrasi konsep dasar roughing *dengan teknik parallel*

Seperti proses *roughing* pada umumnya, pahat secara bertahap, *layer* per *layer* ke arah vertikal, akan memakan material dari Z_{max} sampai Z_{min} dengan interval tertentu sehingga terbentuk model kasar hasil *roughing*. Beberapa hal penting yang harus diperhatikan untuk menghasilkan lintasan pahat ini antara lain:

a. Lintasan pahat tidak boleh mengenai bagian model yang tidak boleh terpotong. Bagian ini biasanya berbentuk 'gundukan' material (disebut juga dengan *island*). Untuk itu, sebelum sampai mengenai gundukan ini, pahat harus diangkat sedemikian sehingga cukup aman untuk menghindari persinggungan.
b. Untuk mengetahui ada tidaknya gundukan pada suatu *layer* Z_i, maka harus ditentukan perpotongan antara model dengan bidang pada *layer* Z_i. Perpotongan ini akan menghasilkan titik-titik potong yang menjadi rambu bagi pahat supaya tidak bersinggungan dengan gundukan. Bagaimana menghasilkan titik-titik potong ini, secara ringkas bisa dilihat pada bagian selanjutnya di bawah.
c. Lintasan pahat yang dihasilkan sebenarnya adalah sekumpulan titik-titik koordinat *x,y,z* yang berada di atas model (yang boleh dipotong).
d. Variabel yang menentukan kerapatan pemotongan adalah *slicing interval* dan *step over*. *Slicing interval* menentukan jarak antara Z_i dengan Z_{i+1}, sedangkan *step over* menentukan jarak antara Y_c dengan Y_{c+1}. Ada pula variabel *tool offset z-plane* yang menentukan ketinggian pengangkatan pahat (misalnya pada saat menghindari persinggungan dengan *island*).

Bagaimana menghasilkan titik-titik potong antara model dengan bidang pada *layer* Z_i?

Untuk menghasilkan titik-titik ini, secara sederahana dapat dilakukan dengan mengiriskan suatu bidang $Z = c$ (dalam sistem koordinat XYZ) dengan model. Irisan ini akan menghasilkan titik-titik perpotongan antara bidang *Z* dengan model. Perhatikan ilustrasinya pada gambar 3 di bawah.



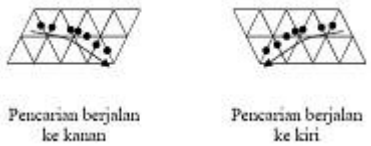
Gambar 3 Perpotongan antara bidang-bidang z dengan model

Karena model berbasis faset segitiga, maka perpotongan terjadi antara bidang pada *layer 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 diinginkan. Jadi, masukan (*input*) pada proses menentukan titik-titik ini adalah interval antar bidang potong. Interval antar bidang potong ini disebut dengan *slicing interval*.

Pencarian titik-titik potong 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 *u*, misal sisi *s*, yang berpotongan dengan bidang *Z*, kemudian tentukan titik perpotongannya. Simpan titik ini pada sebuah struktur data *Vector*.
2. Cari segitiga yang memiliki sisi *s* sebagai salah satu dari tiga sisi-sisnya selain segitiga *u* di atas.
3. Lakukan kembali langkah no. 1 di atas.

Pencarian titik-titik di atas akan tampak seperti pada gambar di bawah ini.



Gambar 4 Ilustrasi pencarian titik-titik potong yang bersebelahan

Pencarian 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*.

Pencarian ini dilakukan untuk setiap kali pengirisan bidang pada *layer Z* dengan model faset. Sehingga dengan demikian jika seluruh *layer Z* selesai dilakukan pengirisan, maka akan diperoleh titik-titik potong antara model dengan bidang pada *layer-layer Z*.

Bagaimana menghasilkan titik-titik lintasan pahat *parallel roughing*?

Berikut ini adalah langkah-langkah untuk menghasilkan titik-titik *parallel roughing* yang dibuat dalam penelitian ini.

1. Tentukan nilai *step over* ***SO*** dan *slicing interval* ***SI***.
2. Tentukan arah awal pemotongan, misalnya dari Y_{min} ke Y_{max} dan dari X_{min} ke X_{max}. Arah pemotongan berikutnya harus bisa berubah secara otomatis, sehingga membentuk zig-zag.

3. Tentukan titik-titik potong antara model dengan bidang z_i. Titik-titik ini adalah hasil perpotongan model dengan *layer-layer* Z yang dijelaskan pada bagian sebelumnya, sehingga dengan demikian akan memudahkan *parallel roughing*.
4. Pada setiap pemotongan *slice* z_i, tentukan semua titik *x,y,z* yang masuk dalam wilayah pemotongan (*cutted plane*) dan menghindari persinggungan dengan *island*. Titik-titik potong dari langkah 3) akan sangat membantu menghindari persinggungan pahat dengan *island*. Simpan semua titik ini dalam sebuah struktur data *Vector*.
5. Setiap penghindaran dari *island* pahat diangkat sejauh *tool-offset*. Koordinat *x,y,z* yang menjadi letak mata pahat setelah diangkat juga disimpan dalam *Vector*.
6. Semua titik hasil langkah 4) dan 5) ini disebut *parallel roughing points*.

Parallel roughing points ini selanjutnya akan digunakan untuk visualisasi dan simulasi proses *roughing* dengan teknik tersebut.

4. IMPLEMENTASI PEMROGRAMAN

Teknik ini diimplementasikan dalam bahasa pemrograman Java menggunakan J2SDK versi 1.5, sesuai dengan rancangan keseluruhan pengembangan sistem CAM berbasis model faset 3D ini.

Teknik parallel ini diimplementasikan dalam sebuah *method* yang diberi nama `parallel()`. Di dalam method ini, ada beberapa variabel lokal yang terlebih dahulu harus ditentukan nilainya, antara lain:

- `step_over` : interval antara y1 dan y2
- `slicing_interval` : interval antar *layer*
- `maxCoord` : sebuah array yang menyimpan koordinat *x, y,* dan *z* (saling lepas) tertinggi dari titik-titik dalam model.
- `minCoord` : sama halnya dengan `maxCoord` untuk koordinat *x,y,z* terendah.
- `y_min_to_max` : variabel *boolean* untuk menentukan arah lintasan pahat apakah dari Y_{max} ke Y_{min} atau sebaliknya.
- `x_min_to_max` : variabel *boolean* untuk menentukan arah lintasan pahat apakah dari X_{max} ke X_{min} atau sebaliknya.
- `Vector` : sebuah instance dari strutktur data jenis *Vector* untuk menyimpan titik-titik hasil roughing.

Selanjutnya implementasi *method* ini dibuat ringkasannya sebagai berikut.

```
public synchronized void parallel(){
        // inisialisasi variables

        /* looping mulai dari z tertinggi hingga z terendah dengan slicing interval
           tertentu
        */
        for (float z=maxCord[2]; z>=minCord[2]; z-= Configuration.SLICING_INTERVAL ){

            /* pada current z, ambil titik-titik potong model dengan bidang */
            Vector roughingPoints_at_Z= getRoughingPoints(z);
            // ...
            /* looping dimulai, berjalan dari yStart hingga yEnd */
            for (float y= yStart; ; y+=stepOver){
                /* vector berisi batas-batas perpotongan dengan kemungkinan island
                   jumlahnya bisa genap (2,4,dst) bisa jg ganjil (1,3,dst)
                */
                Vector xxx= new Vector();
                // ...
                if(xxx.size()==0){
                    // ...
                    /* masukkan ke dalam parallel_points bagian atas yang dipotong
                       pertama kali, dimana tidak ada island.
                    */
                } else {
                    /* mengecek di perbatasan titik potong dengan island, bagian mana
                       yang cutted, apakah bagian luar atau bagian dalam dari perbatasan itu.
                       x0, y, z --> titik potong bidang terhadap model pada cuurent z,
                       current y, dengan x=x0.
                    */
                    // ...
                    do{
                        Vector vector_bucket= bucket.getBucket(bucket_x, bucket_y);
                        for(int i=0; i<vector_bucket.size(); i++){
                            /* mencari titik-titik (vertek) yang berpotongan dengan
                               current y, atau dekat dengan current y; */
                            // ...
                            /* evaluasi posisi titik-titik ini terhadap x0 dan current z;
                               dimana letak cutted plane? apakah setelah x0 atau sesudah.
                            */
```

```
                    // ...
                }
                // ...
            }
            while(counter < state.length);
            // ...
            /* ------------------------------------------------------------
             start: kurang_x0 bernilai true berarti plane sebelum
             x0 pertama dipotong atau cutted plane; */
            if(kurang_x0){
                float x;
                /* start: untuk path dari x min ke x max */
                if(x_min_to_max){
                    int i=0;
                    do{
                        /* tentukan x minimum dan x0 (x sebelum ketemu island) */
                        //...
                        /* masukkan nilai koordinat dari x sampai x0; */
                        for (; x<x0; x++)
                            parallel.addVertex(new Vertex(x,y,z));
                        /* sesuaikan letak tool untuk menghindari island; */
                        // ...
                        /* masukkan ke dalam parallelPoints, kecuali jika
                           batas akhir sama dengan max x. karena akan disambung
                           dengan next y; */
                        i++;
                    }
                    while(i<xxx.size());
                    /* end: */
                }
                /* start: untuk path dari x max ke x min */
                else{
                    int i=xxx.size()-1;
                    do{
                        // sama seperti di atas
                        // ...
                    }
                    while(i >= 0);
                }
                /* end: */
                x_min_to_max= !x_min_to_max; // path berbalik arah x;
            }
            /* ------------------------------------------------------------ */

            /* ------------------------------------------------------------
            sebaliknya, cutted plane adalah setelah x0 pertama; */
            else{
                float x;
                /* path dari x min to x max; */
                if(x_min_to_max){
                    int i=0;
                    do{
                        // ...
                    }
                    while(i<xxx.size());
                } else{ /* path dari x max to x min; */
                    int i= xxx.size()-1;
                    do{
                        // ...
                    }
                    while(i>=0);
                }
                x_min_to_max= !x_min_to_max; // path berbalik arah x;
            }
            /* ------------------------------------------------------------ */
        }
    }
    /* vector berisi vector parallel points */
    parallelPoints.add(parallel);
}
```

Gambar 5 Implementasi method parallel.

5. VISUALISASI DAN SIMULASI

Visualization Toolkit (VTK)

Implementasi proses *roughing* beserta visualisasi dan simulasi bertujuan untuk memberikan tampak visual bahwa titik-titik yang dihasilkan adalah benar. Untuk visualisasi dan simulasi ini, digunakan *class-class* dari *library* VTK (*Visualization Toolkit*). VTK sangat banyak digunakan untuk keperluan *image processing*, visualisasi dan simulasi proses, *rendering* objek, dan sejenisnya.

VTK dikembangkan pertama kali untuk bahasa pemrograman *C++*. Namun, karena semakin tingginya kebutuhan untuk dapat menggunakan VTK pada bahasa pemrograman yang lain, maka VTK melakukan *porting* ke dalam bahasa pemrograman yang lain. Bahasa pemrograman tersebut antara lain: *Java*, *Python*, dan *Tool Command Language*(*TCL*).

Tidak hanya dapat berjalan pada bahasa pemrograman lain, *vtk* juga telah tersedia untuk komputer dengan *Operating System (OS)* yang berbeda, bahkan telah tesedia untuk komputer dengan arsitektur yang berbeda dengan arsitektur komputer *PC*. VTK telah dapat berjalan dengan baik pada *OS Windows*, dan berjalan cukup baik untuk komputer dengan *OS* turunan *UNIX*. VTK juga dapat berjalan dengan cukup baik, meskipun masih terdapat banyak kekurangan pada komputer dengan arsitektur *Power PC* dengan *Mac Tiger OSX*.

Meskipun *library* VTK memiliki lisensi *open source* yang identik dengan *OS Linux*, namun dukungan secara penuh masih dipegang oleh bahasa pemrograman *C++* dengan *OS Windows*. Hal tersebut dapat dilihat karena semua program *vtk* yang ditulis dalam bahasa apapun akan kembali di*porting* ke dalam bahasa pemrograman *C++*. Untuk bahasa pemrograman Java, *libarary* untuk mengakses kelas-kelas VTK terdapat dalam *archieve vtk.jar*.

Visualisasi dan simulasi

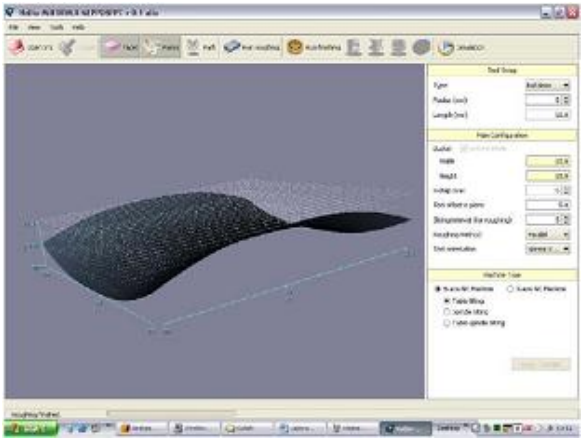
Untuk melakukan visualisasi, beberapa class dalam VTK yang diperlukan antara lain: `vtkPoints`, `vtkPolyVertex`, `vtkUnstructuredGrid`, `vtkDataSetMapper`, `vtkActor`, `vtkRenderer`, dan `vtkRenderWindow`.

Titik-titik *parallel roughing* yang sudah dihasilkan disimpan dalam struktur data `vtkPolyVertex` dalam bentuk (tipe data) `vtkPoints`. Selanjutnya untuk menampilkannya secara grafis di layar, digunakan class `vtkActor` dan `vtkRenderer` untuk merender objek-objek dari titik tersebut. `vtkRenderWindow` sendiri berperan ibarat sebagai *stage* yang menjadi tempat objek tadi divisualisasikan.

Adapun untuk simulasi, class-class yang digunakan sama dengan simulasi, ditambah dengan `vtkLineSource`, `vtkTubeFilter` dan `vtkSphereSource`. Ketiga class tambahan ini digunakan untuk memodelkan bentuk pahat yang digambarkan dengan model silinder ditambah setengah bola di ujungnya.

Simulasi pergerakan pahat berarti pergerakan objek silinder tersebut di atas model produk. Untuk menghasilkan efek silinder yang bergerak, maka dilakukan *looping* perubahan titik pusat silinder sesuai dengan titik-titik potong yang dilaluinya. Dengan demikian akan tampak pergerakan pahat di atas titik-titik *parallel roughing*.

Di bawah ini adalah *screenshot* implementasi visualisasi *parallel roughing* dan simulasi pergerakan pahatnya.

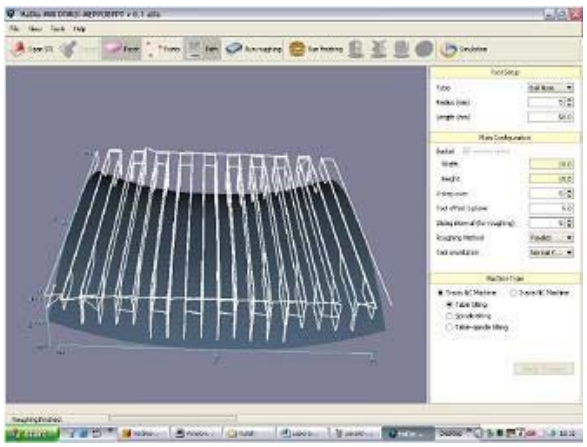


Gambar 6 Visualisasi titik-titik parallel roughing

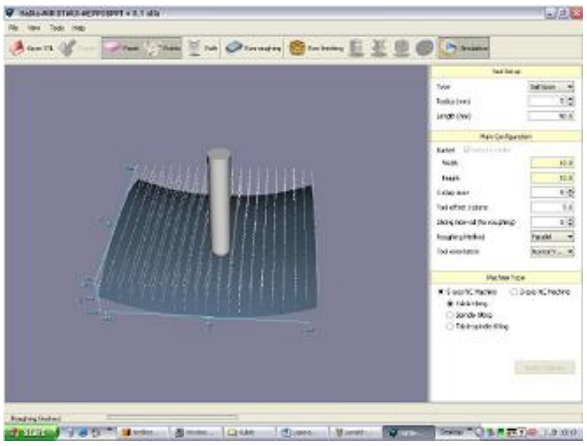
Titik-titik putih pada gambar di atas adalah titik-titik hasil parallel roughing. Untuk memastikan tidak ada kesalahan, bisa dilihat bahwa tidak ada titik-titik yang jatuh di bawah model produk. Jika ada, berarti *roughing* menabrak *island*. Ini harus dihindari.

Pada gambar 7 di bawah, titik-titik *parallel roughing* divisualisasikan dengan garis yang menjadi lintasan pahat. Pada gambar tersebut tampak lintasan pahat berjalan seperti alur zig-zag, ke arah X_{max} (menjauhi pengamat) dan ke arah X_{min} (mendekati pengamat) secara bergantian.

Adapun pada gambar 8 di bawah adalah hasil pengambilan gambar (*screenshot*) pada saat simulasi pergerakan pahat berlangsung. Pahat dimodelkan dengan bangun silinder berwarna abu-abu, dengan bangun setengah bola pada ujungnya (model pahat *end-ball*).



Gambar 7 Visualisasi ilntasan pahat parallel roughing



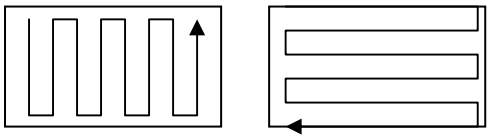
Gambar 8 Simulasi pemotongan parallel roughing

Ketiga gambar di atas adalah hasil implementasi pemrograman menggunakan bahasa Java J2SDK dipadukan dengan *clas-class* dari *library* VTK untuk keperluan visualisasi dan simulasi.

6. KESIMPULAN DAN PENGEMBANGAN LANJUTAN

Dari penjelasan-penjelasan di atas, *parallel roughing* adalah salah satu teknik yang digunakan untuk memakan material dalam waktu cepat dan dengan cakupan wilayah 'makan' yang cukup luas. Arah pemotongan zig-zag dan pengangkatan untuk menghindari *island* dapat dengan cepat membuang material, karena pahat bergerak secara parallel ke arah titik-titik yang paling dekat. Ukuran diameter pahat bisa disesuaikan dengan kebutuhan untuk mempercepat waktu membuang material.

Saran untuk pengembangan berikutnya adalah adaptasi arah pemotongan sesuai dengan bentuk model produk. Artinya, jika pada paper ini arah pemotongan berjalan searah sumbu koordinat X, maka bisa dicoba arah lain yang searah sumbu koordinat Y. Perhatikan gambarnya berikut ini.



Gambar 9 Variasi arah pemotongan parallel (a) vertikal (b) horizontal

Ada beberapa kondisi di mana arah pemotongan vertikal bekerja lebih efisien dibandingkah arah horizontal. Demikian pula sebaliknya. Untuk itu, pemilihan arah yang tepat dapat memaksimalkan hasil proses *parlallel roughing*.

DAFTAR ACUAN

[1] Kiswanto, Gandjar. (2005). **Pengembangan dan Pembuatan Sistem CAM (Computer-Aided Manufacturing) yang Handal berbasis Model Faset 3D untuk Pemesinan Multi-axis dengan Optimasi Orientasi Pahat dan Segmentasi Area dan Arah Pemesinan**, Laporan Kemajuan RUT XII Tahap II Tahun 2005, Kementrian Riset dan Teknologi dan LIPI.

[2] www.vtk.org