OptiPath - A Toolpath Optimization Software

Reference Manual

Hongcheng Wang (hwang13@uiuc.edu)
University of Illinois at Urbana-Champaign
Apr. 20, 2002

Contents

0

Overview 1

About the Software

The software concerns the problem of 2D toolpath optimization, called OptiPath. It is based on my Master's work with Professor James A. Stori in the Department of Mechanical and Industrial Engineering at University of Illinois at Urbana-Champaign.

This software is an implementation of a metric-based approach to 2D tool path optimization for high speed machining [Hongcheng 2001, Hongcheng & James Stori 2002]. OptiPath is written by C/C++, and be debugged in Microsoft VC++ Version 5.0 and above. It can run in Windows 98/2000 or Windows NT 4.0 and above.

The software can optimize 2D convex and non-convex pocketing toolpath from Pro/E Manufacturing software based on curvature and engagement angle metrics, which are both very important to high speeding machining processes. The output toolpath can reach near optimal in terms of these metrics.

About the Manual

The manual describe how to use the software to obtain the near optimal toolpath from original Pro/E cutting location files. It contains the following chapters:

Chapter 1: <Overview> Overview of OptiPath Software

Chapter 2: <Get Started> Describe how to use this software, including installing OpenGL, and run the program

Chapter 3: <Description of Main Functions> Describe the parameters of the main functions.

1

Chapter 4: <Case Studies> In this chapter, several examples are given.

Chapter 5: <Bugs and Future Development> Summarize possible bugs of this software, gives some suggestions for future development.

Finally, <References> List the other references for this manual.

2 Get Started

OpenGL Installment

To run this program, you need first to install the OpenGL in your computer. If you are using Visual C++ 5.0, you need to download the basic opengl library from Opengl website: http://www.opengl.org. If you are using Visual C++ 6.0, you do not need to do so. However, whatever version you use, you still need to download GLUT library from this website since some menu and mouse API are used for friendly user interface in this program, which are GLUT routines.

After downloading the GLUT, use winzip, or some other unzipping utility, to decompress the archive. Once you have done so, you should have the following files:

glut.dll
glut.h
glut.lib
glut32.lib
glut32.dll

The next step is to move the files into the appropriate directories, so Visual C++ can find them for linking when compiling your program.

Place the glut.h file in to the C:\Program Files\DevStudio\VC\include\gl directory
place the glut.dll and the glut32.dll file into the C:\Windows\System directory
place the glut.lib and glut32.lib file into the C:\Program Files\DevStudio\VC\lib\ directory.

Thus, OpenGL is completely installed to your computer.

2

Run the Program

To run this program, you only need to double click Toolpath_Spline.exe file in the <debug> directory under Optipath directory, or if you want to edit the source code or add more functions, just double click the workspace file Toolpath_Spline.dsw the directory of OptiPath. Then you can compile and start the program.

This program use the keyboard and mouse input interface, and menu in the graphics interface. After starting the program, there are some hints for using this program. These hints are the brief introduction to the menu.

Please use right button to show the menu!

Tool Path - *Draw the tool path according to the data file*

Quintic-Spline Path - *Interpolate the original toolpath by Quintic Spline Toolpath*

Curvature Graph - *Plot the curvature VS. path length graph(old)*

Engagement Graph - *Plot the engagement angle VS. path length graph(old)*

Mark Bad Point - *Find the bad point according to Metrics*

Begin Simulation - *Begin the global machining process by animation*

Improve Toolpath - *Improve Toolpath based on iterative improvement approach*

Check Final Result - *Check the feasibility of the final toolpath*

Clear Screen - *Clear the graph on the screen*

Quit Program - *Quit the program,*

or press ' q' or 'Q' to quit the program

The Curvature Graph and Engagement Graph are used in debugging the program. They are not active now.

2

The input/Output parameters will be discussed in next chapter. After input the cutter location file and toolpath parameters, a black graphical window will display. Press the right button of the mouse, you will a pop-up menu:

Input Parameters

Then the system will let you input the parameters as following:

---INPUT PARAMETERS---

*Please enter the cutter location file name:*_____

The input file here is the cutting location file in text format which is generated from Pro/E Manufacturing by preprocessing. The input file will be discussed in later in <Input File> and in next chapter - Description of Main Functions - <Cutter Location Generation from Pro/E Manufacturing>.

*Please input the curvature weight (0,50,100):*_____

The curvature weight parameter is the parameter that describe the weight of curvature in the toolpath optimization procedure with respect to that of engagement angle. The cost function for optimization is:

$$C = CurvatureWeight \times Curvature + EngagementWeight \times Engagement$$

The engagement weight has default value 1.0. For example, if the curvature weight is 0, the cost is only a function of engagement. The program can be modified to adapt to any weight to curvature and engagement. For the moment, three weights: 0, 50, 100 are given for your choice.

*Please input spiral type of path: 1-out 0-in:*_____

(Be consistent with the file name)

This parameter is a redundant one in this test phase. The program can detect if the input file is a spiral in or spiral our tool path according to the file name. For example, if the input file is: exampe1_in.txt, then the program will know it is a spiral in toolpath. So this parameter should BE CONSISTANT WITH THE FILE NAME.

Visualize the simulation (Yes/No):____

This parameter is a boolean parameter to tell if you need to visualize the simulation and optimization process in graphics. In normal case, you need the visualization.

Input File

The input file is the cutter location file, which comes from Pro/E Manufacturing, and is modified to satisfy our requirements. The how-to is discussed in next chapter in detail. Here we only give the format of the input file.

The input file is composed of successive segments of the toolpath. It includes two basis elements: line and arc, represented by ‘l’ and ‘c’ in the input file. The x, y, z coordinates of each cutter location is following with element description. The format is like:

3

l
5.019057 3.162912 -0.400000
l
5.019057 0.896550 -0.400000
...
c
3.750000 1.900000 -0.400000
l
3.940050 1.962298 -0.400000
l
4.130100 2.024596 -0.400000
c
3.950057 2.381786 -0.400000
l
4.350057 2.381786 -0.400000

Output File

There are two output files after running the optimization program: *finaldata.txt* and *results.txt*.

In *finaldata.txt*, it records the x, y coordinates, Engagement angle and Curvature value of all the data points of the FINAL resulting toolpath, its format is like following:

3

Index X Y Engagement Curvature
0.000000 402.227420 148.000073 6.283185 0.000000
1.000000 395.400900 148.000073 3.122893 0.000000
2.000000 389.400900 148.000073 3.085493 0.000000
3.000000 383.400900 148.000073 3.085493 0.000000
...
1580 506.962065 252.530570 0.972398 0.002265
....
1964.000000 418.698687 302.373742 0.000000 0.000000
1965.000000 412.699270 302.382933 0.000000 0.000000
1966.000000 406.699514 302.392624 0.000000 0.000000

For this example, there are 1966 discrete points in the last round of the optimization.

File *results.txt* records the values of total engagement variation and curvature variation for each optimization round, each line in the file is organized in the following order:

Optimization Time - Total u value - Total Cost - Total Engagement Variation - Total Curvature Variation - Total Points

3

1 26.803280 918.834603 918.834603 9.264937 1186
2 4.041026 899.307980 899.307980 9.156656 1180
...
7 0.098958 891.562329 891.562329 8.982535 1178
...
11 0.104702 890.885262 890.885262 8.981202 1178
12 0.078958 890.776931 890.776931 8.982491 1178

The u value is the scale value to move the bad point in the optimization. The engagement variation and curvature variation is the difference between the simulation value and the target value for engagement angle and curvature, respectively.

All the plots in the thesis and paper are based on these two files.

4 Description of Main Functions

The software includes the following files:

readfile.c

quicksort.c quicksort.h

quintic.c quintic.h

toolpath.c toolpath.h

The *readfile.c* is used to generate the input file for toolpath optimization. The *quicksort.c* and *quicksort.h* files are used to sort the simulation points according to their cost to find bad point for each improvement. The *quintic.c* and *quintic.h* files are used for quintic spline toolpath generation from the original line and arc segments. The *toolpath.c* and *toolpath.h* are the main files used for toolpath optimization.

Cutter Location Generation from Pro/E Manufacturing

readfile.c

This file is used for cutter location generation. The input for this file is the original cutter location file saved from Pro/E Manufacturing.

P. S. This file works semi-automatically, i.e., it needs some manual manipulating. You need to find the outmost profile shape for the machined pocket manually.

The format of Pro/E cutter location file are discussed in detail in reference [Hongcheng Wang, 2001].

4

Spline Toolpath Generation

void computeQuinticCurve()

This function is used to generate equal_distance discrete points along the near-arc length quintic spline toolpath. The distance between two discrete points is defined by a global variable - *constant_distance*. You may need to change this distance to get best unit-tangency property for a specific toolpath.

void computeQuinticTangent()

The function is used to find the tangent of points on the quintic spline toolpath. The tangents are used later on to find the neighborhood points of one bad point.

void computeCubicCurTor()

This function is used to find the curvature of points on the cubic spline toolpath. The quintic spline toolpath adopts this value as its curvature. The curvature can also be approximated by the three-point algorithm.

Simulation and Optimization

double calculateCurvature(VERTEX P1, VERTEX P2, VERTEX P3)

Input to this function are three point P1, P2, P3 defined by VERTEX structure. This function uses the three point algorithm (use a circle to approximate the neighboring three control points) to find the curvature of P2. The curvature value is returned.

void drawCurvature(), void drawEngagement(), void drawAxis()

These three functions are used to draw the plots of engagement angle and curvature in a xyz coordinate system. This is used for debugging purpose.

void drawWorkpiece();

This function is used to visualize the workpiece pixel-by-pixel. It is mainly used to visualize the boolean operation process in testing the algorithm.

void drawTool(int x, int y);

This function is used to test if the boolean operation is right or not since the drawing will take much time, so it is just used for testing. The input parameter (x,y) is the coordinate of tool position. This function is called by the ***beginSimulation***() Function.

double areaAngle(VERTEX P1, VERTEX P2, PointType P3);

This function takes three points as input and find the area defined by these three points and return the integer. If the area is negative, P3 is on the left of the line defined by P1 and P2, otherwise, it is on the right side of the line.

This function will be used to test the feasibility of the resulting pocket geometry by finding the entry angle and exit angle of the cutter.

void beginSimulation()

This function implements the machining simulation process by the boolean operation between the tool and workpiece. By find the number of 0s and 1s along the cutter profile, the engagement angle can be found. At the same time, the entry angle and exit angle location can be found by the transition of 1 and 0 on the tool circle. By the boolean minus operation between the cutter and workpiece, the workpiece will be updated for each cut. If more accurate simulation results are needed, the swept area can be calculated here. But this will cost more time. Normally, we do not need this since the distance of two simulation points is very small. More detailed discussion can be found in [Hongcheng Wang, 2001].

After the simulation, the cost will be re-evaluated. And the bad point will be found based on the new cost.

void findBadPoint()

This function is used to find the bad point with largest cost. The cost is defined by:

Cost = K1*|Curvature| + K2*|Delta_Engagement|

where, K1 is the curvature variation weight, and K2 is the engagement variation weight.

The ***quickSort()*** algorithm will be called in this function for many times.

void moveLocalBadPoint()

This function implemented the quadric optimization algorithm. The bad point, and its two neighborhood points can be approximated by a quadric parabola. The adjusting the bad point, the optimal solution can be found for this improvement.

int checkUncutArea()

This function is used to check if there is uncut area for this optimization iteration. The exit angle and entry angle will be checked here. A integer value will be returned.

void optimalSimulation()

This function is used to update the bad point using the optimal position. Next improvement will continue.

void findNeighbor()

This function implemented the algorithm described in PP. 63 of the thesis [Hongcheng Wang, 2001]. It is used to find the neighborhood points of the bad point.

void improveToolPath()

This function implements the loop optimization process. The engagement and variation data are saved for each iteration.

void checkFinalResult()

Check the final result by simulation and test the feasibility of the final resulting file.

void drawNewPoint()

Visualize the new position of the bad point.

void fillCircle(double R, int circle_segment)

This function is used to fill the circle using pixels. The circle is define with radius *R* and number of line segments, *circle_segment*, which.

void drawCutter(x, y);

This function uses OpenGL to draw the cutter in the position of *x* and *y*.

void drawToolPath()

This function is used to visualize the toolpath by connecting the control points and draw in lines.

void getVP()

This function implements the initial interpolation of the line and arc segments from cutter location file.

void calculateSpline()

Get the N points name V from the initial interpolation, and approximate the points with near arc-length quintic spline.

void drawSplinePath()

DrawSpline is used to draw the spline segments.

void display(void)

Display function is called by the main event loop to refresh the contents of the display and to show the graphics.

4

void quit(void)

Quit function is called to quit the program.

void resetInput(void)

ResetInput function is called to clear the screen and reset the global variable to 0.

void myinit();

Myinit is called at program invocation and sets to clear the background and set the initial value of some variables.

void usage(void)

Print the usage of the program usage() is called at the program start to print out a short help message on how to use the program

void openFile();

This function is used to open the input cutter location file. This file is re-processed for initial interpolation.

void compare(double tempx, double tempy)

This function is used to get the maximum and minimum value of the input date for the scale of the size

void getPoints()

Get the coordinates of the points

void getCircle(PointType p1, PointType p2, PointType p3)

This function take two end points and one center points of the arc as input parameters, and interpolation the arc using near equal distance line segments.

void getline(PointType p1, PointType p2)

This function take two end points of a line segment as input parameters, and interpolation the line using near equal distance line segments.

double uMini(VERTEX f)

This function is used to find the optimal scale value, *u*, to make the cost function smallest. Input parameter is a vertex *f* and a double u value is returned.

void circleMidpoint(int xcenter, int ycenter, int radius)

This function implemented the midpoint circle algorithm. Two coordinates *xcenter* and *ycenter*, and *radius* are input parameters.

void Initialization();

This function is used to initialize the tool and workpiece represented by pixel matrix.

4

void InitializeParameter()

This function initializes the data for each example. This part should be incorporated into the input parameter part. But due to time, here we just give some examples. The *profile_p* is the matrix for the profile points. *profile_num* is the number of the profile points. *stationary_num* means the points which are kept un-changed in the optimization. For example, for spiral-out toolpath, we will not change the first several points. *TROW* is the cutter size. *Engagement_target* is the target engagement angle. For different cutter size, we need to chose appropriate engagement_target.

This part can be done in a program automatically to decrease manual manipulating and eliminate redundancy. This should be a part of future work.

5 Case Studies

Three examples are given to illustrate the approach. Several parameters are used to describe each example.

- **Geometry Type**: There are three different geometries used in the case studies. They are referred to as: Island-Pocket, No-Island-Pocket, and Rectangular-Pocket.
- **Toolpath Type**: Both spiral-in and spiral-out strategies are considered for each geometry.
- **Cutter Diameter**, *D*, and Target Engagement, θ_t: The cutter diameter and stepover decide the target engagement. We list the cutter diameter and target engagement angle here. Two different target engagements are discussed for the island pocket example**.**
- Curvature Weight, K_c: There are three normalized curvature weights discussed for each example, 0, 0.50 (or 50) and 1.0 (or 100). The engagement weight for all examples is 1.0. Thus, we consider cases where curvature receives no consideration, as well as cases in which the engagement and curvature metrics are equally weighted.

For each example, we give three plots: cost plot, toolpath plot and curvature and engagement plot. In the cost plot, the total engagement, total curvature, total cost, toolpath length, and total *u* value are plotted separately. The total cost is the combination of engagement and curvature cost.

5

Island Pocketing

Spiral-out Island Pocket Example

Cutter Diameter, $D = 80$

Target Engagement, $\theta_t = \pi/2$

Curvature weight, $K_c = 0$

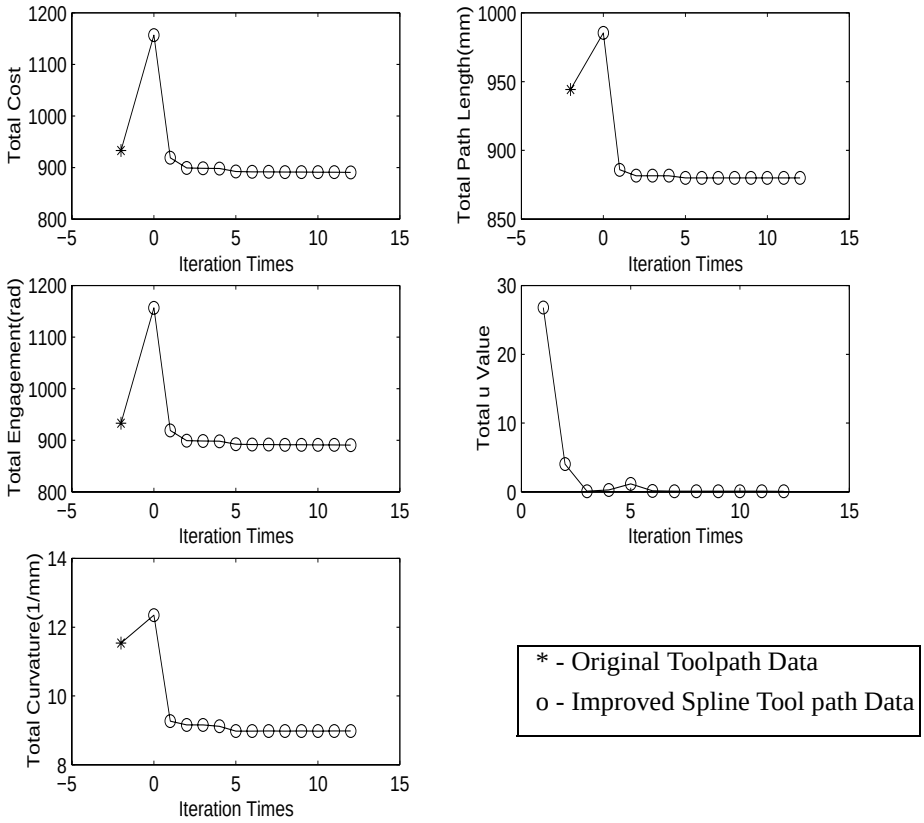

FIGURE 1. Island-Pocket Algorithm Progression, Spiral Out, $\theta_t = \pi/2$, $K_c = 0$

5

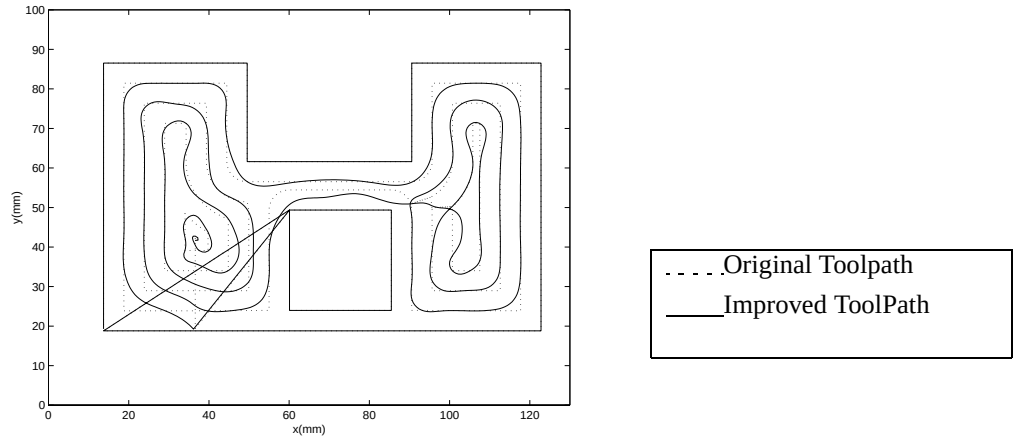

FIGURE 2. Island-Pocket Toolpath, Spiral Out, $\theta_t = \pi/2$, $K_c = 0$

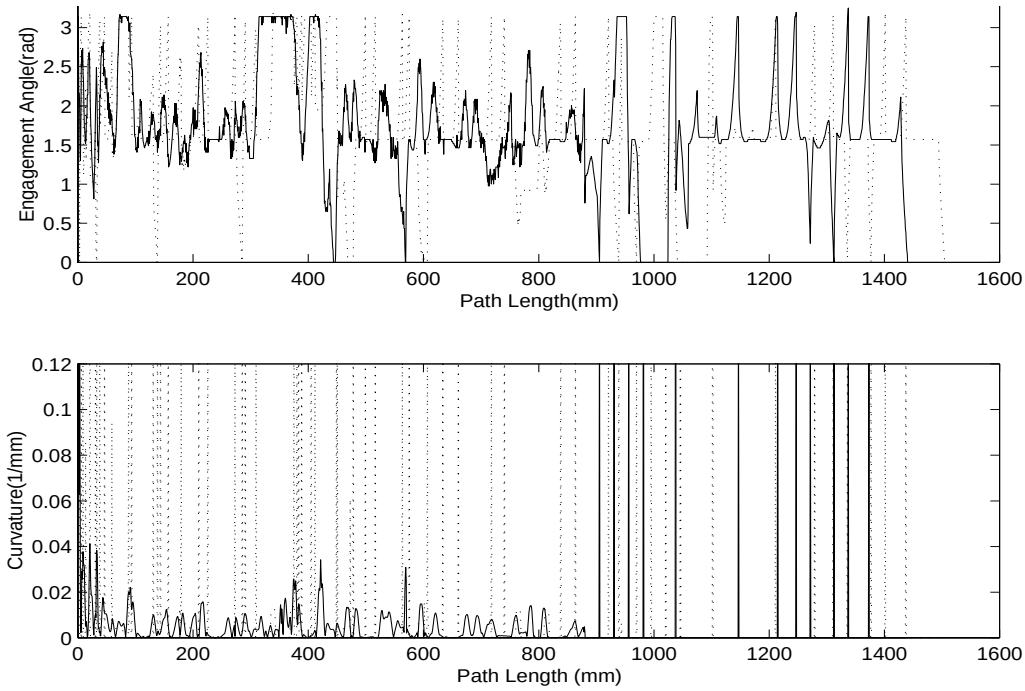

FIGURE 3. Island-Pocket Metrics, Spiral Out, $\theta_t = \pi/2$, $K_c = 0$

5

The plot for curvature_weight = 50, 100, and for spiral-in toolpath can be found in reference [Hongcheng Wang 2001]. A summary table is illustrated in PP.69.

No-Island Pocketing

Please see thesis [Hongcheng Wang 2001].

Rectangular Pocketing

Please see thesis [Hongcheng Wang 2001].

Bugs and Future Development 6

Known and Unknown Problems

The programs are still incomplete due to time. Many problems are still need improved:

1. The I/O interface is not very convenient
2. The program needs to be modified into completely Object-Oriented Programming.
3. The memory is sometimes a problem in debugging.

Copyright and License

The software is only for research purpose. Any one who uses this software for research work should cite this paper [Hongcheng Wang & James Stori, 2002]

This software is a part of the work when I was pursuing my Master Degree under Prof. James A. Stori. It is only used for our research purpose. The program can be freely distributable and be provided without guarantee or warrantee expressed or implied.

This program is -NOT- in the public domain!

hwang13@uiuc.edu

Department of Mechanical and Industry Engineering

University of Illinois at Urbana-Champaign

References

- Hongcheng Wang, *A Metric-Based Approach to 2D Tool Path Optimization for High Speed Machining*, Master's Thesis, University of Illinois at Urbana-Champaign, 2001.
- Hongcheng Wang, James A. Stori, *A Metric-Based Approach to 2D Tool Path Optimization for High Speed Machining*, International Mechanical Engineering Congress, 2002.