AMPL Toolpaths (ver. 0.9.6)
Created by Newell Moser
March 27th, 2019

Documentation for Developers

Overview

In this guide, I will discuss the layout of the code for *AMPL Toolpaths*. However, this guide will be most useful if the reader/developer has already met a few prerequisites:

- A working knowledge of **C++11** – not just conventional C programming. More specifically, the developer should be familiar with topics like the philosophy of object-orientated programing, data structures (e.g., standard template library, containers, iterators, algorithms, etc.), dynamic memory, friendship/inheritance, polymorphism, and so on. More information can be found in reference manuals or online < http://www.cplusplus.com/doc/tutorial/ >

- A working knowledge of **Qt 5.8**. More specifically, the developer should be familiar with Qt's abstraction of "signals and slots" rather than MS Window's "events and listener" abstraction. Additionally, Qt has an enormous number of classes (or widgets) to work with, all of which come together to offer (relatively) rapid development of a Graphical User-Interfaces (GUI). More information can be found online < https://doc.qt.io/archives/qt-5.8/index.html >

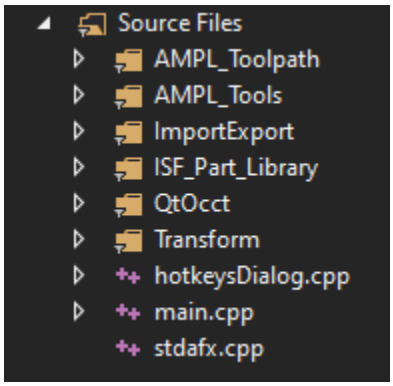
- Experience with developing programs in **Microsoft Visual Studio**. Although Visual Studio is an Integrated Development Environment (IDE) with many similarities to others, like Eclipse IDE, it is useful to learn the specifics of Visual Studio's debugger, performance profiler, configuration files, and so forth. For more information, see Microsoft's online documentation < https://docs.microsoft.com/en-us/visualstudio/ide/?view=vs-2017 >

- A working knowledge of the classes in **Open CASCADE Technology**. This comprehensive library is a software development platform that provides services for 3D surface/solid modeling, CAD data exchange, and visualization. Though Open CASCADE Technology provides a lot of functionality that is similar to what one would expect in a professional software suite like SolidWorks, there is, however, no Graphical User-Interface (GUI) in Open CASCADE Technology – it is simply a programming library. I have created a basic GUI using Qt in order to make certain features more accessible to non-programmers. More information on Open CASCADE Technology can be found online < https://www.opencascade.com/doc/occt-7.1.0/overview/html/index.html > and < https://www.opencascade.com/doc/occt-7.1.0/refman/html/index.html >

Due to the breadth and depth of knowledge needed to program *AMPL Toolpaths*, it may be slow to get going regarding the development of new extensions. The objective of these developer's notes is to describe how key classes function in *AMPL Toolpaths*, so as to lower the initial learning curve for future development. I will focus on the source files that I have coded as well as some of the key classes that are used throughout *AMPL Toolpaths*.

Solution Explorer of the High-Level Folder Layout

I will begin by describing the high-level categorization of the source code within the project, which can be seen in the solution explorer (see below). Note, the folder directory in the solution explorer is independent of the physical directory structure on the hard drive.



Tip

Files appended with the word "Dialog", such as hotKeysDialog.cpp, are primarily concerned with classes and functions that create dialog windows (which I have programmed). In general, these files create a new class through inheritance of an automatically created dialog class by Qt; all of the automatically generated files by Qt are located within the Generated Files folder of the solution explorer, and these automatically generated files are prepended with "ui" or "moc", such as "ui_hotKeysDialog.h" and "moc_hotKeysDialog.cpp". Within my "Dialog" files, specific functions are coded in order to perform certain actions as the user clicks on the various buttons (or types into the text-lines) within the dialog window. My classes are inherited from the generated files since coding directly within the generated files is volatile – these generated files get deleted and recreated upon compiling.

Tip

The front-end interface of *AMPL Toolpaths* is the parent dialog window, which is called QtOcct and "lives" within the namespace of QtOcct. Due to the size of this class, I decided to the spread out the definitions of the many members across multiple files. Any file with that contains "QtOcct" in its name belongs to this parent dialog window and corresponding namespace. The full list is as follows: QtOcct.h, QtOcct.cpp, QtOcctDemoParts.cpp, QtOcctISFPartLibrary.cpp, QtOcctImportExport.cpp, QtOcctTools.cpp, and QtOcctSingFeatureInput.cpp. All other dialog windows that get created (e.g., import/export dialog windows, the contact points dialog window, etc.) are children of the QtOcct dialog window. Therefore, if QtOcct gets closed while another window is open, then all of the children dialog windows are closed first (otherwise, the allocated memory would not be released correctly, and Windows would throw an error).

Descriptions of the Source Files

Each header file (".h" files) also has a corresponding source file (".cpp" files). Classes and members are declared in the header files, and then defined in the corresponding source files. For brevity, I will only provide descriptions of the source files. The following nomenclature, **[…]/**, denotes a folder name within the solution explorer. The source files themselves are italicized and boldfaced, and they are appended with ".cpp".

- **[AMPL_Toolpath]/**
  - **[AMPL_Algorithms]/**
    - ***amplWireIntersection.cpp*** Defines the class "amplWireIntersection". This class was designed to be a wrapper for the "wire" data structure in Open CASCADE. During an intersection of a plane with a surface, the result should be a wire. However, Open CASCADE returns an array of disconnected edges. This class attempts to sort through the edges to construct a continuous wire. Unfortunately, it does not seem to be very robust, since Open CASCADE has so many data structures for edges, and this class does not catch them all. So, the edges may not always be correctly sorted into a continuous wire (it needs more work). Currently, this class is used to discretize all of the intersected edges into a series of points, and then finds a continuous path through the points based on a seed point (i.e., the starting construction point defined in the advanced menu of the contact points dialog window). It turns out that working with a bunch of points (specifically of type gp_Pnt) is far more robust.
    - ***planeIntersecter.cpp*** Defines the class "planeIntersecter". This class is a helper class that is used to perform an intersection between a plane (i.e., gp_Pln or TopoDS_Shape) and a surface (i.e., TopoDS_Shape). This class is fundamental to all of the contact point algorithms. It provides many members to post-process the resultant intersection, which also utilize the amplWireIntersection class. In addition to performing the intersection and extracting contact points from the resultant wire, this class also extracts the surface properties at each contact point.
    - ***toolpathZHeightProps.cpp*** Defines the data structure "toolpathZHeightProps". The fundamental data structure *of AMPL Toolpaths* is "amplToolpath", and this class acts as a local helper class that parallels the amplToolpath data structure. After a surface has been intersected during the generation of contact points, there is no need to store the edge-data structures, and so, toolpathZHeightProps takes on the role of data storage for each contour. It contains containers for all of the surface properties, contact points, etc. Admittedly, it started getting a bit clumsy in how it was coded, but it also is used to generate and store the contact points for the supporting tool – this involves using the Sine Law to offset the points. When the generation of contact points is complete, this class then pushes all of the stored data into the global amplToolpath data structure for permanent storage; the toolpathZHeightProps class gets deleted after the toolpath algorithm is finished.
  - **[Help_Dialogs]/**
    - ***contactFormatHelpDialog.cpp*** Defines the class "contactFormatHelpDialog". This dialog window informs the user what each of the 17 columns mean when importing the contact points as a .csv file. It pops-up by clicking one of the "Help" buttons.

- ***firstTimeUserHelpDialog.cpp*** Defines the class "contactFormatHelpDialog". This dialog informs the user about the general limitations of *AMPL Toolpaths*. It is activated when the user clicks the "Help" button within the dialog window used to define one-stage, single-feature contact point generation.
- ***incSheetFormingInfoDialog.cpp*** Defines the class "incSheetFormingInfoDialog". While setting the parameters for contact point generation, the user can click on the "Information" button in panel 1). Upon doing so, this dialog window will pop-up and inform the user of the different configurations of incremental sheet forming that are supported.
- ***surfaceNormalHelpDialog.cpp*** Defines the class "surfaceNormalHelpDialog". While importing contact points from a .csv file, this dialog window can be shown by clicking one of the "Help" buttons. For each configuration of incremental sheet forming, this dialog window informs the user what the assumed directions are for the surface normal properties. If this methodology is not adhered to, then *AMPL Toolpaths* will generate tool-tip points in the wrong direction.
- ***toolpathParamHelpDialog.cpp*** Defines the class "toolpathParamHelpDialog". While setting the parameters for contact point generation, the user can click on the "Information" button in panel 3). Upon doing so, this dialog window will pop-up and inform the user of the definitions for each of the parameters specific to toolpath generation (e.g., incremental depth, start/end heights, etc.).
- ***toolSheetMetalParamHelpDialog.cpp*** Defines the class "toolSheetMetalParamHelp Dialog". While setting the parameters for contact point generation, the user can click on the "Information" button in panel 2). Upon doing so, this dialog window will pop-up and inform the user of the definitions for each of the parameters specific to the chosen machine configuration (e.g., tool diameter, sheet thickness, etc.).

- **[ISF]/**
  - ***ADSIF.cpp*** Defines the class "ADSIF". This is the high-level class that drives the generation of contact points and tool-tip points for all ADSIF toolpaths. It uses helper classes from [AMPL_Algorithms]/ to do this. In addition to driving the primary toolpath algorithms, it also handles refreshing the graphics after each Z-contour slice. More specifically, after initialization, this class exposes two primary members to the public: "calcContactPnts()" and "calcToolTipPnts(const bool, const int)".
  - ***DSIF.cpp*** Defines the class "DSIF". This is the high-level class that drives the generation of contact points and tool-tip points for all DSIF toolpaths. It uses helper classes from [AMPL_Algorithms]/ to do this. In addition to driving the primary toolpath algorithms, it also handles refreshing the graphics after each Z-contour slice. More specifically, after initialization, this class exposes two primary members to the public: "calcContactPnts()" and "calcToolTipPnts(const bool, const int)".
  - ***SPIF.cpp*** Defines the class "SPIF". This is the high-level class that drives the generation of contact points and tool-tip points for all SPIF toolpaths. It uses helper classes from [AMPL_Algorithms]/ to do this. In addition to driving the primary toolpath algorithms, it also handles refreshing the graphics after each Z-contour slice. More specifically, after initialization, this class exposes two primary members to the public: "calcContactPnts()" and "calcToolTipPnts(const bool, const int)".

- ***TPIF.cpp*** Defines the class "TPIF". This is the high-level class that drives the generation of contact points and tool-tip points for all TPIF toolpaths. It uses helper classes from [AMPL_Algorithms]/ to do this. In addition to driving the primary toolpath algorithms, it also handles refreshing the graphics after each Z-contour slice. More specifically, after initialization, this class exposes two primary members to the public: "calcContactPnts()" and "calcToolTipPnts(const bool, const int)".

- ***amplToolpath.cpp*** Defines the class "amplToolpath". This is the foundational data structure used throughout *AMPL Toolpaths* to fully define a toolpath. The primary goal of the dialog windows that are used to initialize the algorithms for contact/tool-tip points is to fill-in this amplToolpath data structure. It contains members for the contact points, (modified) tool-tip points, surface properties, forming direction, tool geometry properties, and so forth. Additionally, this class contains some helper functions to copy/reset its contents. Finally, the header file also contains the serialization code (from the Cereal library) that is used to determine which contents get saved into an ".ampl" binary file.
- ***QtOcctSingFeatureInput.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the QtOcct class defines functions to show all of the dialog windows within the high-level "Toolpath" menu. Specifically, the dialog windows corresponding to contact point generation, tool-tip generation, and converting a toolpath into machine code. Additionally, this portion of the QtOcct class extracts the information from these toolpath dialog windows, and from this information, initializes/calls the correct class for generating the toolpath – either the ADSIF, DSIF, SPIF, or TPIF class.
- ***singFeatureToolpathInputDialog.cpp*** Defines the "singleFeatureToolTipGenDialog" class. This class defines the single-feature contact points dialog window, as well as what happens when the user clicks the various buttons within the contact points dialog window. When the user is done and clicks "Start", then this class extracts all of the information from the user-interface (e.g., bullet lists, text boxes, etc.) and stores it into the amplToolpath class via a shared pointer.
- ***singleFeatureToolTipGenDialog.cpp*** Defines the "singleFeatureToolTipGenDialog" class. This class defines the single-feature tool-tip points dialog window, as well as what happens when the user clicks the various buttons within the tool-tip points dialog window. When the user is done and clicks "Start", then this class extracts all of the information from the user-interface (e.g., bullet lists, text boxes, etc.) and stores it into the amplToolpath class via a shared pointer.
- ***toolpathParamAdvancedDialog.cpp*** Defines the "toolpathParamAdvancedDialog". This class defines the advanced dialog window within panel 3) of the contact points dialog window, as well as what happens when the user clicks the various buttons within the tool-tip points dialog window. When the user is done and clicks "Accept", then this class saves all of the information from the user-interface (e.g., bullet lists, text boxes, etc.). The actual information, however, is not stored into the amplToolpath class until the user clicks "Start" within the "singFeature ToolpathInputDialog" class.
- ***toolpathVisibilityDialog.cpp*** Defines the "toolpathVisibilityDialog" class. This class defines the toolpath visibility dialog window, which can be accessed from the front-end, icon-based menu just above the OpenGL graphics renderer. The "toolpathVisibilityDialog" class gives the user the ability to alter the graphical representation of the toolpath (i.e., points, lines, or points and lines) as well as the chosen colors.

- **[AMPL_Tools]/**
  - ***amplLineRegion.cpp*** Defines the "amplLineRegion" class. This class is used within the "Post Tools" high-level menu, specifically for the surface-to-surface deviation function termed, "Two-Surface Deviation Map". To calculate the errors between two surfaces, the user can select either line or rectangular region. The "amplLineRegion" class defines the line region.
  - ***amplRectRegion.cpp*** Defines the "amplRectRegion" class. This class is used within the "Post Tools" high-level menu, specifically for the surface-to-surface deviation function termed, "Two-Surface Deviation Map". To calculate the errors between two surfaces, the user can select either line or rectangular region. The "amplRectRegion" class defines the rectangular region.
  - ***calc2SurfDeviationErrorDialog.cpp*** Defines the "calc2SurfDeviationErrorDialog" class. This class defines the dialog window used to define the parameters for the "Two-Surface Deviation Map" post-processing tool. It has members for importing the two surfaces (as ".igs" or ".step"), defining the region of interest, and where to save the deviation map on the hard drive.
  - ***exportCrossSectionDialog.cpp*** Defines the "exportCrossSectionDialog" class. This class defines the dialog window that is used to initialize the "Extract Cross Section" tool under the high-level menu item, "Post Tools".
  - ***lineRegionDefinitionDialog.cpp*** Defines the "lineRegionDefinitionDialog" class. This class defines the dialog window that is used to define the line region class (see amplLineRegion class).
  - ***QtOcctTools.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the QtOcct class defines functions to show all of the dialog windows within the high-level "Post Tools" menu. Specifically, this portion of the class shows the dialog windows for the "Extract Cross Section" and "Two-Surface Deviation Map" tools. Then it extracts this information and performs the desired operations. Note, that the "Two-Surface Deviation Map" is coded to handle the event that a desired query point does not lie on the imported surface, and hence, fails to intersect the surface. In this case, it is skipped, and no deviation error will be calculated.
  - ***rectRegionDefinitionDialog.cpp*** Defines the "rectRegionDefinitionDialog" class. This class defines the dialog window that is used to define the rectangular region class (see amplRectRegion class).
- **[ImportExport]/**
  - ***exportAmplToolpathDialog.cpp*** Defines the "exportAmplToolpathDialog" class. This class defines the dialog window that is used to collect the parameters needed to export the contact points and tool-tip points into a ".csv" file.
  - ***exportIGSDialog.cpp*** Defines the "exportIGSDialog" class. This class defines the dialog window that is used to collect the parameters needed to export the current CAD part into an ".igs" file.
  - ***exportToGen1MachineCodeDialog.cpp*** Defines the "exportToGen1MachineCodeDialog" class. This class defines the dialog window used to collect the information needed to export the tool-tip points into text files of machine code that can be read by NU's Gen. 1 DSIF Machine.
  - ***importCSVContactPntsDialog.cpp*** Defines the "importCSVContactPntsDialog" class. This defines the dialog window used to import contact points (saved as a ".csv" file) into *AMPL Toolpaths*.

- ***importIGSDialog.cpp*** Defines the "importIGSDialog" class. This class defines the dialog window used to collect the parameters needed to import a CAD surface that is saved as an ".igs" file.
- ***openAmplToolpathDialog.cpp*** Defines the "openAmplToolpathDialog" class. This class defines the dialog window to collect information for loading a ".ampl" file from the hard drive.
- ***QtOcctImportExport.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the "QtOcct" class contains all of the import and export features of *AMPL Toolpaths*, including the calls to the necessary dialog windows, as well as the actual file parsing (via a CSV file parser library). Due to the possibility of the user wanting to export some parts of the toolpath, and not other parts, the source code turned into a mess of if-statements and for-loops – my sincerest apologies.
- ***saveAmplToolpathDialog.cpp*** Defines the "saveAmplToolpathDialog" class. This class defines the dialog window to collect information for saving a ".ampl" file to the hard drive.

- **[ISF_Part_Library]/**
  - ***funnelDialog.cpp*** Defines the "funnelDialog" class. This class defines the dialog window that is used to collect the necessary parameters to construct a funnel CAD surface.
  - ***QtOcctDemoParts.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the QtOcct class defines the procedures used to generate the "Demo Shapes" under the high-level menu, "Part". These demonstration shapes are not actually used for the generation toolpaths. Actually, these features are artifacts of the original source code when I was still learning to use the Open CASCADE features; this was my sandbox area, and it just never got deleted.
  - ***QtOcctISFPartLibrary.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the QtOcct class defines the procedures used to generate the CAD surfaces in the "ISF Part Library" under the high-level menu, "Part". Specifically, the funnel, pyramid, and cone parts. Note, the declarations for makeTrunCone, makePyramid, and makeFunnel functions (under the namespace "part") are actually declared in the QtOcct.h file.
  - ***trunConeDialog.cpp*** Defines the "trunConeDialog" class. This class defines the dialog window that is used to collect the necessary parameters to construct a truncated cone CAD surface. Additionally, this class is reused when collecting the necessary parameters to construct a truncated pyramid CAD surface.
- **[QtOcct]/**
  - ***QtOcct.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. This portion of the QtOcct class was the original beginnings for all of the member definitions; recall that the QtOcct class is the parent dialog window for *AMPL Toolpaths*. It contains the constructor and destructor for the QtOcct class, initialization procedure for the OpenGL graphics widget, the signals/slots for displaying the high-level menu items, and all of the buttons used to modify the view. Additionally, this portion of the class contains the transformation definitions used in the "Move Part" features under the high-level menu, "Part". The "About" dialog window is also defined here.

- ***QtOcctInputEvents.cpp*** A partial definition of the class "QtOcct"; see the header file "QtOcct.h" for all of the declarations of the QtOcct class. The portion of the QtOcct class defines what happens in the renderer whenever specific mouse buttons or keyboard buttons are pressed.

- **[Transform]/**
  - ***clipPlaneDialog.cpp*** Defines the "clipPlaneDialog" class. This class defines the dialog window used to collect the necessary information to clip the model (i.e., a section view).
  - ***rotationDialog.cpp*** Defines the "rotationDialog" class. This class defines the dialog window used to collect the necessary information to rotate the model.
  - ***scaleDialog.cpp*** Defines the "scaleDialog" class. This class defines the dialog window used to collect the necessary information to scale the model.
  - ***translateDialog.cpp*** Defines the "translateDialog" class. This class defines the dialog window used to collect the necessary information to translate the model.
- ***hotkeysDialog.cpp*** Defines the "hotkeysDialog" class. This class displays a non-interactive dialog window (i.e., a picture) informing the user of the built-in hotkeys and mouse movements that can be used in *AMPL Toolpaths* to modify the view.
- ***main.cpp*** This is the programs main entry point. Every C/C++ code needs to start somewhere, and this is it. All it does, however, is call the QtOcct class, and when it is destroyed by exiting the dialog window, the program terminates.
- ***stdafx.cpp*** This is used to simply call the header file, stdafx.h. This is done so that the compiler can use the ".pch" file for precompiling the header files. By doing so, the project can compile much more quickly.