Technical Report: Implementing the Exporting Function in Triclops

Yixin Geng

4/30/2012

1. **Abstract**

This technical report is to show the transplantation work of “exporting” function from Digiclops to Triclops. Since the Point Grey Research Inc. will not support the Digiclops anymore, we need to implement some accordingly or dependently developed functions such as “exporting” in the new Triclops.

1. **Introduction**

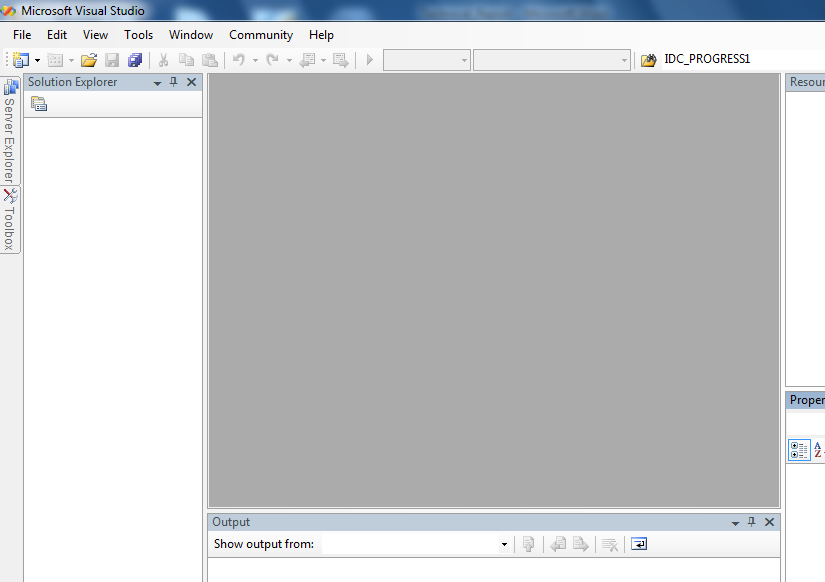
Digiclops and Triclops are the two different generations of camera manipulating, capturing and processing software used for Bumblebee 2 camera. The main objective for the whole project is using the Bumblebee 2 camera to capture some specific shapes, generate raw 3D files called point cloud, and process the files into milling format which can be used directly by the milling software to produce some shapes. The driver of the camera is separate from the Triclops called Flycapture. Hence different from Digiclops, the Triclops does not contain any driver files. The following figure is showing the overview of the whole project:

Since the Point Grey Research Inc. will not support the Digiclops anymore and the new Triclops does not contain the “exporting” function, we need to transplant it in Triclops.

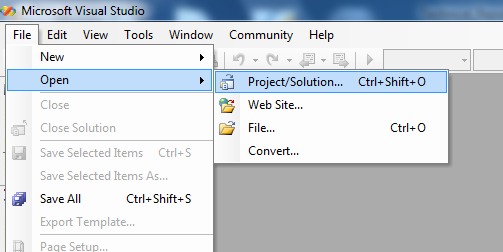
1. **Sample milling File Generation Instruction**

By today (4/30/2012), I have finished the transplant and release the updated version of Triclops source code into the SVN. Now we will use an example to demonstrate how to generate a sample milling format file. The IDE we use and recommend here is VS2005.

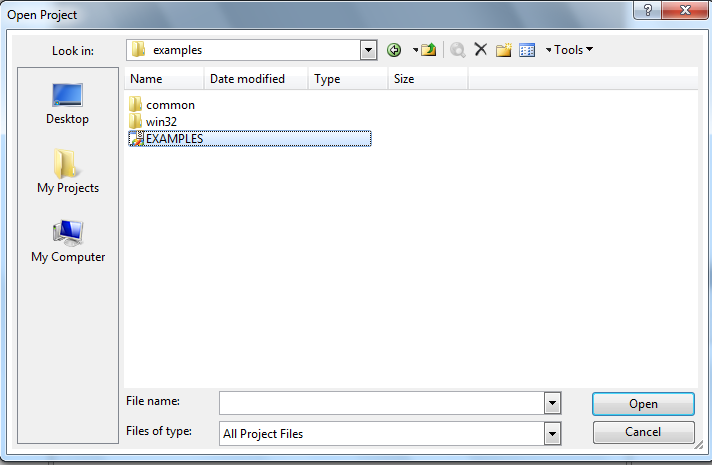
* + 1. Open the Visual Studio 2005



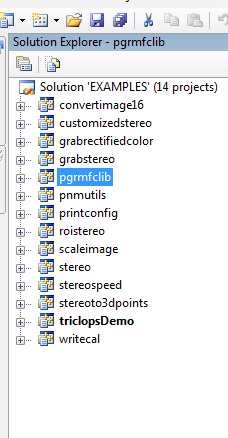
* + 1. File->open->project/solution



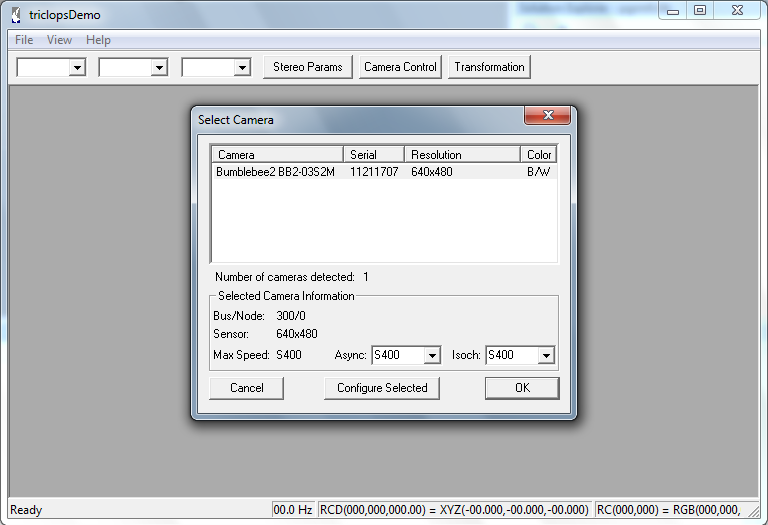
* + 1. Src->examples->Example.sln



* + 1. Now we can see all the files in our solution



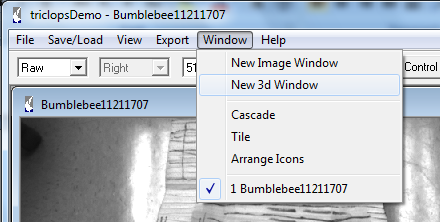
* + 1. Press F5 to start debug, after compiling finish, you will see the following dialog box. Make sure you have the camera connected to your computer. Then press OK.



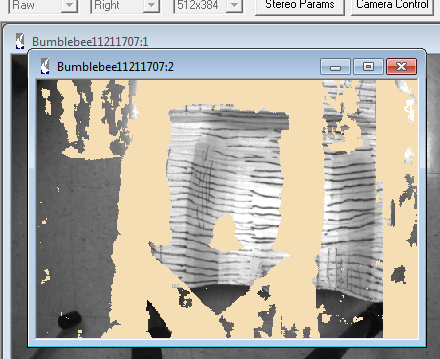
* + 1. Now you will see the main interface of the program. The highest resolution for the image here is 512X384. Please do not change the resolution since the exporting function is designed depending on it.



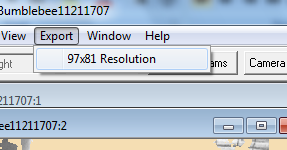
* + 1. Before generating the exporting function, we need to open the new 3d window.



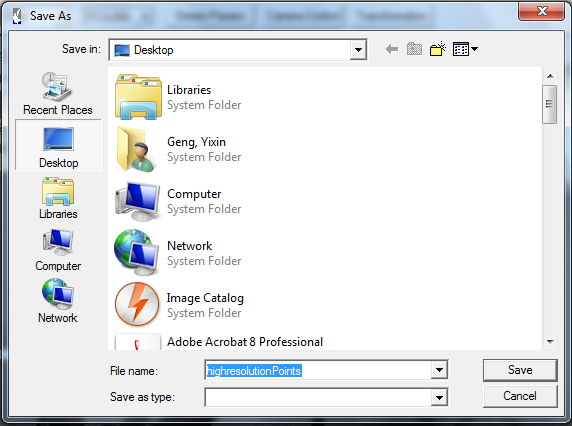
* + 1. Then you will see a new window with some shadowing image like this:



* + 1. Make sure the shadow is almost the shape like the model you used by adjusting the height of the camera at an appropriate position. Next step is to generate the exporting milling format file: Export-> 97X81 Resolution:



* + 1. Now you need to wait for almost 4 to 5 minutes. Please be patient since the waiting time is really long and the whole program seems get stuck or crashed. After that, you will see a dialog box ask you to save the file. The default name is highresolutionPoints.dat. Find your saving folder and press save.

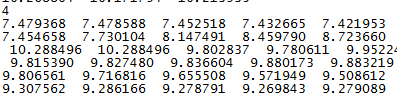


* + 1. Open your saving folder you will see the exporting files directly. The highresolutionPoints.dat is just what we want. However in the file the data is in the format of

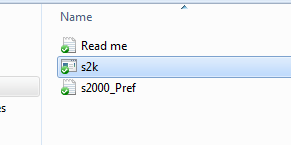
x

y y y y y y y y y

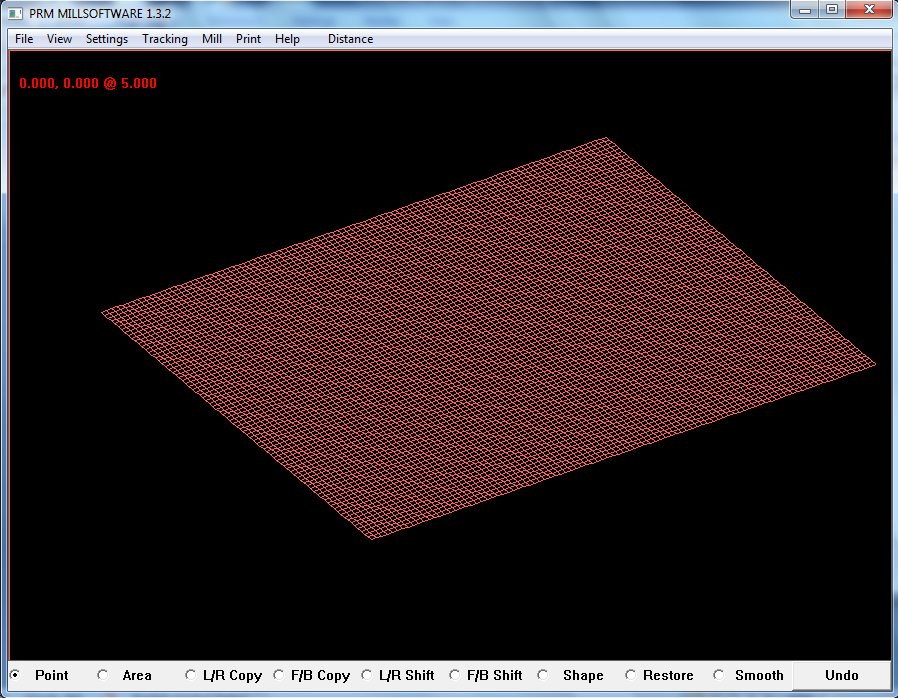
The x means the line number and the y means the point’s height at which column. For instance, in the part of following sample file, it means at 4th row, 1st column, the height is 7.479368, 2nd column is 7.478588 etc.



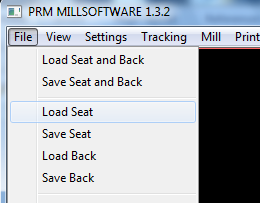
* + 1. To see a visual plotting image of the data, we need to use the milling application called s2k. The latest version is v1.3.2.1.



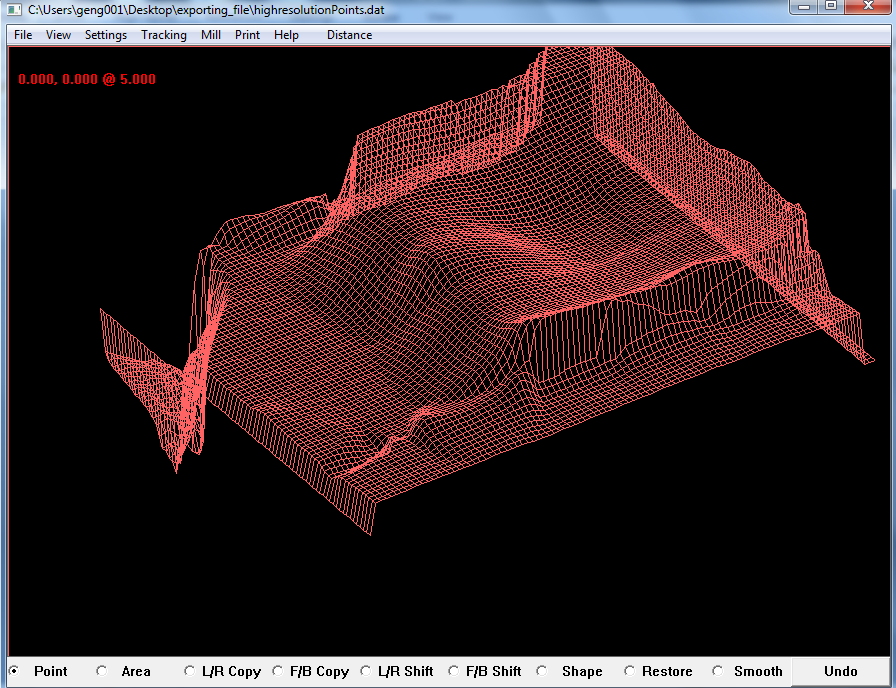
* + 1. Open it you will see the following, this is a 97X81 format grid:



* + 1. Then file->load seat



* + 1. Open your .dat file and finally you will see the shape of the exporting file.



1. **Principle of the exporting function**

The main idea of the function is to convert the point cloud file into the exporting format. The pointcloud file is a delicate type of 3D image including around 16000 points’ XYZ coordinate and their color representing in RED, GREEN, BLUE format. However, the exporting format does not need such high resolution. Hence, the principle is to project the 97X81 grids on the points of pointcloud file and assign the points on the grids a height.

First step is overlapping their center points to make sure the shape will appear in the center of the grids.

Next step is to assign the values for each node in the grids. Here we apply an algorithm which is setting a range for each node and take average height of all points in this range as the height of the node. The advantage for this algorithm is smoothing the shape of the image preventing some sharp edges or stings caused by errors. The value of range is a variable called smsurface in the source code. In the pointcloud file the unit is meter and in the grid the unit is inch. The actual size for one single grid is 1/4inch X 1/4inch. Hence here needs to convert the unit.

Another important setting is the ROI (Region of Interest). This is used for cutting the noise at or close to the edges of pointcloud file since the closer to the edges, the errors will increase and influence the whole shape. To prevent such situation happens, we set a region centered in the pointcloud image with a size of 400X300. Hence we only calculate the points in this range or they will not be counted.

1. **Briefly introduction for the added functions and classes in the source code**
2. The main function to implement the exporting is the following:

Void CPGRStereoDoc::OnFileExportHighResolution(float baseline, float smsurface)

The function is directly triggered by the “Export 97X81” icon in the menu. It implements the algorithm of calculating the grid points and finally exports the 97X81 format files. The value of baseline and smsurface is assigned in the class CPGRStereoControlDialog::CPGRStereoControlDialog( CWnd\* pParent /\*=NULL\*/) : CDialog

(CPGRStereoControlDialog::IDD, pParent)

1. The newly added files are grid.h and grid.cpp. These two files contain all the functions and classes for the grids projecting on the pointcloud file including the grids’ size and unit converting part.
2. C3dColourPointRC\* CPointList::getNearestPosition(C3dColourPointRC \*p, CPointList \*vecMap)

The function is added in the Pointlist.cpp file which is used to search the whole point list in the point cloud file and judge which points are in the defined range. If yes the height will be summed and take average to assign the grid. If not it will continue to search the next point.

1. double CPointList::getNearestPixelAverageZ(C3dColourPointRC \*p, CPointList \*vecMap, float smsurface)

The function is added in the Pointlist.cpp file which is used to calculate the average value of height.

1. CPointList \*CPointList::getResolutionToPointList

(

CPointList \*grid,

CPointList \*realWorldPoint,

double startx,

double starty,

double endx,

double endy,

float smsurface

)

The function is added in the Pointlist.cpp file which is used to place the grids on the pointcloud file.

1. bool CPointList::saveToMillFormat(const char \*pszFileName, int gridXnum, int gridYnum)

The function is added in the Pointlist.cpp file which is used to save the grid points into the exporting format.

1. **Prospective**

There are two parts which can be improved in this project. The first one is to build a progress bar for the exporting since the waiting time is too long. Progress bar will help people to know the progress. Another part is the camera can only capture the image with a certain angle or the image will be twisted. We can develop a function to calibrate and rectify the image that no matter the image is captured in what angle; the shape will still be good.

1. **Troubleshooting**

During the transplant process, I found a serious problem. When I try to compile the Digiclops’ original code, I found the Triclops cannot be compiled successfully. The error is a runtime error says, there is an unknown problem with the dynamic link to the library. I tried to debug in the Triclops’ original code but it did not work. Hence I reinstalled the software Triclops downloaded from the company’s website. Then it could be compiled without any errors. The problem I suspect is the dynamic link problem related to a library which is required by both Digiclops and Triclops. When the Digiclops code is compiled, it may automatically change the linking address of the library which Triclops cannot find. Hence it causes the error.