Determining the Volume of a Comet Particle Track via an Automatic Best Fit Circle Method

Victoria A. Tielebein*, Amanda J. White, and Denton S. Ebel

Department of Earth and Planetary Sciences, American Museum of Natural History, Central Park W. at 79th St., New York, NY 10024, USA

*Corresponding author. E-mail: victoria.a.tielebein@gmail.com

INTRODUCTION

Comets are one of the most beautiful spectacles to light up the night sky. For millennia, people have marveled at the wonder and majesty of these small Solar System bodies, but only in the past couple centuries have scientists begun to wonder by what mechanism this beauty comes to be. From the early study of comets until now, huge technological advancements have revolutionized our ability to analyze these majesties. Most recently, the NASA *Stardust* spacecraft returned to Earth with a collection of carefully obtained comet dust particles from the comet Wild 2. Wild 2 spent most of its 4.5 billion year lifetime in a distant, circular orbit, with an orbital period of roughly 43 years. However, in 1974, this comet passed a little too near Jupiter, whose strong gravitational pull perturbed Wild 2's orbit, bringing it into the inner Solar System and reducing its orbital period to roughly 6 years ("Comet"). Since comets form in the outer reaches of our Solar System (the Kuiper Belt), Wild 2's new found closeness has opened a window into studying the early Solar System like no other comet has before.

Stardust returned more than 10,000 particles in the 1 to 300 μm sized range that are believed to represent the nonvolatile component of the interior of the comet. This is due to the fact that Wild 2 only recently became close enough for solar heat to cause the sublimation of water ice that lead to vast losses in gas, rocks, and dust. As a result, all of the particles ejected from the comet in this manner date back to the formational period of our Solar System's history ("Comet 81P/Wild 2 Under a Microscope").

Aerogel is a synthetic, porous material manufactured by removing the liquid component of the gel and replacing it with a gas. This "frozen smoke" is a solid with an extremely low density and thermal conductivity, which structurally collapses under a modest level of pressure ("Aerogel"). This makes it an excellent material for use in capturing comet particles as it allows the particles to travel deep within the container, and leave a perfectly preserved track behind them, while not changing the overall shapes of the particles as much as some other media would. Knowing the shapes of the particles is important for identifying which elements can be found in comets, and knowing the shapes of the tracks is important so as to understand exactly how the hypervelocity capture event occurred ("Comet 81P/Wild 2 Under a Microscope").

Stardust tracks have been categorized into three different types (A, B, C) resembling carrots, turnips, and bulbs respectively. Through further study of these tracks, namely recreating them in a lab environment, it has become clear that track formation is a highly complex process, involving thermal and physical alterations to track particles of all shapes and sizes. In order to help better characterize these tracks, the extent of the thermal alteration, as well as the chemical composition and original physical properties of the particles needs to be determined (Greenburg).

It has been experimentally shown that the shape and volume of impact tracks in stardust aerogel reflect cometary dust

properties (Kearsley et all). Therefore, having the most accurate measurements possible is crucial to achieving a better understanding of comets. There are various microscopy methods which make possible the characterization of whole particle tracks in three dimensions using solely non-destructive methods. However, of these methods, the most accurate to date combines laser scanning confocal microscopy (LCSM) with synchrotron X-ray fluorescence (SXRF) (Greenburg).

The imaging process takes several hours, and due to the length of the tracks, must be manually stitched together before the picture is complete. Next, the track is re-sliced into cross-sections. In order to determine the volume previously, one would have to sit down with a copy of the stitched together track and draw frustums, cone shapes with their tips cut off, on the track such that the least amount of extra space is included in the track measurement. The location of the top and bottom of the frustum would then be measured out with a yard stick, and the analyst would manually trace around the particles on the image slices in question in an image-processing software. The area of each the top and bottom of the frustum would then be used to calculate the frustum's volume. The sum of these volumes yields an approximate volume for the entire track.

This process of determining the volume of a comet particle track is as tedious and time consuming as it sounds, appealing to the need for automation. In addition to saving a lot of time and energy determining track volumes, the volumes could hypothetically be determined more accurately by a computer, as every slice can be analyzed in place of frustum approximation. For this reason, I was solicited to create software to automatically determine track volume.

The first step in automatically analyzing these tracks is to extract data points which represent the locations and sizes of the particles on each image slice. These data points can then be run through a best fit circle program to determine the approximate area of the particles on that slice. Adding up the area calculation for every slice in the track yields the total volume of the track.

Extracting Data Points Using ImageJ

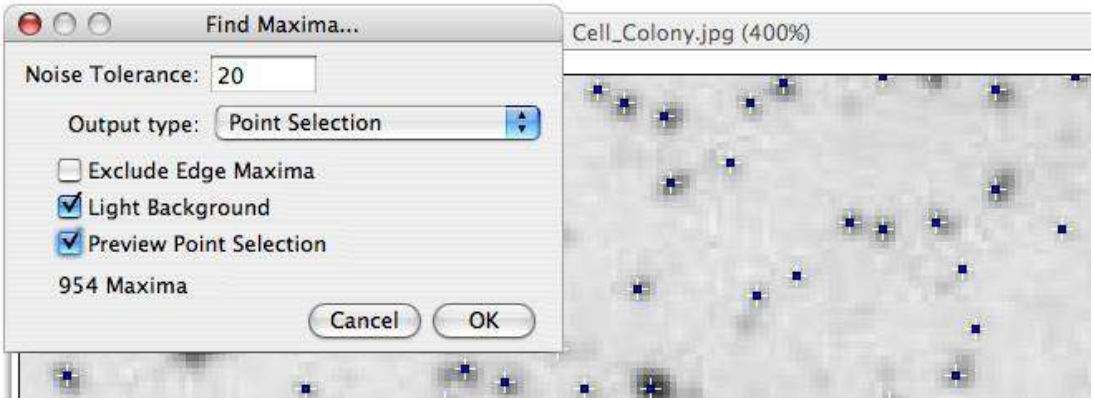
I chose to use the imaging software ImageJ (or Fiji, which is an extension of ImageJ, and in this case no different than ImageJ itself) to process the images for a few reasons. Namely, ImageJ has a lot of in built features which make writing plugins simple. Beyond that, ImageJ has built-in functions for smoothing the images and adjusting the images' brightness and contrast. Integrating these functions into a plugin is beyond simple, leaving only the most crucial step to figure out.

What is the best way to extract data points which accurately depict the area of the particles on the inputted image? There are several options, but the one which proved most reliable was the "Find Maxima" function. This function determines which pixels have a maximum or minimum brightness (depending on user settings) and places data points at those coordinates (see Image 1 for an illustration of this)(Ferreira and Rasband). It is a simple matter to save these data points to a file, and since the brightest spots in the image are hypothetically the particles, this is a great mechanism for converting images into data sets.

Another great feature of the "Find Maxima" function is the ability to toggle the noise tolerance. In one instance, a step to make the image binary was added before finding the maximum pixels, effectively making any pixel with coloration a data point. This step was then removed, and different noise levels were tried. Comparing the results from each of these methods shows that a noise tolerance level of 500 yields the most accurate results (see Chart's 1 and 2).

See the attached code and documentation for specific details on how this plugin works and the steps for running the plugin on a given track's set of image slices.

Image 1- Illustration Find Maxima Function taken from ImageJ Documentation (Ferreira and Rasband)



The Best Fit Circle Method

The best fit circle process works by first determining the approximate center of the data points. To do this, it "draws" a line between every possible set of two points contained in the file and determines the center of this "line". This center cumulates and is then divided by the total number of lines drawn. This yields the approximate center of the data points. Next, the distance between every point in the file and this approximate center is calculated, and this average is the approximate radius. This radius is then used to calculate the area of the best fit circle for this set of data points. Summing all of these areas together for every slice in the track yields the approximate volume.

The best fit executable has two methods of analysis which the user can call upon: processing a solo image's data set, or multiple images' data sets. There is one particular file format that the program will accept because of the way in which ImageJ outputs its data sets. The program will only be able to run successfully on files which contain two elements in the first line (which will be discarded in any case), and three on every subsequent line. All of the elements in the file must be separated by the same type of place-holder (spaces, tabs, newlines, etc). If the user wishes to run the multiple file option, then the files must each be titled "data{i}.txt", where the {i} is filled in with the index of that particular image in the full track. Ideally, the images should be indexed the same as whichever slice they correspond to in the comet particle track, as will be the case if the user runs the ImageJ plugin, and then the best fit program for the entire track. For more specific details on how to run this program, see the attached code and documentation.

Error

Analyzing the error of the automated process is tricky for a number of reasons. First and foremost, all of the automated results are compared to manually traced results which depend entirely on the skills of the analyst. However, in order to minimize human error in this regard, the particular track discussed in this section (track 163) was traced by two different people a number of times to yield the closest results possible. The total volume found in this manner is much smaller than the results published by Michael Greenburg earlier this year by around a factor of ten, but this is not unexpected as the microscopy and tracing methods have been developed for further accuracy in the meantime (Greenburg).

Considering all of this, it isn't so unreasonable to have large percent errors since the standard deviation between the new and the old manually traced results is rather large. This means that despite glaringly large percent errors, the program may still be doing its job correctly. In order to look more closely at what was occurring, I compared the different

methods of analysis with each other in a couple of different ways.

First, I took a look at the percent error stretched out over the entire track. Chart 1 shows the percent error at each slice used for the manual error calculation. It is fairly obvious that the most accurate of the methods is that which uses a non-binary image with a noise tolerance level of 500, meaning that a large number of pixels were eliminated as data points for not being close enough to the maximum in coloration. However, despite being convinced that this method of analysis was the most accurate, I still was unsure whether or not the results were accurate enough to make this software's calculations useful.

Then I thought that perhaps the images simply had too much noise to yield appropriate results. If this were the case, it would mean that there were a large number of data points lying outside of the particle's area circle that were being factored in to the total area calculation. For this reason, I assumed that the linear representations of these tracks should have similar shapes, with the automatically traced lines being generally higher. Looking at Chart 2, one can, in fact, see the similarities between the manually traced results and the noise tolerance level 500 results. Some parts are a bit odd, but for the most part, where there is a dip in the manual trace, there is also a dip in the automatic, etc. However, the changes in area in the automatic track are much more extreme. This can be explained if one considers how the brightness/contrast is adjusted for each individual slice, as opposed to once for the entire track. This makes a difference because the brightness/contrast function works by averaging over the image's histogram, dimming the brightest pixels and brightening up some of the darker ones. If there is less actual material on the track, then more of the noise pixels will be considered important during this step, and will therefore be considered data points in subsequent steps. At this point, I pretty much stopped considering the other methods, as they simply did not seem to line up at all, and had higher percent errors, making them twice as likely to be incorrect.

Chart 1

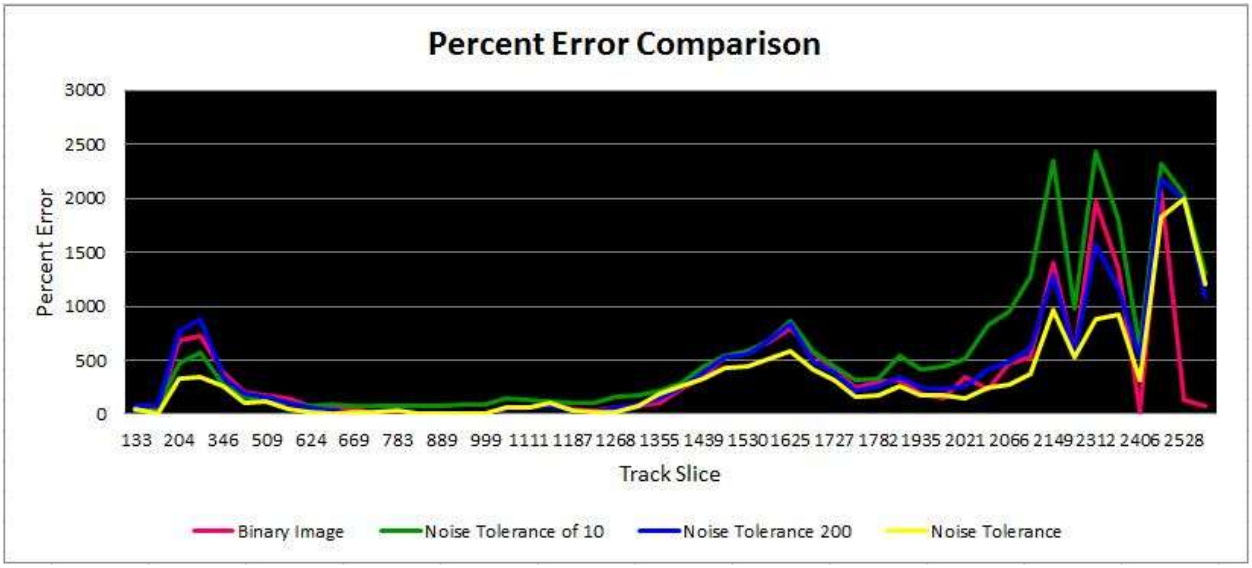
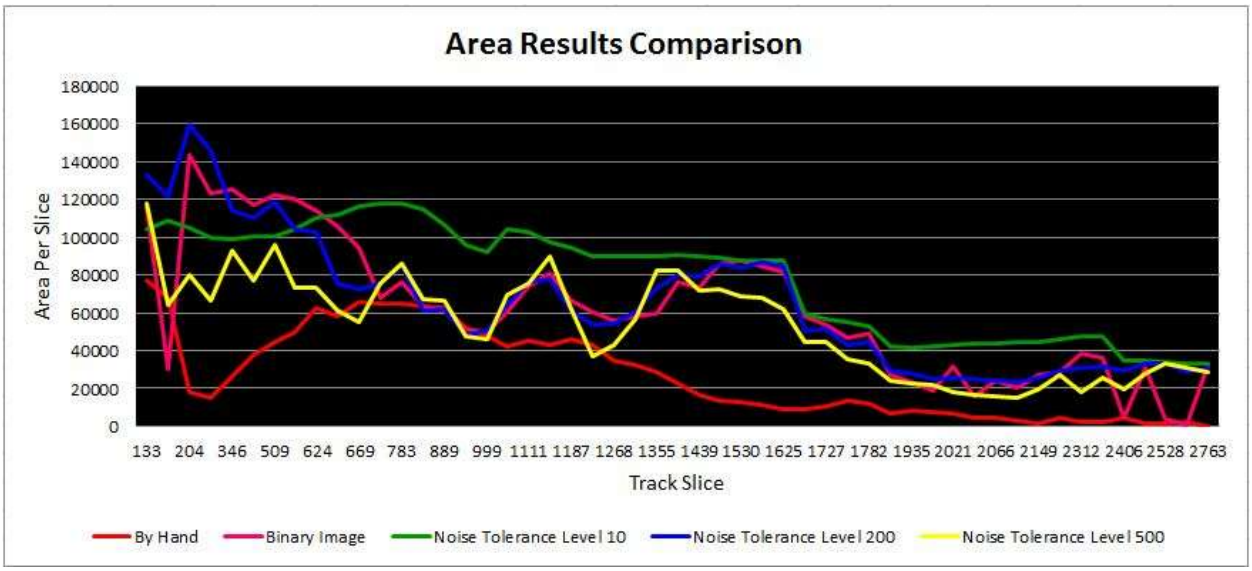


Chart 2



I did consider that the error could be the result of my best fit circle algorithm, and so in order to check that the best fit process worked sufficiently well, I drew several small circles on graph paper and assigned integer valued data points that roughly fit the overall shape to each. The results of these calculations were almost exact, as shown in Charts 3 and 4.

Chart 3

Radius	cx	cy	Area	
Actual Results				
4.0	7.0	5.0	50.265	
3.0	4.0	-4.0	28.274	
3.0	-1.0	2.0	28.274	
Program Calculated Results				% Error
4.100	7.000	5.032	52.821	5.1
2.902	3.850	-4.050	26.468	6.4
2.947	-0.944	2.056	27.283	3.5

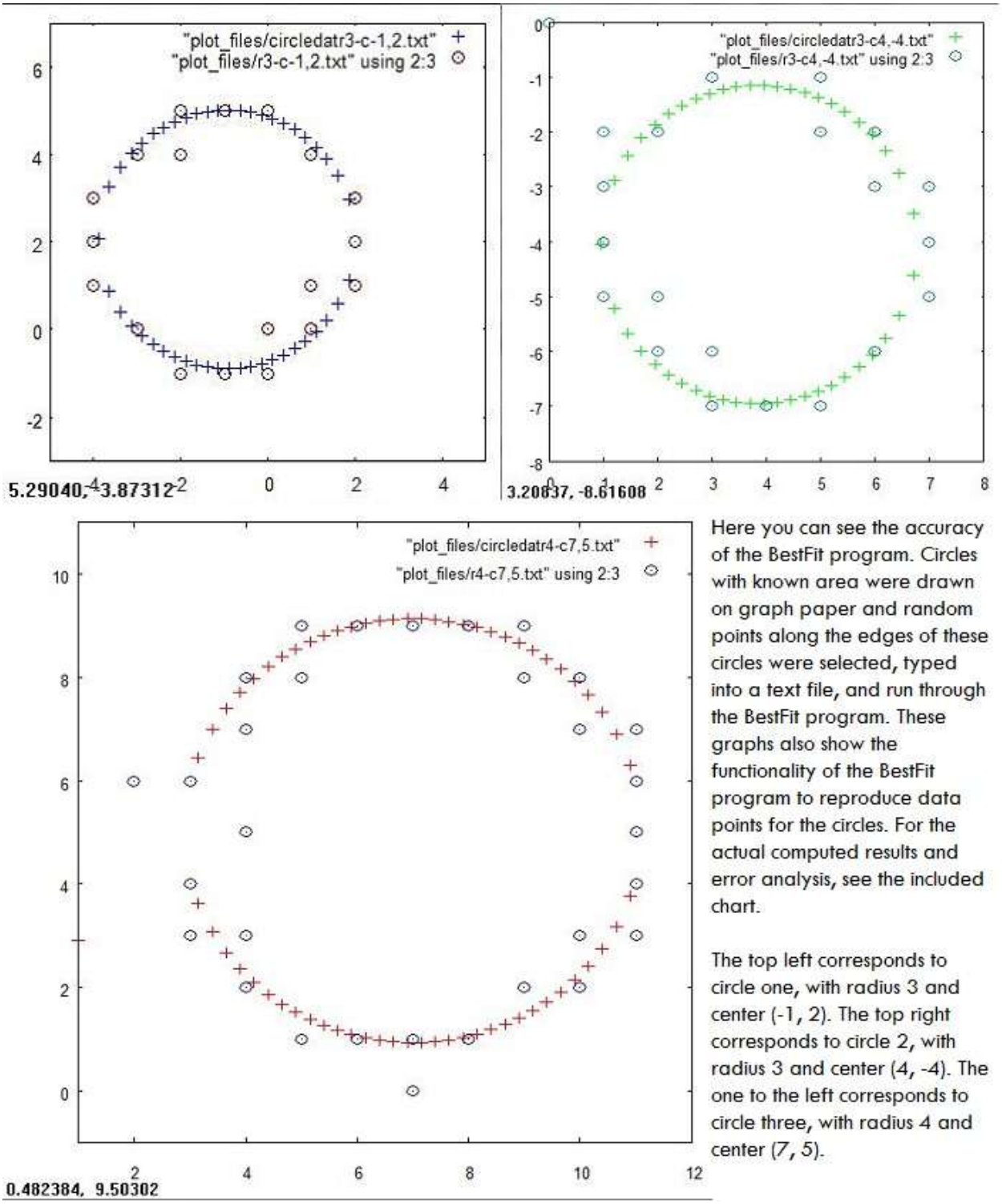
Chart 3 displays the calculated results as determined by the best fit program, and shows the percent error. The actual area was determined through plugging the radius into the $A = \pi r^2$ formula.

The actual volume as determined by the automatic noise tolerance level 500 method was approximately 1.75×10^8, while the result from the manual tracing was 2.97×10^7. The result from the automatic tracing was very similar to Michael Greenburg's result of 2.20×10^8 from earlier this year. However, this is not necessarily a good thing, as the obtainment of a better microscopy instrument should yield newer results which are more accurate. It is therefore safe to assume that the calculated results should be closer to the new measurement of 2.97×10^7, and so are not accurate to within a little under a factor of ten. This means that the percent error of the manually measured versus automatic volume is roughly 488%.

To show where this problem may come from, see Images 2 and 3 included below. Image 2 depicts a processed image with the data points overlaid on top of it next to those same data points plotted against the automatically calculated best fit circle. One can see that due to the number and placement of data points

which result from noise, the best fit circle which is calculated is misplaced. While the center of the circle is not too far from accurate, the average radius is incredibly high, and so the area which results is much too high. This is shown in further detail back in chart 2.

Chart 4



Here you can see the accuracy of the BestFit program. Circles with known area were drawn on graph paper and random points along the edges of these circles were selected, typed into a text file, and run through the BestFit program. These graphs also show the functionality of the BestFit program to reproduce data points for the circles. For the actual computed results and error analysis, see the included chart.

The top left corresponds to circle one, with radius 3 and center (-1, 2). The top right corresponds to circle 2, with radius 3 and center (4, -4). The one to the left corresponds to circle three, with radius 4 and center (7, 5).

Image 2- Processed image with data points overlaid (right) and the same data points plotted with the best fit circle output (left) (data taken from image slice 133 from track 163)

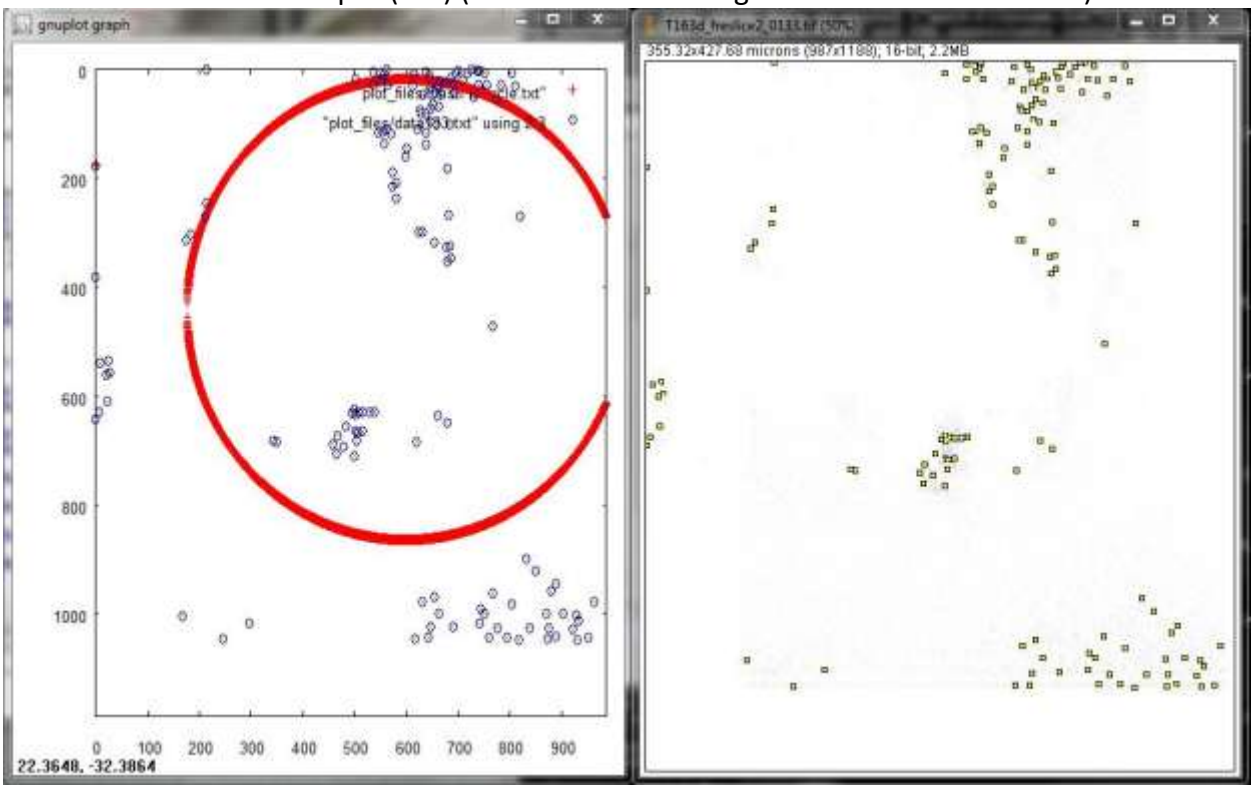
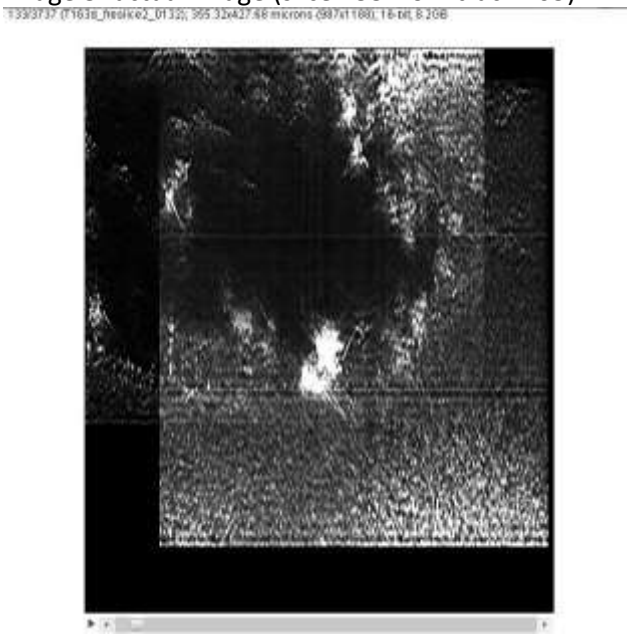


Image 3- actual Image (slice 133 from track 163)



CONCLUSIONS

Upon review of the program's total error, it becomes apparent that the software is not quite ready for use. However, it is a good start, and there are a variety of ways in which this process can be made more accurate.

One such way would be to start the process with the known center of the first track, and a radius beyond which any data point should be considered noise. This radius can be scaled in a way proportional to track length such that as the process gets nearer and nearer to the end of the track, the circle in which data points are accepted gets smaller. Because of the possibility that the track will curve, the center will have to be recalculated after each slice. My recommendation for implementing this idea would be to add a subroutine to the BestFit.cpp

code which removes invalid data points from the data set after everything has been imported, and then use the same center calculation done by the code to determine the approximate center (the same as is done currently, but with potentially less noise being considered). Then, this center can be averaged with the center from the previous slice before the program determines the average radius. This means that the user would have to analyze one slice manually, edit a few of the measurement parameters, and run most of the same code that is already written. This would reduce the possibility that noise is affecting the results a great deal, lowering the overall volume determined as well as making the results more consistent and with less radical fluctuations in area from slice to slice.

Another idea to improve results is to implement some type of weighted average during the approximate center and radius calculations. The idea is to take particles that are within a certain distance of each other and consider them to be as one particle. The user could add a few lines of code to the BestFit.cpp code to implement this. Add a parameter which represents the maximum distance between two points under which they are considered the same point. Then, an average of their distances could be taken, and they could effectively be combined. This could occur for a variety of particles in the program, and since most of the clusters belong to noise specks, the noise would be considered less in the final calculation of the average radius. This will re-establish a balance between the sections of the image which have a lot of data points representing noise and the sections with very few points representing particles (see Images 2 and 3 for an illustration of this effect).

Best Fit Circle Program Code:

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cstdlib>
#include <string>
#include <sstream>
#include <fstream>
#include <cmath>

using namespace std;

/********** Functions used in this program include: *************/

/* Print functions for a 1D or a 2D vector */
void print_1d(vector<double> V);
void print_2d(vector<vector<double> > V);

/* A function to find the radius of the best fit circle for a given set of data points */
vector<double> findRadius(vector<vector<double> > & V, double & radius);

/* And a function to calculate and output to file the points lying along the edge of a circle with radius r.
*/
string outputCirclePoints(vector<double> c, double r);

/* There are two built-in ways to run this code: for a solo image or an image stack */
int ProcessSoloImage(int slice);
int ProcessImageStack(int firstSlice, int lastSlice);

/*****************************************************************/

/* Global Variables */
double pxtomic = 0.458; //conversion factor from pixels to microns
string multiAreaOut = "Areas.txt"; //output file for the individual areas per slice (if
ProcessImageStack(int, int) is called)

/***************************** Main *****************************/
int main(int argc, char *argv[])
{
	cout << "Best Fit Circle Program" << endl;
	cout << "Make sure that you have run the ImageJ/Fiji plugin \"FindMaximaPerSlice.txt\" prior to
running this software." << endl;
	cout << "1. Process Solo Image" << endl;
	cout << "2. Process Image Stack" << endl;
	cout << "How would you like to use this software? ";
	int selection, ft, lt;
	cin >> selection;
```

```
	if(selection == 1)
	{
		char yorn;
		cout << "Analyze a data file outputted from \"FindMaximaPerSlice.txt\" plugin? (y/n) " ;
		cin >> yorn;

		if(yorn == 'y')
		{
			cout << "Please input the index of slice/data file. " ;
			cin >> ft;
			ProcessSoloImage(ft);
		}
		else
			ProcessSoloImage(-1);
	}
	else if(selection == 2)
	{
		cout << "Please input the range of the track you wish to analyze separated by a space.
Ex: 0 3736" << endl;
		cin >> ft >> lt;
		if(lt > ft)
		{
			cout << "Calculating. This may take a few moments depending on track length."
<< endl;
			ProcessImageStack(ft, lt);
		}
		else
			cerr << "Inputted range [" << ft << ", " << lt << "] is invalid." << endl;
	}
	else
		cerr << "Invalid Selection." << endl;

	return 0;
}


/* Subroutine to determine the volume of all slices in a given length of track */
int ProcessImageStack(int firstSlice, int lastSlice)
{
	string inputfilename;
	double volume = 0.0;

	ofstream fout( multiAreaOut.c_str() );
	fout << "Slice #" << setw(9) << "Area" << setw(13) << "Radius" << setw(16) << "Center" << endl;

	for(int z = firstSlice; z <= lastSlice; z++)
	{
```

```
		// First recalculate the name of the input file //
		stringstream ss;
		ss << z;
		inputfilename = "data_files/data" + ss.str() + ".txt";
		/* Code assumes that the location of the data files is ./data_files and the name of the */
		/* files is data{z}.txt as that is what the output of the ImageJ/FIJI plugin should be. */
		ifstream fin( inputfilename.c_str() );

		// Define initial variables //
		vector<vector<double> > V;
		vector<double> S;
		double temp, col1, col2;
		char fl1, fl2;

		if( fin.is_open() )
		{
			int i = 0;

			fin >> fl1 >> fl2; //first line is outputted from ImageJ/FIJI plugin with useless
data, for which this line placeholds

			while( fin >> temp >> col1 >> col2 )
			{  //the three columns from ImageJ/FIJI plugin output: an arbitrary number, the
x coordinate in pixels, the y coordinate in pixels.
				V.push_back(S);
				V[i].push_back(col1*pxtomic); //multiply each of these by the
conversion factor to get microns
				V[i].push_back(col2*pxtomic);

				i++;
			}
		}
		else
		{
			cerr << "Cannot open file \"" << inputfilename << "\"." << endl;
			return 0;
		}

		if(V.size() > 1)
		{
			double radius = 0;
			vector<double> c = findRadius(V, radius);
			double area = M_PI * radius * radius;
			volume += area;

			/* Uncomment for details about specific slices. */
			/* ADVISED: ONLY UNCOMMENT IF RUNNING PROGRAM FOR SMALL NUMBER
OF SLICES */
```

```
                        /*cout << "Circle " << z << ":" << endl;
                        cout << "The average radius length is " << radius << " and the approxiamte
center is (" << c[0] << ", " << c[1] << ")." << endl;
                        cout << "The approxiamate area of the best fit circle is " << area << "." << endl;
                        cout << "The data points lying along this circle can be found in " <<
outputCirclePoints(c, radius) << "." << endl;
                        cout << endl;*/

                        fout << setw(4) << z << setw(14) << area << setw(12) << radius << setw(7) << "("
<< c[0] << ", " << c[1] << ")" << endl;
                }
                else
                        fout << setw(4) << z << setw(12) << "nan" << setw(12) << "nan" << setw(18) <<
"nan" << endl;
        }

        //outputs results to console
        cout << "The volume for these " << (lastSlice - firstSlice) + 1 << " slices is approxiamately " <<
volume << "." << endl;
        cout << "The areas for each individual slice can be found in the file \"" << multiAreaOut << "\"."
<< endl;

        //outputs results to file
        fout << "The volume for these " << (lastSlice - firstSlice) + 1 << " slices is approxiamately " <<
volume << "." << endl;
        fout << "The areas for each individual slice can be found in the file \"" << multiAreaOut << "\"."
<< endl;

        return 0;
}


/* Print Functions for 1D and 2D arrays */
void print_1d(vector<double> V)
{
        for(int i = 0; i < V.size(); i++)
        {
                cout << V[i] << " " ;
        }
        cout << endl;
}

void print_2d(vector<vector<double> > V)
{
        for(int i = 0; i < V.size(); i++)
        {
                for(int j = 0; j < V[i].size(); j++)
                {
```

```
                        cout << V[i][j] << " ";
                }
                cout << endl;
        }
}


/* This function determines the average center of the circle, and then calculates the average radius from this center to each point */
vector<double> findRadius(vector<vector<double> > & V, double & radius)
{
        vector<double> center; center.push_back(0.0); center.push_back(0.0); //create void center counter (calculated values will update this below)
        int totalLinesDrawn = 0;

        for(int i = 0; i < V.size(); i++)
        {
                for(int j = i+1; j < V.size(); j++)
                {
                        double x1 = V[i][0], x2 = V[j][0];
                        double y1 = V[i][1], y2 = V[j][1];

                        double cx, cy;

                        cx = (x1 + x2) / 2; cy = (y1 + y2) / 2; //center of the line between these two points

                        center[0] += cx; center[1] += cy; //update the counter for the average center calculation
                        totalLinesDrawn++;
                }
        }

        // finds the average center
        center[0] /= totalLinesDrawn; center[1] /= totalLinesDrawn;

        int counter = 0;

        for(int i = 0; i < V.size(); i++)
        {  // takes the center from above and determines the average radius to each point
                double x1 = V[i][0], y1 = V[i][1];
                radius += sqrt( pow( (x1 - center[0]), 2 ) + pow( (y1 - center[1]), 2) );
                counter++;
        }

        radius /= counter;

        return center;
```

```
}


/* This function can output the data points for the best fit circle of a particular image to a file for
plotting purposes, etc. */
string outputCirclePoints(vector<double> c, double r)
{
	string outputfile = "BestFitCircle.txt";
	ofstream fout( outputfile.c_str() );

	double ypos, yneg;

	c[0] /= pxtomic; c[1] /= pxtomic; r /= pxtomic; // converts back to pixels for plotting

	for(double x = (c[0] - r); x <= (c[0] + r); x += 0.25)
	{	// (x-cx)**2 + (y-cy)**2 = r**2
		ypos = c[1] + sqrt( pow( r, 2 ) - pow( (x - c[0]), 2 ) );
		yneg = c[1] - sqrt( pow( r, 2 ) - pow( (x - c[0]), 2 ) );

		fout << x << " " << ypos << endl;
		fout << x << " " << yneg << endl;
	}

	return outputfile;
}


/* In this routine, only one data file (user specified via file or command line) is analyzed, and the results
are outputted to the console immediately. */
int ProcessSoloImage(int slice)
{
	// First recalculate the name of the input file //
	string inputfilename;
	if(slice < 0)
	{
		cout << "Please specify file where data is stored (\"path\\filename.txt\"): ";
		cin >> inputfilename;
	}
	else
	{
		stringstream ss;
		ss << slice;
		inputfilename = "data_files/data" + ss.str() + ".txt";
		/* Code assumes that the location of the data files is ./data_files and the name of the */
		/* files is data{slice}.txt as that is what the output of the ImageJ/FIJI plugin should be. */
	}
```

```
	ifstream fin( inputfilename.c_str() );

	vector<vector<double> > V;
	vector<double> S;
	double temp, col1, col2;
	char fl1, fl2, fl3;

	if( fin.is_open() )
	{
		int i = 0;

		fin >> fl1 >> fl2; //first line is outputted from ImageJ/FIJI plugin with useless data, for which this line placeholds

		while( fin >> temp >> col1 >> col2 )
		{   //the three columns from ImageJ/FIJI plugin output: an arbitrary number, the x coordinate in pixels, the y coordinate in pixels.
			V.push_back(S);
			V[i].push_back(col1*pxtomic); //multiply each of these by the conversion factor to get microns
			V[i].push_back(col2*pxtomic);

			i++;
		}
	}
	else
	{
		cerr << "Cannot open file \"" << inputfilename << "\"." << endl;
		return 0;
	}

	if(V.size() > 1)
	{
		double radius = 0;
		vector<double> c = findRadius(V, radius);

		cout << "The average radius length is " << radius << " and the approxiamte center is (" << c[0] << ", " << c[1] << ")." << endl;
		cout << "The approxiamate area of the best fit circle is " << M_PI * radius * radius << "." << endl;
		cout << "The data points lying along this circle can be found in " << outputCirclePoints(c, radius) << "." << endl;
	}
	else
		cout << "There is only one point in this file, therefore the radius is nan." << endl;

	return 0;
}
```

ImageJ/ Fiji "FindMaximaPerSlice.txt" plugin:

```
// The number of slices cannot exceed 9,999 without requiring edits to the code.
// This results because of the way the track name is defined below.

// Track Length Variables- for full track, set the first one to 0 (or 1, depending
// on how the image indexing starts) and the second one to the last image's index

firstSlice=133; lastSlice=133

// Path Variables- The first path is the location of the slices and the second path
// is the location of the outputted data files. This should be the same directory as
// that which contains this file and the file "BestFit.exe".

pathToTrackSlices="G:\\Tori\\T163d_freslice2\\T163d_freslice2_";
pathToData="C:\\Users\\crazygirl\\Documents\\AMNH\\";

for( i = firstSlice; i <= lastSlice; i++ )
{
	current = i;

	// Selects a slightly different file name based on the index. Must be editted for
	// slice indexes greater than 9,999. In the current system, slices indexed less than
	// 10 have only one number digit, and therefore 3 leading zeros. Slices indexed between
	// 10 and 99 have two digits, and therefore only 2 leading zeros, etc. If the track in
	// question has 10,000 or more slices, simply add another else if statement, and up all
	// of the leading zeros by one (such that indexes under 10 will have 4 leading zeros etc)

	if(current<10)
		open(pathToTrackSlices + "000" + i + ".tif");
	else if(current>=10  && current<100)
		open(pathToTrackSlices + "00" + i + ".tif");
	else if(current>=100 && current<1000)
		open(pathToTrackSlices + "0" + i + ".tif");
	else if(current >= 1000 && current < 10000)
		open(pathToTrackSlices + i + ".tif");

	// Process used to filter images and output data points
	run("Smooth");
	run("Brightness/Contrast...");
	run("Enhance Contrast", "saturated=0.35");
	run("Close");

	// Inverts the image and makes it binary so that there are less factors to
	// consider in the next step.
	run("Invert");
	//run("Make Binary");
```

```
        // The noise option here can be toggled for maximum efficiency in the track.
        // The higher the number, the more data points will be eliminated, as it
        // will require the maxima to be more acute.
        run("Find Maxima...", "noise=500 output=List light");

        // Saves the results to a different data file per slice, all in the directory "data_files/"
        // which should have the same parent directory as this file, and the file "BestFit.exe"
        saveAs("Results", pathToData + "data_files\\data" + i + ".txt");

        // Closes each image after the process is run.
        //close();
}
```

README:

Documentation for the Best Fit Circle Process for Comet Particle Track Analysis

This distribution should contain the following: an ImageJ/Fiji plugin called “FindMaximaPerSlice.txt”, a C++ source file called “BestFitCircleProgram.cpp”, an executable called “BestFit.exe”, and an initially empty directory titled “data_files”.

The purpose of this distribution is to determine the volume of a comet particle track from a series of cross-sectional images. Each image of this sequence should be a tif file, meaning that the entire track is essentially a 3D image itself. Through this process, each of these cross-sectional images are fed through the image processor ImageJ/Fiji, which determines a list of data points based on the maximum valued pixels contained in each image. These data points are saved in separate data files in the directory “data_files”. For more specific details and instructions on how to run this step, see the below section titled “Step 1: ImageJ/Fiji Plugin”. Next, each data set is individually analyzed for its best fit circle. The area of these best fit circles are added together to yield the cumulative volume of the entire track. The file “Areas.txt” created in this step contains details for each image slice, including the area, radius, and approximate center. The resulting volume should display automatically, but will also output as the last line in “Areas.txt”. For more specific details, see the below section titled “Step 2: Finding the Best Fit”.

/************************** How to Use This Software ******************************/

Step 1: ImageJ/Fiji Plugin:

First and foremost, be sure that the directory “data_files” exists in the parent directory of this distribution. If it does not exist, make a directory and title it “data_files”.

Next, open up the text file titled “FindMaximaPerSlice.txt” and locate the path variables. Please specify the path to the track’s cross-sectional images using the same syntax listed in Windows, replacing the

"\\" with "/" in Linux or MacOSX. The images should have the same name except for the last few digits, which represent their position in the track. This software is capable of analyzing tracks with a total number of cross-sections between 1,000 and 9,999 because of the way the plugin determines the current image's name. By changing the number of leading zeros at each threshold, this limit can be altered. See the in-file comments for details.

Once you have the path to the tracks specified, specify the location of this parent directory. If there is no "data_files" directory in this parent directory, and/or the path to this directory is not specified correctly, then the plugin will not be able to save any of the data points for later steps.

Next, look for the track length variables. You have the option of analyzing however many images slices you would like. Simply set the first track length variable to whichever slice you wish to start at, and the second variable to the last slice you would like to analyze. DO NOT INCLUDE LEADING ZEROS which may be contained in the name of the track. For example, if the first track in the file is "track_path/track_name0000.tif", then set the first variable to 0, not 0000. The leading zeros will be accounted for later on (see the first paragraph in this section for details).

Once you have the track length and path variables properly specified, open ImageJ/Fiji (it doesn't matter which in this case) and click the menu option "Plugins". On the drop down menu, hover over the option "Macros", and select the option "Run…" In the browsing window, navigate to this folder, or type in the system path. Select the plugin "FindMaximaPerSlice.txt" and hit open. The plugin will then start running. It will take quite a while to complete the full procedure, but there's no need to worry about memory overflow as no more than one image is open at a time.

Once this step is completed, there should be a data file for each image slice contained within the directory "data_files".

Step 2: Finding the Best Fit

To find the approximate volume for this length of track, run the executable file "BestFit.exe". You can do this by navigating to the appropriate directory via the command line and typing "BestFit" or by opening the directory in the graphical user interface and double clicking the executable file. The following text should appear in the console:

> "Best Fit Circle Program
>
> Make sure that you have run the ImageJ/Fiji plugin "FindMaximaPerSlice.txt" prior to running
>
> this software.
>
> 1. Process Solo Image
>
> 2. Process Image Stack

How would you like to use this software?"

Option 1 allows the user to analyze an individual image slice. This option will display the results of the best fit circle calculation directly on the console, as well as create a data file containing the data points of this best fit circle in a text file titled "BestFitCircle.txt". To use this option, the user must have a data file titled "data{i}.txt", where {i} is the index the program will ask the user to specify, or they must know the location and name of a specific file they wish to use. The format of the files inputted into this software is very particular due to the way in which ImageJ/Fiji outputs the data points. In order to run this software successfully, any file inputted must have two elements on the first line, and three on each subsequent line, with the same separator between each.

Option 2 allows the user to analyze an entire image stack. This could be the entire meteorite particle's track, or a portion of it. The volume of the track will be displayed on the terminal, as well as in the bottom of the file "Areas.txt". The individual area, radius, and approximate center for each image in this sequence are also stored in the file "Areas.txt". If the user selects this option, then the data points for each best fit circle are not outputted. The range should be specified with the first track to analyze first and the last track to analyze second with only a space in between (ex. 0 3736). If one of the files does not exist, the program should report which file it couldn't locate, and then skip it. In this case or in the case that there are not enough data points in the file, the resulting output in "Areas.txt" will be 'nan', meaning 'not a number'. This does not impact the overall result so long as it occurs in only a small fraction of the total analyzed files.

/********** Possible Errors and Troubleshooting Tips **********/

The accuracy of the best fit program is analyzed in more detail in the paper, "Determining the Volume of a Comet Particle Track via an Automatic Best Fit Circle Method", which details the whole process used to determine the best method of automatically analyzing the comet particle tracks. It is important to note that the amount of noise on the image slices is directly proportional to the accuracy of the results, and so reducing the error caused by the instruments plays a key role in the calculation of a track's volume. The said, one semantic error that may impact the results heavily is the number of viable images used. A viable image is one which produces more than one data point when run through the ImageJ/Fiji plugin. If the image is so noisy that all of the points are eliminated, perhaps it would be prudent to lower the noise tolerance used when calculating the maxima. This will not make a difference if the image is made binary, as the noise tolerance is based on how close a pixel comes to the maximum amount, and in a binary image, there is only two pixel colors used (white and black). Playing around with the noise tolerance (removing the "make binary" step), and whether or not the image is inverted, is a great way to customize this plugin to the individual needs of different tracks. See the in-file comments for details about where these edits can be made.

References

"Aerogel." Wikipedia, 20 Aug 2012. 30 Aug 2012. < http://en.wikipedia.org/wiki/Aerogel>.

Burchell, Fairey, Wozniakiewicz, Brownlee, Horz, Kearsley, See, Tsou, Westphal, Green, Trigo-Rodriguez, Dominguez. "Characteristics of Cometary Dust Tracks in Stardust Aerogel and Laboratory Calibrations." Meteoritics and Planetary Science 43, Nr ½, 23-40. The Meteoritical Society, 2008. <http://meteoritics.org>.

"Comet." Wikipedia, 21 Aug 2012. 30 Aug 2012. <http://en.wikipedia.org/wiki/Comet>.

"Comet 81P/Wild 2 Under a Microscope." Science, Vol 314. 15 Dec 2006. <www.sciencemag.org>.

Ebel, Greenburg, Rivers, Newville. "Three-Dimensional Textural and Compositional Analysis of Particle Tracks and Fragmentation History in Aerogel." Meteoritics and Planetary Science 44, Nr 10, 1445-1463. The Meteoritical Society, 2009. <http://meteoritics.org>.

Greenburg, Michael. "Properties of Original Impactors Estimated from Three-Dimensional Analysis of Whole Stardust Tracks." Department of Earth and Planetary Sciences, American Museum of Natural History, NYC, USA. The Meteoritical Society, 2012.

Ferreira and Rasband. "ImageJ User Guide." 5 July 2012. Section 29.4. <http://rsbweb.nih.gov/ij/docs/guide/146-29.html#toc-Subsection-29.4>.

Kearsley, Price, Burchell, Cole, and Foster. "How The Shape and Volume of Impact Tracks in Stardust Aerogel Reflect Cometary Dust Properties: Experimental Evidence." Department of Minerology, Natural History Museum, London. 43rd Lunar and Planetary Science Conference, 2012.

Sandford, Scott. "Fluorescence Imaging of the Stardust Collector Trays." 3 Oct 2006.