Expansion of Search Capabilities for the Target Fabrication Database

Benjamin Smith

Expansion of Search Capabilities for the Target Fabrication Database

Benjamin S. Smith

Webster Schroeder High School
Webster, NY

Advisor: Luke Elasky

Laboratory for Laser Energetics
University of Rochester
Rochester, NY

August, 2007

Abstract

In order to develop more perfect cryogenic fusion targets, the Laboratory for Laser Energetics (LLE) maintains a database with information on the quality of past targets and the process details used to manufacture them. The database contains three tables, for file information, image information, and analysis information. These tables are organized in the same way and contain an identification number unique to each specific target image. Though access to this information is relatively simple on an image-by-image basis, there is no routine in place to access the same information on a large scale. This project entailed the creation and development of an application to fit that need, called DBsearch. The application, implemented in MATLAB (a data processing environment), allows a user to compile a list of files according to the user's parameters – for example, only files whose comments contain the word "rotation" – and then request a return of any value from those files across the three tables. Once the data is retrieved, it can be viewed as text, plotted to a graph using MATLAB's plotting utility, or exported to the MATLAB workspace for further processing. Selecting data according to parameters allows a user to view a large segment of data without unwanted variables.

1. Introduction

One of the primary goals for the LLE is to research Inertial Confinement Fusion. For this research, cryogenic targets filled with isotopes of hydrogen are imploded by the laser in order to create the environment necessary for a fusion reaction. The creation of these targets is a complicated process with many variables to account for. The targets are created and processed in Moving Cryostat Transfer Carts (MCTCs). The target shells, pre-manufactured hollow plastic spheres, are filled with fuel by diffusion when the surrounding pressure is slowly increased to up to one thousand atmospheres. Once the fuel is inside the shell, the targets are cooled to cryogenic temperature and then transferred into MCTCs. The entire cart is then moved to a separate station used to monitor the target during the processes used to smooth the frozen hydrogen. While at this station, and usually after the target has been frozen and the hydrogen ice smoothed, a program called CryoView, developed at the LLE, takes images of the target and stores them in a database (for an example, see appendix 1). Different types of information are stored in different tables of the database, TFAB. Background information about the image-- such as the target's ID, or the cart the target is in-- is entered into a table called cryo_vdp_file. Information about the circumstances of the image-- such as the camera type, the illumination level, or the relative position of the target-- is entered into the table called cyro_vdp_dataset. After the image is analyzed, that data-- including the ice thickness, the layer quality of the target (smoothness of the ice), and other measured values-- is entered into cryo_vdp_analysis. Between the three tables, much of the relevant information about the target's fabrication and quality are stored.

In the past, the information stored in those tables has been accessed using a direct Structured Query Language (SQL) query to TFAB. The central problem with this was that it required the user to build extremely complex queries in order to receive the appropriate results for relatively simple requests. The tables are structured on a three-column system: the first holds an identification number for each record, the second holds a description of the data in that particular row, and the third the actual value of the data (Figure 1).

Targt ID	Descrip	Value
2Y19354	MCTC	1
2Y19354	Shell Mat.	StrongCD
2Y19354	WallThick.	2.1
2Y19354	Fill ID	171
2Y19354	Ice Thick	94.6
2Y19354	Station	1X
1X22320	MCTC	2

Figure 1: an example of the three column database format.

So for each image generated by CryoView, a unique ID is given to the file and used across all three tables. Each image has a series of values input into each table, each with a description and a value set into an individual row. In order to access the information in a particular row, the user only has to know the file

ID and the description to get the value. The user can also find the appropriate file ID by using SQL to compare values, using a statement like “SELECT cvf_file_id FROM cryo_vdp_file WHERE cvf_descrip = 'MCTC' AND cvf_value = '1'”, selecting all files that originated in MCTC #1. The disadvantage of SQL is that developing more useful searches almost always leads to much more complicated queries. In order to simplify the process of obtaining useful information from the database, a utility needed to be written.

2. Functionality

DBsearch has two focuses for functionality: to simplify the process of finding the appropriate files, and to simplify the process of analyzing the resulting data. The entire process is broken up into three steps, the first two of which are concerned with locating the data, the third with presenting the data.

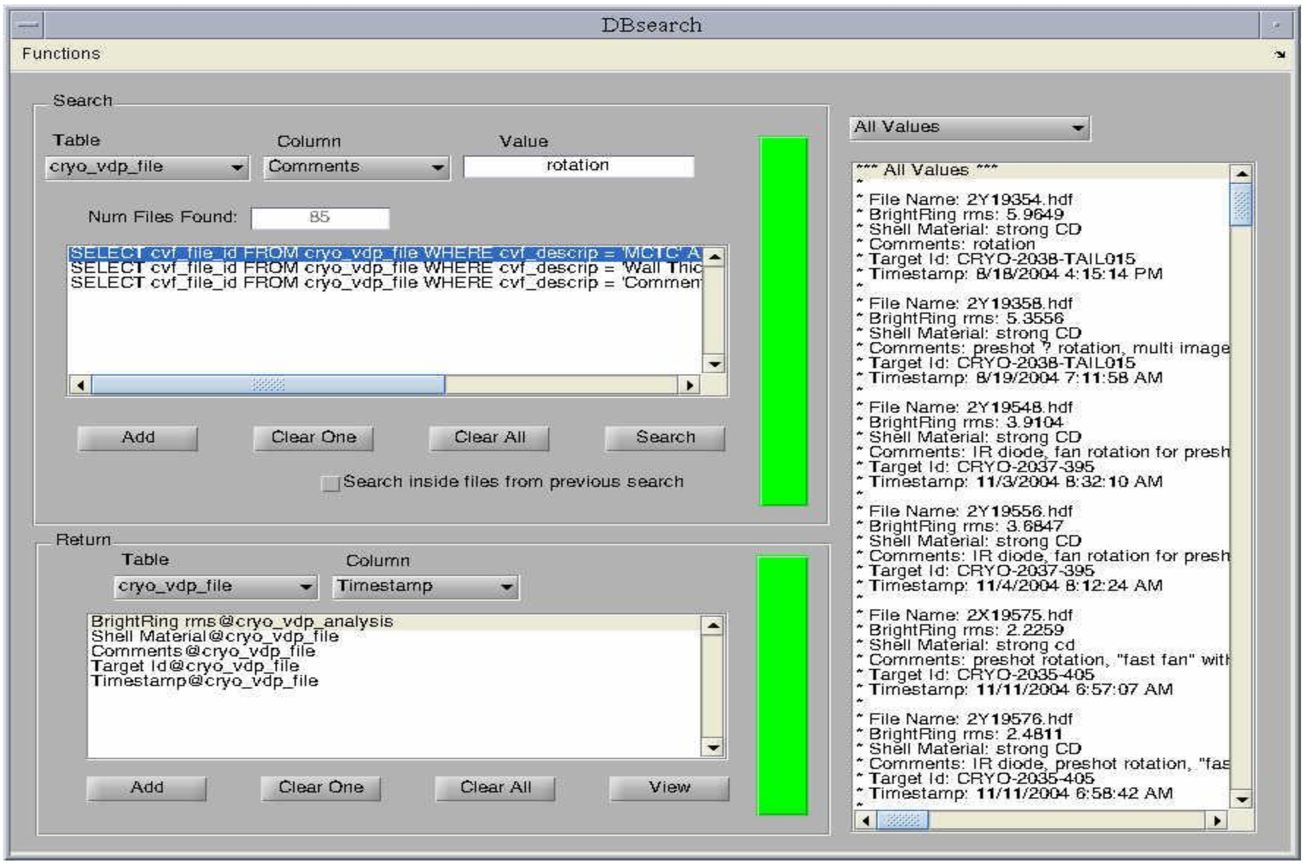


Table: cryo_vdp_file
Column: Comments
Value: rotation
Num Files Found: 85

```
SELECT cvf_file_id FROM cryo_vdp_file WHERE cvf_descrip = 'MCTC' A
SELECT cvf_file_id FROM cryo_vdp_file WHERE cvf_descrip = 'Wall Thic
SELECT cvf_file_id FROM cryo_vdp_file WHERE cvf_descrip = 'Commen
```

Search inside files from previous search

Table: cryo_vdp_file
Column: Timestamp

```
BrightRing rms@cryo_vdp_analysis
Shell Material@cryo_vdp_file
Comments@cryo_vdp_file
Target Id@cryo_vdp_file
Timestamp@cryo_vdp_file
```

All Values

```
*** All Values ***
*
* File Name: 2Y19354.hdf
* BrightRing rms: 5.9649
* Shell Material: strong CD
* Comments: rotation
* Target Id: CRYO-2038-TAIL015
* Timestamp: 8/18/2004 4:15:14 PM
*
* File Name: 2Y19358.hdf
* BrightRing rms: 5.3556
* Shell Material: strong CD
* Comments: preshot ? rotation, multi image
* Target Id: CRYO-2038-TAIL015
* Timestamp: 8/19/2004 7:11:58 AM
*
* File Name: 2Y19548.hdf
* BrightRing rms: 3.9104
* Shell Material: strong CD
* Comments: IR diode, fan rotation for presh
* Target Id: CRYO-2037-395
* Timestamp: 11/3/2004 8:32:10 AM
*
* File Name: 2Y19556.hdf
* BrightRing rms: 3.6847
* Shell Material: strong CD
* Comments: IR diode, fan rotation for presh
* Target Id: CRYO-2037-395
* Timestamp: 11/4/2004 8:12:24 AM
*
* File Name: 2X19575.hdf
* BrightRing rms: 2.2259
* Shell Material: strong cd
* Comments: preshot rotation, "fast fan" with
* Target Id: CRYO-2035-405
* Timestamp: 11/11/2004 6:57:07 AM
*
* File Name: 2Y19576.hdf
* BrightRing rms: 2.4811
* Shell Material: strong CD
* Comments: IR diode, preshot rotation, "fas
* Target Id: CRYO-2035-405
* Timestamp: 11/11/2004 6:58:42 AM
*
```

Figure 2: Main Window of DBsearch

2.1. Locating Data

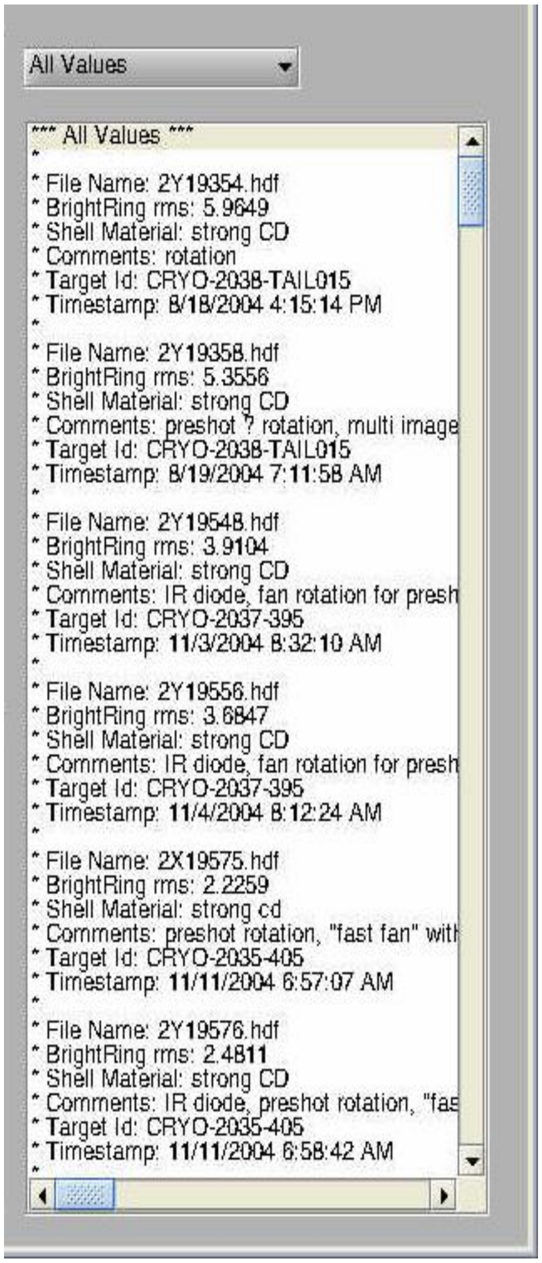
Because DBsearch locates files through search parameters, the first step in accessing the desired output data is to find which files meet search criteria. DBsearch uses a menu and text box system to read the user's input into the program (Figure 2). The first menu contains a list of the three tables that DBsearch can access; the second contains the list of values contained in that table for each file. The third input is variable, depending on what type of parameter has been selected in the second menu. If the parameter has a limited number of different values, for example which MCTC the target was processed in, the third input field is a menu from which the user selects the appropriate value from the list. If the parameter is a string-- a combination of numbers and letters-- then the input is a simple text field. Finally, if the parameter is a numeric value, then the input is a text field with an additional field for a tolerance, allowing the user to input a value such as “37 +/- 2” using both parts of the input. Once the desired values are represented by the inputs, the criteria are added to the stack with three additional controls. The first control adds the criteria currently in the input fields to the stack, the second removes the selected criteria from the stack, and the third clears the entire stack. Once the stack accurately represents the search the user wishes to perform, the search control begins the process of searching the database with those criteria to compile a list of the fitting files. The colored status bar in association with this part of the interface will turn red while the program is busy; once the routine has finished, the bar will return to the original green color. In addition, the user has the option of only searching through the files in the list from the last search, in order to reduce the time taken to finish the search.

The second step involves selecting the data which the user would like returned. This uses a second part of the interface with similar controls to the first (see Figure 2, above). The first menu again selects the table which the user is accessing, the second the data within that table to be returned. These menus show the same information as the first two menus in the first step part of the interface. Also, the three controls to manipulate the stack are similar: one to add, one to clear a specific row of the stack, and a final one to clear the entire stack. The view control in this section initiates a routine that proceeds through the list compiled in the first step, storing the data in local memory. Another status bar behaves in the same way as the first, red when busy and green when done.

2.2. Presenting Data

The main interface displays the first option for presenting data: a simple text output (Figure 3). The display consists of another drop-down menu and a scrolling text box. The menu allows the user to select either "All Values"-- to display each file with all returned values shown-- or select from the list any one of the values returned to display the files with only that value shown. The text box shows in the first line which display is being shown, and then the following lines show the file name followed by each value, followed by a blank line, then another file in the same format, and so on.

The remaining display and output options are accessible from the functions menu: extract and plot. The extract function displays a separate window with a menu, check box and button. The menu allows the user to select any of the returned values, the check box selects either numeric or not numeric, and the button initiates the



```
*** All Values ***
*
* File Name: 2Y19354.hdf
* BrightRing rms: 5.9649
* Shell Material: strong CD
* Comments: rotation
* Target Id: CRYO-2038-TAIL015
* Timestamp: 8/18/2004 4:15:14 PM
*
* File Name: 2Y19358.hdf
* BrightRing rms: 5.3556
* Shell Material: strong CD
* Comments: preshot ? rotation, multi image
* Target Id: CRYO-2038-TAIL015
* Timestamp: 8/19/2004 7:11:58 AM
*
* File Name: 2Y19548.hdf
* BrightRing rms: 3.9104
* Shell Material: strong CD
* Comments: IR diode, fan rotation for presh
* Target Id: CRYO-2037-395
* Timestamp: 11/3/2004 8:32:10 AM
*
* File Name: 2Y19556.hdf
* BrightRing rms: 3.6847
* Shell Material: strong CD
* Comments: IR diode, fan rotation for presh
* Target Id: CRYO-2037-395
* Timestamp: 11/4/2004 8:12:24 AM
*
* File Name: 2X19575.hdf
* BrightRing rms: 2.2259
* Shell Material: strong cd
* Comments: preshot rotation, "fast fan" witl
* Target Id: CRYO-2035-405
* Timestamp: 11/11/2004 6:57:07 AM
*
* File Name: 2Y19576.hdf
* BrightRing rms: 2.4811
* Shell Material: strong CD
* Comments: IR diode, preshot rotation, "fas
* Target Id: CRYO-2035-405
* Timestamp: 11/11/2004 6:58:42 AM
*
```

Figure 3: Example of simple text output

routine. The result is that the function DBoutput, used in the MATLAB workspace, can be used as a variable to access the data. If the data was numeric, then the variable is an array of doubles (datatype in MATLAB used to store numeric values); if the data was not, then the variable is a cell array of strings (a datatype in MATLAB used to store a list of different length character combinations-- words). The plotting function also displays a separate window. The interface allows the user a variety of options: what type of axis to plot on, which variables to plot, and which files of those variables to plot. The selectable types of axes are two dimensional rectangular, three dimensional rectangular, and two dimensional polar. Both the two dimensional plots allow two values to be selected, and the three dimensional allows three values. Also, a single variable can be plotted against index (the number representing the values position in the list of all values). Any value returned by the previous search can be plotted, assuming that the value is numeric. A text box displays the last value selected from one of the menus, and a combination of that text box and two text fields allow the user to select a range of files from the entire set that they wish to plot. Finally, the plot can be shown in the interface window, or it can be plotted to another window to allow access to MATLAB's plotting tools.

3. Implementation

The code written for DBsearch uses event-initiated routines-- meaning that everything done by the program is begun by the user activating a control. Some controls are value controls, which do not activate an event but instead represent a value to be read by the program, some controls are event controls, meaning they begin a routine inside the program, and some include features of both types. Each time a new step is initiated, that step proceeds in a different workspace than all previous steps, so none of the stored values for the previous steps are available to the current step without using a different type of storage. In order to pass important values from step to step, those values either need to be stored by the interface, by a value control, or stored in a MATLAB global variable-- these variables can be accessed from any part of MATLAB once they have been declared.

3.1. Compiling a List of Files

The first step includes two sets of value controls and a main event control to initiate the search. The first set of controls includes the input fields. All of these are value controls, but the first and second menus also have event components. The first menu, where the user selects one of three tables, queries the database for all possible descriptions in that table, so as to have a complete and current list for the user to use. The second menu checks the value against a series of lists; if the value is in one of those lists, the input field changes type (for example, if the value is on the list of menu types, the input will be a menu). The second set of value controls are the stack display box and its event controls. The add event will generate a simple SQL query from the first set of value controls, and then add it to the display stack. The other two controls either clear one item from the stack, or all of the items. The last control is the main event control for the first step of the process-- the search control. When activated, the program begins to go through the stack and narrow down a list of viable files. The first query in the stack is executed as written, returning a list of files from the database that meets those criteria. If there are more files in the stack, the program modifies them before execution, adding a tag to the end of the statement. The tag contains a second criterion for the search that the files must be inside the list from the previous query. Because SQL only supports a certain length of lists in criteria and the number of files often exceeds that limit, DBsearch breaks up the list into 900 file long segments, executes them separately, and adds the segmented list back together after the most recent criteria is met. Once all of the queries in the stack have been run, the length of the list is displayed in the interface and the list itself is stored in a global variable for access by other event routines.

3.2. Locating the Desired Data

The second segment of the interface also has two sets of value controls and one main event control. The first set works exactly like the first two menus used to locate the files. The second set works similarly to its counterpart as well, with a minor difference in the add control. Instead of creating an SQL query, it merely prints the description and the table to the display. When the view control, the main event control for the second segment, is activated the actual database interaction for the second segment begins. DBsearch runs through the list of files compiled for the first segment, and for each file goes through the stack of desired data in the second stack display. Each individual value, once retrieved from the database, is stored in another global variable-- this one a structure organizing the data by file. Once all of the data is stored locally, the routine then generates the strings to be shown in the main output display. These are created by accessing the data stored locally, as opposed to querying the database again.

3.3. Presenting the Data

The main interface has one control and a display for showing the data. The display shows in text form the strings compiled by the main event control in the second segment. While this is not necessarily the easiest way to read and compare data, it is simply a way for the user to ensure that the program has retrieved the correct data. The output reads the data from the local storage, and prints a text display. The control allows the user to choose which data are being displayed at that particular time; the user can choose to have either all returned data displayed or only one field of data (each of these are displayed by database file identification number). The display then shows a label for each ID number, a tree of the fields beneath that label, and a value for each field.

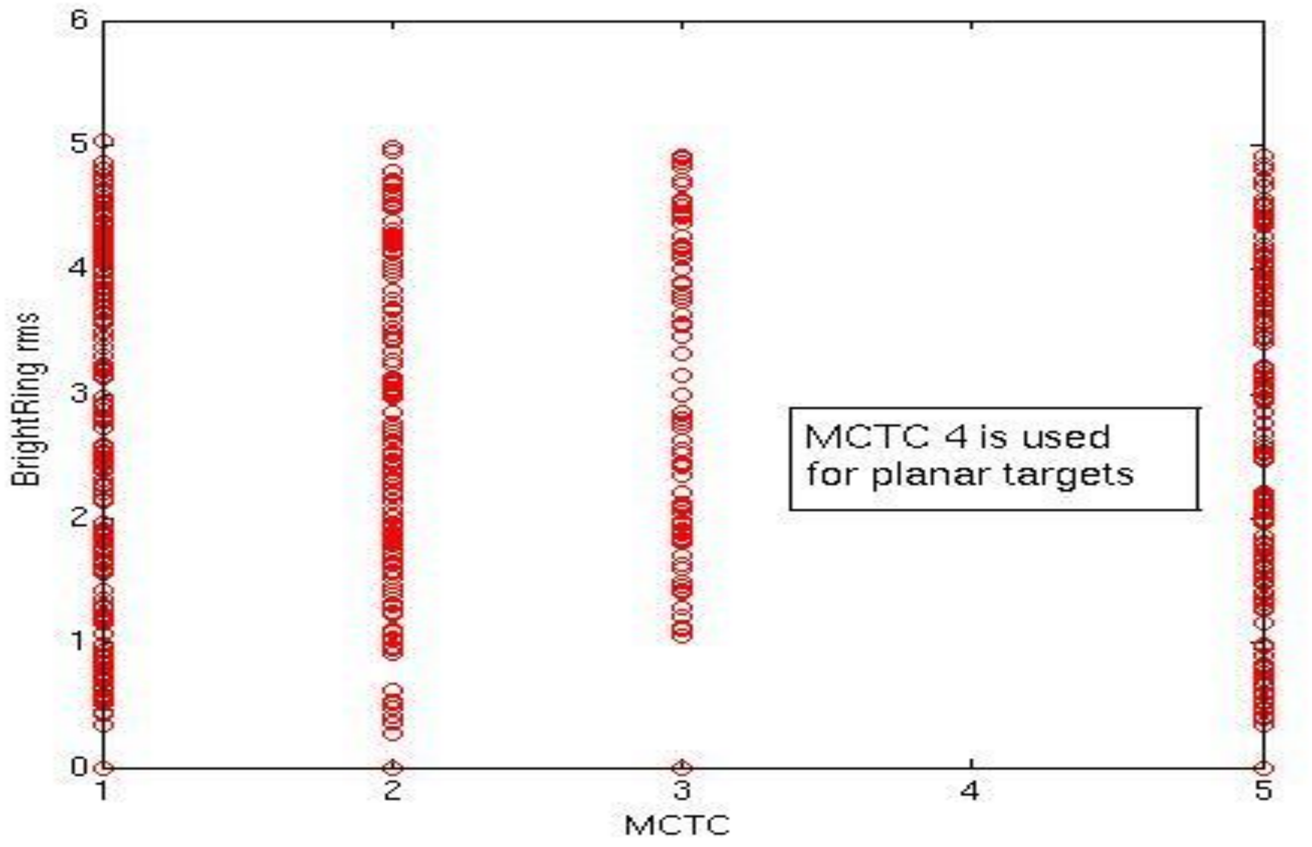


Figure 4: Sample Plot created using DBsearch

The most basic way to present the data in useful fashion is to use the plotting routine. This presents a new window, with several controls and a main display for showing the data. The controls are mainly value controls to interpret what the user would like to be displayed, with a single event control to initiate the actual plotting. The routine then puts the desired variables into arrays readable by MATLAB, according to the input by the user. Depending on the type of plot desired (two or three dimensional), either two or three arrays are generated. In the case of an indexed plot, only one of those arrays contains data from the database, while the other contains a simple incremental list to plot said data against. In the case of any other plot, both arrays are generated from the database. The routine then calls MATLAB's native plotting routine with parameters to place the resulting plot into the display on the window (Figure 4).

In cases where the plotting tool written into DBsearch cannot produce the desired plot, a separate tool can be used to output the data to the general MATLAB workspace. From there, a

MATLAB user can use the more powerful tools available to produce the intended result. The exporting tool from DBsearch's menu allows the general workspace to access either the entire structure used to locally store the data accessed by DBsearch, or an array generated by DBsearch that contains only a segment of that structure. While the structure cannot be read directly into the MATLAB's plotting routine, the array segments can be.

4. Notes for Improvement

This section details a few of the concepts that could prove useful in the future when implemented into DBsearch.

4.1. Movie/Slide Show Generator

This addition would require two additions to DBsearch: a routine for creating and storing slides, and a routine for displaying the slides. The first could be added to the plotting function, as that already produces images that are the primary sources for any slide. All that would then be required is for a button to add the slide to a list that is stored, either in a permanent file or temporarily (to be made permanent later). The second could be made fairly easily out of the code existing for the plot function window. The display can be modified to show images in sequence, and the inputs can be modified to choose the particular slide show to be displayed.

A third addition which could prove useful would be another window for displaying and ordering specific slides, in order to create the desired show. This would allow users to tailor the show more easily, as they would not have to generate the slides in order and they would not have to re-generate slides used in the past (if those slides themselves were stored separately).

4.2. Additional Data Tables and Databases

The tables currently accessed were chosen because they all held data for the same files. There are other tables in the database, and many more databases. The code used in DBsearch could be fairly easily made to fit almost any table set up using the three-column system. The higher functions, such as plotting, may or may not prove useful in the new context, but the searching and returning code, the bulk of the code written for DBsearch, could be re-used.

The new tables would not necessarily be linked to the same application currently called DBsearch. This would make little sense, as the other tables hold very few related data. These would be run by other teams working with databases on projects not always associated with the manufacture of cryogenic targets.

4.3. Code Efficiency

One of the basic tenets of software coding is that the code can always be made more efficient. For example, the code for DBsearch makes almost no use of MATLAB’s number-crunching ability. As a language, MATLAB was designed to perform calculations on a large amount of data. DBsearch, on the other hand, uses an algorithm which processes tasks in sequence. The code could be made more efficient if the algorithm was better suited to the language’s structure.

This is only one example of the many ways any piece of code could be improved. As an application, DBsearch runs fairly slowly—mostly because of the database interface and the time it requires retrieving data. Any improvement in processing time is an improvement that could be very helpful when processing large queries.

5. Acknowledgements

Dr. R. Stephen Craxton, who accepted my application to the Laboratory’s summer internship program;

Luke Elasky, for mentoring me through the creation of DBsearch;

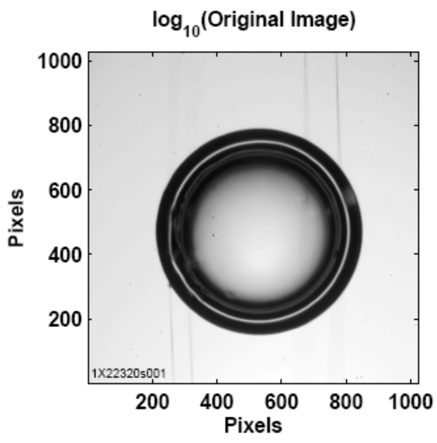
The LLE ITT staff, for providing assistance in a huge variety of ways;

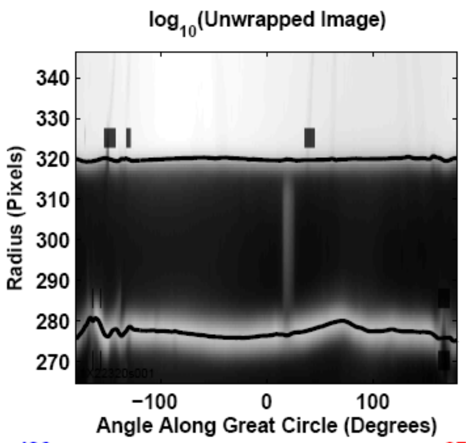
The Laboratory for Laser Energetics, which hosted an amazing internship program and which provided support to each and every one of its interns on every project.

Appendix 1: Example CryoView Output

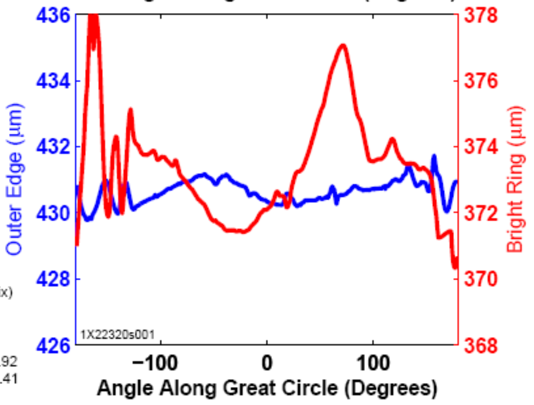
1X22320.hdf set# 1
1/11/2008 6:19:13 AM

CRYO−2020−1081





```
Surface Edge        Ring      Ice       Layer
R(pix)    319.9     277.4     246.1      73.8
R(µm)     430.6     373.5     331.3      99.4
Height    1842.9    951.4
Width     3.4       4.9
RMS(µm)   0.34      1.57      1.56       1.72
%interp.   6.9%     3.6%
```

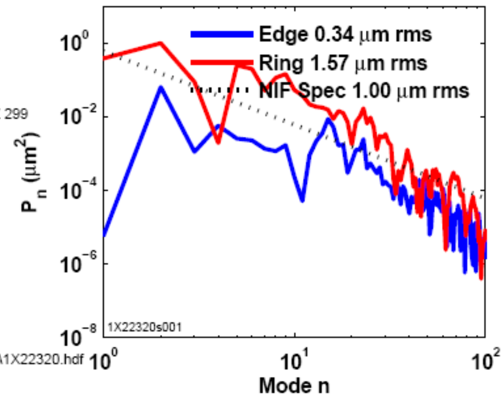


Target Info
Name: CRYO−2020−1081
MCTC: 5
Shot: n/a
Outside Diameter: 861.3
Wall Thickness: 2.1
Shell Material: Strong CD
Fill ID: 171
Fill Pressure: 1017.72
Ice Thickness (Pressure): 96.4
Ice Thickness (MolesOut): 101

Analysis Info
ViewCryo 4.0 (D.E. 12/31/04)
11−Jan−2008 06:39:43
Center: X=530.21, Y=471.18 (pix)
Offset: 44.9589 ± 0.0000 (pix)
DarkField_1x73285539378
nIter: 3
Edge: R 319.93,W 3.39,H 1842.92
B Ring: R 277.44,W 4.91,H 951.41

Imaging Info
Station: 1x
Imaging F:
Imaging Mag:
Camera: B1064
LED: Luxeon III LXHL−PD09
LED Wavelength: 630
Illum. F:
Source:
Fiber: Short
Illum. Mag:
µm/pixel: 1.35

Dataset # 1 Info
Target View:θ=26.6, ϕ=140.1
Target Pos: X −145, Y 1647, Z 299
Sphere Temp: 18^{0}K
LED Current: 2000
Pulse Length: 200
RepRate: 1
Camera Gain: 1
Camera Binning: 1
Integration Time: 50
Laser Setting: 210
Image Size: 1024x1024



\\REDWOOD\cryodiag\Room157data\DevelopmentData\2008−01\1X22320.hdf
Rotation
Set# 1: Rotation