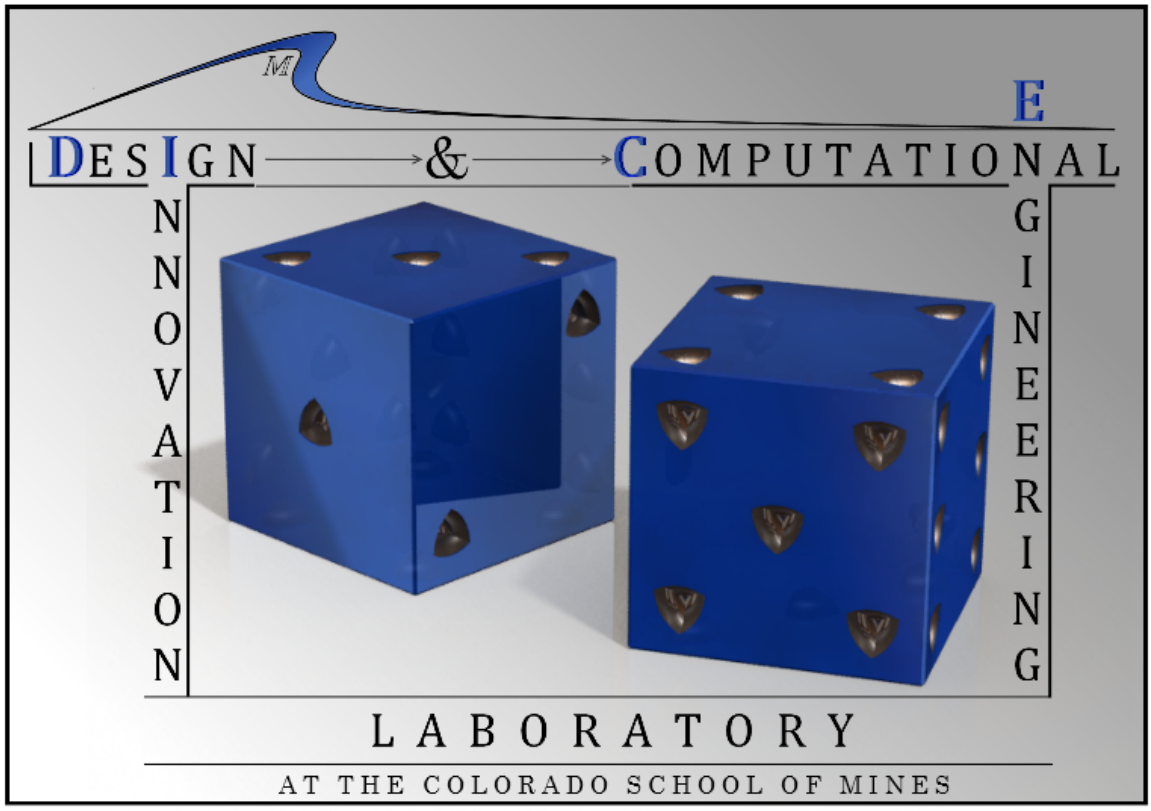

Matt Bailey, Kevin Emery, John Phillips, Timothy Waite

Dr. Cameron Turner

June 18th, 2013

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

Our client, Dr. Cameron Turner, is a mechanical engineering professor at the Colorado School of Mines. Dr. Turner earned his Ph.D. from the University of Texas at Austin and his research interests now focus on the combination of computing and engineering. Many of his projects utilize computers in order to optimize and visualize the design process involved with engineering. One of Dr. Turner's visualization projects is a program designed to model non-uniform rational B-splines (NURBS). This type of software is fantastic for modeling smooth surfaces from an approximation of points. For example, NURBS have been used to create smooth profiles for cars, which consumers find more aesthetically pleasing.

However, Dr. Turner's code was written and last compiled in Microsoft Visual Studio 6.0 over 7 years ago, and is not currently compatible with the C++03 standard. This project is National Science Foundation (NSF) backed and the grant is running out shortly, so Dr. Turner had asked a team to help build a web application to make this software more widely available, as well as optimization and updates to the code. The major goal of this project was to create an integrated system of programs that allows a user to submit a modeling request via a website, then receive the results of the NURBS modeling software, and finally provide a Matlab or Mathematica script that provides a visual representation of the data. The other portion of the project involves a major refactor of the original codebase. The refactor was primarily focused on making the code C++03 compliant and object oriented. The major secondary goal of the overhaul is to optimize the code and increase it's speed.

2. REQUIREMENTS

2.1 FUNCTIONAL

1. There needs to be a website a with graphical user interface (GUI) for job submission to the modeling software

2. Jobs submitted through the website must be run in a specific order based on who submitted the data.

3. The main program must be object oriented and up to date with C++ standards

4. Main program must have a GUI interface (separate from website GUI), which matches the flow of the existing code

5. There needs to be some sort of script that allows for model visualization

6. The refactored code base must be well documented to aid in further modifications

2.2 NON-FUNCTIONAL

1. Modeling software refactored to improve efficiency

2. Threading added to main program to improve efficiency

3. GPGPU optimization added for increased efficiency on RAMA

4. Program able to run on any workstation, regardless of number of CPU cores or a lack of GPU

5. Cross-platform capability

3. SYSTEM ARCHITECTURE

3.1 FLOWCHART

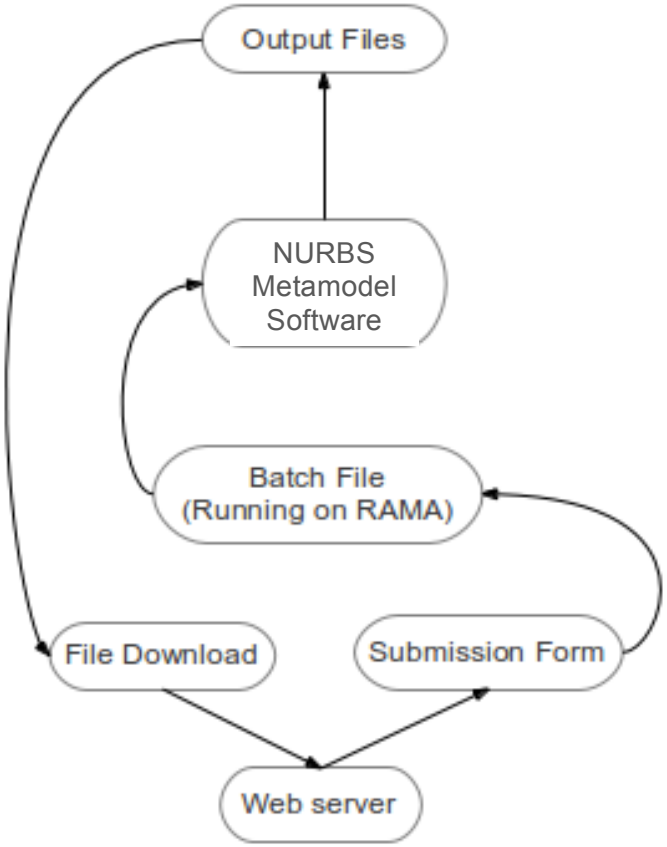

FIGURE 1 - FLOW OF DOCUMENTS SUBMITTED THROUGH THE WEBSITE

3.2 HIGH LEVEL DESIGN

The basic design flow of information for our system is relatively straightforward. The flow begins on the website, where a user sets some basic configuration settings and submits an input file. This web form will then create a configuration file and transfer both the configuration file and the input data to a priority queue stored on RAMA. When the job reaches the top of the queue, RAMA will execute the NURBS program with the input files received from the input. Upon completion the files will be sent back to the website for temporary hosting. The user who submitted the job request will receive an email so they can download their results file as well as links to Mathematica scripts that will provide them with a way to visually represent the model.

3.3 DETAIL DESIGN

WEBSITE AND BATCH FILE

The website is the main interface between the NURBS program and the outside world. It allows people to submit data they want modeled (we call these 'jobs') and configure how it is modeled. To submit a job, the users visit dicelab.mines.edu. Upon their arrival, visitors who want to create a model select the 'NURBS-Metamodel Submission System' left-hand menu item. This creates a submenu on the left hand side and allows them to click on the 'Create New Model' link that takes them to a page to either upload both the configuration file *and* their data, or create their configuration file through a form and upload their data file. The user then clicks submit, at which point the data will be validated and sent to the server. If they submitted invalid data they are given an error message and asked to return and fill out the form again. Once their data is submitted an email message is sent from the server to Dr. Turner notifying him that new jobs have been submitted so that he can execute a batch script (currently residing on RAMA) that downloads the jobs and runs the NURBS model building program on the data, which then uploads them back to the server and sends notification emails to the people that submitted the jobs. People who have finished jobs are then directed to visit the 'retrieve results' page, where they enter their email and job name and download the zip file containing the result data.

NURBS METAMODEL SOFTWARE

Due to the requirements given by Dr. Turner, there are two versions of the metamodel software. Both of these programs fit differently into the system architecture. The first version of the software will simply be executed by the batch file. When the batch file sees that there are new jobs in the queue ready to be executed it will pass the file to the integrated version of the metamodel software. This software will then run the inputs provided to it and return the necessary output files to the Batch file.

The second version of the NURBS Metamodel software is the stand-alone executable. This will allow Dr. Turner to bypass the web interface, as well as provide several more options for modeling. The architecture for the stand-alone system is relatively simple. A GUI will prompt the user for either a configuration file or the desired inputs for each variable. Then the program will run and return all the desired files to the computer that is executing the modeling software.

Although the batch file never executes the Mathematica scripts, it remains a critical aspect of the system architecture. These scripts allow users to actually visualize the data that has been modeled. After the outputs have been made available to the user who submitted a job request, they will be provided with the scripts. In a sense, this is providing the user a whole new way output from the program, and will simplify working with the outputs for outside users. Without this software, somebody from outside the DICE Lab may not have a way to visualize the model they received.

4. TECHNICAL DESIGN

4.1 WEB APPLICATION AND BATCH FILE

The Web Application is made out of a mixture of three languages - JavaScript, PHP and HTML. When a user visits the site, they get an HTML form with JavaScript interactivity added.

The dataUpload.html page allows people to first choose which form they want to complete - either uploading a configuration and data file, or uploading just a data file and completing a form that builds a configuration file. The page allows the user to dynamically add and remove more integer variables, or change which form they want to fill out. Once they have filled out the form, the data is submitted to a PHP script that validates the information. If the data is invalid, an error message is displayed that prompts the user to resubmit the job. If the data was valid, the user is given a message telling them their submission was successful. At this point on the server contains a zipped directory containing three text files: A data file (Data.txt), a configuration file (Config.cfg), and a job file (Jobs.txt). The name of the job zip file has also been written to one of three queue files - queue1.txt, queue2.txt, or queue3.txt depending on its priority. If the job was in a priority email array specified in the actual php code(currently it only contains Dr. Turner's address), the job's zip filename is appended to queue1.txt. If it ends in '@mines.edu' or '@mymail.mines.edu', and therefore belongs to a student or faculty member of Mines, it is appended to queue2.txt. All other emails are appended to queue3.txt.

When the job is submitted, an email is sent Dr. Turner containing the job name. From there, it is his responsibility to run the batch script pullJobs.bat, which pulls down all the jobs from the server where the php part of our website is hosted. It uses pscp.exe to download the jobs, which is a windows port of secure shell copy (scp). It then executes all the jobs from queue1.txt in order, then queue2.txt, then queue3.txt. The batch script does this by unzipping the job directory and then calling the model-building program inside the job directory. Once the program is complete, the script zips up the current job's directory and sends it back to pebble again via scp.

Once a job is completed, an email is sent to the person who submitted the job, notifying them that their results are ready to be picked up, and that they have two weeks to pick up their results before they are deleted from the server. At this point the entire script is finished and the server is waiting for the user to download the finished job file to look at the output directory. When the user visits the website again, they download their job by filling out a form asking for their email and job name. If there is a job that matches the name and email they submitted, the zip file for that project is downloaded to their computer, and then deleted from the server. In order to analyze the results of this model, users also have the option to go to the download software page, where they can download analysis and visualization software. For more information regarding the implementation of the website and the batch scripts, see Appendix A.

4.2 NURBS METAMODEL SOFTWARE

When we approached this project, we decided to go through and do a full rewrite of the entire code base. This result is a massive refactoring of the original code given by Dr. Turner. The major effect of the refactoring is a conversion to object oriented code, instead of the older, C-style, pass-by-reference style that was in use previously.

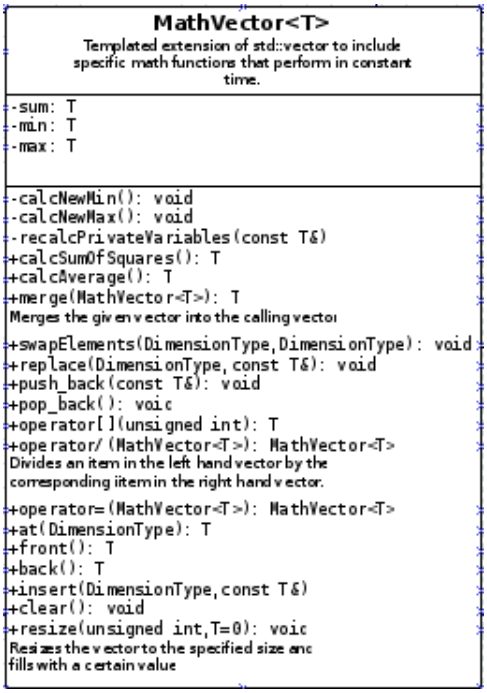

FIGURE 2 - UML OF THE MATHVECTOR DATA STRUCTURE

The first object that was written was the MathVector class, seen in Figure 2. The original code had it's own vector class that was written by Dr. Turner in order to allow dynamically resizing arrays, as the Standard Template Library (STL) vector did not have appropriate functionality when the NURBS software was first written. Because the STL vector now supports most of the necessary functionality our group decided to simply

extend the STL vector and add the additional math functions that were required. Because this is simply a vector with added math functions, the name MathVector was chosen. For efficiency purposes all of the functions were overloaded and written to allow the sum, average, minimum and maximum to be returned in constant time (i.e. no calculations for are needed to get these values)

The next object is the VectorMatrix object, seen in Figure 3. Our team decided to write this class because there was no library that had a matrix class that would provide all the necessary functionality. VectorMatrix is a dynamically resizing matrix composed of MathVectors. This class also has additional math functions that are able to take advantage of the speed of the MathVector class to reduce execution time.

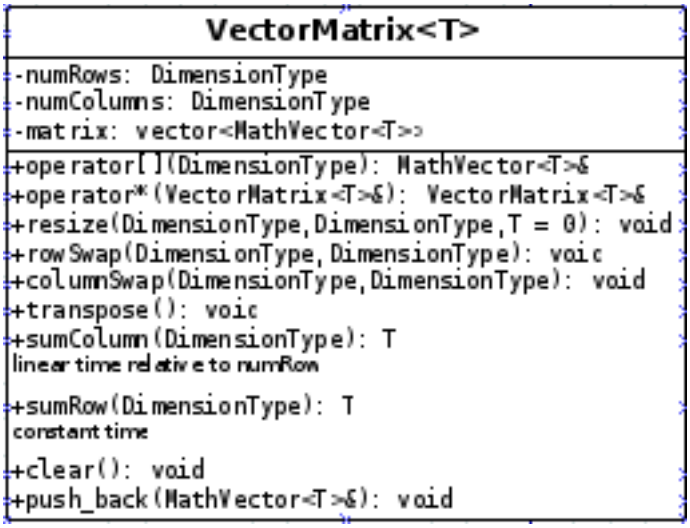

FIGURE 3 - UML OF THE VECTORMATRIX DATA STRUCTURE

The Model class contains the majority of the functionality for the program. When examining Dr. Turner's original code it became apparent that a number of variables were being passed through most of the functions. Because of this, it was decided that these variables should simply be private variables of the model class. This eliminates the need to pass most of the data from function to function, allowing for largely simplified functions and function calls. The primary functions in the Model class are buildModel() and analyzeModel(). These are the two public functions that will be called in the main() function. While this class is unusually large, this is due to the amount of manipulation that is done on a single set of data.

However, not all of the functionality in this software relates directly to a certain piece of data in a model. For example, part of the manipulation that is done on the data is an LU-Factorization in order to solve a system of equations. For this reason, we have the Globals.h file, seen below in Figure 4, which contains the prototypes of many different global functions that are used.

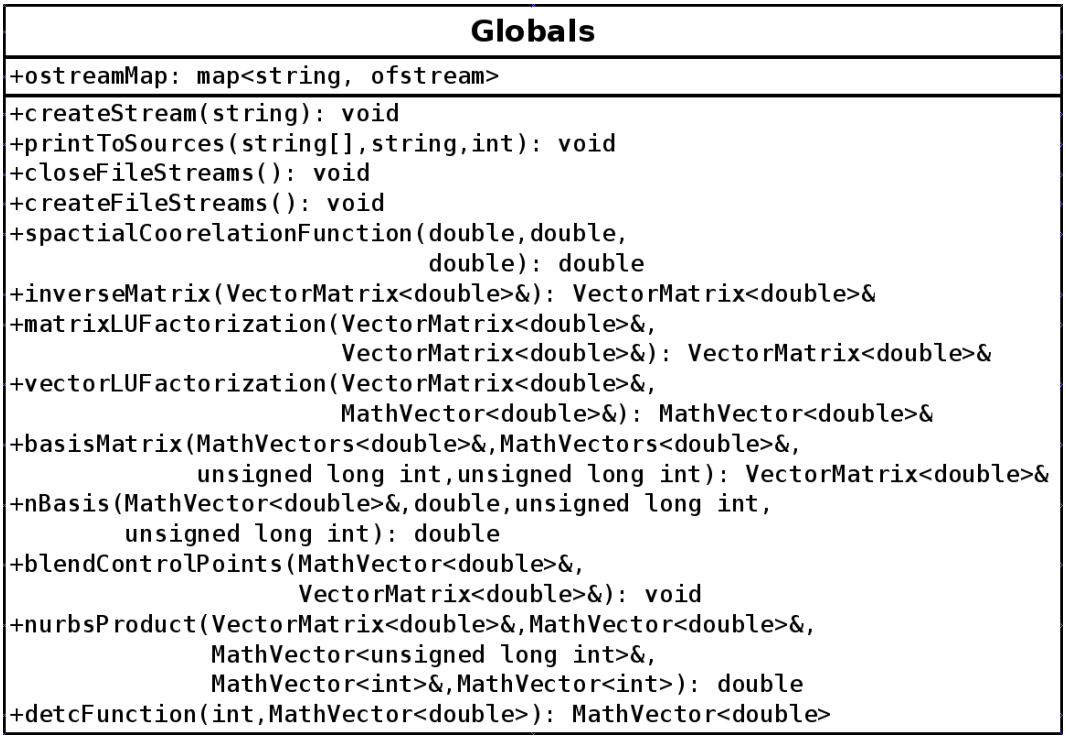

FIGURE 4 - UML DIAGRAM OF THE GLOBAL FUNCTION PROTOTYPES

4.3 MATHEMATICA SCRIPT

The Mathematica scripts were designed to model three function types: 1D in 1D out, 2D in 1D out, and 3D in 1D out. The resulting shapes of these functions were 2D, 3D, and 4D, respectively. The scripts each show two graphs, one of the control points before they are run through the NURBS metamodel software, and the other as a smoothed curve or surface after the software had been executed. The 2D graphs appear on an x-y coordinate system, and the 3D graphs are shapes on an x-y-z coordinate system. The 4D function was graphed on ansy x-y-z coordinate system, with the 4th dimension of points represented by a color depending on the normalized range between 0 and 1.

5. DESIGN AND IMPLEMENTATION DECISIONS

5.1 NURBS METAMODEL SOFTWARE

When beginning this project, the first decision that was made was how to design the metamodel software. There were two options — the first was to refactor Dr. Turner's code from scratch and the second was to finish the code from the previous team who worked on this project. After examining the code from the previous team it became clear that a lack of documentation would make it very difficult to work with their code. For this reason our team decided it would be best to refactor Dr. Turner's original code base, essentially starting from scratch.

Once we decided to rewrite the code base from scratch we needed to decide the appropriate C++ standard to use. Currently there are two primary releases of C++. These releases are C++03 and C++11. Because all compilers do not currently support C++11, our team decided that C++03 would be the best choice.

The last major decision related to the NURBS Metamodel software was the library that we were going to use for the graphical user interface. Because the code needs to be run on both Linux and Windows, the QT GUI library was chosen. This will allow the code and GUI to be cross-platform compatible and used on any operating system.

5.2 WEBSITE

The next set of decisions were all related to the website. Because CCIT is encouraging all campus websites to follow their template-based designs, our first decision for the website was to pick one of the templates. Originally, our group planned on using the department template, but CCIT requested that this be changed to the inside.mines.edu template, and therefore we ported over the code into that template.

The next major decision that was made was whether the web language used would be PHP or ASP.net. It was decided that PHP would be a better choice for two reasons. First, two of our group members had some prior experience with PHP. Second, PHP is more widely used and documented than ASP.net.

The last major decision that was made regarding the website was where to host all of the files that users uploaded and were returned. Because we were unable to get the appropriate permissions on pebble.mines.edu set up, we decided to use RAMA as a web server for the file hosting. In the end, this decision proved to be a better choice, as it allows for far superior security options, and RAMA can also be turned on and off as desired.

5.3 VISUALIZATION

The last major decision was picking the scripting language that would be used for the visualization. After discussing this topic with our client, we found that there were two choices — Matlab and Mathematica. Further discussion revealed that Mathematica is a far more popular program than Matlab. Dr. Turner and John Steuben had also already written some Mathematica scripts for visualization. Because of Mathematica's popularity, the previously written scripts, and time constraints, our group chose to write the necessary visualization software for Mathematica.

6. LESSONS LEARNED

Many of the issues that our group had were related to the amount of work that we had. A lot of the work was not incredibly difficult, simply time consuming. However, there were still a couple of issues that we found taught us valuable lessons.

The first lesson we learned was that it is important to familiarize yourself with the server before you begin the project. We ran into a number of problems while setting up the website, because we were unaware of many of the limitations of the server. For example, in the early stages of development we were attempting to use CCIT's Illuminate server, which did not enable PHP. Because of this we had to spend extra time transferring all the PHP sections of the website to Pebble (an alternate server). Another issue we had with the server was an upload limit. The NURBS Metamodel software sometimes takes very large data input files, upwards of 20 megabytes, but Pebble's PHP upload limit was two megabytes. Because of this we were get odd errors when trying to upload large files and more time had to be devoted to solving this issue. Both of these mistakes could have been avoided if we had taken more time to learn the limitations and restrictions of the servers used for the website.

The second lesson we learned is that it is important to begin integration early. Our project consists of a number of different parts that all come together to make one large integrated software system. Because of the scope of this project, it became difficult to integrate all of the sections together because of a number of compiling errors due to development on multiple operating systems and IDEs. Although we were able to complete the integration in time, if we had been sure to develop on one operating system and begin integration just a little bit earlier, it could have saved our group a lot of stress.

Overall we feel that we tackled this project in a fairly efficient manner. There were some little hiccups throughout the process, but we did not have many that stopped our team from working. Many of the hitches that we ran into were quickly resolved by pairing to find workarounds or solutions to our problems.

7. RESULTS

This project had two major goals. The first goal was to create a set of applications that allowed users to submit jobs through a website that go into a queue. These jobs would then be executed in the order with which they were submitted, with priority being given to students at Colorado School of Mines. The results then need to be returned to the website where the user could download them, along with some visualization software.

The second goal was to refactor the original NURBS Metamodel code base into clean, C++03 compatible, object-oriented code. The major non-functional requirements were focused on optimizing the code for efficiency.

The software we have created meets all of Dr. Turner's functional requirements. The website allows for users to submit jobs with configuration settings of their choice, and a set of data points to the server. This information is then transferred to RAMA in the DICE lab and executed by its priority in the queue (lab members, then Mines students, then outside world). After the NURBS modeling software has completed modeling the inputs, the output files are hosted for download and an email is sent to the user. The user can then download the results, the visualization software written in Mathematica, and a program with which they can analyze the results.

The NURBS metamodel code has been refactored to be object oriented, but still contains all the functionality of the original code. Care has been taken to ensure that all the code is commented and remains similar to the original code so that Dr. Turner and his students are still able to read and modify the code as desired. Unfortunately, due to time constraints, we were unable to do any optimization of the code base, but that work can be done by another team in the future.

8. APPENDICES

APPENDIX A – BATCH FILE AND WEBSITE MAINTENANCE

THE BATCH FILE – IMPORTANT METHODS AND VARIABLES

In order to work correct, the batch file requires several programs to be in a directory specified by %programDirectory% script. Theses are listed below

- Blat.exe - a console-based email client
- 7z.exe - a utility used for zipping and unzipping jobs
- pscp.exe - putty scp, a windows secure copy (scp) client.
- plink.exe - putty link, a windows secure shell client.
- stunnel.exe - an ssl encryption wrapper used by blat.exe so that we can send gmail emails from the console, using the notification emailer. This executable isn't used during the script but must be installed on the machine that pulls jobs and must be scheduled as a startup job in windows (this is done by putting a shortcut for it in the startup folder).

Additionally, there are several other important variables

- %fromEmailAddress% - the notification email address, emails are sent from this address to people to tell them that their jobs are finished.
- %fromEmailAddressPassword% - the password associated with the notification email address
- %sshUserName% - the username to the pebble account that will be used to copy the files
- %sshPassword% - the password to the user account on pebble used during the script.
- %jobDirectory% - this is the directory where the jobs are stored on the pebble server. Currently /www/dicelab2/submissionsystem/jobs, it may later be /www/dicelab2/jobs.
- %finishedJobsDirectory% - the location on the server where the finished, zipped up jobs with output directories will be stored, currently /www/dicelab2/submssionsystem/finishedjobs.

BATCH FILE OPERATION SUMMARY

The batch script pulls down all the jobs on the server and executes them one at a time. Here is an outline of the process

- Download the queue files in order - queue1.txt first.
- Loop through the zip files inside the queue file, and download them from pebble using pscp.exe
- Unzip the current job zip, change into the unzipped directory, and execute the modeling-building software.
- Zip the results back up, and use pscp.exe to copy it back to the finished jobs directory on pebble.
- Send an email to whoever uploaded the job notifying them that it is ready to be picked up
- Removes the original job submission using plink.exe, leaving only the completed job on the server
- Clear the queue files from the server using plink.exe
- Run plink.exe to delete all files in the finished jobs directory older than 14 days.

WEBSITE – IMPORTANT VARIABLES

The website is composed of JavaScript, HTML and PHP. However, all of the backend work is done by PHP. Much of this work is done by uploadJobCore.php, located in the folder /www/dicelab2/submissionsystem/. This is the file that all the submission data runs through when a user submits a job. It has two purposes: First and foremost, it verifies all of the data that the user inputs. Secondly, it writes all of the configuration data, job data, and mathematical data and then zips it up into a single directory. uploadJobCore.php makes heavy use of utility functions defined in utilityFunctions.php.

At the top of the file there are a series of important variables and constants.

- JOB_NOTIFY_ADDRESS - the email address to send a notification email to.
- SAVE_CONFIG_FILE_NAME - the configuration file save name. Currently the modeling program depends on this file being called Config.cfg.
- SAVE_DATA_FILE_NAME - the data file save name, currently Data.txt
- DATA_FILE_NAME - the name of the data file handle as submitted by the html form, this is defined in the html form displayed on the previous page.
- CONFIG_FILE_NAME - the name of the configuration file handle as submitted by the html form, also defined by the html form on the previous page. These names are used as indexes into the $_FILES array, a global array containing uploaded file information.
- SAVE_JOB_FILE_NAME - the name of the job file, which we use to save the email, used by the batch script.
- FROM_JOB_ADDRESS - the address to send the email from.

WEBSITE – IMPORTANT METHODS

Below, a few of the functions that are critical to the program's exection are detailed

- exitAndUnlink - If any of the tests fail, this function is called, which unlinks all files related to the current job on the server and displays an error message.
- writeJobFile - writes the job file, which currently only contains the email of the user submitting the job and the job name
- verifyModelParameters - takes as arguments integerVars, integerVarIndices, nDv, nCp, nPi, RMSErrorTolerance, maxRMSErrorTolerance, and correlationCoefficient and ensures several things:
    - That all arguments are numeric
    - That nDv, nCp, nPi are all integers.
    - That RMSErrorTolerance is, maxRMSErrorTolerance, and correlationCoefficient are between zero and one.
    - That the size of the integerVars array is less than or equal to nDv+nCp
    - that the size of integerVars is equal to the size of integerVarIndices.
    - that integerVars and integerVarIndices are all integers.
- writeConfigFile - writes the configuration file, according to the user specified parameters.
- addToQueue - adds the job to the queue, deciding which queue by email address.
- writeQueue - actually does the job of writing the job filename to the queue.

The main body of uploadJobCore.php lies below all of these functions. Within this function, there are two main branches – either the user uploaded both a data file and a configuration file, or they just uploaded a data file. In either case, the script looks to see if a job already exists under the specified name in either the finished or unfinished job directories, then it verifies the validity of the files and their contents. A job directory is created, the data, configuration and job files are written to the directory, and then the zip file is written.

There are a couple more PHP scripts that perform simple and quick functions, which are outlined below

- downloadConfigFile.php - when the job is submitted, this script allows the user to download their configuration file that they just submitted.
- download.php - the main script that handles the user downloading their finished job.

WEBSITE FILE STRUCTURE

To end, here is a brief summary of the file structure behind the portion of the website that handles everything related to the modeling software.

- /www/dicelab2/submissionsystem/ - this is the root of the model creation site.
    - dataUpload.html
    - download.php
    - downloadSoftware.html
    - formDisplay.js
    - invalidDownload.html
    - retrieveResults.html
    - uploadJobCore.php
    - uploadJob.php
    - utilityFunctions.php
- /www/dicelab2/submissionsystem/downloads/ - a directory containing downloadable software.
- /www/dicelab2/submissionsystem/finishedJobs/ - a directory containing finished jobs available for download.
- /www/dicelab2/submissionsystem/queue/ - a directory containing the queue files.
- /www/dicelab2/submissionsystem/jobs/ - a directory containing all the job zip files.