Towards a Universal, Quantifiable, and Scalable File Format Converter

Kenton McHenry, Rob Kooper, Peter Bajcsy
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign, Urbana, IL
{*kmchenry,kooper,pbajcsy*}*@ncsa.uiuc.edu*

Abstract. *This paper addresses the problem of designing a universal file format converter. File format conversion is a necessary part of data dissemination and curation. Complete and robust converters however are hard to find and build due to the abundance of file formats, the fact that many formats are closed, and the complexities within individual format specifications. On the other hand many software applications exist that are capable of performing some degree of data conversion between a subset of the available formats. To take advantage of this we introduce a data structure called an I/O-Graph to store the available input and output formats of these applications. Based on a concept of* ***imposed code reuse*** *we use this to develop a service, NCSA Polyglot, which through this graph is capable of performing the larger union of conversions supported by the underlying software. The Polyglot system is designed to be easily extensible, scalable with the number of conversion requests, and inclusive of all available third party software. Given a data set of files from a particular domain, we are able to assign weights to the edges within the I/O-Graph indicating the amount of information retained during a conversion. These edge weights allow the system to then choose conversion paths with the least amount of information loss.*

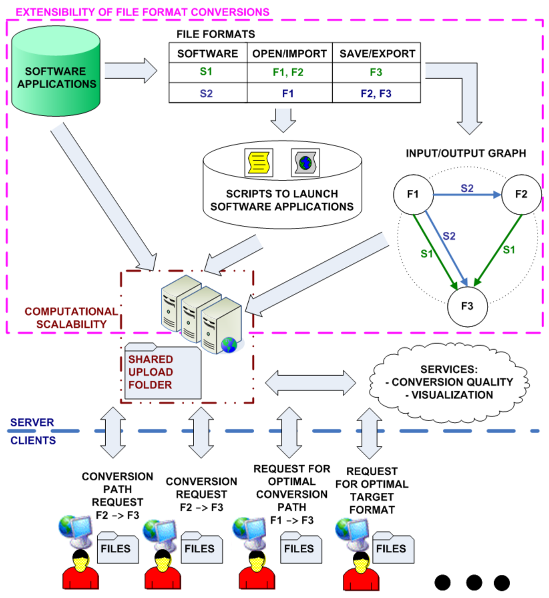
1. Introduction

The ability to effectively convert between file formats is an important concern with regards to digital curation and data dissemination. For every digital file domain (e.g. documents, images, video, etc...) there exists a large number of file formats. Having multiple ways of storing the same data and different software to view that data hinders the ability to distribute it and by that diminishes the lifespan of the data. In order to preserve data beyond the life of a particular format, which is often linked to the life of a particular piece of software, one can convert the file to a format that is: open, standardized, accepted, and independent of vendors.

The main application driver for our work comes from the 3D domain that includes both graphics and CAD file formats. The difficulty of sharing 3D data is fairly infamous [6]. It can pretty safely be stated that nearly every vendor of 3D software has created their own file format. We have counted over 140 different 3D formats supported in some form by the 22 software applications we have looked at. We emphasize "some form of support" as the code to load the file into the program is rarely complete with regards to the formats specification. While the format associated with the vendor of an application often supports many attributes and features such as material properties, texture, and animation this support is not always available and/or implemented for other formats. Often times geometry support is considered sufficient and all other data is dropped. In reality, what is supported by a particular program's file loaders is very much dependent on the vendor and the programmers who actually implemented this functionality.

In addition to concerns over implemented functionality one must also be aware of the fact that not every format supports the same data content. For example not all graphics formats support animation or particle physics effects. What does one do when they convert from a format that does contain a particular feature to one that does not? The easiest solution is to simply drop that part of the data. Conversion of the actual data is sometimes an option, as in the case of moving from a format that supports a boundary representation (B-Rep) of a surface (often used in CAD formats) to a faceted representation (often used in graphics formats). However this is not always an option as in the case of the inverse of this procedure, going from a faceted representation to B-Rep, which is a non-trivial task.

Our objective is to support data sharing, dissemination, and preservation. Thus, our goal is to design and build a system that can convert between as many formats as possible, as robustly as possible, as well as provide additional services for evaluating conversion quality, and visualizing file contents. Our approach to achieve these goals is based on the following two observations. One, though there is no one real all encompassing file format converter, most applications support conversions between some subset of file formats. For example, though there is no one standard 3D file, most 3D applications support wavefront *.obj and/or VRML



EXTENSIBILITY OF FILE FORMAT CONVERSIONS
SOFTWARE APPLICATIONS
FILE FORMATS
SOFTWARE
OPEN/IMPORT
SAVE/EXPORT
S1
F1, F2
F3
S2
F1
F2, F3
SCRIPTS TO LAUNCH SOFTWARE APPLICATIONS
INPUT/OUTPUT GRAPH
F1
F2
F3
S1
S2
COMPUTATIONAL SCALABILITY
SHARED UPLOAD FOLDER
SERVICES:
- CONVERSION QUALITY
- VISUALIZATION
SERVER
CLIENTS
CONVERSION PATH REQUEST F2 -> F3
CONVERSION REQUEST F2 -> F3
REQUEST FOR OPTIMAL CONVERSION PATH F1 -> F3
REQUEST FOR OPTIMAL TARGET FORMAT
FILES

Figure 1: *An overview of the Polyglot conversion system. Third party applications are documented and scripted so as to be used as modules within the overall system. A web interface provides one method of user access for file conversion and visualization.*

*.wrl to allow importing and exporting of 3D data from other applications. Two, though vendors often create their own proprietary closed file format, they do provide support for that files loader within their software. Since it is their proprietary format this loader is likely the best implementation one could hope for. Since they often support imports/exports to and from a handful of other formats we have a means of getting in and out of this proprietary format.

It is under the above observations that we propose the notion of **imposed code reuse**. The benefits of code reuse are well known and accepted. Having access to expertly written and tested file loaders would be an incredible asset in implementing a universal format converter. On the other hand, software vendors rarely have it in their interest to provide the needed access to their code. We define imposed code reuse as the wrapping of 3rd party software, utilizing whatever interfaces the software vendors make available, to provide API like access to embedded code. Based on this we have created NCSA Polyglot, an extensible conversion service which takes advantage of conversions available in 3rd party software. The overall design of NCSA Polyglot illustrated in Figure 1 can be thought of as a compact version of the CORBA-based PRE framework used to create Conversion Central [7, 8]. However here our sole focus is conversion. In order to take maximum advantage of 3rd party software we make no assumptions as to the interfaces provided by 3rd party software. We wish to take advantage of all software, from those that are command line driven to applications that have their functionality locked away behind a graphical user interface (GUI). Our largest contribution within NCSA Polyglot however is in providing a framework for measuring the quality of individual conversions and allowing for the use of this information in the choosing of optimal conversion paths (i.e. paths that will result in the least information loss between formats).

In the next section we describe the design of the conversion system. In Section 2.1 we describe the I/O-Graph data structure which we use to store the formats supported by each application. In Section 2.2 we describe how we perform the conversions stored in this graph in an automated manner and in Section 2.3 we describe how the process can be parallelized. In Section 2.4 we then describe how we measure information loss within the 3D domain so as to choose conversion paths that preserve the most information. In Section 3 we give details on the prototype we have created.

2. Designing the Universal Converter

2.1. I/O-Graphs

We store all conversion information about an application within a data structure called an Input/Output-Graph (see Figure 2). The vertices of this graph represent the union of all input and output file formats supported by various applications. The edges within the graph are directed indicating an application that supports a conversion from source format A to target format B. Parallel edges are allowed and indicate multiple applications capable of performing the same conversion. The I/O-Graph in Figure 2 contains information for 17 applications and is densely packed with edges between the various formats. As mentioned already, direct conversions (i.e those utilizing a single application), are represented by a single edge in the graph. We however are interested in all possible conversions (i.e. those produced by chaining together the direct conversions within each application). This can be accomplished by searching for a shortest path between a given source and target file format (assuming for now that fewer conversions are better in terms of preserving information). A shortest path between *.lwo and *.stp is shown. The path uses Blender to first convert the *.lwo file to a *.3ds file and then converts this file to a *.stp file through Adobe 3D Reviewer.

2.2. Automating Conversion Execution

The I/O-Graph contains all the information required to perform, search for, and chain together conversions between file formats. If we are to use this information

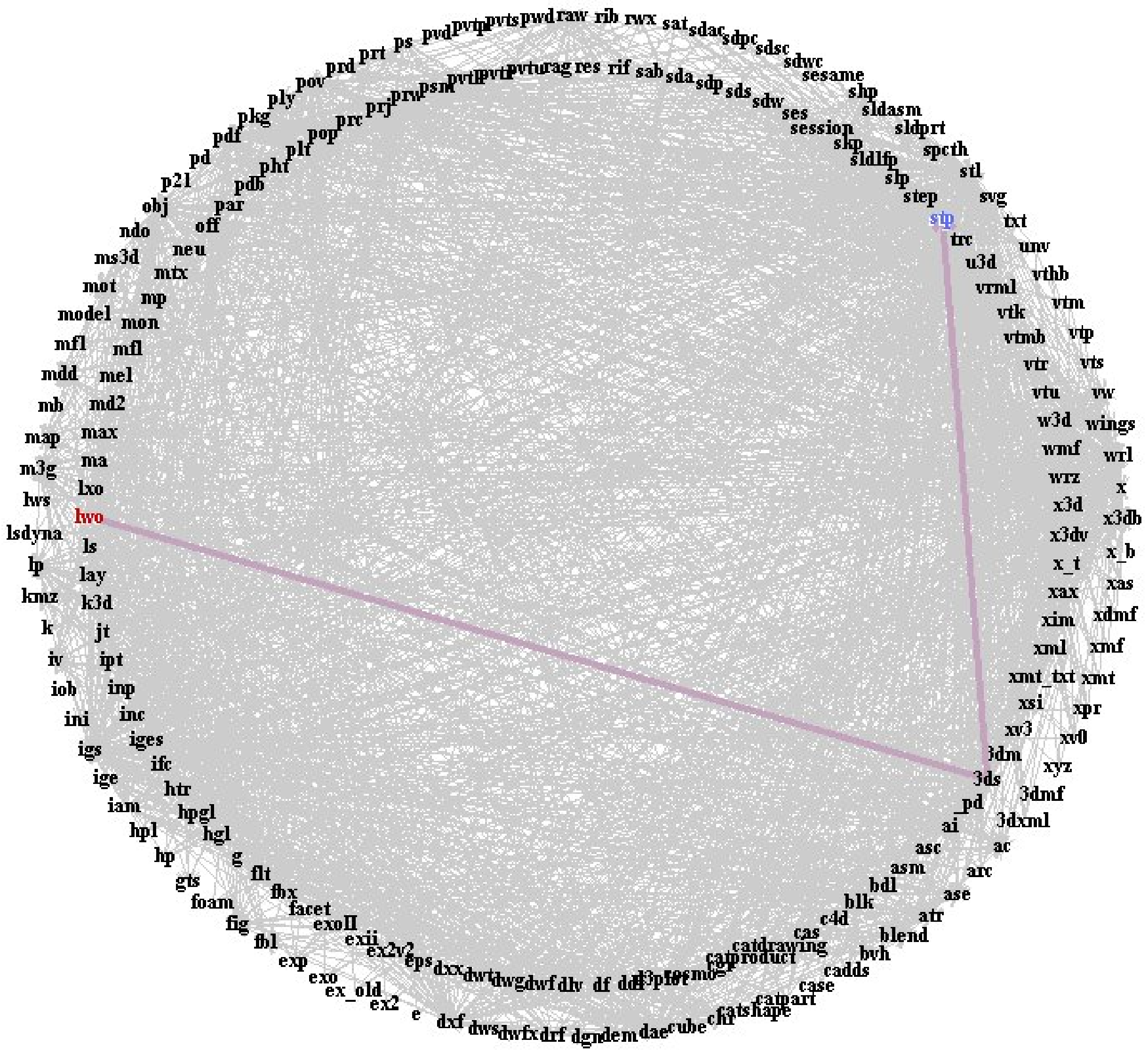


Figure 2: *A visual depiction of an I/O-Graph containing information on the following 3D software applications: 3DS Max, Adobe 3D Reviewer, AutoCAD, Blender, CINEMA 4D, Cyberware PlyTool, IDA-STEP, Inventor, Google SketchUp, K-3D, Lightwave 3D, Maya, NIST VRML/X3D, ParaView, POV-Ray, VTK, and Wings 3D. Supported conversions are represented by directed edges between the vertices representing a particular format. An example chained conversion is shown, found by searching for the shortest path between *.lwo and *.stp. The path found uses Blender to first convert to the *.3ds format, then uses Adobe 3D Reviewer to convert this file into the *.stp format.*

for a conversion system the only question that remains is how to automate the actual conversions. For applications that are command line driven this is trivial enough to accomplish through one of many available scripting languages. For GUI based applications however this isn't as simple. Many modern applications are Windows based with only GUI interfaces. To make our system as flexible as possible we must make no assumptions as to alternative means of accessing the functionality of these applications. Thus, in order to automate the interaction with 3rd party software we chose to make our service Windows based and use the AutoHotKey scripting language. This choice was motivated largely by the need to utilize the many high-end 3D applications available only on the Windows OS which have only a GUI and no other means of interaction. Since most open source software tends to have a Windows port we felt that we could choose a Windows based system without sacrificing access to software predominantly found on Linux machines.

The AutoHotKey scripting language is capable of bypassing the usual message passing that occurs within the Windows OS to communicate between windows (which includes widgets which are instances of windows). Messages involved with GUI interaction usually come from a mouse, however, this need not be the case. If the desired message is known in advance it can be sent within a script at an appropriate time to produce a desired action. To obtain these messages tools such as WinSpectorSpy[1] can be used to capture the message from a sample interaction. We note that not all windows based GUI applications use the Windows API widgets and can instead use a light weight widget of its own. In these cases AutoHotKey allows for the recording of mouse clicks at specified coordinates to perform GUI based tasks. The latter approach is not as robust as care must be taken to ensure the window is sized properly and enough time is given for the GUI to respond to the click. Regardless, the AutoHotKey language can be used to effectively turn a GUI based application into a command line based application.

We chose to make the conversion system web based in a manner similar to other conversion sites[2,3,4]. As we are using scripted GUI based applications the choice seemed natural as a user would likely become annoyed by windows automatically popping up on their desktop and the mouse cursor being taken away from them. In addition, the state of the desktop needs to be known and maintained for these scripts to execute in a robust manner. Hiding everything on a remote server behind a web interface is a convenient way of ensuring this. The system consists of a daemon running on a web server where the needed applications are installed. The daemon simply monitors a set folder for jobs in the form of subfolders containing 3D files and a single text file indicating the job to perform. Files are uploaded to the "jobs" folder via a web based interface utilizing JavaScript and PHP. The job is created by querying the I/O-Graph and storing the resulting path in a text file as lines containing a program, an input, and an output format. The daemon reads this file and performs the action on each line on each of the files in that directory which match the target input. At the end of the operation all files matching the final output format are displayed in the browser for downloading.

The system, called NCSA Polyglot (after one who speaks many languages), is easily extended. To add an additional conversion program one simply needs to add another script to the systems "ahk" folder. The scripts, all following a preset convention, contain all the required information to be used automatically. Each script is named with an alias for the program (e.g. A3DReviewer), an underscore, and the operation which it performs (e.g. open, save, import, export, convert, monitor, exit, or kill). Depending on the operation the script will take 0, 1, or 2 arguments indicating the input/output files. The first four lines of the file comprise a header made up of comments. The first comment indicates the pretty name of the program (e.g. "Adobe 3D Reviewer"). The second line indicates the file domain which the program deals with (e.g. document, image, 3D). The last two lines are dependent on the arguments and contain comma separated lists of valid input/output formats. The rest of the script is left to carry out the desired operation in a manner that is as robust as possible. When a new script is added to the "ahk" folder and the Polyglot daemon is restarted it will examine the headers of each script to reconstruct a new I/O-Graph. When a user uploads files to the system the I/O-Graph will be queried for the job to perform and the corresponding scripts will be executed accordingly. Script operations such as "monitor", "exit", and "kill" take no arguments and are instead used by the Polyglot daemon to make the overall system more robust to errors within the 3rd party applications. We note that these scripts are not very difficult to create and the largest thus far has been only 110 lines long.

In addition to the web interface we have built an API within Java to allow programmers to make conversion

[1]http://www.windows-spy.com
[2]http://www.ps2pdf.com
[3]http://www.zamzar.com
[4]http://media-convert.com

requests to a Polyglot server from within another application. For each request a thread is created that uploads the request, waits for the converted file, and then downloads the resulting file to a specified folder.

2.3. Computational Scalability

As mentioned in the previous section the Polyglot daemon monitors a folder for conversion jobs. One can parallelize the conversion system by making use of a file lock and sharing the folder across multiple machines. One machine can be set up as if it were a stand alone system and share its "jobs" folder. Other machines with installed Polyglot daemons can then be setup to point to this shared folder rather than their own local copy. Since the bottleneck in the system is the actual conversions through 3rd party applications, the cost of distributing files through a shared folder is negligible and thus we would expect the overall computation time to scale as $1/n$ with added CPU's. We could use this same method to parallelize execution on machines with multicore processors as well. Even if the installed 3rd party applications are not designed to take advantage of multiple processors we can effectively run one instance of each application on each core by using the described method on multiple virtual machines installed on the same physical machine.

2.4. Conversion Quality

The quality of a conversion depends on both the formats involved and the implementations of the loaders within the software used for the conversion (Figure 3). As mentioned earlier not all formats support the same information so parts of the actual content is sometimes converted or dropped (see Table 1). Also the entire specification for each file format is often not implemented within every applications file loader, so information is lost in this way as well. In order to choose the optimal conversion path between two formats we really need to have some estimate of the information lost along each path. In other words, we need to assign a weight to each edge in the I/O-Graph that indicates how good the conversion is along that edge.

In order to assign these weights we must compare the 3D content, independent of format, before and after a conversion. Various measures of 3D similarity are available [1], each capturing some aspect of what it means for objects to be similar. We list a few possible measures below and point out that the choice of measure should be dependent on the needs of the conversion system. Note that the measures listed here consider only geometry.

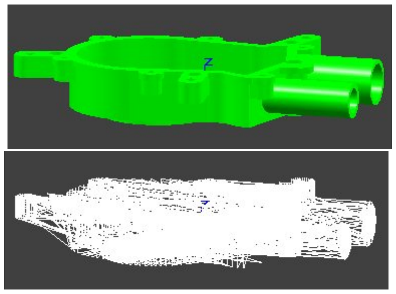


Figure 3: *An example of the kind of noise that can occur during conversions (***top:** *the original model in *.stp format,* **bottom:** *the model after conversion to *.igs and back using Adobe 3D Reviewer and Cyberware's PlyTool). In this example all faces were lost and extra edges were added.*

- **Statistics:** The mean and standard deviation of the vertices within the 3D model are compared by concatenating their values into a 1D feature vector and using the Euclidean distance as a measure of difference between two such points. An alternative would be to use the Mahalanobis distance. Since this measure uses vertex positions it is sensitive to the size of the model and somewhat sensitive to the model's shape. Being simple this measure can be computed fairly quickly.
- **Surface Area [2]:** The sum of all faces within the model. Also sensitive to size and shape of the model this measure is invariant to various transformations on the 3D shape.
- **Spin Images [4]:** This technique records within a 2D histogram the relative positions of the vertices neighboring each vertex. These 2D histograms can be clustered, mapped into a 1D feature vector, and compared using the Euclidean distance between two such points. This measure ignores surfaces and is only concerned with the relative vertex positions within the model. As such this measure is invariant to rigid transformations of the model. This measure is useful for capturing a notion of "high level" similarity between objects (e.g. a chair being more similar to a table than to an airplane). We note that this measure requires $O(n^2)$ calculations per model, where n is the number of vertices, and is thus costly to compute.
- **Light Fields [3]:** This measure compares model silhouettes taken from various viewing angles. Again the resulting 2D images are mapped into 1D feature vectors and compared via the Euclidean distance. Since the model is first rotated into a canonical position based on it's moments, this measure is also invariant to rigid transformations. The measure is fairly efficient to compute and is sensitive to the shape of the convex hull of the object.

Format	Geometry				Appearance				Scene				Animation
	Faceted	Parametric	B-Rep	CSG	Color	Material	Texture	Bump	Groups	Trans.	Lights	Views	
3ds[1]	√	√			√	√	√	√	√	√	√	√	
igs[2]	√	√	√	√	√				√	√			
obj[3]	√	√			√	√	√	√	√				
ply[4]	√				√	√	√	√					
stp[5,6]	√	√	√	√	√				√	√			
wrl[7]	√	√			√	√	√	√	√	√	√	√	√
u3d[8]	√				√		√	√	√	√	√	√	√
x3d[9]	√	√			√	√	√	√	√	√	√	√	√

Table 1: *Content supported by a number of popular 3D file formats (see referenced footnotes for details). Geometry can be represented as a faceted surface consisting of polygonal faces, parametric surfaces such as NURB patches, a boundary represented solid, or a solid represented through constructive solid geometry. The models appearance can be as simple as a color value, based on physical properties of materials such as diffuse and specular components, stored in a texture image, or stored in a bump map as surface normals at various locations. The scene can consist of groups of objects, transformations on those groups, light positions and viewing directions. Some files also support animation which can take numerous forms from key frames to motion captured skeleton rigged models, to simulated physics. One should note that the listed attributes by no means encompasses all possible features stored in 3D files.*

In order to fill in the weights of the I/O-Graph we must be able to compare the 3D content of a file before and after conversion. This entails being able to load every 3D file available (every possible source format and every possible target format). This is clearly not an option, as if we could do this we would not need to use 3rd party software to perform conversions, or for that matter even need a universal converter. We instead estimate edge weights by supporting only a few formats and converting from one such format to all reachable formats and then converting the resulting file back to the original format (again through the various paths available). The result is a quality score for the entire conversion (from source A, to target B, back to A). These scores are assigned to each edge along the path and averaged over the different paths that traverse them. The approach is similar to Q-Learning in the field of Artificial Intelligence. Even if one edge along a path results in a very poor quality conversion and penalizes an otherwise good conversion on that same path, there will likely be other paths that use that edge and the average score should stabilize at some higher value.

3. Experimental Prototype

The Polyglot prototype system has the following applications scripted: Adobe 3D Reviewer, Blender, Cyberware PlyTool, K-3D, NIST VRML/X3D Converter, and VTK. Our experiments utilized a data set consisting of 40 models (20 *.stp models provided by NARA from the TWR-841 data set [5] and 20 *.obj models obtained from the web). Conversions were performed on each of these files from their source format to every reachable target format. If multiple paths existed between formats the shortest paths (possibly multiple) would be chosen. Once at the target format the shortest path back to the source format was traversed (again traversing any equally long parallel paths as well). The resulting scores, for each of the four measures outlined, where tallied and averaged for each edge providing us with the desired edge weights indicating conversion quality. Note that since the scores were based on distances (or difference) and we desire similarity we first flip the values by subtracting each score from the maximum score.

Once the I/O-graph is weighted by these quality measures we can, instead of choosing the shortest path between two formats, use Dijkstra's shortest weighted path algorithm to choose paths that will result in the least amount of information loss. On top of this we can also make several interesting inquiries. First, which edge retained the most information (i.e. which applications input/output pair preserved the most 3D content)? Second, which file format is the "best" to convert to. In the context of preservation we define the "best" format as the one that retains on average the most information when converted to by other formats. This again can be determined by using Dijkstras algorithm on the weighted I/O-Graph.

Given the applications scripted in the Polyglot service and our data set we observe the following. When using light fields as our measure, a measure which focuses on shape, the optimal single conversion is through

[1] http://www.ibrtses.com/opengl/fileformats3d.html
[2] http://www.wotsit.org
[3] http://www.fileformat.info/format/wavefrontobj
[4] http://local.wasp.uwa.edu.au/ pbourke/dataformats
[5] STEP Part 203: Configuration Controlled Design, ISO 10303-203, 1994.
[6] STEP Part 214: Core Data for Automotive Mechanical Design Processes, ISO 10303-214, 2003.
[7] The Virtual Reality Modeling Language (VRML)-Part 1: Function Specification and UTF-8 Encoding ISO/IEC 14772-1, 1997.
[8] Universal 3D File Format, ECMA 363, 2006.
[9] Extensible 3D (X3D) Specification, ISO/IEC 19776-1, 2006.

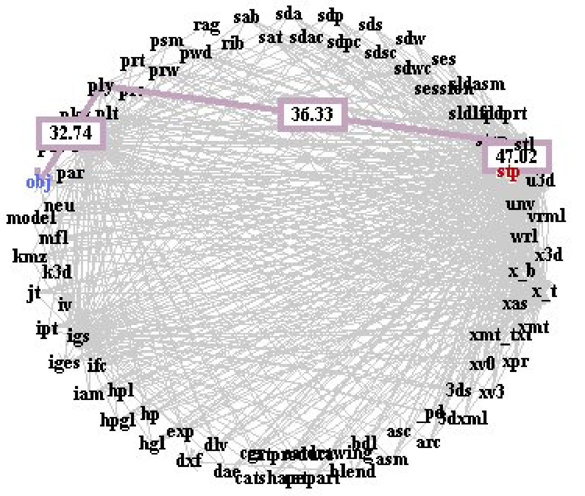


Figure 4: *The I/O-Graph with edge weights assigned from conversions using 40 test models and compared using the light fields descriptor. The highlighted path indicates a conversion path from *.stp to *.obj. The numbers along the edges, ranging from 0 to 100, indicate the average amount of information retention along each edge. The higher the number, the more information preserved.*

Adobe 3D Reviewer between the *.pdf and *.stp format (with a value of 61.67). We note that 3D data stored in *.pdf files are stored internally as *.prc or *.u3d. The format with the on average highest information retention from other formats is the *.stp format with a value of 40.73. Note, these values do not indicate a percentage of information preserved. They are relative values comparable within a particular measure with larger numbers indicating more information being preserved. In contrast, when we use the spin image measure, a measure that focuses on shape through vertices we get an optimal single conversion again through Adobe 3D Reviewer, this time from *.obj to *.pdf (with a value of 59.07) and we get *.stl as the optimal format (with a value of 34.89). The *.stp format being a CAD format has vertices that can change depending on the tessellation resolution used during conversions. Because of this the spin image would encounter more variation in these conversions than it would with conversions between strictly graphics formats such as *.obj and *.stl. Overall, based on the set of conversion applications and data set used, assuming shape preservation was a top concern, we observe that *.stp and *.stl are good formats to convert to.

We have performed preliminary experiments with the scalability of the system. The prototype system consists of three machines, one sharing the Polyglot resource folder and the other two referencing that remote folder and synchronizing via a file lock. Preliminary benchmarks indicate an as expected $1/n$ performance improvement with a test set of 20 step files being converted to every other reachable file format (see Figure 5). The conversion takes a total of 33 minutes 6 seconds with one machine running the Polyglot daemon. With two daemons running in parallel on two different machines the conversion takes 16 minutes 40 seconds (i.e. approximately half the time). With three machines the total time falls to 11 minutes 40 seconds (i.e. approximately a third of the time). Adding further machines to the Polyglot service would require only minimal effort, requiring one to simply install the needed conversion applications on a new machine, install the Polyglot daemon, and change one line in the daemons configuration file so as to point to the shared folder as opposed to its local copy.

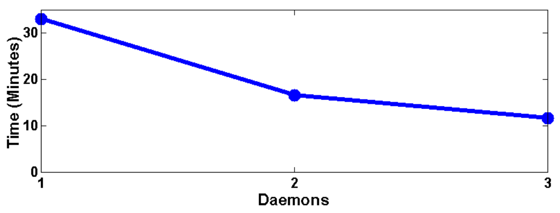
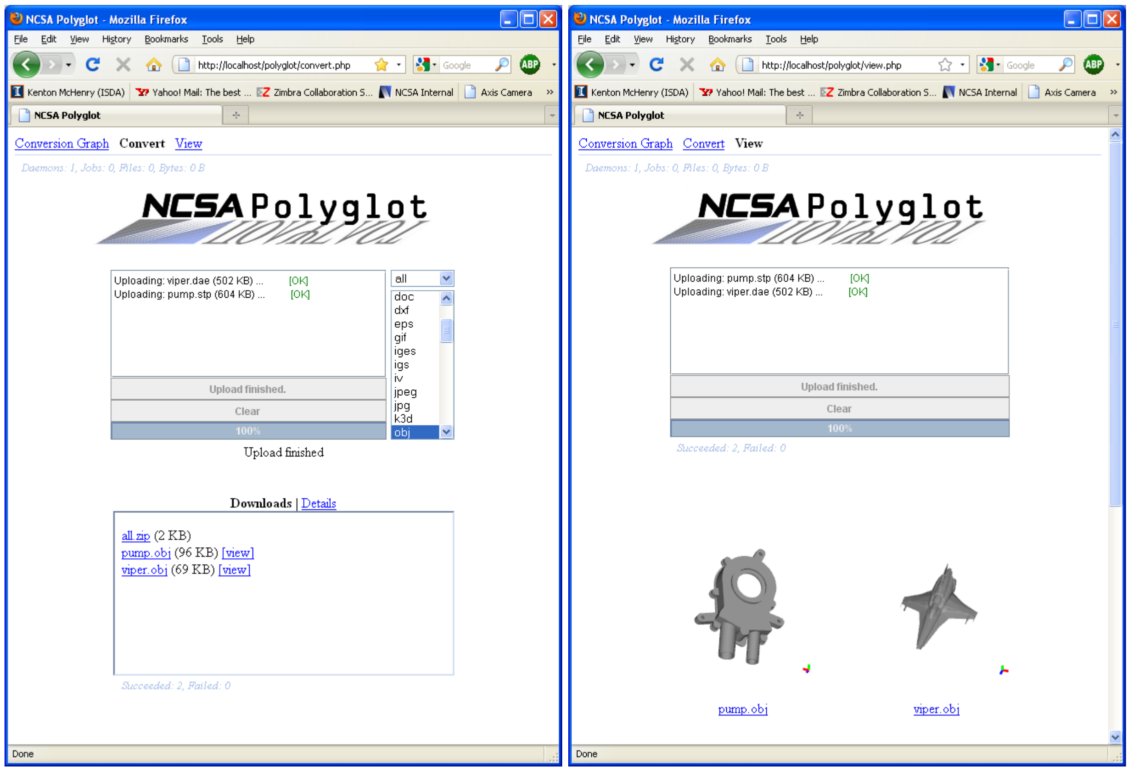


Figure 5: *The number of parallel Polyglot daemons versus the running time to convert 20 *.stp files to every reachable format. The running time drops approximately as $1/n$ with the amount of parallelism.*

4. Conclusions

We have constructed an extensible conversion system which uses imposed code reuse to take advantage of 3rd party applications in order to perform conversions. Information about each 3rd party application is stored in an I/O-Graph which can then be searched for optimal conversion paths. Using AutoHotKey scripting the found conversion path can be executed automatically as needed. Using a data set of sample files and comparing file content before/after conversions, weights are added to the edges of the I/O-Graph indicating the amount of information retained.

Aside from the ability to convert between many formats another useful application of such a potentially “universal” converter is in the form of a “universal” viewer. Given the ability to view one format in each domain one could potentially view them all with such a converter by converting every file to this target format (see Figure 6). The Polyglot web interface only supports the direct viewing of wavefront *.obj 3D files. However it is capable of converting many formats to this format. A user can upload any file to the server and if a conversion path exists to the *.obj format it will automatically convert the file and display the 3D model.


NCSA Polyglot - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/polyglot/convert.php
Kenton McHenry (ISDA) Yahoo! Mail: The best ... Zimbra Collaboration S... NCSA Internal Axis Camera
NCSA Polyglot
Conversion Graph Convert View
NCSA Polyglot
all
doc
dxf
eps
gif
iges
igs
iv
jpeg
jpg
k3d
obj
Clear
100%
Downloads | Details
all.zip (2 KB)
pump.obj (96 KB) [view]
viper.obj (69 KB) [view]
Done
NCSA Polyglot - Mozilla Firefox
http://localhost/polyglot/view.php
Conversion Graph Convert View
NCSA Polyglot
Clear
100%
pump.obj
viper.obj
Done

Daemons: 1, Jobs: 0, Files: 0, Bytes: 0 B

Uploading: viper.dae (502 KB) ... [OK]
Uploading: pump.stp (604 KB) ... [OK]

Upload finished.

Upload finished

Succeeded: 2, Failed: 0

Daemons: 1, Jobs: 0, Files: 0, Bytes: 0 B

Uploading: pump.stp (604 KB) ... [OK]
Uploading: viper.dae (502 KB) ... [OK]

Upload finished.

Succeeded: 2, Failed: 0

Figure 6: *The web interface to the Polyglot service.* **Left:** *Users drag and drop a number of files to the top area, select a target output file in the list, and click "Upload". When the files are converted they appear in the area below.* **Right:** *The web interface to Polyglot's universal viewer. Again users drag files to the top area. This time however there is no choice for the output format as it is automatically set to the type supported for viewing by the web interface (*.obj). When the files are converted they are displayed in the area below where the user can then rotate and zoom in on the objects. In the image above the left model was originally in the STEP (*.stp) format and the right model in the COLLADA (*.dae) format.*

Future work includes adding more software to our Polyglot server. In addition to the scripts in the experiments of the previous section we have scripted: Google SketchUp, ParaView, and 3Ds Max. We also realize that there is likely benefit in having a high level tool for creating the AutoHotKey scripts, as currently it is somewhat like hacking. We are looking into the design implications of creating a graphical tool to capture sample user interactions with other applications and automatically generate clean/robust scripts that can simply be placed into the Polyglot "ahk" folder.

Acknowledgements. This work is supported by a National Archive and Records Administration (NARA) supplement to NSF PACI cooperative agreement CA #SCI-9619019.

References

[1] A. Del Bimbo and P. Pala. Content-based retrieval of 3d models. *ACM Transactions on Multimedia Computer, Communications and Applications*, 2006.

[2] S. Brunnermeier and S. Martin. Interoperability cost analysis of the u.s. automotive supply chain. *RTI International Research Publications*, March 1999.

[3] D. Chen, X. Tian, Y. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. *Eurographics Computer Graphics Forum*, 2003.

[4] A. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1999.

[5] B. Kassel and P. David. Long-term retention of product model data. *Journal of Ship Production*, 2007.

[6] K. McHenry and P. Bajcsy. An overview of 3d data content, file formats and viewers. *Technical Report ISDA08-002*, 2008.

[7] C. Pancerella. The use of agents and objects to integrate virtual enterprises. *SANDIA Report 8226*, 1998.

[8] R. Whiteside, E. Friedman-Hill, and R. Detry. Pre: A framework for enterprise integeration. *SANDIA Report 8505C*, 1998.