Taking Matters into Your Own Hands: Imposing Code Reusability for Universal File Format Conversion

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

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, audio, video, etc...) there exists a large number of file formats. Having multiple ways of storing the same information and different software to view the data representations of information hinders the ability to distribute data files 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 some particular software) one can convert the file to a format that is open, standardized, accepted and independent of vendors.

If we know that we need to convert between file formats then the question becomes how do we do it? Where does the converter from format A to format B come from? This seemingly mundane and trivial problem is in fact anything but that. Consider the 3D file domain, a domain infamous for its numerous file formats. It can pretty safely be stated that nearly every vendor of 3D software has created their own file format to store their data. We have counted over 140 different 3D formats supported in some form by 22 software applications we have looked at [3]. These proprietary formats are often closed which implies that their specification is not publicly available. Without the specification how does one convert in an out of that format? There are two possibilities. One, attempt to reverse-engineer the format and use this to write a converter to/from this format to some other, likely open format. Or two, use the software provided by the format vendors to convert into and out of their format to one of the other formats they support.

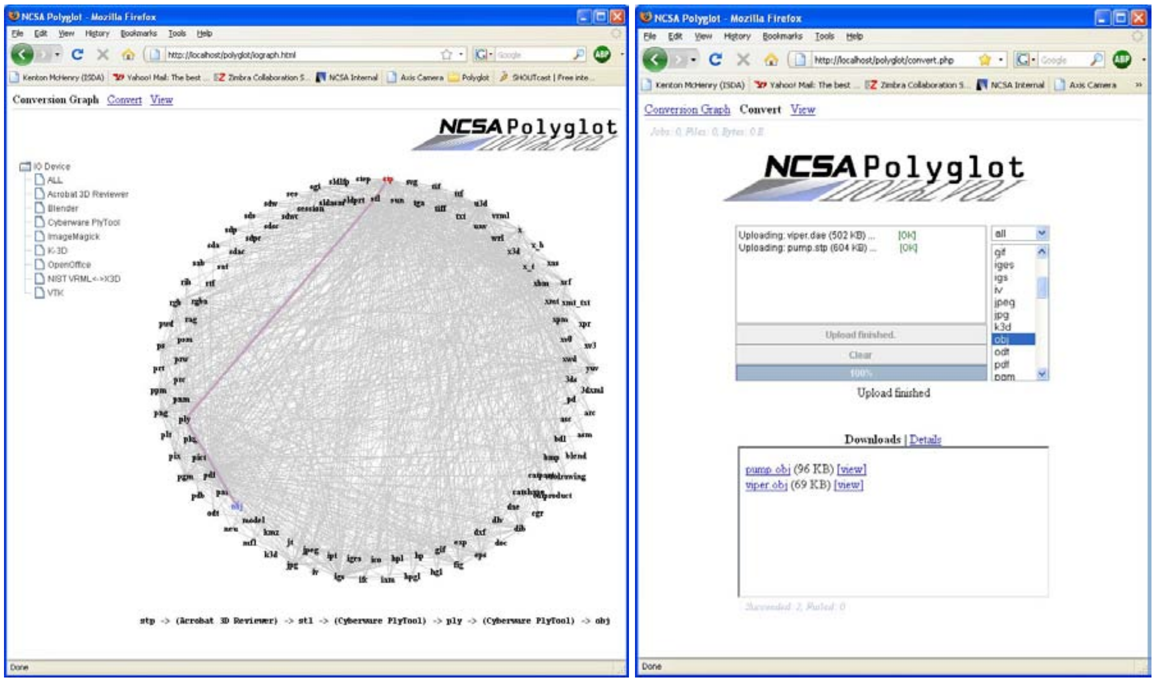
Both options have pitfalls. The former one will likely result in an incomplete specification and thus some content would be dropped through conversions. The latter only allows conversions to formats supported by the vendors' software which often results in dropped data as well. Many formats have extensive specifications (e.g. some of the STEP AP's have specifications well over 2000 pages long). It is often the case that the implementations of the file loaders only support part of the specification (usually geometry in the 3D domain) and simply ignore everything else. However, a very big difference between these two options is that the latter one is practical!

The real question to building a converter is where do the loaders for the particular file formats come from? Who is going to write them? There are many to write, many without specifications, and many with specifications that are large and complex. In particular in the 3D domain, which we are using as an example, there are few if any common libraries to take advantage of. Having format schemas, as in the case of STEP file format in the EXPRESS language, only defers the question. Who is going to write these schemas? The practical side of the matter is that vendors have provided loaders for their closed propriety formats within their software. In addition they have provided partial implementations of loaders for other formats. Thus, a practical solution to the conversion problem should take advantage of the existing code.

We introduce a conversion system called NCSA Polyglot which does just this. We begin by introducing a data structure called an Input/Output-graph. This graph stores as vertices the formats supported as inputs and outputs in a number of software applications. The applications themselves are represented as directed edges within the graph indicating that they are capable of performing a conversion from a source format A to a target format B. Within this graph we have all the information required to perform conversions among the union of the formats supported by the underlying applications. A conversion path between two formats can be found by searching the graph for a shortest path between the vertices representing those formats. To execute the found conversion we turn to the AutoHotKey scripting language. Many of the high end 3D applications we are concerned with are Windows applications driven by a graphic user interface (GUI). The AutoHotKey language allows for the scripting of the messages used within the Windows environment and allows us to effectively turn GUI based applications into command line applications. By imposing a few simple naming and comment conventions on the AutoHotKey scripts we are able to construct an I/O-graph automatically from the scripted GUI based and command line applications. From the I/O-graph and scripts we have created a web based service capable of accepting uploaded files, querying the graph, and executing the appropriate conversions.

As we have argued that many implemented loaders are incomplete it is clear that some conversions will be better than others. We quantify the information lost across conversions in the case of the 3D domain by using a number of measures [1, 2] to compare the objects before and after conversions. Given a data set of sample 3D files we can then assign weights to the edges of the I/O-graph indicating the quality of the conversions that they represent.

From these weights we can then find an overall optimal conversion, an optimal format to convert to, and in the case of specific conversions choose the path of least information loss with algorithms such as Dijkstra's.



```
stp -> (Acrobat 3D Reviewer) -> stl -> (Cyberware PlyTool) -> ply -> (Cyberware PlyTool) -> obj
```

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

Upload finished

pump.obj (96 KB) [view]
viper.obj (69 KB) [view]

Succeeded: 2, Failed: 0

Figure 1: *The web interface to the NCSA Polyglot conversion system.* ***Left:*** *The I/O graph containing vertices and edges for a set of scripted applications. The web interface allows users to check for the existence of conversion paths by clicking source and target formats. A highlighted path is shown for a conversion between the *.stp and *.obj fie formats. The path uses Adobe 3D Reviewer to first convert to the *.stl format, then uses Cyberware's PlyTool to convert to the *.ply format and then to the *.obj format.* ***Right:*** *The main conversion interface to the Polyglot service. Users drag and drop files into the top area, select a target format in the list and click "Upload". Once the files are converted they are displayed in the bottom area for download.*

It is well known that code reuse has its benefits. We have argued that attempting to implement file loaders for the huge number of large, complex, closed formats available is difficult. We have also argued and demonstrated that even if certain pieces of code are locked away behind GUI's they can still can be reused and in fact can lead to a practical conversion system.

References

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

[2] 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.

[3] K. McHenry and P. Bajcsy. *An Overview of 3D Data Content, File Formats and Viewers*, Technical Report ISDA08-02, 2008.