

Innovations Report

A Prototype Digital Asset Management System for Maya Scene Files

Matthew Ovens
BACVA3
2006

Abstract

Digital Asset Management is not usually considered an interesting subject, but it has become an integral part of most companies' workflow. Without it, production pipelines become convoluted and difficult to control, and time and money are wasted. This report looks at what DAM is, why it is useful and how aspects of it can be implemented into a prototype asset management tool designed to work with Maya scene files. Inspiration for the tool was taken from commercial DAM applications, as well as personal opinions on what features would make it useful. The result of this is a tool which can display Maya files as thumbnails and integrates file versioning.

What is Digital Asset Management?

An asset is generally thought of as anything of value owned by an individual or a group. In respect to the world of media production, digital assets can be defined as any digital media files required by a user. In order to manage assets, they must have certain rights associated with them, or have metadata attached in order to more easily identify their contents, but ideally both. A file lacking either of these attributes is not a very useful asset, and the greater the number of files in a project the more important this information becomes. Without it, files become difficult to manage and hard to find.

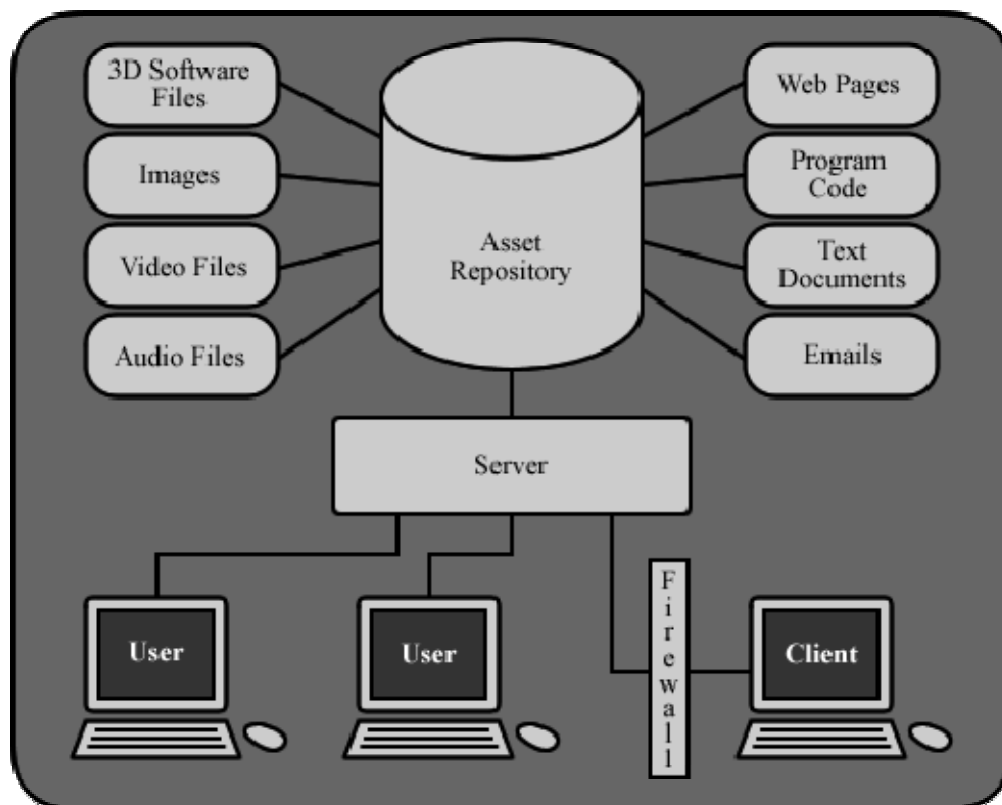


Figure 1 *Example of a theoretical DAM system setup*

In its simplest form, Digital Asset Management might consist of merely organising your files with logical naming and placing them in obvious locations. However, this

may not be enough to track their development and ensure that they are not lost or overwritten accidentally. Using a DAM system can take a lot of the work out of trying to maintain a useable set of assets, and is now common practice in many industries.

Features typically associated with asset management software

File organisation

When working on a project, it is essential to know exactly where required files are. If files are not able to be found quickly, it can result in a much slower workflow – something that should be avoided in virtually any industry. Attaching metadata to files enables advanced search abilities, speeding up asset location and helping to identify them. Such information might include the author of a file, the date it was created and last modified, a description of its contents, etc. The loss of assets through careless file management is also a potential danger and can bring a production pipeline to a halt, but it can happen easily when dealing with large numbers of files. This is almost never a problem when using a reasonably proficient asset management system.

Version control

Versioning is an important technique for maintaining a history of changes made to a file or collection of files. Most conventional version control methods are ideally suited to text files, as it is possible to store the differences between different versions rather than a new copy of the entire file. This is much more difficult when dealing with binary files, such as images. For example, a simple tint operation will result in every pixel's colour value changing, so the new version of the file is entirely different to the old in the eyes of a computer. Using version control shows users which is the latest revision of a file and allows them to revert back to an older edition of a file when necessary. One benefit of this is that if an asset is judged to have developed in an undesirable direction, a previous version can be used as a basis for redevelopment. This is a much faster solution than either starting again or trying to reverse engineer the asset.

Version comparison

A user will often wish to see the differences between alternative versions of a file. This can be a slow process if there are many versions and each one must be opened to determine how it differs from another. There is also the possibility of having to open the versions multiple times because the user is unable to keep the differences in mind during what could be a lengthy comparison process, slowing assessment further. A visual side by side comparison of file versions is a much simpler method to use, and allows a user to identify a version they are looking for much more quickly.

File logs and annotation

File logs can help users by informing them of what has changed between versions of a file, who last worked on it, the date and time it was last worked on, and more. It can aid in ascertaining the correct version of a file to use, or can permit a supervisor to leave notes on changes they would like to see in a file. At the end of an asset's development it can also be used to mark it as a final version.

User access control

The larger a team is, and the greater the number of people working on it, the more important it becomes to allocate access to assets accordingly. Members of a team working on a project should have access to the assets for that project, but whilst others might be authorized to view them, they should not be able to make changes to those assets.

Thumbnailing and file preview

Digital Asset Management systems can benefit greatly from thumbnail views, especially those that deal with multimedia files. It enables assets to be identified by a user far more easily, and makes sense in a situation in which most of the files being handled are of a visual nature.

Why is it useful?

When dealing with any project larger than a few files, or being worked upon by more than one user, managing the assets for that project becomes a necessity. Knowing exactly which version of a file to use in can prevent costly mistakes and make a production pipeline flow much more smoothly. In their book on implementing DAM systems, Jenson *et al.* (2005, p.25) state that the five key benefits expected as a result of asset management are:

1. Higher productivity
2. Organisational scalability
3. Increased reliability
4. Better quality
5. Greater flexibility

Increased productivity can be achieved by reducing the time taken to perform simple filing tasks, giving content creators more time to do actual work. Implementation of a DAM system will support the ability of a company to expand by providing a structured workflow that can maintain as many projects and users as required. The reliability of a business to meet the expectations and deadlines of its customers improves, as monitoring the status of the elements of a project becomes far easier. Quality increases as a result of more time to carry out work, and because the occurrence of costly errors and losses is decreased considerably. Owing to the ability to keep track of all aspects of a project, if changes are suddenly required they can be implemented very quickly. These features are applicable whatever the size of the group using the system, whether it is a small team working on a single project or a large company working on many projects simultaneously.

Case study: A typical second year group project on the BACVA course

A second year group project consisting of five members starts with each member deciding upon the roles they will fulfil. The student whose story they will be realising puts themselves forward as director, and the other group members are happy to let them have the job. As well as director, they also decide to take on some of the

modelling tasks, along with another student. Two of the three remaining members choose to be animators on this project, and the last member agrees to do concept work along with the lighting. They all agree that the remaining aspects of the project will be shared out equally amongst them.

The student who agreed to produce the concept drawings gets to work, receiving input from the group during their bi-weekly meetings and during the day whilst they are working together. The art is created in sketch books and on sheets of loose paper, and is all kept by the artist. After a few days of discussion and thumbnailing, they get started on creating a first draft of the storyboard. Over the next couple of weeks, the group go through several drafts of the storyboard, and the design work is revised. Three weeks into the project they have final designs for the main character and half of the props in the animation, and the modellers can start work.

A few days later, the group decide they have some scenes in their storyboard that they will definitely carry through to the final version. The animators in the group feel that this would be a good time to begin making an animatic, as they do not have much to do at the moment. They scan the storyboard panels that they require and begin to edit clips together. Meanwhile, the concept artist is still producing designs for the environment and revising the storyboard.

Five weeks into the project, the group have a final version of their storyboard. The new panels are scanned in and edited together to complete the animatic. The animators have also started a blocktest based on the animatic they had so far. However, no one has realised that one of the clips in the animatic has been made from panels that have since been updated. The animators continue with the blocktest, but it now contains sequences that are incorrect.

Two weeks after this, a problem occurs. In an effort to tidy up the project directories, a member of the group accidentally overwrites a scene file containing a large portion of the environment with a much earlier version. As there were no intermediate versions, the models have to be built up from this basic scene. However, the person that modelled it can not remember the design. The concept artist attempts to find the design sheet, but believes that it was given to one of the modellers a while ago. The design must be redrawn before the model can be rebuilt, and this takes an extra week of work in total. In the meantime, the animators have begun animation, but they are still using the incorrect sequence included in the blocktest.

After ten weeks of working, the project directories are filled with obscurely named model and animation files. There has been some attempt to create a logical directory structure, but files have slipped into the wrong folders. The lack of a consistent naming convention means that they can not be easily identified. For example a file called “mainCharacterWithShoes_02.ma” has been found in a directory outside of the main character’s directory, but inside the directory, there is a file named “mainCharacterWearingJacket_07.ma”. Without opening the file, there is no way to tell which of these files is the latest version. Eventually, they discover that the file outside the proper directory is the most up to date, despite the fact that it has an earlier creation date. Apparently, the other model was imported into it to incorporate its elements into the main file.

As the project deadline nears, the group begin rendering scenes. The file names have still not been organised properly, and renders regularly have to be redone because the

wrong file has been used. Additionally, the members of the group forget which files they have rendered and end up rendering several files more than once. This is a huge waste of time, and the group end up having to stay up all night several times in the week before the deadline in order to get the work done.

Eventually, the deadline arrives. The group has had to work up until the last hour to produce their animation, and they feel slightly disappointed that it is not as polished as they had hoped it would be. If they had had a structured way of maintaining all their files and designs, they would have had a lot more time.

An example of an existing DAM system

Avid Alienbrain

Alienbrain is a well renowned production asset management system. It provides sophisticated versioning functionality for all file types, as well as the operations users expect in revision control software. Some of its features include: checking in and out of files, user access control, attaching metadata to assets, file annotation and version history viewing. The last two are especially clever when dealing with image files, which are commonly viewed as being difficult to maintain in this regard. Images may have notes left on them in diagrammatic form to point out exactly what the writer is focusing on. For example, a comment about the size or shape of an element in the image may be accompanied by a superimposed line drawing displaying clearly which aspect is being mentioned and/or what the amendments should be. Viewing the version history of image files is also more advanced than with other systems, in that it allows the side by side visual comparison of different revisions of a file. This is the only practical solution to version comparison in this situation, as traditional line difference comparison is only applicable to text files.

All of Alienbrain's features are accessible through its client, which is very visual and allows thumbnailing and previewing of assets. This applies to files from asset creation applications too, for instance Adobe Photoshop and Maya. With Maya (and other supported 3D packages), thumbnails are created using a plug-in which lets a user take a screen shot of the file to act as both thumbnail and preview image. The application plug-ins also allow you to work directly with the Alienbrain repository, and provide much of the functionality of the client.

Because of its huge feature set and suitability for working with a wide range of media formats, Alienbrain is used by digital content creators around the world. However, it is not inexpensive, and is therefore often an unfeasible option for small studios. Clients include companies such as Electronic Arts, Sony Pictures Imageworks and Lionhead.

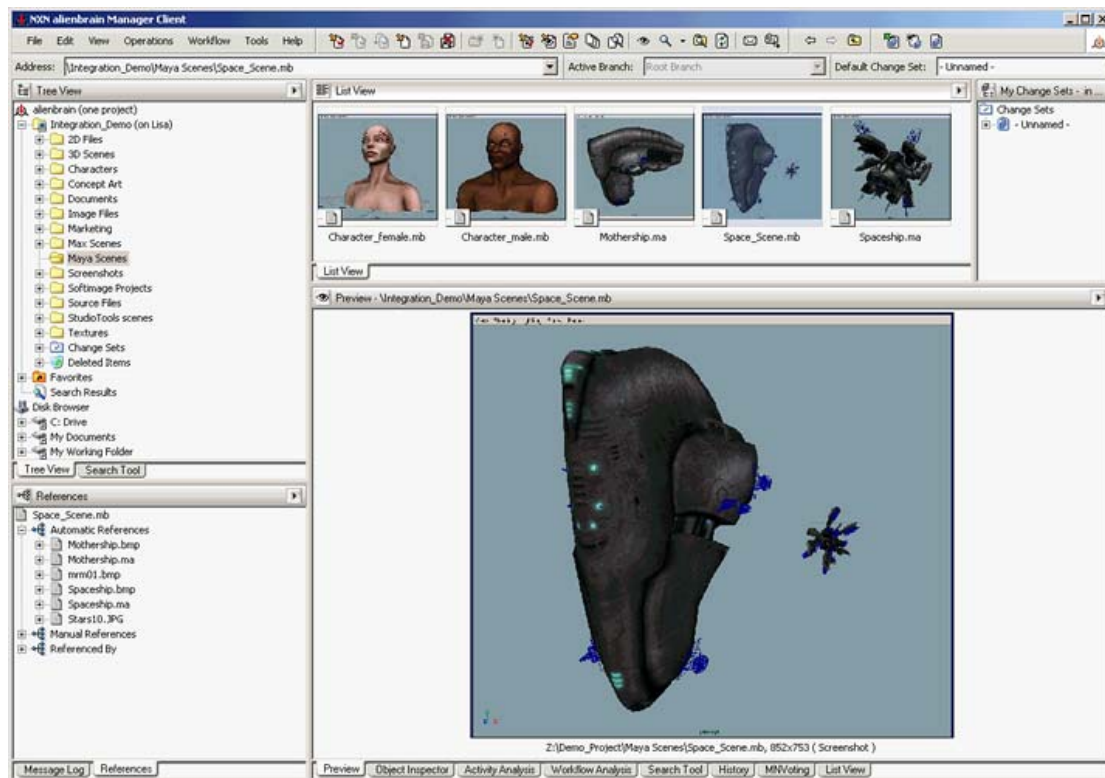


Figure 2 Alienbrain's user interface displays thumbnails and previews of Maya files created from screen captures (image from Alienbrain User Guide [2005])

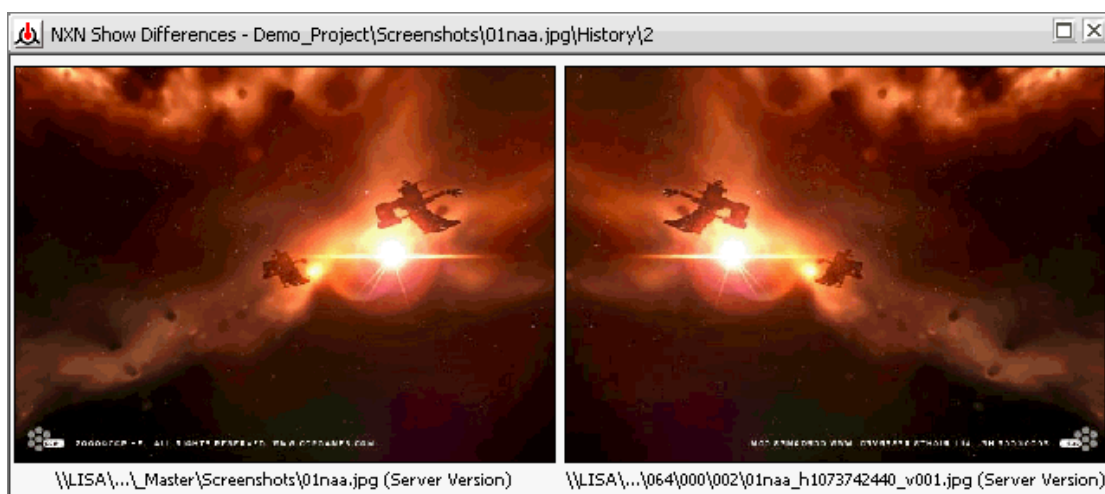


Figure 3 Different versions of a file can be placed side by side for visual comparison (image from Alienbrain User Guide [2005])

Features to implement in the tool

Of the many features available in DAM systems, the ones that are intended to be implemented in the prototype are file versioning, checking in and out, version comparison and thumbnail file viewing. These should be attainable within the scope of this project and result in a useful tool.

How it was approached and the problems encountered

A versioning system

RCS - Using “diff” to find differences between versions proved impractical

As the focus of this project is on asset management, and a large part of that is based on file versioning, it was necessary to furnish the application with a versioning method. The scope of the project did not afford the time to develop such a mechanism from scratch, so the decision was made to use one already available. Open source programs were therefore an ideal resource, as most could be used freely under the GNU license.

The Concurrent Versions System is a widely used version control package. Primarily designed for collaborative software development, its system is based on handling text files. The versioned files are stored on a server from which users may check them out to a client machine as a “working” file, either read-only (e.g. for checking) or editable. Once changes have been made to the file, it is checked back in to the server as a new version. The user checking it in updates the log file to describe the changes made, and the version number of the file is incremented. CVS also allows multiple users to check out the same file and work on it independently. When both altered versions of the file are checked back in, they may be merged to combine both sets of changes. Of course, this only works with text files, where line by line comparison is possible. If a merge fails, a user must manually decide which version to store.

Subversion is a more advanced system based upon the CVS architecture. It has more efficient handling of binary files and a failsafe method for checking in, ensuring that files are not damaged if the check in process is interrupted. It can also version entire directory trees, and maintain their history if they are moved or copied, something CVS is unable to do.

Both of these applications are built on top of the Revision Control System. RCS has the basic functionality of checking files in and out, maintaining a version history by storing the differences between versions (when dealing with text files) and keeping a log for each revision. However, it is generally viewed as being tailored more to individual users as it does not require a server/client setup, while still allowing multiple user access. Since it has the core operations necessary to a versioning system, and as the commands are relatively simple to use, RCS seemed an appropriate choice to use in the prototype.

It also has an inbuilt method for checking the differences between versions of text files using the *diff* command. This returns the disparities with the line numbers they occur on. To begin with, this presented itself as a useful way to find the variations between versions of Maya ASCII (.ma) files. As they contain text in the form of Maya’s MEL code, the *diff* tool would work on them. It was thought that these differences could then be applied to an earlier version of the file in Maya to bring it up to date, and so show the changes between the revisions in Maya itself. This would have been ideal, as a user could explore the scene to get a more complete understanding of the differences. Unfortunately though, the nature of the MEL held in .ma files is such that the differences between versions do not always contain the context in which they are applied, i.e. changes made to an object in a scene will be returned as a difference, but the object to which they are applied to may not as it could have been created in a previous version. There is also the problem that operations

applied to objects in the scene are inserted after a command selecting that object. This means that many of the lines in a file will not contain both a command and the element in the scene it is being applied to.

Version 1

```
...
createNode transform -n "pCubel";
createNode mesh -n "pCubeShapel" -p "pCubel";
    setAttr -k off ".v";
    setAttr ".vir" yes;
    setAttr ".vif" yes;
    setAttr ".uvst[0].uvsn" -type "string" "map1";
    setAttr ".cuvs" -type "string" "map1";
    setAttr ".dcc" -type "string" "Ambient+Diffuse";
    setAttr ".covm[0]" 0 1 1;
    setAttr ".cdvm[0]" 0 1 1;
...
```

Version 2

```
...
createNode transform -n "pCubel";
    setAttr ".t" -type "double3" -0.822 0 0 ;
createNode mesh -n "pCubeShapel" -p "pCubel";
    setAttr -k off ".v";
    setAttr ".vir" yes;
    setAttr ".vif" yes;
    setAttr ".uvst[0].uvsn" -type "string" "map1";
    setAttr ".cuvs" -type "string" "map1";
    setAttr ".dcc" -type "string" "Ambient+Diffuse";
    setAttr ".covm[0]" 0 1 1;
    setAttr ".cdvm[0]" 0 1 1;
createNode transform -n "pPlanel";
    setAttr ".t" -type "double3" 0 0 -1.2 ;
createNode mesh -n "pPlaneShapel" -p "pPlanel";
    setAttr -k off ".v";
    setAttr ".vir" yes;
    setAttr ".vif" yes;
    setAttr ".uvst[0].uvsn" -type "string" "map1";
    setAttr ".cuvs" -type "string" "map1";
    setAttr ".dcc" -type "string" "Ambient+Diffuse";
    setAttr ".covm[0]" 0 1 1;
    setAttr ".cdvm[0]" 0 1 1;
...
```

Using *diff* on the two versions will return the highlighted lines – those that are different in each version. The new polygonal plane in version 2 is returned as a complete set of MEL commands for its creation, and could be recreated using this code. However, the change in the polygonal cube's translation values is returned as a single line without any reference to the cube itself. Executed on its own, this command would not be applied to the cube, but rather to the currently selected object.

Figure 4 *Example of the problem with using diff to return the differences between two versions of a Maya ASCII file*

Maya command line rendering – a slow process

To create file thumbnails required a method of producing a visual record of the contents of each scene. The most obvious way to do this was to render each file. Since only a small image was required for each file, the render times could be very small. The simplest way to go about this task was to use Maya's command line renderer. By passing it rendering options and the name of a file, it would create the necessary images. However, because it can only render a single file at a time, the render command must be called once for each file to be thumbnailed. This is inefficient, as when it is called it must start a new instance of Maya from which to render. The loading of Maya takes time in itself, and when the time to load plug-ins and each scene are also taken into consideration; multiple calls to the command line renderer become a relatively slow solution to the problem.

C++ - application was difficult to use; not visual enough

Initially, it was decided to develop the project as a C++ application. Integrating the RCS and command line rendering was achieved using the *popen()* function, which forks the process and creates a pipe to enable commands to be sent the shell for execution. After developing a prototype that worked from the command line, the problem of a user friendly interface needed to be addressed. Whilst useable, the application was not intuitive or very appealing from a visual point of view, and in light of this it was felt that a graphical user interface would be beneficial to the system. Therefore, Qt Designer was investigated as a means of doing this.

How the approach changed to overcome the problems*Changed from C++ to MEL*

As the project is based around managing Maya scene files, and it is possible to create user interfaces with MEL, the decision was made to translate the application into this medium. This would allow tighter integration with Maya, and give access to the higher level functionality built into the scripting language. It was also felt that creating the user interface would be a faster and simpler process than attempting to learn to create one for a C++ application, and would have the advantage of not being limited to one platform, as the majority of MEL commands are the same regardless of which operating system Maya is being used with. One disadvantage to coding in MEL rather than C++ is that it is a scripting language, and therefore interpreted. This means that the script is run through an intermediary program (Maya) which translates it into code which can then be executed by the computer processor. While this can lead to slower execution, the benefits it provides were felt to be enough to justify the move.

The *popen()* command also exists within MEL, so this could still be used. It also has the advantage of working under the Windows operating system too, whereas the C++ command is not supported. As well as this, MEL provides a *system()* function that allows single lines of instructions to be sent to the shell for execution. This is useful when it is not necessary to send further directions to the process.

```
string $result = system( "co /usr/projects/file.ma" );
```

The `system()` function returns the result of the command it executes as a string. This is valuable if you wish to check whether a command has executed properly, or require the output generated by the command inside your MEL code.

Example 1 *Using the `system()` command*

Used `popen()` and “maya -prompt” to pipe file rendering for faster results

Due to the inefficiency of the command line rendering method, other techniques were looked for to perform the job of thumbnailing. One solution which presented itself was to use the `popen()` function to open a pipe to a command prompt version of Maya.

This loaded an instance of Maya without the GUI in which each file could then be opened and rendered, rather than loading and unloading Maya for every render required. Additionally, the `-noAutoloadPlugins` flag was used to speed up the load time by preventing plug-ins from being initialised. Because the `popen()` function creates a pipe to the process it initiates, commands may be sent to the process as though it were a file using `fwrite` until the pipe is closed using the `pclose()` function. Using this method meant that the MEL render commands could be used and render settings within the scenes could be accessed, giving a greater level of control to the process. This provided a substantial time saving to the thumbnailing procedure.

```
int $pipe = popen( "maya -prompt -noAutoloadPlugins", "w" );
```

The returned integer is a file descriptor to the pipe created. A pipe is way of directing the output of one program into the input of another, and is only able to send information one way. In this case, `popen` has created a pipe to a command prompt instance of Maya. The “w” indicates that the pipe can be written to, but using an “r” would mean that it could be read from. Sending instructions to the piped process would be done using `fwrite`, e.g.

```
fwrite $pipe "file -open \"/usr/projects/file.ma\";";
```

This line will write the command in quotes to the pipe `$pipe`, opening the specified file in the prompt version of Maya. Note that the quotes within the command must be escaped using the forward slash, else the string would terminate prematurely. If the pipe is created using “r”, a command such as `fgetline` could be used to read the result of the command executed using `popen`.

When using `popen`, it is important to remember to close the pipe once you have finished with it. It may also be necessary to send instructions for the spawned process to quit. With regards to the Maya process created in this example this would be done in the following way:

```
fwrite $pipe "quit -force;";  
pclose( $pipe );
```

Example 2 *Using the `popen()` command with a pipe*

A problem encountered with this technique occurred when the render globals in the file were set to render animation. In this instance, if software rendering was selected whilst the tool attempted to create thumbnails it would cause the frame range specified in the file to be rendered, as opposed to a single image. To prevent this from happening, the render globals in the file were changed to disable the rendering of more than one frame.

fwrite \$pipe "setAttr defaultRenderGlobals.animation 0;";

This command is sent to the piped instance of Maya prior to the render command to ensure only the current frame is rendered.

Example 3 *Setting the animation render globals*

Changed to rendered version comparison

After further deliberation on the challenge of comparing file versions, the possibility of writing a custom *diff* function was explored. However, it was deemed to be beyond the scope of the project, and so alternatives were taken into consideration. The most logical of these was to create renders of the different versions for visual comparison by a user. This could then utilize the thumbnailing method already developed and allow a user to see the different revisions of a file side by side. Whilst this may not provide as complete a picture of the changes that have taken place as the *diff* method would have, when used in conjunction with the log information it should be sufficient for a user to identify the version they require. The only caveat is that, as with creation of the regular thumbnails, the default perspective camera must be in an appropriate position in the scene in order to give a useful overview of the contents. If every version of a file has the camera in a different location, it may make the comparison slightly harder for the user.

The prototype

What it does and how to use it

The final product from this project is a tool which creates and displays Maya file thumbnails, and allows the files to be versioned using RCS. Users can navigate to a directory and set up the thumbnail and RCS directories if they do not exist. Scenes in the directory may be opened, checked in and out, have version logs displayed, have their versions compared and have their thumbnail refreshed using either hardware or software rendering in Maya.

To begin with, a user runs the MEL script to define the necessary functions and display the interface. If this is the first time they have used the tool, they may wish to view the user manual in the *About* tab. Using the *Browse* button, they navigate to a directory containing Maya scene files either in the directory itself, or in an RCS subdirectory. If they have not been to this folder with the tool, they will most likely be told that there is not a directory in which to store the thumbnails. In this case, they use the *Setup Directories* button to create the required folders. Once this has been done, hitting the *Update* or *Refresh Thumbnails* buttons will start the thumbnailing process. If the folders already exist, the tool will proceed to create thumbnails for all the files in the selected directory and the RCS directory. Preference is given to working versions of RCS files, so thumbnails will reflect the contents of checked out files if they exist. In the event that a user has already created thumbnails for files in this directory before, the previous thumbnails will be displayed rather than new ones rendered out upon each visit. This is designed to prevent unnecessary re-rendering of files, which would slow the system down, but new thumbnails may be generated using the *Refresh Thumbnails* button (for all thumbnails) and the individual *Refresh*

option in each thumbnail menu. By default, the tool will use Maya's hardware renderer to create thumbnails. This is in an effort to keep the thumbnailing process as fast as possible. Users do have the option to use software rendering instead though. The choice can be made using the radio buttons above the file view, and the selection applies to both refresh options.

Subsequent to the rendering being completed, each thumbnail is placed into the file view along with a label indicating the name of the file and whether it has been versioned, denoted by the prefix "RCS:" (Fig. 1.5). Each thumbnail also has a popup menu attached to it which is generated according to the state of a file at the moment the user clicks the right mouse button. If the file is only present in the selected directory, i.e. there is no versioned counterpart, the only options available to a user are to open the file using the *Open* option, check it into the RCS directory and refresh the thumbnail. Similarly, if the file is versioned the menu will allow the user to check any version of the file out as either read-only or editable, display the version log and render thumbnails of every version of that file for visual comparison. Additionally, a user may open the file once it has been checked out, clean the file from the directory if it is read-only, and can see if the file has been locked. Locking is indicated using (*l*) and occurs when the file is checked out as editable. A locked version of a file can not be checked out by another user in anything other than read-only mode. If it becomes necessary to break a lock on a file, the user may also do this. Breaking a lock may be required if a user accidentally checks out more than one version for editing, or if another user requires immediate access to an editable version of the file.

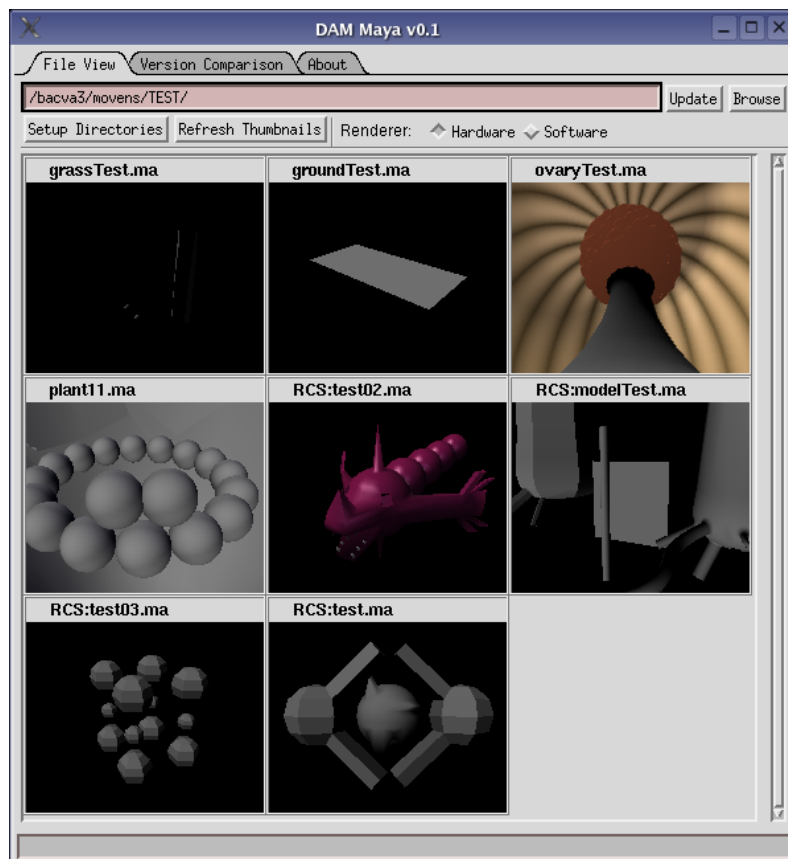


Figure 1.5 The file viewer. This shows labelled thumbnails of Maya files in the selected directory

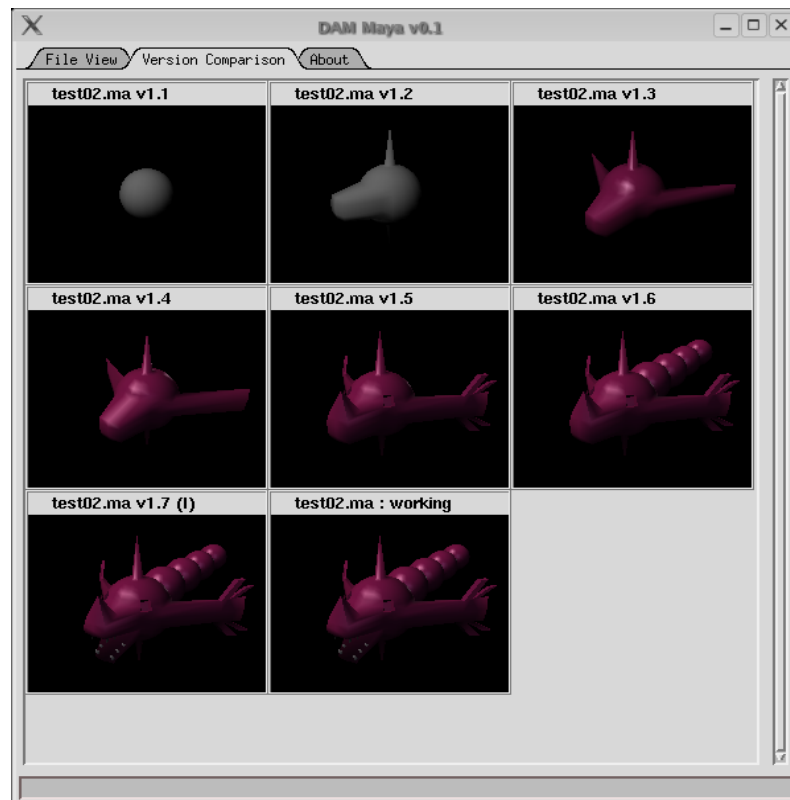


Figure 1.6 The version comparison view. File revisions are shown chronologically

When a user chooses to check a file into the RCS directory, a window appears in which they can enter either the file description or the log message for that version, depending on whether or not this is the first time the file has been checked in. If the file has not changed since it was last checked in, the log message is disregarded and the file is removed from the working directory (the parent directory of the RCS folder). The log can be viewed using the *Info* option in the menu. This brings up a window containing details of the log messages from all previous revisions, the revision number of the latest version, the authors of various revisions, dates and times each revision was deposited and more. If the file was checked out as read-only, the *Check In* option is replaced with a *Clean* option. This removes the file from the working directory.

When dealing with versioned files, the *Compare Versions* option will appear in the menu. This will render out thumbnails of all the versions of that file, including the working file if there is one. It does this by temporarily renaming the working file if there is one to protect it from being overwritten, after which it checks out each revision of the file and renders it. The images are placed sequentially from earliest to latest in the *Compare Versions* tab (Fig. 1.4). The file name and revision number are displayed above each image, and as with the *Check Out* option, locked versions are displayed using (l). The working file is identified with the suffix “: working”.

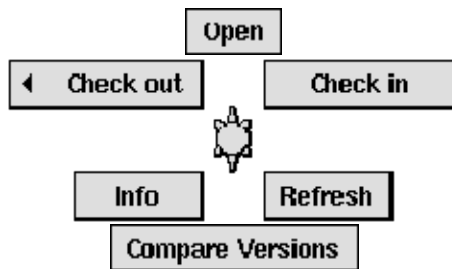


Figure 1.7 Menu for a checked out editable file

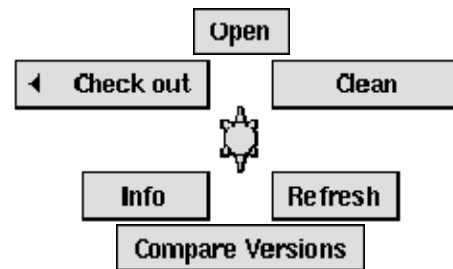


Figure 1.8 Menu for a checked out read-only file

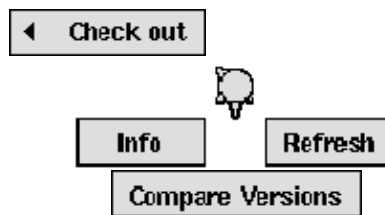


Figure 1.9 Menu for a file that has not been checked out

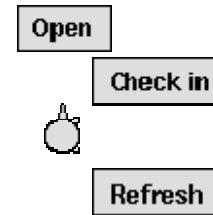


Figure 2.0 Menu for a file that is not versioned, but is present in the selected directory

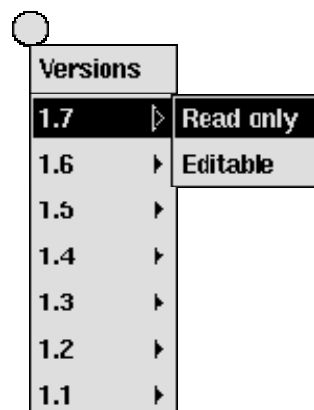


Figure 2.1 Menu allowing a user to check out any version of a file

Changes and additions for future development

Though the application works, and might be found useful by some, there are features which could be developed to add functionality and improve performance.

Make the tool compatible with the Windows OS

The tool is currently only for use with Maya on UNIX based operating systems, and will not function under Windows. This is because of the different ways in which their file systems work. A Windows specific version of the tool could be created to remedy this, or OS checking could be done in the current code to adapt the tool to whichever system it is being run on.

Perform thumbnailing work in the background, and make it progressive

With the present setup, even though the rendering is performed in a separate instance of Maya, the tool must wait for the renders to finish before it can create the thumbnail view of each file. If the rendering process could be done completely in the background, and the thumbnails could be added to the browser once they have been produced, it would make it much more interactive. Making the rendering progressive would also help enhance interactivity by displaying a lower resolution image initially, which may be enough for a user to identify the file, but increasing the resolution in a background process.

Include options to adjust the render setting, and include other available renderers

Giving users the power to change certain settings for the thumbnail renders might help in situations where identifying a file requires being able to see shadows, post effects or other features that are currently disabled to speed up the process. Some files may also necessitate the use of other renderers. For example, a file in which Renderman shaders have been used would benefit from being thumbnailed using a Renderman compliant renderer.

Handle other file types: obj (and other importable formats), images, scripts, etc.

Currently, files are opened in the piped instance of Maya and then rendered to create the thumbnails. However, it would be relatively easy to create thumbnails for .obj files (or any other file format Maya can import). By checking the extension, the script could decide whether to open the file or import it to perform the render. It could also be made to handle image file thumbnails and script previews, making a more complete asset management tool.

Render flick books of files, specifying a frame range according to the first/last frames of animation in the file.

As well as displaying still images of files, it would be useful to be able to render a “flick book” of a file to be able to preview animations. The start and end frames could be specified by the user, or could be retrieved from the file itself. This would be a useful feature because a thumbnail of a file containing animation may not be enough to identify it, or to distinguish it from other versions of the file.

Proper file annotation

In addition to version logs to keep a note of what each version contains, it would be advantageous to have file annotation. This would permit users to leave additional notes detailing any changes that should be made to it, whether it is a final version, etc. It could also let other users to leave comments for those working on a file, allowing a supervisor to comment on the work being done, for example.

Allow users to render a larger image of a file for previewing, and to specify the size of the thumbnails.

Despite being a quick and practical way to identify files, the thumbnails may not always provide enough visual detail for a user due to their small size. In this situation, the ability to inspect a larger preview image of a scene would be a helpful option. It

might also be useful to be able to stipulate the size of the thumbnails created; if you have files which contain a lot of fine detail, larger thumbnails could make the identification of those files much easier.

More control over the camera thumbnails are rendered from.

At present, the system will only generate thumbnails from the default perspective camera. It is possible, however, that a user may wish to generate the thumbnail from another camera in the scene. In this situation, the ability to select the thumbnailing camera from a list of those available in the file could prove useful. The list of cameras would probably have to be stored outside the file, as opening each file to find which cameras exist would be a very intensive operation. Alternatively, if dealing with Maya ASCII files, the MEL code in the file could be searched to provide the names of the cameras, which would be a much faster method.

Full file browser capabilities, allowing files to be moved/copied/deleted

As well as being able to view file thumbnails, it might also be useful to be able to perform the regular file related commands, such as moving, copying, deleting and renaming. Whilst such file maintenance can be performed from within the operating system, incorporating this practicality could help to streamline workflow by removing the need to change between the two environments.

Enable batch rendering of a file from the browser.

If a scene has been marked as a final version, a user may wish to have the option to batch render the file from the thumbnail browser. This would provide the features of a command line render, but could present a more user friendly method of invoking it by showing the options in a clearer interface than the command line. It could also include options to perform rendering on remote machines.

Setting user access lists for RCS versioned files.

Files versioned using RCS can have a user access list defined for them. This means that only specified users have the ability to make updates to the file, but still allows others to check out a read-only copy. Tying this into the system would be helpful if a user is part of a larger team working on a project.

Change to a server based versioning system, such as Subversion.

Changing the versioning to a system like Subversion would allow files to be stored on a central server, which is a more practical setup in a multi-user environment. It would also provide more efficient handling of binary files, something which RCS is not ideal for.

Allow users to send messages to people on a file access list to notify them of changes, etc.

When a user wishes to convey details of the changes they have made to a file, or modifications they feel are necessary, they could send a message to the appropriate member on the access list for that file. The process could also be automated to send messages whenever a new version of a file they may have worked on is checked in.

This would help to keep users up to date without the need for them to verify the status of files manually.

Custom “diff” command to enable more thorough version comparison of Maya ASCII files (could convert Maya binary files first).

Despite the fact that it was outside the scope of this project, the custom *diff* command for use with Maya ASCII files would still be a useful feature in the system. It would provide a way to show exactly where the changes have been made between versions of a file, without relying on a user being able to spot the differences in a visual comparison.

Conclusions and results

As companies become larger and projects become even more complex, Digital Asset Management can only take an even more important role in ensuring a smooth production pipeline. Whilst the tool created in this project is very basic compared to commercial DAM systems, it still has useful features and might enable a project to be managed more efficiently, even if only a personal project.

To test the tool, it was given to a few people familiar with Maya.

“I found the system fairly easy to get to grips with, once it had been explained the first time. The interface is quite simple, but it seems to allow you to do a lot.”

“This would have been very useful for our group project last year. Managing files was a nightmare, and the thumbnail view would have really helped.”

“I think it takes a bit too long to make the thumbnails, but once they are there everything works fine.”

As a prototype, the tool fulfils the intentions laid out at the beginning of the project. The project has introduced me to the theory behind DAM systems, and has taught me how important they are to industry.

References/Bibliography

JACOBSEN, J., SCHLENKER, T., AND EDWARDS, L., 2005. *Implementing a Digital Asset Management System for Animation, Computer Games, and Web Development*, Oxford: Focal Press

TICHY, W.F., 1991. *RCS—A System for Version Control*, RCS manual

BELL, S., 2006. *Writing Your Innovations Report*
available from <http://ncca.bournemouth.ac.uk/sbell/notes>

AVID TECHNOLOGY, 2000-5. www.alienbrain.com :
website for Alienbrain® DAM software.

RIGHT HEMISPHERE, 1999-2006. www.righthemisphere.com :

website for Deep Exploration™ DAM software

PERFORCE, 1996-2006. www.perforce.com :
website for Perforce Software Configuration Management software

PICDAR, 2000-5. www.picdar.com :
website for Picdar's Media Mogul® DAM software

SENO SOFTWARE, 2000-5. www.senosoft.com :
website for SENO Software's P3dO Explorer photography and 3D viewer file
management software

WIKIMEDIA FOUNDATION, INC., 2001-6. en.wikipedia.org :
online encyclopaedia
entry for Perforce: <http://en.wikipedia.org/wiki/Perforce>
entry for CVS: http://en.wikipedia.org/wiki/Concurrent_Versions_System
entry for RCS: http://en.wikipedia.org/wiki/Revision_Control_System

Acknowledgements

Thanks go to Ari Sarafopolous, Eike Anderson, and Karl Erlandsen for help with various aspects of C++ and to Jimmi Gravesen for help with MEL scripting.