

Ba Computer Visualisation & Animation, Year 3

Innovations in Computer Animation

Report

Student Name: Matthew Smith

Abstract Animation Generator

Aim

My aim was to present the ‘essence’ of a Maya scene. I wanted to capture its important elements and present them in a ‘broad brush stroke’ way, that was original. The tool should enable an iterative and/or experimental working process, and should be intuitive enough to be comfortably used by a user with no 3D or programming knowledge.

The reason that my Innovations project is innovative for me is because I have thoroughly researched the area in which I am working, and I have not found anything that is similar to my work.

Research

I started my project by researching what has already been done. My proposal stated that I would research “existing systems for procedural camera movements... digital cubism and camera motion metaphors based on film grammar.” I have researched procedural camera movement systems and will explain how they have informed my work. I researched digital cubism to a point which allowed me to decide that I was not going to use it. (This decision is similar to one I made in my Major project, which consists of many small parts joined together. I had considered using methods similar to digital cubism, or montage, etc to join these parts together. But I decided that I didn’t like it for the reason that images were becoming too cluttered and dense with layers and visual information. It is for this same reason that I decided not to research digital cubism further in my Innovations project.)

I researched camera motion metaphors based on film grammar. However, this was actually a step backward, as I didn’t discover anything that I hadn’t already learnt in

Cinemaography lessons. For example, I found nothing no new camera motion metaphors, in the area of procedural camera movements, because all the systems that are out there at the moment, seek to imitate established filmic conventions. Therefore, although my work certainly does not establish any new conventions, by its lack of adherence to existing principles, it is already slightly innovative.

Therefore, my research consists of procedural camera systems. I will go into detail about four particular papers I researched. However, I would like to first give the reader an overview of the range of research completed in the area of virtual cameras and their control systems at the present time, as I think it is important to illustrate why my work is innovative:

Conics-enhanced vision approach for easy and low-cost 3D tracking

Tracking elliptical projections of a planar circle in a sequence of 2D images to determine its translation and rotation, and using this to control camera movement.

Paracatadioptric camera calibration using lines

“Catadioptric sensors refer to the combination of lens-based devices and reflective surfaces... Sensors combine a parabolic shaped mirror and a camera, inducing an orthographic projection, which is called paracatadioptric.”

Speed control in path motion

Controlling object motion along a curve using a velocity curve and lookup tables based on parameter.

A Camera Motion Metaphor Based on Film Grammar

“context driven camera motion... a camera operator oriented interface which incorporates camera motion rules derived from accepted conventions in filming practice.”

Digital Cubism

As an alternative to split screen, a way of showing “simultaneous action happening in different places” by merging images together. Visual interpolation.

A tutorial on developing a computer-controlled camera system

An in-house computer-controlled camera, as an alternative way of getting images, instead of using a commercial art firm. Note: 1986.

CINEMA: A System for Procedural Camera Movements

“Camera placement... Then enquire information directly about the 3D world through which the camera is moving... With this information, procedures are developed that correspond to natural camera specifications. Plus, methods for overcoming deficiencies of the procedural approach.”

CamDroid: A System for Implementing Intelligent Camera Control

“encapsulates camera tasks into modules... Camera modules can be programmed and sequenced, and thus can be used as the underlying framework for controlling the virtual camera in widely disparate types of graphical environments.”

The reason I have covered these, albeit briefly, is to show the kinds of areas that research has previously been aimed at. As far as creating a system which queries what is already in the scene, which is what I am interested in, all the research then heads in a direction which tries to mimic filmic conventions, or, requires clearly defined user parameters that specify exactly what the user wants the end result to be.

The reason that my project is innovative is that, while it does query information from the virtual world, it then uses this data in a more fluid way. By this I mean that the raw data, for example, the world position of objects, is only the starting point. The user parameters which are then set are, in turn, not exact. They take the queried data and manipulate it using filmically unconventional techniques. For example, dynamic

forces such as turbulence. I have not been able to find a similar amalgamation of techniques used for such a purpose. Therefore, I believe my work to be innovative.

I will now go into more detail as to the research I carried out. One of the papers I chose to read thoroughly was: `CINEMA: A System for Procedural Camera Movements`. This from the *Special Issue of Computer Graphics, Proceedings of the 1992 Symposium on Interactive 3D Graphics, Pages: 62-70. By Steven M. Drucker, Tinsley A. Gaylean, David Zeltzer [1992]*. The reason I chose to look in detail at this paper was that it generated camera motion by a combination of two sources. The first is an enquiry into a database of world objects and events. The second is cinematic notions. I am more interested in the first. As I mentioned before, one of the ways my project is innovative is that it does not combine the enquired data with filmic ideals, but abstract ones.

In the paper's abstract, an important point is made:

“In addition to the basic commands for camera placement, a key attribute of the CINEMA system is the ability to inquire information directly about the 3D world through which the camera is moving.”

This is the first thing I took from this paper. Up until I read this I had always intended to make a very random abstract animation generator. By random I mean that I was more interested in the idea of iterative experimentation. Similar to taking a video camera and placing it somewhere without looking through the viewfinder, and pressing record. Seeing what events the camera recorded, what it had ended up looking at, when you hadn't predetermined any of these factors. This was the way of working I had intended to emulate.

Initial Attempts

So I did some initial tests. These were my first attempts. In them I set up four cameras, each perpendicular to the next, and moved them along a motion path. This had been my initial solution. I had thought about taking an arbitrary Maya scene, running a script that would build motion paths around the objects, by first querying their positions. It would then attach cameras to these motion paths, render out

specific frames, (eg, 10, 20, 30). The user would then stop. Adjust the motion paths using an interface with sliders, that would maybe rebuild the curves, adjust their degree, curvature, etc. The user would then be able to render out the same frames again. Adjust the curves again, render out the same frames again...etc.

The idea I had was to end up with a collection of frames of the scene from slightly changing viewpoints. I was then going to write a Shake script to overlay these separate positions and to dissolve between sequential frames in each sequence. The result I was aiming for was a ghostly, ethereal animation of the scene from different places at the same time, shifting and blending.

The idea I had in my head was something like stop-motion animation, except the scene would stay still, the camera would be moved instead.

The user would be able to change the parameters of the curve movement, camera attributes, Shake script, etc. And then run again. So it would be an iterative process.

Development

There were several reasons why I decided not to use this approach. Firstly, as you can see from the images, the layering to produce a ghostly image of different angles of a scene at the same time didn't really work. There were nice moments when the video was played, when the geometry and colour came together in momentary combinations. But most of the time, it was just a mess.

Secondly, I didn't like the idea of using curves to control the motion of the cameras. I think the reason I thought of this in the first places was simply because I had already done a bit of work with curves and MEL scripting for my Major project at the time. Instead of curves, I thought about using particle systems. This had many benefits over curves, even soft-body curves. Firstly, using particle systems gives you control over both the individual elements in the system and over the system as a whole. I thought about how this could be put to good use by making large, sweeping changes to all the cameras in a scene using dynamic forces such as turbulence. I could then change their motion as one, but still have precise control over any irregularities or

‘noise’ in the particle system, if I wanted. Particle systems also, of course, give you more axes of movement than along/around a curve.

Also, particle expressions are optimised. Working in MEL, with a potentially heavy scene, this would help to speed up the execution time and ensure that the user would instantly be able to see the effects of their iterative changes. Finally, I wanted to apply changes to the world position data that were a) filmically and cinematographically unconventional, and b) could be applied to the scene as a whole, as well as to individual elements in it. For this reason, particle systems are good because of the wide range of dynamic forces and fields that are available in Maya, all of which produce effects that are more akin to physics-based effects animation than the normal way a camera, real or virtual, usually moves.

By using particle systems to control the camera motion, I can let the camera motion be controlled independent from the world with unconventional, dynamics-based changes.

Another advantage of using Maya dynamics to control camera movements is that the camera movements can be as smooth or as ‘noisy’ as the user wants. For example, by adjusting the turbulence field. This allows the user to achieve the smooth camera motion that previous camera control systems have strived for, with the use of Bezier curves as motion paths, or alternatively jerky, ‘hand-held’ movements.

Aspects of my initial attempt that I chose to keep include the fact that it should be able to work with any given Maya scene, and that it is still an iterative process.

I also kept the Shake script idea, as this would help me achieve my aim of making the tool fluid and interactive to use, by making the compositing part very user-friendly.

Switching to using particle systems was the first major change in direction for my project. The next came when, after consultation with my tutor, I realised that my animation should not be completely random. Indeed, it was important that my script generated animations that, although abstract, had some sense of progression. Not necessarily in the filmic sense, I decided. It was at this point that I decided to look at image colour and rhythmic editing as ways to achieve this progression.

But why is this the case? Why is it that you can take a camera and point and click and know that, every once in a while, you'll get an image or a video which you really really like. Yet in a Maya scene, my initial random camera movements hardly ever produced worthwhile results. I thought about this, and I think it is simply because of the way that 3D scenes are constructed. When an environment is modelled in Maya, for example, the modeller, even if they don't know the exact camera positions, will usually have an idea which parts are going to be looked at, in the foreground, most important to carry the meaning of the shot, etc. (Indeed, most 3D environments are built for a purpose; the world around us is never built for the photographer or cinematographer's ideas. That's why studios and film sets exist.) So, if you then introduce a random camera movement system into that scene, as I did, the results will not be the same as in reality because there are big gaps in the Maya scene. Not just spatially, where there is no geometry because the camera never looks there, but gaps in the 'reality' of the scene, the atmosphere, the presence. Unless the scene is really really good and expansive and detailed. (Which most of the time it isn't, because it will be broken down into smaller scenes for convenience and to aid workflow.)

It is for this reason I decided that a completely random, 'see what happens' approach would not produce good results the vast majority of the time, and that querying the scene would be necessary. As in the paper.

One of the advantages of this more informed approach, ahead of a completely random one, is that it saves the user time. If the camera is going to be pointing at empty parts of the scene 70% of the time, with a simple, sparse Maya scene, then the user is going to have to carry out many more iterations before they get the result they want. Also, not all users are going to be interested in the iterative element of the process; they may want results faster and with less interaction and repetitions. My script has to be easy for the user to operate, whatever they want to get out of using it.

One of the points that is made quite early on in this paper is the need for a system that works in different domains. On existing techniques for interactively specifying camera movement:

"Each of these techniques has provided an interface for solving a problem for a particular domain, but all of them have remained independent making it impossible to use them across domains."

By domains, the paper mentions “keyframe based computer graphic animation techniques, navigation of virtual environments, general 3D interaction, automatic presentation and synthetic visual narratives.”

I think this is an important point, and it is one that influenced later versions of my script. In my first versions, the script deleted everything in the user’s scene that it wasn’t going to use. In later versions, existing objects, particle systems, fluids, dynamics and animation is kept in the scene. This means that my script works on any kind of Maya scene, whether it is an environment, a character model, a landscape, building, dynamics systems, heavy, light, etc.

The paper also mentions another paper that I think is representative of the general tendency of research to strive for cinematic values: *Karp, P., S.Feiner. (1990) Issues in the Automated Generation of Animated Presentations. Graphics Interface '90 (Halifax, Nova Scotia). 39-48.* “Their work emphasises using a knowledge based system for automatically selecting camera placement and for choosing appropriate camera movements based on cinematic considerations.” Again, a database of objects and events is used together with cinematic knowledge to create a narrative.

An important point is made in the paper:

“Many descriptions of how to film, frame and navigate the scene (by both screenwriter and layperson) are with respect to the objects in the world. For example one might ask for the camera to move alongside object A while looking at object B.”

I think this is important because it highlights why it is so important not to just have a completely random system of camera movement, as I had originally intended. The movement of cameras in film and the direction in which they point is surely always defined by what the director or cinematographer wants to show, and the camera will then point to and focus at that point. Any other kind of camera movement is, I think, only used for a specific effect. This may seem obvious, but I think it is important to analyse this, as it is the difference between making a camera system that is at still at least partially recognisable to viewers, and one which is unrelated. By making such an unrelated system, I feel I would have disconnected the audience from the camera to quite a large extent.

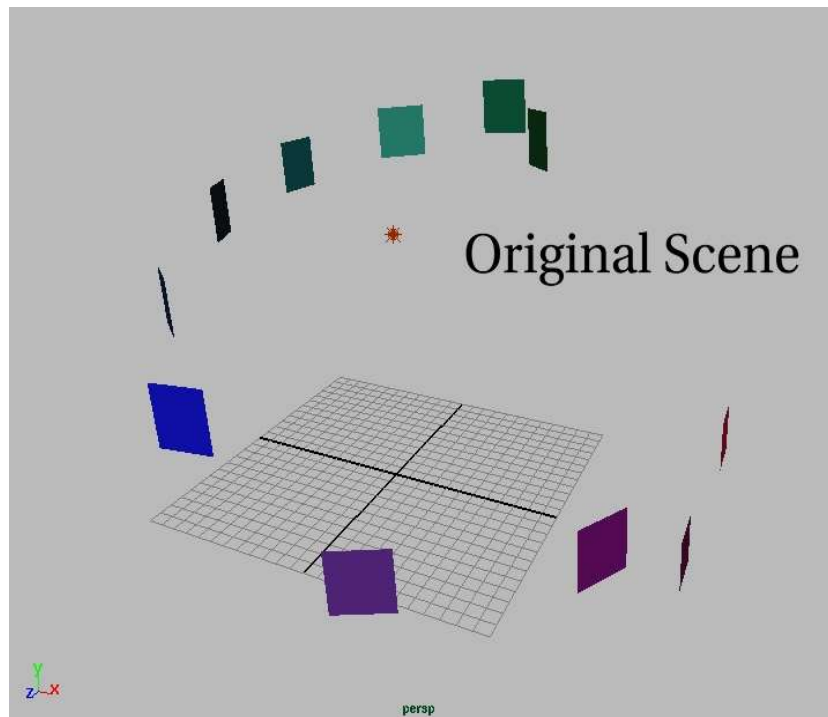
According to the paper's authors, one of the problems with the CINEMA system when the paper was written was that there was no way to combine the current procedure with the ability for the camera to avoid certain objects, for example, without having to create a new, separate procedure. Further work was required:

“By specifying the camera’s relationship to other objects via weighted constraints, the system can find the best position that satisfies certain criterion. These constraints are maintained as the objects, and the camera moves throughout the environment.”

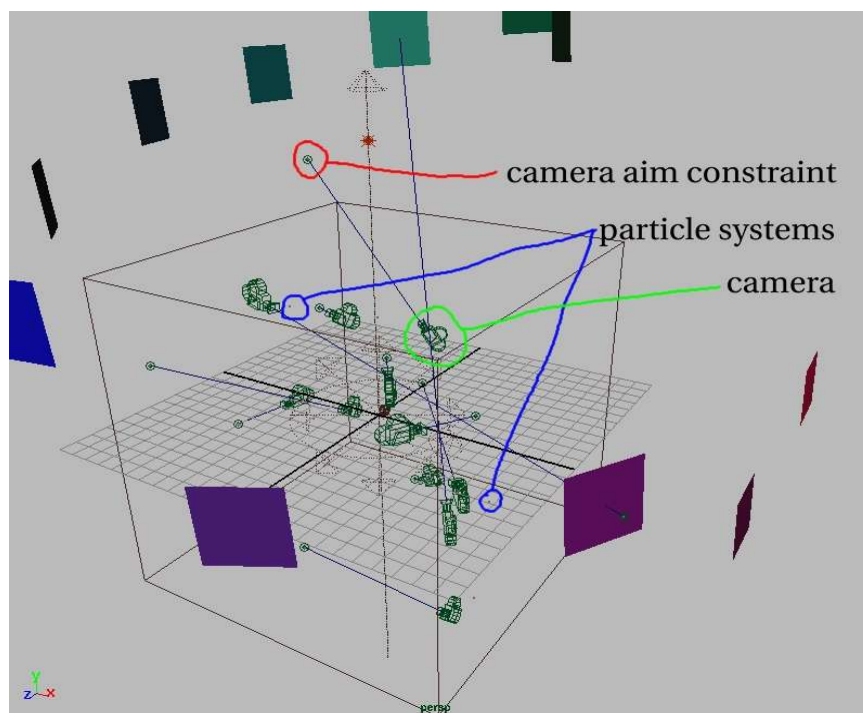
This was perhaps the biggest influence this paper has had on my script. Since I had already considered using a particle system in Maya, combining this with constraints was a logical decision. So I decided to use camera aim constraints and particle goals.

Overview of my script

So, the approach I finally decided to use was having the position of the cameras controlled by one particle system, 'particle1'. A second particle system, 'particle2', is used to drive the positions of the second goal objects of the cameras, which are called aimSpheres. A third particle system sets up the camera aim objects themselves, which are called cameraAimObjects. These are the objects that drive the camera aim constraints, and are therefore where the camera points. I then created particle systems with just one particle in them for each of the objects that were already in the user’s scene. For the constraints that I mentioned, I used particle goals. The strength of these is controlled with expressions that allow the oscillation of the camera aim constraints between the objects themselves and the second goal objects that are controlled by the second particle system.



Before



After

Challenges

One of the challenges I faced was one that occurred because I wanted to make the script as user-friendly as possible. Ideally, I wanted the user to be able to click just one button, and the whole rendering process would take place automatically. The rendering process would consist of batch rendering from one camera, then the next, then the next, until there were no cameras left to render from. Then, the rendered sequences of images should be composited together in Shake, automatically. This was quite an important part of the project simply because the creative process was supposed to be iterative, and therefore as much user time and effort needed to be saved as was possible. Secondly, there were likely to be several cameras in a complex Maya scene, because there would be several objects. This would mean several rendered sequences as well, and therefore several layers in Shake.

It would have been easier and more efficient to batch render from the command line. But I wanted to keep the script as user-friendly as possible, especially for people who had no programming knowledge. So I decided to keep the rendering process within Maya.

The first problem I encountered was that the ‘batchrender’ command in MEL does not return anything. Therefore, there is no way to find out when a particular batch render has finished, in order to start a new one. Which results in each render just stopping the previous one. I tried using Pre-Render and Post-Render MEL scripts, however, these did not allow me to start a new batch render from within themselves. Another problem with using Pre-Render and Post-Render MEL scripts was that querying and changing attributes of objects in the scene is of no use either. I found out that this was because when you start a batch render, Maya saves another version of the scene you are batch rendering, and renders from that one. Therefore, the queries I was making were of no use because the temporary scene file is deleted when the render has finished, anyway.

Anyway, this is the Pre-Render MEL script I wrote:

```
global proc postRenderScriptNextCamera()  
{
```

```

print ("I'm in the script!\n");

/* POST-RENDER MEL SCRIPT, THAT RENDERS FROM THE NEXT NUMBERED
CAMERA, WHEN THE CURRENT RENDER IS FINISHED */

int $currentFramesRendered = `getattr counterSphere.framesRendered`;
$currentFramesRendered++;

/* setattr counterSphere.framesRendered $currentFramesRendered; */

setattr counterSphere.sx 20;
setattr counterSphere.sy 20;
setattr counterSphere.sz 20;

int $totalFrames = `getattr counterSphere.totalFrames`;

if($currentFramesRendered == $totalFrames)
{
    /* ready for next time */
    setattr counterSphere.framesRendered 0;

    int $currentCamerasRendered = `getattr
counterSphere.camerasRendered`;
    $currentCamerasRendered++;

    string $nextCameraNumberString = $currentCamerasRendered;

    setattr counterSphere.camerasRendered $currentCamerasRendered;

    /* TO CHANGE RENDER CAMERA */

    string $nextCamera = ("camera" + $nextCameraNumberString);
    string $nextCameraShape = ("cameraShape" +
$nextCameraNumberString);

    changeMayaSoftwareCamera;
    connectControl -index 2 rgbChannel ("|" + $nextCamera + "|" +
$nextCameraShape + ".image");
    connectControl -index 2 alphaChannel ("|" + $nextCamera + "|" +
$nextCameraShape + ".mask");
    connectControl -index 2 depthChannel ("|" + $nextCamera + "|" +
$nextCameraShape + ".depth");

    /* get list of cameras in scene */

    string $camerasList[] = `ls -type "camera"`;
    string $i;

    /* set all cameras to be unrenderable */

    for($i in $camerasList){
        setattr ($i + ".renderable") 0;
    }
}

```

```

/* set the camera you want to render from to be renderable */

setAttr ($nextCameraShape + ".renderable") 1;


/* AND, BATCH RENDER AGAIN */

BatchRender;
}

}

```

My next idea was to use render layers and to try putting only one renderable camera in each render layer. I decided to do this after reading a discussion on CGTalk about the Pre-Render and Post-Render MEL scripts, my previous problem.

From this thread, and reading around in the Maya Help, I found out that the way that Maya works when specifying which camera to use for rendering, is pretty illogical, in my opinion. What happens when you specify which camera you want Maya to render from in the Render Globals Window, is that it makes all the cameras in the whole scene, un-renderable, except for the one you want to render from. I found this information from this thread on CGTalk. The reason this is stupid, in my opinion, is because there is an *if* statement in line 1044 of:

C:\Program
Files\Alias\Maya6.5\scripts\others\createMayaSoftwareCommonGlobalsTab.mel

Note: I found this information from the following thread on CGTalk:

<http://forums.cgsociety.org/showthread.php?t=272009&highlight=render+camera>

This *if* statement means that the above actually only works if there is only one renderable camera in the scene. Just one. So, if you were to manually go to the attribute editor and make another camera renderable, then, when you batch-render, Maya just takes the first renderable camera it finds, and uses that to render from. Therefore, if you went to the Render Globals tab and set to render from camera3, for example, but then you went to the attribute editor and manually turned on the 'renderable' attribute for 'camera1' and 'camera5', when you batch rendered, it would batch render from 'camera1', simply because that would be the first renderable camera it found. By this I mean 'camera1' would be returned first in a list of the

renderable cameras in the scene, because it is lowest numerically.

After finding this out, however, I decided I could make use of this quiriness to solve my problem anyway. By simply making only one camera in each of my new render layers renderable, when the batch render goes through and renders from all of the cameras (even though they are not renderable), the only ones that will produce any valid images are the ones from the camera in each layer that was renderable.

In the end, I was able to eliminate the use of rendering into separate render layers, because it was easier to create a Shake script that used images that were in the same directory, but had different file prefixes.

The advantage of this method is that there is now only one batch render, instead of spawning off lots and lots, as I was going to. Also, the user doesn't have to go into the command line, which makes it easier to use, as they only have to click a button in the interface.

One of the interesting things about my system, in my opinion, is the way it allows users with different aesthetic preferences to develop images which suit their own particular tastes, whilst only using the sliders in the interface. This was an important design consideration for me, as I wanted the user parameters to be fairly universal. By this I mean that the user parameters should create changes that, together, are likely to enable the production of images that the user will find aesthetically pleasing, or to their own taste. Yet these parameters are only tools. They have to be versatile enough to create different effects when they are used in different combinations with each other.

Colour and time were the main things I was interested in. So, the sliders in the interface, whilst they may actually be controlling the magnitude of a turbulent force being applied to a particle system, had to be an abstraction of this to the user. This aids clarity, as long as the user can anticipate what changes each slider will make to the system. I found this marriage of technical aspects with taste quite interesting.

Conclusion

When I started this project, the challenge I set myself was to produce a tool that would generate something else from a Maya scene. Something that wasn't an animation, wasn't a fly-through, or a render. I thought about it at the time as capturing the essence of the scene. My aim was to take what was unique about the

scene and condense it all down together in a way that hadn't been done before. This was the challenge I set myself.

I think I have come up with a tool that achieves this to a certain extent. It is not a complete success, because in my head I had always wanted it to be more intuitive to use. By this I mean I had wanted the user to be able to have more free-flowing interaction with it. As it is, it is an iterative process, where the user goes through the decision-making process in a trial and error way. Which is fine. But in order for it to have been a perfect marriage between the technical and aesthetic sides, there would have to have been more feedback at every stage.

For example, I think it would have been a more natural working process if, at every stage, the user could see exactly what the final rendered image looked like. Instead of having to wait to batch render all the frames. In terms of making an aesthetic decision, the more information that is available, the better, in this case. In practical terms this would mean having another window that updated constantly, much like an IPR Render.

My failure to achieve this part is because of two reasons. Firstly, my planning. Although the nature of this project meant that my work went in some directions that I could not anticipate, this particular requirement was something I had thought about before starting. If I had paid more attention to it, it wouldn't have got lost amongst the technical aspects of the project, which I think is what happened.

Having said that, this balance between what I am technically able to do, and what is inside my head has been a constant theme throughout my year, and in my Major Project especially. It is not unique to my Innovations project.

On the other hand, because I have had to learn more about MEL scripting to do this project, my Major Project has benefited. This is the most work I have ever done in MEL, and just the fact that I have spent so much time with it has meant I have been able to work much faster on the MEL scripts in my Major Project.

In fact, I think this has been the main thing I have learnt from the project, rather than any MEL techniques. Just the ability to disconnect myself from thinking in MEL terms, especially when the script isn't working very well. Being able to stop, and look at the work as a whole. Being able to see alternative solutions to problems. Knowing when to stop working on something and try a different method. Knowing when to keep trying. These are all the things I have got from this project. Different

ways of thinking about programming, because I am not really a natural programmer.

If I did this project again, or a similar project, I would know to think things through before I sat down in front of a computer. Plan my scripts out on paper. I know this is common advice, but I think it is even more applicable if your end target is liable to change.

I have made progress in my knowledge of MEL. I think one of the ways this is most apparent is that I have become very good at debugging my own code. Just being more familiar with error messages and warning messages now has meant I am able to realise what is making my code fail a lot faster than I was a few months back.

My code is about 1000 lines long. Yet, over time, it has become more and more efficient as I have learnt new ways to do things. There are some fiddly bits in MEL that I can now do quite quickly, that, a few months ago, I would have struggled with. For example, concatenating strings is quite fiddly, with escaped characters (`\` `\n`). Yet in my code I have concatenated strings within other concatenated strings.

Programming was not something that came naturally to me, before this project. It still isn't, but it is a lot easier.

I think this project has given me the confidence to continue doing this kind of work. I now know that my technical skills are sufficient to let me execute more of the things that I want to do.

It should be noted that towards the end of the project. during the testing phase, I discovered that my script actually could not handle Maya scenes that had animation in them. By this stage it was too late to fix this. I could have avoided this shortcoming by testing my script more thoroughly at every stage, rather than waiting until the end of the project.

How my script works

For a more detailed explanation of how my script works, please see the commented source code.

First, the script deletes any cameras that are already in the scene. This is to avoid them interfering with the rendering later on. This is a good example of the way that my script evolved. If I had had a rigid idea of what I wanted to create before I started,

I would have been able to plan out my code more thoroughly beforehand. This would have meant I could have anticipated this workaround, and had more specific camera naming conventions later on, instead.

Next, the script renames any particle objects and emitters that are already in the scene. This is another workaround that could have been avoided with more specific naming conventions, but it does enable the user to keep existing particle systems in the scene, so they are rendered in the final product.

Next, the script gets a list of all the polygonal and NURBS geometry that is in the user's scene into an array. The size of this array is therefore the number of cameras, particle system aim constraints, rendered sequences, etc that the script will then create. Note that because the script only makes use of polygonal and NURBS geometry in the scene, anything that does not fall into these categories is left alone. This means that any Fluids or lights, for example, that are already in the user's scene, will be left alone, so they appear in the final render. But they will not be included in any of the calculations like the NURBS or polygonal geometry, and so will appear in unpredictable places in the final render. This was one of the things that I wanted to have in my script, the use of 'found' objects in the scene.

Next, the script creates three particle systems that will be used to provide the fluidity of movement in axes that I talked about above. The first particle system, 'particle1', is used to drive the positions of the cameras that will be created later. The second particle system, 'particle2', is used to drive the positions of the second goal objects of the cameras, which are called aimSpheres. This means that when the cameras are created, they will move with the particles from the particle system 'particle1'. Also, when the particle goals are set up later, the cameras' aim constraints, which is where the camera points at, which are driven by the cameraAimObjects, set up by the third particle system, 'particle3', will oscillate between two positions. The first position will be the position of the object; the second will be driven by the position of the particles in the particle system 'particle2', the aimSpheres.

Next, the script creates an individual particle system for each aim constraint. These particle systems only have one particle in them. This particle has two goals. The first goal is the object this particle system is for. The weight of this goal is determined by the colour component of the object, which I will go into later. The second goal for the particle in this particle system is the 'aimSphere' that relates to this particle system. The weight of this goal oscillates from zero to one as a default. This can be adjusted interactively via the interface, during playback. The position of this 'aimSphere' is

controlled by the second particle system, 'particle2'.

Next, on line 251, the script creates a vortex, turbulence and radial field, and connects them to 'particle2'. This means that the user can apply these dynamic, physics-based forces to the second goal objects of the cameras.

Next, the script adds per-object attributes to 'particle1', to hold the names of the objects in the scene. This makes it easier to reference this information later.

Next, the script create a particle creation expression for 'particle1'. This expression creates one camera for each object in the user's scene. It also makes a couple of checks to ensure that all the right cameras have been created, that there aren't too many cameras created. It also deletes any of the user's original cameras. This is necessary because they would interfere with the rendering process later on. Also, there is no reason that the user would want to keep these cameras, as long as they make a copy of their scene, before running my script on it.

Next, about line 410, the script creates the runtime after dynamics expression to set the position of the cameras to be the position of the appropriate particle, on every frame. Although the position of the cameras is not something that needs to be updated on every frame, I have included it here so that any enhancements in the future can be more easily made. For example, the script could be extended to provide control over the motion of the cameras themselves, again using particle dynamics, fairly easily. As it is now, the overhead of updating the position on every frame is negligible.

Next, the script creates the particle creation expression for 'particle2'. This creates the aimSphere for each object, and sets its position to the particle's position for the first frame. The runtime expression for 'particle2' simply updates the position of these aimSpheres to be the position of the appropriate particle, on every frame.

The creation expression for 'particle3' on line 438 creates the cameraAimObjects for each camera, which are polygonal cubes. It sets their positions to be that of the appropriate particles, for the first frame. It then sets this cube to be invisible, so that it doesn't appear in the renders.

The script then steps forward a few frames, so that the necessary aim constraints are created by the particle expressions, in the creation expressions.

Next, goals are set up for the 'particleAIM_systemParticle' particle systems, each of which only has one particle in it. Therefore, this individual particle gets set the two goals just for itself. The first goal is the corresponding object in the user's scene. The second is the corresponding aimSphere. At this stage, both goals are given a default weight of 0.3, although this will be changed later on.

Next, on line 476, the script gets a list of all the shaders that are attached to the objects. It then extracts from this the material nodes that are attached to the objects. This is what is used to determine the colour components of each object, for later goal adjustment. At the moment the script only takes into account the colour components of these material nodes. It would clearly be more useful for most Maya scenes to find the average colour of a texture map applied to the objects. This would be the place to do such a procedure in my code. The reason that I haven't done this is simply because I ran out of time. Better planning would have enabled me to achieve this.

The script then sets the dominant colour to be the colour that the user called the script with, either "red", "green" or "blue". Then the script gets the colour components of the material into a vector array. Then, for each object, the script gets the appropriate colour component, based on the dominant colour chosen by the user earlier. Next, the script puts the required colour components into an array for later.

The next stage is to add attributes to 'particle1' to hold goal weight values for the necessary objects. Also, attributes are added to 'particle1' to hold values for the oscillation of the goal weights, and the user-defined goalPower, which affects the goal smoothness.

Next, the script creates the particle runtime after dynamics expression for the 'particleAIM_systemParticle' particle systems. These expressions get the value that the user has specified via the interface for the oscillation value. They also read the value from 'particle1' for the appropriate goalValueObject. These were stored in attributes in 'particle1' earlier. Using the number of the object and the user-specified oscillation value, the script then calculates a second goal weight, which will provide the strength of the 'distraction' away from the object itself. It then gets the user-specified goalPower and assigns that to the goalSmoothness of the 'particleAIM_systemParticle' system. Also, it sets the first goal weight of this particle system to be the goal weight that was calculated based on the colour component of the object, and the second goal weight to be that which was calculated as an oscillating value. Finally, it moves the cameraAimObjects and the camera aim constraints to follow the position of the one particle in the appropriate particleAIM_systemParticle

system.

Next, on line 662, the script creates a window with a slider to change the value of `particle1.goalPower`. It also creates sliders that directly change the values of the following attributes, which I have listed here together with the abstractions I have given them:

Attribute	Abstraction
<code>particle1.goalPower</code>	Camera speed
<code>particleShape2.conserve</code>	Camera Aim Speed
<code>particle1.oscValue</code>	Oscillation Speed
<code>vortexField1.magnitude</code>	Vortex strength
<code>turbulenceField1.magnitude</code>	Turbulence strength
<code>radialField1.magnitude</code>	Radial strength

On line 698, the script creates a text box which will later contain the Shake script which will be used to layer all the rendered sequences together. It also gets the file prefix that the user entered into the appropriate text box in the interface, and sets up the button that will enter and record this file prefix when the user enters it. Next, the script gets the start and end frames from the Render Globals.

The next stage is to start creating strings to be concatenated into a Shake command later on. First a string is created that will itself get the file prefix. The next string that is created is actually the part of the Shake script that will contain the flags to specify the start and end frames for the Shake script. The next string is a default start Shake command to put into the text box, to start with. Also, a text box is created with this default Shake command in it, for now. Next, buttons are created that will later be used to update the Shake 'system' command, and to actually run it.

Next, on Line 782, is the process of concatenating the new Shake command string. First, a string array is created to hold each part of the Shake command, one part for each rendered sequence. The first part of the Shake string, which is therefore the first element in the string array, gets the time range and the file prefix.

The clever bit is that this string also creates another string, within itself. This new string is actually the first part of the actual Shake command that will be executed. It creates a 'fileIn' node within Shake which gets the first image sequence, labels it 'BG1', and sets the time range for the Shake command.

Next, on Line 824, for each object, and therefore each rendered sequence of images, the script creates a new part to the string, that will be used to update the Shake command later on. Again, within these strings, the script creates new strings, which are the next parts of the actual Shake command that will be executed. These parts of the Shake command each create a 'fileIn' node in Shake which gets the next rendered image sequence, creates an 'over' node to layer it over the previous 'BG' layer, eg 'BG4', and finally labels the new layer, eg 'BG5'.

The next stage is to concatenate together all the strings that have just been created, to create the final Shake command that will actually be used. On Line 877, the script starts with an empty string, then, for each of the objects, and therefore each of the rendered sequences, and each part of the Shake command, the script adds each part of the Shake command to the previous, by adding together the consecutive array element strings together into one long string. (Note that this concatenated string is not the actual Shake command that will be executed, it contains the strings that will set this final command up).

Next, on line 918, the script concatenates together a string that will itself join together as many parts of the above Shake command strings as there are. Starting with an empty string, it creates a string within itself that will get the first part of the final Shake command from the first element of the above array. It then gets the remaining pieces of the Shake command by taking the remaining string elements of the above array, and concatenating them, one after the other, to the string that was just created.

On line 962, the script creates two strings that it will need later, because it will later need to concatenate strings within concatenated strings, and will need to escape escaped characters, eg `\\\"` and `\"\\\"`).

Next, on Line 972, the script adds, to the above string, a string that will assign the string `$newStringFinal` that was created within the above string, to the text field that displays the Shake command to be executed.

Finally, on Line 991, the script joins all the bits together, and on Line 1002, it assigns the final concatenated command to the button that updates the Shake command. This means that when this button is pressed, it will update the Shake command, taking into account the file prefix, time range, and how many sequences of rendered images there are.

On line 1013, the script creates a string for the button that gets text from the text field and executes it, using the MEL 'system' command to run Shake. On Line 1019, the script adds the necessary bits to make it a 'system' command, by adding strings to the start and end. Finally, on Line 1027, the script prints the 'system' command, before executing it with the MEL 'eval' command.

On line 1043, the script assigns the finished command to the button that runs Shake. The result of all this is that the user can run Shake, layer together all the sequences of images they have rendered out, with alpha maps, and instantly produce a flipbook render of the final, composited result, all with one click of a button in my interface.

The final stage of my script, on Line 1062, is the Rendering. On line 1066, the script goes through all the cameras and makes them all unrenderable, including the default 'persp', 'top', 'side' and 'front' cameras. Next, it creates a render layer with the original geometry and created cameras in it, after, on line 1090, going through all the cameras the user will need to render from, and making them all renderable.

Appendix

Conics-enhanced vision approach for easy and low-cost 3D tracking

[*Pattern Recognition*](#)

[*Volume 37, Issue 7*](#) , July 2004, Pages 1441-1450

Tien-Lung Tien-Lung [Sun](#)  

DOI Bookmark: [doi:10.1016/j.patcog.2004.01.009](https://doi.org/10.1016/j.patcog.2004.01.009)

Paracatadioptric camera calibration using lines

[*Ninth IEEE International Conference on Computer Vision \(ICCV'03\) -*](#)

[*Volume 2*](#) p. 1359

[Joao P. Barreto](#), University of Coimbra

[Helder Araujo](#), University of Coimbra

DOI Bookmark:

<http://doi.ieeecomputersociety.org/10.1109/ICCV.2003.1238648>

Speed control in path motion

Proceedings of SPIE -- Volume 2644

Fourth International Conference on Computer-Aided Design and Computer Graphics, Shuzi Yang, Ji Zhou, Cheng-Gang Li, Editors, March 1996, pp. 318-324

[Jieqing Feng](#) and [Qunsheng Peng](#)

Zhejiang Univ. (China)

[doi:10.1117/12.235537](https://doi.org/10.1117/12.235537)

A Camera Motion Metaphor Based on Film Grammar

Journal of Visualization and Computer Animation. 1996, Vol. 7, Issue 2, pp 61-78

Patrizia Palamidese

Digital Cubism

IEEE Computer Graphics and Applications, vol. 24, no. 3, pp. 82-90, May/June, 2004

Andrew Glassner

DOI Bookmark:

<http://doi.ieeecomputersociety.org/10.1109/MCG.2004.1297016>

A tutorial on developing a computer-controlled camera system

ACM SIGGRAPH Computer Graphics

Volume 20 , Issue 1 (February 1986)

Pages: 7 - 16

Year of Publication: 1986

Neil Sullivan

C Durward Rogers

Stephen Daniel

ISSN:0097-8930

CINEMA: A System for Procedural Camera Movements

Special Issue of Computer Graphics

Proceedings of the 1992 Symposium on Interactive 3D Graphics

Pages: 62-70

Steven M. Drucker, Tinsley A. Gaylean, David Zeltzer [1992]

CamDroid: A System for Implementing Intelligent Camera Control

Proc. 1995 Symposium on Interactive 3D Graphics, Monterey CA, April 9-12, 1995, pp. 139-144.

S. M. Drucker and D. Zeltzer,