Innovations Report

dlScript

A Maya tool for generating animated textures using light-baking

Daniel Lim (c1462178)
NCCA BACVA3
2005/06

Abstract

This report present a method for generating animated textures using light-baking. This is done using occluding objects to block the light rays from a surrounding light globe. This is has been put into an automated MEL tool that takes a user through the steps needed to generate these animated maps.

1. Introduction

1.1. What are wet maps and why would you want animated ones?

To understand wet maps, it may be beneficial to first examine why materials look wet. Essentially an object looks wet because it has a thin layer of water on its surface. Water has no obvious diffuse component so it is made up completely of specular reflection and holds no colour information of its own. It's also refractive but the only indication of this is a darkening of the surface, since the layer of water on the surface is so thin:

"In hard surfaces like a rock, this is due to internal reflections between the water surface and the rock surface, which dims the light reflected back to the eye. In porous surfaces like cloth, a similar phenomenon is happening, the surface absorbs water, so when the surface is struck by light the light scatters deeper into the material, and so hits more particles before leaving the cloth. Each particle absorbs a slight amount of the light, so when the light emerges from the material back to the eye, the surface appears darker because more of the light is absorbed in the material."

"Wet-looking Materials", Neil Blevins - CG Education, 2005 [1]

A wet map is simply an added layer of control, allowing a user to decide which sections of an object are wet and which are not. In the case of this project, the goal was to have droplets of water running down the face of a computer-generated character, doing some kind geometry-based simulation where by the droplets that have been animated (for example, using dynamics) would then be used to generate wet maps for each frame of the animation, thereby leaving a wet trail where the droplets have been previously.

1.2. Light-baking/Light-mapping

Light-baking (also referred to as light-mapping or pre-lighting) is a technique for approximating the global illumination in a scene. You place your lights and objects in your scene and, providing they don't move, the light information can be "baked" into texture maps for the objects, removing the need for lighting to be re-rendered again and again for every frame.

I was already familiar with the concept of light baking and its application in the field of real-time graphics, however during a conversation with colleagues regarding my project, we discussed the possibility of using light baking to generate animated textures. Having never heard of using light baking for such a purpose I thought it would be an interesting prospect to explore. I have yet to find an example where someone has used light-baking to generate animated textures.

2. Related Work

2.1. What other ways are there to represent water running over a surface?

There are a couple of packages available on the market that are capable of representing fluids. Perhaps the most famous of these is RealFlow by NextLimit Technologies.

"[RealFlow is] the world's most advanced particle and fluid simulator. This [particle based] method differs from the traditional system of finite elements in its adaptability to complex environments and deformations. With RealFlow, the particles themselves can be considered part of a global fluid that acts according to the viscosity, pressure and surface tension forces of neighbouring particles and, through constant development, is capable of handling up to 2 million particles in a standard PC."

*"**Real Flow Features**:*

- *Incredibly fast!*
- *Cross-platform (Linux, Mac OS X, Windows)*
- *New user interface and command line options*
- *Fully integrated Help Menu*
- *New fluid engine solver*
- *Improved rigid body solver*
- *Rigid body constraints*
- *Mesh generation engine*
- *Waves and buoyancy (RealWave features)*
- *Efficient memory management*
- *New curve and expressions editor*
- *Daemons which can affect fluids and objects*
- ***Wet texture mapping ("wetmap")***
- *Particle and mesh texturing options*
- *Integration with the 3D platforms*
- *…and more"*

"What is RealFlow?", RealFlow Website, 2006 [6]

RealFlow also has a system where the particles of the fluid collide with an object and "paint" over its UVs. There is also an in-house software package called Flowline by the German company Scanline [2], which is capable of similar simulations, however this product is not available to buy on the market.

3. Aims & Objectives

My aim is to create a tool that has:

- a simple system to set up dynamics simulations of water droplets on surfaces
- a fast simulation method
- an ability to create light globes
- ways to set up scenes for light-baking
- the capability to generate animated wet maps
- an easy-to-understand and use interface
- a good-looking interface
- all its parts in the same place (one script)

4. Production

4.1. Simulation & Curve Creation

The first component of the project was to create a system for conducting simulations. To begin with, a simple scene is created with a polygon sphere and cube. The plane is modelled into a bumpy surface, purposefully shaped to have natural lines of flow and was designated as a passive rigid object (in other words it stays still and other 'active' objects interact/collide with it). The sphere (the active object) has 2 gravity fields applied to it; one pulling the sphere downwards and the other, slightly less powerful, field pulling it towards the polygon plane. At simulation time, the sphere moves downwards and towards the plane, collides and will then proceed to "roll" down the surface.

The original idea for the project was to use simple dynamics simulations to generate a series of locators based on an object's position as it moved across a surface. The locators could then be used to generate curves from which a number of possibilities arise, for example, the curves could be made into motion paths, along which geometry could be animated.

For the curve creation, a locator is created and point constrained to the centre of the sphere, thereby locking the locator's position to wherever the sphere goes. Coming back to MEL having not written any for a long time meant that writing this code was not as quick as it should have been, but eventually it could successfully generate locators for every frame of the sphere simulation.

```
//LOCATOR-CREATION EXPRESSION
//Locator Creation based on position of 'Centre' locator every frame
if(pSpherel.Evaluate_Exp) {
        vector $vector;
        $vector = `xform -ws -q -t "centre"`;
        spaceLocator -p ($vector.x) ($vector.y) ($vector.z);
}

//Variables
float $acc = 250;
vector $vector[250];

//Vector
for($time=1;$time<=$acc;$time++) {
        currentTime -e $time;
        //assign locator positions to vector
        $vector[$time-1] = `xform -ws -q -t ("locator"+$time)`;
        $holder = `currentTime -q`;
        print($holder+"\n");
};

//Printing out each locator position individually
for($time = 1; $time <= $acc; $time++) {
        print("Vec"+$time+" = "+$vector[$time-1]+"\n");
}

//Create string
string $command = "curve -d 3 ";

//Append all curve points (locator positions) to string
for($time = 1; $time <= $acc; $time++) {
        $command += (" -p "+ $vector[$time-1]);
}

//Create curve from string
eval $command;
```

Figure 1 – MEL script for saving locator positions and for creating a curve based on those positions

The code was then changed and the tasks of storing the position as a vector and creating a locator based on those values were made into expressions. It makes sense to use expressions on tasks that happen every frame since that is how often expressions are evaluated. Of course, this means that every time the user scrubs through the time line, the expression is evaluated and locators are created for each frame that is visited. To prevent this, a custom attribute is added to the sphere that allows the expression to be switched off once the locators have been positioned. It is also recommended that playback looping is switched off in the Maya preferences.

4.2. Creating the Wet Look

It became apparent that whilst recording the positions of the sphere as locators and generating from them would be attainable, the skill and knowledge required to generate decent water simulations lay outside both the scope of the project and my abilities as a coder, particularly

as one of the most technically astute of my colleagues has set himself a similar challenge for his major project [3].

As a result, the project's focus changed to concentrate on another aspect of the task: creating surfaces that look wet. To begin with, simple tests were created; applying a texture to a polygon primitive and attempting to make it look wet using the "simple" definition of how wet objects look: high, sharp specular and no diffuse in the wet layer and the object texture, depending on the material from which it's made, should be darker and more saturated. In the Hypershade, the object texture and wet layers are put into a layered shader, with the wet layer on top; since it has no diffuse component, the object texture can be seen underneath the wet layer. This is taken a step further by connecting a bump map into the wet layer. The object texture is used to both colour the object and provide a bump for the wet layer.

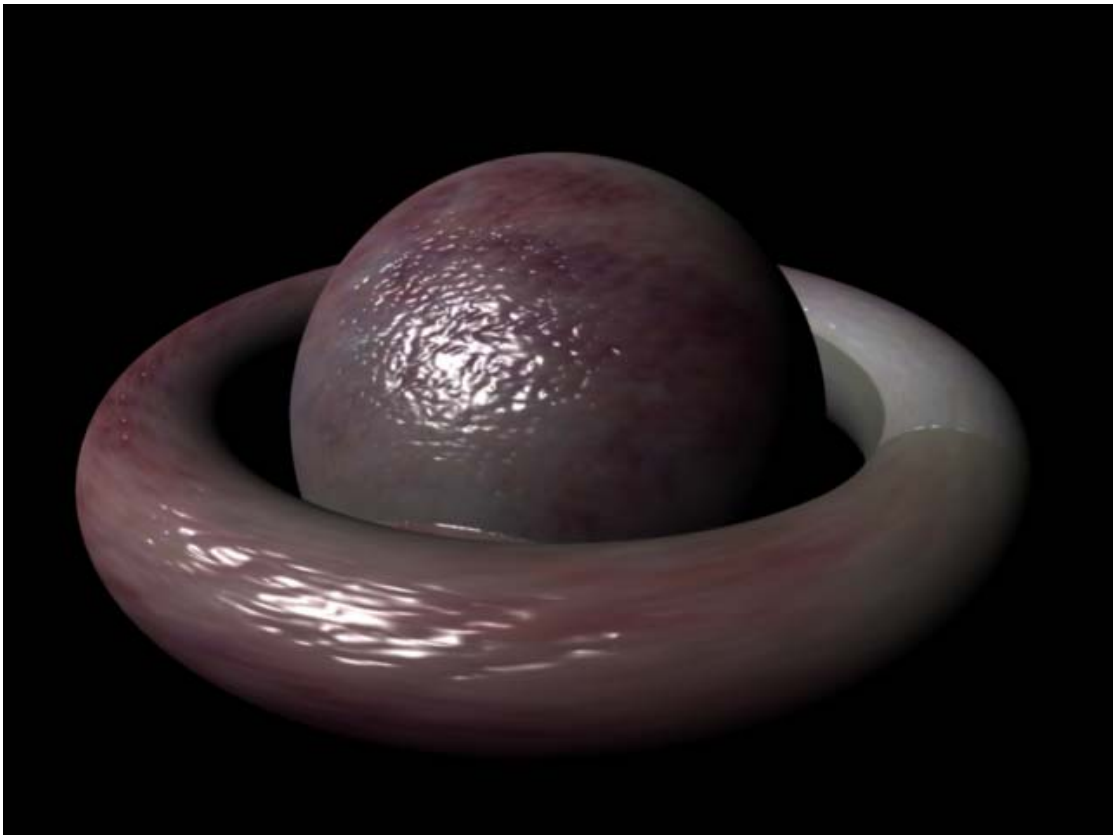


Figure 2 – MEL script for saving locator positions and for creating a curve based on those positions

Whilst the results were pleasing based on prior knowledge, it was clear that more had to be done.

Rather than spend an inordinate amount of the remaining project time modelling, UVing and texturing a suitable model, I was generously donated one by Matthew Birkett-Smith [4]. This came fully textured with three maps (colour, specular and bump). This model is taken through similar steps as previously described; another specular, specifically for the wet layer, is added

onto the layered shader that Matt had already set up. Unlike previous experiments however, the purpose-built bump map is used as the bump for the wet layer rather than just using the colour map.

Figure 3 – Head model with colour, bump, specular and wet layer added directly on top

The results aren't as immediately pleasing as last time! There are many more lights in this scene (as opposed to just the one directional light in the last) and as such, the specular is over-pronounced and in great abundance.

If we look at a photo of wet skin, we can see that, generally speaking, only the parts of the skin that are facing towards the camera are giving off reflections. This means another node must be added to the wet layer. The samplerInfo node is a utility you can use to get all kinds of useful information for creating shaders. It provides you with information about each point on a surface as it is being "sampled" (i.e. calculated for rendering purposes).

Figure 4 – A photo of wet skin, showing how specular reflections are only visible on surfaces that are facing the camera. Source: flickr.com, Owner: unknown

Perhaps its most commonly used attribute is facingRatio:

"Facing Ratio gives you a number between 0 and 1 that tells you if the point being sampled is facing towards or away from the camera. A value of 1 means that it is facing the camera head-on. A value of 0 means that is facing 90 degrees from the camera. In mathematical terms, Facing Ratio is the cosine of the angle between Ray Direction and Normal Camera. "

Maya Help Documentation [4]

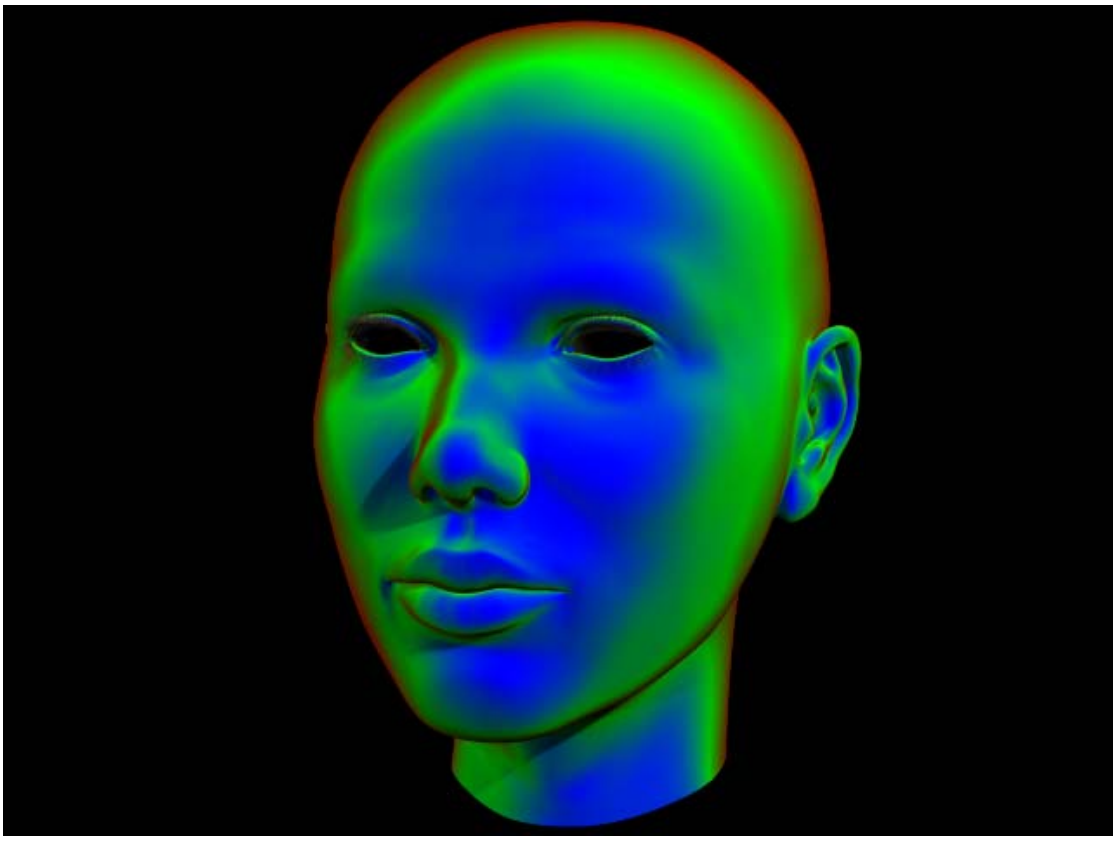


Figure 5 – Head model with a ramp controlled by facingRatio

When plugged into a ramp, facingRatio can show the user which points on the surface are facing towards or away from the camera based on what colours are used in the ramp. In the example above, blue represents where the surface is facing towards the camera, red represents where the surface is facing 90 degrees or close to 90 degrees away from the camera, with the green band representing everything in-between. Using reference, the ramp is tweaked until the blue area is roughly the right size for representing the specularity of wet skin. It is then connected to the wet layer such that specular reflections only occur when the skin surface is close to facing the camera head-on.

Figure 6 – Head model with an added (facingRatio) wet layer

4.3. Generating Animated Textures

The light-baking code was generated by conducting normal light-bakes and checking the Script Editor for which commands are executed each time. It is always useful to turn on "Echo All Commands" and follow what is being dong at each step of a process. The 'whatIs <command name>' is also a useful tool; reporting whether a given command is indeed a command or whether it is in fact a MEL procedure. The location of the MEL file that it is contained within is also printed.

The key drawback of light-baking is down to its usual purpose. The whole point of light-baking is that you render your lighting once in order to prelight a scene and then save time by not having to render lights and shadows every frame (naturally this only applies to lights and objects that don't move within the shot). As it's only meant to be done once, it is highly restrictive as to what filenames render images receive. By default, the texture bake set (all the objects that are to be baked together) has the prefix "baked_". One the first batch baking, the rendered image is called "baked_<shadingGroupName>_<objectName>.<fileExt>". Subsequent baking results in having the "baking_" prefix and "<_objectName>" suffix being added *each time*. This is because the shadingGroup was renamed each time to replicate the outputted filename. As you might imagine, the filenames can become very unwieldy over the course of only a few frames.

It was initially thought that by renaming the shadingGroup before the first bake and after every subsequent bake that this problem could be avoided and each filename would simply imitate the shadingGroup name. The original procedure renamed the shadingGroup "Frame_xxx" where 'xxx' is the padded frame number. Unfortunately, this did not solve the problem as the shadingGroup cannot be renamed until after the first bake, by which time the shadingGroup (and hence the render file) is designated a prefixed and suffixed name. Maya does not accept this formatting of filename for use as an animated texture.

The immediate contingency plan was to rename the object name each time (which also doesn't work incidentally). Luckily, changing the prefix was tested before the procedure was finished; if you provide your own prefix, it is also used as the filename that is generated, so it's simply a case of changing the bake set's prefix attribute every frame, according to the user-specified file prefix (an option I built into the tool). The script will only allow users to specify the prefix to the filename. This is because Maya is very specific about what naming conventions it will allow as image sequences. In the case of this project, the filenames appear as "<prefix>.xxx.tif". This is far from a perfect method for frame padding (as will be discussed later), however I am pleased that it is an original solution;

```
int $int1 = 0;
int $int2 = 0;
int $int3 = 0;
int $total = 0;

for($time=1;$time<=150;$time++) {

   $total += 1;
   //Checking for comparison
   print "total ";
   print $total;

   if($total % 100 == 0) {
      $int1 += 1;
      $int2 = 0;
      $int3 = 0;
   } else if($total % 10 == 0) {
      $int2 += 1;
      $int3 = 0;
   } else {
      $int3 += 1;
   }

   //Checking for comparison
   print "\n";
   print $int1;
   print $int2;
   print $int3;
   print "\n";
}
```

Figure 7 – Frame Padding code

There is currently no choice as to the image file type or size. It is worth pointing out that had this method for file naming also failed, there would be no choice but to call the sysFile command and rename the files, post-rendering, using platform-specific commands.

4.4. Cross-platform Issues

The ‘about’ command was necessary to check what platform the user is running Maya on. There is an if statement that checks and draws windows of different sizes depending on whether the user is running Windows or Linux because Linux uses a slightly wider default font. Naturally this poses a lot of layout problems if the tool is developed on Windows. Due to the simple nature of dlScript’s layout, very few problems were encountered and those that were normally solved by increasing the window very slightly on Linux to allow the text to fit.

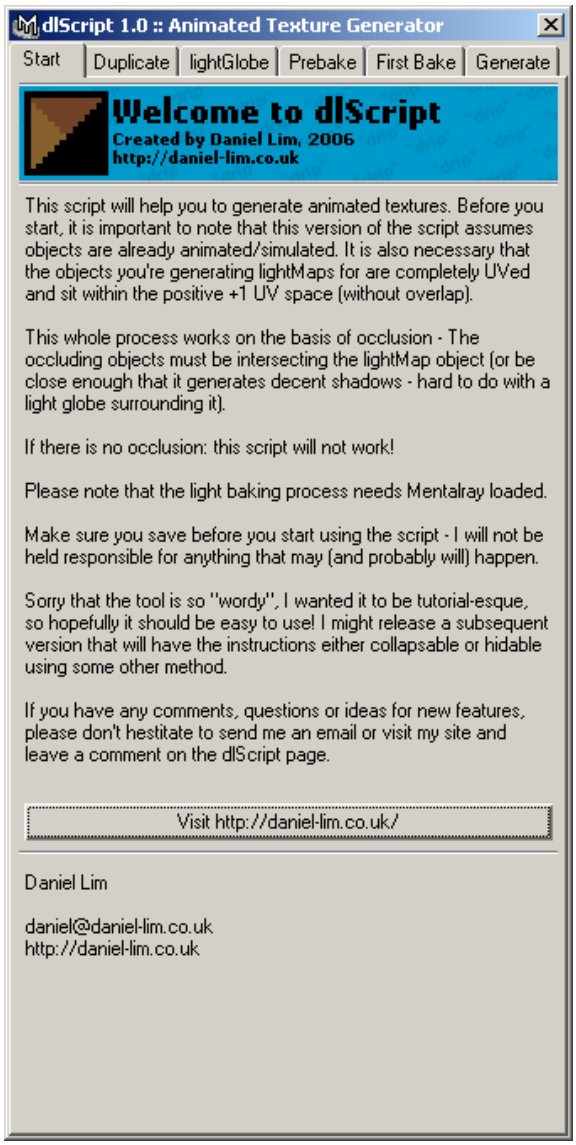
Window preferences are deleted every single time you open the script; otherwise Maya saves window sizes and positions by default. This issue is less of a problem as the script’s window has also been defined as non-sizable, allowing no room for possible display issues.

I had to change the way I created my light globe spotlights. Originally the lightGlobe procedure called the defaultSpotLight procedure for its spotlight creation, however despite having arguments that you can pass that work on Windows, it seems that the Linux install ignores them, as such the spotlights were not being created with RayTrace shadows on by default.

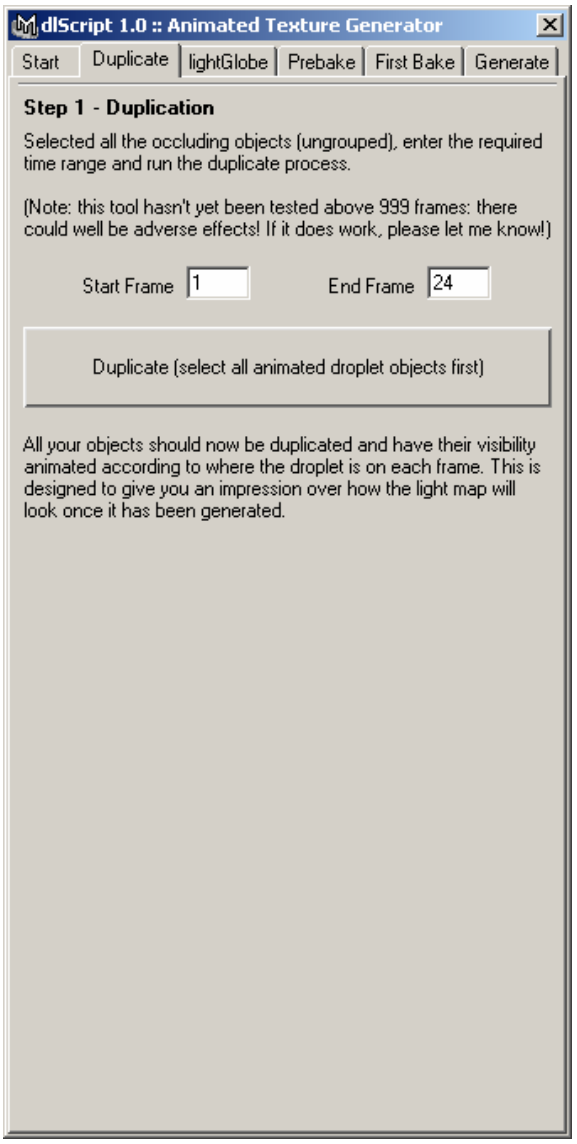
5. Results

5.1. What was produced?

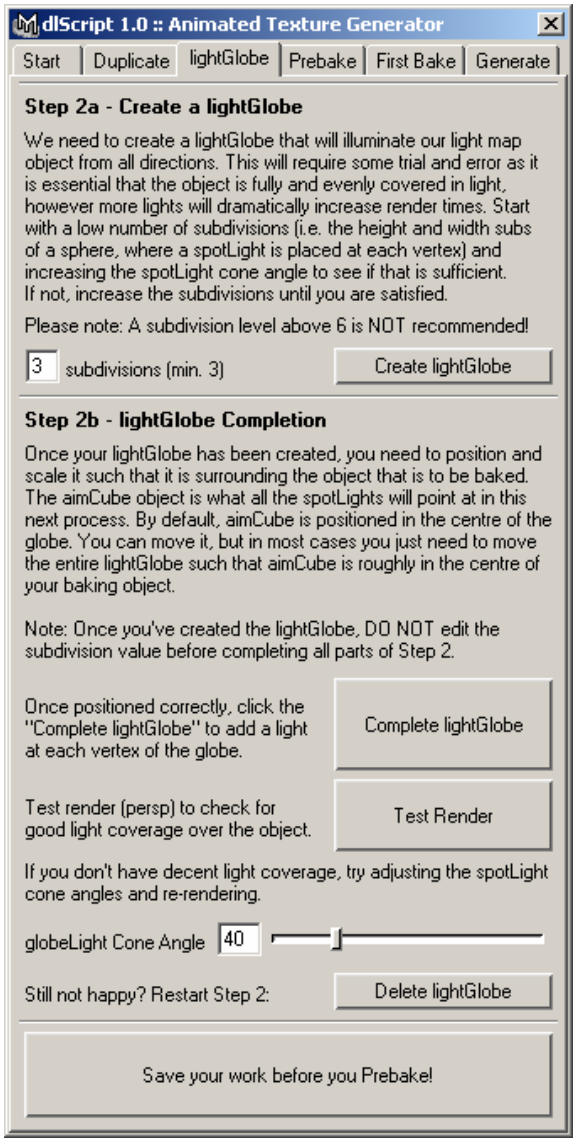
The final product of this project is dlScript. Using light-baking and occlusion, this tool is capable of generating animated textures for numerous uses (e.g. animated wet maps) and concerns itself with all the steps necessary to produce these textures. Once the texture frames have been rendered, the tool can also apply them as an animated material for previewing purposes. As the focus of the project changed, the focus of the tool changed with it. Originally it was designed to help set up simulations and then use those simulations to generate animated wet maps. It's actually now a tool that can be used for generating animated textures of any kind, not specifically wet maps, it has other applications; for example, given an animation/simulation of a ball rolling down a grassy hill, a user could easily use the tool to create an animated texture for flattening down the grass or determine where it parts as the ball rolls across the surface.



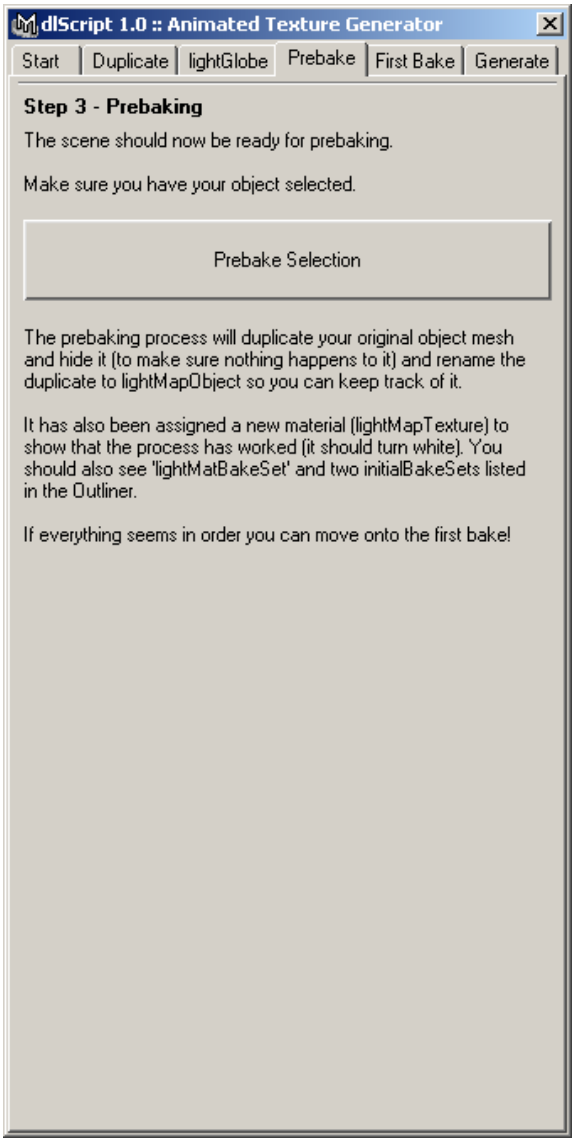
1. The opening page, there is a long explanation of what the tool is, and what caveats there are to using the script.



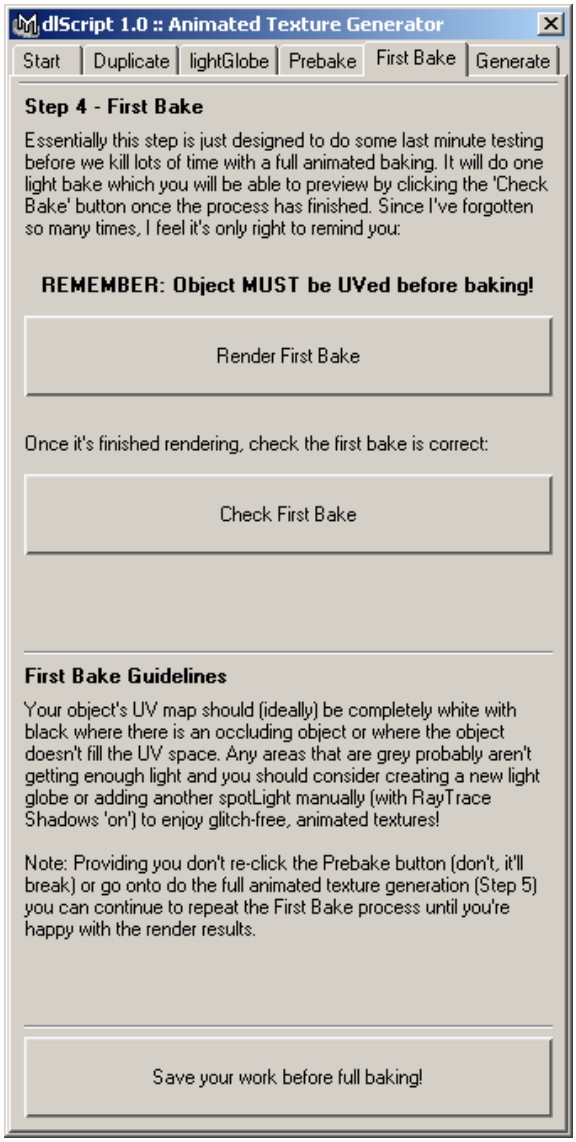
2. In this tab a user can duplicate all the occluding (droplet) objects and have them duplicated and their visibility keyed according to what frame the animation is on.



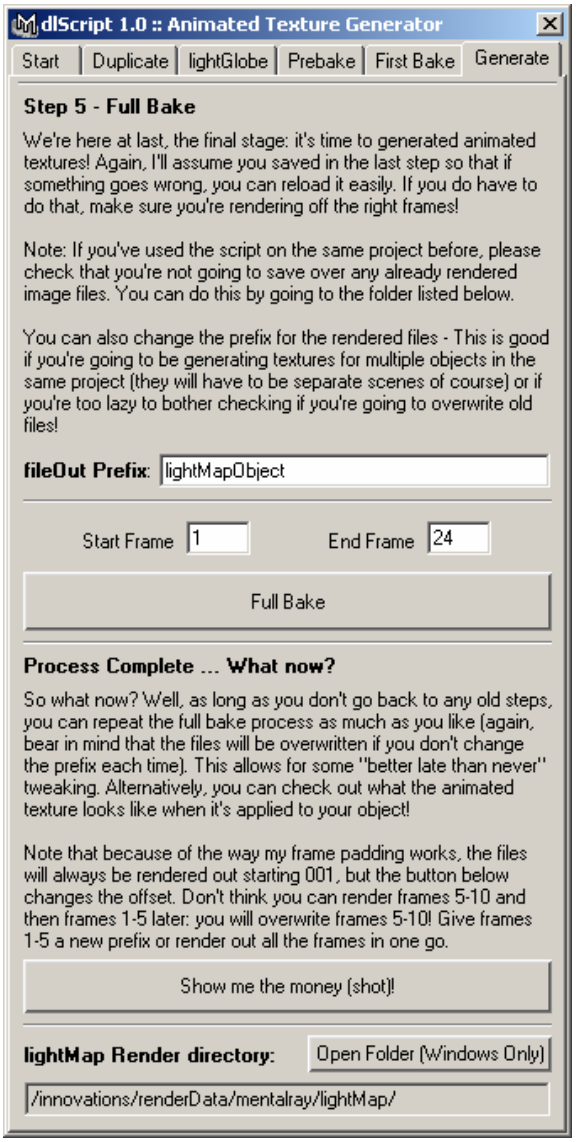
3. The lightGlobe tab walks the user through setting up a light globe (of their specifications) to completely illuminate the object their generating their animated textures for. The buttons in Step 2b do not become clickable until Step 2a has been completed.



4. The Prebake tab does some renaming of objects, materials and shading groups to prepare for the first bake where an actual image will be rendered out.



5. The First Bake is essentially a form of previewing to check that the textures are rendering out properly and give the user an opportunity to tweak any settings before rendering out textures for multiple frames.



6. This is where the user can generate their animated textures, assuming everything else has gone to plan. They can specify a prefix for their filenames, conduct the full bake and even apply the textures to their object to see the animation in motion! There is also a button (currently Windows-only) that opens the folder containing the rendered files.

6. Critical Analysis

6.1. What could be changed/added to the product?

Although the tool is completely functional, there are a number of changes and additions that could be made to make it even more useful. To begin with, the script won't run on Maya 6.5 and I don't know why. This needs fixing urgently.

Perhaps the biggest problem with the tool as it is now: it can certainly be made *much* more efficient. It currently uses a lot of evaluating and doesn't pass any global variables as arguments through the script; they just get reassigned locally each time even though they might be the same name and value. For example, the coneAngleEdit() and coneAngleSilder() procedures are essentially the same thing but working in reverse to each other. I'm sure they could be merged into just one procedure in the future.

Based on my original aims, in theory the tool is still not finished. In the future it would be great to be able to build in a simulation system or perhaps incorporate Jimmi Gravesen's major project into it, to simulate particles and then instancing droplet shapes onto them.

Another key problem is that if the occluding objects move too quickly, gaps appear in the path of the object when it is rendered out. It would be worth looking at different ways the occluding object could work. By changing the way the objects are made, it could be possible to use a simulation to generate a curve and then have an object extrude along it.

The frame padding is broken – It is impossible to start rendering frames with an offset, for example starting at frame 5 instead of frame 1. The way it's been written also means that it will probably break above 999 frames, although naturally, the chance to render out 999 bakes has not arisen thus far!

As well as a spotlight cone angle slider, it would be good to add a spotlight intensity slider as well, saving the user from having to select all the lights and go into the Attributes Spreadsheet Editor.

A fairly big extension to the script could be to include some way of baking in colour as well, so you can do proper animated textures (not just greyscale ones).

Lots of procedures require that the user not change names of objects as they're working through the script. It also relies on them not closing the script and starting it up again mid-way through the process – if they do, they have to make sure the start and end frames are the same and that the subdivisions are equal to what they were before they closed the window. It would be good to make these global variables. Equally, there should be some option that can be used to specify the output file resolution – currently locked at 512x512 (for ease).

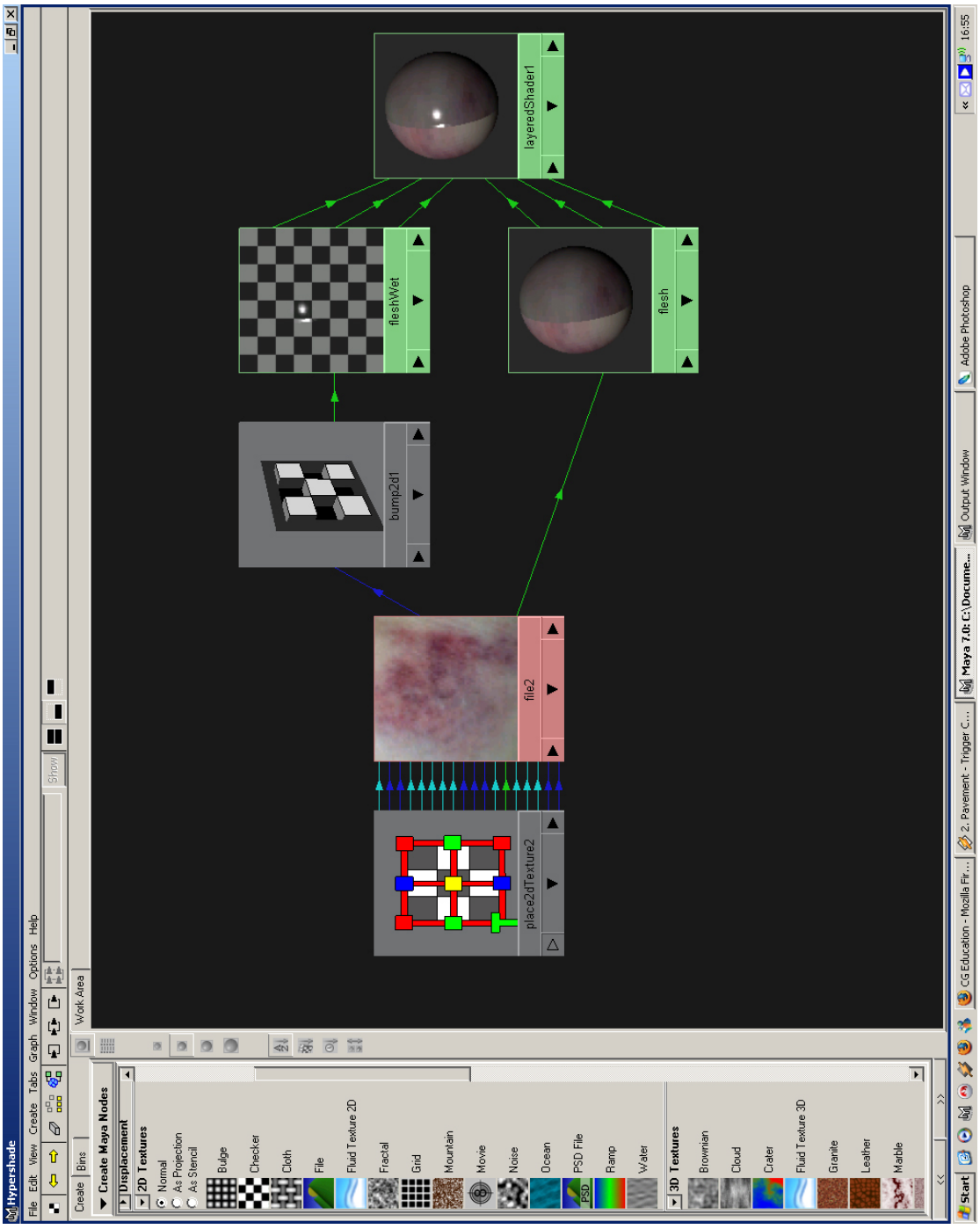
6.2. What has been learned?

Over the course of this project, I have learned lots of new things. Firstly I understand shaders a lot better now, in particular, how to make things look wet and how to layer up shaders in the Hypershade. Knowing more about shading and how it actually works will definitely help my major project, as will creating this tool since there's a scene requiring water droplets running down the face of my character. I know more full understand the process of light-baking and its applications in CG and not only that, but also that there are always numerous ways to take on a task and it can be really fun to try a particularly unconventional way. The project has also forced me to re-learned lots of MEL that I had forgotten since the first (and to a degree the second) year. I also learned lots of new MEL including how to create UIs; something I've never done in great detail before.

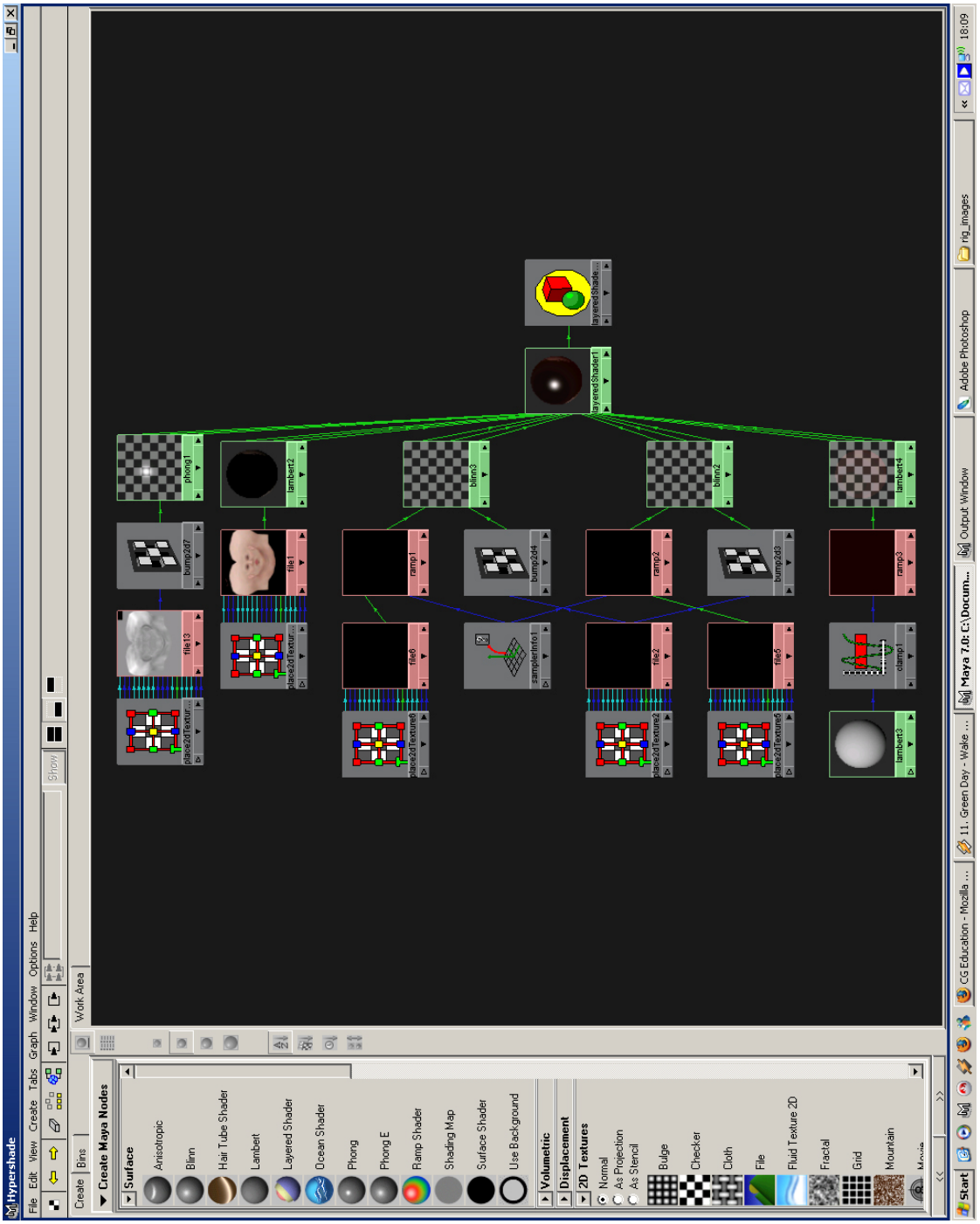
7. Conclusion

I think overall, the project was successful. Admittedly, I didn't achieve what I originally set out to do, but perhaps I had set my sights a little too high. The change of focus to looking at creating wet materials and then writing a script to generate animated texture maps proved to be a breath of fresh air and something that I could really get stuck into. I really enjoyed creating the script and the UI; it's definitely something I'll consider doing again in the future. I think the project is successful because I've got a fully functional tool that is both easy to understand and use and it's actually useful, particularly for my major project.

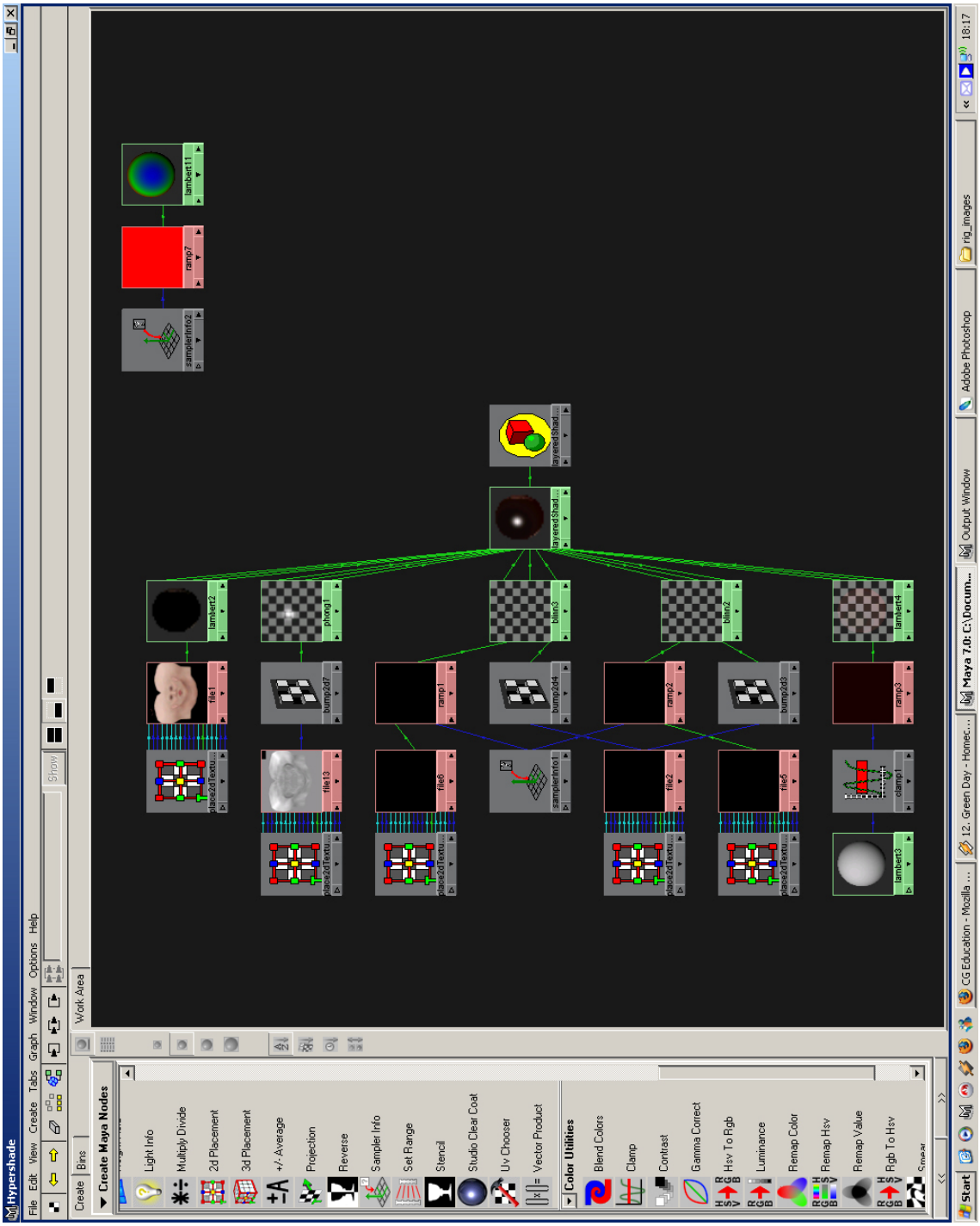
8. Appendices



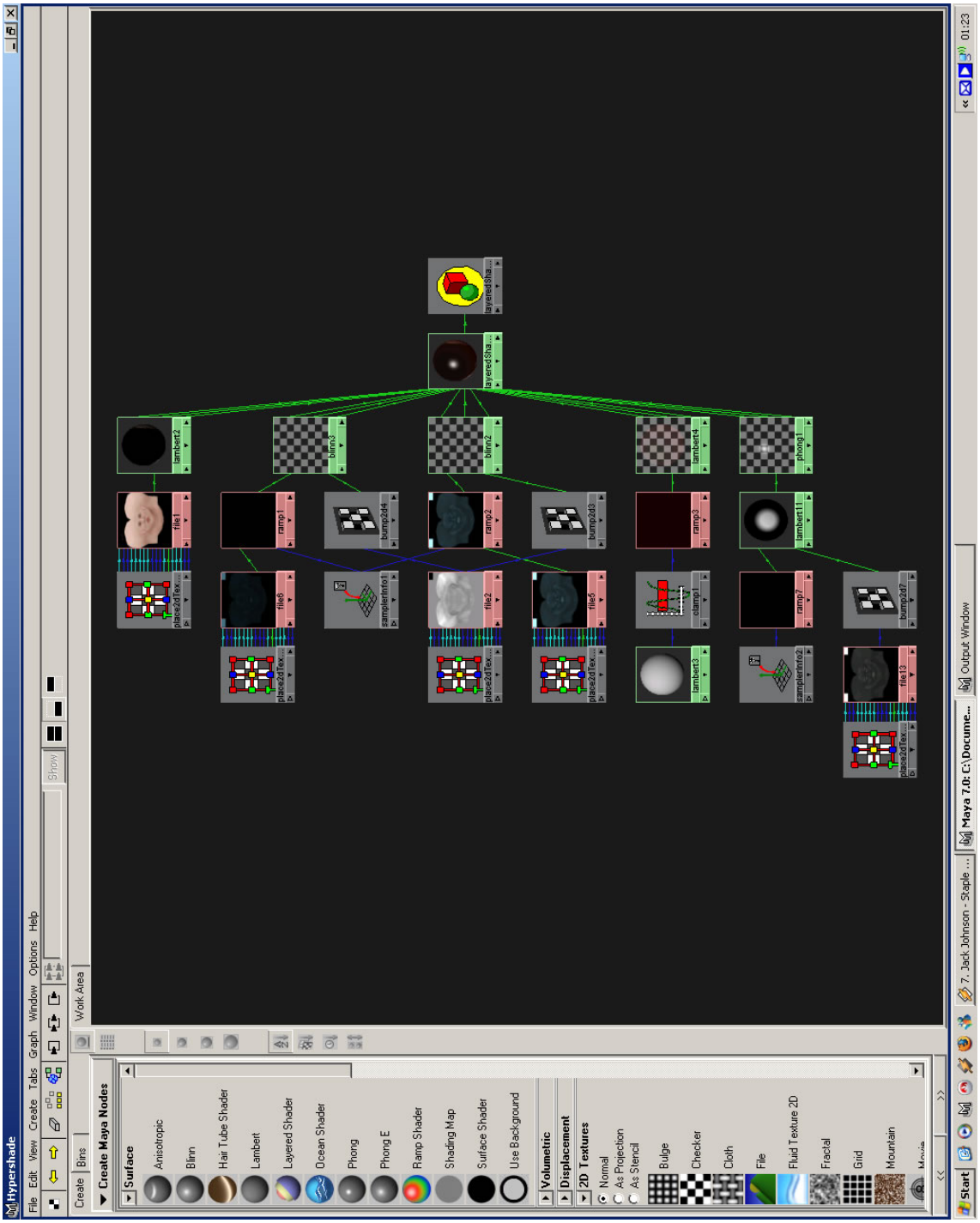
Appendix 1 – Skin texture with bumped specular (Figure 2)



Appendix 2 – Head model network with wet layer added directly on top (Figure 3)



Appendix 3 – Head model with facingRatio shader (Figure 5)



Appendix 4 – Head model network with facing ratio wet layer added (Figure 6)

9. References

[1] Blevins, N., Technical Director, Pixar Animation Studios
http://www.neilblevins.com/cg_education/wet_materials/wet_materials.htm

[2] Scanline Flowline
http://www.flowlines.info/

[3] Gravesen, J., 3rd Year Student BACVA
http://www.jimmigravesen.com/

[4] Maya Documentation
(Location depends on your install of Maya – Press F1 in Maya to access)

[5] Birkett-Smith, M., 3rd Year Student BACVA
http://www.vertpusher.com/

[6] RealFlow UV wet maps
http://www.nextlimit.com/realflow/html/training/tutorials/Particle_interaction_Particles_Vs_Objects/wet_effect/wet_effect.html

10. Acknowledgements

Thanks go to Eike Anderson (main tutor), Adam Vanner (initial tutor), Matthew Birkett-Smith for his head model, Matthew Ovens for his help with MEL UIs and Marie Borgesson for her advice and encouragement.