Forest Generation with Maya and Renderman

David Basalla
a1479196
MSc Computer Animation
Masters Project

September 5, 2005

Contents

INTRODUCTION:

Creating and rendering digital environments covered in dense vegetation is a complex and challenging problem in computer graphics. This is due to the visual complexity demanded by such scenes. These sort of scenes are both computationally expensive to render and difficult to specify in existing software. Software such as Alias' Maya is usually designed to handle a few individual surfaces and will be not be able to handle complex forest scenes with acceptable interactivity. Therefore a solution will have to be found that does not rely on one software package and its standard features.

Aim:

The aim for this project is to create a pipeline which is based on production proven software solutions such Maya and Renderman to create vast jungle and forest environments. Ideally the resulting system could be applicable in a professional 3D visual effects pipeline.

PREVIOUS WORK:

Previous Research:

Deussen et al. (1998, ref. #1) produced a paper which outlines an implementation of a system that allows for modeling and rendering scenes with thousands of plants and trees. Their work is based on a multilevel pipeline, where the scene creation is broken down into several distinct tasks, which are terrain modeling, specification of plant population, modeling of individual plants and rendering of the entire scene. This pipeline allows for dealing with each task separately so that for example the plant distribution does not need to be concerned with individual tree models.

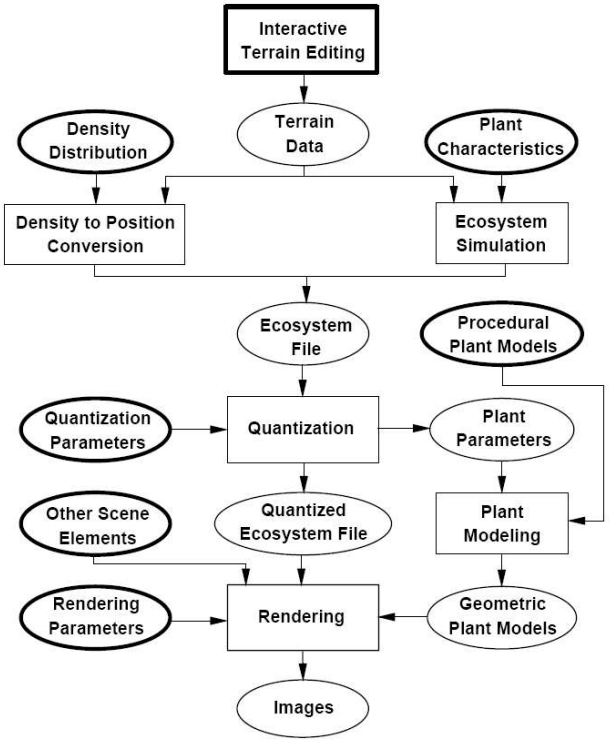

Figure #1: Architecture of Deussen's scene synthesis system (Deussen 1998, ref. #1)

Plant population can be specified explicitly for example by using a paint program or it can be specified procedurally by running a simulation of ecological plant type distribution. These ecological simulations are based on real-world experience findings and theories. Eventually they produce a map of type and location for various plants. Individual tree models were based on L-systems.

In order to maximise performance and to minimise memory requirements, approximate instancing is used. This technique is based on specific instancing where geometrically identical objects can be considered in terms of their unique transformations and one general geometrical model call. This means that the geometrical data (for example in terms of polygons) only needs to be stored once in memory. Approximate instancing applies the same concept to geometric models that are similar but not actually identical.

This system presents an efficient approach to the problem in terms of ease of use and in terms of render speed and memory allocation. However this system relies mostly on proprietary software that was specifically designed by the research team and therefore makes it not feasible to use for this project.

Commercial Software Solutions:

Apart from various research papers, a number of commercial software programs exist which aid in creating plants and trees. These range from specific tree modeling to general scene solutions.

The recently released landscape editor *Vue 5 Infinite* provides the most spectacular results to date, allowing the user to create vast and complex ecosystems scene that contain up to billions of polygons. Its technology has been hailed as major breakthrough in the field and is of yet unequaled by any other software package (Broomfield 2005, ref. #7). Incidentally, renders from Vue 5 were part of the motivation to undertake this project.

Vue 5 Infinite uses a rough visual feedback within the program *(Figure 2.1)* so it is not too computationally demanding. The ecosystems themselves are applied to a surface as a shader and allow for setting density and different types of trees. The tree models are accessed from a library, which also contains other objects such as houses and other inorganic items. At render time the trees (or other specified objects) are instanced to create scenes of vast complexity of millions to billions polygons. As part of research for this project, a trial version of *Vue 5 Infinite* was used to easily create a scene that contained 60 million polygons. The visual result was not as convincing as intended but this may have been due to the limited trial version and not enough knowledge about the software.

The drawback of this software is that it does not easily integrate into a pipeline that mainly consists of highend software packages such as Maya and Rendeman, as ecosystems have to be rendered within the software (Broomfield 2005, ref. #7).

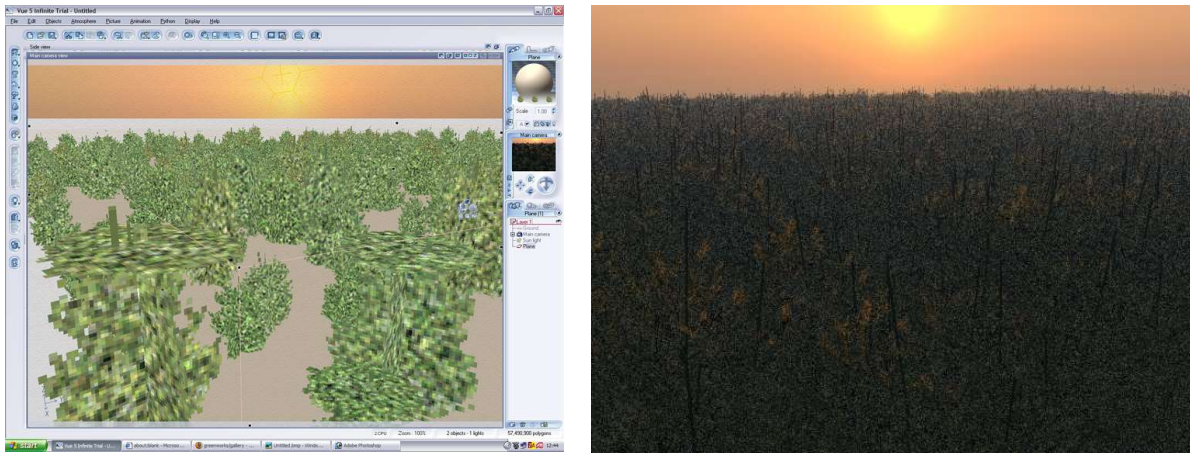

Figure #2: Vue 5 Infinite Visual Feedback (2.1) and rendered frame (2.2)

Apart from the macro solutions discussed above, there also exist plenty of individual tree creation programs. A lot of them rely on the use of L-systems. A three-dimensional L-system already bears heavy resemblance to a tree due to its branching structure. Therefore any computer graphics package that supports L-systems (such as Houdini) can be used to create basic tree models. Weber (1995, ref. #2) explains in his paper how L-systems can be used to great effect to create various forms of plants and trees.

X-frog, a program that is specifically dedicated to create trees and plants also uses L-Systems (ref. #11). It employs a node-based architecture in which the user can create trees by adding several nodes together (such as branch and leaf nodes). Deussen, whose work has been described earlier, is the co-founder of Greenworks, the company that is responsible for X-frog.

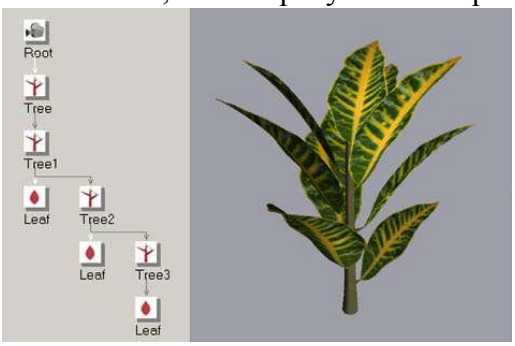

Figure #3: X-frogs node based plant modeling software(ref. #11)

Visual References:

In addition to looking at existing software solutions to the problem, a number of visual references of forest and jungle areas were analysed. These ranged from actual photos of jungles from around the world to various computer generated movies such as “The Incredibles” and “Madagascar” that contains extended jungle sequences. Attempting to research how Pixar and Dreamworks have created these scenes was difficult, as that sort of information does not seem to be readily available.

Figure #4: Visual Reference from Pixar's 'The Incredibles'

Figure #5: Visual Reference from Dreamworks' 'Madagascar'

TECHNICAL BACKGROUND:

In the first stages, render tests were done with Maya, its software renderer and the integrated mental ray renderer. While using Maya's Software Renderer, the rendering was not the problem but there was a huge interactivity drawback. With 4000 instances of a tree group, Maya took very long to respond, dedicating a lot of time to updating the scene. When using the integrated Mental Ray Renderer, the rendering time shot up, mostly because Maya needed to export all the geometry to a Mental Ray compliant geometry format. Considering that 4000 tree instances would be nowhere near enough to specify a full forest and were already pushing Maya to its limit, it became apparent that big complex tree landscapes would not be doable in Maya and a different approach would have to be used.

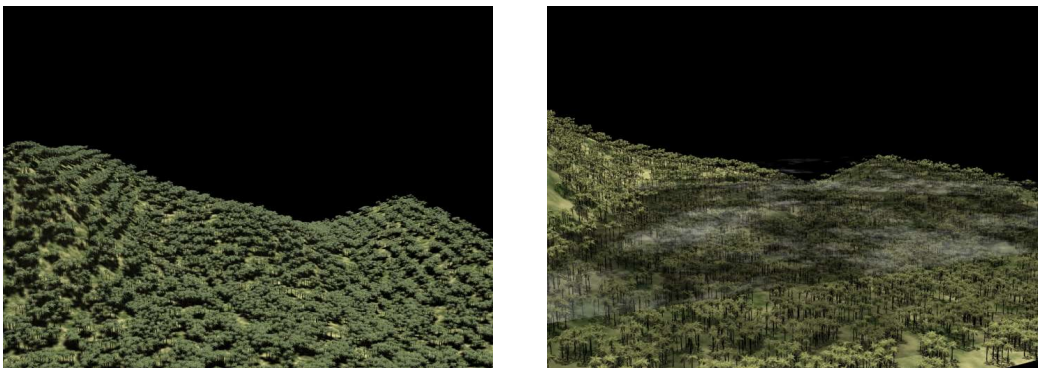

Figure #6: Early test renders with Maya Software Renderer (6.1) and Mental Ray (6.2)

Renderman is a file format for translating text commands into rendered images. There exist a number of Renderman-compliant renderers that accept Renderman files (also known as RIB files). Possibly the most well known of these is Pixar's Photorealistic Renderman (referred to as Prman in this report). It is designed to handle complexity at feature film level and has been production-proven in many feature films. More importantly it has been used by Pixar to render the jungles in 'The Incredibles' and therefore seems like a valid choice for this project. A few more examples (among many) of Prman's ability to handle complex scenes are 'Troy' and a 'A Bug's Life'. In 'Troy', Framestore CFC and The Moving Picture Company use Renderman to render vast crowds and an armada of battle ships (Moran 2004, ref. #9). In 'A Bug's Life' Pixar use Prman once more to create complex organic environment scenes.

Figure #7: Pixar's Prman used on 'Troy' (7.1) and 'A Bug'sLife' (7.2)

The advantage of using Renderman in regard to this project is that there is no graphical interface required that would act as a bottle neck for handling and displaying the multiple numbers of objects. Everything only exists as text files up until the point where Renderman translates the RIB text commands into a rendered image. Also Maya contains a RIB exporter as well as Renderman Artist Tools which both can be used to extract data from Maya and render it with Prman.

IMPLEMENTATION:

Since a high unit and polygon count was essential to rendering the forest, using Maya was not considered to be the best option for the whole forest generation and rendering process. Therefore it was decided to use Maya only as a starting point and to let Pixar's powerful Prman to handle the rendering of the geometry. To realise this project a multilevel modeling approach was used as described in Deussen's paper (1998, ref. #1), attempting to break down the forest creation process into several steps. This resulted in a simple pipeline based on a number of MEL and shell scripts.

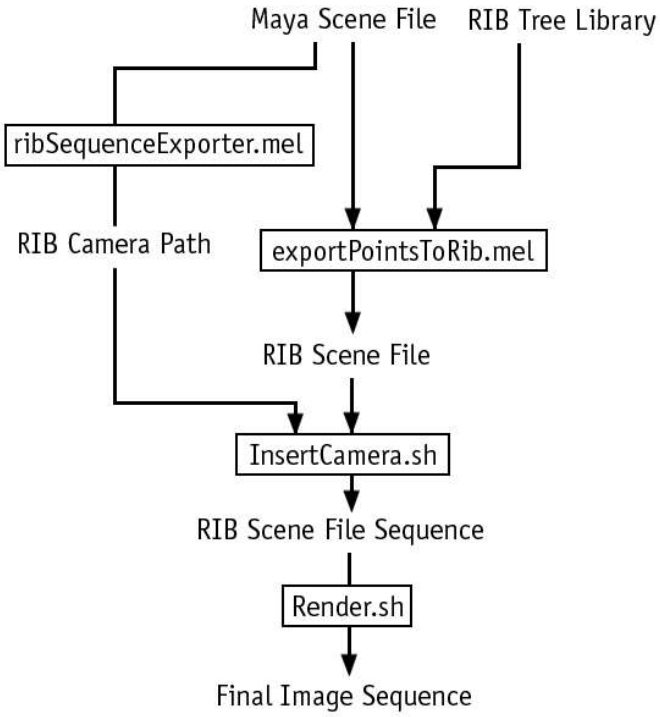

Figure #8: Forest generation architecture. Frames represent procedures

At first Maya is used to create the initial landscape surface. A MEL script then calculates a predefined number of positions on the surface and writes these into a file as RIB commands. The user also specifies which tree models to use from a library. The resulting RIB file contains multiple positions and calls to tree models. This RIB file is then passed to the main scene RIB file and can be rendered with Prman.

Terrain Specification:

For creating the terrain, any of Maya's NURBS editing tools can be used to create then terrain. For most examples, individual control vertices were manipulated in order to model the landscape. Other times, Maya's Paint Sculpt Tool was used in order model the landscape. Furthermore other approaches could be used to modify the landscape surface such as using height maps for increased realism. The current tool is limited to NURBS surfaces although integrating Polygonal and Subdivision Surfaces should not be difficult.

Type Maps:

Having created the landscape, the user can create presence, colour and partition maps within Maya. This can be done with the 3D Paint tool that allows the user to directly paint onto the surface. For convenience, these controls have been bundled in a MEL User Interface Window.

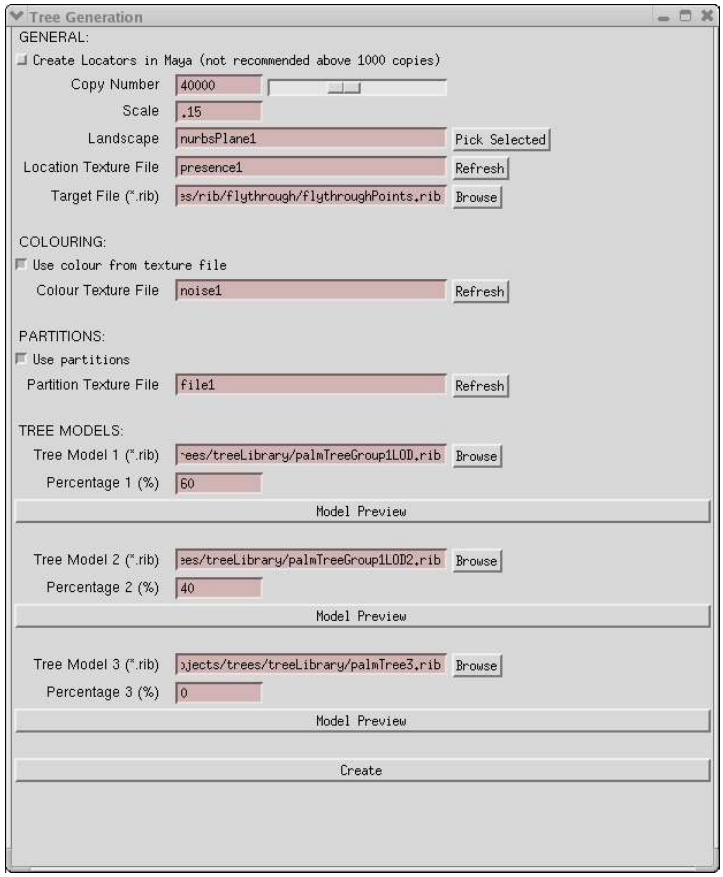

Figure #9: Maya user interface window for tree generation

A presence map is like a density map but it only supports 0% and 100% density (Figure 10.2). Trees will only be placed on white areas and omitted for black areas. This feature could be used for drawing specific patterns onto the surface.

For colour maps, any texture can be used. The map will then be used to colour the trees in that position (Figure 10.3). This allows the user to chose any of Maya's provided colour shaders or even paint his own colour map in Photoshop or using the 3D Paint tool to color the forest. For the test scenes, generally a noise or fractal texture was applied in order to break up the colour of the forest.

Partition maps can be used to break up the landscape into different regions. The user can draw different regions onto the landscape and then export tree positions only for desired regions (Figure 10.4). In an earlier version of the tool, this could be achieved by using different presence maps, but it proved to be tricky to align the maps so that no overlap would occur. Partition maps are more convenient and accurate to use. At the moment only 6 different regions are supported (primary and secondary RGB colours).

After having created the various maps, the user needs to specify which trees to use in the forest. Currently the user has a choice of three different object types as well as their percentage of

the whole population. The percentages are additive which means that if the first two tree models have a combined percentage equal to 100% the third specified tree model's percentage will equal 0%.

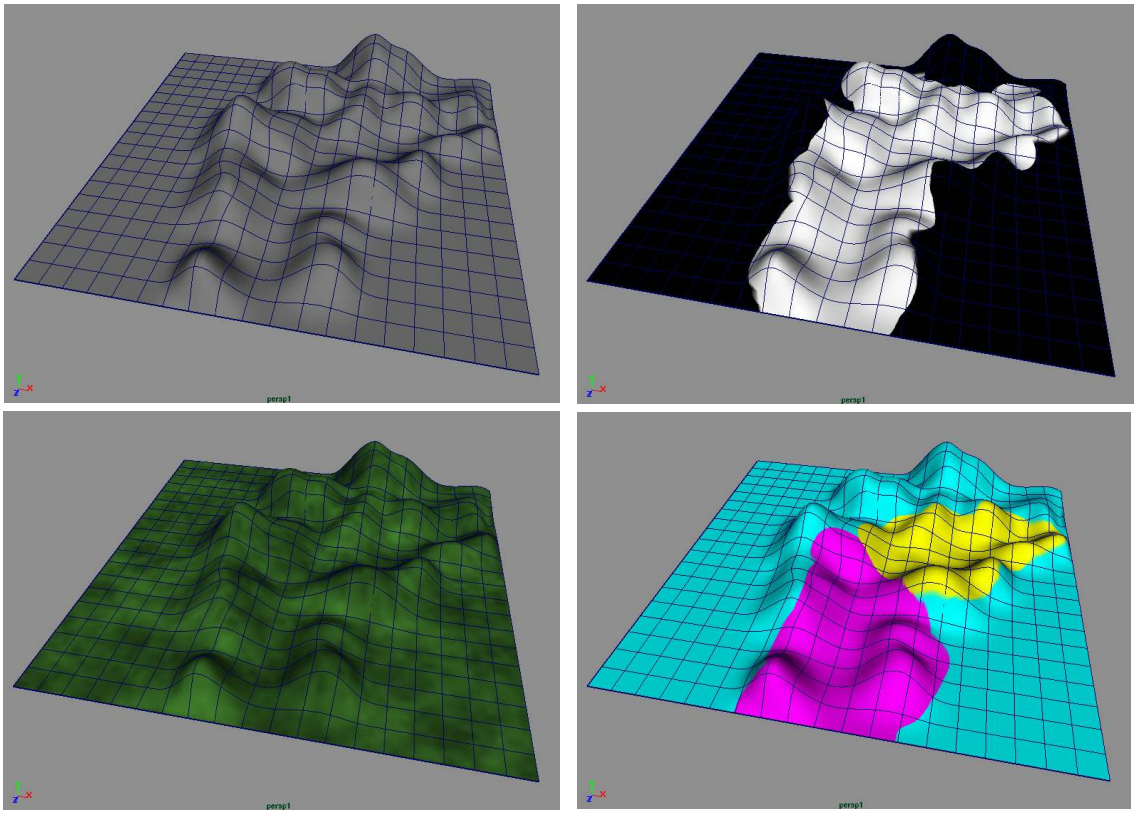

Figure #10: Different maps for a landscape surface - surface mesh (10.1), presence Map (10.2), colour map (10.3) & partition map (10.4)

Exporting tree transformation data to a RIB file:

Once the user has created the various type maps and specified the tree models, the MEL Script will randomly scatter points across the chosen surface. This is done creating a pointOnSurface Node in Maya and connecting it to the landscape surface. Then a random x and z value are passed to the node, which returns the appropriate y position on the surface. The resulting x,y and z information is the written into a rib file as part of a Renderman Translate command, including a Procedural DelayedReadArchive call to a tree model. If a partition map was specified, the MEL script will write the resulting rib commands into separate RIB files. The resulting files would often be several mega bytes big.

Tree distribution was done by randomly creating points across the surface. A number of previous research projects (Deussen 1998, Lane 1995) have implemented a realistic distribution pattern of trees based on biological patterns, but that sort of biological realism was not deemed essential for this implementation.

Maya's RIB Exporter:

To export the actual scene file including camera and surfaces, Maya's inherent RIB exporter was used. The exporter is very basic as it only exports the perspective camera and does not export the UV layout for surfaces. As explained in Javier Romero's report (2005, ref. #4) the first problem can be fixed by parenting the perspective camera to the specified camera by linking all transformations of the two cameras together. To work around the second problem, SLIM was used as it exports UV information for each surface.

After having exported the scene file from Maya, the user needs to add a Procedural DelayedReadArchive Call to the previously generated tree transformations file before the World Begin statement. This will ensure that Renderman loads the transformation data and object calls to the trees. The resulting RIB scene file would have the
following structure:

Object definitions of surface
WorldBegin
    Light definitions
    Procedural Delayed Read Archive to Tree Point Files
    Calls to Surface Objects
World End

Instancing:

Instancing is handled by Prman's 'Procedural DelayedReadArchive' Function. This call only loads a model into memory when its specified bounding box appears on screen. The Delayed Read Archive loads a model from the tree library specified by the user. Therefore many calls can be made to the same model which is memory efficient as the original model's scene description only exists once. The bounding box of the specific model is either set by the user or it can be determined by a scripted process. For this system, the bounding box is always set to 10 by default.

Tree Library:

To give the user a wider choice of tree models, several tree models had to be created. This was done by both manual modeling and by using Maya's Paint Effects.

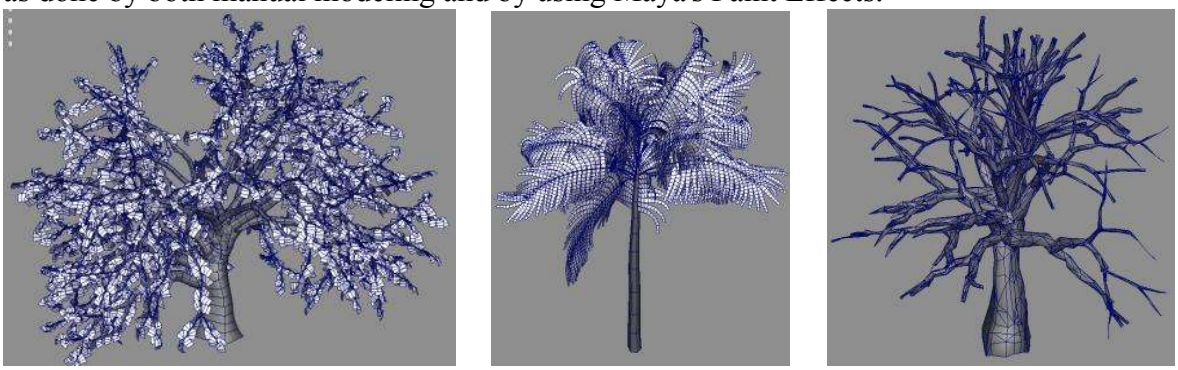

Figure #11: Models from Tree Library created with Maya Paint Effects

Maya Paint Effects contains a variety of high resolution tree models (Figure 11). These can be converted to polygonal models. However it is quite hard to simplify these to less than 10000 polygons and therefore Paint Effects trees are mainly used as foreground trees or as very small percentages of the entire tree population. The bulk of trees usually consisted of manually modeled tree models as these were easier to simplify (Figure 12). For the palm tree group seen below, a high level of detail model was modeled as well as a medium and a low level of detail version.

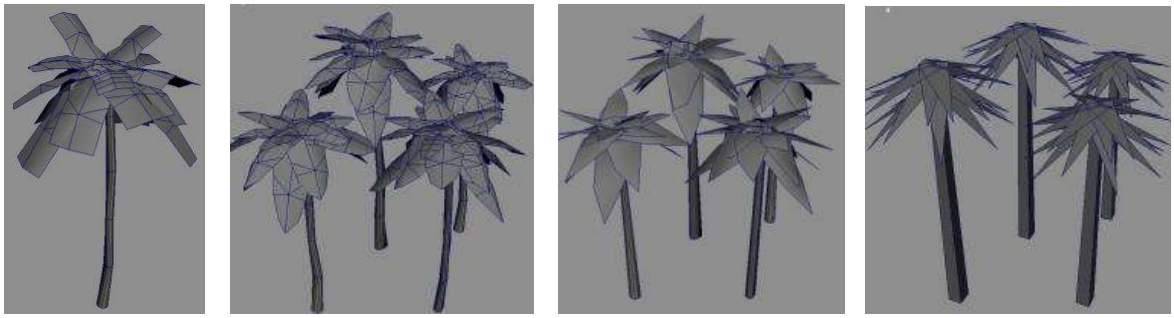

Figure #12: Models from Tree Library created manually, discrete levels of detail

Once the tree library was set up and contained a variety of models, it became difficult to extract information about the current tree model without opening its source Maya file. Therefore a 'Preview' option was implemented which allows the user to view a pre-rendered image of the model. This requires the user to create a representational image of a tree model if he wants to add it to the library. While it is not ideal, this helps to identify different tree models.

Shaders:

For shading the trees, a general shader was considered to be more convenient so that all trees would share one shader. The shader could be applied to the whole forest RIB file and therefore the general look could be adjusted easily. On the other hand the tree source RIB file could have contained a shader, but this would have made it difficult to apply a general shader such as the noise map from Maya. As the overall appearance was of primary concern, the first approach was used.

For specific shaders such as the shader on the rock in the test scene, shaders were first created with SLIM. Then the resulting source code for the shader was copied, adapted and applied to the object. While using SLIM shaders was a good starting point, for clarity and stability stand-alone RSL shaders were later implemented. When exporting a shader from SLIM, it contains all available attributes. While allowing for greater control, most of the attributes from the SLIM shaders were not strictly necessary and made the shader less transparent and harder to modify. Therefore a simple texture shader was implemented that supported image files as surface color and as opacity color. The support for opacity was useful for creating more detailed palm leaves where an alpha map would provide more detail. This shader would either determine the model's surface colour from a texture map or it could get its surface colour from a colour map (such as a noise map as discussed earlier) created within Maya.

Lighting:

Light types and positions are also extracted from Maya. Since spotlights in Maya do not have fall-off by default, the light intensity would have to be adjusted in the scene rib file. Good lighting was one of the harder things to achieve, mostly because it took a long time to render a frame with a forest scene.

In an attempt to facilitate lighting, a dome light rig in Maya was created. This was intended to create realistic lighting, based of environment textures. Even though it was easy enough to create in Maya, moving this to Renderman proved quite difficult. Also for the light dome to work properly, each light would have to have an associated shadow map. This would have taken a considerable amount of time considering that one shadowmap could take more than an hour to render. In the end a simple two point lighting setup was used with one key light for the main sun light and a fill light to lift the shadows.

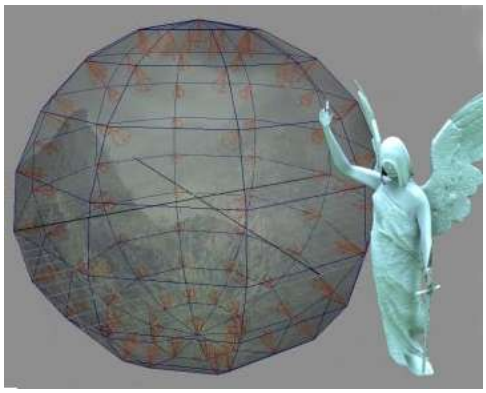

Figure #13: Dome light setup with rendered text object

Shadows:

Depth Map Shadows were used to create shadows. At first creating the shadow RIB files proved difficult, since the camera has to placed at the light's position. However, the program Cutter was used which automatically creates shadow rib files for a specified light source. Rendering the shadow maps took longer with increasing tree numbers, taking more than an hour for a forest of 100000 tree instances. Later on the scenes were broken down to use several lights instead of one main light in order to make changes faster. Also these shadow map calculations only had to be done for one frame and could be applied to consecutive frames of the same scene. Renderman also supports ray-traced shadows but these have not been thoroughly tested since they significantly increase render times and have to be calculated for each frame.

Shell Script:

In order to fly a camera through the scene, the camera path had to be exported from Maya. As Maya's RIB exporter is very basic, the whole scene had to be exported with a MEL script for each frame. Then a shell script was written to strip the rib files down so that only the camera transformation was left. This resulted in a sequence of simple RIB files which only contained a transform command. Another shell script was used to write this camera information into the according scene rib file. This was done by adding specific text tags into the original scene file. The shell script could therefore recognise the tag and write the camera information into the file at that point. Finally the output file name had to be changed for each frame which was handled by another shell script. For simplicity's sake, these three shell scripts were combined into one script.

Motion Blur:

In order for motion blur to work, each rib file needs to have the relevant information about the next rib file. Since motion blur was only required for the camera movement, the existing InsertCamera.sh shell script was adapted to insert the current and the next camera position into each rib file.

Analysis:

Several case studies were undertaken in parallel to developing the forest generation system once the fundamental architecture had been created. This helped the development process by clarifying problems and shortcomings in the code. For instance the partition map feature was only added after it became evident that using many different presence maps to act as layers was too inaccurate and convoluted. The following case studies aim to demonstrate the capabilities of the developed pipeline.

Case study #1:

The first scene contained a Maya Paint Effects tree model in the foreground and the background was created with the forest generation tool. The foreground object was lit and rendered with Maya's integrated Mental Ray renderer. The same lighting was then exported to Renderman, and the developed MEL scripts were used to create a jungle background with some river paths traversing through it. The final scene contains 48000 tree instances and roughly 48 million polygons. A noise colour map was used to break up the regularity of the generated forest. For this example only manually modeled trees were used. Some post processing in Shake had to be done to correct the colour, include a distance visibility fall-off and motion blur.

Figure #14: Final render of First Test Scene

The final visual result was successful in that it conveyed the feeling of a dense jungle. Improvements could be made to the foreground tree, as currently its animation is too subtle to be noticed. Also this scene is fairly simplistic, as the camera looks onto the forest at almost a perpendicular angle. The distance variation to the camera is very small, so that no level of detail change needs to occur. Render times per frame were roughly 15 minutes.

Case study #2:

This scene is more complex as it contains a close-up foreground and a distant background. The scene was first modeled in Maya with NURBS surfaces. The rock in the foreground left was shaded with SLIM for reference. The resulting surface and displacement shaders were then exported as SL files and simplified in order to make modifications more easily. A number of different presence maps were painted so that foreground, middle ground and background could be edited separately. The foreground layer contains mostly Paint Effects Trees and high resolution manually-made models. The middle ground uses mostly manually-made models of high and medium resolution as well a few Paint Effects Trees to break up the regularity. The background plane uses only medium and low resolution tree models. The background plane also does not have a shadowmap applied as it would not make a difference visually. Finally some post-processing in Shake was used to add some more mountains in the distance, add a sky and correct the colours.

Figure #15: Final render of Second Test Scene

The visual result is satisfying although the tree layer in the foreground does not look quite convincing. However the great complexity gives the impression of a chaotic and dense jungle. The final scene contains 68000 tree instances which translates into roughly 50 million polygons. When batch rendering this scene, severe aliasing appeared in the background areas. However adjusting the pixel samples to 8 and setting the shading rate to 0.25 reduced most of it.
This increased the render time per frame from 20 to 25 minutes.

Case study #3:

The third test scene was designed to use Motion Blur. This required a larger scene, so that the camera could move through the environment at a rapid speed. Therefore partitions were used in order to manage the greater complexity. This scene contains 130000 tree instances and roughly 105 million polygons. In order to keep the polygon numbers down, only manually-made models of high, medium and low resolutions were used. This greater complexity became a problem when rendering out shadowmaps. Previously one shadow depth map was always used for the whole scene. However in this scene, a shadow map could take longer than 2 hours to render which made it hard to make changes to the forest once the shadowmap had been rendered. Therefore the scene was lit by two lights, each casting a shadowmap of different tree regions. This allowed for faster shadow map render times.

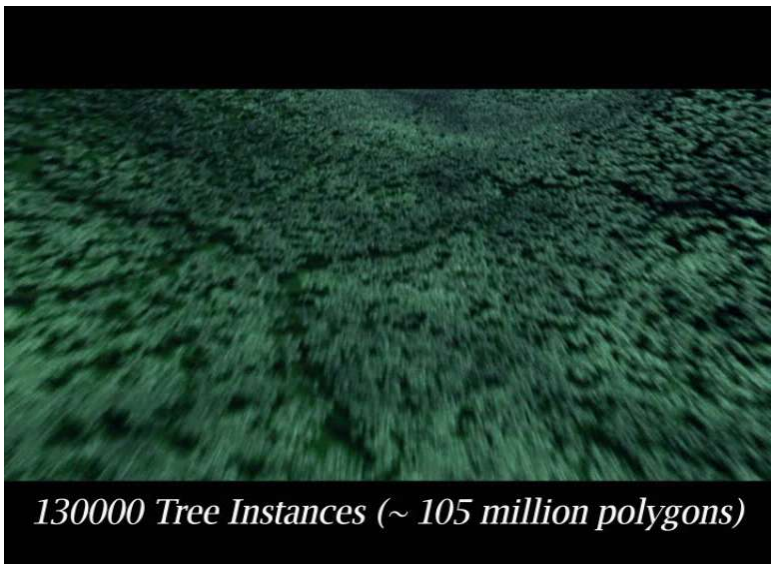

Figure #16: Final render of Third Test Scene

As previously discussed the InsertCamera.sh script had to be altered in order to insert the current and the next camera position into the rib files. Some post processing was done in Shake. The final render of this scene is not as refined as the previous two renders but it serves the purpose of showing motion blur in action. Also the amount of instances in this scene were the most instances in the whole project and generally worked well apart from having to split the shadow maps. Render times per frame ranged from 30 to 40 minutes. The increased render time compared to the other two case studies is caused by the additional Motion Blur.

Bugs:

Currently there are two bugs which have not been fixed yet. Firstly, the option to create locators in Maya to display where objects are placed is flawed. It creates locators but does not translate them to the calculated position. This is due to the fact that in Maya locators do not behave like normal objects and require more complex command calls in order to create and translate them.

Secondly, when partitions are being used, the MEL script will generate file names as intended where the file names take the specified output file name as a prefix and add the colour of the partition (eg. PointDataMAGENTA.rib). However it also generates an empty file name with just the specified output filename (eg. PointData.rib). This is trivial issue which does not hinder the actual forest generation process but it can be misleading.

Ramifications:

Currently there are a number of things which could be improved. However the developed tool set is easy to adapt and open-ended due to its multi-level pipeline and could be extended to account for any of the issues discussed below.

Lack of tree animation:

At the moment, there is no tree animation implemented. This is not a major issue for wide shots, as most of the test scenes have been. However when the camera is closer to the trees, the lack of animation becomes evident. In order to fix this ramification, looping animation cycles would have to be created for each tree and stored in the library. When creating the final animation rib files, the animated frames would have to be inserted into the rib files per frame. A further issue would be that shadow maps would have to be calculated per frame. As they are the biggest bottle neck in the system, this could increase render times to extreme lengths and a new shadow generation approach would have to be investigated.

No procedural level of detail implementation:

As of yet there is no procedural level of detail implementation. Instead this was done mostly manually by using different partitions. Renderman does include a Level of Detail call which can switch between differently detailed models depending on their distance to the camera and the screen space they take up. Furthermore a texture level of detail could be used as well. However creating different levels of detail for complex tree structures is a difficult problem and can not be solved by simply reducing the number of polygons in a tree model since its resulting topology could change significantly.

Simple tree shading:

At the current stage the individual tree shading is not very well developed, as the main concern was to achieve a convincing look for whole forests. However the Paint Effects tree shaders and textures could easily be converted to Renderman shaders.

Due to the ramifications mentioned above, the current system is not ideal for close up shots. If all these issues were resolved, this system would allow for cameras to focus on an individual tree in a forest and could then zoom out to display a whole forest. At the moment the system is more suited to display forest scenes from a distance such as demonstrated in the example case studies.

Slow lighting feedback:

As mentioned before, for lighting scenes the feedback was very slow and it was sometimes difficult to get a satisfactory result. This is partly due to the nature of the project. It could be fixed by creating a rough previsualisation of lighting conditions such as seen in Vue 5 Infinite, but this would be beyond the time limits of this project.

Uneven UV distribution:

As the tree placement is based on NURBs surfaces and therefore fixed UV coordinates, this can lead to uneven distribution of trees across the surface (Figure 17). Especially when control vertices are pulled out to an extreme, this leads to an uneven UV distribution and stretching which can be seen when a texture is applied. In future it would be useful to include polygonal landscapes

as well, where the UV layout can be edited according to topology. Alternatively, a more complex algorithm would have to be implemented that calculates the relative size of each uv parametric square and compensates the copy number of trees accordingly.

Figure #17: Uneven UV distribution for NURBS surface

Other Improvements:

Apart from working on the issues presented above, it would be interesting to see whether this project was feasible using a free Renderman-compliant renderer such as 3Delight, Angel or Aqsis. This would make the work presented here relevant for people or companies who can not afford to render with Pixar's Prman. Also it would be interesting to tackle the problem as a shader approach such as employed in *Vue 5 Infinite*.

CONCLUSION:

The developed system for generating and rendering forests satisfies the original aim of the project in that it allows the user to generate and render complex forest scenes. The system is able to generate scenes containing more than 100000 tree instances and 100 million polygons and thereby far succeeds the object management and rendering capabilities of Maya. The render time is acceptable although further tweaks could probably reduce it even more.

The system is stable and has been tested on three case studies as well as a number of previous test renders. By allowing for separate partitions to be exported, it makes easier to generate and edit vast forest scenes. Through the use of different tree models and colour maps, the regularity of the forest is broken which results in a more organic and chaotic look.

Essentially the developed tool can be used as a general instancing tool and is not limited to creating forests, although it has been adapted for that purpose. As an example, one could use models of houses to create a city. Another possibility could be a crowd of people spread over a large area. There would only have to be an appropriate rule set for distributing the models across the surface. Since forests and jungles are quite randomly distributed anyway, this was not as much of an issue as it would have been for creating a city. Unfortunately time constraints did allow a further investigation of generating urban environments with this pipeline.

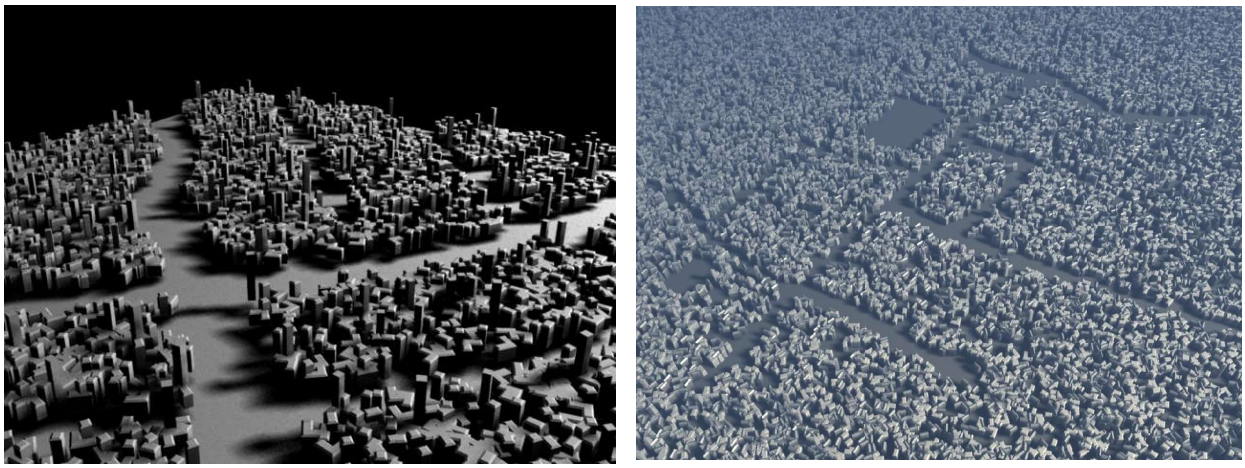

Figure #18: Examples of using forest generation system to create urban environments

Acknowledgements:

I would like to thank Hannes Ricklefs for introducing me to Cutter and for explaining how to use it properly.
I would also like to acknowledge Javier Romero for his previous work on tree rendering pipeline using Renderman.

References:

1.) DEUSSEN et al., 1998. Realistic modeling and rendering of plant ecosystems *In: Proceedings of the 25nd annual conference on Computer graphics and interactive techniques ,* pp. 275 - 286

2.) LANE, B. & PRUSINKIEWICZ, P., 2002. Generating spatial distributions for multilevel models of plant communities. *In: Proceedings of Graphics Interface 2002* (Calgary, Alberta, May 27−29, 2002), pp. 69 − 80.

3.) WEBER, J. & PENN, J., 1995. Creation and Rendering of Realistic Trees. *In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques,* pp. 119 - 128

4.) ROMERO, J., 2005. *Realistic modeling and rendering of complex foliage* Term 3 Animation Project Report, NCCA at Bournemouth University

5.) APODACA , A. & GRITZ, L., 2000. *Advanced Renderman.* San Diego: Academic Press

6.) STEPHENSON, I., 2003. *Essential RenderMan fast.* London: Springer

7.) BROOMFIELD, M., 2005. Review: Vue 5 Infinite. *3D World,* July 2005, pp. 90 – 91

8.) NVIDIA, 2005. *Technical Report - Nature Scene.* Santa Clara. Available from: http://developer.nvidia.com/object/nature_scene.html

9.) MORAN, P., 2004. *VFXTalk.com Interviews Framestore-CFC Compositors & TD's for 'Troy'* [online]. Available from: http://www.vfxtalk.com/forum/showthread.php?t=2523

10.) KESSON, M., *Cutter Text Editor* [online]. Available from http://fundza.com/

11.) *What is X-frog* [online]. Available from: http://www.xfrogdownloads.com/greenwebNew/products/productStart.htm

Appendix - More Test Renders: