

The Virtual Kiwi Project



Shareef I. Mostafa
smos012@ec.auckland.ac.nz



Supervisor: Burkhard Wünsche

Computer Science Department
University of Auckland
New Zealand

July - November 2002

Abstract: In this work I model a realistic kiwi bird that walks through a forest using the commercial software package 3D Studio Max. As part of this work I also present an overview of 3D modeling and animation techniques and discuss their merits in the context of this project. The kiwi was created using box and free form modeling techniques and brought to life using multi-level texture maps. The feathers represented a significant challenge and were implemented using a particle system plug-in really designed to model hair. Giving the kiwi some basic movement was achieved using forward and inverse kinematics and skin modifiers. The kiwi was constrained to an internal artificial bone structure and animated through the forest using basic key-framing techniques.



1 Introduction

Significant advances in hardware and software in recent years have brought virtual reality ever closer to fruition. Creating complex virtual worlds poses a number of significant challenges including effective terrain, vegetation, and wildlife modeling, precipitation simulation, physically-based modeling, and even AI techniques for animal behavior control. This project addresses a particularly interesting topic, the creation of virtual creatures, and more specifically the kiwi bird. The kiwi, a bird unique to New Zealand, represents a particularly interesting challenge because of the current limitations in modeling and animation of realistic feathers.

As part of this project I explore popular modeling techniques, current trends in feather generation, and various animation techniques in detail. The ultimate goal of this project is the modeling and animation of a kiwi bird through a forest. This project has a wide variety of practical applications from advertising and tourism to education and even game development. Also exciting future work in physically-based modeling and artificial intelligence make this project take on even more significance as future researches will be able to test their systems on my kiwi model.

This project set out to achieve several ambitious goals that include researching various modeling, animation, and feather rendering techniques, and using this research along with the software package 3D Studio Max to build an autonomous, realistic kiwi that walks his way through a forest. The main focus of this project was research into aspects of 3D modeling, current trends in feather generation, and achieving a realistic kiwi model. Animation techniques were assessed and discussed, however 3D animation was not the primary focus of this project and is subsequently discussed in less detail, unfortunately this was strictly due to time limitations on this project.

1.1 Background

The Graphics Group (GG) at the University of Auckland is a research group within the Computer Science Department. The GG is currently working on a major project, *Virtual Tramping in New Zealand*, which aims to realistically portray the wilderness and rainforests of New Zealand along with local wildlife and vegetation. This project ultimately envisions users begin able to explore lush sub-tropical rainforests and a diverse range of wildlife in stunningly vivid detail. This area of research covers a broad range of challenges.

These challenges include efficient CLOD terrain representation and traversal algorithms; CLOD referring to continuous level of detail. Current algorithms

including Lindstrom-Koller and the ROAM algorithm are inept at handling the problem effectively [19]. Another challenge resides in the determination of reliable height fields that are consistent with other obstacles including rivers, lakes, and appropriate sea levels [19]. Also, more difficulties reside in realistically and efficiently "vegetating" the terrain through 3D texturing [19]. This would include a wide variety of texture mapping techniques that would produce breathtakingly realistic flowers, trees, snow-capped mountains, and even beaches. A final research interest in Terrain Visualization is the simulation of precipitation to yield river flows and snow fall and physically-based modeling of rivers and waterfalls using simulation data [19].

Another research interest of the Graphics Group and tied directly to this project is the Virtual Creature. Autonomous virtual creatures are a great research challenge covering a wide array of topics from physically-based modeling to AI for various levels of behavioral control. At the University of Auckland, MSc student Robin Otte has researched physically-based modeling and animation which can be used for constructing skeleton models of synthetic creatures which adhere to the laws of physics. This system could be combined with virtual brains to control the creature's muscular system and ultimately its movement [19]. This system could in theory be added to any virtual creature and this project represents the first significant step forward in creating a virtual creature that will provide the testing ground for this future work.

2 Literature Review

3D Modeling and Animation techniques play a large role in a variety of diverse areas such as animated films, games, education, and even tourism to name just a few. With advances in recent years, greater and greater realism in computer graphics has been achieved, yet there are still many hurdles to be overcome. Current 3D modeling techniques including polygon and subdivision modeling [1,2,3] have produced very impressive models at the cost, however, of geometric complexity. Other modeling techniques including splines [6,7,20] and Metaballs[21] appear to provide the user with greater control, and give a greater degree of scalability, however their drawbacks appear to lie in the often time consuming nature the user must endure.

A classic problem encountered in computer graphics is creating realistic hair, fur, and feathers and is such a challenge because of the large number of strands, their flexibility and complex interactions, and the subtle and intricate shadows and reflections of each strand. Recent research in [9] has successfully created realistic feathers using parametric L-systems and bi-directional texture functions. While research in [10,11] has used Bezier curves to create far less spectacular results, especially in [11] where an overly complex methodology appears to have hindered their final results. Also, as seen in [12], a particle system can be used to generate realistic feathers using a number of segments rendered as triangle strips that produce surprisingly good results.

Animation techniques including inverse and forward kinematics as seen in [14,15,16] are discussed along with their respective advantages and disadvantages. Adding an artificial bone structure as well as key-framing techniques are discussed that allow the animation of any object using interpolation between key-frames based on the internal constraints of the artificial bone structure.

In this project a number of these modeling techniques are used to effectively model the kiwi. A particle system is also used to create the kiwi's feathers quite effectively, and along with basic key-framing and inverse kinematics the kiwi is able to walk through a forest with some basic movement. An overview of techniques, approaches, and results are presented in the subsequent sections commencing with the software package used solely in this project, 3D Studio Max.

3 3D Studio Max™ R3.1

3D Studio Max is a commercial 3D software package used extensively in the games and movie industry. Traditionally, 3DS Max has enjoyed a strong reputation as a leader in 3D game modeling, but recently has also come to play a strong role in special effects for film and television. 3D Studio Max, the world's most widely-used 3D modeling, animation, and rendering software, contains the essential high-productivity tools required for creating eye-catching film and television animation, cutting-edge games, and distinct design visualizations.

3DS Max is the premiere choice of game developers, known for its proven reliability, extensibility, and rich toolset for creating real-time and pre-rendered games and cinematics. From blockbuster films to hit television shows, 3DS Max is the world's most widely used professional 3D software used for sophisticated and high-impact film and television animation and effects. 3DS Max offers designers and engineers professional high-quality results with its advanced animation & rendering tools for visualizing industrial, mechanical, and architectural designs. Games such as Command and Conquer: Renegade, Diablo II, and Lord of the Rings and movies such as XMen 2, Minority Report, and The Mummy Returns clearly show that 3DS Max is a heavyweight when it comes to 3D software packages.

This project was completed using 3D Studio Max R3.1, released in 1999, and was the latest version I could get my hands on. They currently are shipping Release 5, however due to University budget constraints and at a cost of over \$10,000 US a copy, Release 3.1 it was.

4 Modeling

Modeling was a primary focus of this project and is accordingly dealt with in detail in this section. I start by introducing some of the various modeling techniques predominately used in 3D modeling and animation. These techniques range from Polygon and Subdivision modeling to Spline, NURBS, and Metaball Modeling. A brief overview of these topics are discussed in the next section followed by several sections detailing thoroughly the techniques I employed in creating my kiwi using 3D Studio Max.

4.1 Modeling Techniques

Polygon Modeling refers to a branch of modeling in which objects are composed of a number of individual polygons, together forming a polygon mesh that defines the object. Box modeling is a common polygon modeling technique where the modeling begins with a simple box, usually with a low polygon count for efficient

rendering [3]. This box, made up of a number of polygons, then undergoes surface transformations at the modeler's discretion to carve it into any desired shape necessary. One of the major advantages of polygon modeling is its simplicity and ability to model very complex and even organic objects accurately and effectively. However, the strengths of polygonal modeling also become one of its weaknesses. One of the drawbacks to polygonal models is that at the polygon level, their surfaces are flat, and in order to make an object smoother, you have to increase the number of polygons that make up the object. This can sometimes drastically increase the model's complexity and therefore increase rendering time. Polygon Modeling is often used in conjunction with Subdivision Modeling, discussed next, which allows the creation of extremely smooth surfaces.

Subdivision Modeling is a procedure for creating smooth models while keeping the total polygon count at a low level. Subdivision defines a smooth surface as the limit of a sequence of successive refinements and has several advantages including scalability, uniformity, and its ability to be applied to virtually any topology. Subdivision Modeling is achieved via a MeshSmooth modifier in 3D Studio Max and is applied to polygon meshes. The main advantage of subdivision surfaces over splines and NURBS is of the mesh can have any topology, this is indeed a very powerful notion.

Spline Modeling involves the use of splines, a mathematical structure defining a curve that is derived from several control points. One of the biggest advantages of spline models is that they are resolution independent. Resolution independence, as described in [20], is best exemplified with its polygon modeling counterpart. When you zoom into a polygon model the polygons get bigger and bigger, thus details can look blurred if up close, and has the opposite effect when you zoom out. However, with a spline model this doesn't occur because the curve is simply recalculated at the current resolution. A much more powerful and general curve can be created not through splines but from its big brother NURBS. NURBS, or non-uniform rational B-spline, is a mathematical representation that can accurately define geometric shapes ranging from a simple two-dimensional shapes to the most complex three-dimensional organic surface or solid [20]. They are a class of geometry that is easily manipulated by computers, allowing for a great deal of flexibility in modeling. A main advantage of NURBS is their ability to create smooth curves and surfaces quickly using only a small number of control points.

Metaballs are an alternative to parametric methods and make use of implicit representations, where the surface is defined as the zero contour of a function of two or three variables [21]. Metaballs, also known as blobby objects, are a type of implicit modeling technique. A metaball, as described in [21], can be thought of as a particle surrounded by a density field where the density associated with the particle decreases directly with its distance from the particle. A surface is implied by taking an isosurface through this density field where the higher the isosurface value, the nearer it will be to the particle. A powerful consequence of metaballs is the way they may be combined to produce very smooth blendings [21]. Metaballs and blobby objects strengths lie in operations such as blending and transformations, however their weaknesses appear to lie in their computational complexity.

4.2 Polygon and Subdivision Modeling

The kiwi and the terrain were modeled using polygon and subdivision techniques introduced earlier. The kiwi was appropriately broken down into subparts

including its head, body, beak, feet, and claws which are discussed in sections 5.2.1 and 5.2.2. The terrain was similarly modeled using polygon modeling techniques and is discussed in 5.2.3. The kiwi's eyes were modeled using Spline Modeling techniques and are discussed in detail in section 5.3.

4.2.1 The Head & Body



Figure 1: Snapshots of both kiwi models used during the modeling process.

The first step that needs to be taken in any form of modeling is obtaining accurate and detailed images from various perspectives of the object you wish to model. This is an extremely important step that cannot be stressed enough, for your 3D model will only ever be as good as your images you have to work with. I used two small kiwi models I had purchased and took multiple digital photographs of them from the left, top, front, and a perspective angle as seen in Figure 1. The next step that simplifies matters is to ensure that all pictures are the same size, this ensures your model remains in proportion later on.

The first step is to setup the images appropriately in 3DS Max by setting the background image in the Front viewport, under Views Menu. The viewports here refer to the respective camera perspective, Front viewport is directly in front of the object, Left viewport is the view from the left of the object. Once setting the background image to the appropriate kiwi image, as seen in Figure 2, the fields *Match Bitmap*, *Lock/Zoom Pan*, and *Active Only* need to be set. These ensure the image file maintains its original dimensions, the image doesn't move when you zoom or pan around, and only applies the background to the currently selected viewport respectively. I then applied the same setup to the Left Viewport this time using the appropriate left view image of my Kiwi, see in Figure 2.

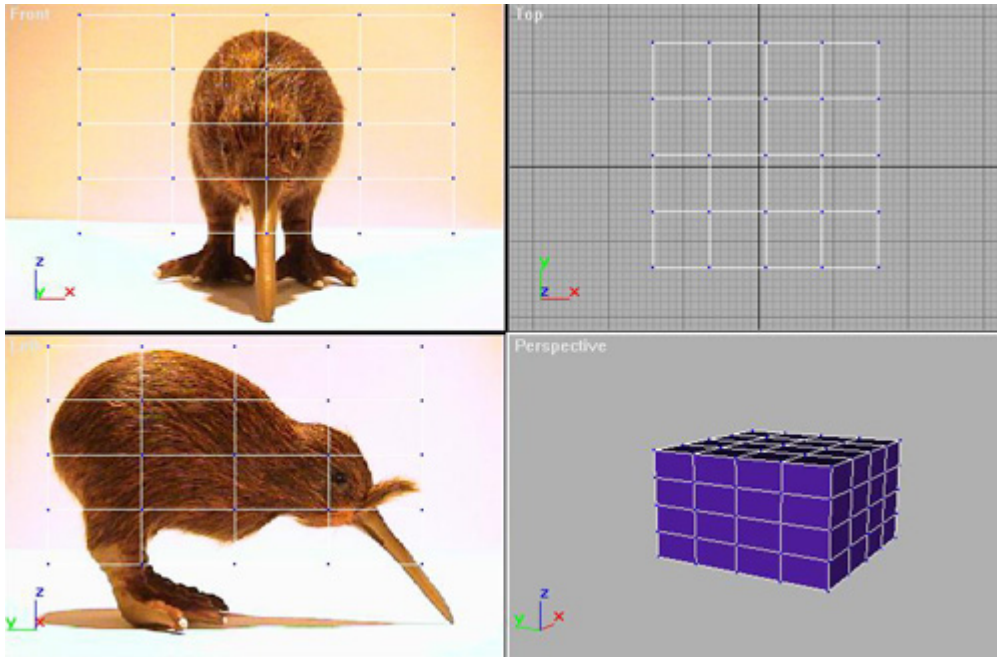


Figure 2: Modeling the Kiwi using Box Modeling Techniques.

I started using a box modeling technique I learned of in [3] and proceeded to create a box in the left viewport with a length, width, and height of 100 and segment lengths of 4 as seen in Figure 2. A box is created using the **Create**→**Geometry**→**Standard Primitive**→**Box** panel found on the far right of the screen. I then proceeded to line up my box in all viewports so that it overlapped my images. I then converted my box to an editable mesh, this is simply done by right clicking the box and selecting **Convert to Editable Mesh**. This now allows the box to be modified at various subobject levels ranging from *vertex*, *edge*, *face*, *polygon*, to *element*. Selecting *vertex* allows the modification of the box at the vertex level and by selecting individual or multiple vertices makes it possible to move vertices around and perform other operations as desired. I took the approach to first work with the left viewport and edit all the outer vertices so that they formed the outline of the kiwi. However, I started running into serious problems with vertices not lining up and it required an extensive amount of time to manually manipulate every vertex into its correct position. It was at this point I decided there must be a better way to go about things, and a series of ideas gave birth to my own modeling system which I aptly named "Sphere Modeling".

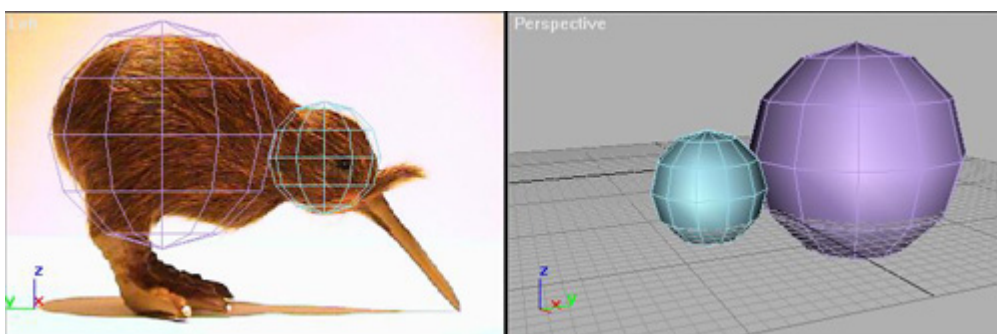


Figure 3: Sphere Modeling.

Sphere Modeling, as the name suggests, uses spheres to model objects. This approach makes much more sense since the majority of natural occurring life on Earth does not possess straight, sharp edges that run in perfectly parallel lines. Humans, animals, and more specifically the kiwi have bodies that are an intricate system of curves with no severe, or defined straight lines, rather they are very round and overall very smooth. So it only makes sense that any modeling approach would take the object being modeled into account, so as for this project Box Modeling is now officially out. Clearly though, Box Modeling may be more appropriate for other more man-made objects and has many other suitable applications, it is ultimately a choice for the modeler.

So with my new modeling approach in hand I went about the problem from scratch. I created two spheres using the command sequence **Create**→**Geometry**→**Standard Primitive**→**Sphere**, to represent the body and the head. The body sphere had a radius of 90 and a segment size of 12, while the second head sphere had an initially smaller radius and I later adjusted it as necessary so that it fit tightly around the kiwi head in both the background images. I then used the Left and Top viewports to orientate the spheres into their correct positions. I then proceeded, as earlier in the Box example, to convert the spheres to an editable mesh. This is done through right clicking the sphere and selecting **Convert to Editable Mesh**. This should result in what is seen in Figure 3.

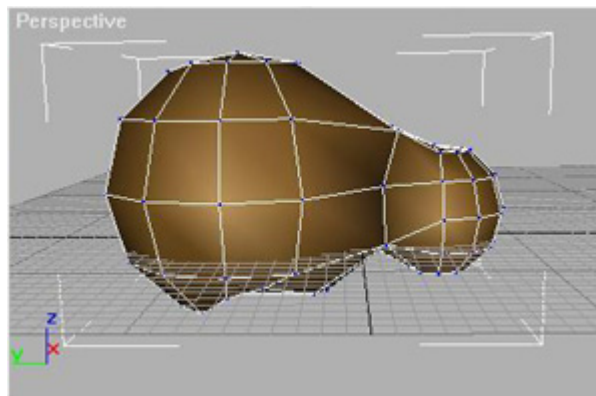


Figure 4: Combining Head and Body.

I then proceeded to connect the two spheres so that there was a smooth transition between the head and body. This was done by connecting individual vertices between the head and the body. Once again, this is done by selecting the sphere, and specifically the vertices through **Modify**→**Sub-Object**→**Vertex**. Now, I proceed to select the appropriate vertices on the head and body and performed the **Collapse Selected** command, simply found by right clicking after you have your vertices selected. This operation combines the selected vertices into one basically taking the centroid of all vertices, hence the faces become one. I continued to do this for all vertices between the head and body spheres that were not connected until I had the right overall shape of a neck. I then proceeded to individually alter certain vertices and edges between the head and body to make the transition smoother. This was done by selecting the various sub-object types, including vertices, edges, and faces and adjusting their positions until I was satisfied with the overall shape. I also

used various operations including **Create**→**Vertex** and **Weld/Break** operations to connect vertices, add vertices, and smooth out the result, the results of this can be seen in Figure 4.

At was at this stage in my project when I happened to come across a small kiwi in a souvenir store that appeared much more realistic in shape than my existing model, the second model seen earlier in Figure 1. I purchased my new model and set out to adjust my existing model to give it a more natural shape. Once again, I used sub-object operations at the vertex level to manipulate many vertices in both the head, shrinking it, and the body flattening it. I also used the Uniform Scale operation, found by right clicking over the object. This took away the appearance of a too rounded appearance and gave a more natural body shape.

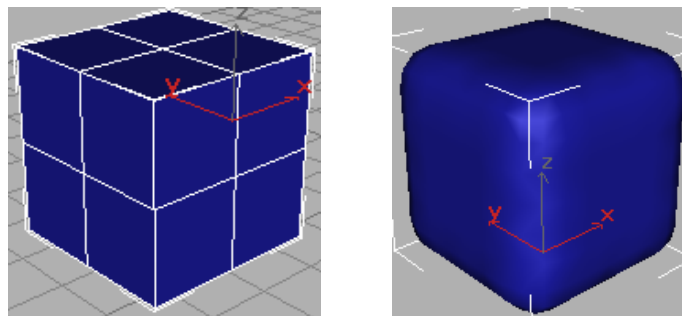


Figure 5: Applying MeshSmooth Modifier to Cube using NURMS output

The final step in modeling the body and head is applying subdivision modeling techniques discussed earlier, completed in 3DS Max using the smoothing modifier MeshSmooth. MeshSmooth, found under **Modifiers**→**MeshSmooth** modifier, smoothes geometry by adding faces at corners and along edges, and has the effect of rounding corners and edges as if they had been filed smooth, as seen in Figure 5. The MeshSmooth modifier results in an extra face being generated for every vertex and edge [1]. The modifier provides several parameters of control, ones of particularly interest include MeshSmooth Type and SubDivision Amount. MeshSmooth Type provides 3 types of output: Classic, NURMS, and Quad Output. Classic produces three- and four-sided facets and is unchanged since earlier Max releases [1]. NURMS, Non-Uniform Rational MeshSmooth, produces an object similar to a NURBS object in that you can set different weights for each control vertex [1]. You can further control the object's shape by changing edge weights. The final type, Quad Output, produces only four-sided facets and through a *Strength* parameter allows the relaxation of the original vertices and the new edge vertices into the mesh [1]. In applying this modifier to my kiwi head and body spheres, I found the NURMS type the most visually effective leaving my kiwi with a smooth and evenly curved shape.

The SubDivision Amount group provides two parameters of interest, *Iterations* and *Smoothness*. Iterations sets the number of iterations used to smooth the mesh, where each iteration generates new faces using the vertices created from the previous iteration [1]. A word of caution is noted when setting the Iteration value. The number of vertices and faces in an object can increase as much as four times for each iteration, and this directly results in possible very long calculation times. From my experience I used a maximum of 3 iterations for objects and found the level of smoothing more than adequate. There is finally a *Smoothness* parameter that

determines how sharp a corner must be before faces are added to smooth it. Smoothness is calculated as the average angle of all edges connected to a vertex, a value of 0.0 prevents the creation of any faces, while a value of 1.0 adds faces to all vertices even if they lie on a plane [1].

4.2.2 The Beak & Feet

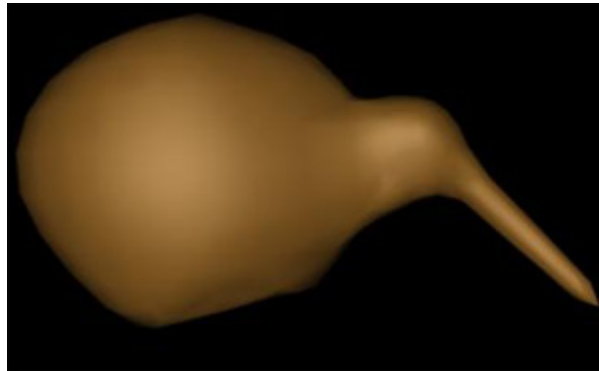


Figure 6: The Pointy Beak.

The beak and feet were both modeled using the general polygon mesh techniques. I initially modeled the beak using an Extrude operation to extrude two of the polygons at front of the kiwi's head to drag them into the appropriate shape. This at first appeared acceptable but later after applying smoothing operations looked much too sharp and far more like some sort of pointy sword and not much like a beak at all, as seen in Figure 6. I then proceeded to model the beak using a cylinder, **Create**→**Geometry**→**Standard Primitive**→**Cylinder**, and applied the bend and taper modifiers, **Modify**→**Modifiers**→**Bend & Taper**. For the bend modifier I used a 10 degree bend along the z-axis and then rotated the cylinder along the z-axis in the perspective viewport. Rotation, along with Movement and Scale, are simply found by right clicking on the object in question. I then applied the taper modifier and altered the *Amount* parameter, determining to what extent the cylinder was tapered until I was satisfied with the result visually. I then lined up the beak with the head so that it fit appropriately and proceeded to once again manipulate the beak at the vertex sub-object level to adjust it to fit more tightly to the head.

The kiwi's feet were modified using a pyramid object, **Create**→**Geometry**→**Standard Primitive**→**Pyramid**, with a width, depth, and height of 100 containing segments of 3,4,4 respectively. I then converted the object to a polygon mesh by performing the **Convert to Editable Mesh** operation. I then proceeded to select the appropriate sub-objects, including vertices and polygons, and went about the task of carving the pyramid into the appropriate shapes. I used the **Collapse Selected** command and the **Weld**→**Drag Selected Vertices to Target** command frequently to dissolve unnecessary vertices and combine particular vertices, the results of which can be seen in Figure 7b. I also used the **Cut**→**Polygon** to add several extra polygons to make the small back part of the foot. I then used **Extrude**→**Polygon** along with **Create**→**Polygon** to create the appropriate faces and make them look accurate.

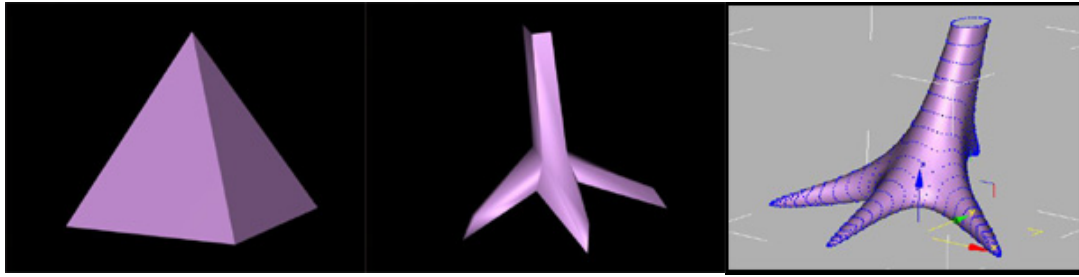


Figure 7: The feet were modeled from a simple Pyramid.

I now, as performed earlier on the body, applied a MeshSmooth modifier, found in the Main Toolbar, to the foot. Using a NURMS Output type with an *iteration* value of 3 and *smoothness* of 1 resulted in a very nice looking foot that is extremely smooth, as seen in Figure 7c; hard to believe it was once just a humble pyramid.

Once satisfied with the overall shape I used one of the better features of 3DS Max, the Mirror operation located on the Main Toolbar. This operation saves the user extensive time by mirroring any objects the user wishes. This allowed me to simply select my newly created right foot, apply Mirror and amazingly I now have a perfectly shaped left foot. I then simply selected both feet and moved them into relative position under the body and adjust them with through a uniform Scale operation, as seen in Figure 8. Now that we have a nice pair of feet we need to add some claws.

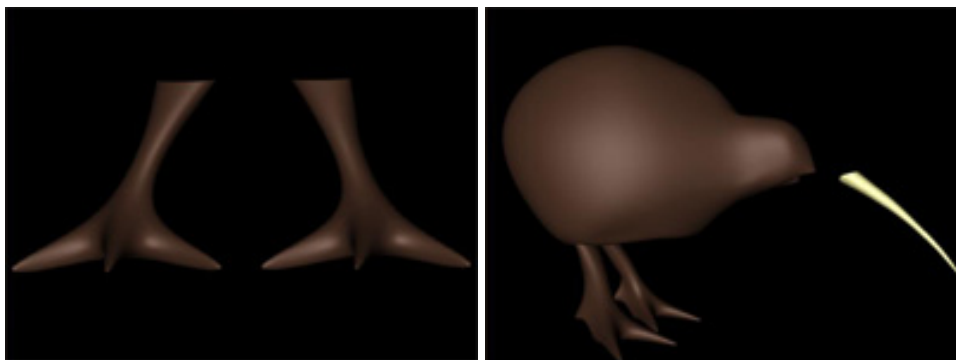


Figure 8: Both feet are created and positioned into place under the kiwi's body.

Claws were created using a Cone, **Create**→**Geometry**→**Standard Primitive**→**Cone**, with one radius of 0, the tip of the claw, and the second radius of 20 for the base of the claw. I then applied a Bend modifier, **Modify**→**Modifiers**→**Bend**, with a 45 degree bend around the z-axis, and rotated the claw so that it was upright. The claw was then converted to an editable mesh, **Convert to Editable Mesh** operation, and modified at the vertex, sub-object, level. Once I was satisfied with the overall shape I applied a MeshSmooth modifier with a NURMS output and an *iteration* value of 3. This resulted in a very smooth looking claw, seen in Figure 9, which was subsequently cloned twice, for the left and right toes. I then rotated each claw appropriately according to the toe it was going to be placed on. Once finished I mirrored all the claws over and placed them on the other foot.

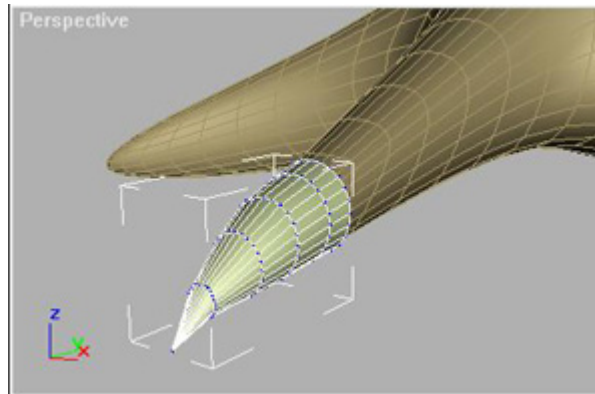


Figure 9: Claws were created using Polygon and Subdivision Modeling.

4.2.3 The Terrain

The terrain consists of a ground plane, a height elevation map, a sky, and a forest. I started by creating the ground plane using the **Create→Geometry→Patch Grids→Quad Patch** with a 500x500 plane with a segment size of 5. I then applied a "muddy" texture map to the plane to give it depth and a hint a realism. Texture mapping techniques are further discussed in 5.4. I then proceeded to create a mountainous terrain, learned of in [4], by creating another plane using **Create→Geometry→Patch Grids→Quad Patch**, and created a 200x200 plane with 10x10 segment size. I then created a terrain elevation map using Adobe Photoshop, this map was a 200x200 plain black image with some highlight points. Highlight points were created using an airbrush tool with very low opacity. Basically, the brighter the points in the elevation map, the more elevated these regions will be. Black regions represent ground level and all other gray values in between are interpolated to determine their height. This map can be seen in Figure 12. Once I completed my map in PhotoShop, I returned to Max where I applied a displace modifier to the mountain plane, **Modify→Modifiers→More →Displace**. I then adjusted the strength to 65 which basically rescales the elevated areas to 65% of their original value. After applying a simple "rocky" texture map, the result is an average looking mountainous region as seen in Figure10a. I say average, because I subsequently dropped the mountain terrain and opted for an Environment Map which proved to look much more realistic and was much more effective.

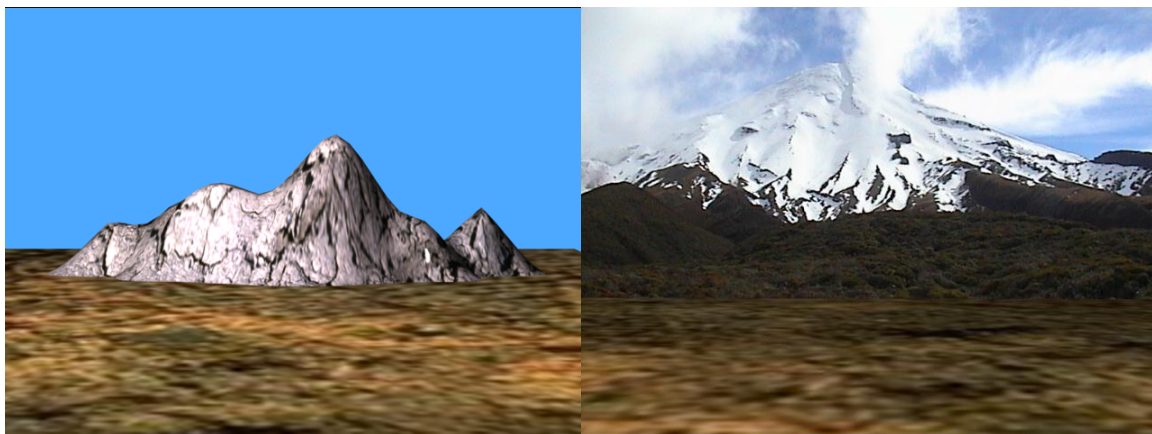


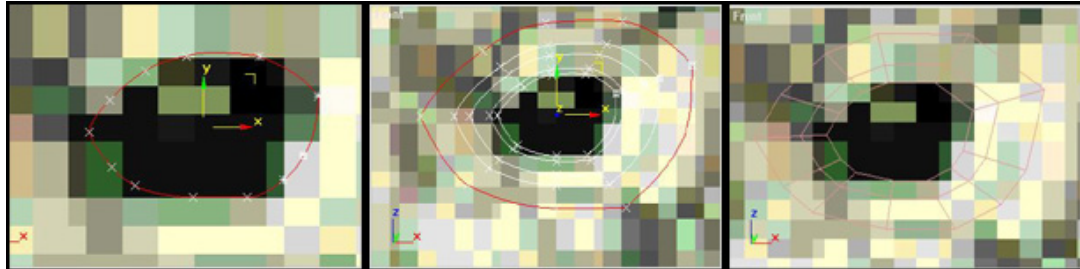
Figure 10: a) Mountain created using Elevation Map and Texture Mapping.
b) Great shot of Mt. Taranaki used as an Environment Map

An Environment Map as the name implies is a simple mapping to the global environment. It takes an image as a map and simply scales this image to fill the viewport, and is only seen once the scene is rendered. The Environment Map is found in the Main Toolbar under **Rendering**→**Environment**, and allows the user to select an appropriate image. I recently had the good fortune of being in the Taranaki region of New Zealand where I managed to capture a stunning shot of Mt. Taranaki, a beautifully picturesque snow-capped mountain. After applying an Environment Map using this photo I ended up with the picture in Figure 10b, which looks far more attractive than my earlier mountain. Now that I have my ground plane and a nice mountainous background we need to turn to the skies.

I came across an interesting tutorial in [5] detailing a realistic cloud animation and set out trying to implement a similar style of sky. However, after I completing the sky using geospheres and multiple layers of texture mapping, the results were far from satisfying. Instead I decided the sky background Figure 10b looked much better and decided to stick with that. Now that we have a nice terrain and an environment map, we need to fill in the terrain with some trees. The forest was handled through the plug-in ForestPack Lite which is discussed further detail in the section 7. I now turn the discussion to Spline modeling techniques that were used to model the kiwi's eyes.

4.3 Spline Modeling

Figure 11



a) Outline eye using a Spline. b) Scaled and Repeated Splines. c) Cross-Section Modifier

Spline Modeling was used to model the kiwi's eyes and follows ideas found in [6,7]. I started by zooming into the eye area of my background image in the front viewport, so that the kiwi's eye made up roughly 50% of my viewport. I created a Spline object using **Create**→**Shapes**→**Splines**→**NGon** with a parameter setting of 10 sides. By setting the NGon to 10 sides we are in essence creating a decagon. A decagon was chosen to give us more vertices to work with allowing the eye to be modeled more accurately. The next step is to adjust individual vertices, using **Modify**→**Edit Spline**→**Vertex**, of the spline so that they approximate the outline of the kiwi's eye, the idea is clearly shown in Fig 11a. Once satisfied with the overall shape, the next step is to create several more splines that will define the subtle curvature of the eye. I created 3 more spline objects that were increasingly larger in size. This was completed by scaling my currently selected spline while holding down the shift key, as seen in Figure 11b.

The next step is to apply two modifiers to my newly created splines, namely the CrossSection and Surface modifiers. The CrossSection modifier creates a "skin" across multiple splines. It works by taking 3D splines and connecting their vertices to form a skin [1]. The resulting object is another spline object that can be used with the Surface modifier to create a patch surface. The CrossSection modifier can build a skin across various shaped splines with different vertex counts and open/closed status. The more different the splines in vertex count and complexity, the more likely the skin will have discontinuities [1]. For the eye this isn't a problem because of the relative simplicity of the splines.

The Surface modifier generates a patch surface based on the contours of a spline network. A patch is created wherever the segments of the interwoven splines form a three- or four-sided polygon [1]. The Surface modifier and the CrossSection modifier, taken together, are referred to as Surface Tools. They allow for the creation of complex or organic surfaces which make up a large extent of natural occurring objects including the human face. One of the benefits to modeling using splines and Surface Tools is the ease of editing the model. At almost any stage of modeling, you can add a nostril, ear, limb or body by simply adding splines. This lends itself to a free-form approach to organic modeling, however from my research [1,2,3,7] and experience using 3DS Max, Sphere/Box modeling techniques used in conjunction with MeshSmooth modifiers results in much more efficient work and are far easier to work with to create low polygon objects that look just as good if not better than their spline counterparts. I now turn the discussion back to using these modifiers to create a realistic eye.

The next step is to apply the CrossSection modifier, **Modify**→**Modifiers**→**More**→**Cross Section**, with its default settings and you end up with the figure seen in Figure 11c. Next we apply the Surface modifier, **Modify**→**Modifiers**→**More**→**Surface**, with a *Topology* parameter of 0. Another parameter of interest is the *Threshold* parameter which determines the overall distance that is used to weld the vertices of the spline object. All vertices/vectors within the threshold of each other are treated as one [1]. *Topology* parameter determines how many steps are used between each vertex. The higher the step count, the smoother the curve you will get between vertices [1]. It is also worth noting that if the eye disappears after applying the modifier to ensure the *flip normals* parameter is selected, this is necessary because the modifier seems to, by default, have the normals pointing in the opposite direction to the viewport.

The next step involves a Bend modifier, **Modify**→**Modifiers**→**Bend**, to give the eye a more curved appearance. I bent the eye 25 degrees around the x-axis, and then used the **Convert to Editable Mesh** operation to convert my splines into a polygon mesh. I next edited the mesh at the vertex level and manipulated individual vertices to adjust the overall shape of the eye making it appear more real. I next applied a MeshSmooth modifier, **Modify**→**Modifiers**→**More**→**MeshSmooth**, using the NURMS output type and an *iteration* value of 3 resulting in the image seen in Figure 12a. Now that we have a nice eye shape we need to add an eyeball and a pupil.

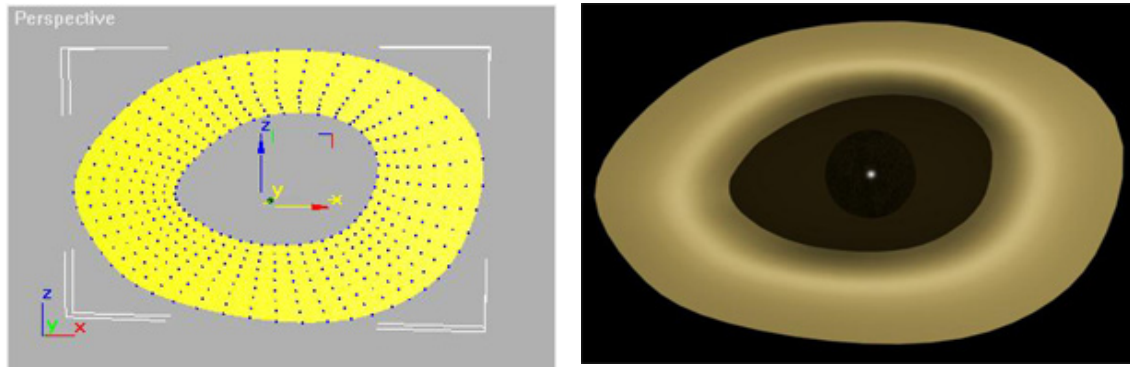


Figure 12

a) After applying MeshSmooth Modifier. b) Final Result after Texture Mapping

I created the actual eyeball using a geosphere, **Create**→**Geometry**→**Geosphere**, and proceeded to scale it down and applied a Stretch modifier to elongate it so it would fit behind the eye socket. The Stretch modifier, **Modify**→**Modifiers**→**More**→**Stretch**, contains two parameters of interest: *Stretch* and *Amplify*. *Stretch* parameter determines the scale factor that is applied to the selected stretch axis and the inverse of this scale factor is applied to the minor axes, while *Amplify* amplifies the scale factor that is applied to the minor axes. Once I had an eyeball I was satisfied with and fit tightly behind the eye, all that was left was the pupil. The pupil was created using a small geosphere which I also scaled and then moved into place slightly in front of the eyeball. I then positioned the eye appropriately along the side of the kiwi's head. I then had to adjust the eye using rotation to get it to stick closely with the kiwi's head and not expose any gaps. I then used the Mirror operation to make a mirror copy of the eye and then moved this copy over to the right side of the head, which I further adjusted. I then applied texture maps to pupil, eyeball, and eye and ended up with the image seen in Figure 12b. The texture mapping techniques used are discussed in detail in the next section.

4.4 Texture Mapping

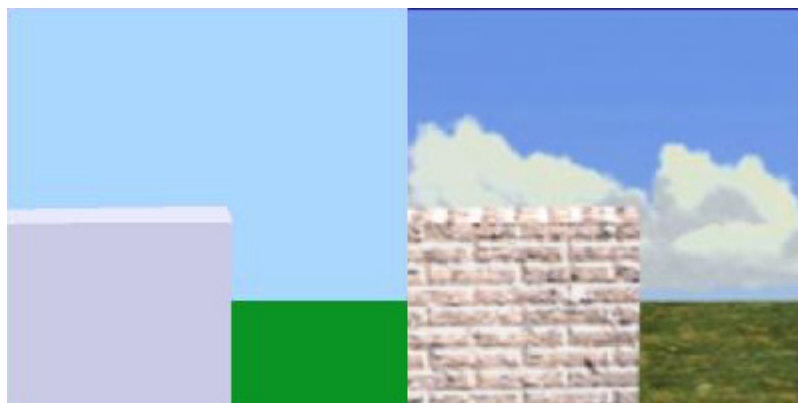


Figure 13: Demonstrating how texture maps add depth and realism to polygons [8].

Texture Mapping refers to the technique of applying an image, or "texture", to an object to give it added depth and feel, and a generally greater sense of realism. Texture maps are used because they provide the best compromise between geometric complexity and aesthetic acceptability. Generating fine detail, like that we see and take for granted everyday, increases modeling time and might reduce flexibility when modifying a model, and as a direct consequence lead to the idea of texture maps. Using texture maps and their associated mapping techniques is much cheaper than trying to replicate fine detail at the individual polygon level. A simple yet effective example of what texture mapping is all about is seen in Figure 13, where the image on the right portrays a brick wall, a lawn and the sky [8].

Texture mapping can be divided into two-dimensional and three-dimensional techniques. Two-dimensional techniques place a two-dimensional flat image onto an object using methods similar to pasting wallpaper onto an object [8]. While three-dimensional techniques are similar to carving a shape out of wood. There are various map shapes that determine how the texture is specifically mapped to the object. I will briefly discuss the four most common two-dimensional maps which include planar, cylinder, sphere, and box.

For a planar map, we take an (x,y,z) value from the object and project one of the components, this leaves us with a two-dimensional planar coordinate [8]. We use the planar coordinate to look up the color in the texture map. Planar projection is useful when only one side of an object needs to be mapped. It is also useful for obliquely mapping multiple sides, and for mapping two sides of a symmetrical object [1].

A second shape used in texture mapping is a cylinder. An (x,y,z) value is converted to cylindrical coordinates of $(r, \theta, height)$. For texture mapping, we are only interested in θ and the height. To find the color in two-dimensional texture map, θ is converted into an s-coordinate and height is converted into a t-coordinate where (s,t) are the image coordinates[8]. This wraps the two-dimensional texture map around the object

When using a sphere as the map shape, the (x,y,z) value of a point is converted into spherical coordinates. For purposes of texture mapping, we keep just the latitude and the longitude information. To find the color in the texture map, the latitude is converted into an s-coordinate and the longitude is converted into a t-coordinate [8]. Seam and mapping singularities occur at the top and bottom of the sphere where the bitmap edges meet at the sphere's poles [1]. Spherical mapping is useful for objects that are roughly spherical in shape.

Using a box as the map shape is similar to planar mapping. Instead of using one texture map, box mapping uses six; one each for the left, right, front, back, top and bottom sides of the object. To texture map the front and back sides, we eliminate the z-component of an object's point and use the remaining x and y-components to locate the color in the corresponding texture maps [8].

A major drawback to 2D texture mapping is the necessity of handling singularities, as mentioned in the sphere mapping where two singularities occur at the top and bottom of the sphere and in the cylinder. These drawbacks are some of the inefficiencies inherent in texture mapping, however their strengths far outweigh any other currently viable techniques. The next paragraphs explain how material properties and texture mapping were applied to the kiwi and the terrain.

4.4.1 The Beak & Feet

In 3D Studio Max, material properties and texture maps are applied through the Material Editor found under the Rendering tab. To texture map the beak I implemented a Multi-Layer shader that has a set of two specular highlight controls. The highlights are layered, letting you create complex highlights that are good for highly polished surfaces. Highlights in the Multi-Layer shader can be anisotropic. Anisotropy measures the difference between sizes of the highlight as seen from two perpendicular directions [1]. When anisotropy is 0, there is no difference at all. The highlight is circular, as in Blinn or Phong shading. When anisotropy is 100, the difference is at its maximum. In one direction the highlight is very sharp; in the other direction it is controlled solely by Glossiness.

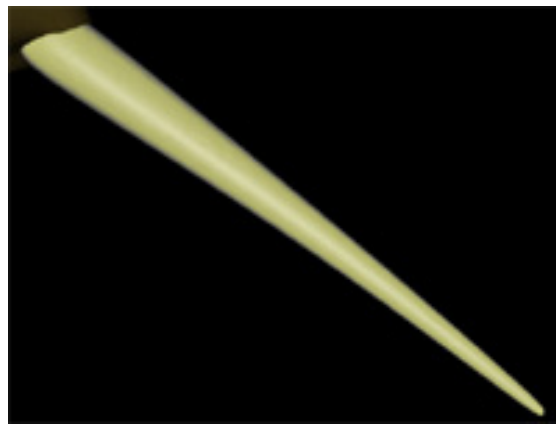


Figure 14: A Multi-Layer Texture Mapped beak.

For the beak I set a yellow/beige color for the diffuse component and a black for the ambient. I then set the first specular highlight to a light beige with *Level* of 30 and a *Glossiness* of 25, where *Level* represent specular intensity and *Glossiness* defines the size of the highlight. I then set the second specular highlight to a dark brown as seen in Figure 21, and proceeded to apply a bump map. A bump map, as the name suggest, creates bumps and dents in a texture and is often desired to mimic real life, examples include oranges, carpets, and beaches. I applied the bump map "Dent" with appropriate *strength* and *size* values so as to provide some fine dents in the beak. Finally, I applied a Diffuse Roughness map where the white pixels in the map increase roughness, black pixels reduce roughness to 0, and intermediate values adjust roughness accordingly. The overall results can clearly be seen in Figure 14.

4.4.2 The Eyes

For the eye I also used the Multi-Layer Rendering Shader to mix a hard specular and with a softer one. For the eyeball I used a Phong Shading which smooths the edges between faces and renders highlights realistically for shiny, regular surfaces. This shader interpolates intensities across a face based on the averaged face normals of adjacent faces [1]. It calculates the normal for every pixel of the face. Phong shading can accurately render bump, opacity, shininess, specular, and reflection maps [1].

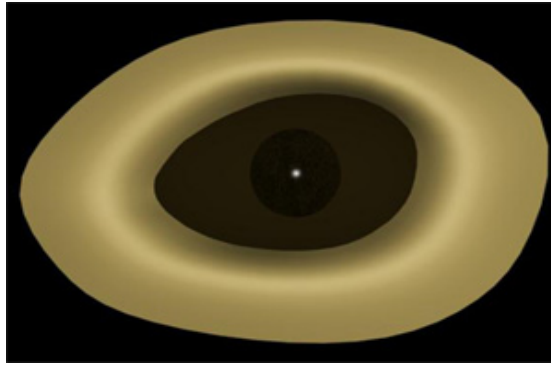


Figure 15: Final Result of Eye after Texture Mapping

For the pupil I used Blinn shading which appears visually quite similar to Phong probably because it is actually a subtle variation on Phong shading. The most noticeable difference is that highlights appear rounder. In general, you don't need to use the *Soften* parameter as often as you do with Phong shading. With Blinn shading, you can obtain highlights produced by light glancing off the surface at low angles [1]. These highlights are lost when you increase the value of *Soften* using Phong shading. I applied a very grainy texture map to the opacity channel which gives the pupil a faintly textured look and is visually appealing. I also applied a very strong specular highlight with a level of 100 and a glossiness level of 65 giving the eye a shiny white reflection adding realism and depth. I also set the opacity level to 65 so that the pupil isn't solid, and can be seen through slightly. The results of these shading methods can be seen in Figure 15.

4.4.3 The Terrain

The terrain was rendered using diffuse mapping, a texture map in the diffuse channel, to map a mountain like texture onto the surface. This is done by bringing up the Material Editor and setting the mountain plane's diffuse component to the texture map. The ground plane was texture mapped using the same procedure, however this time using a more dirt like ground texture. The results were seen earlier in Figure 10a. As mentioned earlier, I subsequently dropped the mountain terrain and opted for the Environment map.

5 Feathers

Modeling and rendering feathers as well as hair and fur effectively has traditionally represented one of the classical problems encountered in computer graphics. Modeling and rendering feather represents such a difficulty for several important reasons. Firstly, there are a lot of feathers to model, the sheer number of feathers required to model and render requires extremely efficient algorithms to make the task even computationally feasible. Secondly, each feather is flexible to some degree and this must be taken into account during animation by which algorithms must be able to handle this flexibility in a believable and extremely efficient manner. Once again this task is made much more challenging because of the large number of feathers found on birds. Thirdly, there are a number of complex interactions that occur between feathers naturally during a bird's flight or even while still, such as the

response to wind or other natural phenomena. Fourth and finally, creating realistic feathers represents a significant challenge because of the many subtle reflections and absorptions that occur on and between feathers making it computationally and algorithmically very expensive.

I will now discuss three of the latest techniques used in creating realistic feathers along with how effective their results are. The first system makes use of L-systems and a Bi-directional Texture Function (BTF) [9]. The second uses a biologically inspired Bezier system, as seen in [10,11]. Finally, I discuss the particle system, as seen in [12], used to create the kiwi's feathers and evaluate its respective benefits and drawbacks.

5.1 L-Systems & BTFs

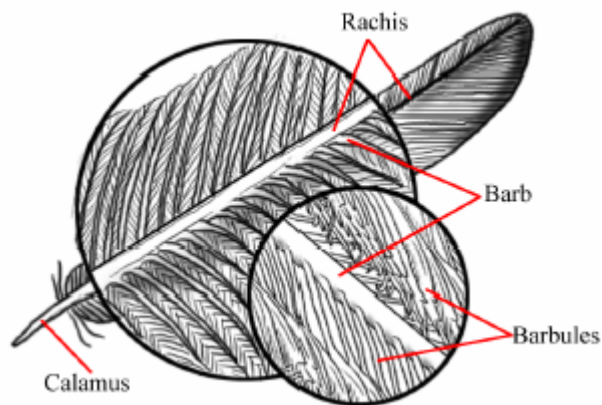


Figure 16: Feather Anatomy [9].



Figure 17: Eagle Created in [9].

In the SIGGRAPH 2002 paper: *Modeling and Rendering of Realistic Feathers*, the authors present a method of modeling feather geometry through the use of a parametric L-system [9]. The system allows the user to create a variety of feathers by simply changing a number of input parameters including the rachis, barb, and outline curves as seen in Figure 16. An L-system represents the development of a branching structure by productions [9]. A production replaces a predecessor module by several successor modules in a recursive fashion. The length of feather, density of the barbs and barbules are determined through the L-system modules as specified through the user input parameters. The rachis, barbs, and barbules are created in a recursive, iterative fashion where by the feather is generated in small units, piece-by-piece until the entire feather is created. The system also takes into account the random nature of individual feather barbs and barbules. The original system produced feathers that were a little too perfect, but in reality subtle variations and imperfections are what give feathers their unique and realistic appearance. The authors handle this by introducing external forces into their L-system, where by forces can break the interlocking nature of the barbules if the total external force is sufficiently strong enough. For each barb, the total external force is incremented by a random force and subsequently compared to the current force exerted by the barbules. If the total external force is greater, then the barb is rotated in a direction and by an

angle predetermined by a random number and a user specified parameter. This results in a much more realistic feather that exhibits the naturally occurring imperfections.

They proceed to render the feather using a bi-directional texture function or BTF, which allows for a very realistic portrayal of the finer details of a bird's feather. The Bi-directional function captures the mesostructure and the directional radiance distribution at each point on the feather surface. A BTF is a 6D function which uses the Position on the feather, the Viewing Direction, and the Lighting Direction to render the feather from all viewing and lighting settings. This rendering is done offline by a RayTracer which captures the intricate shadows, reflections, occlusions, and specularities in vividly realistic detail, as is evident in Figure 17.

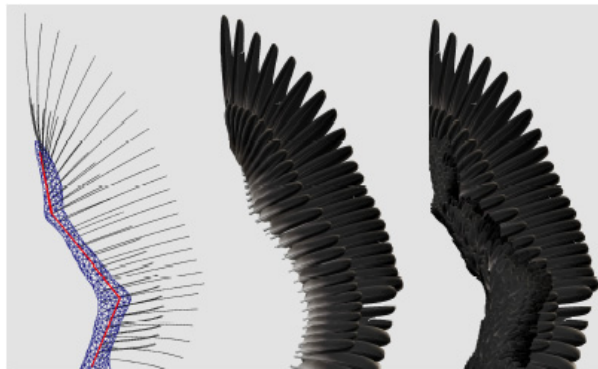


Figure 18: Skeleton and Various Feathers [9].



Figure 19: After Collision Detection [9].

Feathers are placed on the bird using a skeleton structure for the wings and the tail, as seen in Figure 18. The skeleton is simply a collection of polylines consisting of line segments where each polyline will ultimately have a feather placed upon it. The user specifies a number of key feathers of desired shape and type, and places eight of these key feathers onto the wings and another four onto the tail. The system then generates all other feathers based on these key feathers. This process of feather generation involves polygon retiling, feather interpolation, and collision detection. Polygon Retiling is needed because the vertices of a given polygon model usually can not be used directly as feather growing positions [9]. Feathers on different parts of a bird have different sizes, and small feathers need to grow densely in order to cover the bird's skin. Furthermore, feathers have a tendency to distribute evenly in regions of constant feather density [9]. Due to these tendencies, Turk's Algorithm is used to evenly distribute vertices over regions of constant density. After this retiling, the vertices of the polygon model are now the growing positions for the feathers. The next step involves determining all growing directions for all the feathers. Since all key feather growing directions have been specified by the user, all other feathers' growing directions are determined through interpolation. This simplified interpolation results in many feathers that inter-penetrate each other which is not desirable since real feathers do not exhibit this feature to such a degree. The authors rectify this problem using a recursive collision detection algorithm, where by all neighboring feathers are checked for a collision. If a collision is detected between these feathers, the growing direction at that vertex is altered slightly towards that of the surface normal. The algorithm recursively iterates this process altering all feather growing positions if collisions are detected, the final results can be seen in Figure 19.

The results of this paper are very impressive, and are by far the best feathers I have come across during my research. The user interface allows the user to create a large variety of feathers of different shape, size, and color and allows the user to completely cover any polygon mesh using a few simply defined key feathers. The eagle in Figure 17 contains around 3,500 feathers, and took the user only 30 minutes to specify the 50 key feathers. The rendering speed was around 1 minute per frame, which is relatively good when compared to other approaches, as seen in a particle system approach I implemented. Some limitations of this system include the lack of support for down feather rendering, oil-film interference, and iridescence which are an important characteristic of many birds including ducks and hummingbirds. Also the authors show an interest in animated birds and feathers, but appear to leave this as possible future work. Overall, the authors present an excellent approach to modeling and rendering of realistic feathers and their work represents the state-of-the-art in feather generation.

5.2 Bezier Curves

Modeling feathers using Bezier Curves is another technique that two recent papers have applied, in very similar fashion, to create realistic or perhaps not so realistic feathers.



Figure 20: The feather in the middle is the real feather [10].

Franco and Walter in their paper: *Modeling and Rendering of Individual Feathers*, present a simple approach to modeling feathers using Bezier Curves [10]. To model a single feather the user initially defines a cubic Bezier to define the rachis, and additionally two more Bezier curves to define the outline or vanes of the feather. From the rachis a variable number of barbs, each made up of four control points, are created at the user's discretion and along with a number of unique input parameters all other barbs are created. The authors create their feathers based on seven unique parameters including: Number of Barbs, Variation on Length of Barbs, Variation on Form of Barbs, Symmetry of Vanes, Uniformity of Barbs (form), Uniformity of Barbs (length), and Start of Second Segment. Most of these parameters are self-explanatory but to clarify some that are not. Variation on Length and Form of Barbs allows the

user to specify different lengths and shapes for each individual barb found on the feather respectively. Symmetry of Vanes allows the user to specify whether the feather is symmetric or asymmetric. Uniformity of Barbs in form and length allows the user to select whether the barbs will all be uniform in shape and size, or not if set to false. Finally, the Start of the Second Segment parameter allows the user to specify different parts of the rachis that will have different types of feathers found upon it. For example, it is possible to define uniform barbs for first segment of the rachis, and non-uniform for the second segment. These seven parameters were designed to allow the user to create a wide variety of feather shapes and size quickly and with relative ease, one example of the final results can be seen in Figure 20.

Once the feather structure has been built based on these seven parameters, the feather may be render in a non-photorealistic way (NPR) or can be rendered realistically using actual feather images as texture maps. For each feather a set of control points for the rachis and barbs is maintained and used to create line segments between these curves. The user is able to select the sample size from low resolution (few sample points) to high resolution (many sample points), the latter would obviously be more desirable for close-up shots while the former for long-distance shots where the finer details of the feathers aren't as important. The resulting two feathers seen in Figure 20 used 30 control points and it is quite hard to distinguish between the real feather and the synthetic ones. Overall, this paper takes a much more simplistic approach when compared to the previous L-system and BTF approach by Chen et al, and appears to be more of an introductory paper on creating feathers. That being said it still has created some nice feathers, and has significantly better results than the other Bezier inspired paper by Streit and Heidrich, as seen next. Areas that may be improved include: More System Automation, Possible Illumination Model, Covering a Bird's Body with Feathers. These goals seem to mimic what Chen et al have already achieved, and with great success as we have seen. Therefore, I believe that although Franco and Walters' approach is good one, it really appears to be lacking in many areas and represents more of a good start, rather than a completed or even nearly completed paper on generating feathers.

The second Bezier inspired feather generation paper: A Biologically-Parameterized Feather Model, by Streit and Heidrich, appeared recently in EUROGRAPHICS 2002 Number 3 [11]. However, I must admit I really am not sure how it even qualified to be printed with some of its results, seen below in Figure 21.

Figure 21:

Top Feather Coat: Bilinear interpolation of four key feathers generated from scans of Budgie feathers.

Bottom Feathers: an interpolation between two of the key feathers [11].

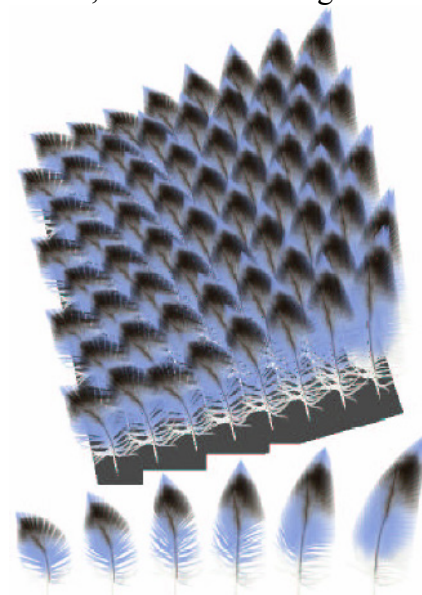




Figure 22: Left: Generated Hawk feather. Center: Flatbed scan of an actual Hawk feather. Right top: Generated Budgie feather. Right bottom: Flatbed scan of actual Budgie feather [11].

The approach taken in this paper is very similar to that of Franco and Walter, with some slight variations. In this paper, feathers are also modeled as a collection of Bezier curves, where by the user specifies key barbs and all other barbs are determined through interpolation. This paper initially set out with more ambitious goals than the previous paper since growing directions of feathers and generating a feather coat were part of its aims, while Franco and Walter were merely concerned with generating a single feather based on seven input parameters specified by the user.

In this paper, a feather is designed using a set of parameters that define the length and curvature of the rachis, length and angle of the barbs, and a set of key barbs for the vane curvature [11]. The curvature of all other barbs is then determined through interpolating all key barbs. Once the feather is created, we use a number of key feathers, again as seen in Figure 21, to create a feather coat where by all other feathers are determined through interpolation based on these key feathers.

The parameters for the rachis, barb, and vane curvature can be determined a number of ways. They may be determined artificially or through image processing techniques. Through edge detection and feature extraction algorithms the barb length and width can be determined along with other details concerning the rachis, vanes, and even key barbs can be automatically detected and then synthetically recreated. So once the overall structure is determined, either by the user or through automated image processing techniques, alpha blending, texturing, and generating the feather coat are handled. The authors apply an alpha blending technique, which I believe looks extremely poor and not at all realistic as seen below in Figure 22. The ends of the barbs seem to fade away too harshly and don't realistically portray a natural feather's appearance. No specific details are given about how they implemented the alpha blending, but it would appear a very heavy-handed approach was taken that simply decreased the opacity as the end of the barb was being reached. For rendering, the feather is drawn using a generalized cylinder for the geometry of the shaft, and polylines or triangle strips for the geometry of the barbs [11]. The rachis is also simply a Bezier curve scaled by the rachis length. Generating a feather coat is achieved by the authors using bilinear interpolation to blend the closest key feathers

found within the convex hull of these key feathers. The results of which are even worse than the individual feathers, early seen in Figure 21.

This paper clearly demonstrates that more ambitious goals do not necessarily result in a more favorable outcome. This is clearly seen when compared to the Franco and Walter paper which had much more humble goals but achieved them successfully. Also, Streit and Heidrichs' paper seem to give even more weight to the link between poor results and overly complicated methodologies. Often, as encountered in my research, some authors present simple ideas in a more obscure and complex way as to make them appear more intelligent than they really are, unfortunately, this appears to be the case for this paper. I now turn the discussion to a particle system that I used to create feathers that were ultimately placed on the polygon mesh of the kiwi bird discussed earlier.

5.3 Particle System Plug-in

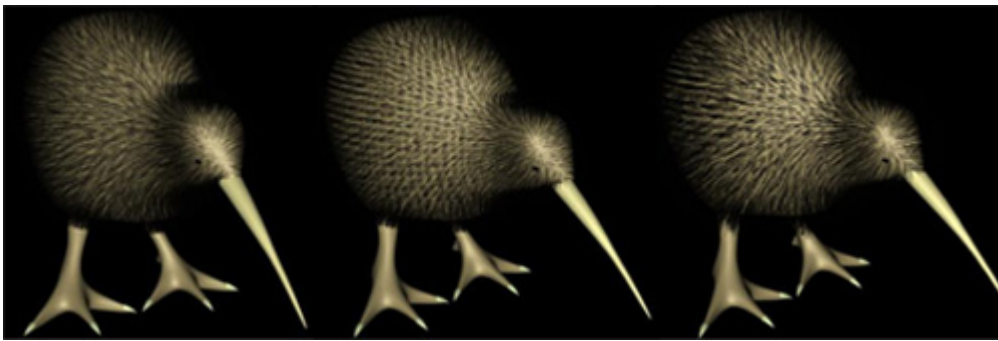


Figure 24: Not very realistic Feathers generated using **Shag: Fur Demo** by *Digimation*

I modeled the kiwi's feathers using a Hair Plug-in created by an individual by the name of Peter Watje who has written a suite of plugins, tutorials, and 3D tools all of which he provides for free along with source code, seen in [12]. I had initially anticipated on using the commercial plug-in **Shag: Fur** by *Digimation*, however, I soon realized the demo version I downloaded didn't allow you to save and would reset all the parameters every 15 minutes. This meant you only had 15 to get everything setup and get a screen dump before you lost everything. Some of the shots I created can be seen in Figure 24, and I initially thought these shots looked quite good, however in hindsight I think they look quite average. That being said, I'm sure with a full working version of the plug-in and enough time to learn to use it, the kiwi's feathers could have been modeled very effectively. However \$500 US a copy was a little steep for my budget, so I began my search elsewhere. After doing some research and reading through some polygon hair modeling methods, which mostly looked terrible, I had the good fortune of discovering the Hair Plug-in created by Peter Watje.

This Hair Plug-in basically simulates hair using a particle system. A particle system is used to model an object or effect that can best be described as a large collection of similar objects behaving in a similar fashion [1]. Obvious examples of such effects include rain and snow, but other equally valid examples include water, smoke, ants, crowds of people, and in our case feathers. A particle is a point in space with associated attributes like color, lifetime, and weight for example. In physical

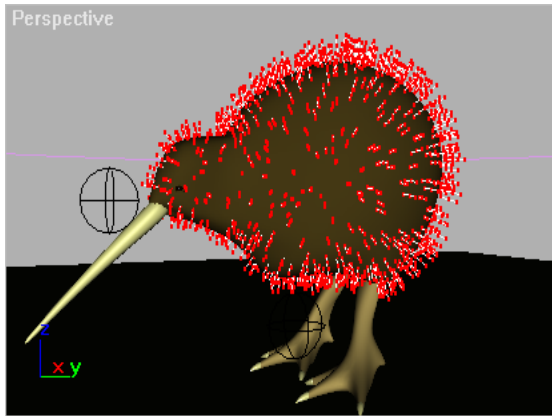


Figure 25: Particle System of Feathers containing Segments made up of 3 Particles



Figure 26: Kiwi with Rendered Feathers using the Particle System (60,000 feathers total)

computations such as the effects of gravity or wind, a particle is treated like a mass point. For visualization purposes, as seen in rendering, every particle is replaced by a boundary represented object. This object's shape, color, and other attributes are directly computed or inherited from the particle. Particle system may consist of hundreds to thousands or even tens of thousands of particles as seen in the number of feathers used on my kiwi.

Although this is technically a hair plug-in, it still may be used to model short fur and feathers, especially those found on a kiwi because its feathers appear more fur like than those of a traditional bird. The first step is to designate an emitter object from which the feathers will grow, in this case the kiwi's body and head. After specifying the emitter objects, a number of input parameters including number of feathers, feather width, feather length, and clumping of feathers are specified, and the results of which can be seen, prior to rendering, in Figure 25. The plug-in allows a number of different rendering options, of which, I used the 3 Sided option where each Segment is rendered as a 3 sided Triangle, using Triangle Strips. This in essence appears as small spikes when the camera is zoomed in, however since I used a relatively small width, from a distance it gives the illusion of a feather. One problem I had is that the feathers by default use the surface normal of the assigned polygon face as the growing direction. This results in a more "fluffy" feather appearance which isn't really desirable since a lot of the effort that went into modeling an accurate body shape is lost under all the feathers. The plug-in does provide for control of the growing directions of feathers through a spline cage where by each feather would follow the direction of the spline it is nearest to. However, unfortunately since this plug-in is free and provided as-is there are quite a lot of bugs, and the spline cage happened to be one of them. After trying tirelessly to get the spline cage working, I found the program kept on crashing, and was forced to abandon the quest for better looking feathers and eventually settled with my fluffy kiwi as seen above in Figure 26.

6 Animation

Animation of this kiwi model was, as mentioned earlier, more of a nice finishing touch rather than any main goal of this project, and so is touched upon in appropriately less detail. Animation of this kiwi falls mainly into 4 stages: Kinematics, Boning, Skin Modifiers, and Key-Framing.

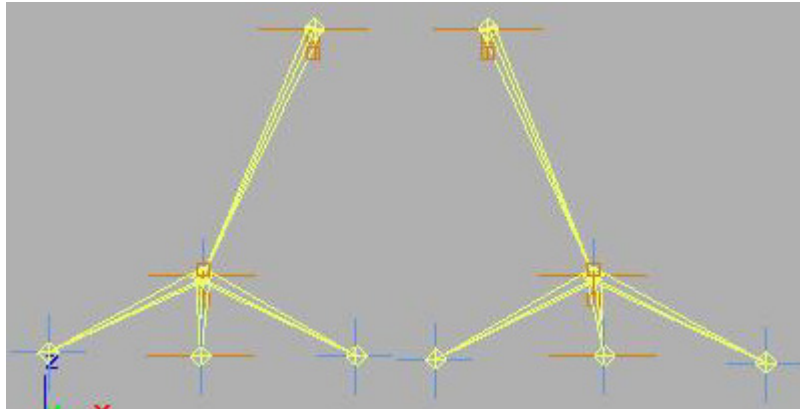


Figure 27: Inverse Kinematic Bone Setup for the Kiwi's Legs and Feet

Kinematics refers to the branch of mechanics that studies the motion of a body or a system of bodies without consideration given to its mass or the forces acting on it [16]. As used in 3D Studio Max, an inverse kinematic system was used to model the motion of the kiwi's legs and feet subject to positional and rotational constraints on joints and bones. This system allows the user to quickly and easily, in theory, to animate any polygon mesh, subject to internal bone constraints, using just a few simply key-frames. The system will then interpolate all possible positions between these key-frames subject to the user specified bone structure and accompanying constraints. For the kiwi, I modeled his legs and feet along the lines similar to that of a human being's knees and feet and their respective constraints. To start with I followed roughly ideas I discovered in [14,15,16] to create a bone structure for the kiwi's feet. I started by creating a knee bone, and ankle bone, and three toe bones, and mirrored the results for the other leg, as seen in Figure 27. I then applied rotational constraints to the knee and ankle bones that they could only bend around the x-axis. I made the knees only able to bend a total of 90 degrees starting and ending in appropriate places which was determined through extensive testing. I also applied the same idea to the ankles, however they could only rotate 60 degrees, and once again the constraint was placed so that the foot could move in a reasonably realistic manner.

It is worth noting that 3D Studio Max offers two types of kinematics: forward and inverse. Forward kinematics refers to the hierarchy of connected objects where the root or parent object must be linked to its children. This means that all motions of the root object are transformed or carried over to the children objects, the parent drives the children's motion. I used forward kinematics for the body, head, and beak of my kiwi, where if the body were to move forward, the head and beak (children) would subsequently follow. I used inverse kinematics for the legs and feet where by the hierarchy is now reversed and children drive the parents motion. So in the case of legs, any motion from the feet moving forward would be transformed to the parent

object which would make sense as the kiwi's feet would be driving the entire kiwi forward.

Once the boning system is in place, we need to provide a mechanism to attach the polygon mesh of the kiwi with this internal bone structure. 3D Studio Max provides a Skin Modifier to do this very procedure. This modifier simply attaches every vertex the polygon mesh contains to a particular bone, and when the bone structure is moved, as seen later in key-framing, the vertex follows this bone appropriately, or at least it is supposed to. The skin modifier provides the user with two main controls over how vertices are to be associated with bones. This is done

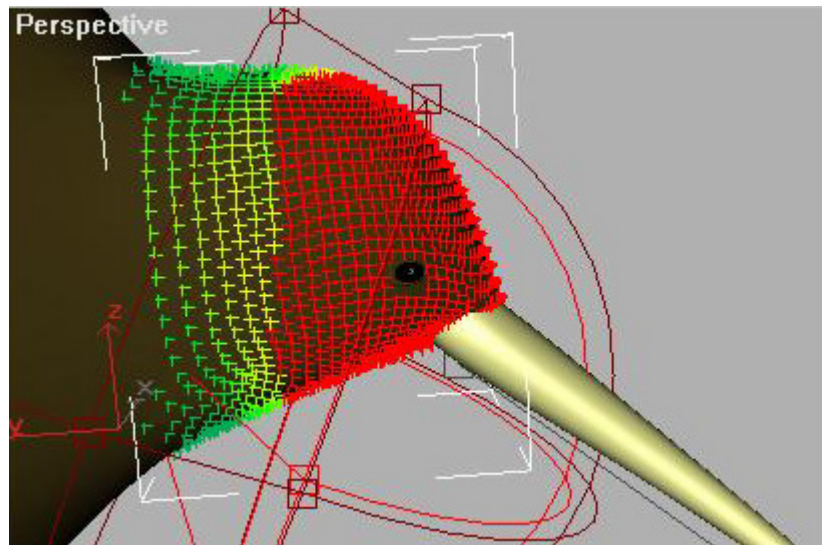


Figure 28: Skin Modifier with Absolute and Relative Falloff Envelopes (red and maroon respectively)

through two envelopes: Absolute Falloff and Relative Falloff. These envelopes can be seen in Figure 28, where by the Absolute envelope in red means that any vertices found within this envelop will follow the nearest bone exactly. They will not be affected by any other nearby bones, and will only move if this nearest bone moves. The second envelope specifies the relative falloff where by vertices that fall within this envelope will be affected by multiple bones and depending on their color, as seen Figure 28, will be affected more or less strongly by a particular bone. Red represents absolute (100%) and means this vertex will follow directly the bone it is nearest too. Blue (0%) means this vertex will not follow any of these nearby bones at all. All other colors fall somewhere in between and appropriately interpolated between the bones they are attached to.

This Skin Modifier and more specifically this way of specifying envelopes was extremely difficult to use and very poorly designed. I had a number of problems in getting the kiwi to walk properly without his feet falling apart because a few vertices were either not attached to any bones or attached to the wrong bones. The system seems unnecessarily complicated and was extremely frustrating to use. An example of the kinds of problems I faced can be see in Figure 29 where the kiwi's toes were melting away because vertices were not in the correct envelope.

The final stage, once the kinematic system, bones, and skin modifier have been setup is to set up a number of key-frames. Key-framing is quite a simple process

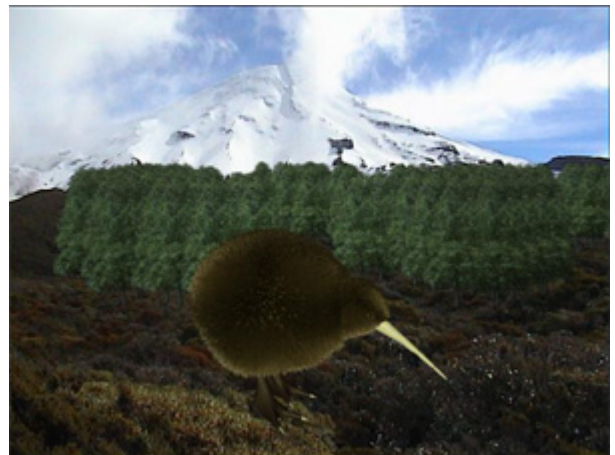
by which the user sets up a number of key-frames and the system solves differential equations subject to the inverse kinematic constraints to determine a solution to the system. In the case of the kiwi, I setup four key-frames at different stages of the kiwi's walking motion. My first key-frame had the kiwi with both feet stretched out as far as possible, the second they were placed together, the third had the same stretched out position with the ordering of the feet swapped. In the first key-frame the kiwi has his left foot in front and his right foot at the back, in the third, his left foot is at the back and the right foot is in front. The final key-frame is once again the same as the second with kiwi's feet placed together. By running this loop you get the false illusion of motion, the only final step is to add a final key-frame at the very end of the frame count that is concerned with the movement of the entire kiwi, not just the feet. So there are really two different motions occurring here. One set of key frames is simply concerned with moving the feet, and the second set is controlling the movement of the whole kiwi. The final result is a kiwi with some basic movement that gives the illusion of him/her walking through the forest (see video).



Figure 29: One of many problems encountered using Max's Skin Modifier

This release of 3D Studio Max 3.1 was more than capable of handling all modeling needs for this project, however the one area lacking was the animation system, more specifically the inverse kinematics system and boning system used in this Release. It proved to be extremely complex, non-intuitive, and quite frustrating to use, and from what I have read I'm not the only one who feels this way. It appears it was so badly received they had to completely revamp the system in Release 4. An interesting note is that in an attempt to salvage disgruntled R3 users, they shipped an additional plug-in called Character Studio along with R3 to ease the burden of character animation. Character Studio apparently makes animation easier and less time-consuming to perform, however this plug-in was outside the scope of this project so I have no personal comments on its effectiveness. I now provide some higher resolution images of some of my final results and conclude with a brief review and possible future work.

7 Results



All pictures and movies were rendered on a Pentium III 1.0Ghz, 256 RAM Toshiba Satellite Notebook with an NVIDIA GeForce 2 Go graphics card. All pictures were rendered using 3D Studio Max's standard rendering at a resolution of 640x480. The sample movie file demonstrating the kiwi walking through a forest was created using Adobe Premiere. The movie is a Microsoft AVI file made up of four individual clips and an audio track. The four clips of the kiwi walking were rendered in 3D Studio Max at a resolution of 800x600 and took an average of three hours for each clip. Each clip is only between three and four seconds long, and so an hour of rendering for one second of video clip is obviously extremely slow. This is precisely due to the current limitations described earlier in generating realistic feathers efficiently. My kiwi model contains 60,000 feathers (50,000 on body & 10,000 on its head) and 1.8 million polygons in total.

3DStudio Max does offer a network rendering option which is a nice feature, and would have sped up the process of rendering significantly. However, it was not a viable one for this project since all computers on the network would need 3D Studio Max as well as licenses to run, not to mention hardware locks and the appropriate plugins.

8 Conclusion

The Virtual Kiwi project has presented an overview of the current 3D modeling and animation techniques, and presented some of the latest trends in generating realistic feathers. This research was used along with the commercial software package 3D Studio Max to successfully create a realistic kiwi and give him some basic movement that allowed it to walk through a forest. Modeling techniques including Polygon, Subdivision, Spline, and Metaball Modeling were discussed along with three of the current trends in creating realistic feathers. Chen et al. presented a unique feather modeling technique, with impressive results, using parametric L-systems and BTF functions. Franco and Walter, along with Streit and W. Heidrich, present very similar feather generation methods based on Bezier curves. Franco and Walter present a more simplified approach with much nicer results, while Streit and Heidrich present a slightly more complex approach but with quite poor results. Finally, Animation techniques including Kinematics, Boning, Skin Modifiers, and Key-Framing techniques were also discussed and how they were used to animate the kiwi.

Some limitations of the kiwi are the modeling of the feathers which do not appear as natural as desired, but were the best that could be achieved under the circumstances. Also the animation is not realistic, although it was never intended to be, but exciting future work in physically based modeling could result in a much more naturally moving kiwi bird. Also future work to use artificial intelligence techniques would potentially allow for the behavioral control of the kiwi, enabling it to react as necessary to its current behavioral state.

Acknowledgments

I would like to acknowledge two people who really helped me out during the course of this project: Shahin Maghsoud and Burkhard Wünsche. Shahin for being incredibly generous and really going out of his way to help me, thanks. Burkhard for an understanding and appreciation of this work, and valuable insight into this project.

References

- [1] Autodesk, "3D Studio MAX Release 3 Reference Volumes I & II," 1999.
- [2] Autodesk, "3D Studio MAX Release 3 Reference Tutorials," 1999.
- [3] Michael B. Comet, "Modeling a Head with Polys in 3DSMAX," *Comet-Cartoons* 2001. <http://www.comet-cartoons.com/toons/3ddocs/headpolymodel/>
- [4] Jaime Fuller, "A Quick and Easy Terrain", *Reel3d*.
<http://www.reel3d.freewire.co.uk/toots/beginner/terrain/index.htm>
- [5] Pål Vågsæther Karlsen, "Animatable Sky Backgrounds," *3D Cafe*, 2001.
<http://www.3dcafe.com/asp/tutorials/max/animatedsky/animatedsky.asp>
- [6] Grégory Chevalier, "Model an Eye," *3D Cafe*, 1999.
<http://www.3dcafe.com/asp/tutorials/max/modeleye/modeleye.asp>
- [7] Michael B. Comet, "Facial Modeling - Spline Modeling Heads," 1998.
<http://www.comet-cartoons.com/toons/3ddocs/facialmodeling/modeling.html>
- [8] Rosalee Wolfe, "Teaching Texture Mapping Visually," *Computer Graphics*, Nov 1997. http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm
- [9] Y. Chen, Y. Xu, B. Guo, H. Shum, "Modeling and Rendering of Realistic Feathers," *Computer Graphics (Proceedings of SIGGRAPH 2002)*, July 2002.
- [10] Cristiano G. Franco, Marcelo Walter, "Modeling the Structure of Feathers," In *Proceedings of SIBGRAPI 2001 - XIV Brazilian Symposium on Computer Graphics and Image Processing*, page 381, October 2001.
- [11] L. Streit, W. Heidrich, "A Biologically-Parametrized Feather Model" *EUROGRAPHICS 2002*, Volume 21, Number 3.
- [12] Peter Watje, "PW Hair Alpha 0.95 Plug-in for Max 3.0", 1999.
<http://max3dstuff.com/>
- [13] Itoo Software, "Forest Pack Lite 1.4 Plug-in", 2002.
<http://www.itoosoft.com/english/frameset.html>
- [14] Cameron James, "Animating with Bones the Easy Way", *My 3D Studio Max*.
<http://www.geocities.com/my3dstudio/tutorials/animating/bones/bones.htm>
- [15] Michael B. Comet, "3D Studio MAX New IK Setup", *Comet-Cartoons*, 1999.
<http://www.comet-cartoons.com/toons/3ddocs/iksetup/index.html>
- [16] Fred Ruff, "Simple Inverse Kinematics," *Ruff-Stuff*. <http://www.ruff-stuff.com>.
- [17] Anonymous, "Faking Global Illumination," *NawYecky*,

<http://www.nawyacky.com/tuts/tut-gi.htm>

- [18] Jay C. Miller, "Bullet Time V2.1", *3D Tutorial Pages*.
<http://www.angelfire.com/fl5/jcmiller/Bullet.htm>

- [19] Graphics Group, "GG Research Areas", University of Auckland, 2002.
<http://www.cs.auckland.ac.nz/GG/research.html>

- [20] Mike Nibeck, "An Intro to Infini-D's Spline Modeler," 1998.
http://www.wwug.com/site/articles/nibeck_mike/nibeck8/modeler.html

- [21] Matthew Ward, " An Overview of Metaballs/Blobby Objects,"
<http://www.cs.wpi.edu/~matt/courses/cs563/talks/metaballs.html>