A System for the Non-Linear Modelling of Deformable Procedural Shapes

T. Lewis and M. W. Jones
Department of Computer Science, University of Wales Swansea
Singleton Park, Swansea SA2 8PP, UK
cstim@swan.ac.uk m.w.jones@swan.ac.uk

ABSTRACT

This paper describes a fully interactive modelling system, which we have called ClayWorks, for defining the implicit geometry of 3D objects using parametric base primitives and deformations. The goal of this system is to provide a means for describing object data and its creation history by using a series of parametric steps that can be modified post-creation in a *non-linear* manner. The advantages of this approach are that unlike other modelling approaches, it enables compact object descriptions, an intuitive modelling process, a simple approach to multi-resolution objects, real-time feedback during the modelling process and a render independent object representation.

Keywords
Procedural modelling, interactive modelling, real-time procedural modelling, object parameterisation.

1 INTRODUCTION

The modelling and rendering of geometric models is a fundamental element within a diversity of fields such as architecture, CAD modelling, simulation and computer games. Research and development of modelling techniques have been broadly divided into real-time systems that allow direct interaction with the object usually using a triangular mesh or NURBs representation, and off-line systems which allow objects to be modelled using a descriptive procedural language. In either case the user will iterate through a process of adjusting object parameters, rendering and viewing the object, and then deciding whether the object satisfies the demanded requirements. In order to be successful, modelling software should enable the iterative process to occur as fast as possible – ideally the modelling should take place in real-time rather than off-line. In some cases real-time interaction is made possible by trading visual quality vs. speed of rendering. This results in the object being rendered at a lower quality, and therefore decisions on whether the object parameters need adjustment may be predicated upon a poor representation of the object.

Journal of WSCG, Vol.12, No.1-3., ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

The work presented herein attempts to bridge the divide between the two approaches by providing the user with a powerful and descriptive *procedural modelling language* that is entirely generated through real-time interaction with the geometric object via an intuitive user interface. The main contributions of this work are that it allows:

- procedural objects to be specified interactively;
- modelling to take place independent of representation (meaning the modelling paradigm has not been chosen for rendering convenience rather than modelling convenience);
- non-destructive and coherent changes to the object specification at any stage during the modelling process in a real-time non-linear manner;

The approach separates rendering and modelling as far as is feasibly possible. Although triangular meshes are rendered, this is entirely to enable real-time performance.

The rest of this document is divided up into sections looking at some related work for modelling objects procedurally (Section 2), introducing the proposed method (Section 3) along with the various operations enabled within the system, and in Section 4 demonstrates a couple of modelling examples. Section 5 concludes the paper and offers suggestions for future work.

2 RELATED WORK

The term *non-linear modelling* is defined here as the ability to change any parameter of a modelling operation at any time. If n operations are used to create an object, the user may edit parameters in step s, where $s \leq n$. There are several well known and widely used modelling tools, such as 3D Studio Max, Maya, AutoCAD, etc. These tools are popular as they allow object manipulation to take place in real-time, and in an intuitive manner. The main drawback of these tools are that they do not offer robust non-linear access to the object specification. Although most store an object history and allow reliable *undos*, changing a parameter for any previous operation will cause the object to behave unpredictably.

In addition to these mainstream modelling tools, object modelling research has considered procedural modelling to be a useful alternative. Generative models, proposed by Snyder [SK92], allow models to be created by specifying a parametric generator, and allowing it to be transformed by various operations to give the final object model using a C like interpreted language. A similar method, by Cutler et. al. [CDM+02] also uses a C like language to specify solid models. In this case distance fields are generated (based on a surface mesh) which also allow object interiors to be specified at varying thicknesses. Vlib [WC01, WC02], has features of both approaches. It allows objects to be generated from parametric curves (similar to Snyder's approach), surfaces (similar to Cutler et al.) and volumes. As a volumetric approach, it allows object interiors to be specified in a controlled manner. Vlib models itself on the OpenGL language, and also uses distance fields (amongst other representations).

L-systems allow a procedural approach to modelling of various geometric objects which exhibit self similarity or limited instances of patterns that can be replicated according to a grammar. Such an approach has been used for plants [PL90], shells [FMP92], and cities [PM01]. These methods are extremely powerful for creating complex objects with compact descriptions, but lack fine control over the object itself.

Cellular texture [FLCB95] operates using a procedural approach but is directed at providing fine surface detail rather than allowing control over surface shape.

The BlobTree as proposed by Wyvill et. al [WGG99] encompasses both CSG and implicit surface modelling in one structure. Objects are modelled using surface primitives which are constructed using boolean and blend operators. Complex models (e.g. shells) can be created with a compact description, but with a great deal of trial and error, user experience and interaction. The HyperFun project [ACF+99] is also developed along similar lines.

Procedural modelling techniques tend to be domain specific (such as using L-systems for cityscapes), or require intricate knowledge of the functional representation of the object in order to convert it into an expressive language. These systems are characterised by being largely non-interactive (modelling takes place in the off-line sense), and by not allowing direct user control over the surface (the user must alter parameters within the descriptive language rather than interact directly with the geometric representation of the object). Quite often modelling will require adjustments to the scripting file, parsing of the file and rendering by an external agent.

3 PROPOSED METHOD

The proposed method can be viewed as a system that provides real-time interactive geometric object editing in the same sense, and with all the same features, as well known surface modelling packages such as Maya, but takes advantage of the fact that the *object* and *scene* are represented internally using a procedural description. The *creation-history* encodes the procedural description of the scene with nodes in the history representing *geometry generators*, *selections*, or *modifiers* and edges representing data flow including geometry, images (for textures, displacements etc.) and selections. Each node has associated with it the parameters for the base object or operation represented by the node, which can be adjusted interactively and non-linearly. Figures 1 and 2 show the key concepts.

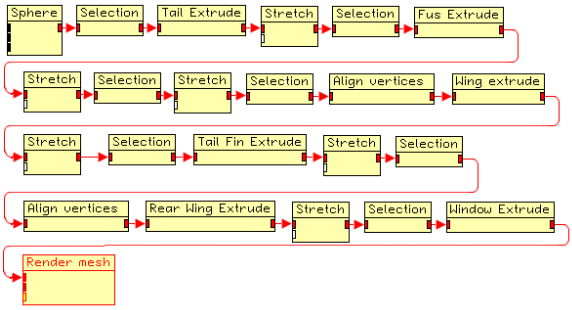

Figure 1: An example *creation-history* generated through the interactive manipulation of a geometric object. Each node stores interaction by the user with the object. Any node can be accessed non-linearly and its parameters adjusted. This creation-history represents the plane model of Figure 8

The next few sections demonstrate how nodes in the creation-history relate to interaction between the user and the geometric object.

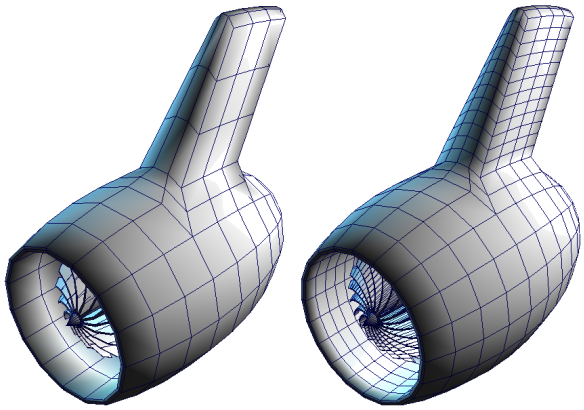

Figure 2: A sphere has been modified to create an engine turbine for a plane. This demonstrates non-linear editing by showing that the user has decided to go to the first node in the history (a sphere geometry generator) and adjust the resolution of the tessellation of the sphere. The left geometry has 1000 triangles, and the right 2000 triangles. The model is stored using less than 1500 bytes in both cases.

3.1 Geometry Generators

There are nine parameterised geometry generators currently implemented. These include the sphere, cylinder, toroid, super parabolic quadratic ellipsoid, cone, cube, polygon, circle and grid (Figure 3). Further platonic solids are implemented, but as the faces are fixed, these are not parameterised. The geometry generator nodes contain the parameters necessary to create the object and once the geometry is generated the node is recorded in the creation-history. The parameters can be altered at any stage of the object modelling process, which results in new vertex, edge and polygon data being sent through the creation-history. All nodes newer than the altered node are recomputed on the updated data flowing through the history. Carrying out this operation in current modelling packages (where allowed) will usually destroy the model as a later modifer is dependent upon a selection that is itself geometry dependent.
The geometry created by generator nodes can be manipulated using selections and modifiers as discussed in the next sections.

3.2 Selection Nodes

The concept of selection described herein, is that the user is selecting a volume (subset of the model) rather than specific topology on the model. It is this procedural aspect of selection that breaks the dependence upon any low level geometric representation such as polygons or patches, and this *representation independent selection* allows robust non-linear editing of previous

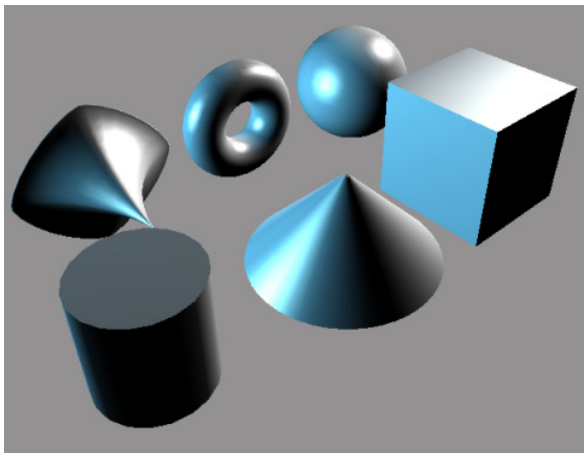

Figure 3: This image shows a selection of base primitives. The cube, cylinder and cone show the use of smooth groups to create sharp edges. Smooth groups can be automatically assigned based on the angle between faces.

parameters in creation-history nodes. The selction volume is stored as mentioned below in a persistent sense, in that the selection is stored in its parametric form within a creation-history node, and is accessible non-linearly.
A selection geometry can be any procedurally defined shape for which it is possible to do an inside/outside test – i.e. any closed shape could be a candidate selection geometry. The most commonly used geometries are extrusions of shapes (rectangle, ellipse, lasso, polyline) drawn in 2D onto the view plane of an editing window resulting in a fustrum, or truncated cone, etc. Additional selections supported by ClayWorks are the basic shapes that appear as geometry generators (Section 3.1), and a convex hull of points. Figure 4(a) shows a rectangular selection in the perspective window generating the shaded fustrum visible in the top, left and front view windows. The parametric form of the selection volume is stored in the selection node, rather than the actual polygons selected (as most modelling packages would store).
When making a selection the user may choose whether this is a *NEW* selection; i.e. any previous selections are ignored, or to *OR*, *AND* or *XOR* the new selection with a previous selection, or to *SUBTRACT* the new selection from a previous selection. Multiple selections can be made on the object using these operations, all of which are stored within the selection node (Figure 4(b)). This ability to add new selections to existing selections using these operations allows users to modify the selected part of an object (e.g. extrude), and then to use the same selection for another modification, or perhaps select the remaining part of the object (using XOR) to carry out a different modifier. Several modifiers export a selection that will be useful to the

user. For instance the extrude modifier will alter the selection so that the selection volume now includes the whole of the extruded part of the object.

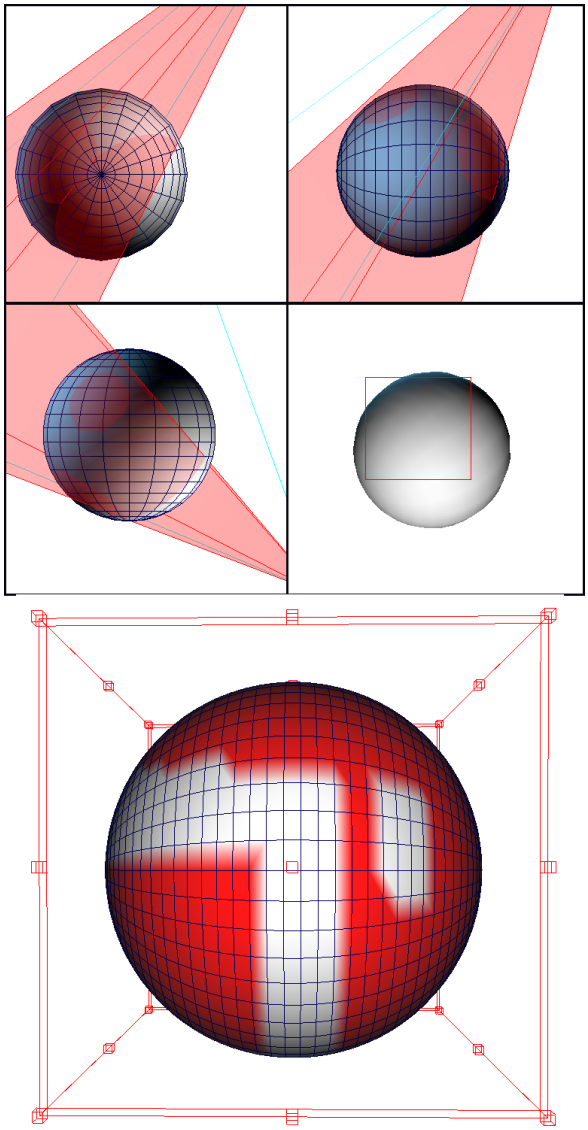

Figure 4: (a) The selection fustrum generated by a rectangular selection in the perspective window. (b) A selection made up from 3 ORs and one SUBTRACT.

Although this representation independent selction has many benefits, in practical applications, it is often useful to select an area on a per element (edge, polygon or vertex) basis. Per-element selections are allowed but in keeping with the representation independent selection ideal, they are not stored as per-element references. Rather, a convex hull (or series of convex hulls depending on the connectivity of selected areas) are stored that represent the selected area at the resolution it was first defined in. If the resolution of the underlying mesh is altered, the selection will still define the area that the user originally intended.

3.3 Modifiers

Modifiers are nodes that receive object data and alter it in some way. All of the modifiers implemented in the system make use of the selection channel which is used to determine which areas of the object will be affected by the modification. Different nodes will affect different combinations of mesh data channels; some will only alter the vertex colours, texture coordinates or smooth groups, whilst others will alter the polygon list of an object in some way, for example rebuilding the whole object based on some procedural function. The term *modifier* is a term for nodes that have in common the fact that they alter one or more of the data channels within a mesh object and export these changes to connected nodes in the creation-history. In keeping with the procedural design philosophy of the system, modifiers are able to reconstruct their output data based on internal parameters no matter what the input data and these internal parameters may be adjusted non-linearly. Modifiers are stored as nodes in the creation-history. To allow real-time interaction the object data is rendered as polygonal meshes, and modifiers execute on this mesh data by one of three categories. The first, and simplest, category involves modifiers that alter the values of per-vertex level elements such as position, texture coordinates, colours, weights or other coefficients. These modifiers do not require any new elements to be added, or any re-indexing of edges or polygons. The second category includes modifiers such as *extrude*, *triangulate* and *subdivide*. These modifiers do not remove any per-vertex level elements but they do can add to the vertex level lists. No re-indexing of edges or polygons which are unaffected by the selection is required. Selected edges that are affected by the selection need to be recreated using newly created vertex level information as well as references to the old vertices. The third category includes modifiers such as *delete* and *merge*. These modifiers are destructive in that they remove vertex level information and require the polygon list to be re-indexed.
The current modifiers implemented in this system are stretch, rotate, extrude, per-polygon extrude, smooth groups, subdivide, triangulate, detatch, delete, align vertices, material application, vertex merge, flip normals, plug holes, mesh expand, texture modifiers (noise functions, UV mapping for textures etc.). Some are depicted in Figure 5.

4 SAMPLE RESULTS AND COMMENTS

In order to demonstrate the flexibility of this interactive procedurally based approach to modelling, an example creation of three models will be studied. The first will examine the use of ClayWorks to create an

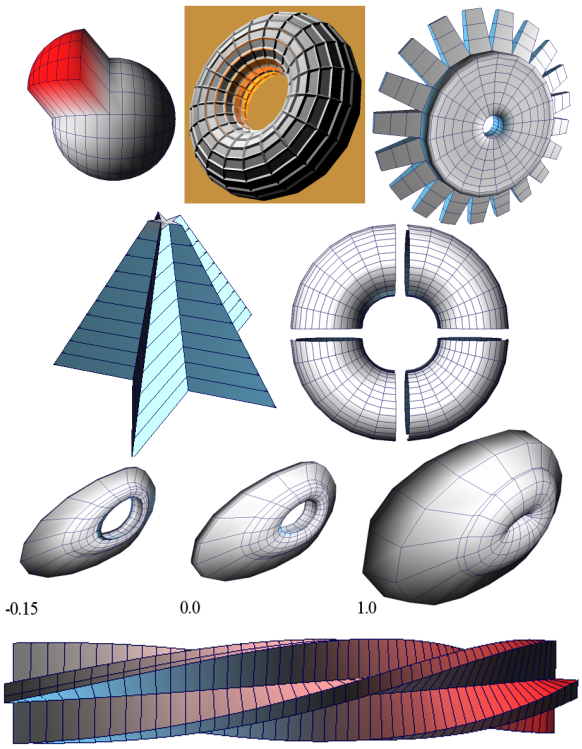

Figure 5: Some modifiers in action. From top: extrude, per-polygon extrude (twice), taper=extrude+stretch, detatch, polygon expand (distance field like modifier), twist=extrude+rotate.

aircraft, and will introduce the modelling process, discuss the creation-history, indicate how this may be useful for creating multi-resolution objects and include metrics on the mesh size. The second will demonstrate how effective use of the parameterisation could lead to useful object animation. The final shows a complex object being modelled, and gives another example of object parameterisiation to give different levels of detail.

The steps to create the aircraft are shown in Figure 6. A sphere is used as the base shape for the plane. A selection of half the sphere is deformed into a rounded conical shape, and the other half is extruded to make the fuselage. A deformation creates the smaller rounded nose. Part of the fuselage is selected ready to extrude the wings (both are made at the same time using a mirror operation), but the vertices are aligned first (to create a flat edge for the wing tips). After the extrude a deformation sweeps the wings back, creates the taper effect and raises them off the horizontal plane (this is controlled by adjusting the vertices of a bounding box during the deform operation and is stored as just one deform node in the creation-history). The same steps are taken for the rear tail wings. Finally a per-polygon extrude is made to create the cockpit windows.

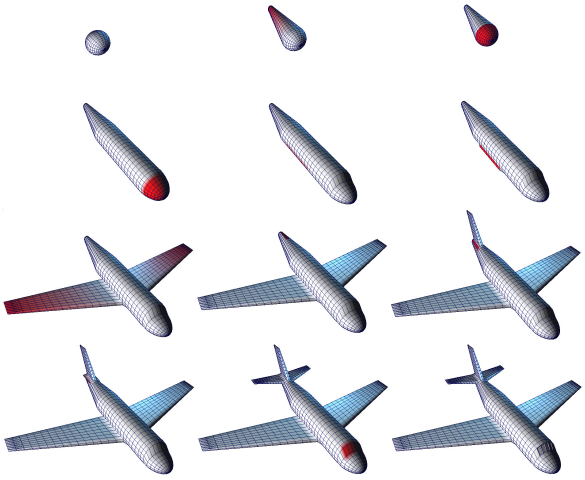

Figure 6: The stages required to model a plane. The creation-history produced from this process is pictured in Figure 1.

Altogether it takes about 2 minutes to create this model using ClayWorks, and at all times the modelling takes place in real-time on a modest PC – i.e. at least 50 frames per second on a 1.4GHz with GeForce 3 (when designed by interacting with a few thousand polygons). In order to achieve this real-time performance special attention was paid to the tesselation algorithm by keeping track of which parts of the object representation need remeshing with information from the creation-history. Z-buffer information is used to determine which parts of the scene need to be re-rendered. This also demonstrates the advantage that models can be built at lower resolution to enable real-time modelling, and then can be generated at the desired resolution. It is a future goal of this system to include adaptive tessellation.

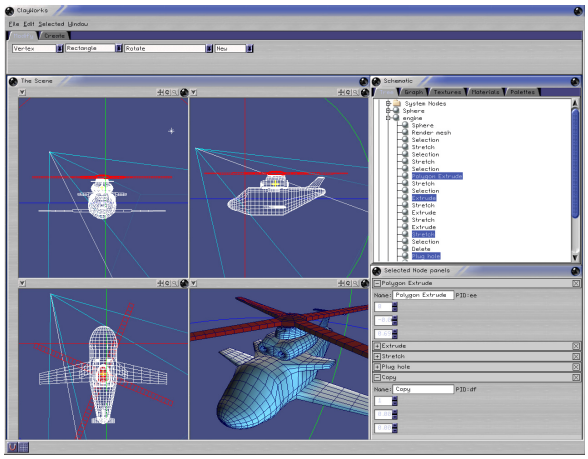

Figure 7: ClayWorks in action. Notice the lower right window allows access to the parameters of individual nodes.

Figure 1 shows the creation-history generated by this

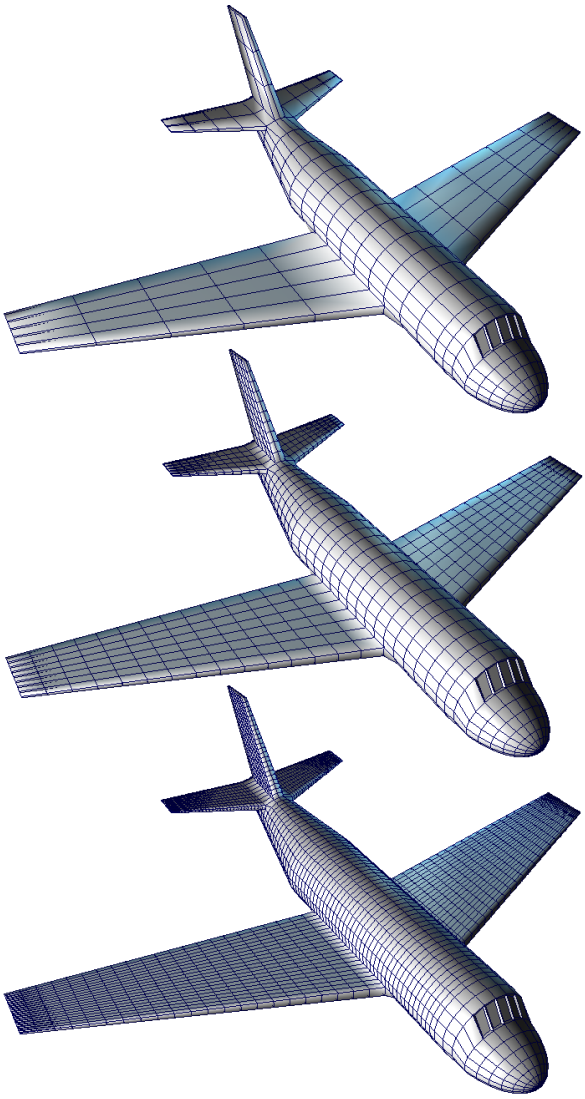

Figure 8: Multiresolution models can be achieved simply by adjusting parameters in the creation-history. From the top down, each plane has 886, 1946 and 5074 vertices; 3576, 7816 and 20328 edges; 906, 1964 and 5092 planes. The model requires 1160 bytes of storage.

modelling process with the extrudes renamed in order to allow the reader to correlate the models and creation-history. The creation-history representation completely models the object in the sense that the rendered model is generated as a visual and interactive representation and is not required for storage. The generator, each selection node and each modifer node are completely parameterised thus allowing the user to go back at any time and alter any of the values. The creation-history offers an extremely compact representation. In the case of the aircraft, only 1160 bytes are required to store the object. At the moment no effort has been made to reduce the storage size, but several improvements can be made to reduce the amount of information stored at each node which will greatly reduce the overall file size.

This storage size is independent of the mesh resolution as demonstrated in Figure 8, where the tesselation parameters in the various extrude nodes have been adjusted to increase the resolution of the mesh (an adaptive tesselation function could work well with this system). The plane can be stored at any of the chosen resolutions at no extra cost. This leads to a simple approach to multiresolution object representation [Hop96] where rather than a post-creation analysis to create a multiresolution mesh, a *controller* node could be introduced and related to the tesselation parameters in each of the extrude nodes. Adjusting that one parameter could vary the resolution, and therefore model quality, and thus give a coherent way to control level of detail. The compact representation would also be of use for transferring objects over the internet where the client could decide upon which resolution level to use based upon the graphics hardware available.

The second example (Figure 9) demonstrates how a control node could feed values into a various modifiers in order to create a simple approach to complex object animation. The object used in this example is chosen to have an organic feel, and perhaps could be something that is to be animated as it flows within a sea current. Firstly this object has been created from the gear wheel shown in the upper right of Figure 5. The per-polygon extrude length parameter has been altered to create longer *arms*. These extrusions have then been rotated about the centre of the object within the plane of the wheel. This is a simple operation as the extrusions are already selected, so it just involves selecting the *rotate* modifier and indicating the point and angle of rotation. Finally the extruded arms are rotated about a second axis. It is this rotation that has been parameterised for Figure 9, although in a final implementation the control node could feed into several different modifier nodes creating quite complex object animation. One advantage of this approach is that, once again, the storage size for this object is less than 1000 bytes, which gives access to all of the dif-

ferent variations seen in Figure 9. Traditional non-parameterised modelling would require many different models, at perhaps 1000 polygons each, to be stored.

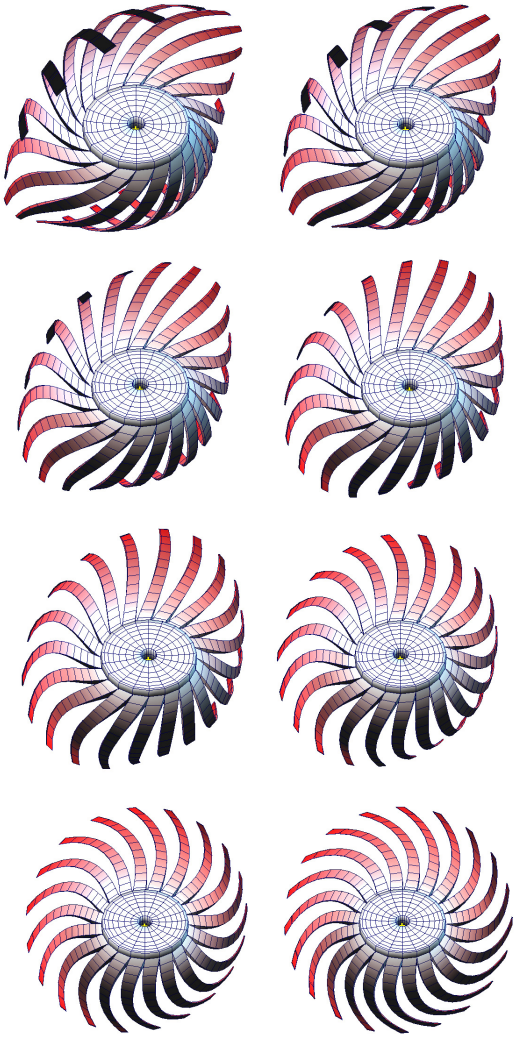

Figure 9: The rotation parameter has been controlled parametrically to give the various models in this sequence.

Most modellers will work with a photograph of a model they are trying to achieve, and so any modelling application should enable the designer to create a plausible replica of the original. The final example demonstrates that ClayWorks, unlike previous procedural modelling languages is domain independent and enables this working practise. Figure 10 shows a photograph of the target model, whilst Figure 7 shows ClayWorks in action. The final model is rendered in Figure 11 with two different resolutions. The blade rotation is parameterised, and so animation of the helicopter in motion could be achieved easily. The final image (Figure 12) incorporates some other features of ClayWorks such as bump mapping, texture mapping and Perlin noise, and the file size for this whole procedural scene is 4300 bytes. An animated fly-past sequence would take no more than 5Kbytes once the animate node has been implemented.

Figure 10: The target model.

5 CONCLUSIONS AND FUTURE WORK

This paper has presented a geometric modelling system that allows objects to be defined using a procedural approach. Unlike procedural modelling languages, the user may interact directly and in real-time with the object, thus providing an intuitive and familiar way for modelling objects. Unlike previous modelling software, the models are stored procedurally and are accessible in a non-linear manner. The system records user interaction with the object in the form of a creation-history in which the nodes are broadly divided into geometry generators for the base geometry, selections for selecting parts of an object, and modifiers which enable the various geometry and property altering operations to be carried out.

In addition to compact models, tests demonstrated the applicability of the method for procedural animation and level of detail models.

Future work will concentrate on including appropriate *control* nodes to enable procedural animation to be described within the creation-history and adaptive tessellation. It is anticipated that this system is of direct relevance to the areas of computer games and web graphics as the advantages of low storage overhead, procedural animation and automatic level of detail models are particularly attractive to these applications.

Other areas open for research are to use the mesh expand function to define objects with interior structure, to study rendering techniques to determine if

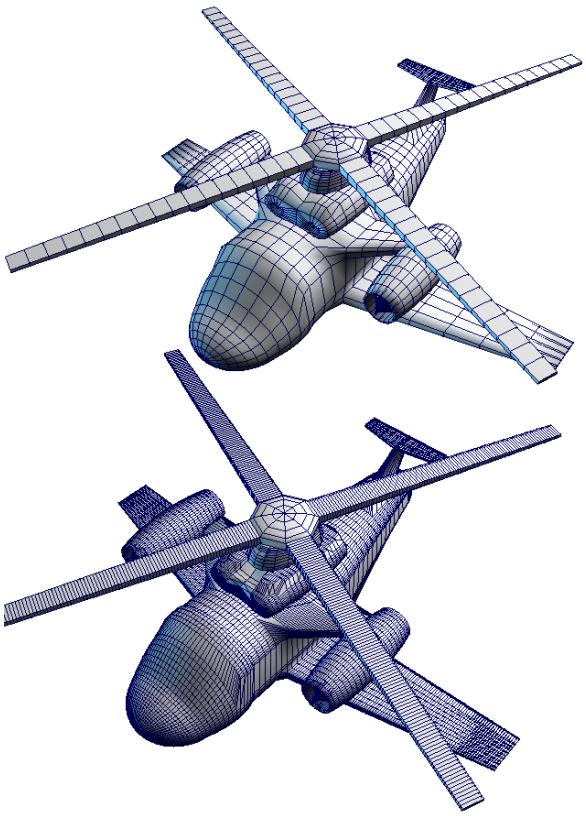

Figure 11: The final model rendered using 3,500 and 34,000 polygons.

costly rendering techniques, such as ray-tracing, can be performed faster using the knowledge that many objects are created from parametric transformations of a known base shape – e.g. a sphere, and to determine if collision detection may be carried out faster using the base objects in the creation-history rather than the object.

In summary, we believe that ClayWorks is a significant step in the merging of off-line procedural modelling techniques with real-time interactive geometric modelling, offering the fundamental benefits of both approaches.

References

[ACF+99] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, and V. Savchenko. Hyperfun project: A framework for collaborative multidimensional f-rep modeling. In J. Hughes and C. Schlick, editors, *Implicit Surfaces '99*, pages 59–69. Eurographics/ACM SIGGRAPH Workshop, September 1999.

[CDM+02] B. Cutler, J. Dorsey, L. McMillan, M. Müller, and R. Jagnow. A procedural approach to authoring solid models. *ACM Transactions on Graphics*, 21(3):302–311, 2002.

[FLCB95] K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr. Cellular texture generation. In *Proceedings of SIGGRAPH*, pages 239–248, 1995.

[FMP92] D. R. Fowler, H. Meinhardt, and P. Prusinkiewicz. Modling seashells. In *Proceedings of SIGGRAPH*, pages 379–387, 1992.

[Hop96] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH*, pages 99–108, 1996.

[PL90] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PM01] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, pages 301–308. ACM, 2001.

[SK92] J. M. Snyder and J. T. Kajiya. Generative modeling: A symbolic system for geometric modeling. In *Proceedings of SIGGRAPH*, pages 369–378, 1992.

[WC01] A. S. Winter and M. Chen. vlib: A volume graphics API. In *Volume Graphics 2001*, pages 133–148. Springer, 2001.

[WC02] A. S. Winter and M. Chen. Image-swept volumes. *Computer Graphics Forum*, 21(3):441–450, 2002.

[WGG99] B. Wyvill, E. Galin, and A. Guy. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, 1999.

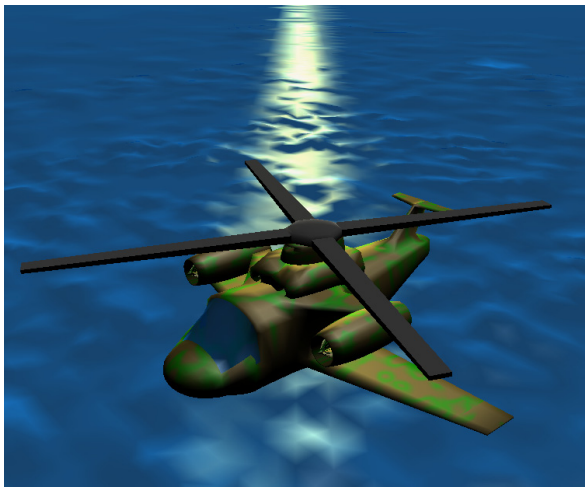

Figure 12: Final model rendered in ClayWorks with texture, transparent window and water (with perlin noise).