EG UK Theory and Practice of Computer Graphics (2011)
Hamish Carr, Ian Grimstead (Editors)

Remodelling of Botanical Trees for Real-Time Simulation

D. T. Reynolds S. D. Laycock A. M. Day

School of Computing Sciences, University of East Anglia
Norwich, NR4 7TJ, UK
{daniel.reynolds|s.laycock|a.day}@uea.ac.uk

Abstract

This paper proposes a technique to use virtual trees created with an industry recognised modelling tool. Initially the skeletal structure is extracted and processed to generate a continuous mesh suitable for high quality, real-time rendering and simulation. Utilising the inherent hierarchical structure of botanical trees, the bone system is calculated from existing, low quality geometry. Once an ordered skeleton is available, a low resolution surface is created around the form as a single continuous mesh providing smooth, continuous connections where branches diverge, avoiding artefacts introduced by overlaid surfaces. Creation of the vertices relative to the skeletal structure ensures no miss-classification in assigning bone influence, allowing for realistic animation and effective mesh refinement introduced dynamically using GPU based techniques.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling— I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

1. Introduction

Realistic simulation of virtual environments is an important part of modern computer graphics with the expectation of vastly increasing quality in real-time applications as hardware capabilities improve. An area of this which has always proved to be challenging is the rendering and simulation of vegetation such as trees, due to the complexity of the models and the complicated structures that govern their behaviour. The field of virtual trees is split into two main areas, the procedural generation of trees adhering to correct structure and formation, and the rendering of tree models in real-time and at high quality. Although they are two separate research topics, the issue with this distinction is that advances in one field will often be unusable to the other. Many elegant solutions developed to generate accurate trees using complicated growth models produce geometry which is ill-suited for animation and high quality rendering, with little enhancement for modern graphics technology. Also many rendering and simulation techniques use time consuming manually modelled geometry or procedural geometry which adheres to an over-simplified, inaccurate rule set.

This paper describes a technique to use data created by a popular piece of tree modelling software, which produces detailed, highly realistic models. The physical geometry is regenerated using the created structure to produce a virtual tree suitable for high quality, real-time simulation and rendering. The Xfrog software detailed by Deussen and Lintermann, [DL97] [DL99], implements an easy to use interface to create highly realistic tree structures, however, the geometry exportable from this application is unsuitable for very high quality visualisations and important structural information such as a usable tree skeleton is not present, making it difficult to create or control tree movement. The techniques described in this paper use the output from Xfrog to derive a full skeletal structure organised appropriately to describe the inherent hierarchy of the tree. New geometry is generated around the skeleton as a single, continuous polygonal mesh introducing higher quality modelling of branch joins and incorporating full bone weighting of the mesh, giving an animatable system similar to applications of character skinning. The model is optimised for cutting edge rendering techniques utilising recent advancements in graphics hardware, most prominently the tessellation engine which is used to both add considerable detail procedurally and remove unnecessary complexity providing efficient, dynamic level of detail representation.

The paper is organised as follows: Section 2 describes previous related work. Section 3 details the contribution made

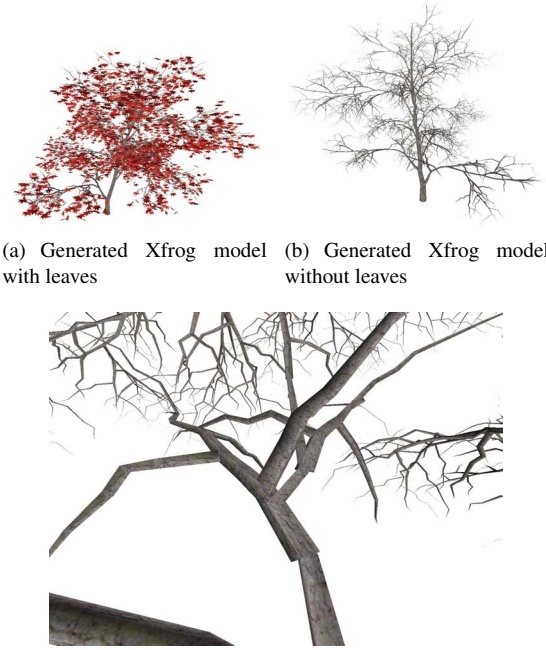
DOI: 10.2312/LocalChapterEvents/TPCG/TPCG11/001-008

by this paper. Section 4 shows results and comparisons of the work and Section 5 concludes the findings.

2. Related Work

While many artists have modelled excellent, incredibly realistic virtual trees, the difficulty and required skill for this task make graphical projects requiring trees a very time consuming and inefficient task to perform manually. In response to this, a large amount of research is carried out in the field of procedurally generating representations of vegetation and arguably, one of the most popular approaches is Lindenmayer Systems, or L-Systems, first put forward by Lindenmayer, [Lin68]. In order to generate effective graphical representations of branching structures in three-dimensions, the original L-System theory had to be expanded giving rise to several new variations. Prominent methods used in simulation include the most simple variety, DOL-Systems (deterministic context-free L-Systems), along with OL-Systems and parametric L-Systems. To generate a three-dimensional structure from these systems, turtle graphics can be used to interpret parts of the rewritten string as geometrical information, much of this process and many examples of it are detailed by Prusinkiewicz, [Pru86], and by Prusinkiewicz and Lindenmayer, [PL90], creating the structure of a plant around which geometry can be formed. The applications of plant modelling using L-Systems developed since the release of [PL90] are surveyed in depth by Prusinkiewicz et al. [PHHM96] with particular attention to the development of extensions of the L-System formalisation as well as new biological uses for the rewriting mechanism. Another survey of more recent developments is carried out in [Pru04].

Plant structures are typically too complex to model manually, however, there are cases where a single plant or tree is a main point of focus in a given simulation and more control over its form is required than can be given by purely procedural modelling techniques. To this end, work has been done producing solutions which take a semi-procedural approach to vegetation modelling in that a structured rule-set it used to generate the hierarchy of elements, within a user-friendly environment for manual creation. One such system, Xfrog, is put forward by Deussen and Lintermann, [DL97], where a hierarchy of user controlled structures is employed to generate a complex tree model. The system functions by giving the user a selection of useful elements such as branching shapes and leaf nodes, and allowing them to be consecutively stacked to form the basis of a botanical model. Modifiers are available to be set as well, governing the shape produced and the distribution of child elements. In addition, the availability of world constraints such as gravity and light allow for simple generation of high quality single models. The system was substantially improved in [DL99] by adding much more control over the individual elements. One major advantage of Xfrog over it's competitors is that it models every individual element including each leaf in a simple manner without using a technique known as billboarding where several elements will be grouped together into one image which is used to texture a plane as a simplified representation. By avoiding billboarding and creating every element, the produced trees can be used with highly detailed simulation techniques such as those put forward by Ota et al., [OTF*04]. While the proposal can be used to create excellent tree structures which appear to be very realistic, as shown in Figure 1(a) and Figure 1(b), there are two major drawbacks to the geometry which is generated. One large limitation is that only the mesh of the tree itself is accessible when exported for inclusion outside of the software. Without access to the structural skeleton of the tree and how this relates to the physical geometry, it becomes impossible to use the model in any sort of dynamic scene. With enhancements in the quality of graphical applications, the expected realism in simulations often requires animation of elements in accordance with weather conditions and physical interaction. The lack of structure to properly simulate this movement is one of the major issues tackled in the work put forward in this paper. The second drawback to the approach of the software, is that models are created using a separate, unconnected mesh to represent each branch. The effect of this approach is that if the root of a child branch does not have a width matching that of its parent at point of connection, unrealistic artefacts are caused as shown in Figure 1(c). At close range, this inaccuracy and over-simplification of the branch junctions



(a) Generated Xfrog model with leaves

(b) Generated Xfrog model without leaves

(c) Artifacts caused by modelling branch joins as a non-continuous mesh

Figure 1: *Young Japanese Maple tree generated using Xfrog*

becomes very noticeable, and drastically reduces the visual quality once independent branch motion is introduced. The software provides an effective and easy to use tool to create the form and structure of a virtual tree. However the geometry produced is inappropriate for animated scenes where the trees may become the visual focus, such as in recent simulations performed by Habel et al. [HKW09].

The techniques of procedurally modelling the geometry of a tree can understandably be split into two areas, modelling the structure and form of a tree and modelling the actual geometry created around that structure. Given the pre-generated structure of a maple tree, Bloomenthal details a method of generating high quality geometry to provide a visually realistic rendering of the tree, [Blo85]. A method is shown which maps a three-dimensional circle of points at intervals around the line of a given tree branch, which are then used as the basis to create the polygonal mesh of the limb. At points where limbs join, complex procedures are required and implemented to join the meshes forming a *ramiform* at the junctions without intersecting or overlapping. Although the geometry created by this proposal is of a very high visual quality and is very realistic, it is inappropriate for use in real-time or animated applications. The costly computation of smooth mesh curves at branch junctions is performed as a pre-process to rendering and is too inefficient to compute in real-time. This disadvantage makes the model completely static as any change in the shape of the tree, especially where a child branch connects to its parent limb, would require re-calculation of the mesh. In addition to this problem, the technique creates tree structures of a very high complexity which may be appropriate if the tree itself is the main focal point of the scene, but makes it infeasible if groups of trees or alternative points of focus are required without drastically reducing the mesh resolution. The complications of procedurally modelling limb junctions is also tackled specifically by Lluch et al., [LVM04], where a pre-calculated structure using an L-System approach is taken as a skeleton upon which to generate the geometry in a single polygonal mesh. The idea of the research is to join different sections of the tree together using only one continuous mesh. The proposal performs this by identifying the intersection points of elements and grafting the polygons together with a higher resolution triangle mesh, ensuring complete continuity. While the approach rectifies the issues of unconnected surfaces whilst maintaining the form of an original input tree, it shares the same limitations as the previous proposal in that the models produced are created using a very high number of polygons making it ineffective in many real-time simulations. The remodelling of branch connections is based upon a static tree. Should motion and animation be introduced to the tree structure, connected limb junctions would need to be recalculated and the geometry regenerated, making the proposal infeasible for dynamic scenes. Section 3 of this paper details a developed technique to create a tree from a simple, low polygon connected mesh which can be animated efficiently whilst keeping it's continuous surface, having detail and higher resolution added dynamically based on visibility.

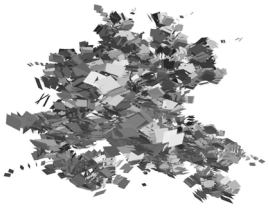
3. Remodelling Trees for Simulation

The remodelling technique detailed in this paper uses trees created by the Xfrog software mentioned previously. This is chosen due to both its high usability and ability to decide how branches of a generated structure are represented. Being a low cost solution compared to its main competitors, such as SpeedTree, also makes it a more accessible tool to work with in both the entertainment and academic fields. The technique will be demonstrated using a stock tree model packaged with the software depicting a Young Japanese Maple with the choice of primitive representing the branches being the only customisation required before exporting the model using Wavefront OBJ format. Using the exported data, the tree skeleton is extracted and used in the creation of a continuous mesh, modelling the branch structure. Automatic bone weighting is employed to transform the mesh and add further detail, as described fully in this section.

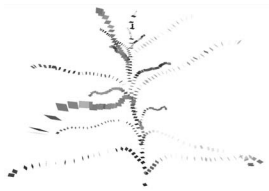
3.1. Generation of the Skeleton

The primitive selection process available in Xfrog simplifies the bone extraction task, by choosing a square representation, each branch of the tree is created using square cross-sections positioned at the point of segmentation along each branch length. This creates an array of small planes forming the line of the limb as shown in Figure 2(a) and Figure 2(b), both in full and only showing the first two levels of branch for clarity. By giving a name to each level of the structure within Xfrog, the exported geometry is grouped within the resulting OBJ by individual branch, each labelled with a sequential number pre-fixed with the name given to that level of branch. This grouping and naming convention is used in the processing of the data to separate the polygons representing each branch and group the collections by the level they belong to. A polyline of bones making up the branch is then generated by connecting the midpoint of each square to the midpoint of the next. Already knowing which level of the structure an individual branch belongs to, the distance between the root of the branch skeleton and the points of the next lowest level can be checked to find the closest point and, as such, the parent branch and point along that branch which the current limb connects to. Using this information, the collection of branches is re-ordered and combined into one hierarchy of elements, storing this main trunk which contains a list of child branches. In turn each possesses a list of child branches until the entire tree is defined. Using this approach, not only is the skeletal information present for bone weighting and animation, but the inherent hierarchy of the tree is fully captured giving an important and useful simulation aid essential for animation techniques such as those put forward by Akagi and Kitajima, [AK06] and techniques proposed by Weber, [Web08]. In terms of basic animation purposes

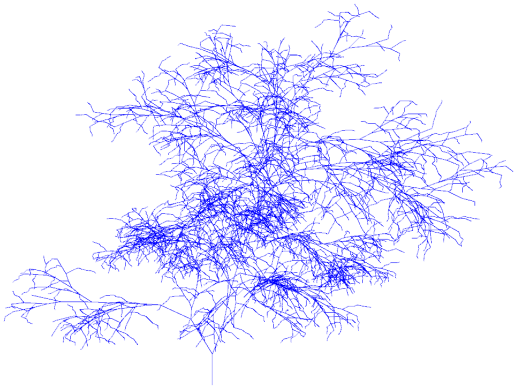
it provides an efficient system to use during the real-time generation of stacked transformations across the structure. For physical simulation the network of branch connections it is readily available for the calculation of force transferral among other interactions present between parent and child.



(a) Exported Xfrog branch representation in full



(b) As 2(a), showing only first two branch levels for clarity



(c) Skeleton extracted from Xfrog representation (blue)

Figure 2: *Skeleton generation from Xfrog tree*

As one of the aims of the project is to implement a system of adding detail dynamically based on the visibility of a section of tree, it is important that the base geometry be a simple as possible. To this end, further simplification can be made to the skeleton itself. As curvature and smoothing of a limb can be added dynamically, described in Section 3.3, the level of segmentation and number of bones in a branch imported directly is greatly unnecessary. By reducing the number of bones present, the number of polygons in the final mesh is greatly lowered and to achieve this, all branches of the lowest level of the hierarchy are left as they are to maintain the uneven crookedness which is desired. All branches of higher levels are redefined by their root points plus the roots of all child limbs. As a result of this procedure, all segmentation and curvature of a branch between the roots of its child branches is removed to be added dynamically, connecting the child roots with single straight bones, as shown in Figure 2(c). The overshooting of parent branches past the root of their last child branch, which occurs in the original model, is also removed as upon inspection of real vegetation patterns, the tendency of natural branches is to terminate at the beginning of a smaller branch shoot or fork into smaller branches rather than continuing on. This process gives a much less complex base structure to be used as the lowest level of detail, with smooth curve detail occurring in real-time as a result of tessellation of the final mesh in combination with bone weighting.

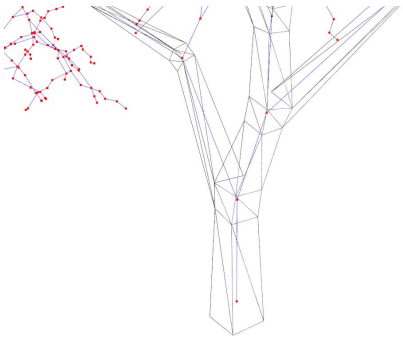
3.2. Creation of the Polygonal Mesh

To solve the problem of branch roots not connecting to their parent branches to form a continuous mesh, rather than editing the existing model to form elegant joins as detailed in [LVM04], the geometry of the tree was completely regenerated around the extracted skeleton. This process allows a tree structure to be created specifically at a low level of detail base which can be easily tessellated to create the desired complexity. Vertices are created using the position of the skeleton points and a function to describe the width of a branch at any given point. This can be the original width of the exported Xfrog tree, or optionally a more realistic curve to add to the general form of the tree. In the case of the example shown in this paper, the width of a branch is calculated to simulate the curve of $\frac{1}{x}$ for $1 \leq x \geq 4$ with the starting width at the root being the width of the parent branch at that point, as shown in Expression 1.

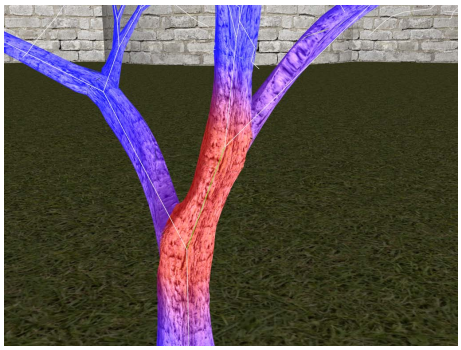
$$\frac{1/(3\times BranchLength\times LengthAlongBranch+1)-0.25}{0.75}\times RootWidth \tag{1}$$

The procedure to wrap the tree in geometry starts at the root of a branch and iterates down the length creating vertices around each bone joint and is then called recursively on each of the current branches child shoots. At the lowest level of detail, the tree is comprised of four sided branches throughout to give a very low polygon count but also to facilitate clean simple joins. To generate a ring of four vertices around each bone joint, first the tangent of the joint between the two connecting bones is found at the joint and used to align the new points. As a more efficient way of calculating the four corners, the closest of the three global axes to the tangent is calculated and the line is projected into two dimensions down both of the remaining axes. By finding the lines perpendicular to the projected tangent, the two remaining local axes around the tangent are found. To generate the four vertices the calculated local axes are made into vectors with a length of half the branch width at that point and added to the bone joint position in all four combinations of the two vectors and their inverse. Once these are calculated they are joined with the points around the previous bone joint by a ring of triangular faces and the algorithm proceeds along the limb to the next segmentation. In the situation where a child limb joins the current section of geometry, two rings of points are created parallel to each other separated by the width of the branch, forming a knuckle on the linear extrusion and giving the child element a simple,

clean join. The side of the branch closest to the direction its sub-branch shoots from is left unconnected by faces forming a hole in the mesh, the four vertices surrounding the gap being passed to the algorithm wrapping the child branch to be used as the initial ring of points, as shown in Figure 3(a). There are several advantages to modelling a tree in the form of a continuous mesh such as this, the most evident being the vastly improved visual quality. Branch joints appear to be more natural and consistent whilst removing problematic and unrealistic artefacts which occur at the junction of two unconnected surfaces. In addition to the aesthetic improvements, representing connections as a consistent surface allows for the proper simulation of force transferral between the elements and the effect applied force and movement has on the junction itself.



(a) Generation of a new model around the calculated skeleton



(b) A single bone's influence on the geometry

Figure 3: *Procedural mesh recreation*

One of the common problems encountered when 'skinning' polygonal meshes (applying a weighting to each vertex to calculate how much influence a given bone has on the point), is the miss-classification due to complex structures of elements being very close to each other and often closer to an incorrect bone than the desired one. Skinning a mesh is a very complex problem, however, as the geometry has been generated entirely relative to the skeleton, it is implicitly known which elements should influence any given section. When wrapping a ring of new vertices around a bone joint, they are each given an influence from the two connected bones of 0.5, with the exception of branch end points which only have influence from one bone at 1.0. As the geometry is modelled as one continuous mesh, where one branch joins another the vertices are shared allowing the influence from both limbs to be accumulated at the points and scaled down to total 1.0 as an additional process after all weighting has been assigned. Once all weighting per vertex has been calculated, an algorithm iterates through the mesh to calculate all bone influences on a per face basis. By examining each face and compiling a list of all skeletal elements that affect any of the three points included, a comprehensive list of all the bone weights for any given face can be generated. In the case where a bone influences some points of a triangle and not others it is assigned a weighting of 0.0 to the uninfluenced vertices, resulting in a constant gradient of influence across the face of the triangle. With traditional 'skinning' techniques it is usually necessary to generate the polygonal mesh in the desired form separately from the skeleton, which is then applied to the model using bone weighting. Depending on the form and complexity of the model it is often impossible to procedurally assign bone influence using a simple algorithm without misclassification of vertices due to distance or shape being closer to the range of a nearby but incorrect bone. As the generated tree is created to directly replicate the previously developed skeleton and during formation of the individual points, the relevant bones are instantly accessible, vertex grouping can be performed during the process ensuring complete accuracy as demonstrated in Figure 3(b). In addition to allowing the mesh to be properly deformed by skeletal animation, the bone weighting assigned provides a blending definition which can be used to refine the geometry itself, as described below.

3.3. GPU Enhancements

The implementation of the tree rendering after processing the data is created using OpenGL 4.1 on an NVIDIA GeForce GTX 460. One of the major advancements of graphics developments in recent times is the introduction of the programmable rendering pipeline, allowing developers to control how vertex and face calculation is performed at the rendering stage. However, the most important and newest breakthrough the implementation makes great use of is the tessellation engine. Tessellation works by taking an input face in the form of a simple triangle and subdividing it into multiple faces before adjusting the position of newly created vertices according to a displacement map or function. As this is performed within the rendering pipeline, no new data needs to be copied across to the GPU per frame allowing for an efficient method of incorporating dynamic level of detail representations without changing the initial low polygon mesh. The engine includes two new shaders into the rendering procedure, the tessellation control shader which is executed on every face of the initial model and determines

the level of subdivision to be performed, and the tessellation evaluation shader which executes on every vertex of the newly subdivided mesh to calculate appropriate point positions. Starting with the simplified, low polygon model created by wrapping the skeleton, extra detail is added by tessellating the surface based on several factors. These factors are considered to determine the correct complexity depending on how much of the mesh can be seen. The main factor is the distance of the tree from the viewer and a linear progression is used to increase the level of subdivision as the viewer approaches the model. The implementation allows for customisation of how effective the mesh refining is by allowing the user to set the maximum desired level of tessellation and the distance from the tree at which tessellation should begin. In addition to distance, this is combined with the area of the face being processed, with a larger face needing higher levels of subdivision due to its greater visibility, as shown in Expression 2. Triangle area and other static per face attributes are calculated as a preprocessing stage to avoid costly computation at render time. The final factor contributing to the calculation is what level of branch the triangle belongs to, allowing further detail to be added only to the areas of the tree which have the highest visual impact such as the trunk or larger limbs. This dynamic range of tessellation which is calculated on a per face basis gives a smooth transition between representations, avoiding the issue of 'popping' occurring when switching the rendered object with one of a different level of detail by recalculating the mesh gradually.

$$(1 - \frac{1}{MaxTessellationDistance} \times FaceDistance) \times (\frac{1}{MaxFaceArea} \times FaceArea) \times MaxTessellationLevel \quad (2)$$

An additional enhancement that can be made using the tessellation control shader is the dynamic culling of faces. By setting a tessellation level of 0.0 the face is removed from the rendering set and travels no further through the pipeline, allowing individual triangles to be turned off when unnecessary. There are two ways in which the implementation utilises this ability to increase rendering efficiency, the first being to remove small branches at such a distance where their rendering is unnecessary for the visual appearance of the tree. Given that the level of each branch face is already known along with its distance from the viewer, the face can be removed if it is too far away. This is demonstrated in Figure 4, which shows four different level of detail representations. The effect of this procedure is that at key distances the highest visible level of branches will be completely removed from rendering, happening per face to give a smooth gradual transition between representations which is not casually noticed with the inclusion of rendering leaves.

One of the most costly procedures during rendering is the copying of data from memory onto the graphics hardware, making it vital that as much as possible is transferred as a pre-processing step and remains unchanged. To enable as much data to be pre-processed as possible, the implementation compiles the information into a continuous array and loads it to the GPU in the form of a texture. This allows the tessellation shaders to look up the relevant information required at render time using their own inbuilt ID as an index. There are two major collections of data being generate prior to rendering, the first being per face information accessed from the tessellation control shader. This includes the area of each face, the level of branch it belongs to and a list of all skeleton bones having influence on the face along with their weighting at each individual vertex. The list of bones included serves as an index into the second collection of data which is information pertaining to the bones themselves such as their length, their position along the branch they belong to, the branch's minimum and maximum width and the endpoints defining the individual bone. This data is used within the tessellation evaluation shader to generate the appropriate position of all new vertices and saves recomputing costly calculations which would severely impede rendering speed.

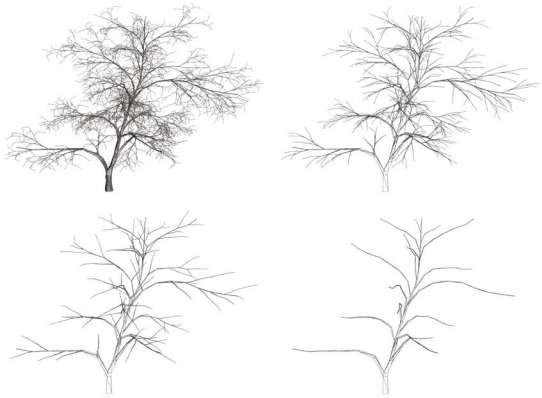


Figure 4: *Dynamic level of detail using tessellation*

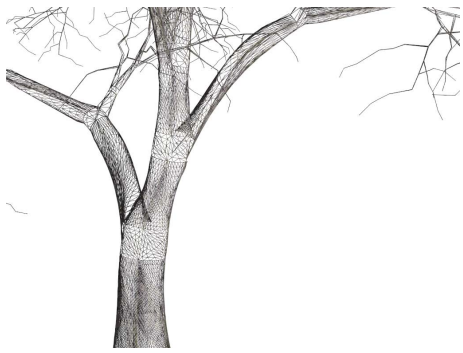


Figure 5: *Added detail and mesh smoothing applied to high poly representation at close range*

Shaping and adding detail to the branches is performed on the mesh after subdivision to create an accurate form for any given level of detail. Face information is accessible using vertex attributes for all three points of the triangle, with newly generated vertices being defined by their barycentric coordinates within the face. For each bone having influence on the face the closest point on the bone to a newly generated vertex is found, the vector between these points giving an offset direction to ensure all new points are positioned uniformly around the skeletal structure. By using bone information such as the position of the bone along the branch and the size of the limb, the same branch width calculation as used for the generation of the initial mesh is used, as shown in Expression 1. This ensures a perfectly consistent rounding of the new limb model when the vertex is offset to this width along the previously calculated vector. This new position is calculated relative to each necessary bone and the final position is averaged across all calculated ones in accordance to the bone's weighting at the particular point, as shown in Expression 3. This merging provides not only a smooth realistic rounding of each limb, but a continuous, gradual, connection at branch junctions and a steady curvature along the length of the branch. Using the weights pre-calculated at each original vertex and barycentric coordinates of the new points, bone weighting is linearly interpolated across the surface of a triangle. However, the fall-off rate for weighting which is present at one side of a triangle can be altered. In the case of branch junctions, a more gradual interpolation is used for the influences of bones belonging to a lower level of branch, causing the large branches and trunk to have a greater effect on the form of the connections than the smaller limbs diverging. Other than shaping the form of the tree, the main advantage of dynamic tessellation is to add finer details when required which this approach allows quite easily. A displacement map in the form of a single channel texture is included to introduce variety in the mesh surface, which when accessed using texture coordinates calculated at the stage of initial geometry production, gives a displacement value which is simply added to the vertex offset distance from the bone. Multiple levels of displacement can be combined into one texture allowing the physical modelling of large detail elements such as knots and splits in the wood along with fine detail such as the unevenness of the bark without introducing further computational cost. With tessellation being variable, at great distances the displacement mapping has no effect on the form of the mesh. However, if the viewer moves very close to the tree, a wide range of complex distortions of the surface is modelled in high detail, providing much greater realism than other techniques used to only simulate the effect such as parallax mapping and parallax occlusion. The culmination of these techniques to provide a highly realistic tree model is shown in Figure 5.

$$\sum_{i=0}^{numberOfBones} PostionRelativetoBone_i \times BoneWeight_i \quad (3)$$

4. Results

Viewer Distance	Frame Rate	Polygons
13.5	120	185000
25	255	25934
45.5	350	6776
66.5	400	1624

Table 1: *Rendering frame rates achieved and generated polygon count at given viewer distances from tree*

Table 1 shows approximate frame rates achieved when rendering the generated tree model at different levels of detail with the number of polygons rendered. When compared to that which was achieved rendering the static model obtained directly from the Xfrog software, approximately 1660 fps consistently, the static model is displayed dramatically faster than the virtual tree produced using the technique put forward in this paper, however, rendering time alone is not a fair comparison between the two. The proposed tree is calculating considerably more at time of render. Displaying the tree using techniques described is inherently including full bone weighting of the mesh and skeletal deformation at every frame ,which is necessary for a dynamic, movable scene element. As the limitations of the static mesh were not ones of efficiency, but rather of missing structural information and poor visual quality, comparison must be qualitative rather than quantitative. Although the results do show that the dynamic level of detail approach used without the introduction of impostors does greatly improve rendering times and shows that the implementation can be effective for non-static scenes where the virtual trees are not necessarily the constant visual focus.

(a) Artefacts apparent in original static mesh

(b) Generated model without connection artefacts

Figure 6: *Comparison of branch junctions*

One of the major drawbacks of the Xfrog model is visual artefacts introduced by disconnected branch geometry and Figure 6 compares the original branch connections formed with remodelled geometry as described, demonstrating that generating a continuous mesh around the structure removes

Figure 7: *Fully tessellated tree showing curvature and shape being a function of skeletal influence*

the unrealistic effects caused by separated surfaces and increases visual quality. As newly tessellated points are procedurally positioned as a function of skeletal influence, Figure 7 shows that the automatically generated bone weighting, calculated at mesh creation, provides an effective solution without manual interference. This is demonstrated by introducing smooth blending between branch segments without artefacts as well as gradual curvature along limbs and across joins. These results can be viewed in a video at http://tinyurl.com/TreeRemodelling.

5. Conclusions

There are software packages available such as Xfrog, which provide an intuitive, highly interactive tool for the generation of virtual plants which require more artistic manipulation than purely procedural methods can allow. However the resulting geometry can be inappropriate for high quality rendering and lack vital structures for the animation and control of movement. This paper presents a technique for extracting key data from exported trees to generate a modifiable skeletal structure and remodel the physical geometry around the structure to allow high quality visualisation and dynamic refinement using recent GPU techniques. New geometry is created to correct inaccuracies caused by unconnected surfaces and to allow consistent modelling of limbs and branch connections when motion and deformation is applied. By modelling the tree in a simplified form around the bone system, dynamic recalculation of the base mesh based on skeletal motion can be performed at a much lower computational cost. High resolution modelling of complex sections such as limb connections is performed in real-time using the tessellation engine and bone weighting is incorporated in the smoothing of the mesh and generation of new geometry in addition to its conventional use of the transformation of existing vertices.

The proposed procedure allows developers to create vegetation using an existing, industry accepted tool and to reformat its output. This allows inclusion in dynamic, animated scenes and simulation for which it was not suitable, without any further manual interaction. The procedure uses novel techniques to base the geometry solely on the extracted skeleton, creating a simple, but highly refinable continuous mesh with automatic bone weighting avoiding many common limitations of procedural surface skinning.

5.1. Future Work

The main proposal of future work is the incorporation of an efficient animation and simulation framework. While traditional animation processes are applicable to the developed technique, given the presence of a fully skinned skeleton, approaches must be explored to fully utilise the hierarchy of the data structure created to provide the most efficient and effective methods of simulation.

References

[AK06] AKAGI Y., KITAJIMA K.: Computer animation of swaying trees based on physical simulation. *Computers & Graphics 30* (2006), 529–539.

[Blo85] BLOOMENTHAL J.: Modeling the mighty maple. *SIGGRAPH Comput. Graph. 19* (1985), 305–311.

[DL97] DEUSSEN O., LINTERMANN B.: A modelling method and user interface for creating plants. *PROC GRAPHICS INTERFACE* (1997), 189–197.

[DL99] DEUSSEN O., LINTERMANN B.: Interactive modeling of plants. *IEEE Comput. Graph. Appl. 19* (1999), 56–65.

[HKW09] HABEL R., KUSTERNIG A., WIMMER M.: Physically guided animation of trees. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 523–532.

[Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development. parts 1 and 2. *Journal of theoretical biology 18* (1968), 300–315.

[LVM04] LLUCH J., VIVÓ R., MONSERRAT C.: Modelling tree structures using a single polygonal mesh. *Graphical Models 66*, 2 (2004), 89–101.

[OTF*04] OTA S., TAMURA M., FUJIMOTO T., MURAOKA K., CHIBA N.: A hybrid method for real-time animation of trees swaying in wind fields. *The Visual Computer 20* (2004), 613–623.

[PHHM96] PRUSINKIEWICZ P., HAMMEL M., HANAN J., MECH R.: Visual models of plant development. *Handbook of formal languages 3* (1996), 535–597.

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty Of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[Pru86] PRUSINKIEWICZ P.: Graphical applications of l-systems. In *Proceedings on Graphics Interface ́86/Vision Interface ́86* (Toronto, Ont., Canada, Canada, 1986), Canadian Information Processing Society, pp. 247–253.

[Pru04] PRUSINKIEWICZ P.: Modeling plant growth and development. *Current opinion in plant biology 7*, 1 (2004), 79–83.

[Web08] WEBER J.: Fast simulation of realistic trees. *Computer Graphics and Applications, IEEE 28* (2008), 67–75.