COVER SHEET

Using Genetic Algorithms to Optimise Triangle Strips

Kieren Lord[+]

Faculty of Information Technology
Queensland University of Technology
Queensland Australia

Ross Brown[*]

Faculty of Information Technology
Queensland University of Technology
Queensland Australia

Abstract

There is an ever increasing demand for higher levels of visual detail in graphical applications, particularly in computer games and applications employing visualisation. Triangle strips have been commonly used to optimise the rendering of large geometric meshes. This paper investigates the process of generating optimal triangle strips through the use of genetic algorithms (GA), to remove the need for special knowledge of the intended hardware platform. Two methods – L-System encoding and parameter tuning of an established algorithm were implemented and tested. The results of this work show that over an extended period of time, solutions can be achieved that are comparable to existing triangle stripping techniques, but the best results were obtained from using the GA to tune the parameters of an existing triangle stripping algorithm.

CR Categories: I.3.8 [Computer Graphics] – Applications; I.3.3 [Computer Graphics] Picture/Image Generation – Viewing algorithms; I.3.5 [Computer Graphics] - Computational Geometry and Object Modeling - Geometric algorithms, languages, and systems

Keywords: Triangle Strip Optimisation, Genetic Algorithms, L-Systems.

1 Introduction

A common approach for optimising 3D rendering is to represent 3D polygon meshes as sets of triangle strips that can be rendered faster than the unprocessed 3D mesh data. These are generated from 3D polygon meshes using triangle stripping algorithms. Most current triangle stripping algorithms create sets of triangle strips that adhere to rules allowing them to be rendered quickly on a target platform (ie the algorithms assume that the target platform has a given configuration) [Holland '92] [El-Sana, et al. '99] [Sucglaboratory '94] [Julstrom '95] [Kornmann '99].

[+]e-mail: hmmklord@optusnet.com.au
[*]e-mail: r.brown@qut.edu.au

This approach is considered acceptable for most triangle stripping techniques as all graphics hardware have geometric factors which influence performance. However, these factors are not always consistent between different hardware, some factors (such as length of triangle strip) are assumed to be a common factor for most hardware. It would be convenient and in some cases preferable for many problems to have a single technique for generating a set of triangle strips that represent a triangulated mesh as a set of triangle strips specifically optimised for rendering on the current platform. The current alternatives are to use many different hardware specific techniques or to use a single technique that is assumed to be hardware-generic.

The aims of this paper are twofold. Firstly, to establish a technique that uses genetic algorithms to create solutions that are optimised for their rendering performance rather than a technique that uses geometric characteristics. As a part of this aim the paper will propose a technique that does not rely on assumptions about hardware configurations. Secondly, the paper analyses the merits of using a genetic algorithm based technique and how such a technique compares to existing methods with respect to performance.

This paper describes genetic algorithms and how they can be used to optimise triangle strips to represent triangulated meshes that are target hardware optimised for rendering speed, without prior knowledge of the hardware. This approach was applied in two ways. Firstly, the GA was used to process an L-System representation of the traversal of the triangle strip. Then it was used to tune the parameters for an already established triangle stripping program developed by NVIDIA, called NVTriStrip [Nvidia '04].

The paper is divided up into the following sections. Section 2 describes relevant background theory in the area of triangle strip representations. Section 3 reviews the literature on algorithms to create optimised triangle strips. Section 4 describes the techniques used in developing this approach to triangle stripping, including the development of an L-System representation scheme and a GA encoding of the triangle strip traversal, and the parameter tuning approach. Section 5 describes the experimental approach and results obtained. Section 6 concludes the paper with a discussion of the results and further work to be performed.

2 Triangle Stripping Theory

A triangle strip is a method for representing a set of triangles as an ordered list of vertices, where each triangle shares one of its edges with the previous triangle in the triangle strip. Each

triangle in the triangle strip is defined by the previous 3 vertices in the list. As a result, the first triangle is specified by the first 3 vertices in the list, and each triangle that follows is specified by a single vertex. Figure 2.1 shows an example triangle strip vertex ordering for a simple triangulated polygon mesh.

Triangle strips have been used in computer graphics for many years. The technique was initially introduced as an iterative form of data compression for large vertex buffers to reduce both memory cost and repeated vertex transform calculations (by a factor approaching 3 in the best case).

In recent years greatly varied and advanced computer graphics hardware has been developed, many of which support triangle strips as a method for describing geometry (in hardware). This is most often used as a means for improving the total rendering speed by optimising various bottlenecks and inefficiencies specific to each platform.

Because the differences in the conditions that create bottlenecks and inefficient computation will vary between hardware platforms, the factors that influence how effective a triangle strip solution is as an optimisation (of overall render time) on a specific platform. Some of these factors are common and beneficial to many hardware platforms (such as using fewer and longer triangle strips).

2.1 Dual Graph Representation

The *dual graph* is a hypergraph which can be used to represent the connectivity of polygon meshes. Figure 2.2 represents a dual graph of the mesh that is shown in Figure 2.1. Dual graphs are commonly used in techniques to find solutions for various mesh traversing problems such as triangle stripping of triangulated meshes [Oliver Matias Van Kaick and Pedrini '04] and silhouette edge detection [Lander '01].

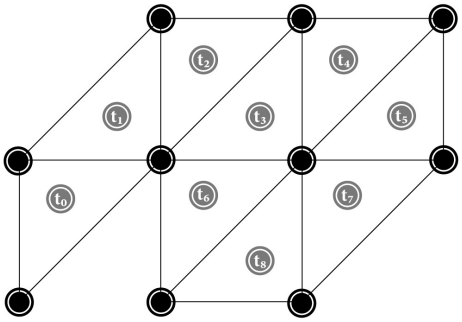

Figure 2.1 An example mesh consisting of 9 connected triangles

A hypergraph is defined as an ordered pair of sets (V, E) of vertices V and edges E respectively, where each edge e1 in E is a subset of V. In the present context a hypergraph representation of a polygon mesh will be referred to as the dual graph of that mesh. The dual graph is defined as the hypergraph where vertices V represent the set of polygons, and the edges E represent shared edges between polygons. This representation is called the dual graph because it can be defined by finding the dual of another hypergraph representation of a polygon mesh.

Using a dual graph representation for a triangulated polygon mesh allows the problem of generating triangle strips to be viewed as a higher level graph traversal problem. It provides a means to directly map any set of traversals through the dual graph such that each node in the graph is visited once and only once to a triangle strip solution of the mesh. A traversal of the dual graph represents an ordered set of polygons where each polygon shares an edge with the previous one in the set. In the case of a dual graph of a triangulated polygon mesh, this ordered set represents a valid triangle strip if no graph vertex is visited twice.

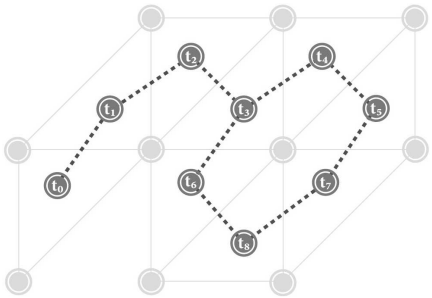

Figure 2.2 The dual graph for the polygon mesh in Figure 2.1

A dual graph representation was used in this approach to implement an instruction based traversal method to produce triangle strips. This representation was chosen because it allows the solution for a mesh to be computed one strip at a time and it does not restrict the problem search space.

Triangle strips are defined as sequences of vertices. The process of generating a triangle strip vertex sequence from a traversal of the dual graph is as follows:

Step 1: Select the triangle represented by the first node in the traversal. Add its 3 vertices to the strip in the predefined starting direction (counter clockwise in this example or clockwise) using the vertex that is not on the edge leading to the next as the first of the three vertices.

Step 2: Examine the next two nodes in the traversal, and determine if it is possible to reach the second of these assuming that the next triangle will have the opposite direction to the current one. If it is inaccessible, add a duplicate of the second most recent vertex that was added to the strip. This creates a zero area triangle along the edge between the current triangle and its successor which is called the *swap triangle* that maintains cache hit rates by making sure that the previous two vertices are in the cache from the last triangle [Estkowski, et al. '02, Hoppe '99, Oliver Matias Van Kaick and Pedrini '04].

Step 3: Move to the triangle represented by the next node in the traversal. Add the vertex that is not shared with the previous triangle to the strip.

Step 4: Repeat from step 2 until the last node in the traversal has been reached.

The purpose of the *swap triangle* is to reverse the direction of the next triangle in the strip. By reversing the direction of a triangle in a triangle strip, the edge which the following triangle will start from is changed. Without adding a swap

triangle as mentioned above, the strip will not follow the sequence of triangles defined by the traversal. As mentioned earlier, the triangle strip is made longer by one vertex each time a swap triangle is inserted. There can be at most 2 + 2n vertices in a strip that represents n triangles.

3 Triangle Stripping Algorithms

There have been many contributions to the algorithms for creating and refining triangle strips [El-Sana, et al. '99] [Hoppe '99] [Estkowski, et al. '02] [Chow '97] [Evans, et al. '96]. These developments have been made possible by increased processing power and changes in computer graphics standards and platforms that can take advantage of previously unused information for more efficient rendering performance.

An appealing factor of triangle strips is that they compress the vertex data being sent to the renderer (at a best case $n/3 + 2$ for n vertices) in such a way that can be rendered in a more computationally efficient way than a random rendering sequence of individual triangles.

To process triangle strips a system must have a vertex cache that holds at least the last two vertices processed. More recently, hardware with vertex caching of large numbers of vertices has become common [Bogomjakov and Gotsman '01]. Previous work on creating triangle strips that have large vertex caches had been done prior to this [Chow '97], but had focused on using this as a compression technique for storing geometry (and required a great deal of control over the cache). This has since become adapted [Hoppe '99] to make efficient use of hardware caching to achieve better rendering performance.

Many triangle stripping algorithms [Kornmann '99, Marshall '02] [Hoppe '99] for meshes of triangular polygons represent the input mesh as a dual graph, which is well explained in [Oliver Matias Van Kaick and Pedrini '04]. A dual graph of a mesh is a (potentially non planar) graph representing the connectivity of the mesh. In the dual graph, each triangle of the mesh is represented by a unique node, and for any two triangles that share two points (therefore joined along one of their sides) an edge will connect the dual graph nodes representing those triangles.

The program STRIPE [Evans, et al. '96] uses a hybrid of local and global algorithms that creates a set of sub-graphs which have known paths, and a sub-graph of all the remaining nodes which are solved by a local algorithm based on the SGI *Tomesh* algorithm.

Another category of triangle stripping algorithms derive solutions from a higher level representation of the mesh connectivity than the dual graph. The approach of using a spanning tree of the dual graph was proposed in [Estkowski, et al. '02] which became the basis for the FTSG program.

Despite the NP-complete status of the problem of finding a solution with the least number of triangle strips possible, Estkowski et al. [Estkowski, et al. '02] proposed two algorithms for finding such solutions (which were also incorporated into the FTSG program). Prior work on finding these solutions [Arkin, et al. '96] had focused on triangulation of single planar polygons into sets of triangles with the minimum number of triangle strips (for each polygon's triangulation).

There are questions arising from the literature about how to address the research question. Mainly, should triangle strips be used? There is a case against using triangle strips when transparent vertex caching is supported in hardware which may soon make the use of triangle strips redundant. An obvious alternative is to attempt to apply genetic algorithms to the universal rendering sequence techniques proposed by Bogomjakov et al [Bogomjakov and Gotsman '01]. It should be noted however, that if the platform being used has no support for transparent vertex caching then universal rendering sequences will be *significantly* less efficient than using triangle strips. Literature has also shown [Hoppe '99] that triangle strips can make effective but less efficient use of transparent vertex caching.

Triangle strips will remain a valid and commonly used representation of geometry in the foreseeable future as universal rendering sequences perform poorly in comparison on any hardware that does not support transparent vertex caching. There still exists a demand for graphic applications that must accommodate such hardware in software that is written for, or to accommodate legacy graphics hardware and new mobile devices/gaming platforms that do not support this hardware feature.

Secondly, to support the use of GA optimised triangle strips, there are similarities between the travelling salesman problem and the triangle stripping problem that infer that some of the encoding techniques and operators may be adaptable to this research topic.

4 A Genetic Algorithm Approach to Triangle Strip Optimisation

4.1 Genetic Algorithms

A genetic algorithm (GA) is a form of optimisation algorithm. GAs have similarities to gradient descent optimisation algorithms in that they sample proximal solutions within the solution space to estimate a better solution than the current solution. Therefore in this context, terms such as global minima and local minima have the same meaning with respect to GAs. GAs can only be applied where solutions to the problem can be expressed as a genome and a function exits that can be used to rank the effectiveness of a solution (or phenotype [Holland '92]). The usefulness of a GA in solving a problem is a product of the suitability of the encoding of the problem, and the accuracy of the fitness function.

GAs have been used for graph traversal problems that are similar to triangle stripping - such as the travelling salesman problem [Julstrom '95]. GAs can be useful for problems where little is known about the options within the solution space, and/or if the function used to determine the relative quality of a solution is noisy [Holland '92].

Genetic algorithms can thus be applied to optimisation of triangle stripping due to the ability to represent a particular solution as genomes that represent the commands to traverse a

triangle strip, as listed in Table 4.1. Typically, the genome encodes the information in a bit stream representation, and is manipulated at this level by the GA, we use a command level representation here for clarity. For example, a simple graph traversal may be represented by a four gene genome as in the following Figure 4.1.

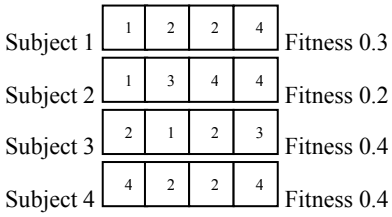

Figure 4.1 Example genome encoding of four subjects of triangle strip traversal instructions, with attached possible fitness scores.

A random population of triangle strip traversal genomes is generated, and tested via the fitness function. The fitness function for our application is rendering efficiency, measured as the time to render the mesh. This efficiency score gives the subject a chance to proceed to the next generation in the breeding process, if it is deemed fit enough. Once the best performing progeny are selected, then the genes of this fit population may be *crossed over* – parts of the gene representation of the parents may be swapped to produce new genomes in the offspring or *mutated* - discretely changed by some process (refer to Figure 4.1). The fitness assessment and mutation/cross over process is repeated for enough generations to produce an acceptably optimised solution.

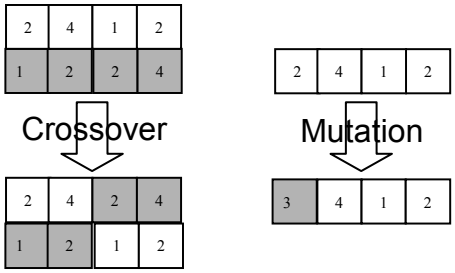

Figure 4.2 Examples of mutation and cross over operations on four element genes.

From the perspective of our example, this enables the breeding of populations of incrementally improving subjects that are more and more efficient in being rendered as triangle strips. It also meets the goal of removing the need for understanding of hardware specifications with regards to the development of optimised geometric content. Specifics of the application of the GA technique to an L-System representation are now detailed.

4.2 L-System / GA Triangle Strip Optimisation Technique

In an L-System grammar, every symbol in the alphabet is associated with a re-writing rule (or production). When a string is re-written by the grammar every instance of a symbol in that string is replaced with the contents of its associated re-writing rule.

Recursive re-writing is performed by re-applying the L-System grammar to the string that is generated by the re-writing that has occurred to the previous string. Figure 4.3 demonstrates the process of recursive re-writing.

When the re-writing is performed recursively, the string that is created contains similar patterns to both the rewriting rules and the originating (or parent) string. This concept is relevant to encoding solutions to graph traversal problems as a genome (see the next paragraph). By using rewriting rules as a genome it is possible to create a solution string of any length that shares patterns with the re-writing rules/ parent genome. The patterns within the rewriting rules are present within the solutions strings and by using this string to define a graph traversal, a traversal is created that has a pattern that is defined in the genome.

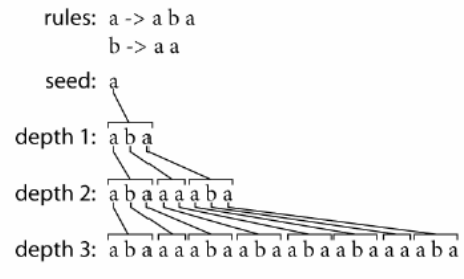

Figure 4.3 Example of an L-System grammar (D0L-System) and process of recursive re-writing.

L-Systems have been applied as a GA encoding for other similar problems [Julstrom '95]. For example, they have been used to create realistic plants and furniture in computer graphics [Hornby and Pollack]. L-Systems are a preferred option to other genome alternatives because the small, fixed size encoding can be used to create solutions of any size.

The technique employed in this context used GAlib [Galib '05] to create a 2D binary genome of fixed length L-System rewriting rules. By doing this, each member of the GA population represents an L-System grammar. The L-System rewriting rules process the genome to generate a string of commands that are interpreted as traversals of the dual graph.

A command string is generated from a fixed seed string. The command string is recursively rewritten until it is long enough to ensure that it contains enough commands to traverse the entire mesh. Each command in the string refers to an ordered list of triangle operations. In this technique each of the commands refers to one of the following lists of triangle operations (see Table 4.1).

Command	*Triangle Operations*
1	next, swap, end
2	next end
3	swap, next, end
4	end

Table 4.1 Commands/Triangle Operations

Triangle operations are commands that are used to choose a traversal of the dual graph of a mesh one step at a time. The triangle strips are generated by attempting each triangle operation in the list (Table 4.1) for the current command in

order until one succeeds. The end operation cannot fail so it is not possible to run out of operations with respect to a command. The command string is executed in this way until every triangle in the mesh has been added to a triangle strip. Thus the L-System does not generate a complete triangle strip traversal, but generates a potential set of commands, and tests these commands on the mesh. So, if a *next* command fails, it is retried with a *swap*, and then an *end*. The commands are:

- *Next* adds the triangle that is on the leading edge of the current triangle. Fails if no triangle exists on the leading edge or if the triangle has been already added to a triangle strip.
- *Swap* adds a new triangle on the leading edge of the current triangle by repeating the second last vertex of the current triangle and adds the next triangle that was on the leading edge. Fails if no triangle exists on the leading edge or if the triangle has been already added to a triangle strip.
- *End* terminates the current triangle strip. Always succeeds.

The fitness function for this technique is required to grade the performance of each genotype (or L-System grammar). This function is defined as the time required to render the triangle strip that is produced by the command string. This technique uses an average of the rendering time for a fixed number of iterations - in order to achieve an accurate value for the fitness function, as each iteration is only a fraction of a second. The number of iterations is determined by the user having a mind to the processing power of the hardware. The number of iterations is applied consistently to each trial and thus it does not impact on the results (assuming the number of iterations is high to ensure the accuracy of the fitness function for the platform on which the technique is being used). The rendering processes of the fitness function in this context are referred to as a *stress test* – rendering an object multiple times per test to improve time difference assessment.

The GA uses standard binary crossover/mutation operators to optimise the L-System grammars in order to minimise the time required for the stress test. However, crossover operators are restricted to triangle operator boundaries in the encoded bit string, to avoid gene splitting. Thus the triangle traversal commands listed in Table 4.1, are not destroyed by the GA operators, and are preserved for the next generation.

It is expected that this approach will have strengths in its ability to handle the subtleties of the underlying hardware, as the L-System has closer control over the structure of the triangle strip that is generated.

4.3 NVTriStrip Parameter Optimisation Technique

NVTriStrip is a free triangle stripping library made available by NVIDIA that is commonly used in the computer games industry [Nvidia '04]. It uses a number of techniques to optimise triangle strips for hardware. Firstly, it uses stitching to join a number of triangle strips into one big strip via the use of degenerate triangles to jump to new locations in the mesh. Stitching thus removes the overhead of stopping and starting a large number of triangle strips when rendering a mesh. Secondly, its techniques are aware of the size of a vertex cache on GPU cards, and so can adjust the arrangement of triangle strips to suit the hardware specified via index remapping, to improve spatial locality in the vertex buffers on the graphics hardware.

Genetic algorithms can be used to tune parameters to established algorithms, and so they can be used to leverage an already established approach. NVTriStrip has the following parameters:

- *EnableRestart* - Enables primitive restart and enforces stitching;
- *Cache Size* - Sets the size of the vertex buffer cache to use;
- *Stitch Strips* - Enable disable strip stitching;
- *Min Strip Size* - Sets the lower bound on strip sizes. Any strips smaller than this are added to one large strip.

These have been encoded into a direct genomic representation of floating point values in a binary stream, using standard mutation and cross over operators as used for the L-System approach in Section 4.2. Again, the fitness function is the stress test method of assessing rendering efficiency.

This parameter tuning approach meets the research goals of hardware independent optimisation by seeking to tune the parameters of an algorithm that are hardware dependent via a GA approach. In this case the representation is not so closely coupled with the triangle strip, which may bring the advantages of using an already established triangle stripping technique. However, it may struggle with the subtleties of the hardware configuration because it has no direct control over the structure of the triangle strip.

5 Experimental Results

Two experiments were conducted to evaluate the performance and potential benefits of the proposed GA based technique. The purpose of the first was to compare the L-System GA based technique with a benchmark like NVTriStrip, with respect to their relative performance and to evaluate the respective merits of each approach.

The second experiment involved testing the tuning of parameters for an existing tri-stripping technique NVTriStrip. This indicated whether a more effective triangle stripping method would be to use the GA to optimise an existing algorithm's set of parameters.

Detailed descriptions of both experiments follow. These descriptions provide a comprehensive outline of the procedures undertaken.

5.1 L-System GA Experiment

The purpose of the experiment was to determine if the proposed GA based technique was effective at generating triangle strips specialised for a target platform. The results of this experiment were compared with solutions generated by the NVTriStrip library [Nvidia '04]. The aim of the experiment was to determine if the GA algorithms provide more consistent performance across a range of platforms than

would be achieved by the NVTriStrip library and similar algorithms and techniques.

The conditions under which the experiment was conducted was as follows. The machines used were:

- *Machine A* - Desktop PC with an AMD Athlon64 3000+, 1Gb of ram, NVidia GForce4 4400Ti, running Windows XP (with WinXP service pack 2);
- *Machine B* - Toshiba Portege 3840CT with an Intel Pentium III 600, 64Mb ram, S3 Savage/IX, running Windows 2000 (with Win2k service pack 3).

The experiment used a partially decimated version of the Stanford Bunny mesh [Sucglaboratory '94]. This mesh was chosen because the results from using this mesh can be compared to techniques used in other research. An example generated triangle strip is shown in Figure 5.1.

In an effort to eliminate the impacts of other programs and services on the experiment, some non essential programs and services on both machines were closed - eg. virus scan and multi media services. An effort was made to ensure that both machines were running the same programs at the time of the experiment.

A set of triangle strip solutions for the Stanford Bunny mesh were created using the NVTriStrip library on machine A and machine B. Six solutions were generated for each machine and each solution had a different combination of target cache size and stitching mode. These solutions were graded and the results were stored on each machine in a log that recorded output solution and performance.

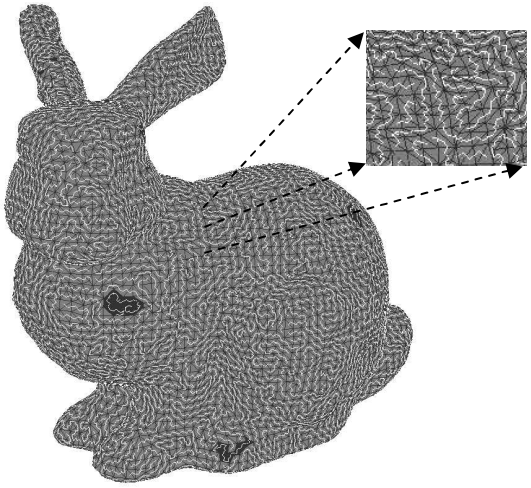

Figure 5.1 Illustration of triangle strip generated from the Stanford Bunny mesh. The triangle strip traversal is represented as a white line passing through the barycentre of each triangle, as shown in the blown up section top right.

In order to perform a comparative analysis of the results of the two techniques, the mean and standard deviations of a set of runs of each respective technique for each machine were needed. This was required because both techniques required different input parameters and the machines used were not equal in processing power, making it impossible to compare individual results of the techniques or machines directly.

The next stage of the experiment involved running the proposed technique on both machines with population sizes varying from 5 to 20, within rule lengths varying from 100 to 400.

The range of rule length sizes chosen for this experiment took in to account the number of triangles that make up the test mesh. This was done because the sizes of the triangle strip patterns produced by this technique are relative to the length of the L-System rules stored in each genotype.

For each chosen rule length, a set of runs with a set of fixed GA population sizes were tested in this experiment. The population sizes for these were chosen from the range that the technique could run in an acceptable timeframe on both the machines used in this experiment. The best solution from each run and the performance of all solutions examined were stored in a log for each run.

Results & Analysis

Table 5.1 and Table 5.2 summarise the results from the experiment. Table 5.1 reports the results of the experiment that applied the technique proposed in this paper. The table contains the results of nine runs and the parameters used for each run. Columns headed *strips A* and *msec A* report the solutions generated by machine A given the variables used for each run. Similarly, columns headed *strips B* and *msec B* report the solutions generated by machine B using runs that have the same parameters as those used on machine A. The parameters of each run were the rule length and the population. The table also reports on the standard deviation and mean for each machine with respect to the strips and the time taken to render the final solution (ie column msec).

Run	*Rule Lgth*	*Pop.*	*Strips A*	*Msec A*	*Strips B*	*Msec B*
1	100	5	7799	2.96	7514	69.5
2	200	5	7521	2.84	7890	69
3	400	5	7438	2.84	7975	69.5
4	100	10	7601	2.68	7330	70
5	200	10	6956	2.76	7431	69.5
6	400	10	7438	2.84	8116	69
7	100	20	6885	2.76	7578	68.5
8	200	20	7418	2.56	8099	69.5
9	400	20	7326	2.72	8474	69
std dev			291.550	0.115	381.713	0.441
mean			7375.778	2.773	7823	69.278

Table 5.1 Parameters (L-System rule length / GA population size) and Results of GA technique trials (Number of triangle strips generated in the solution on each of the two machines / number of milliseconds required to render the solution generated on the machine that created it)

Table 5.2 reports the solutions generated by the NVTriStrip library on machines A and B. There were six runs that used different combinations of the two parameters *cache size* and *stitching mode*. The columns headed Machine A and Machine B report on the time taken by each machine to render the solution for each run. These solutions were generated using

NVTriStrip with the combination of cache size and stitching mode variables, defined in each run. The table also reports on the standard deviation and mean for the times required by machines A and B to render the solution for each run. Note also the table shows that for each run where the stitching mode was enabled, the solution comprised a single triangle strip.

Analysis

Comparing the results from Table 5.1 and Table 5.2, it can be seen that NVTriStrip produced triangle strip solutions that had better performance than the solutions produced by the GA based technique. There are several possible reasons for this. The GA technique produced solutions that had a larger number of strips on Machine A and Machine B.

Run	*Cache*	*Stitching*	*Strips*	*Machine A*	*Machine B*
1	16	enabled	1	1.32	47
2	16	disabled	3421	1.72	47.5
3	24	enabled	1	1.36	45
4	24	disabled	2777	1.56	45.5
5	32	enabled	1	1.12	43.5
6	32	disabled	2157	1.48	59
Stddev				0.208	5.616
Mean				1.427	47.917

Table 5.2 Parameters (cache size and stitching mode enabled/disabled) and Results of NVTriStrip library trials (Number of triangle strips generated in the solution for the given parameters / performance of the generated solution on each of the two machines).

This larger number of triangle strips created a performance overhead when rendering these solutions. In addition, the solutions created by the NVTriStrip library are also optimised to accommodate hardware that supports vertex caching - this is a common feature in 3D hardware [Hoppe '99] [Evans, et al. '96] [Bogomjakov and Gotsman '01]. It is not known to what extent the GA is able to optimise solutions for vertex caching; however, it is unlikely that it will achieve similar results to the NVTriStrips.

NVTriStrip library has a feature called stitching that is not present in the GA technique. Stitching is a process that uses degenerate triangles to link unconnected triangles together. The reason for this feature is that it reduces the performance overhead that would be incurred if the process had to start multiple triangle strips to reach a solution. This feature is evidenced in runs 1, 3 and 5 of Table 5.2.

NVTriStrip out-performed the GA technique by a large margin on Machine A (the NVTriStrip solutions, on average out-performed the GA technique solutions by a factor of approximately 1.94). This reflects that NVTriStrip was designed to accommodate the hardware configuration of machine A. However, using machine B, NVTriStrips margin of advantage over the GA technique was not as pronounced.

In the trials run on machine B, the mean performance of NVTriStrip solutions was 49.71ms and the mean performance for the GA technique solutions was 69.27ms. This equates to a factor of 1.54 advantage of the NVTriStrip over the GA technique. NVTriStrip performance on machine B is a product of the hardware assumptions in NVTriStrip where it did not accommodate the hardware configuration on machine B as it did on machine A.

While the NVTriStip is better in practice there are merits in the GA technique. The GA L-System technique is able to optimise the triangle strip by a large amount from a random triangle strip, in the order of a 63% speedup from the first random strip generated to the final result on Machine A. Thus, it does show promise as a technique for further development, separate to the parameter tuning method we tested. Furthermore, the GA technique appears to optimise for characteristics other than the number of triangle strips in a solution.

5.2 NVTriStrip Parameter Tuning Experiment

The aim of this experiment is to investigate the efficacy of using GAs to tune the parameters of an already existing triangle striping method. Again, this is to investigate the possibility of abstracting away hardware setup factors, so that one technique can be applied to many hardware specifications without expending effort in designing algorithms for each hardware target.

The NVTriStrip cache size parameter was optimised by the GA using a 48 bit genome encoding as a parameterized equation A*2^B + C (A 16 bits, B 16 Bits, C 16 bits). This enabled the cache size to be optimised around power of two values.

As in the first experiment, the bitwise representation of the parameters was subjected to a standard mutation and parental cross over scheme. The fitness function was again a rendering efficiency stress test. The results were generated from a single run on each machine with an initial solution population of five solutions, for ten generations.

The machines used were the same specification as those detailed in the first experiment. In addition, the same decimated Stanford Bunny mesh was used to perform the experiments.

Results & Analysis

The following Table 5.3 illustrates the improvements in times brought about by GA cache parameter tuning for machine A and machine B. It lists the cache sizes chosen for each machine, and the efficiency of the final mesh.

Cache Size A	*Cache Size B*	*Msec A*	*Msec B*
195	85	1.34	42.5

Table 5.3 Table of results for machine A and B in the parameter tuning experiment.

By inspection of the results, the approach worked for a relatively easy problem of tuning a parameter, but what should be noted is the way the GA has exploited some algorithmic subtleties in order to improve the triangle strips. The cache

sizes chosen are much larger than normally specified for the hardware (195 & 85), but the meshes generated are equal to or more optimal in comparison to normal NVTriStrip results from the first experiment. Overall there was a 20% improvement in triangle mesh efficiency from the random first triangle string for machine A, and a 63% improvement in efficiency from the initial strip on machine B.

This lends supporting evidence to the assertion that the GA is able to exploit unknown aspects of software and hardware to create optimised meshes, without knowledge of the intricacies of the hardware. This would be very useful in more complex optimisations where other parameters have to be tuned.

6 Conclusion

This paper proposes that genetic algorithms can be used as an alternative to existing triangle stripping techniques for triangulated polygon meshes. This research demonstrated that a GA based technique was able to generate a series of triangle strip solutions for a given triangulated polygon mesh that were improved substantially with successive generations, at times in the order of 60% from the first to final solution.

Compared to the NVTriStrip method, the L-System GA is less affected by change in hardware specifications. This gives support to the concept of letting geometry tuning be performed by evolutionary software, to produce a more consistent solution that is sensitive to hardware and software variability.

If the GA based technique had used an assumption about this feature of the hardware when generating triangle strips from its encoding, the average solution performance on these machines would have been greater than those achieved in the method outlined in this paper. Thus in future a compromise may be required that some general assumptions about target hardware are made, and the GA tunes the geometry with reference to these heuristic rules.

Although the L-System GA technique was able to create and optimise triangle strips solutions for a triangulated polygon mesh, it was unable to create solutions (within an acceptable timeframe) that were as effective as solutions generated by NVTriStrip, due to limitations of the encoding method.

The results of the second parameter tuning experiment indicate the efficacy of using established algorithms as a basis for the application of evolutionary computing to triangle strip optimisation. This example experiment was able to produce optimal triangle strips, by exploiting characteristics of the triangle stripping algorithm and target hardware, thus supporting the hypothesis that such approaches can be used to configure geometric content on hardware.

This overall evolutionary approach to triangle strip optimisation can ensure that the solutions generated accommodate the hardware configuration of each target platform. The parameter optimisation method can thus be used at install/load time for a stand alone game, to allow the content of the game to be tuned to the target hardware on which it is to be played.

6.1 Future Work

The GA technique outlined in this paper starts with a population of randomly generated solutions. The L-System based encoding method outlined in this paper could be improved with a new encoding which allows reverse engineering of existing NVTriStrip solutions of a target mesh into genotypes. Furthermore, the L-System based technique could have a stitching component added to the set of commands, which it appears would improve the efficiency of triangle strip solutions.

The parameter tuning approach will investigate other possible parameters for triangle stripping algorithms that could produce even better solutions than just optimising the cache size parameter. We will also investigate deployment factors for tunable geometry in games and simulation systems using one of the methods tested.

References

ARKIN, E.M., HELD, M., MITCHELL, J.S.B. and SKIENA, S. Hamiltonian triangulations for fast rendering. *The Visual Computer*, *12* (9). 429–444.

BOGOMJAKOV, A. and GOTSMAN, C., Universal rendering sequences for transparent vertex caching of progressive meshes. in *Proceedings of Graphics Interface 2001*, (2001), 81–90.

CHOW, M.M., Optimized geometry compression for real-time rendering. in *Proceedings of the 8th conference on Visualization '97*, (1997), IEEE Computer Society Press, 347–ff.

EL-SANA, J., AZANLI, E. and VARSHNEY, A., Skip strips: maintaining triangle strips for viewdependent rendering. in *Proceedings of the conference on Visualization '99*, (1999), IEEE Computer Society Press, 131–138.

ESTKOWSKI, R., MITCHELL, J.S.B. and XIANG, X., Optimal decomposition of polygonal models into triangle strips. in *Proceedings of the eighteenth annual symposium on Computational geometry*, (2002), ACM Press, 254–263.

EVANS, F., SKIENA, S. and VARSHNEY, A., Optimizing triangle strips for fast rendering. in *Proceedings of the 7th conference on Visualization '96*, (1996), IEEE Computer Society Press, 319–326.

GALib, http://lancet.mit.edu/ga/, 2005

HOLLAND, J.H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press,, 1992.

HOPPE, H., Optimization of mesh locality for transparent vertex caching. in *Siggraph 1999, Computer Graphics Proceedings*, (Los Angeles, 1999), Addison Wesley Longman, 269–276.

HORNBY, G.S. and POLLACK, J.B., The advantages of generative grammatical encodings for physical design. in *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, (Samseong-dong, Gangnam-gu, Seoul, Korea, 2001), IEEE Press, 600–607.

JULSTROM, B.A., Very greedy crossover in a genetic algorithm for the traveling salesman problem. in *Proceedings of the 1995 ACM symposium on Applied computing*, (1995), ACM Press, 324–328.

KORNMANN, D. Fast and simple triangle strip generation., VMS Finland, Espoo, Finland, 1999.

LANDER, J. Game programmer magazine. *IEEE Transactions on Evolutionary Computation*, *8*. 23–28.

MARSHALL, C. Triangle strip creation, optimization, and rendering. in D. Treglia ed. *Game Programming Gems 3*, Charles River Media, 2002, 359–366.

NVIDIA, http://developer.nvidia.com/object/nvtristrip_library.html, 2004

OLIVER MATIAS VAN KAICK, M.V.G.D.S. and PEDRINI, H. Efficient generation of triangle strips from triangulated meshes. *Journal of WSCG*, *12*. 1-3.

SUCGLaboratory, http://graphics.stanford.edu/data/3Dscanrep, 1994