Compressing, Streaming, and Processing of Large Polygon Meshes

PhD Proposal

Martin Isenburg

University of North Carolina at Chapel Hill

February 20, 2004

This document proposes a set of deliverables for which—upon completion—I hope to be given the PhD degree. In the next section I motivate why I should spend four years of my life and several ten-thousand dollars of my supervisor's grant money into the development of various algorithms that operate on polygon meshes. In Section 2 I detail the research contributions that I aim to achieve in the order they lead from one to the next. At the same time I reference the research literature that I will familiarize myself in the course of carrying out the proposed research. In Section 3 I list completed milestones and estimated completion dates for missing milestones. In the last section I anticipate the immediate and the potential impact of my work on the field mesh processing.

1 Motivation

Polygonal meshes represent surfaces in 3D and serve as the de facto standard for fast interactive visualization. A large number of polygons can be required to accurately represent a detailed model. Bandwidth in the mesh rendering pipeline is a limited resource in many graphics applications, which has motivated researchers to find suitably compact mesh representations.

The standard representation of a polygon mesh uses an array of floats to specify the vertex positions and an array of integers containing indices into the vertex array to specify the polygons. Optional mesh properties (e.g. surface normals, texture coordinates, ...) and how they are attached to the mesh is specified in a similar manner. For large and detailed

models, this representation results in files of substantial size, which are expensive to store and slow to transmit.

To reduce transmission times in networked environments, a number of *mesh compression* schemes have been proposed. Their main objective is to reduce the amount of data needed to describe a particular polygonal model. The more compact the description, the smaller the delay when transmitting a model from one computer to another across a network with limited bandwidth.

I propose to investigate various improvements and extensions over existing methods. Mesh compression is still a relatively young research field and has mainly focused on fully triangulated meshes. In particular, I will generalize triangle mesh compression schemes to non-triangular surface meshes and hexahedral volume meshes. Furthermore, I will develop techniques for efficient compression of mesh properties, which have been somewhat neglected in previous work on compression but which are important for fast delivery of the property-rich 3D content of Web applications.

Modern scanning technology has enabled scientists to create polygonal meshes of incredible size. The Atlas statue from Stanford's Digital Michelangelo Project [58], for example, has over 254 million vertices and more than 507 million triangles. Ironically, current mesh compression schemes are not capable—at least not on common desktop PCs—to deal with meshes of the giga-byte size that would most benefit from compression. Current compression algorithms can be used only when connectivity and geometry of the mesh are small enough to reside in main memory. I will investigate how to compress such large meshes in one piece on a standard PC using an out-of-core approach.

The resulting compressed format will allow streaming decompression with a small memory foot-print at speeds that are CPU- and not IO-limited. Furthermore, the mesh access provided by the decompressor will enable a new approach for performing out-of-core computations on large meshes. I will showcase that this streaming representation will allow the design of highly efficient mesh processing algorithms with the example of simplification.

The ease with which meshes in the compressed format can be processed will suggest that any type of streaming representation is much better suited for storing large models than current indexed mesh formats. While conceptually simple, a *streaming mesh* format will

allow to redesign many mesh processing algorithms to work in a *streaming*, possibly *pipelined*, fashion. Furthermore it will eliminate once and for all the problem of *de-referencing* indexed meshes, which hampers most large mesh algorithms and usually has to be resolved in a costly pre-processing step. I will design a streaming mesh format, develop measures for different stream qualities, and investigate how to generate and to compress streaming meshes.

2 Research Plan

This sections details my anticipated research contributions in different areas of mesh processing. The first subsections are concerned with maximal compression of small to mid-sized polygon meshes, which is mostly of relevance to the Web3D community. The last subsections are concerned with much larger data sets that are more relevant to researchers and industries working with scientific simulations, high-resolution 3D scanning, and CAD applications.

2.1 Mesh Compression

To reduce transmission times in networked environments, a number of *mesh compression* schemes have been proposed to reduce the amount of data needed to describe a particular polygonal model. The more compact the description, the smaller the delay when transmitting a model from one computer to another across a network with limited bandwidth. Generally mesh compression techniques have focused on encoding fully triangulated data sets—a natural candidate for the lowest common denominator. Among other things I will investigate how to **encode non-triangular meshes directly in their polygonal representation** and show that avoiding the triangulation step will typically further reduce the storage costs for compressed polygon models.

My dissertation will be concerned only with *single-resolution* or *flat* mesh compression schemes. Alternative approaches reduce the *perceived* transmission delay by initially transmitting a coarse and much smaller approximation of the model to be used until the much larger full resolution version is received. Rather than separately encoding two (or more) versions of a model, *multi-resolution* or *progressive* mesh compression methods first send a small base mesh that is incrementally refined all the way up to the original resolution. I have worked on multi-resolution mesh (MRM) compression in the scope of my PhD during a sum-

mer internship at INTEL. There I have prototyped a compression scheme (patent pending) that will be part of the upcoming CAD exchange format (see www.3dif.org).

Relevant previous work includes graph coding [79, 78, 48, 13, 11], flat compression of connectivity [12, 76, 77, 59, 23, 66, 52, 53, 67, 38, 5, 21, 54, 71, 3, 49], of geometry [12, 76, 77, 45, 56, 14, 46, 57, 15, 55, 9, 69], and of properties [12, 75, 5], progressive compression [74, 25, 63, 10, 4, 2, 47], and compression after remeshing [51, 50, 20, 73].

Connectivity: Most compression efforts have focused on efficient encodings of mesh connectivity as this is where the largest gains are possible. Recent schemes traverse the mesh triangles in a manner that is often described as *region-growing*. They encode the connectivity through a set of symbols that describe this traversal, which are subsequently compressed into a bit-stream using some form of entropy coding [80]. The schemes can be classified into *face-based*, *edge-based*, and *vertex-based* approaches, depending on which mesh element they associate each symbols with. Furthermore we distinguish schemes that require storing explicit *split offsets* as part of the encoding from those that avoid them. Face-based approaches are the Cut-Border Machine [23], which uses offsets, and Edgebreaker [66], which avoids them. Edge-based approaches are the Dual Graph method [59], which uses offsets, and Triangle Fixer [27], which avoids them. The main vertex-based approach is the TG coder [77], which uses offsets, no corresponding method exists that avoids them.

I will **extend two connectivity compression schemes to operate directly on polygonal connectivity**, namely Triangle Fixer [27] and the TG coder [77], and show that this achieves superior bit-rates. This will also suggest that most triangle connectivity coders could be extended to more efficiently compress polygonal connectivity in the following way: deterministically triangulate non-triangular faces and record the original face degrees in addition to the other code symbols, *but* (a) compress them with different contexts *and* (b) exploit the mutual correlation between them by using already compressed symbols to further switch between several contexts.

Geometry: Traditionally, the compression of mesh connectivity and the compression of mesh geometry are done by clearly separated (but often interwoven) techniques. Most ge-

ometry compression schemes use the traversal order that the connectivity coder induces on the vertices to compress their associated positions with a predictive coding scheme. Previously decoded positions are used to predict the next position and only a corrective vector is stored. The predictive coding schemes employed in practice exclusively use simple linear predictors [12, 76, 77].

The reasons for the popularity of linear prediction schemes are that (a) they are simple to implement robustly, (b) compression or at least decompression is fast, and (c) they deliver good compression rates. For nearly six years the simple parallelogram predictor by Touma and Gotsman [77] has been the accepted benchmark that recent approaches compare themselves with. Although better compression rates have been reported, it is usually questionable whether these gains are justified in practice given the sometimes immense increase in algorithmic and asymptotic complexity of the encoding and decoding schemes. Furthermore, these improvements are often specific to a certain type of mesh. Some methods achieve significant gains only on models with sharp features, while others are applicable only to smooth and densely sampled meshes.

Current predictive geometry compression schemes work as follows: First the floating-point positions are uniformly quantized using a user-defined precision of, for example, 8, 12, or 16 bits per coordinate. Then a prediction rule is applied that uses previously decoded positions to predict the next position and only an offset vector is stored, which corrects the predicted position to the actual position. The values of the resulting corrective vectors tend to spread around zero. This reduces the variation, and thereby the entropy, of the sequence of numbers, which means they can be effectively compressed with an arithmetic coder [80].

Despite the availability of excellent mesh compression techniques, scientists and engineers often refrain from using them because they modify the mesh data. Although connectivity is encoded in a lossless manner, the floating-point coordinates associated with the vertices are altered when they are quantized onto a uniform integer grid. Although a fine enough grid can usually represent the data with sufficient precision, the original floating-point values are slightly changed. Therefore many people refrain from using compression altogether. I will investigate how to **compress floating-point coordinates with predictive coding in a completely lossless manner** by omitting the initial quantization step and calculating the

predictions in floating-point arithmetic. The predicted and the actual floating-point values will then be broken up into sign, exponent, and mantissa and their corrections will be compressed separately with context-based arithmetic coding. As the quality of the predictions is expected to vary with the exponent, we will use it to switch between different arithmetic contexts. This will be a completing rather than a competing technology that can be used whenever quantization of the floating-point values is—for whatever reason—not an option.

The popular parallelogram predictor introduced by Touma and Gotsman [77] predicts vertex position of triangular meshes to complete the parallelogram that is spanned by three previously processed vertices of a neighboring triangle. I will investigate how to **extend the parallelogram prediction rule to have better performance on polygonal meshes** without increasing its complexity. The premise here is that polygonal faces tend to be fairly planar and convex. Although they are usually not perfectly planar, major discontinuities are improbable to occur across them—otherwise they would have been triangulated when the model was designed. Using information about polygons to perform predictions *within* a polygon rather than *across* polygons should lead to better results.

Using connectivity information to guide the prediction of geometry makes intuitive sense since we expect a strong correlation among the vertex positions of neighboring triangles. In regular samples meshes, where the variation in triangle area is small, we expect this correlation to be especially large. I will investigate how to **recover information about the geometric shape solely from the connectivity graph** of such meshes. This will raise the question whether it is possible to represent shapes through connectivity alone.

Properties: Previous research in mesh compression has mostly focused on connectivity and geometry coding; the compression of properties has received less attention. There are two kinds of information to compress. One specifies each individual property—the property values. The other describes how the properties are attached to the mesh—the property mapping. The compression of per-vertex mappings is straight-forward: properties are mapped and predicted in the same manner as vertex positions. The compression of per-corner mappings, on the other hand, has not been sufficiently addressed. Such mappings accommodate discontinuities in the normal field or the texturing of a polygon mesh. I will **investigate**

predictive compression schemes for mappings with discontinuities. This will include predictive compression of the property mapping as well as the property values. I will predict the characteristics of typical mappings [39] by classifying corners, edges, and vertices as *smooth* or *crease* [39] and by switching arithmetic contexts based on the correlation between them. I will avoid unreasonable predictions of property values in the presence of mapping discontinuities by switching between a set of simple predictions rules. In particular, I will focus here on texture coordinate compression.

2.2 Compressing Stripified Meshes

For interactive visualization not only the speed at which a triangle mesh can be received is important, but also the speed at which it can be displayed. Here the bottleneck is not the data rate at which a mesh can be sent across a network, but the data rate at which it can be sent to the rendering engine. Each triangle of the mesh can be rendered individually by sending its three vertices to the graphics hardware. Then every mesh vertex is processed about six times, which involves passing its three coordinates and optional normal, colour, and texture information from the memory to and through the graphics pipeline.

A common technique to reduce the number of times this data needs to be transmitted is to send long runs of adjacent triangles. Such triangle strips [17, 81] are widely supported by today's graphics software and hardware. Two vertices from a previous triangle are re-used for all but the first triangle of every strip. Depending on the quality of the triangle strips this can potentially reduce the number of vertex repetitions by a factor of three. Since computing an optimal set of triangle strips is NP-complete [16], in practice one resorts to a variety of heuristics for generating good triangle strips [17, 70, 82].

Given the difficulty of generating good triangle strips it would be desirable to do this just once and store the computed stripification together with the mesh. However, currently available connectivity compression techniques do not support the encoding of stripified meshes. Obviously one can enhance any existing compression method by encoding the stripification separately and concatenating the results. However, such a two-pass technique adds unnecessary overhead—it does not exploit the correlation between the connectivity and the stripification of a mesh. I will investigate how to **encode the connectivity and the strip-**

ification of a triangle mesh in an interwoven fashion that fully exploits the correlation existing between the two.

2.3 Compressing Volume Meshes

Unstructured volume meshes can be found in a broad spectrum of scientific and industrial applications including fluid mechanics, thermodynamics and structural mechanics, where such volumetric data is used for both, computation and visualization. Traditionally unstructured volume meshes were composed of tetrahedral elements, but recently also other polyhedra have become popular. Especially hexahedral volume meshes are often used, because of their numerical advantages in finite element computations.

The standard representation for hexahedral meshes uses three floating-point coordinates per vertex to store geometry and eight integer indices per hexahedron to store connectivity. Optionally there are application-specific mesh *properties* such as density or pressure values that are attached to the vertices. For meshes with v vertices and h hexahedra this representation requires $96v$ bits for the geometry and $256h$ bits for the connectivity when using standard 4 byte data types.

For archival, storage, and transmission of the data, a more compact representation is beneficial. There have been several publications concerning the compression of tetrahedral volume meshes [72, 22, 64, 83], but I am not aware of a compression scheme that can handle hexahedral volume meshes. For tetrahedral geometry the best coder I am aware of [22] achieves an average geometry compression ratio of only $1 : 1.6$ after quantizing each coordinate to 16 bits of precision. The authors report that more sophisticated prediction schemes failed, essentially because "tetrahedral meshes are too irregular to predict vertex coordinates much better than with the proximity information of the connectivity alone".

Tetrahedral volume meshes seem irregular by nature because its elements do not allow a requlars tiling of the domain. While the equilateral triangle, for example, tiles the 2D space, the equilateral tetrahedron does not tile the 3D space. This is different for hexahedral meshes. A cube is a hexahedron whose six faces are square and meet each other at right angles. It is the only of the five platonic solids that regularly tiles the 3D domain. This suggests that hexahedral meshes will exhibit significantly more regularity than tetrahedral

meshes and that a compression scheme should be able to exploit that.

Compression schemes for surface meshes based on *degree coding* for the connectivity and *parallelogram prediction* for the geometry automatically adapt to regularity in the mesh. For example, a highly regular triangular mesh composed mostly of equilateral triangles will have a low-dispersion vertex degree distribution with an entropy close to zero, while each pair of adjacent triangles will be in a roughly "parallelogram-shaped" configuration. The same can be observed for non-triangular meshes that correspond to the the other two regular tilings of the 2D domain: squares and regular hexagons. I will investigate how to **extend these compression schemes from surface meshes to volume meshes** while preserving their adaptivity to this kind of regularity.

Going from surface meshes to volume meshes, one can think of the vertices getting stretched into edges. What was a vertex degree in the surface mesh becomes an edge degree in the volume mesh. Hence, the concept of degree coding can be extended to compress the connectivity of volume meshes using *edge degrees*. This will work both for tetrahedral and hexehedral connectivity. However, initial measurements on the edge degree distributions of tetrahedral meshes suggest that the compression rates achievable by degree coding will be worse than those reported by other methods. Hexahedral meshes, on the other hand, exhibit a low dispersion in edge degrees indicating that degree-coding is well suited for them.

2.4 Compressing Gigantic Meshes

Modern scanning technology enables scientists to create digital 3D representations of real-world objects with incredible detail. The Atlas statue from Stanford's Digital Michelangelo Project [58], for example, has over 254 million vertices and more than 507 million triangles. If represented in a standard indexed mesh, this corresponds to 6 gigabytes of triangle and 3 gigabytes of vertex data. Ironically, mesh compression schemes are not capable—at least not on common desktop PCs—to deal with meshes of the giga-byte size that would most benefit from compression. Current compression algorithms and for the most part also their corresponding decompression algorithms can be used only when connectivity and geometry of the mesh are small enough to reside in main memory. Realizing this limitation, Ho et al. [24] propose to cut gigantic meshes into manageable pieces and encode each separately using

existing techniques. However, partitioning the mesh introduces artificial discontinuities. The special treatment required to deal with these cuts not only lowers compression rates but also significantly reduces decompression speeds.

Up to a certain mesh size, the memory requirements of the compression process could be satisfied using a 64-bit super-computer with vast amounts of main memory. Research labs and industries that create giga-byte sized meshes often have access to such equipment. But to decompress on common desktop PCs, at least the memory foot-print of the decompression process needs to be small. In particular, its memory requirements must be less than the size of the decompressed mesh. This eliminates a number of popular *multi-pass* schemes that either need to store the entire mesh for connectivity decompression [76, 66, 5] or that decompress connectivity and geometry in separate passes [39, 45, 73].

This leaves us with all *one-pass* coders that can perform decompression in a single, memory-limited pass over the mesh. Such schemes compress and decompress connectivity and geometry information in an interwoven fashion. This allows *streaming* decompression that can start producing mesh triangles as soon as the first few bytes have been read. There are several schemes that could be implemented as one-pass coders [77, 23, 59, 57].

I will investigate how how to **compress meshes of giga-byte size in one piece on a standard PC using an external memory data structure** that provides transparent access to arbitrary large meshes. This data structure will accommodates the access pattern of a one-pass compression engine to reduce costly loads of data from disk. The resulting compressed format will allow streaming, small memory foot-print decompression at only CPU and not IO limited speeds. This format will have benefits beyond efficient storage and fast loading. It will be a better input format to algorithms that need to perform out-of-core computations on large meshes than standard indexed mesh formats, which are inefficient to work with and often need to be *de-referenced* in a costly pre-processing step or the resulting polygon soups, which are at least twice as big and provide no connectivity information.

2.5 Large Mesh Processing

The straight-forward approach for processing large meshes is to cut them into pieces small enough to fit into main memory and then processes each piece separately while giving special

treatment to the split boundaries. Mesh cutting has successfully been used to, for example, distribute [58], simplify [26, 65, 6] and compress [24] very large polygon models. Despite the apparent simplicity of this approach, the initial cutting step can be fairly expensive when the input mesh is in a standard indexed format.

More recent strategies are *batched* and *online* processing, which avoid the cutting step and process the mesh as a whole: For the first, the data streams in one or more passes though the main memory and computations are restricted to the small amount of data that is kept in memory at any time. For the other, the data is processed through a series of potentially random queries. In order to avoid costly disk access with each query the data is re-organized to accommodate an anticipated access pattern.

Batch processing is CPU efficient and has mainly been used for out-of-core mesh simplification [60, 61, 68, 19]. These schemes operate on de-referenced *triangle soup* in increments of single triangles. The output of this pass either is small enough to fit in memory so that the remaining computation can be done in-core or is directly written to a file, which is then used as input for subsequent batch-processing passes.

Out-of-core approaches based on batch processing are designed to work without explicit connectivity information. This enables them to efficiently do their computations using a few batch processing passes instead of requiring costly online processing on the entire mesh. Large meshes that must be processed out-of-core are therefore treated differently from small meshes that can be processed in-core. Unfortunately, the output of batch-processing based algorithms tends to be of lower quality than that of algorithms that have access to explicit connectivity information.

Online processing is usually much more I/O-limited but makes it possible to run typical in-core algorithms on data sets too large to fit into main memory. This requires data structures that separate references to the mesh data from the physical location of the data in memory. Some schemes simply use the virtual memory functionality of the operating system and try to organize the data accesses such that the number of page faults is minimized [62, 7]. This, however, works only as long as the data amount to less than 4 gigabytes. Going beyond that limit requires dedicated external memory data structures that manage the virtual address space for the data explicitely. These can be accelerated by

caching or *pre-fetching* of data that is likely to be queried soon.

Cignoni et. al [8], for example, propose an octree-based external memory data structure that makes it possible to simplify the St. Matthew statue from 386 to 94 million triangles using iterative edge contraction [18]. Similarly, the out-of-core mesh I mentioned in the last section will make it possible to compress the St. Matthew statue from over 6 GB to less than 400 MB of data using a compressor based on region-growing [77].

Such external memory data structures *enable* traditional in-core algorithms to be applied to large data sets by providing seemingly random access to complete connectivity information in a transparent manner. However, building these data structures is time and space consuming and using these data structures tends to be slow.

I want to explore a completely novel approach to large mesh processing. It is based on the mesh access provided by the streaming one-pass decompressor mentioned in the last section. I will show that this will be useful for all types of mesh processing using the example of simplification. It will **combine the efficiency of batch-processing with the advantages of explicit connectivity information available in online-processing** by restricting the access to the mesh to a fixed traversal order, but at the same time providing full connectivity for the active elements of this traversal.

2.6 Streaming Meshes

The compressed format outlined in the last section allows to stream large meshes while using a relatively small amount of memory. This ability comes from two things: (a) decompressing triangles and vertices in an interleaved fashion gives us streaming, and (b) having knowledge about when a vertex is used for the last time (e.g. knowledge about when it is safe to discard) gives us small memory footprints. As these two things are conceptually simple, one could generally store meshes this way. I will investigate how to **design a streaming mesh format that provides streaming access with small memory footprint to large meshes** that is light-weight enough to be considered as a standard mesh format alongside PLY or OBJ.

Today's standard mesh formats (e.g. PLY, OBJ, IV, OFF, VRML) mesh formats use an array of floats that specifies the vertex positions followed by an array of indices into the vertex array that specifies the polygons. No constraints are imposed on the order of either

vertices or triangles. In particular, the three vertices of a triangle can be located *anywhere* in the vertex array. They need not even be close to each other. And while subsequent triangles may reference vertices at opposite ends of the array, the first and the last triangle could use the same vertex. This flexibility was enjoyable as long as meshes were small or moderately sized. However, with the arrival of gigabyte-sized data sets, this has become a major headache.

These formats were designed in the early days of computer graphics when the Stanford bunny was considered a complex model. This model, which has helped popularize Stanford's PLY format, abuses this flexibility like no other—its vertices and triangles are pretty randomly distributed in their respective arrays. Nowadays the PLY format is used to archive the scanned statues that were created by Stanford's Digital Michelangelo Project [58]. The Atlas statue, for example, has over 254 million vertices and more than 507 million triangles. Having a gigabyte-sized block of triangle data that indexes a gigabyte-sized block of vertex data unduly complicates all subsequent processing.

A *streaming mesh* format will eliminate once and for all the problem of *de-referencing* indexed meshes, which hampers all large mesh algorithms and is usually resolved in a costly pre-processing step. Furthermore it will allow to redesign many mesh processing algorithms to work in a *streaming*, possibly *pipelined*, fashion. Besides designing a streaming mesh format, I will **define metrics that measures different stream qualities**, **develop out-of-core techniques for converting existing data into this streaming representation**, and finally show how to **efficiently compress streaming meshes on-the-fly** directly in their particular stream order.

3 Milestones

The following milestones have already been completed:

- coursework
- teaching requirement
- product requirement
- language requirement

- research
  - **connectivity compression:** edge-based without offsets [27, 28, 39], polygonal [39, 29], stripified [28], hexahedral [30], out-of-core [33], streaming [35].
  - **geometry compression:** polygonal [31], hexahedral [30], lossless [37], out-of-core [33], streaming [35].
  - **property compression:** mapping [39, 40], texture coordinates [44].
  - **compression strategies:** with connectivity [34], for textual formats [42, 41, 43], online benchmark [32], out-of-core [33, 35], streaming [35].
  - **meshing:** unit edge [34], with isotropic [1].
  - **streaming processing:** decompression [33], simplification [36], compression [35].

The following milestones will be completed soon:

- integrative paper (80% remaining, end of april)
- proposal (early march)
- oral exam (early march)
- dissertation (20% remaining, late may)
- defense (late april / early may)
- travel requirement (15% remaining)

4 Impact of the Work

The research activities proposed here will significantly advance the field of mesh compression and large model processing. The main impact will come from the contributions described in subsections 2.4, 2.5 and 2.6. These contribution are *enabling* in the sense that they will allow researchers to operate on large meshes that used to require access to supercomputers. They are *innovative* in the sense that they will lead to a new breed of streaming mesh algorithms that perform online computations at batch-processing speeds.

Replacing indexed or immediate mesh formats with any type of streaming format (optionally compressed) as the raw data representation for large meshes will be beneficial to many researchers. I have spent days just to acquire and pre-process the large meshes of Stanford's Michelangelo Project. The largest scanned statues had to be cut into several pieces

in order to make the download over the Internet feasable. The download of these pieces can take on the order of hours; even loading them from the harddrive into memory takes tens of minutes before any processing can take place. I had to write dedicated out-of-core software to stich the pieces back together. This painstaking pre-processing step created gigabytes of intermediate data files and significant delayed my actual research.

The distribution of large data sets in compressed form is much easier. After completing design and implementation of our out-of-core compressor and the corresponding decompressor I will be able to burn the six gigantic example meshes that we have acquired over the months—originally over 10 Gigabytes—onto a single CD. Loading these data sets from the compressed representation is much faster, copying those data files between file systems no longer lasts the entire lunch break, and downloading them from a remote site is not an overnight job anymore.

The results of the research proposed here will be disseminated in form of a conference and journal publications, as well as technical reports. Especially the work on streaming meshes will be also made available in form of source code and a well-documented API. This will enable other researchers to develop their own streaming mesh algorithm either by following the examples or by using the code in full or in parts. Furthermore, I will make commonly used large models available in a compressed format and provide others with the necessary tools for compressing their own models. Ultimately, I would like to convince Stanford to offer the large models of their Scanning Repository in a compressed streaming format.

References

[1] P. Alliez, E. Colin de Verdiere, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In *Proceedings of Shape Modeling International'03*, pages 49–58, 2003.

[2] P. Alliez and M. Desbrun. Progressive encoding for lossless transmission of 3D meshes. In *SIGGRAPH'01 Proceedings*, pages 198–205, 2001.

[3] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Eurographics'01 Proceedings*, pages 480–489, 2001.

[4] C. Bajaj, V. Pascucci, and G. Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Visualization'99 Proceedings*, pages 307–316, 1999.

[5] C. Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *Data Compression Conference'99 Proceedings*, pages 247–256, 1999.

[6] F. Bernardini, I. Martin, J. Mittleman, H. Rushmeier, and G. Taubin. Building a digital model of Michelangelo's Florentine Pieta. *IEEE Computer Graphics and Applications*, 22(1):59–67, 2002.

[7] P. Choudhury and B. Watson. Completely adaptive simplification of massive meshes. Technical Report CS–02–09, Northwestern University, 2002.

[8] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics*, 2003. To appear.

[9] D. Cohen-Or, R. Cohen, and R. Irony. Multi-way geometry encoding. Technical Report TR-2002, Tel-Aviv University, 2002.

[10] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *Visualization'99 Proceedings*, pages 67–72, 1999.

[11] L. de Floriani, P. Magillo, and E. Puppo. A simple and efficient sequential encoding for triangle meshes. In *Proceedings of 15th European Workshop on Computational Geometry*, pages 129–133, 1999.

[12] M. Deering. Geometry compression. In *SIGGRAPH 95 Conference Proceedings*, pages 13–20, 1995.

[13] M. Denny and C. Sohler. Encoding a triangulation as a permutation of its point set. In *Proceedings of 9th Canadian Conference on Computational Geometry*, pages 39–43, 1997.

[14] O. Devillers and P.-M. Gandoin. Geometric compression for interactive transmission. In *Proc. of IEEE Visualization 2000*, pages 319–326, 2000.

[15] O. Devillers and P.-M. Gandoin. Progressive and lossless compression of arbitrary simplicial complexes. In *SIGGRAPH'02 Proceedings*, pages 372–379, 2002.

[16] F. Evans, S. S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical report, Department of Computer Science, State University of New York at Stony Brook, 1996.

[17] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Visualization'96 Proceedings*, pages 319–326, 1996.

[18] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH'97 Proceedings*, pages 209–216, 1997.

[19] M. Garland and E. Shaffer. A multiphase approach to efficient surface simplification. In *Visualization'02 Proceedings*, pages 117–124, 2002.

[20] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *SIGGRAPH'02 Proceedings*, pages 355–361, 2002.

[21] A. Guéziec, F. Bossen, G. Taubin, and C. Silva. Efficient compression of non-manifold polygonal meshes. In *Visualization'99 Proceedings*, pages 73–80, 1999.

[22] S. Gumhold, S. Guthe, and W. Strasser. Tetrahedral mesh compression with the cut-border machine. In *Visualization'99 Proceedings*, pages 51–58, 1999.

[23] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98 Proceedings*, pages 133–140, 1998.

[24] J. Ho, K. Lee, and D. Kriegman. Compressing large polygonal models. In *Visualization'01 Proceedings*, pages 357–362, 2001.

[25] H. Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

[26] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Visualization'98 Proceedings*, pages 35–42, 1998.

[27] M. Isenburg. Triangle Fixer: Edge-based connectivity compression. In *Proceedings of 16th European Workshop on Computational Geometry*, pages 18–23, 2000.

[28] M. Isenburg. Triangle Strip Compression. In *Graphics Interface'00 Proceedings*, pages 197–204, 2000.

[29] M. Isenburg. Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface'02 Proceedings*, pages 161–170, 2002.

[30] M. Isenburg and P. Alliez. Compressing hexahedral volume meshes. In *Pacific Graphics'02 Conference Proceedings*, pages 284–293, 2002.

[31] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *Visualization'02 Proceedings*, pages 141–146, 2002.

[32] M. Isenburg, P. Alliez, and J. Snoeyink. A benchmark coder for polygon mesh compression. *http://www.cs.unc.edu/~isenburg/pmc/*, 2002.

[33] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH'03 Proceedings*, pages 935–942, 2003.

[34] M. Isenburg, S. Gumhold, and C. Gotsman. Connectivity shapes. In *Visualization'01 Proceedings*, pages 135–142, 2001.

[35] M. Isenburg and P. Lindstrom. Streaming meshes. *submitted*, 2004.

[36] M. Isenburg, P. Lindstrom, S. Gumhold, and J. Snoeyink. Large mesh simplification using processing sequences. In *Visualization'03 Proceedings*, pages 465–472, 2003.

[37] M. Isenburg, P. Lindstrom, and J. Snoeyink. Lossless compression of floating point geometry. In *to appear in CAD'04*, 2004.

[38] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *Proceedings of SIBGRAPI'99 - 12th Brazilian Symposium on Computer Graphics and Image Processing*, pages 27–28, 1999.

[39] M. Isenburg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. In *SIGGRAPH'00 Proceedings*, pages 263–270, 2000.

[40] M. Isenburg and J. Snoeyink. Compressing the property mapping of polygon meshes. In *Pacific Graphics'01 Proceedings*, pages 4–11, 2001.

[41] M. Isenburg and J. Snoeyink. Coding with ASCII: compact, yet text-based 3d content. In *Proceedings of the 1st International Symposium on 3D Data Processing, Visualization and Transmission*, pages 609–616, 2002.

[42] M. Isenburg and J. Snoeyink. Compressing polygon meshes as compressable ASCII. In *Proceedings of Web3D'02 Symposium*, pages 1–10, 2002.

[43] M. Isenburg and J. Snoeyink. Binary compression rates for ASCII formats. In *Proceedings of Web3D'02 Symposium*, pages 173–178, 2003.

[44] M. Isenburg and J. Snoeyink. Compressing texture coordinates with selective linear predictions. In *Proceedings of Computer Graphics International'03*, pages 126–131, 2003.

[45] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH'00 Proceedings*, pages 279–286, 2000.

[46] Z. Karni and C. Gotsman. 3D mesh compression using fixed spectral bases. In *Graphics Interface'01 Proceedings*, pages 1–8, 2001.

[47] Zachi Karni, Alexander Bogomjakov, and Craig Gotsman. Efficient compression and rendering of multi-resolution meshes. In *Visualization'02*, pages 347–354, 2002.

[48] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. In *Discrete Applied Mathematics*, pages 239–252, 1995.

[49] A. Khodakovsky, P. Alliez, M. Desbrun, and P. Schroeder. Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models*, 64(3-4):147–168, 2002.

[50] A. Khodakovsky and I. Guskov. Compression of normal meshes. In *Geometric Modeling for Scientific Visualization, Springer-Verlag*, pages 189–206, 2002.

[51] A. Khodakovsky, P. Schroeder, and W. Sweldens. Progressive geometry compression. In *SIGGRAPH'00 Proceedings*, pages 271–278, 2000.

[52] D. King and J. Rossignac. Guaranteed $3.67v$ bit encoding of planar triangle graphs. In *Proceedings of 11th Canadian Conference on Computational Geometry*, pages 146–149, 1999.

[53] D. King, J. Rossignac, and A. Szymczak. Connectivity compression for irregular quadrilateral meshes. Technical Report TR–99–36, GVU Center, Georgia Tech, November 1999.

[54] B. Kronrod and C. Gotsman. Efficient coding of non-triangular meshes. In *Proceedings of Pacific Graphics*, pages 235–242, 2000.

[55] B. Kronrod and C. Gotsman. Optimized compression of triangle mesh geometry using prediction trees. In *International Symposium on 3D Data Processing Visualization and Transmission*, pages 602–608, 2002.

[56] E. Lee and H. Ko. Vertex data compression for triangular meshes. In *Proceedings of Pacific Graphics*, pages 225–234, 2000.

[57] H. Lee, P. Alliez, and M. Desbrun. Angle-analyzer: A triangle-quad mesh codec. In *Eurographics'02 Proceedings*, pages 198–205, 2002.

[58] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The Digital Michelangelo Project. In *SIGGRAPH 00 Proceedings*, pages 131–144, 2000.

[59] J. Li and C. C. Kuo. A dual graph approach to 3D triangular mesh compression. In *Proceedings of ICIP'98*, pages 891–894, 1998.

[60] P. Lindstrom. Out-of-core simplification of large polygonal models. In *SIGGRAPH 00 Proceedings*, pages 259–262, 2000.

[61] P. Lindstrom and C. Silva. A memory insensitive technique for large model simplification. In *Visualization'01 Proceedings*, pages 121–126, 2001.

[62] S. McMains, J. Hellerstein, and C. Sequin. Out-of-core build of a topological data structure from polygon soup. In *Proceedings of the 6th ACM Symposium on Solid Modeling and Applications*, pages 171–182, 2001.

[63] R. Pajarola and J. Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, 2000.

[64] R. Pajarola, J. Rossignac, and A. Szymczak. Implant sprays: Compression of progressive tetrahedral mesh connectivity. In *Visualization'99 Proceedings*, pages 299–306, 1999.

[65] C. Prince. Progressive meshes for large models of arbitrary topology. Master's thesis, University of Washington, 2000.

[66] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[67] J. Rossignac and A. Szymczak. Wrap&zip: Linear decoding of planar triangle graphs. *The Journal of Computational Geometry, Theory and Applications*, 1999.

[68] E. Shaffer and M. Garland. Efficient adaptive simplification of massive meshes. In *Visualization'01 Proceedings*, pages 127–134, 2001.

[69] O. Sorkine, D. Cohen-Or, and S. Toledo. High-pass quantization for mesh encoding. In *Proceedings of Symposium on Geometry Processing'03*, pages 42–51, 2003.

[70] B. Speckmann and J. Snoeyink. Easy triangle strips for TIN terrain models. In *Proceedings of 9th Canadian Conference on Computational Geometry*, pages 239–244, 1997.

[71] A. Szymczak, D. King, and J. Rossignac. An Edgebreaker-based efficient compression scheme for connectivity of regular meshes. In *Proceedings of 12th Canadian Conference on Computational Geometry*, pages 257–264, 2000.

[72] A. Szymczak and J. Rossignac. Grow & fold: Compression of tetrahedral meshes. In *Proceedings of the 5th ACM Symposium on Solid Modeling and Applications*, pages 54–64, 1999.

[73] A. Szymczak, J. Rossignac, and D. King. Piecewise regular meshes: Construction and compression. *Graphical Models*, 64(3-4):183–198, 2002.

[74] G. Taubin, A. Guéziec, W.P. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH'98 Proceedings*, pages 123–132, 1998.

[75] G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proceedings of the IEEE*, 86(6):1228–1243, 1998.

[76] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[77] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface'98 Proceedings*, pages 26–34, 1998.

[78] G. Turan. Succinct representations of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.

[79] W.T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.

[80] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[81] M. Woo, J. Neider, and T. Davis. *Open GL Programming Guide.* Addison Wesley, 1996.

[82] X. Xiang, M. Held, and J. Mitchell. Fast and efficient stripification of polygonal surface models. In *Proceedings of Interactive 3D Graphics*, pages 71–78, 1999.

[83] C. Yang, T. Mitra, and T. Chiueh. On-the-fly rendering of losslessly compressed irregular volume data. In *Visualization'00 Proceedings*, pages 101–108, 2000.