

Geometry Decompression Hardware

Michael Deering

Sun Microsystems

Material Notes

The following material is a combination of the actual slides that will be used in this section of the course interspersed with more detailed notes (labeled (Notes)). This was done because the slides tend to be terse speaking points; the ancillary material should help explain the points being made.

After the slides, an updated version of the Hardware-ready binary compression format used in the Java 3D™ API and some versions of OpenGL™ are included. This document contains much additional background on the techniques used for hardware geometry decompression. Even more updated versions of this document are accessible from the web, at the URL below:

`java.sun.com/products/java-media/3D/`

This is the web page for the Java 3D API. The geometry compression specification currently resides in Appendix B of the specification, which is accessible from this page.

Focus

- **Compressed geometry formats tuned for hardware-accelerated decompression within rendering hardware**
- **Why hardware oriented formats are different**
- **Where such formats are useful**

What are the Benefits of Geometry Compression?

- **Considerably less space on storage media.**
- **Considerably less bandwidth needed on networks and reduced transmission times.**

What are the Benefits of Hardware Geometry Decompression?

- **Requires considerably less space in main memory.**
- **Requires considerably less bandwidth on bus between memory and 3D graphics subsystem.**

If standardized, a machine independent 'executable' format can be directly transferred from disk or network to main memory without any re-processing.

Performance: 10-15X less space and/or bandwidth.

DEFINITION OF TERM: “in-memory ‘executable’ format” (Notes)

An “in-memory ‘executable’ format” is an application high level language (C, Fortran, etc.) data format that allows descriptions of geometry to be rendered and passed to a given immediate mode 3D graphics API with the least amount of overhead. Typically, this format would be one or more arrays of data structures comprised of single-precision floating point numbers, representing 3D vertex position information, and possibly including normal vector and/or color vertex or facet information (and more recently, possibly texture map coordinates). Sometimes additional flag data would be present to indicate chaining rules between vertices; zig-zag triangle strips, fan triangle strips, etc. Such APIs include IrisGL, StarBase, XGL, OpenGL, as well as formats that also support non-immediate mode: Doré, QuickDraw3D, PHIGS, Inventor, etc. In such formats the storage space for a triangle is typically in the range of 50 to 100 bytes, depending on chaining and attributes included.

However, the entire scene data is not always stored in such an ‘executable’ format. Rather, many applications store the equivalent data in their own specialized format, and while walking their own display list, pay the computational overhead to convert the data to the API’s immediate mode format on the fly every frame. In the past, the conversion could be performed faster than the hardware could render the data, and so the rendering time penalty for systems that did on-the-fly conversion was typically less than a factor of two. However, because 3D

DEFINITION OF TERM: “in-memory ‘executable’ format” (continued)

graphics hardware rendering rates have been getting faster than CPU computational rates, this is no longer the case; for some machines there is a 3X to 5X penalty for not pre-converting geometry to ‘optimal’ in-memory rendering formats. In some systems, this ‘optimal’ format is an opaque display list format, and cannot be pre-computed. Thus some systems have taken to caching a second copy of the geometry in the render-ready format, but this doubles the storage that must be allocated for geometry.

When the in-memory ‘executable’ formats of most existing 3D graphics APIs were being designed, typical workstation hardware triangle rendering rates were on the order of only tens of thousands of triangles per second, with the very highest-end hardware less than tens times faster, in the hundreds of thousands of triangles per second range. Even at 50 to 100 bytes of storage per triangle, a scene that took a quarter second to render would only require a few megabytes to store. But with today’s inexpensive workstations rendering several million triangles per second, a quarter second worth of geometry can take nearly a hundred megabytes of memory.

These two storage dilemmas were the driving force behind the creation of the first optimized geometry compression format [Deering95]. This format was optimized to be directly decompressed by a hardware geometry decompressor at multi-million triangle per second rates.

DEFINITION OF TERM: “in-memory ‘executable’ format” (continued)

Most other geometry compression formats proposed since that time have been designed for purely software decompression, and are optimized to minimize storage space on disk, or transmission bandwidth over networks. Some of these techniques also fold in the ability to progressively transmit geometry at successively higher resolutions, or the ability to generate different levels of detail dynamically.

The real-time requirement of decompressing a triangle in less than 100 nanoseconds severely constrains the sorts of decompression algorithms that can be applied. Most other types of compression formats that have hardware implementations have considerably slower required conversion rates. (These include Fax, MPEG I & II, and various audio formats.)

Applications of HW Decompression

- **MCAD**
- **Walk-throughs**
- **Web pages**
- **On line 3D documentation**

Applications of HW Decompression

- **Games**
- **Scientific visualization (beware of quantization limits)**
- **Virtual reality**
- **3D movies: HoloFlicks**

Related Work

- Deering, M. Geometry Compression. *Computer Graphics* (Proc. SIGGRAPH '95), pages 13-20, August 1995.
- Taubin, G., and Rossignac, J. Geometry Compression Through Topological Surgery. IBM RC-20340, 1996. <http://www.research.ibm.com/vrml/binary>
- Hoppe, H. Progressive Meshes. *Computer Graphics* (Proc. SIGGRAPH '96), pages 99-108, August 1996.
- Chow, M. Optimized Geometry Compression for Real-time Rendering. *IEEE Visualization '97*, pages 347-354, October 1997. <http://home.earthlink.net/~mmchow>
- Staadt, O., M. Gross, and R. Weber. Multiresolution Compression and Reconstruction. *IEEE Visualization '97*, pages 337-346, October 1997.

What are We Compressing?

■ Primary triangulated surfaces

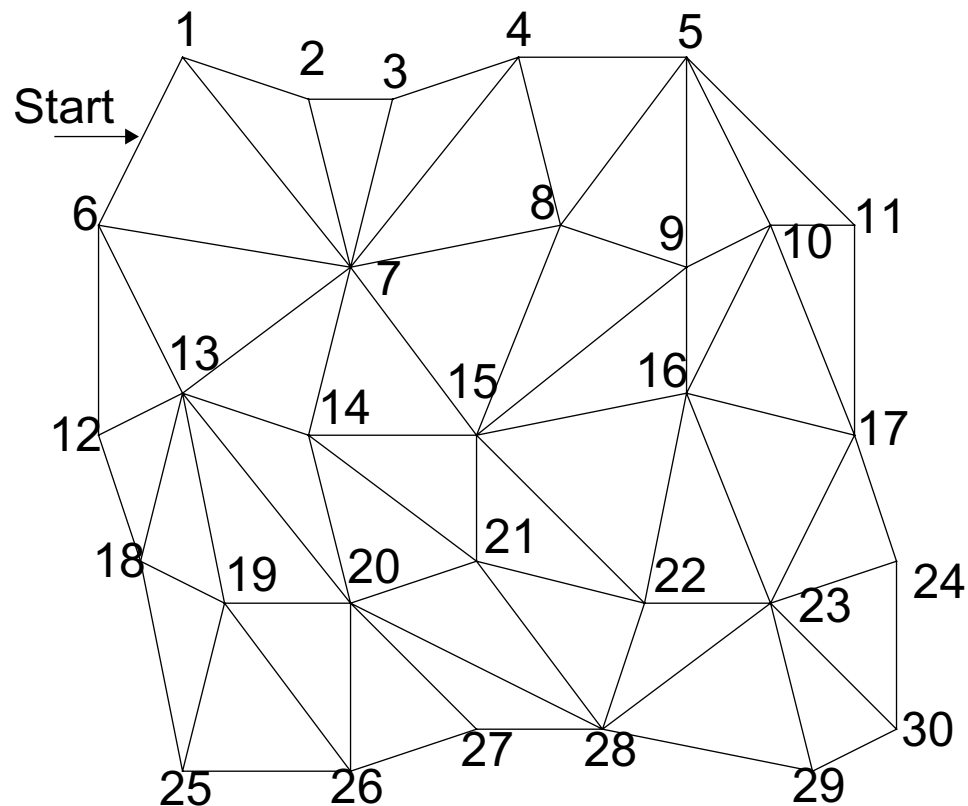
■ Per-vertex attributes:

- position: $x\ y\ z$
- color: $r\ g\ b\ a$
- normal: $n_x\ n_y\ n_z$
- texture map coordinates: $r\ s\ t$

Triangle Vertex Data Can Be:

- **Position normal**
- **Position normal color**
- **Position color**
- **Position**

What is a Generalized Triangle Mesh?



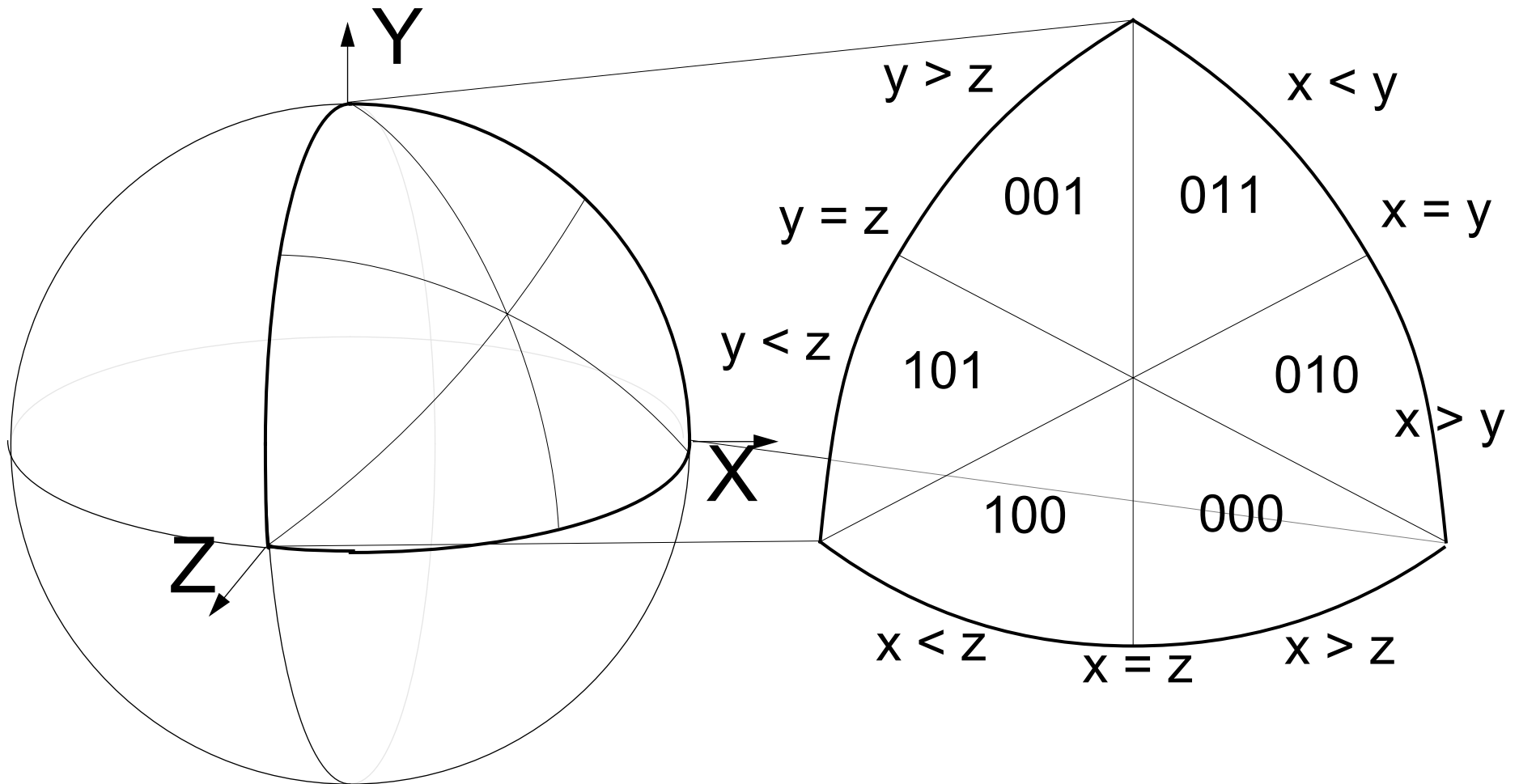
Overview of Compression Process:

- **Normalize positions to unit cube**
- **Compute general triangle mesh**
- **Quantize/encode vertex values**
- **Compute deltas**
- **Create Huffman table**
- **Generate binary output**

Overview of Decompression Process:

- **Shuffle fields/determine opcode**
- **Extract tag**
- **Extract vertex delta value**
- **Un-delta value**
- **Use mesh info to re-form triangle
pass triangles out**

Normal Compression Technique



Short Huffman Tag Technique

- **Maximum 6-bit length Huffman tag**
- **Simple hardware implementation**
- **Modified Huffman encoder needed**

Variable Quantization

- **Can vary quantization dynamically**
- **Separately specified for positions, normals, colors**
- **Positions: 1–16 bits per component**
- **Colors: 1–16 bits per component**
- **Normals: 2–18 bits per normal**

Examples in OpenGL™

- **Demo examples of geometry decompression in OpenGL**

Examples in Java 3D™

- **Demo examples of geometry decompression in Java 3D**

Example: HoloFlicks

- **Compressed Geometry movies**
- **Each frame is an independent Compressed Geometry object**
- **Streamed in real-time from disk**
- **Viewpoint dynamically determined at run-time; supports headtracked stereo (virtual reality display)**

HoloFlick Generation

- **Used RenderMan to generate frames of micropolygons that were then compressed**
- **Surfaces created with animated morphed patches**
- **Used texture maps, bump maps, displacement maps, shadow maps**

HoloFlick Generation (Notes)

All work was done with Pixar's standard RenderMan product (3.8). We had to make a few slight changes to the shaders used. The Opacity had to be very slightly transparent everywhere: Opacity [0.99999 0.99999 0.99999]. The specular component of shading had to be disabled, but could be applied at run time. Reflection maps could not be used. ShadingRate was used to control the final micropolygon count. The "print" command was used to export the micropolygons to the geometry compressor.

The compressor had to stitch the micropolygons back together for compression (re-computing connectivity). The connectivity analysis took almost as long as RenderMan took to render a frame! Each frame was compressed into a separate file, and loaded at run-time as a 3D movie loop using Java 3D. Alternately, the data could have been streamed off a disk array, at a rate of 10 million+ micropolygons per second. A RAID array of several consumer DVD disks can easily contain 20 minutes of animation, with each frame as separate data.

The viewpoint is dynamically computed at run-time by a headtracker, allowing for a Hologram-like look-around display.

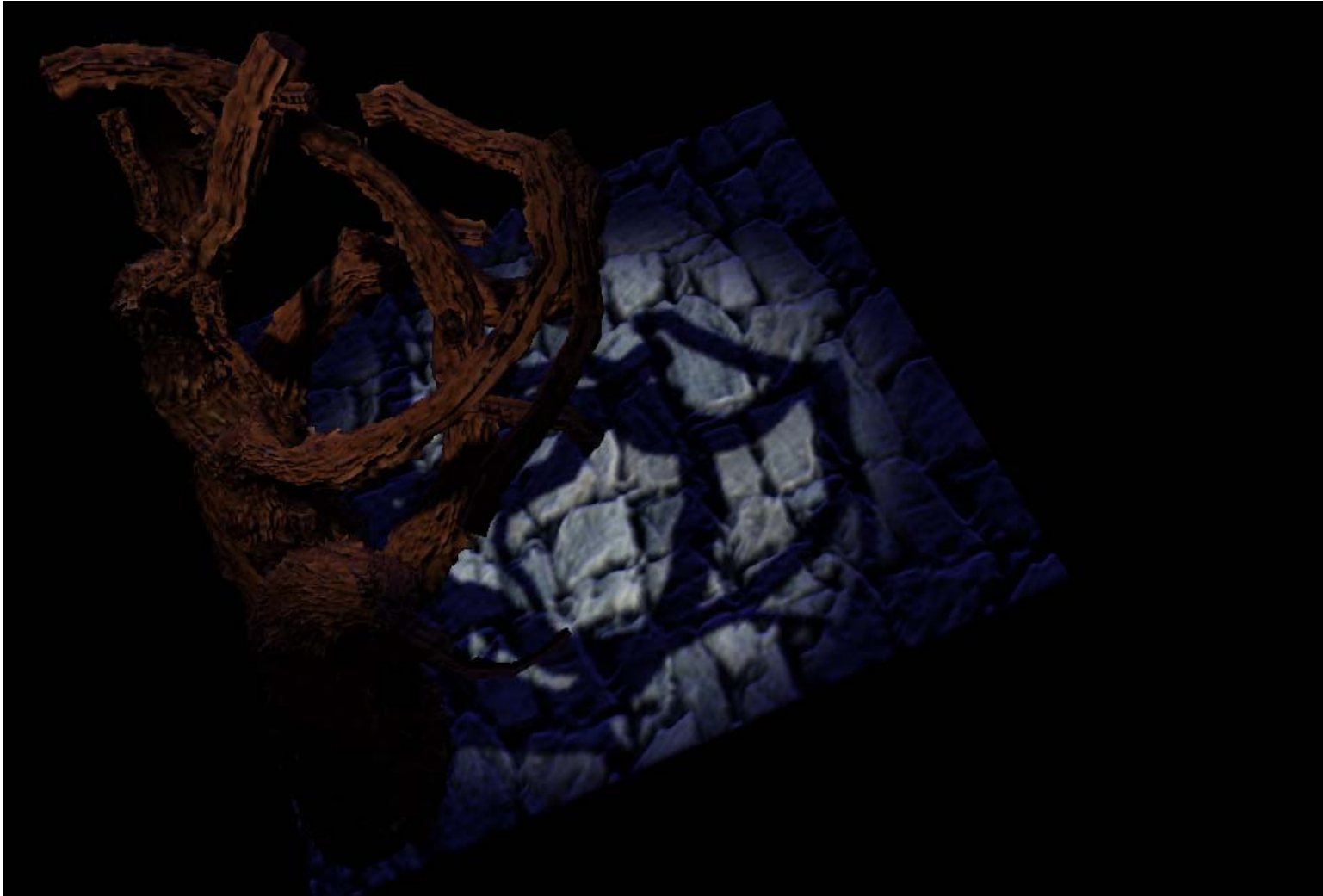
Besides RenderMan, 3D scanners or cameras can be used to generate data to be converted into compressed micropolygons.

Original RenderMan Frame



HoloFlick Real-time Render Frame

This image is rendered compressed geometry. The micropolygons were generated a side effect of RenderMan rendering the previous slide. The viewpoint was changed dynamically at run-time.



Hardware Constraints

- **Any decompression state must be limited in size and local.**
- **The compressed data must be incrementally decompressable in a single pass.**
- **The compressed data format must be decompressable by hardware in 100 ns or less.**

Hardware Constraints

- **The silicon area taken by the decompression circuit must fit on one chip; ideally on a small portion of a chip.**
- **Hardware geometry compression formats must not only compress well behaved geometry; compression times can be quite asymmetric, but not exponential.**

Hardware Constraints (Notes)

This section will list several restrictions that apply specifically to geometry compression formats that support real-time hardware implementations of decompression. Most of these restrictions do not apply to geometry compression formats intended for software only decompression. Indeed, without these restrictions, pure software compressed geometry formats can achieve higher compression densities and support other advanced features.

Any decompression state must be limited in size and local.

Many software decompression techniques rely on building global arrays of all vertex information, and require random access to this state. If a hardware decompressor is trying to save memory size and bandwidth, placing all this decompressed vertex information in main memory defeats the purpose. The idea is that all state should fit on chip, to keep the cost down, and to eliminate the need for expensive off chip access.

Another implication of these constraints is the elimination of the possibility of re-deriving normal vector information by local surface differentiation: the neighboring vertices will not always be resident on-chip at the same time.

Hardware Constraints (Notes)

The compressed data must be single pass and incremental.

In some cases where the compressed geometry data is being directly delivered from a network, there may not even be enough storage to store the compressed data. Rather, the geometry is to be rendered on the fly and discarded. This requires that geometry can be decompressed and rendered on the fly and on the first pass. An example of an application needing this property would be a 3D movie, where a DVD disk or network would supply a continuous stream of pre-computed geometry to be dynamically viewed from a dynamically controlled viewpoint. (MPEG4 is looking at applications like this.)

The compressed data format must be decompressable by hardware in 100 ns or less.

No matter what other advantages, a compressed geometry format that cannot be decompressed even by dedicated hardware as fast as modern machines can render 3D geometry will be of limited utility when used with such machines. The keystones of most geometry compression techniques are variable-length huffman-like tagged data, and clever indexing techniques to reference old vertices in forming geometry. The problem is that decoding of variable

Hardware Constraints (Notes)

length information in general is not very pipeline-able. You can't start processing the next variable length tag until the previous one has been extracted and examined. (This is the same reason why nearly all modern computer instruction sets use fixed length instructions.) Even with modern high speed ICs, each piece of variable bit-length data in a stream will take several clock cycles to process. With two to three such items per vertex, even at clock rates over a hundred megahertz and simple geometry compression formats, it is hard to decompress at modern several million triangle per second render rates. Adding additional variable length fields to each triangle can make a decompression technique too slow for real-time use. Complex indirection rules to find old vertices to be combined into new triangles can also fail this hardware time limit.

The silicon area taken by the decompression circuit must fit on one chip; ideally on a small portion of a chip.

Much greater bandwidth is available on chip than off; a single chip geometry decompression circuit is much easier to implement than a circuit that requires several chips to be interconnected. With most modern low-end 3D graphics subsystems implemented on a single chip, to be cost competitive, ideally a geometry decompression circuit must be able to fit on the same

Hardware Constraints (Notes)

single graphics chip. This also severely constrains the sorts of geometry decompression algorithms amenable to hardware implementation. Large amounts of scratch storage, or even large constant tables are too expensive. As another example, even if all local information was available for differently re-derivation of surface normals, the cost of so many cross products and square roots every few tens of nanoseconds would also prohibit this technique.

Hardware geometry compression formats must not only compress well behaved geometry; compression times can be quite asymmetric, but not exponential.

The bane of many otherwise elegant geometry compression techniques is that they may only achieve good compression ratios when the geometry is well behaved. While regular mesh geometry data is commonly produced by many good tessellators and re-tessellators, highly irregular data is commonly produced by hand digitizers and intelligent re-tessellators. With some geometry compression formats, irregular data may not only compress badly, it may take an extremely long time to compress, or the algorithm may even fail. A hardware geometry decompression circuit will be used by many different applications and geometry types; it should not limit the types of (triangle) geometry that can be compressed to its format.

Hardware Constraints (Notes)

For video compression formats, a rule of thumb is that the compression time should not be more than 100 times slower than the decompression time (real-time). This makes sense given typical video production real-time lengths of several minutes to half an hour (thus several hours to several days of compression times). For 3D geometry, for many applications, the compression times can be slower than this, because the current equivalent of production length is much shorter. A 3D game may only have a few days worth of geometry to compress, even with a 1000X slower compression time to real-time decompression rate. However, geometry compression formats that have much slower compression times than this may not be practical for production use for real-time decompression hardware. (Of course, an extra-high compression rate algorithm that can be optionally run when the time/space trade-off is acceptable is OK, so longer as faster, if less optimal compressors exist for that format.)

While not needed in the near-term, it is not inconceivable that in the future (at rates still slower than real-time) dedicated hardware for geometry compression may be necessary. Eventually, forms of 3D TV may actually employ real-time geometry compression to encode live events for transmission.

Progressive Transmission

- **Hardware requirements may limit progressive transmission techniques.**
- **Some progressive transmission techniques can require global state.**
- **Progressive transmission has speed constraints.**

Progressive Transmission (Notes)

Progressive transmission can require global state.

One scheme for progressive transmission of compressed geometry can be just to send independently compressed objects at successively higher levels of detail. Some redundant information is sent, but this may be less than 20%. Because of the requirement for limited state, most geometry compression formats designed for hardware implementability will not be able to take advantage of any redundancy in successive level of detail transmissions.

Progressive transmission speed constraints.

There is another limitation with progressive transmission techniques. This is the requirement that the decompression time be faster than the transmission time. Thus with low bandwidth lines, a progressive transmission technique may be able to put up a low resolution object in software faster than a lower compression rate hardware format may; but when the bandwidth of transmission exceeds the software decompression time, a less compressed hardware format could put up an image before the software technique could. For corporate environments, networks speeds are high enough that progressive transmission formats not supportable by hardware may not make any sense. Software only progressive techniques are more likely to be of interest in low bandwidth consumer applications.

Progressive Transmission (Notes)

The table below relates network speeds with the rate that a geometry decompressor would have to run to not add additional latency (assuming 2 bytes per triangle):

Bit Rate	Network	Triangle rate
56K	Modem	3.5K
120K	ISDN	7.5K
1.5M	Cable Modem	94K
1.5M	ASDL	94K
10M	Ethernet	625K
100M	Ethernet	6.25M
1G	Net	62.5M

Note that even the 100K triangle per second decompression balance rate for cable modem rates is at the upper end of reported software decompression speeds. While CPUs will continue to get faster in the future, network technology (like graphics) is expected to get faster.

It is also true that in many entertainment applications there may be a desire to ensure that all levels of detail of objects are pre-loaded before any of them are displayed to guarantee a

Progressive Transmission (Notes)

smooth view. In this case only avoiding loading redundant information is of interest, not fastest time to display the lowest resolution image.

In some circumstances it may be important for the server to take the bandwidth of the connection into account, as well as the maximum speed of the remote 3D rendering engine. Then the server may make a policy decision to not send the very highest most level(s) of detail. In many current internet technologies this is accomplished by a crude technique of having two alternate web pages, and sometimes the user herself must make the choice between them. For future 3D web applications, more automatic policies may be possible/desirable.

Analyzing Geometry Compression Formats

- **Entropy of encoding “vertex connectivity” information**
- **Entropy of encoding vertex position information**
- **Entropy of encoding other vertex information (normal, color, etc.)**
- **Overheads (tables, headers, etc.)**

Results

Object	# tris	bits/tri	vert/tri	pos quant	ave quant	bits/xyz	normquant	bits/ norm
Apple	10,556	20.7	0.63	11	9.6	13.3	12	4.2
Dodge Viper	55,864	23.3	0.65	12	9.4	14.3	12	6.1
Scanned Bunny	69,603	21.8	0.62	12	10.5	13.5	12	5.8
Hindu Statue	74,667	22.5	0.69	12	9.9	12.6	13	5.6
Turtle	103,143	21.6	0.62	12	10.8	13.5	13	5.6
Acura Integra	109,826	25.4	0.67	13	10.0	14.9	13	6.3
Marching Cubes Skull	204,529	23.4	0.67	13	10.4	13.4	13	6.4
Schooner Ship	206,311	27.5	0.70	13	11.8	17.5	13	7.3
Scanned Buddha	293,233	21.8	0.63	13	11.3	14.1	13	5.9

Object	original size (ascii)	original size (binary strips)	compressed bytes	time (min)	ratio over binary	ratio over ascii
Apple	996,960	420,784	27,395	0.1	15.4X	36.4X
Dodge Viper	5,649,794	1,934,996	163,100	0.8	11.9X	34.6X
Scanned Bunny	7,062,795	3,070,340	189,725	2.9	16.2X	37.2X
Hindu Statue	7,674,068	3,067,736	208,941	1.8	14.7X	36.7X
Turtle	10,459,787	3,847,592	278,655	1.8	13.8X	37.5X
Acura Integra	11,112,993	4,087,496	339,177	1.7	12.1X	32.8X
Marching Cubes Skull	20,789,349	8,818,096	597,920	11.4	14.7X	34.8X
Schooner Ship	21,064,503	8,624,952	771,585	5.1	11.2X	27.3X
Scanned Buddha	29,937,845	12,335,301	788,655	12.4	15.4X	37.4X

Compression results for hardware format on several representative sample objects. (From [Chow97].)

Compression Example: Schooner



Schooner, original.



Compressed to 3.4 bytes per triangle.

Compression Example: CT Skull



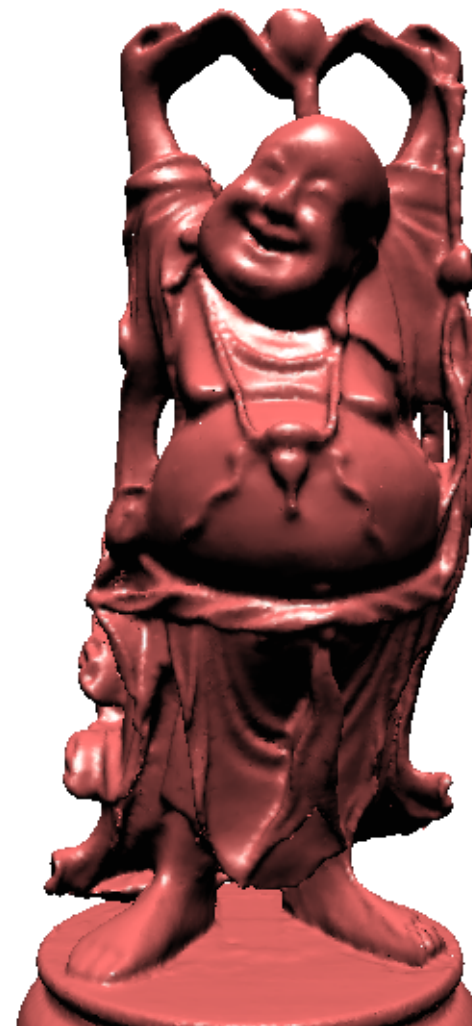
Original Marching Cubes skull.

Compressed to 2.9 bytes per triangle.

Compression Example: Scanned Buddha



Original Happy Buddha.



Compressed to 2.7 bytes per triangle.

When to Compress for Dynamical Apps?

- **Compression in shadow of tessellator**
- **Compression as integral part of tessellator**
- **Compression as lazy memory scrubbing process**

Compressed Geometry Tools & Libraries

- **ASCII assembler, disassembler**
- **Format verifier**
- **Low-level compression utilities**
- **Medium-level compression utilities**
- **High-level compression utilities**

Notes on Tools & Libraries

More details on the compressed geometry ASCII assembly syntax can be found in the Sun compressed geometry specification (reproduced elsewhere in these course notes). The specification also contains detailed rules governing the verification of compressed geometry objects for compliance. These are the rules implemented by the verifier software.

Many of the tools and libraries mentioned here will likely be publicly available in source form on the web by the time these course notes are available; check the Sun Java 3D home page for updates.

ASCII Assembly Format

- **Disassembler to convert Compressed Geometry files to human readable form**
- **Assembler to convert human editable Compressed Geometry test files to binary form**
- **Useful for making test vectors & debugging, NOT for application use**

Compressed Geometry Verifier

- **Expanded software decompressor**
- **Tracks state validity**
- **Enforces special rules**
- **Available as stand-alone tool**
- **Available as library call**

Low-level Compression Utilities

- **Handles header forwarding**
- **Handles bit-field level operations**
- **Outputs individual instructions**
- **Optional debugging trace prints**

Medium-level Compression Utilities

- **Encodes normals**
- **Takes histograms of input data**
- **Generates Huffman tables**
- **File header/tail generation support**
- **Useful when application maintains meshification connectivity info**

High-level Compression Utilities

- **Computes connectivity information**
- **Optionally performs meshing**
- **Generates complete compressed objects**
- **Easy to bolt onto existing applications and format converters**

Hardware Decompression Future

- **Standard “test” objects & metrics**
- **Standards**
- **Advanced techniques**
- **More work on HoloFlicks**

Standard Test Objects: 3D “Lenna”

- **Standard objects/metrics for measuring compression performance/quality**
- **Should include irregular objects, both to prove that compression algorithm can handle them, and to show compression density on re-tessellated objects**

Standards

- **Due to long lead times and investments, standardization is even more important for hardware than software**
- **Interpretability without software transcoding (or decompression / recompression) is important**

Advanced Techniques for Hardware

- **Higher compression densities for special cases**
- **Integration with dynamic level of detail generation algorithms**
- **More sophisticated software environments that take advantage of compression**