Lossless Geometry Compression for Steady-State and Time-Varying Irregular Grids

Dan Chen
Yi-Jen Chiang
Nasir Memon
Xiaolin Wu
Polytechnic University, New York, USA

(Eurographics/IEEE EuroVis 2006)

Graphics Compression for Irregular-Grid Volume Data

- Graphics compression is an emerging need for storing, transmitting, and visualizing large volume datasets
- **Irregular Grids** (represented as **tetrahedral meshes**):
  - The most general class of volume data, with many applications
  - Two components of information:

    geometry -- 3D coordinates & scalar values at mesh vertices

    connectivity – edges, triangles & tetrahedral cells connecting vertices

Previous Work

- Many results in connectivity compression… (see paper)
- Best connectivity compression results: 2.04 -- 2.31 bits/cell on an average
  (# cells = 4.5 * (# vertices) → ~ 2 * 4.5 = ~ 9 bits/vertex)
  - e.g. [Gumhold et al 99], [Yang et al 00]
- Geometry compression results are not equally impressive
  - Typical results: ~ 30 bits/vertex (e.g. [Gumhold et at 99])
    (1. not including scalar values 2. quantized then compressed (lossy))

……

Geometry compression is by far the dominating bottleneck!!

Even worse for time-varying data & lossless compression

Remark: Why Lossless Compression?

[Note: Almost all geometry coders need quantization (lossy)]

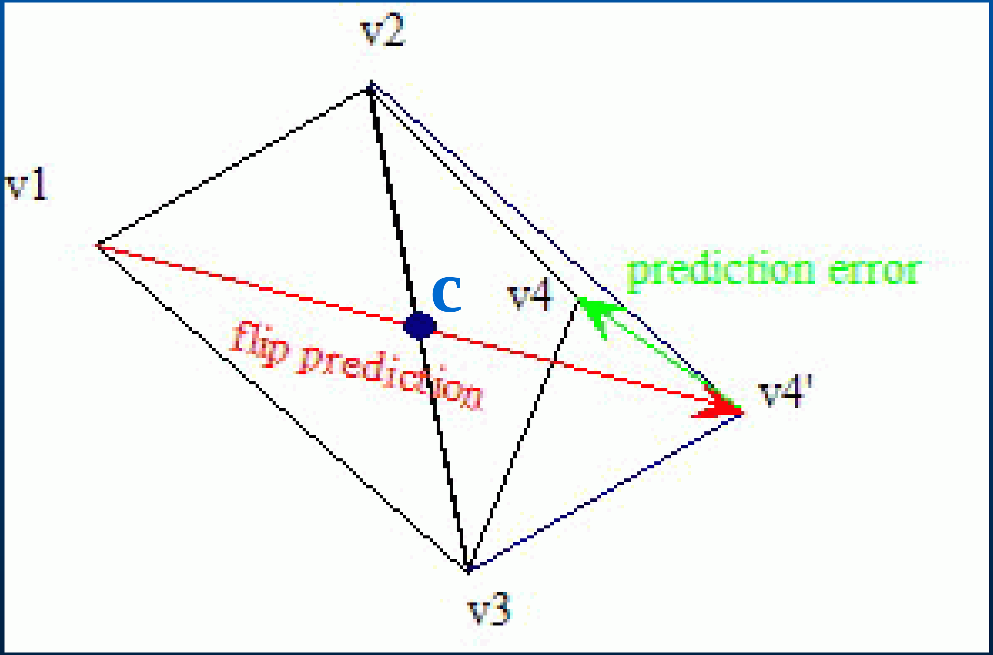
- In many scientific applications accuracy is of vital importance (and quantization is not desirable)
  - Cf. for graphics models, lossy compression might be acceptable to ‘‘fool the eyes’’
- Usually scientists do not like their data to be changed in a process outside their control, and hence often avoid using any lossy compression [Isenburg et al 04]

→ We focus on lossless geometry compression, for both steady-state & time-varying tetrahedral meshes

Previous Work: Geometry Compression

- **Flipping for triangle meshes [Touma-Gotsman 98]**
  - **Dominant, widely considered state of the art**
  - **Traverse triangles by connectivity coder; predict new vertex position of new triangle by flipping using parallelogram rule**
  - **Many other extensions (see paper); [Isenburg et al 04] -- lossless flipping (the only method with no quantization)**
- **Flipping for volume data**
  - **[Isenburg-Alliez 02]: hexahedral meshes**
  - **State of the art for tetrahedral meshes: flipping combined with best connectivity coder [GGS99, YMC00]**

*** No lossless flipping result**



Our New Algorithm

- Lossless geometry compression method, for both steady-state & time-varying tetrahedral volume data
- Truly lossless --- quantization is not needed (but it works equally effective if quantization is performed)
- Can be easily integrated with the best connectivity coder [GGS99, YMC00]
- Novel direction: geometry coder is independent of connectivity coder---they may re-order vertices differently
  - Causes an additional overhead of recording vertex permutation when integrating the two coders
  - Significantly improves the geometry-compression cost even after paying the permutation overhead
  - New feature: geometry coder does not need connectivity information --- suitable for point cloud compression

Key Ideas of Our Algorithm

- Not trying to design a fancy predictor
  - Use differential coding (simplest predictor)
  - Re-order vertices to optimize differential coding --- formulate optimal vertex re-ordering as a traveling salesman problem (TSP). * Solve it with heuristics
- Using kd-tree-like partitioning/clustering to speed up TSP computation
- Final encoding: separate exponent and mantissa
  - Encode sign bit & exponent (signed exponent) by gzip
  - Encode mantissa differences by entropy code
    Problem: many distinct symbols (large alphabet size)
    → big Huffman table; bad probability estimation

    Sol: Use two-layer modified Huffman code [our DCC03 paper] or two-layer modified arithmetic code

Algorithm Steps

* Entry for each vertex v : $(x, y, z, f_1, f_2, \ldots, f_t)$

1. For each vertex v, take mantissa differences of scalar values along time steps: $(x, y, z, f_1, (f_2 - f_1), \ldots, (f_t - f_{t-1}))$ (only for time-varying data)
2. Partition vertices into clusters (using mantissa of x, y, z)
3. In each cluster, re-order vertices by formulating and solving a TSP problem (using mantissa of x, y, z and f_i’s)
4. Take component-wise mantissa differences: mantissa of each component of v_i is replaced by its difference with the corresponding mantissa of v_{i-1}
5. Entropy code the mantissa differences (two-layer modified Huffman or arithmetic code)
6. Compress the signed exponents by gzip (order: all x-values, all y, all z, all f_1, all f_2, …, all f_t)

(* If quantized: quantized integer as mantissa; no exponent)

Step 3: Vertex Re-Ordering

- **Goal: re-order vertices to optimize differential code of mantissa differences**
- **Form a complete (undirected) weighted graph G:**
  *** nodes of G: mesh vertices * edges: all pairs of vertices**
  *** cost of edge (v_i, v_j) = $\lg |x_i - x_j| + \lg |y_i - y_j| + \lg |f_{i1} - f_{j1}| + \ldots + \lg |f_{it} - f_{jt}|$ (difference: component mantissa diff.)**
- **Optimal vertex re-ordering: TSP on G, i.e., a Hamiltonian path that visits each node of G exactly once while minimizing the total path cost**
- **Heuristic algorithms (TSP is NP-complete):**
  1. simulated annealing (SA)
  2. minimum-spanning tree (MST) based approximation: depth-first-search traversal on MST

Step 2: Partition Vertices into Clusters

- The MST heuristic to find TSP takes $O(n^2)$ time since G is a complete graph (n: # vertices)
- Even just computing the edge costs of G takes $O(n^2)$ time

Sol: Partition vertices into K clusters of same size, solve TSP inside each cluster
(TSP time: $O(K\ (n/K)^2) = O(n^2/K)$, speed-up factor: K)

Partitioning algorithm: (let $L = K^{1/3}$)

- Sort all vertices by mantissa of x-values, split into L groups of same size
- For each group, sort by mantissa of y, split into L groups
- Repeat the process by mantissa of z (final groups = clusters)
- O(n log n) time (due to sorting)

Step 5: Entropy Coding Mantissa Differences

- Too many distinct symbols (large alphabet) in mantissa differences to encode directly
- Two-layer modified Huffman code [our DCC 03 paper]
  - Partition range of integer values (alphabet) into intervals
  - Encode the intervals by a Huffman code (first-layer code)
  - Encode the values within each interval by a fixed-length code (second-layer code)
  - Try to optimize alphabet partitioning so that total code length is minimized
    1. dynamic programming : $O(N^3)$ time (N: # distinct symbols)
    2. greedy method: $O(N \log N)$ time; still compresses well
- Here, we extend to two-layer modified arithmetic code with greedy method. Use two-layer arithmetic/Huffman code

Step 5: Entropy Coding (cont.)

- We need to encode multiple clusters & multiple vertex components (i.e., x, y, z, f_i's)
- Two extreme options:
  1. for each cluster, one AC/Huffman code for mantissa of each vertex component --- too many probability-/Huffman-tables (expensive)
  2. a single AC/Huffman code for all components in all clusters --- entropy code less efficient
- Our approach
  - Idea: coordinates are co-related, so are scalar values
  - Steady-state: Combined Greedy (CGreedy) --- one code for all coordinates in all clusters, one code for all scalar values
  - Time-varying: CGreedy*i* --- one code for all coordinates; for scalar values, one code for every *i* time steps in all clusters

Summary: Algorithm Steps

1. For each vertex, take **mantissa differences** of scalar values along **time steps** (time-varying data only)
2. Partition vertices into **clusters**
3. In each cluster, re-order vertices by **TSP**
4. Take component-wise **mantissa differences** between adjacent vertices
5. Entropy code the mantissa differences by **two-layer modified AC/Huffman code**
6. Compress the signed exponents by gzip

(* If quantized: quantized integer as mantissa; no exponent)

* Encoding only performed **once**; decoding takes **linear time**

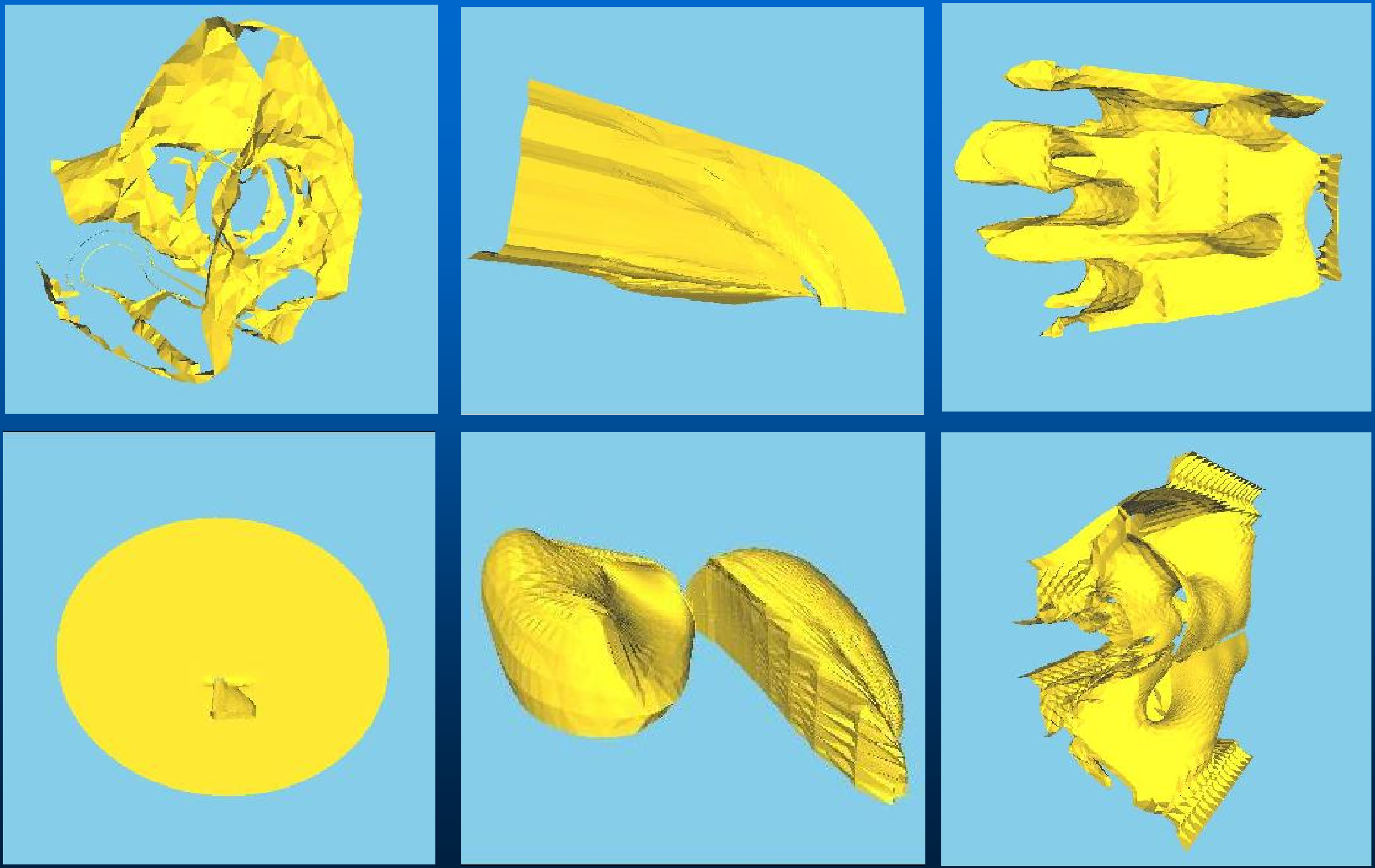
Last Technical Part: Integration with Connectivity Coder

- Easy to do (see paper for details)
- Geometry & connectivity coders re-order vertices differently: permutation sequence
- Encoding the permutation sequence
  - Differential coding on the permutation sequence
  - Issue: many distinct symbols
    → Use two-layer modified Huffman code

Experiments

- 7 well-known tetrahedral-mesh datasets
  - 5 datasets are steady-state
  - 2 datasets are time-varying (10 & 20 time steps) of the same mesh
  - size: small --- moderately large
    (# cells: 12,936—1,005,675, # vertices: 20,108—211,680)
- Evaluate effectiveness of clustering & vertex re-ordering
- Compare our final compression ratios with the following:
  - For lossless compression (no quantization):
    (1) Gzip, (2) arithmetic coding (AC), (3) lossless flipping
  - For lossy compression (quantization first):
    lossy flipping (encoding prediction errors: gzip & AC)

Representative Isosurfaces of Datasets



Results: Statistics Summary

- Clustering & TSP vertex re-ordering: entropy-speed trade-off
  - 100--200 vertices/cluster (i.e., 512 clusters) the best balance
  - TSP-MST: ~ 200 times faster than TSP-SA, with comparable entropy
- Steady-state (AC-CGreedy)
  - Lossless
    - » Always much better than AC/Gzip* (* see paper for more details)
    - » Lossless flipping: bad predictor (entropy > original entropy!!)
  - Lossy (32-bit quantization)
    - » Permutation sequence can be efficiently encoded
    - » Always much better than flipping; gain up to 62.1 b/v (67.2%)
- Time varying (AC-CGreedy*i*)
  - Lossless: always much better than AC/Gzip; gain up to 66 b/v (32.6%)
  - Lossy (24-bit quantization): always much better than flipping; gain up to 61.4 b/v (23.6%)

Conclusions

- Novel lossless geometry compression method via vertex re-ordering by TSP, for both steady-state and time-varying data
- Novel direction: geometry oriented; connectivity information not needed
- Easily integrated with connectivity coder
- Huge improvements in both lossless & lossy settings

Extension

Point cloud compression
[Chen-Chiang-Memon]: PG 05 short paper

Open Question

How can we use connectivity information better than flipping (especially in lossless setting)?
(cf. lossy: optimized flipping for triangle meshes [our DCC05])

Acknowledgments

- C. Silver, H.-W. Shen, Lawrence Livermore National Lab, NASA, Vtk for the test datasets
- National Science Foundation (NSF) (CAREER CCF-0093373, CCF-0118915, ITR CCF-0081964, CCF-0541255)