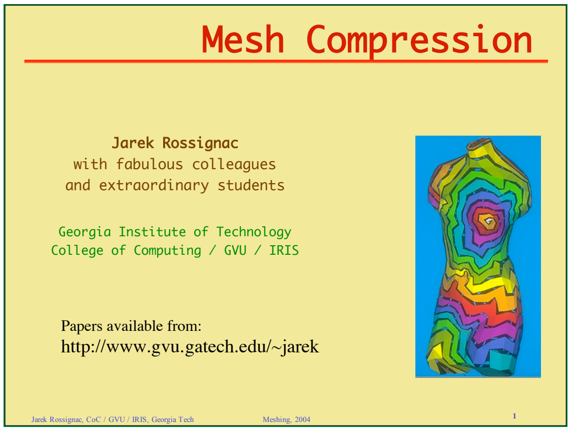
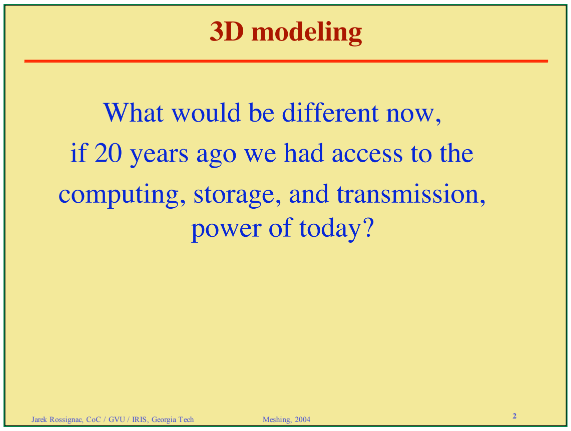


Mesh Compression

Jarek Rossignac
with fabulous colleagues
and extraordinary students

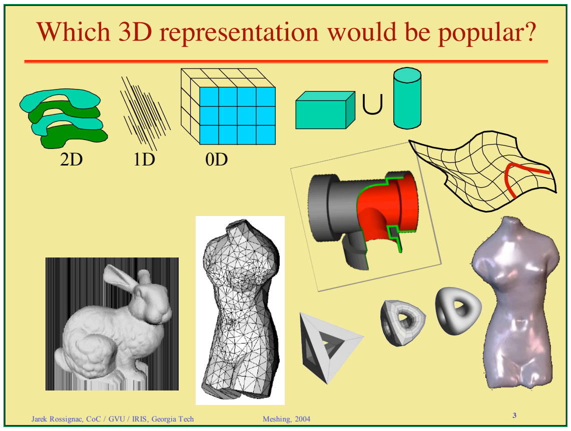
Georgia Institute of Technology
College of Computing / GVU / IRIS

Papers available from:
http://www.gvu.gatech.edu/~jarek

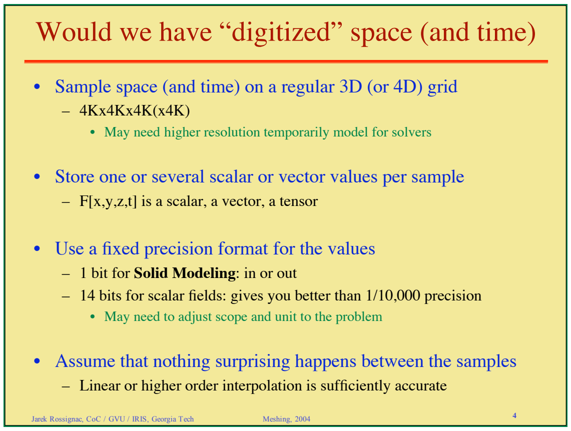


3D modeling

What would be different now,
if 20 years ago we had access to the
computing, storage, and transmission,
power of today?

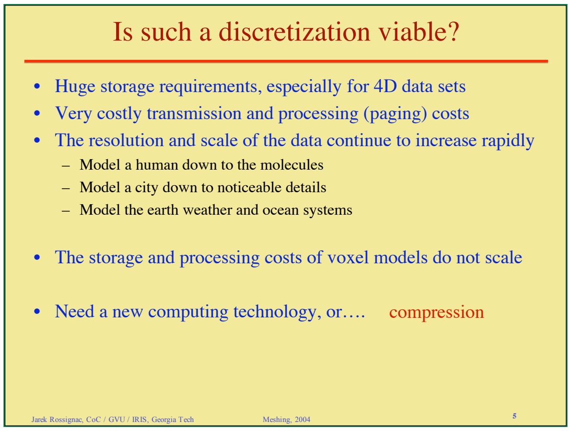


Which 3D representation would be popular?



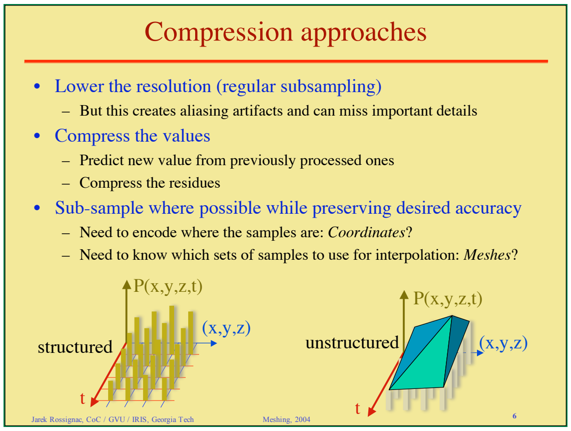
Would we have “digitized” space (and time)

- Sample space (and time) on a regular 3D (or 4D) grid
  - 4Kx4Kx4K(x4K)
    - May need higher resolution temporarily model for solvers
- Store one or several scalar or vector values per sample
  - F[x,y,z,t] is a scalar, a vector, a tensor
- Use a fixed precision format for the values
  - 1 bit for **Solid Modeling**: in or out
  - 14 bits for scalar fields: gives you better than 1/10,000 precision
    - May need to adjust scope and unit to the problem
- Assume that nothing surprising happens between the samples
  - Linear or higher order interpolation is sufficiently accurate



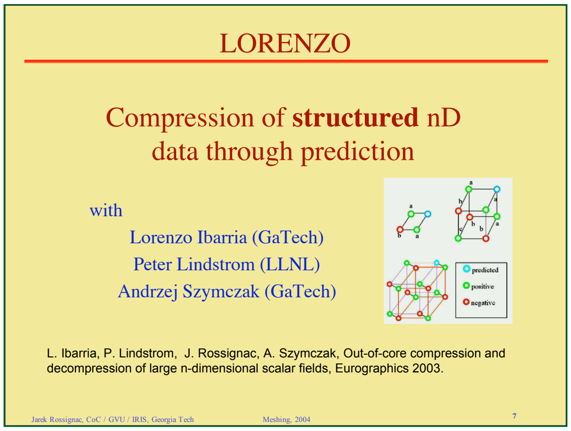
Is such a discretization viable?

- Huge storage requirements, especially for 4D data sets
- Very costly transmission and processing (paging) costs
- The resolution and scale of the data continue to increase rapidly
  - Model a human down to the molecules
  - Model a city down to noticeable details
  - Model the earth weather and ocean systems
- The storage and processing costs of voxel models do not scale
- Need a new computing technology, or…. compression



Compression approaches

- Lower the resolution (regular subsampling)
  - But this creates aliasing artifacts and can miss important details
- Compress the values
  - Predict new value from previously processed ones
  - Compress the residues
- Sub-sample where possible while preserving desired accuracy
  - Need to encode where the samples are: *Coordinates*?
  - Need to know which sets of samples to use for interpolation: *Meshes*?



LORENZO

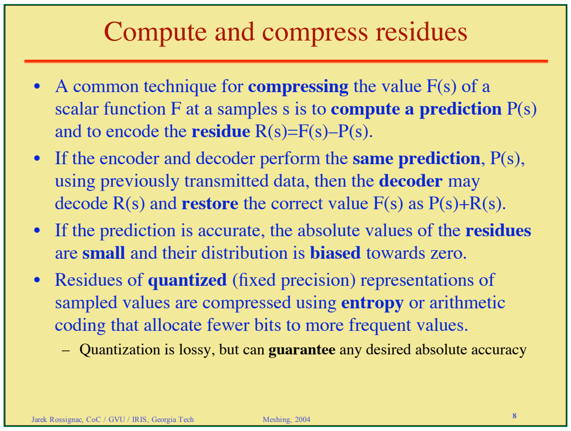
Compression of **structured** nD data through prediction

with

Lorenzo Ibarria (GaTech)
Peter Lindstrom (LLNL)
Andrzej Szymczak (GaTech)

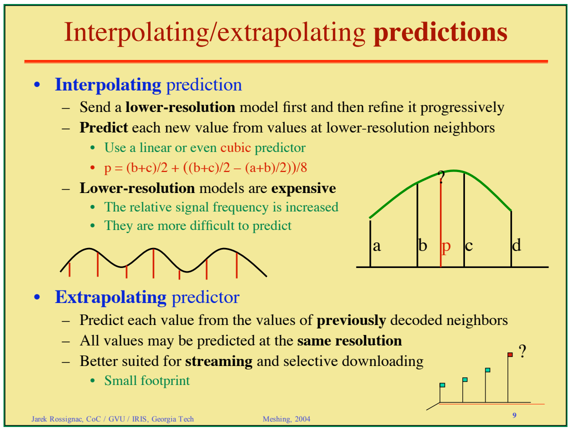
a
b
a
a
b
a
b
b
c
a
predicted
positive
negative

L. Ibarria, P. Lindstrom, J. Rossignac, A. Szymczak, Out-of-core compression and decompression of large n-dimensional scalar fields, Eurographics 2003.



Compute and compress residues

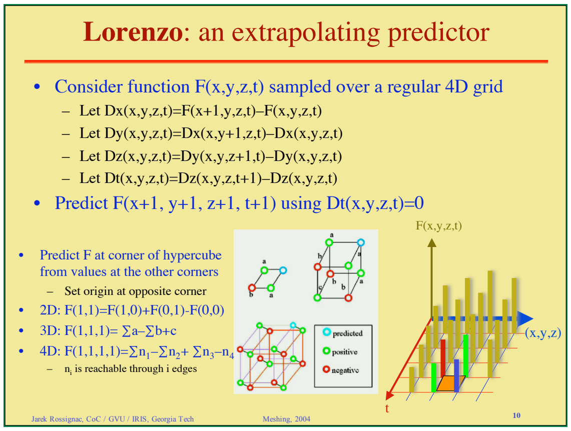
- A common technique for **compressing** the value F(s) of a scalar function F at a samples s is to **compute a prediction** P(s) and to encode the **residue** R(s)=F(s)–P(s).
- If the encoder and decoder perform the **same prediction**, P(s), using previously transmitted data, then the **decoder** may decode R(s) and **restore** the correct value F(s) as P(s)+R(s).
- If the prediction is accurate, the absolute values of the **residues** are **small** and their distribution is **biased** towards zero.
- Residues of **quantized** (fixed precision) representations of sampled values are compressed using **entropy** or arithmetic coding that allocate fewer bits to more frequent values.
  - Quantization is lossy, but can **guarantee** any desired absolute accuracy



Interpolating/extrapolating **predictions**

- **Interpolating** prediction
  - Send a **lower-resolution** model first and then refine it progressively
  - **Predict** each new value from values at lower-resolution neighbors
    - Use a linear or even cubic predictor
    - p = (b+c)/2 + ((b+c)/2 – (a+b)/2))/8
  - **Lower-resolution** models are **expensive**
    - The relative signal frequency is increased
    - They are more difficult to predict

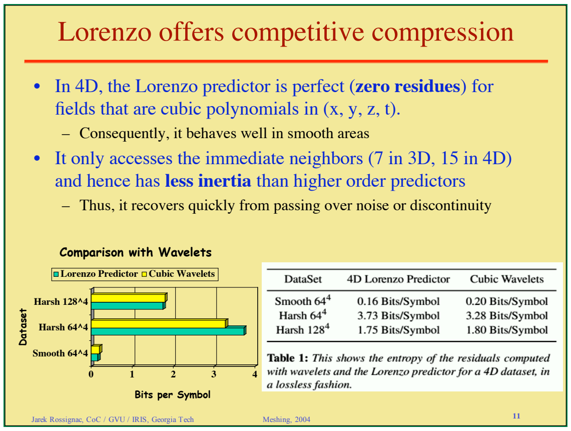
- **Extrapolating** predictor
  - Predict each value from the values of **previously** decoded neighbors
  - All values may be predicted at the **same resolution**
  - Better suited for **streaming** and selective downloading
    - Small footprint



Lorenzo: an extrapolating predictor

- Consider function F(x,y,z,t) sampled over a regular 4D grid
  - Let Dx(x,y,z,t)=F(x+1,y,z,t)–F(x,y,z,t)
  - Let Dy(x,y,z,t)=Dx(x,y+1,z,t)–Dx(x,y,z,t)
  - Let Dz(x,y,z,t)=Dy(x,y,z+1,t)–Dy(x,y,z,t)
  - Let Dt(x,y,z,t)=Dz(x,y,z,t+1)–Dz(x,y,z,t)
- Predict F(x+1, y+1, z+1, t+1) using Dt(x,y,z,t)=0

- Predict F at corner of hypercube from values at the other corners
  - Set origin at opposite corner
- 2D: F(1,1)=F(1,0)+F(0,1)-F(0,0)
- 3D: F(1,1,1)= ∑a–∑b+c
- 4D: F(1,1,1,1)=$\sum n_1 - \sum n_2 + \sum n_3 - n_4$
  - n_i is reachable through i edges

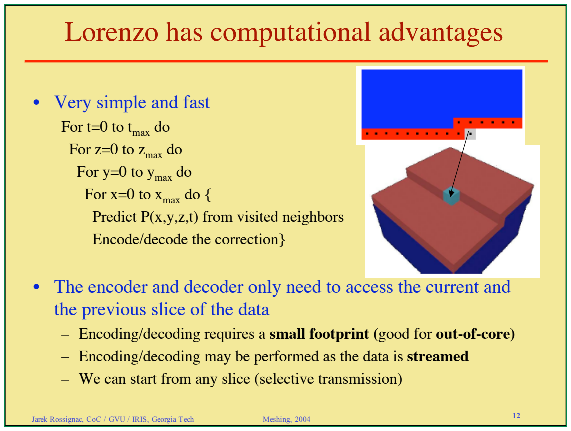


Lorenzo offers competitive compression

- In 4D, the Lorenzo predictor is perfect (**zero residues**) for fields that are cubic polynomials in (x, y, z, t).
  - Consequently, it behaves well in smooth areas
- It only accesses the immediate neighbors (7 in 3D, 15 in 4D) and hence has **less inertia** than higher order predictors
  - Thus, it recovers quickly from passing over noise or discontinuity

DataSet	4D Lorenzo Predictor	Cubic Wavelets
Smooth 64^4	0.16 Bits/Symbol	0.20 Bits/Symbol
Harsh 64^4	3.73 Bits/Symbol	3.28 Bits/Symbol
Harsh 128^4	1.75 Bits/Symbol	1.80 Bits/Symbol

Table 1: *This shows the entropy of the residuals computed with wavelets and the Lorenzo predictor for a 4D dataset, in a lossless fashion.*

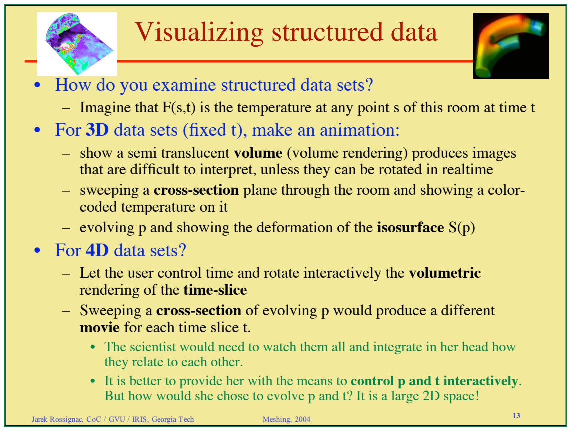


Lorenzo has computational advantages

- Very simple and fast

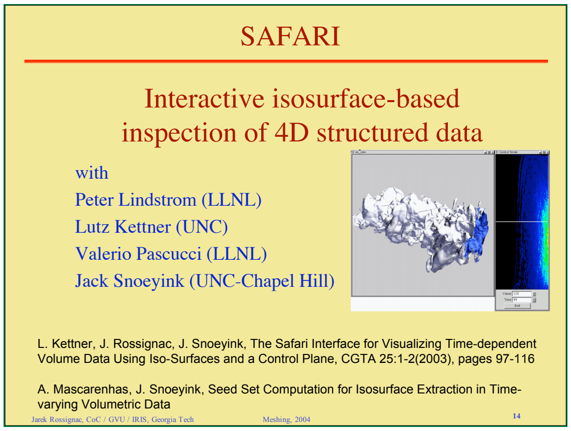
```
For t=0 to $t_{max}$ do
  For z=0 to $z_{max}$ do
    For y=0 to $y_{max}$ do
      For x=0 to $x_{max}$ do {
        Predict P(x,y,z,t) from visited neighbors
        Encode/decode the correction}
```

- The encoder and decoder only need to access the current and the previous slice of the data
  - Encoding/decoding requires a **small footprint** (good for **out-of-core**)
  - Encoding/decoding may be performed as the data is **streamed**
  - We can start from any slice (selective transmission)



Visualizing structured data

- How do you examine structured data sets?
  - Imagine that F(s,t) is the temperature at any point s of this room at time t
- For **3D** data sets (fixed t), make an animation:
  - show a semi translucent **volume** (volume rendering) produces images that are difficult to interpret, unless they can be rotated in realtime
  - sweeping a **cross-section** plane through the room and showing a color-coded temperature on it
  - evolving p and showing the deformation of the **isosurface** S(p)
- For **4D** data sets?
  - Let the user control time and rotate interactively the **volumetric** rendering of the **time-slice**
  - Sweeping a **cross-section** of evolving p would produce a different **movie** for each time slice t.
    - The scientist would need to watch them all and integrate in her head how they relate to each other.
    - It is better to provide her with the means to **control p and t interactively**. But how would she chose to evolve p and t? It is a large 2D space!



SAFARI

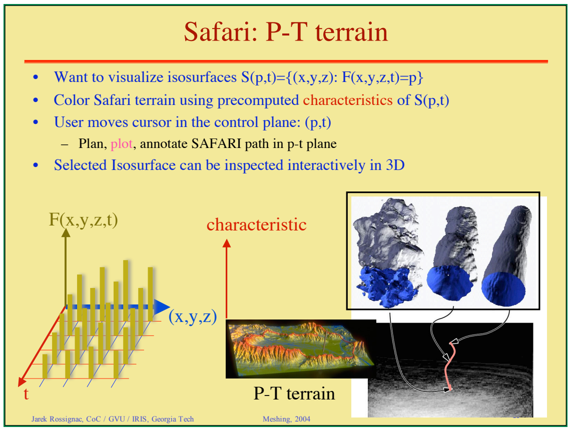
Interactive isosurface-based inspection of 4D structured data

with
Peter Lindstrom (LLNL)
Lutz Kettner (UNC)
Valerio Pascucci (LLNL)
Jack Snoeyink (UNC-Chapel Hill)

Value 128
Time 99
Exit

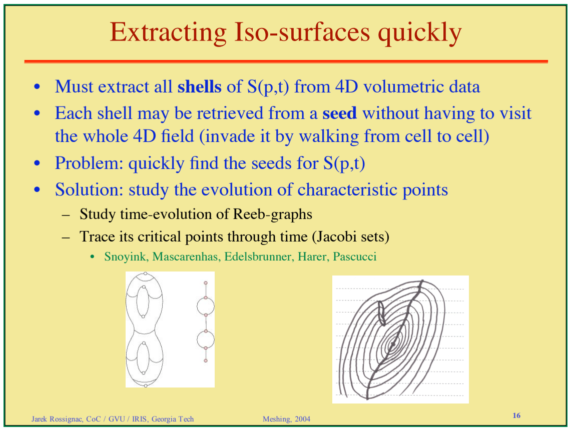
L. Kettner, J. Rossignac, J. Snoeyink, The Safari Interface for Visualizing Time-dependent Volume Data Using Iso-Surfaces and a Control Plane, CGTA 25:1-2(2003), pages 97-116

A. Mascarenhas, J. Snoeyink, Seed Set Computation for Isosurface Extraction in Time-varying Volumetric Data



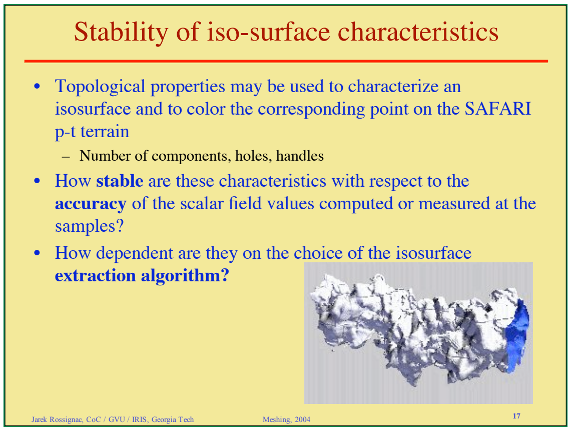
Safari: P-T terrain

- Want to visualize isosurfaces S(p,t)={(x,y,z): F(x,y,z,t)=p}
- Color Safari terrain using precomputed characteristics of S(p,t)
- User moves cursor in the control plane: (p,t)
  - Plan, plot, annotate SAFARI path in p-t plane
- Selected Isosurface can be inspected interactively in 3D



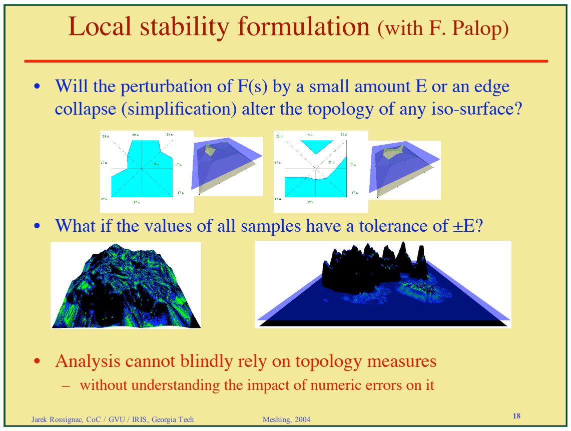
Extracting Iso-surfaces quickly

- Must extract all **shells** of S(p,t) from 4D volumetric data
- Each shell may be retrieved from a **seed** without having to visit the whole 4D field (invade it by walking from cell to cell)
- Problem: quickly find the seeds for S(p,t)
- Solution: study the evolution of characteristic points
  - Study time-evolution of Reeb-graphs
  - Trace its critical points through time (Jacobi sets)
    - Snoyink, Mascarenhas, Edelsbrunner, Harer, Pascucci



Stability of iso-surface characteristics

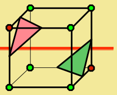
- Topological properties may be used to characterize an isosurface and to color the corresponding point on the SAFARI p-t terrain
  - Number of components, holes, handles
- How **stable** are these characteristics with respect to the **accuracy** of the scalar field values computed or measured at the samples?
- How dependent are they on the choice of the isosurface **extraction algorithm?**



Local stability formulation (with F. Palop)

- Will the perturbation of F(s) by a small amount E or an edge collapse (simplification) alter the topology of any iso-surface?
- What if the values of all samples have a tolerance of ±E?
- Analysis cannot blindly rely on topology measures
  - without understanding the impact of numeric errors on it

MINIMAC



Minimizing the computational and topological complexity of iso-surfaces extracted from structured data

with

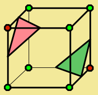
C. Andujar, P. Brunet, A. Chica, I. Navazo, A. Vinacua (UPC)

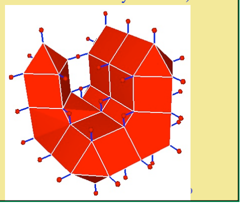
“Optimal Iso-Surfaces", C Andujar, P Brunet, A Chica, I Navazo, J Rossignac, A Vinacua. CAD Conference, 2004. Best Paper Award.

"Optimizing the topological and combinatorial complexity of isosurfaces", P Brunet, A Chica, I Navazo, J Rossignac, A Vinacua. *Journal of Computer Aided Design.*

Isosurface connectivity in 3D

- Threshold the samples: **green** (below p, or inside) or **red** (above p or outside)
- A **cube** (not a voxel) spans interstice between 8 samples
- Each **vertex** of isosurface lies on lattice-edge between red and green vertices
- Only cubes with **mixed** (red/green) vertices contribute to the isosurface
- The contribution is bounded by edges on mixed (red/green) **faces**
- These edges form **loops** on the boundary of each mixed cube
- These loops bound **sheets** (portions of isosurface contributed by the cube)

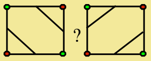




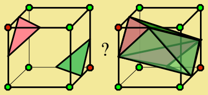
- Is the isosurface connectivity defined by the red/green labeling?

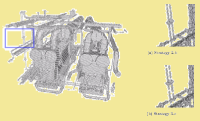
Exploiting ambiguities

- We can choose how to cut an **x-face** (face with alternating red/green vertices)



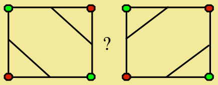
- If a cube has more than one loop, we can choose whether to interpolate them in a single sheet, or have separate sheets





- We make choices that minimize different measures of “complexity”
  - **triangle count** T and (**genus** H or number of shells S)
- Triangulating each loop independently reduces T
- **Increasing L** reduces T (=4V–2L) and (H–S) (= (V–L)/2)

How to maximize L ?



- Cube with no x-faces:
  - Nothing to decide
- Cube with 1 x-face:
  - Slash it one way and then **flip if this adds a loop** on the cube
    - Flip changes by 1 the number of loops for both incident cubes
    - Flipping an x-face changes L by –2, 0, or 2 (can’t improve on our strategy)
- Cubes with several x-faces:
  - Build their **connectivity graphs** (x-faces are links)
  - Propagate choices from **leaves** (cubes with a single x-face)
  - Break **cycles** arbitrarily and resume
- Analysis should use optimal connectivity/topology measures

Transmitting isosurfaces and animations

- **Isosurfaces** could be transmitted implicitly by sending the volumetric data used to derive them
  - **But** the scientist may not be interested in all the isosurfaces
  - And the volumetric data set may be far too large to be transmitted or to fit on the client station
- **Animations** could be transmitted implicitly by sending the initial conditions and the physical attributes or designer’s directives
  - **But** the computation of the simulation (including collisions and dynamics) may be too expensive to be rederived on the client
- Therefore, it is important to develop compact encodings for static or time-dependent (iso)surfaces

EDGEBREAKER

Tri-mesh representation and compression (a quick review)

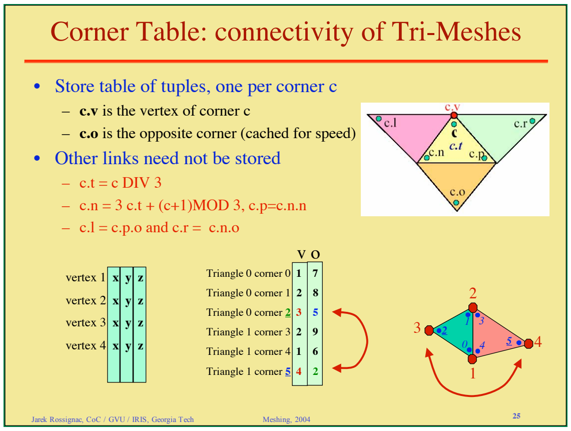
with

Andrzej Szymczak (Georgia Tech)

"Edgebreaker: Connectivity compression for triangle meshes", J. Rossignac. IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, pp. 47-61, January - March 1999. GVU Tech. Report GIT-GVU-98-35.

"Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker", J. Rossignac and A. Szymczak, Journal of Computational Geometry, Theory and Applications, Volume 14, Issue 1-3, pp. 119-135, November 1999. GVU Tech. Report GIT-GVU-99-08.

“3D compression made simple: Edgebreaker on a Corner Table”, J. Rossignac, A. Safonova, A. Szymczak, J. Rossignac, Shape Modeling International Conference, pp: 278-283, Genoa, Italy May 2001.

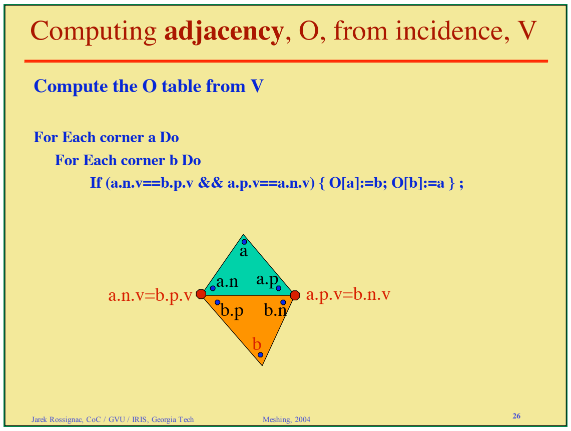


Corner Table: connectivity of Tri-Meshes

- Store table of tuples, one per corner c
  - **c.v** is the vertex of corner c
  - **c.o** is the opposite corner (cached for speed)
- Other links need not be stored
  - c.t = c DIV 3
  - c.n = 3 c.t + (c+1)MOD 3, c.p=c.n.n
  - c.l = c.p.o and c.r = c.n.o

vertex 1	x	y	z
vertex 2	x	y	z
vertex 3	x	y	z
vertex 4	x	y	z

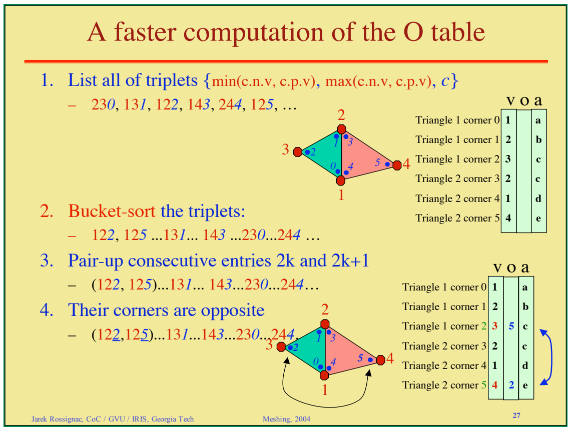
	V	O
Triangle 0 corner 0	1	7
Triangle 0 corner 1	2	8
Triangle 0 corner 2	3	5
Triangle 1 corner 3	2	9
Triangle 1 corner 4	1	6
Triangle 1 corner 5	4	2



Computing **adjacency**, O, from incidence, V

Compute the O table from V

```
For Each corner a Do
    For Each corner b Do
        If (a.n.v==b.p.v && a.p.v==a.n.v) { O[a]:=b; O[b]:=a } ;
```

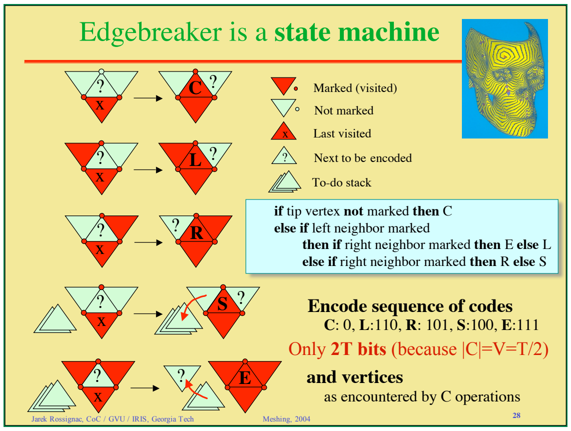


A faster computation of the O table

1. List all of triplets {min(c.n.v, c.p.v), max(c.n.v, c.p.v), *c*}
   - 23*0*, 13*1*, 12*2*, 14*3*, 24*4*, 12*5*, …
2. Bucket-sort the triplets:
   - 12*2*, 12*5* ...13*1*... 14*3* ...23*0*...24*4* …
3. Pair-up consecutive entries 2k and 2k+1
   - (12*2*, 12*5*)...13*1*... 14*3*...23*0*...24*4*…
4. Their corners are opposite
   - (12*2*,12*5*)...13*1*...14*3*...23*0*...24*4*…

	v	o	a
Triangle 1 corner 0	1		a
Triangle 1 corner 1	2		b
Triangle 1 corner 2	3		c
Triangle 2 corner 3	2		c
Triangle 2 corner 4	1		d
Triangle 2 corner 5	4		e

	v	o	a
Triangle 1 corner 0	1		a
Triangle 1 corner 1	2		b
Triangle 1 corner 2	3	5	c
Triangle 2 corner 3	2		c
Triangle 2 corner 4	1		d
Triangle 2 corner 5	4	2	e



Edgebreaker is a **state machine**

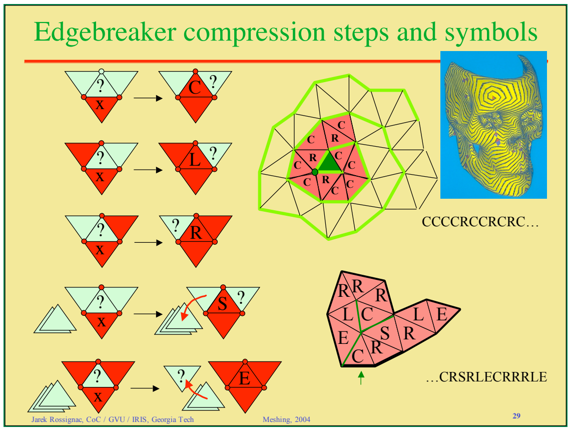
🟥 ●	Marked (visited)
▽ ○	Not marked
x	Last visited
?	Next to be encoded
△	To-do stack

if tip vertex **not** marked **then** C
else if left neighbor marked
    then if right neighbor marked **then** E **else** L
    else if right neighbor marked **then** R **else** S

Encode sequence of codes
C: 0, **L**:110, **R**: 101, **S**:100, **E**:111

Only **2T bits** (because |C|=V=T/2)

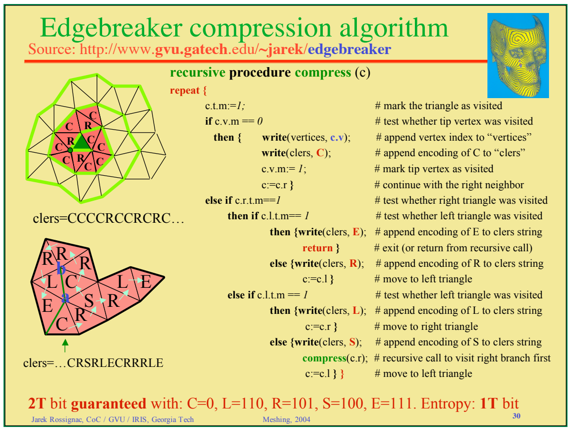
and vertices
as encountered by C operations



Edgebreaker compression steps and symbols

CCCCRCCRCRC…

…CRSRLECRRRLE



Edgebreaker compression algorithm

Source: http://www.**gvu.gatech**.edu/**~jarek**/**edgebreaker**

clers=CCCCRCCRCRC…

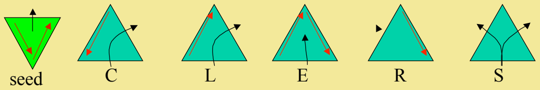
clers=…CRSRLECRRRLE

```
recursive procedure compress (c)
repeat {
        c.t.m:=1;                              # mark the triangle as visited
        if c.v.m == 0                          # test whether tip vertex was visited
          then {    write(vertices, c.v);      # append vertex index to “vertices”
                    write(clers, C);           # append encoding of C to “clers”
                    c.v.m:= 1;                 # mark tip vertex as visited
                    c:=c.r }                   # continue with the right neighbor
        else if c.r.t.m==1                     # test whether right triangle was visited
             then if c.l.t.m== 1               # test whether left triangle was visited
                  then {write(clers, E);       # append encoding of E to clers string
                        return }               # exit (or return from recursive call)
                  else {write(clers, R);       # append encoding of R to clers string
                        c:=c.l }               # move to left triangle
             else if c.l.t.m == 1              # test whether left triangle was visited
                  then {write(clers, L);       # append encoding of L to clers string
                        c:=c.r }               # move to right triangle
                  else {write(clers, S);       # append encoding of S to clers string
                        compress(c.r);         # recursive call to visit right branch first
                        c:=c.l } }             # move to left triangle
```

2T bit **guaranteed** with: C=0, L=110, R=101, S=100, E=111. Entropy: **1T** bit

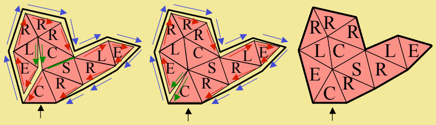
Wrap&Zip EB decompression (with Szymczak)

Orient bounding edges while building triangle tree at decompression. All oriented clockwise (up tree), except for C and the seed triangle:



Then ZIP all pairs of adjacent bounding edges when both point away from their common vertex.

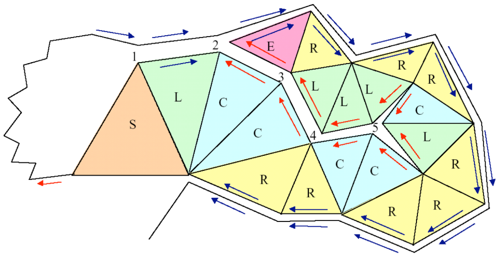
CRSRLECRRRLE



Linear time complexity. Zip only after L and E.

Wrap&Zip more complex **example**

The *clers* string SLCCRRCCRRRLCRRLLLRE will generate the mesh below. The left edge of the first L triangle is not zipped immediately. The left edge of the second L triangle is zipped reaching vertex 5. Then, as we encounter the subsequent three L triangles, their left edges are zipped right away. The first let edge of the E triangle is also zipped. The rest will be zipped later, when the left branch of the split is done.

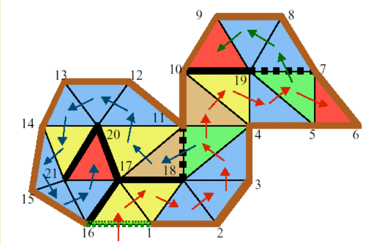


Vertex data compression

1. Reorder
2. Normalize and quantize
3. Predict
4. Compress residues

Reordering the vertices

- Vertices transmitted in the order in which they are first encountered by the Edgebreaker traversal

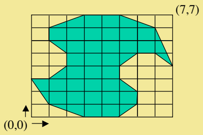


Integers are best for coordinates

- Most Tri-meshes are only **approximations** of real shapes
  - Imprecise measures or toleranced dimensions
  - Limited accuracy in geometric computation
  - Truncated coordinates to nearest float or double
- You should use very short integers to represent their coordinates
  - Floats are bad for geometry
    - Accuracy is not uniform: it grows with distance to origin
    - Most geometric models/computations need uniform accuracy
  - Integers are perfect
    - Uniform accuracy through out the model
    - More precise than floats for same number of bits
  - Don't need 32 bits
    - 11 bits are usually sufficient
      - they guarantee 0.5mm accuracy for an engine block

Vertex normalization and **quantization**

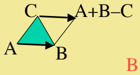
- Represent coordinates as **normalized** integers
  - Coordinates **relative** to bounding rectangle
  - Select unit for **desired resolution** $[0..2^B-1]$
  - Vertex coordinates = **B-bit integers** ($6<B<14$) Error E_B

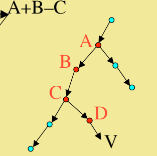


Bunny after an overdose of quantrization

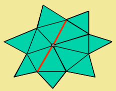
Techniques for predicting a vertex

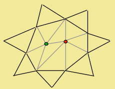
- Displacement from previous vertex (Deering)
- **Parallelogram** (Touma&Gotsman)
- Tree ancestors (Taubin&Rossignac)
  - Construct **vertex spanning tree**
  - Predict using linear combination of **4 ancestors**
    - Compute optimal **coefficients (a,b,c,d)**
  - Encode corrective vector (**X**)
  - Compresses vertex location to 12 bits per vertex
- Vertex split (Hoppe)
- Crowns (Pajarola&Rossignac)

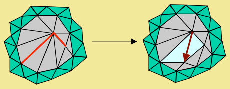




V=aA+bB+cC+dD+X







Techniques for **encoding** the residues

- Normalize and quantize the coordinates
- Predict each vertex from previously encountered ones
- **Compute the residues (actual – predicted) locations**
  - If predictions are good, residues are biased towards zero
- **Use variable length coding to compress residues**
  - More frequent symbols (0, –1, 1,–2, 2…) will have shorter codes then less frequent symbols (297, –319…)

Combined compression and improvements

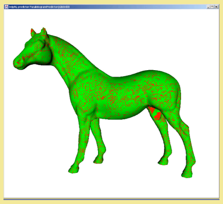
Total: 1 byte per triangle

- Connectivity compression is guaranteed to yield less than **2 bits per triangle** (improved Edgebreaker guarantees 1.8)
  - In practice it yields about **1 bit per triangle** (arithmetic coding)
  - Can we further compress connectivity without loss?
    - Why not predict it?
- Good geometry compression yields about **4 bits per coordinate** (varies widely with smoothness of the model, density of sampling, quantization used).
  - This is about **6 bits per triangle** (T=2V…)
  - What can we do to further increase total compression?
    - Simplify the model (introduce loss)
    - Resample (optimize connectivity and vertex location for compression)

Delphi: Connectivity Prediction

with

Volker Coors



"Guess Connectivity: Delphi Encoding in Edgebreaker", V. Coors and J. Rossignac, TVC

Delphi: Guessed Connectivity = **0.74T bits**

- Predict clers symbols from decoded mesh, encode corrections

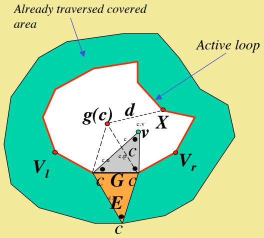
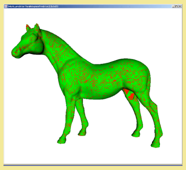


Figure 2: *Connectivity guessed by parallelogram prediction*

Depending on the model, **between 51% and 97% of guesses are correct.**



83% correct guesses: **0.74T bits**

Must encode wrongly guessed S-offsets!

Apollo sequence encoding of Delphi

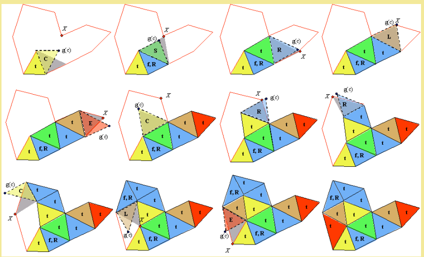
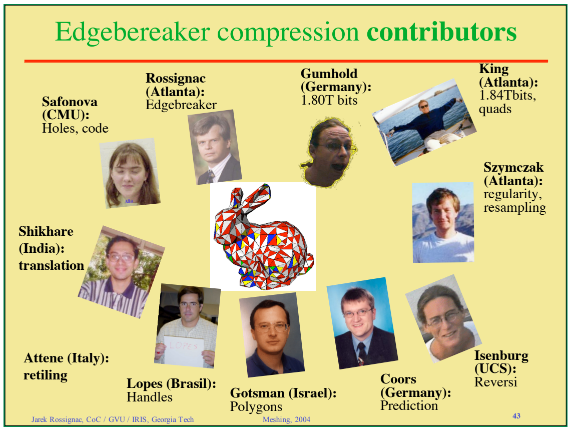


Figure 6: *Example Apollo encoding: Let us assume that we guessed the first triangle of the example correctly as type C. We than predict the tip of the right triangle at g(c) using the parallelogram rule. SinceBecause the distance of g(c) and the active border is too large, we guess again a type C triangle. Unfortunately, that guess was wrong. In fact, the right triangle, shown in gray color in the first picture, is of type R. In the Apollo sequence we encode this situation as (f,R) and continue the traversal with the left triangle of R. The prediction scheme is performed for all triangle in Edgebreaker sequence and leads to the following Apollo sequence: ((t), (f, R), (t), (t), (t), (t), (t), (t), (t), (f,R), (t), (t), (t)). With a trivial encoding scheme we can compress this sequence with 16 bits instead of 32 bits for the corresponding CLERS sequence.*



Edgebereaker compression **contributors**

Safonova (CMU): Holes, code

Rossignac (Atlanta): Edgebreaker

Gumhold (Germany): 1.80T bits

King (Atlanta): 1.84Tbits, quads

Szymczak (Atlanta): regularity, resampling

Shikhare (India): translation

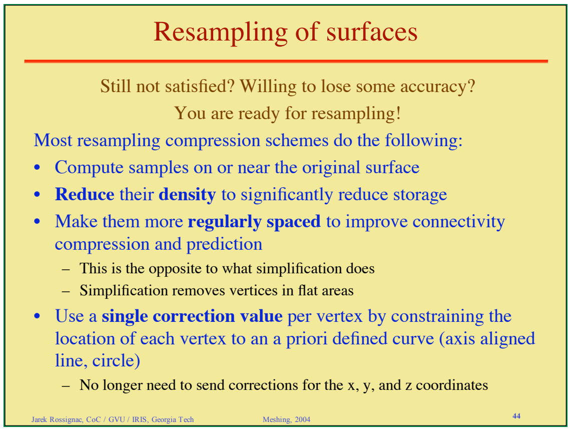
Attene (Italy): retiling

Lopes (Brasil): Handles

Gotsman (Israel): Polygons

Coors (Germany): Prediction

Isenburg (UCS): Reversi



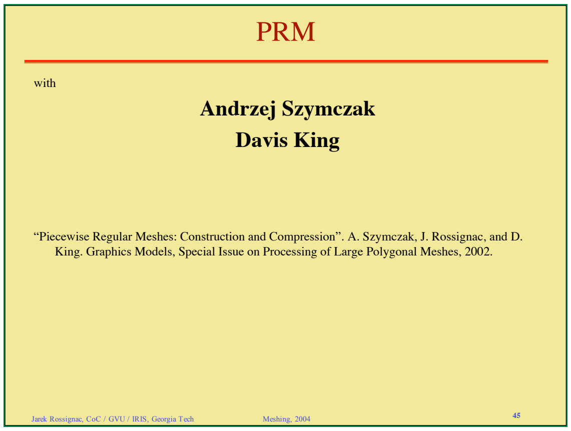
Resampling of surfaces

Still not satisfied? Willing to lose some accuracy?

You are ready for resampling!

Most resampling compression schemes do the following:

- Compute samples on or near the original surface
- **Reduce** their **density** to significantly reduce storage
- Make them more **regularly spaced** to improve connectivity compression and prediction
  - This is the opposite to what simplification does
  - Simplification removes vertices in flat areas
- Use a **single correction value** per vertex by constraining the location of each vertex to an a priori defined curve (axis aligned line, circle)
  - No longer need to send corrections for the x, y, and z coordinates



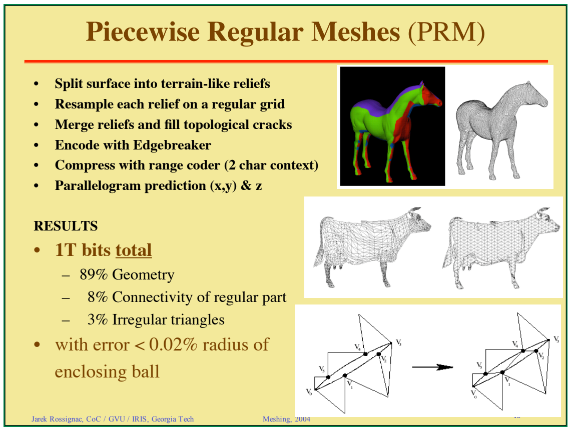
PRM

with

Andrzej Szymczak

Davis King

"Piecewise Regular Meshes: Construction and Compression". A. Szymczak, J. Rossignac, and D. King. Graphics Models, Special Issue on Processing of Large Polygonal Meshes, 2002.

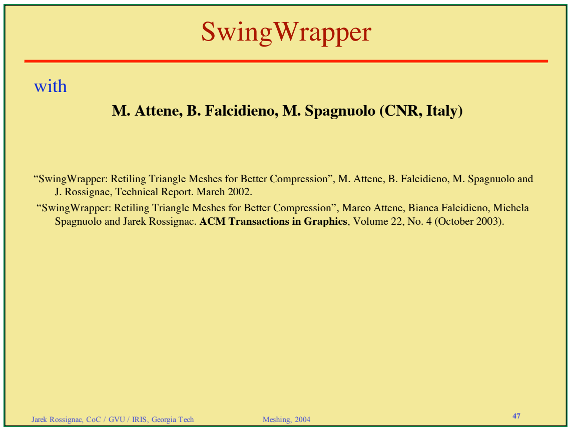


Piecewise Regular Meshes (PRM)

- **Split surface into terrain-like reliefs**
- **Resample each relief on a regular grid**
- **Merge reliefs and fill topological cracks**
- **Encode with Edgebreaker**
- **Compress with range coder (2 char context)**
- **Parallelogram prediction (x,y) & z**

RESULTS

- **1T bits total**
  - 89% Geometry
  - 8% Connectivity of regular part
  - 3% Irregular triangles
- with error < 0.02% radius of enclosing ball



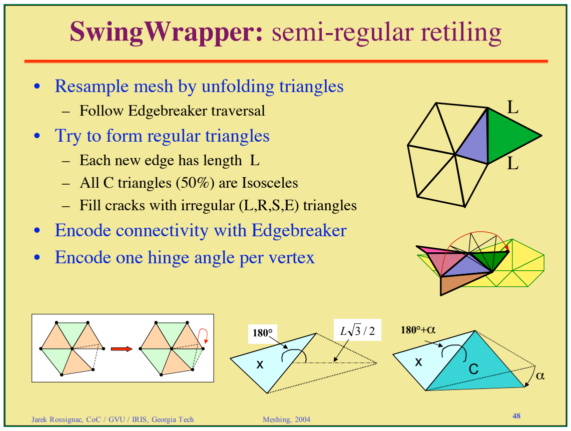
SwingWrapper

with

M. Attene, B. Falcidieno, M. Spagnuolo (CNR, Italy)

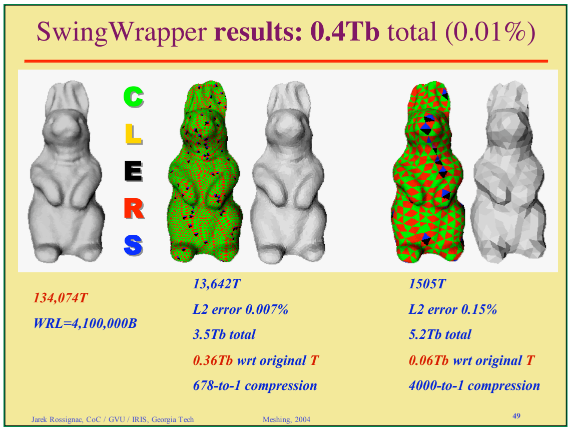
"SwingWrapper: Retiling Triangle Meshes for Better Compression", M. Attene, B. Falcidieno, M. Spagnuolo and J. Rossignac, Technical Report. March 2002.

"SwingWrapper: Retiling Triangle Meshes for Better Compression", Marco Attene, Bianca Falcidieno, Michela Spagnuolo and Jarek Rossignac. **ACM Transactions in Graphics**, Volume 22, No. 4 (October 2003).



SwingWrapper: semi-regular retiling

- Resample mesh by unfolding triangles
  - Follow Edgebreaker traversal
- Try to form regular triangles
  - Each new edge has length L
  - All C triangles (50%) are Isosceles
  - Fill cracks with irregular (L,R,S,E) triangles
- Encode connectivity with Edgebreaker
- Encode one hinge angle per vertex



SwingWrapper **results: 0.4Tb** total (0.01%)

134,074T

WRL=4,100,000B

13,642T

L2 error 0.007%

3.5Tb total

0.36Tb wrt original T

678-to-1 compression

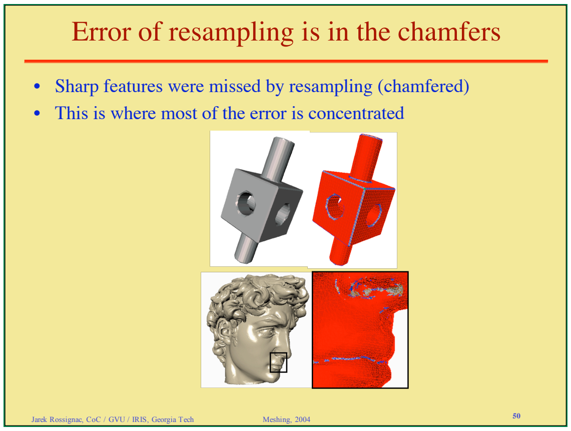
1505T

L2 error 0.15%

5.2Tb total

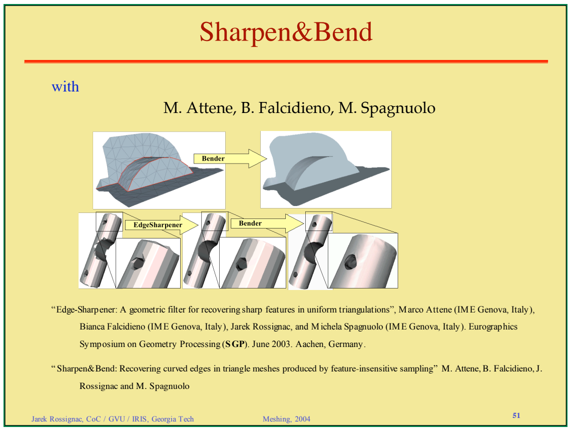
0.06Tb wrt original T

4000-to-1 compression



Error of resampling is in the chamfers

- Sharp features were missed by resampling (chamfered)
- This is where most of the error is concentrated



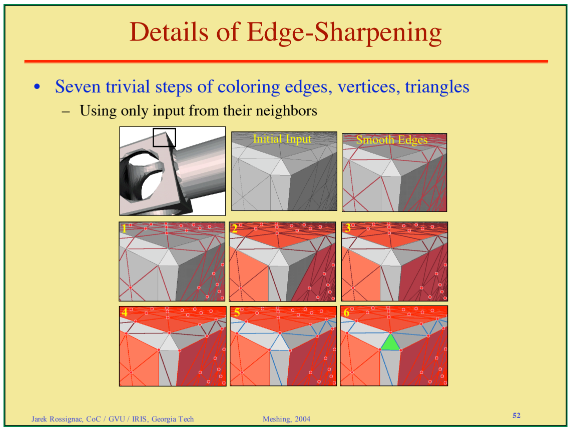
Sharpen&Bend

with

M. Attene, B. Falcidieno, M. Spagnuolo

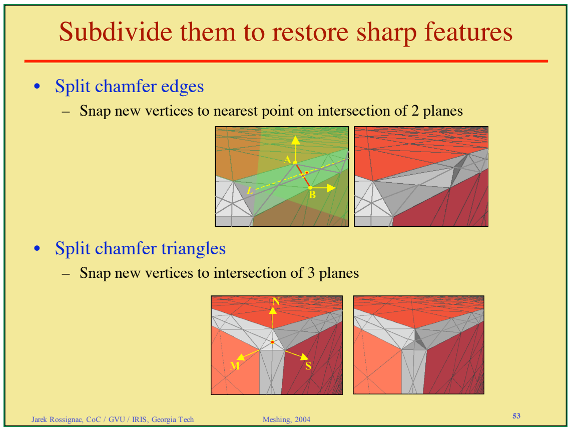
"Edge-Sharpener: A geometric filter for recovering sharp features in uniform triangulations", Marco Attene (IME Genova, Italy), Bianca Falcidieno (IME Genova, Italy), Jarek Rossignac, and Michela Spagnuolo (IME Genova, Italy). Eurographics Symposium on Geometry Processing (**SGP**). June 2003. Aachen, Germany.

"Sharpen&Bend: Recovering curved edges in triangle meshes produced by feature-insensitive sampling" M. Attene, B. Falcidieno, J. Rossignac and M. Spagnuolo



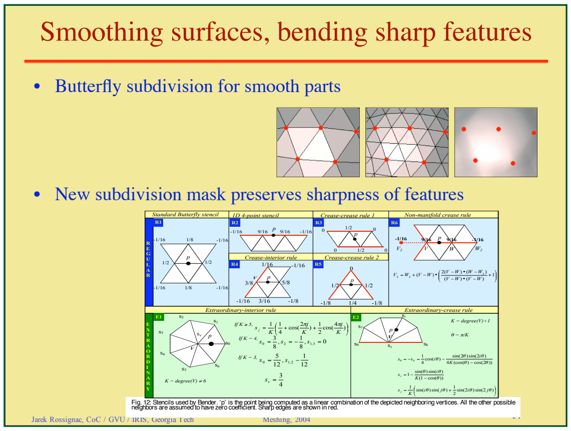
Details of Edge-Sharpening

- Seven trivial steps of coloring edges, vertices, triangles
  - Using only input from their neighbors



Subdivide them to restore sharp features

- Split chamfer edges
  - Snap new vertices to nearest point on intersection of 2 planes
- Split chamfer triangles
  - Snap new vertices to intersection of 3 planes



Smoothing surfaces, bending sharp features

- Butterfly subdivision for smooth parts
- New subdivision mask preserves sharpness of features

$$V_2 = W_2 + (V-W)\bullet\left(\frac{2(V-W)\bullet(W-W_2)}{(V-W)\bullet(V-W)}+1\right)$$

$$\text{If } K \geq 5,\ s_j = \frac{1}{K}\left(\frac{1}{4}+\cos\left(\frac{2\pi j}{K}\right)+\frac{1}{2}\cos\left(\frac{4\pi j}{K}\right)\right)$$

$$\text{If } K = 4,\ s_0 = \frac{3}{8}, s_2 = -\frac{1}{8}, s_{1,3} = 0$$

$$\text{If } K = 3,\ s_0 = \frac{5}{12}, s_{1,2} = -\frac{1}{12}$$

$$s_v = \frac{3}{4}$$

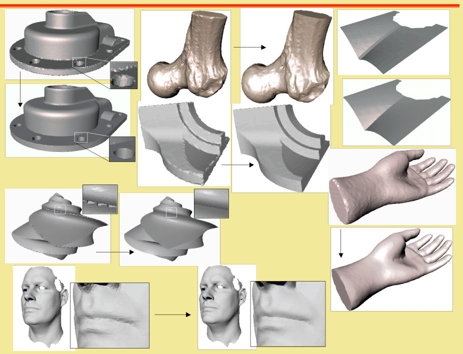
$$s_0 = -s_k = \frac{1}{4}\cos(i\theta) - \frac{\sin(2\theta)\sin(2i\theta)}{4K(\cos(\theta)-\cos(2\theta))}$$

$$s_v = 1 - \frac{\sin(\theta)\sin(i\theta)}{K(1-\cos(\theta))}$$

$$s_j = \frac{1}{K}\left(\sin(i\theta)\sin(j\theta) + \frac{1}{2}\sin(2i\theta)\sin(2j\theta)\right)$$

Fig. 12: Stencils used by Bender. 'p' is the point being computed as a linear combination of the depicted neighboring vertices. All the other possible neighbors are assumed to have zero coefficient. Sharp edges are shown in red.

Sharpen&Bend results



Automatic reduction of resampling error

- **Sharp features** are missed by resampling
- Resampling replaces **smooth surfaces** (or close tiled approximations of them) by coarse polyhedral models
- **Sharpen&Bend restores the sharp features and the smooth surfaces without any further information or user input**
- Significantly **reduces the error** due to resampling

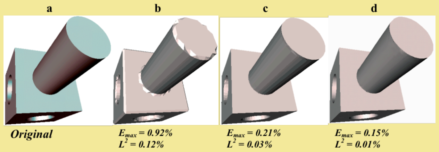

a
b
c
d
Original
$E_{max} = 0.92\%$ $L^2 = 0.12\%$
$E_{max} = 0.21\%$ $L^2 = 0.03\%$
$E_{max} = 0.15\%$ $L^2 = 0.01\%$

Fig. 1: An original model (a) was re-meshed through a feature-insensitive algorithm (b). The sharp edges and corners were restored by EdgeSharpener (c). Then, Bender faired the smooth regions without rounding off the sharp features reconstructed by EdgeSharpener (d). For each model, the maximum distance from the original surface (E_{max}) and the mean-squared distortion (L^2) are reported. All the values are percent of the bounding-box diagonal.

From surfaces to animations

- Many 3D animations are represented (and transmitted) as series of 3D frames (triangle meshes)
- To compress an animation, one may simply compress each 3D frame independently
  - Great if you decompress directly on the graphics hardware (Deering).
- However, better compressions may be obtained by treating the whole animation (or a short clip) as a whole and compressing it by exploiting spatial and temporal coherence.
- Many animations have fixed connectivity
  - It needs to be transmitted only once
    - Unless you simplify the different frames independently

DYNAPACK

Compression and simplification of animations

with
L. Ibarria (GaTech)
P. Lindstrom (LLNL)

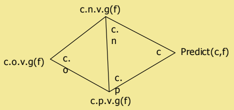
“Dynapack: Space-Time compression of the 3D animations of triangle meshes with fixed connectivity”, L. Ibarria and J. Rossignac. ACM SIGGRAPH Symposium on Computer Animation, 2003.

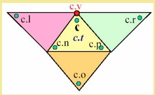
“Clippacker: Simplification and Compression of 3D Animations”, Lorenzo Ibarria, Peter Lindstrom, Jarek Rossignac

Space or time predictors

- Assume **same connectivity** for all frames
  - Send connectivity up-front
    - Compressed with Edgebreaker

- Could predict the **trajectory** of each vertex independently
  - predict(c**,**f) = c.n.v.g(f-1) or higher order (**Time only predictor**)
  - Does not exploit spatial coherence
- Could predict vertex location in each frame (**parallelogram**)
  - predict(c**,**f) = c.n.v.g(f)+c.p.v.g(f)–c.o.v.g(f) (**Space only predictor**)
  - Does not exploit temporal coherence



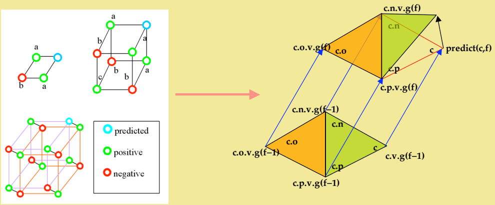


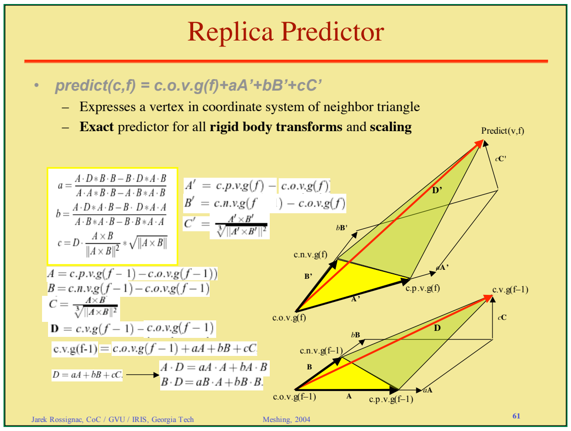
Extended Lorenzo Predictor (ELP)

Use parallelogram to **predict** the **speed** of the next vertex from neighbors’ speed

predict(c,f) = c.v.g(f–1) **+** (c.n.v.g(f) – c.n.v.g(f–1))
+ (c.p.v.g(f) – c.p.v.g(f–1)) **–** (c.o.v.g(f) + c.o.v.g(f–1))

- Exploits both **space and time coherence**
- **Perfect** predictor for **translations**





Replica Predictor

- *predict(c,f) = c.o.v.g(f)+aA'+bB'+cC'*
  - Expresses a vertex in coordinate system of neighbor triangle
  - **Exact** predictor for all **rigid body transforms** and **scaling**

$$a = \frac{A\cdot D * B\cdot B - B\cdot D * A\cdot B}{A\cdot A * B\cdot B - A\cdot B * A\cdot B}$$

$$b = \frac{A\cdot D * A\cdot B - B\cdot D * A\cdot A}{A\cdot B * A\cdot B - B\cdot B * A\cdot A}$$

$$c = D\cdot \frac{A\times B}{\|A\times B\|^2} * \sqrt{\|A\times B\|}$$

$$A' = c.p.v.g(f) - c.o.v.g(f)$$

$$B' = c.n.v.g(f \quad) - c.o.v.g(f)$$

$$C' = \frac{A'\times B'}{\sqrt[3]{\|A'\times B'\|^2}}$$

$$A = c.p.v.g(f-1) - c.o.v.g(f-1))$$

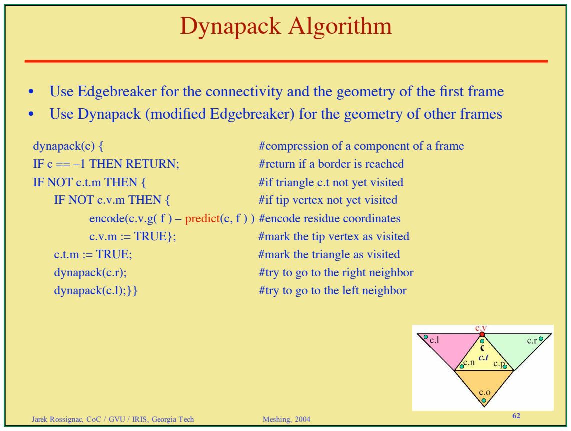
$$B = c.n.v.g(f-1) - c.o.v.g(f-1)$$

$$C = \frac{A\times B}{\sqrt[3]{\|A\times B\|^2}}$$

$$\mathbf{D} = c.v.g(f-1) - c.o.v.g(f-1)$$

$$\text{c.v.g(f-1)} = c.o.v.g(f-1) + aA + bB + cC$$

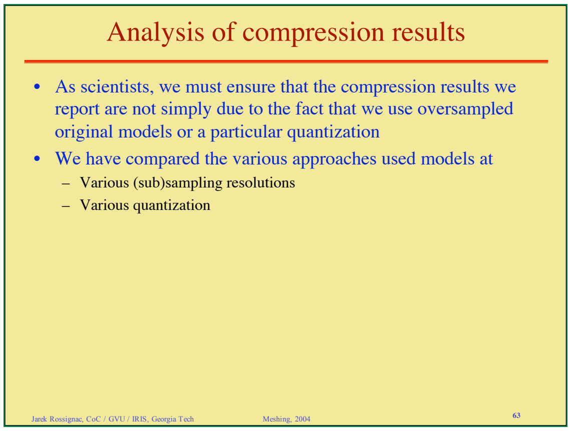
$$D = aA + bB + cC \longrightarrow \begin{array}{l} A\cdot D = aA\cdot A + bA\cdot B \\ B\cdot D = aB\cdot A + bB\cdot B \end{array}$$



Dynapack Algorithm

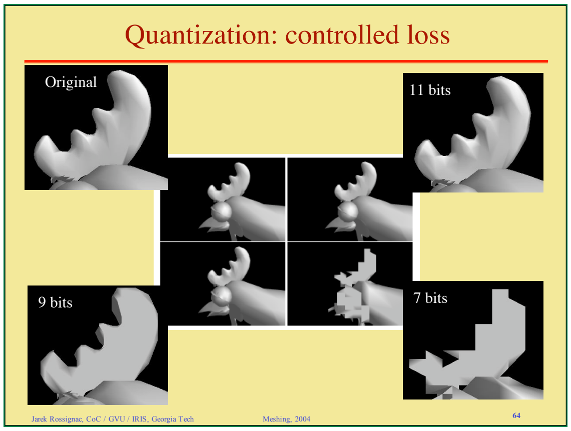
- Use Edgebreaker for the connectivity and the geometry of the first frame
- Use Dynapack (modified Edgebreaker) for the geometry of other frames

```
dynapack(c) {                                   #compression of a component of a frame
IF c == –1 THEN RETURN;                         #return if a border is reached
IF NOT c.t.m THEN {                             #if triangle c.t not yet visited
    IF NOT c.v.m THEN {                         #if tip vertex not yet visited
            encode(c.v.g( f ) – predict(c, f ) ) #encode residue coordinates
            c.v.m := TRUE};                     #mark the tip vertex as visited
    c.t.m := TRUE;                              #mark the triangle as visited
    dynapack(c.r);                              #try to go to the right neighbor
    dynapack(c.l);}}                            #try to go to the left neighbor
```

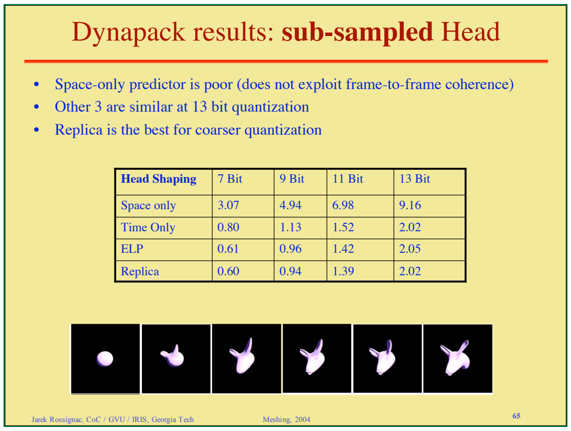


Analysis of compression results

- As scientists, we must ensure that the compression results we report are not simply due to the fact that we use oversampled original models or a particular quantization
- We have compared the various approaches used models at
  - Various (sub)sampling resolutions
  - Various quantization



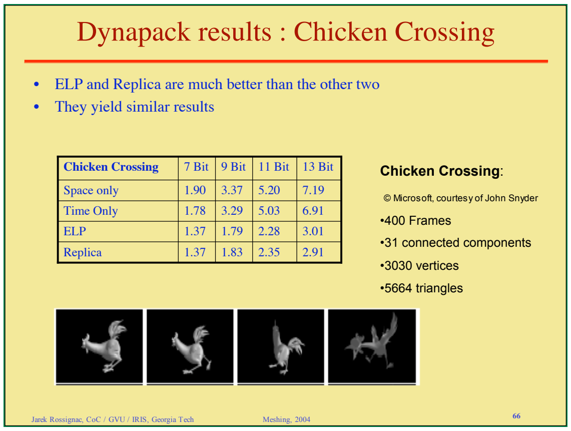
Quantization: controlled loss



Dynapack results: **sub-sampled** Head

- Space-only predictor is poor (does not exploit frame-to-frame coherence)
- Other 3 are similar at 13 bit quantization
- Replica is the best for coarser quantization

Head Shaping	7 Bit	9 Bit	11 Bit	13 Bit
Space only	3.07	4.94	6.98	9.16
Time Only	0.80	1.13	1.52	2.02
ELP	0.61	0.96	1.42	2.05
Replica	0.60	0.94	1.39	2.02



Dynapack results : Chicken Crossing

- ELP and Replica are much better than the other two
- They yield similar results

Chicken Crossing	7 Bit	9 Bit	11 Bit	13 Bit
Space only	1.90	3.37	5.20	7.19
Time Only	1.78	3.29	5.03	6.91
ELP	1.37	1.79	2.28	3.01
Replica	1.37	1.83	2.35	2.91

Chicken Crossing:

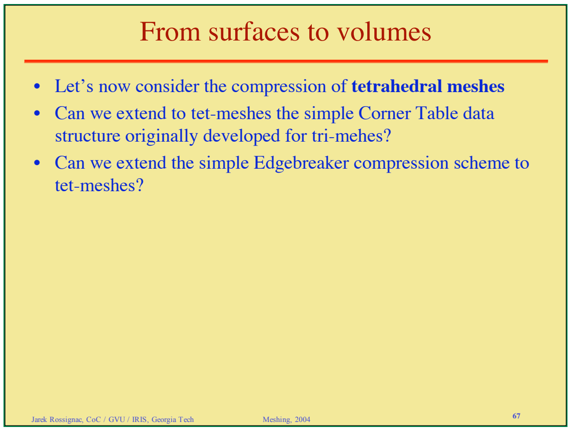
© Microsoft, courtesy of John Snyder

•400 Frames

•31 connected components

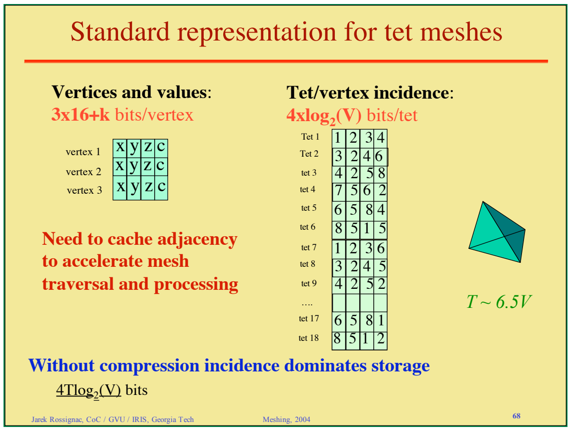
•3030 vertices

•5664 triangles



From surfaces to volumes

- Let's now consider the compression of **tetrahedral meshes**
- Can we extend to tet-meshes the simple Corner Table data structure originally developed for tri-mehes?
- Can we extend the simple Edgebreaker compression scheme to tet-meshes?



Standard representation for tet meshes

Vertices and values:
3x16+k bits/vertex

vertex 1	x	y	z	c
vertex 2	x	y	z	c
vertex 3	x	y	z	c

Need to cache adjacency to accelerate mesh traversal and processing

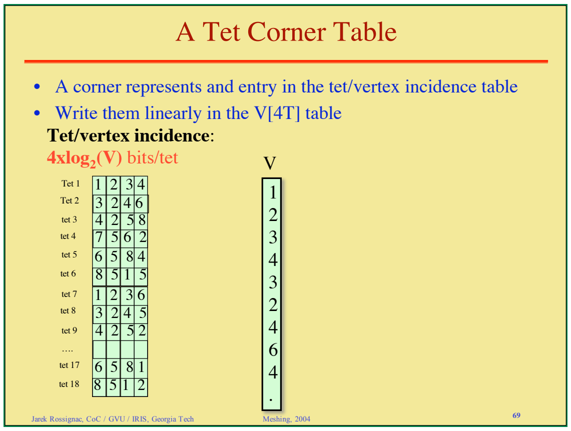
Tet/vertex incidence:
$4x\log_2(V)$ bits/tet

Tet 1	1	2	3	4
Tet 2	3	2	4	6
tet 3	4	2	5	8
tet 4	7	5	6	2
tet 5	6	5	8	4
tet 6	8	5	1	5
tet 7	1	2	3	6
tet 8	3	2	4	5
tet 9	4	2	5	2
….				
tet 17	6	5	8	1
tet 18	8	5	1	2

$T \sim 6.5V$

Without compression incidence dominates storage

$\underline{4T\log_2(V)}$ bits



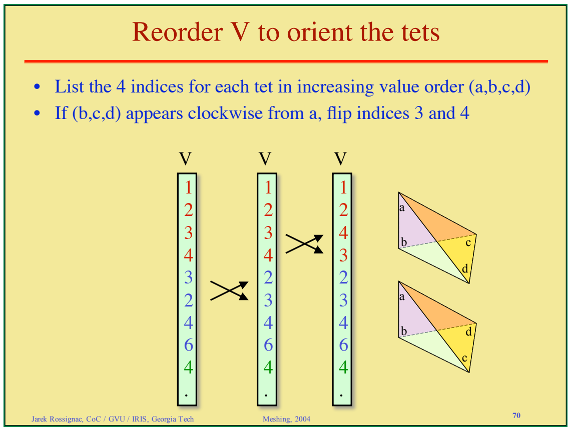
A Tet Corner Table

- A corner represents and entry in the tet/vertex incidence table
- Write them linearly in the V[4T] table

Tet/vertex incidence:
$4x\log_2(V)$ bits/tet

Tet 1	1	2	3	4
Tet 2	3	2	4	6
tet 3	4	2	5	8
tet 4	7	5	6	2
tet 5	6	5	8	4
tet 6	8	5	1	5
tet 7	1	2	3	6
tet 8	3	2	4	5
tet 9	4	2	5	2
….				
tet 17	6	5	8	1
tet 18	8	5	1	2

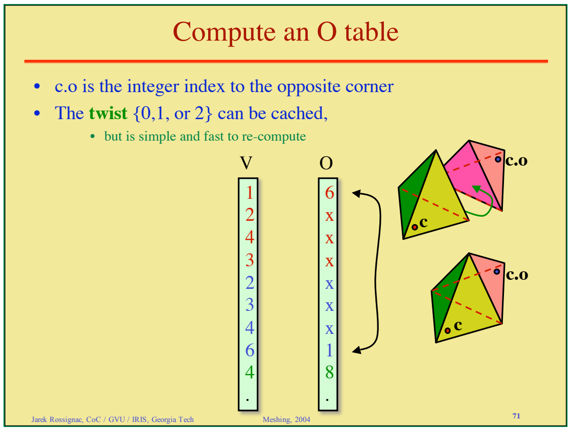
V
1
2
3
4
3
2
4
6
4
.



Reorder V to orient the tets

- List the 4 indices for each tet in increasing value order (a,b,c,d)
- If (b,c,d) appears clockwise from a, flip indices 3 and 4

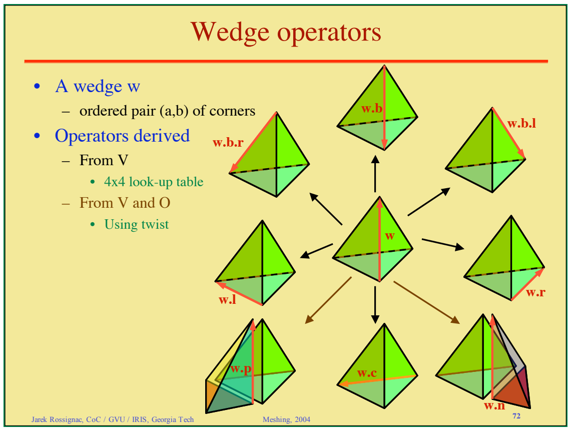
V	V	V
1	1	1
2	2	2
3	3	4
4	4	3
3	2	2
2	3	3
4	4	4
6	6	6
4	4	4
.	.	.



Compute an O table

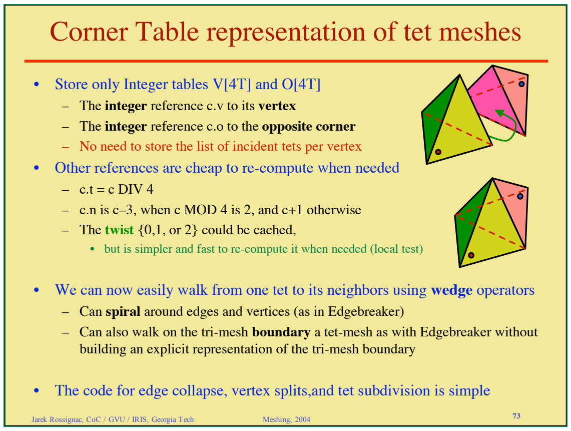
- c.o is the integer index to the opposite corner
- The **twist** {0,1, or 2} can be cached,
  - but is simple and fast to re-compute

V	O
1	6
2	x
4	x
3	x
2	x
3	x
4	x
6	1
4	8
.	.



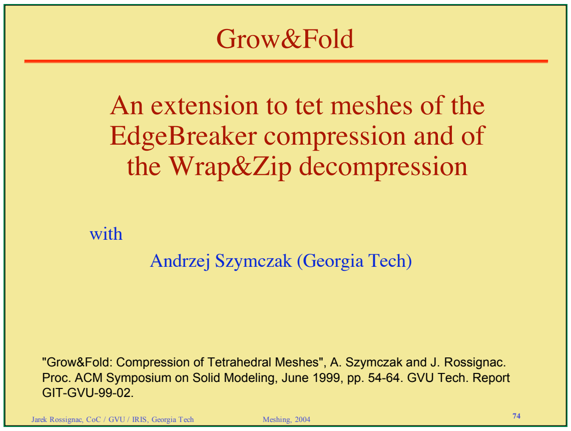
Wedge operators

- A wedge w
  - ordered pair (a,b) of corners
- Operators derived
  - From V
    - 4x4 look-up table
  - From V and O
    - Using twist



Corner Table representation of tet meshes

- Store only Integer tables V[4T] and O[4T]
  - The **integer** reference c.v to its **vertex**
  - The **integer** reference c.o to the **opposite corner**
  - No need to store the list of incident tets per vertex
- Other references are cheap to re-compute when needed
  - c.t = c DIV 4
  - c.n is c–3, when c MOD 4 is 2, and c+1 otherwise
  - The **twist** {0,1, or 2} could be cached,
    - but is simpler and fast to re-compute it when needed (local test)
- We can now easily walk from one tet to its neighbors using **wedge** operators
  - Can **spiral** around edges and vertices (as in Edgebreaker)
  - Can also walk on the tri-mesh **boundary** a tet-mesh as with Edgebreaker without building an explicit representation of the tri-mesh boundary
- The code for edge collapse, vertex splits,and tet subdivision is simple



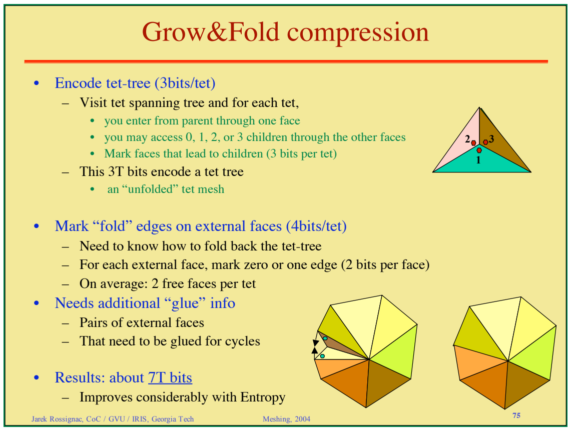
Grow&Fold

An extension to tet meshes of the EdgeBreaker compression and of the Wrap&Zip decompression

with

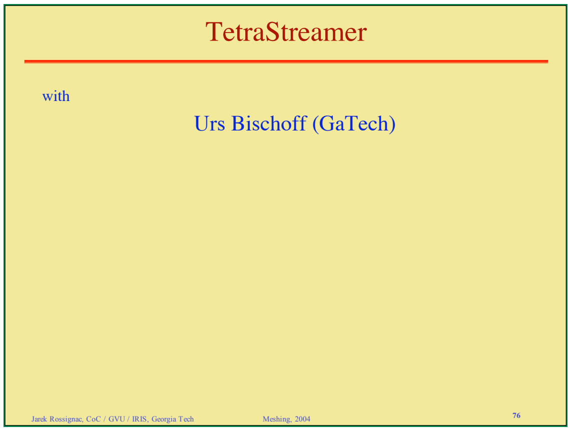
Andrzej Szymczak (Georgia Tech)

"Grow&Fold: Compression of Tetrahedral Meshes", A. Szymczak and J. Rossignac. Proc. ACM Symposium on Solid Modeling, June 1999, pp. 54-64. GVU Tech. Report GIT-GVU-99-02.



Grow&Fold compression

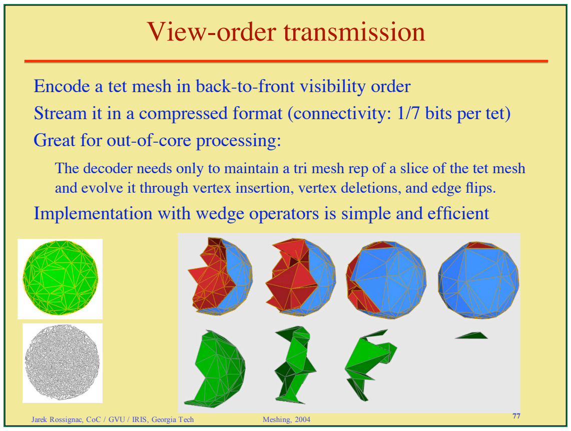
- Encode tet-tree (3bits/tet)
  - Visit tet spanning tree and for each tet,
    - you enter from parent through one face
    - you may access 0, 1, 2, or 3 children through the other faces
    - Mark faces that lead to children (3 bits per tet)
  - This 3T bits encode a tet tree
    - an "unfolded" tet mesh
- Mark "fold" edges on external faces (4bits/tet)
  - Need to know how to fold back the tet-tree
  - For each external face, mark zero or one edge (2 bits per face)
  - On average: 2 free faces per tet
- Needs additional "glue" info
  - Pairs of external faces
  - That need to be glued for cycles
- Results: about 7T bits
  - Improves considerably with Entropy



TetraStreamer

with

Urs Bischoff (GaTech)



View-order transmission

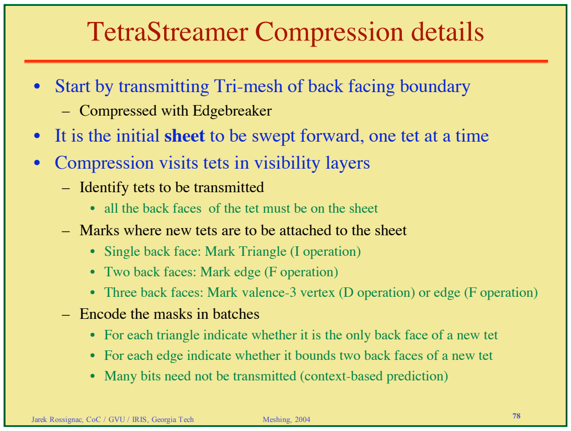
Encode a tet mesh in back-to-front visibility order

Stream it in a compressed format (connectivity: 1/7 bits per tet)

Great for out-of-core processing:

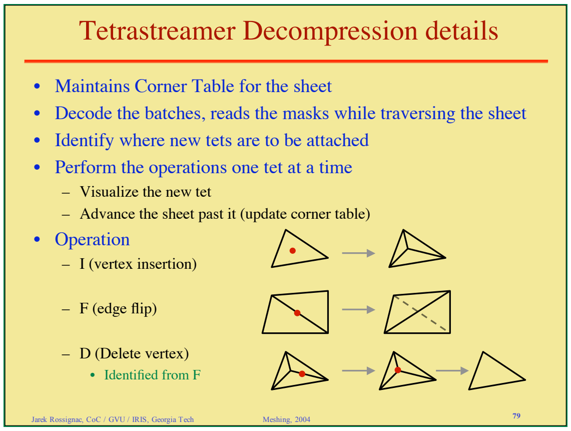
The decoder needs only to maintain a tri mesh rep of a slice of the tet mesh and evolve it through vertex insertion, vertex deletions, and edge flips.

Implementation with wedge operators is simple and efficient



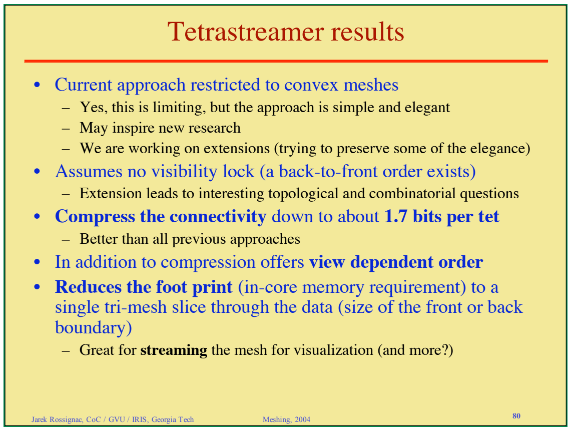
TetraStreamer Compression details

- Start by transmitting Tri-mesh of back facing boundary
  - Compressed with Edgebreaker
- It is the initial **sheet** to be swept forward, one tet at a time
- Compression visits tets in visibility layers
  - Identify tets to be transmitted
    - all the back faces of the tet must be on the sheet
  - Marks where new tets are to be attached to the sheet
    - Single back face: Mark Triangle (I operation)
    - Two back faces: Mark edge (F operation)
    - Three back faces: Mark valence-3 vertex (D operation) or edge (F operation)
  - Encode the masks in batches
    - For each triangle indicate whether it is the only back face of a new tet
    - For each edge indicate whether it bounds two back faces of a new tet
    - Many bits need not be transmitted (context-based prediction)



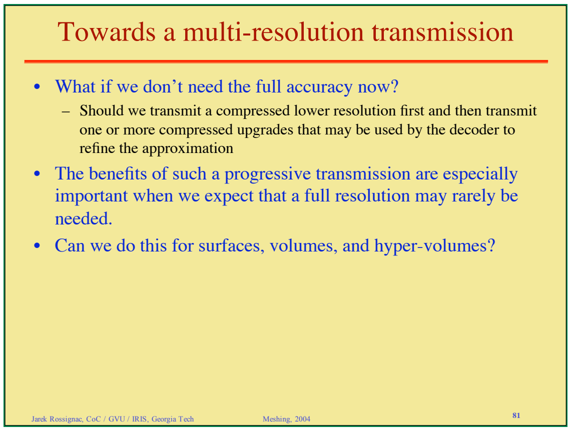
Tetrastreamer Decompression details

- Maintains Corner Table for the sheet
- Decode the batches, reads the masks while traversing the sheet
- Identify where new tets are to be attached
- Perform the operations one tet at a time
  - Visualize the new tet
  - Advance the sheet past it (update corner table)
- Operation
  - I (vertex insertion)
  - F (edge flip)
  - D (Delete vertex)
    - Identified from F



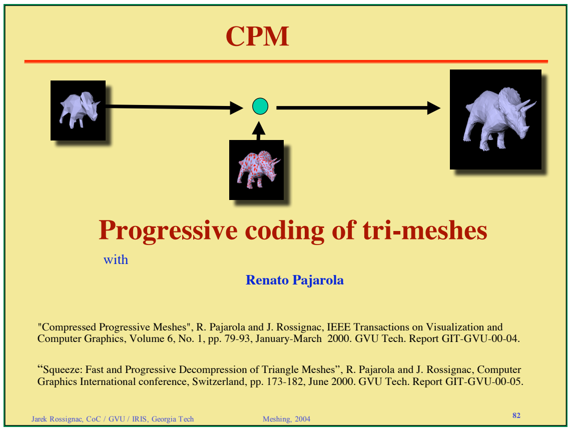
Tetrastreamer results

- Current approach restricted to convex meshes
  - Yes, this is limiting, but the approach is simple and elegant
  - May inspire new research
  - We are working on extensions (trying to preserve some of the elegance)
- Assumes no visibility lock (a back-to-front order exists)
  - Extension leads to interesting topological and combinatorial questions
- **Compress the connectivity** down to about **1.7 bits per tet**
  - Better than all previous approaches
- In addition to compression offers **view dependent order**
- **Reduces the foot print** (in-core memory requirement) to a single tri-mesh slice through the data (size of the front or back boundary)
  - Great for **streaming** the mesh for visualization (and more?)



Towards a multi-resolution transmission

- What if we don't need the full accuracy now?
  - Should we transmit a compressed lower resolution first and then transmit one or more compressed upgrades that may be used by the decoder to refine the approximation
- The benefits of such a progressive transmission are especially important when we expect that a full resolution may rarely be needed.
- Can we do this for surfaces, volumes, and hyper-volumes?



CPM

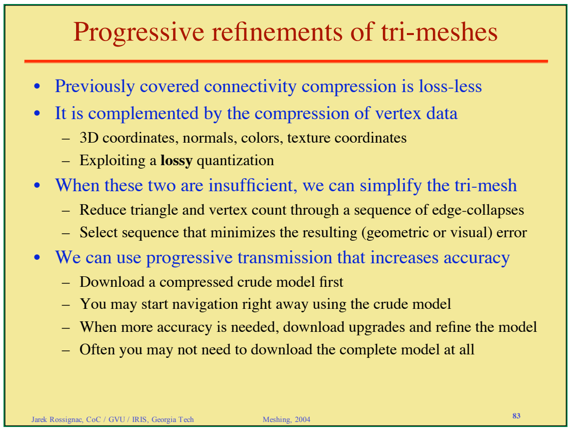
Progressive coding of tri-meshes

with

Renato Pajarola

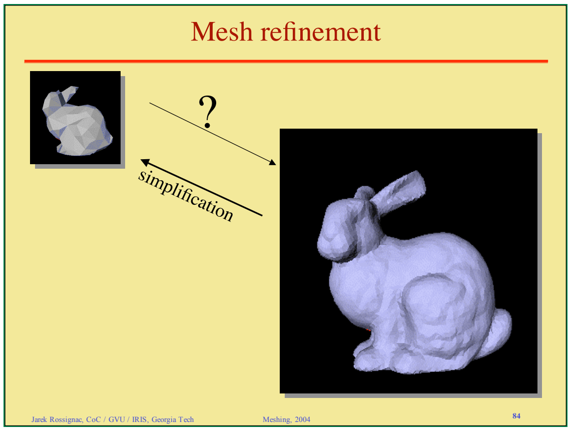
"Compressed Progressive Meshes", R. Pajarola and J. Rossignac, IEEE Transactions on Visualization and Computer Graphics, Volume 6, No. 1, pp. 79-93, January-March 2000. GVU Tech. Report GIT-GVU-00-04.

"Squeeze: Fast and Progressive Decompression of Triangle Meshes", R. Pajarola and J. Rossignac, Computer Graphics International conference, Switzerland, pp. 173-182, June 2000. GVU Tech. Report GIT-GVU-00-05.

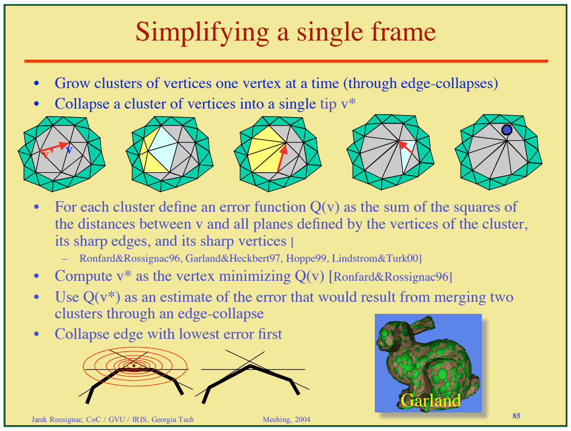


Progressive refinements of tri-meshes

- Previously covered connectivity compression is loss-less
- It is complemented by the compression of vertex data
  - 3D coordinates, normals, colors, texture coordinates
  - Exploiting a **lossy** quantization
- When these two are insufficient, we can simplify the tri-mesh
  - Reduce triangle and vertex count through a sequence of edge-collapses
  - Select sequence that minimizes the resulting (geometric or visual) error
- We can use progressive transmission that increases accuracy
  - Download a compressed crude model first
  - You may start navigation right away using the crude model
  - When more accuracy is needed, download upgrades and refine the model
  - Often you may not need to download the complete model at all



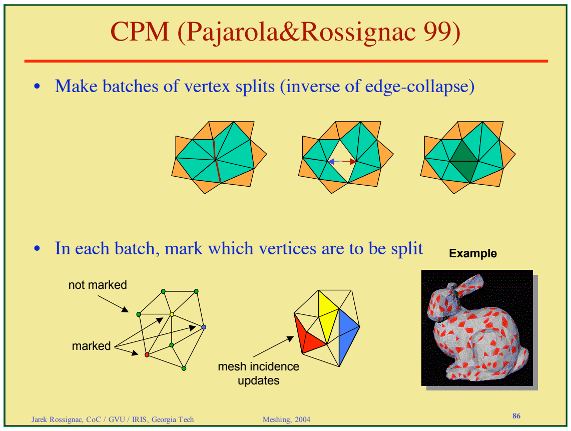
Mesh refinement



Simplifying a single frame

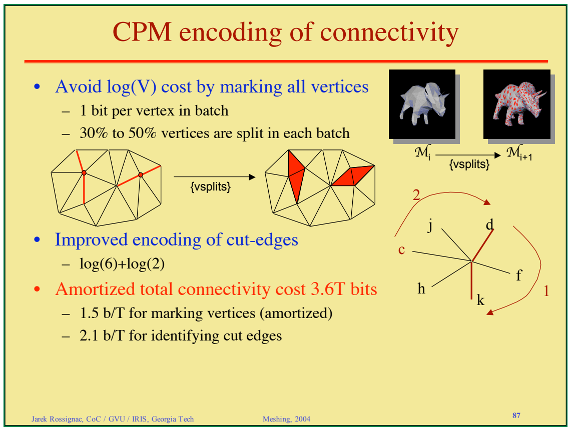
- Grow clusters of vertices one vertex at a time (through edge-collapses)
- Collapse a cluster of vertices into a single tip v*

- For each cluster define an error function Q(v) as the sum of the squares of the distances between v and all planes defined by the vertices of the cluster, its sharp edges, and its sharp vertices [
  - Ronfard&Rossignac96, Garland&Heckbert97, Hoppe99, Lindstrom&Turk00]
- Compute v* as the vertex minimizing Q(v) [Ronfard&Rossignac96]
- Use Q(v*) as an estimate of the error that would result from merging two clusters through an edge-collapse
- Collapse edge with lowest error first



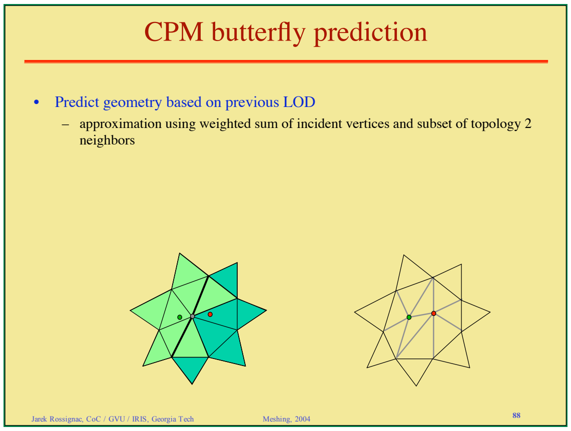
CPM (Pajarola&Rossignac 99)

- Make batches of vertex splits (inverse of edge-collapse)
- In each batch, mark which vertices are to be split



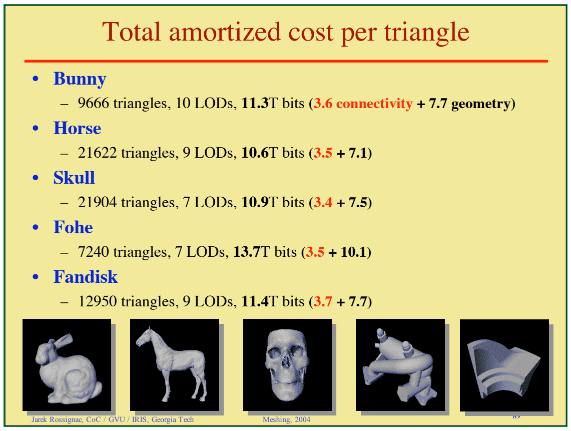
CPM encoding of connectivity

- Avoid log(V) cost by marking all vertices
  - 1 bit per vertex in batch
  - 30% to 50% vertices are split in each batch
- Improved encoding of cut-edges
  - log(6)+log(2)
- Amortized total connectivity cost 3.6T bits
  - 1.5 b/T for marking vertices (amortized)
  - 2.1 b/T for identifying cut edges



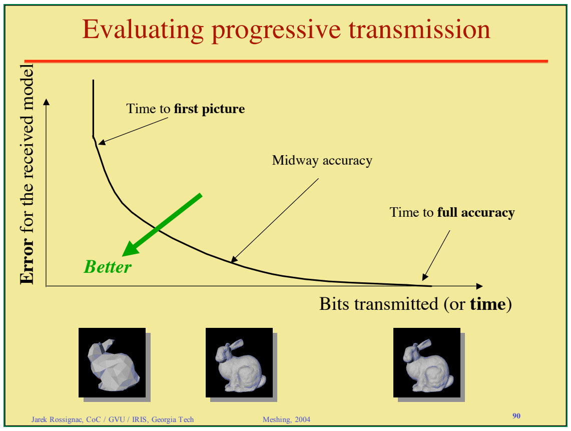
CPM butterfly prediction

- Predict geometry based on previous LOD
  - approximation using weighted sum of incident vertices and subset of topology 2 neighbors

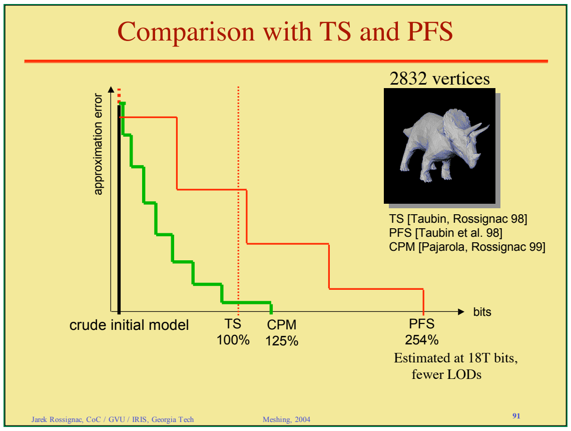


Total amortized cost per triangle

- **Bunny**
  - 9666 triangles, 10 LODs, **11.3**T bits (**3.6 connectivity** + **7.7 geometry**)
- **Horse**
  - 21622 triangles, 9 LODs, **10.6**T bits (**3.5** + **7.1**)
- **Skull**
  - 21904 triangles, 7 LODs, **10.9**T bits (**3.4** + **7.5**)
- **Fohe**
  - 7240 triangles, 7 LODs, **13.7**T bits (**3.5** + **10.1**)
- **Fandisk**
  - 12950 triangles, 9 LODs, **11.4**T bits (**3.7** + **7.7**)



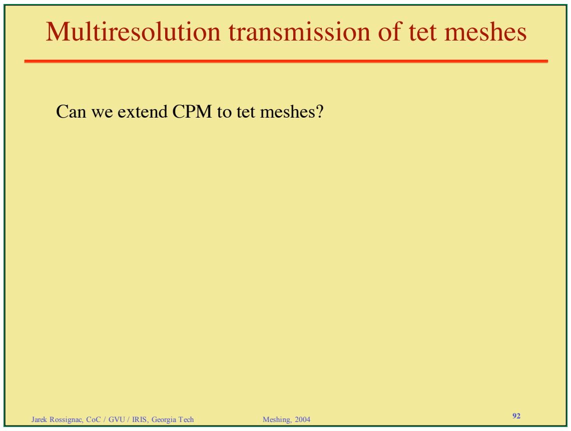
Evaluating progressive transmission



Comparison with TS and PFS

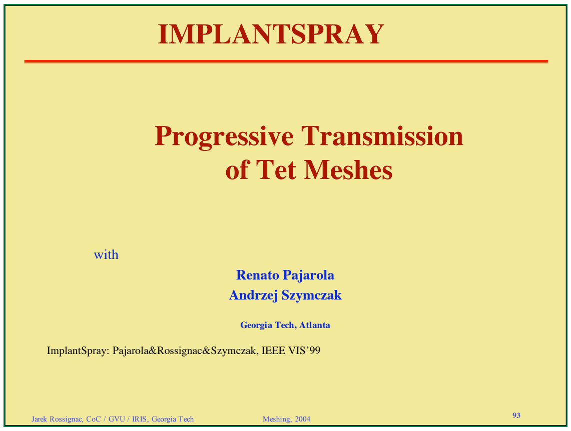
TS [Taubin, Rossignac 98]
PFS [Taubin et al. 98]
CPM [Pajarola, Rossignac 99]

Estimated at 18T bits,
fewer LODs



Multiresolution transmission of tet meshes

Can we extend CPM to tet meshes?



IMPLANTSPRAY

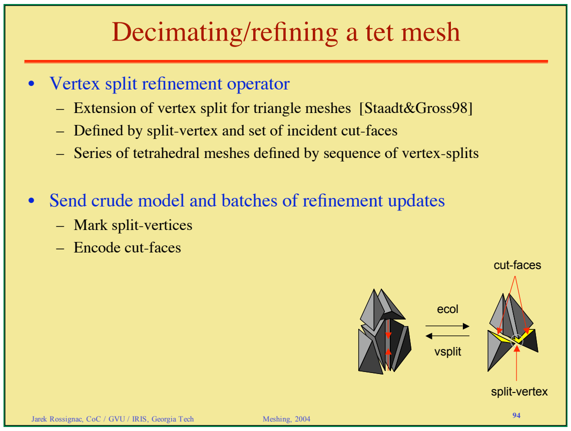
Progressive Transmission
of Tet Meshes

with

Renato Pajarola
Andrzej Szymczak

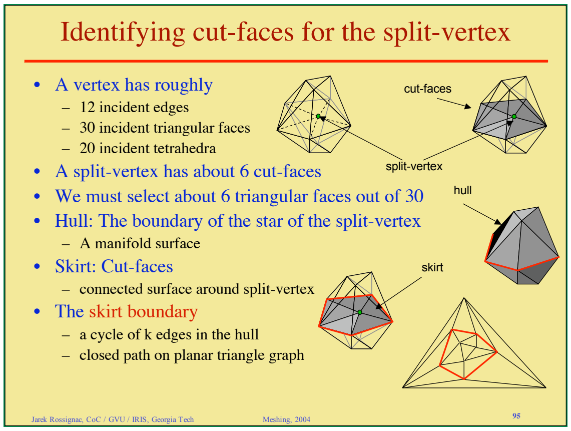
Georgia Tech, Atlanta

ImplantSpray: Pajarola&Rossignac&Szymczak, IEEE VIS'99



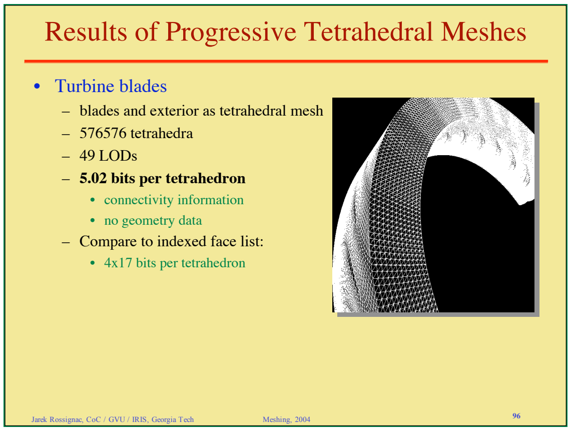
Decimating/refining a tet mesh

- Vertex split refinement operator
  - Extension of vertex split for triangle meshes [Staadt&Gross98]
  - Defined by split-vertex and set of incident cut-faces
  - Series of tetrahedral meshes defined by sequence of vertex-splits
- Send crude model and batches of refinement updates
  - Mark split-vertices
  - Encode cut-faces



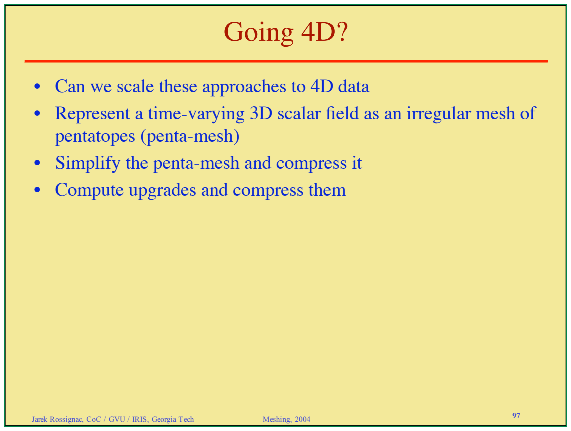
Identifying cut-faces for the split-vertex

- A vertex has roughly
  - 12 incident edges
  - 30 incident triangular faces
  - 20 incident tetrahedra
- A split-vertex has about 6 cut-faces
- We must select about 6 triangular faces out of 30
- Hull: The boundary of the star of the split-vertex
  - A manifold surface
- Skirt: Cut-faces
  - connected surface around split-vertex
- The skirt boundary
  - a cycle of k edges in the hull
  - closed path on planar triangle graph



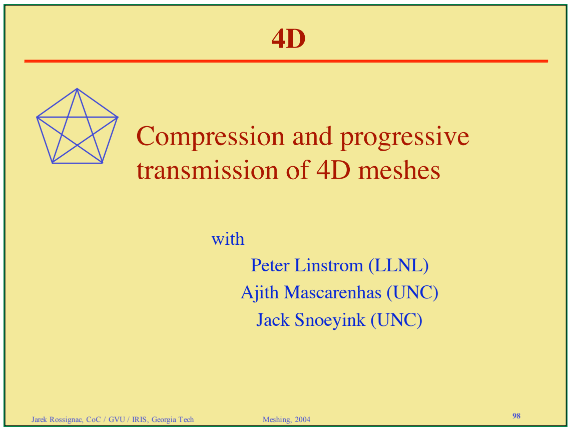
Results of Progressive Tetrahedral Meshes

- Turbine blades
  - blades and exterior as tetrahedral mesh
  - 576576 tetrahedra
  - 49 LODs
  - **5.02 bits per tetrahedron**
    - connectivity information
    - no geometry data
  - Compare to indexed face list:
    - 4x17 bits per tetrahedron



Going 4D?

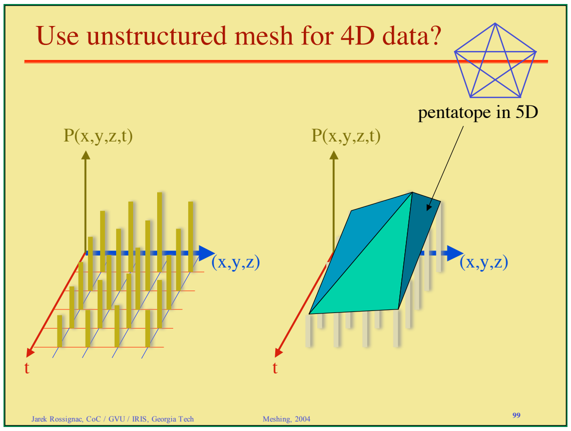
- Can we scale these approaches to 4D data
- Represent a time-varying 3D scalar field as an irregular mesh of pentatopes (penta-mesh)
- Simplify the penta-mesh and compress it
- Compute upgrades and compress them



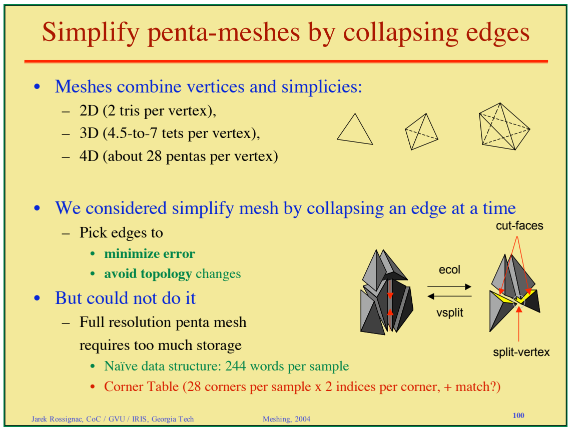
4D

Compression and progressive transmission of 4D meshes

with
Peter Linstrom (LLNL)
Ajith Mascarenhas (UNC)
Jack Snoeyink (UNC)

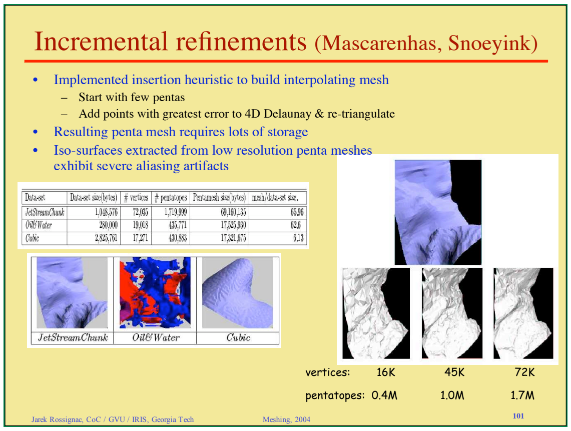


Use unstructured mesh for 4D data?



Simplify penta-meshes by collapsing edges

- Meshes combine vertices and simplicies:
  - 2D (2 tris per vertex),
  - 3D (4.5-to-7 tets per vertex),
  - 4D (about 28 pentas per vertex)
- We considered simplify mesh by collapsing an edge at a time
  - Pick edges to
    - **minimize error**
    - **avoid topology** changes
- But could not do it
  - Full resolution penta mesh requires too much storage
    - Naïve data structure: 244 words per sample
    - Corner Table (28 corners per sample x 2 indices per corner, + match?)

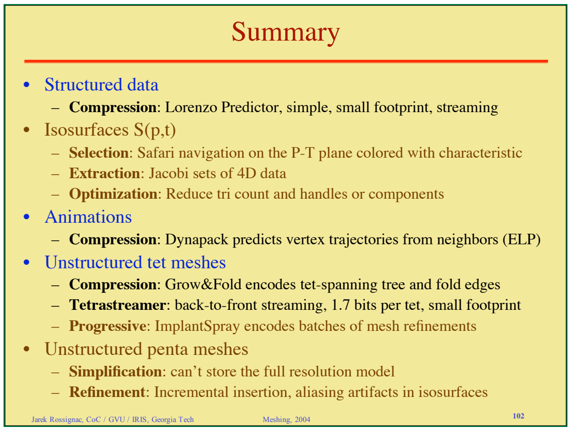


Incremental refinements (Mascarenhas, Snoeyink)

- Implemented insertion heuristic to build interpolating mesh
  - Start with few pentas
  - Add points with greatest error to 4D Delaunay & re-triangulate
- Resulting penta mesh requires lots of storage
- Iso-surfaces extracted from low resolution penta meshes exhibit severe aliasing artifacts

Data-set	Data-set size(bytes)	# vertices	# pentatopes	Pentamesh size(bytes)	mesh/data-set size.
JetStreamChunk	1,048,576	72,035	1,719,999	69,160,135	65.96
Oil&Water	280,000	19,018	435,771	17,525,930	62.6
Cubic	2,825,761	17,271	430,883	17,321,675	6.13

vertices:	16K	45K	72K
pentatopes:	0.4M	1.0M	1.7M



Summary

- Structured data
  - **Compression**: Lorenzo Predictor, simple, small footprint, streaming
- Isosurfaces S(p,t)
  - **Selection**: Safari navigation on the P-T plane colored with characteristic
  - **Extraction**: Jacobi sets of 4D data
  - **Optimization**: Reduce tri count and handles or components
- Animations
  - **Compression**: Dynapack predicts vertex trajectories from neighbors (ELP)
- Unstructured tet meshes
  - **Compression**: Grow&Fold encodes tet-spanning tree and fold edges
  - **Tetrastreamer**: back-to-front streaming, 1.7 bits per tet, small footprint
  - **Progressive**: ImplantSpray encodes batches of mesh refinements
- Unstructured penta meshes
  - **Simplification**: can't store the full resolution model
  - **Refinement**: Incremental insertion, aliasing artifacts in isosurfaces