Georgia Tech College of Computing — MAGIC lab — Modeling, Animation, Graphic, Interaction & Compression — GVU center georgia tech

Shape comparison, simplification, & compression: **Piecewise linear shapes**

Jarek Rossignac

Georgia Tech

Simplification

Compression

Complexity measures

How should we measure shape complexity?

- Number of components, handles and holes
- Stabbing number
- Area^3/volume^2
- Number of branches in skeleton
- …

Here we focus on:

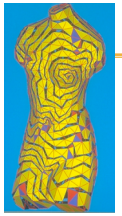
- Number of bits required to store or transmit the model
- Number of vertices
- Wiggles
- Sharp features

COMPRESSION

Reduce bit count

- To reduce storage
- To accelerate model transmission/downloading
- To reduce cost of transfer from memory to GPU

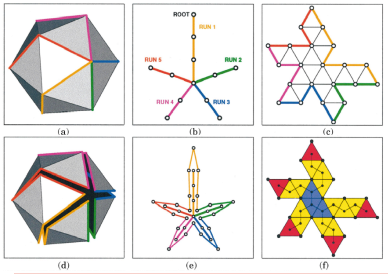
Topological Surgery (TS)

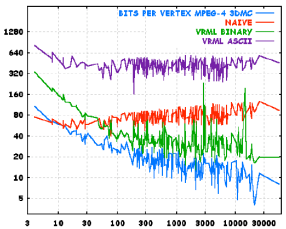


IBM'95, VRML, MPEG-4

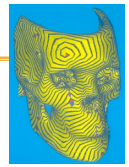
"Geometric compression through **Topological Surgery**," G. Taubin and J. Rossignac, ACM Transactions on Graphics, vol. 17, no. 2, pp. 84–115, 1998.

"Geometry coding and **VRML**," G. Taubin, W. Horn, F. Lazarus, and J. Rossignac, Proceeding of the IEEE, vol. 96, no. 6, pp. 1228–1243, June 1998.

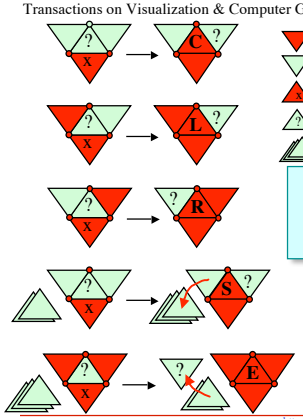




Edgebreaker (EB)



"Edgebreaker: Connectivity compression for triangle meshes," J. Rossignac, IEEE Transactions on Visualization & Computer Graphics, vol. 5, no. 1, pp. 47–61, 1999.

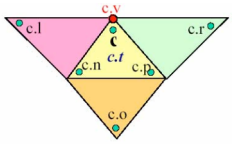


if tip vertex **not** marked **then** C
else if left neighbor marked
  then if right neighbor marked **then** E **else** L
  else if right neighbor marked **then** R **else** S

Encode sequence of codes
C: 0, **L**:110, **R**: 101, **S**:100, **E**:111
2T bits because |C|=V=T/2)
Entropy **0.9T** bits
and vertices
as encountered by C operations

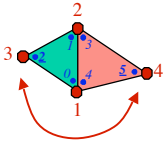
Corner Table: connectivity of Tri-Meshes

- 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



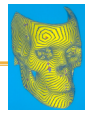
vertex 1	x	y	z
vertex 2	x	y	z
vertex 3	x	y	z
vertex 4	x	y	z

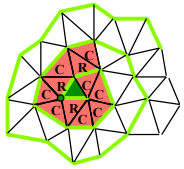
	V	O
Triangle 0 corner 0	1	7
Triangle 0 corner 1	2	8
Triangle 0 corner 2	3	5
Triangle 1 corner 3	2	9
Triangle 1 corner 4	1	6
Triangle 1 corner 5	4	2



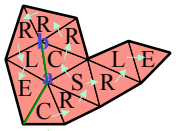
Edgebreaker compression algorithm

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





clers=CCCCRCCRCRC…



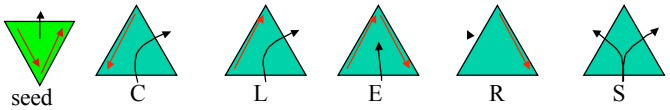
clers=…CRSRLECRRRLE

```
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

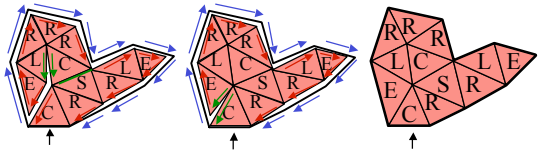
Wrap&Zip EB decompression (with Szymczak)

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

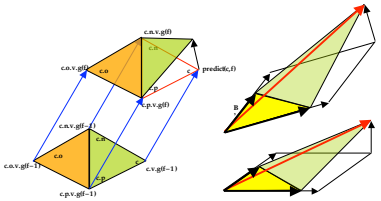


Linear time complexity. Zip only after L and E.

Dynapack

- How to compress the successive frames of a 3D animation?

Dynapack, compresses vertex coordinates of 3D animations to less than **2 bits** each



"Dynapack: Space-Time compression of the 3D animations of triangle meshes with fixed connectivity", L. Ibarria and J. Rossignac. ACM Symposium on Computer Animation (**SCA**), July 2003.

- Time only: predicts velocity of each vertex from its previous positions
- Space only: uses parallelogram to predict vertex location in each frame
- ELP: uses parallelogram to predict vertex velocity from velocities of neighbors
- Replica: uses coordinates of vertex relative to previous triangle in previous frame

Dynapack results

- Space-only predictor is poor (does not exploit frame-to-frame coherence)
- ELP and Replica are much better than the other two
- Replica is the best for coarser quantization

Head	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

Chicken	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

Head Shaping:

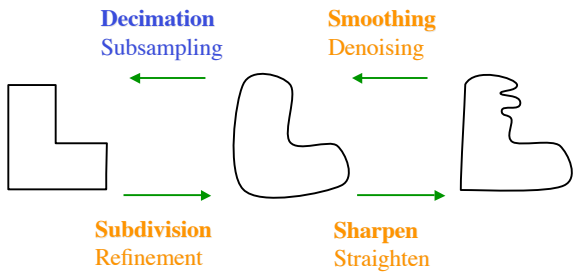
GaTech, Courtesy Ignacio Ilamas

Chicken Crossing:

© Microsoft, courtesy of John Snyder

•400 Frames
•31 connected components
•3030 vertices
•5664 triangles

DIFFERENT SIMPLIFICATIONS



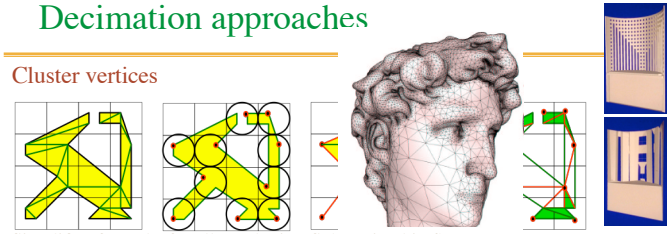
DECIMATION

Reduce vertex count

- To accelerate rendering
- To accelerate collision processing
- To simplify FEM generation and analysis
- To improve compression
- To simplify human-shape interaction

Decimation approaches

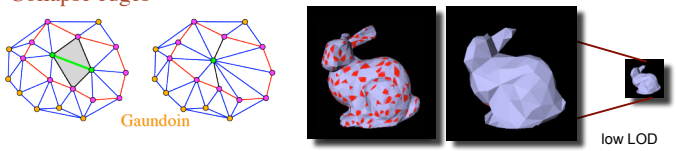
Cluster vertices



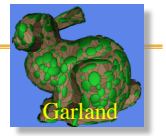
Simplifies & works for all topologies. Sub-optimal in flat areas

Multi-resolution 3D approximations for rendering complex scenes. Jarek Rossignac and Paul Borrel. In *Geometric Modeling in Computer Graphics*, pp. 455-465, Springer, 1993.

Collapse edges

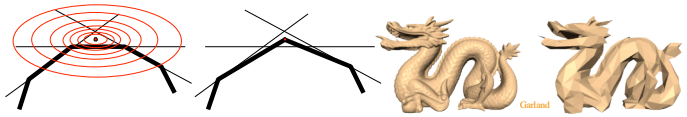


Estimate decimation error



- **Volume** of the **gap** between (XOR) old and new shapes
- Exact **Hausdorff distance** between old and new triangles
  - expensive to compute precisely, and not very reliable
- **Max of square distance** from v to planes of incident triangles
  - Conservative (guaranteed) bound on error
  - Must keep list of all planes supporting triangles incident on contracted edges

    "Full-range approximations of triangulated polyhedra", Remi Ronfard and Jarek Rossignac, Eurographics. **Computer Graphics Forum,** pp. C-67, Vol. 15, No. 3, August 1996.
- **Sum**, instead of max, (Heckbert&Garland)
  - Much faster than max, but not a bound on error
    - L_2 norm, needs only add 4x4 matrices when clusters are merged
    - leads to closed form solution for computing the optimal V' (in least square sense)



RESEAMPING

What if you do not need to preserve the exact model?

- Imprecise vertex locations (on or near the surface)
- Different connectivity
- Simpler topology

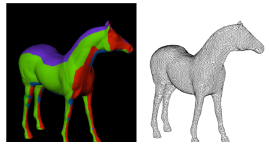
- Resample for better meshes
  - FEM
  - Shading
  - Morphing

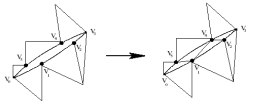
- Resampling for better compression
  - One parameter per sample (**normal** displacement only)
    - Want most vertices to be regular elevation over 2D grid (PRM)
    - Want mostly triangles to be isosceles (SwingWrapper)

Piecewise Regular Meshes (PRM)

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

- Split surface in 3 sets by orientation
- Resample each set on a regular grid
- Merge sets and fill topological cracks
- Encode with Edgebreaker
- Compress with range coder (2 char context)
- Parallelogram prediction (x,y) & z



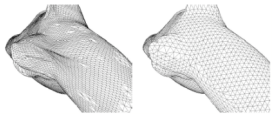


Compression results:

1T bits: connectivity & geometry (89%)

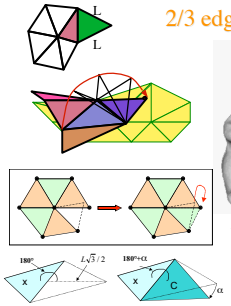
12-bit quantization

error <0.02% of radius of bounding ball (Metro)

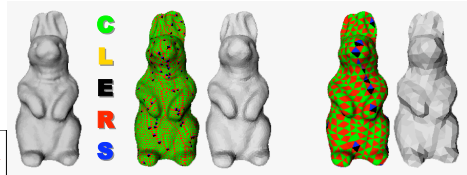


SwingWrapper

M. Attene, B. Falcidieno, M. Spagnuolo and J. Rossignac "SwingWrapper: Retiling Triangle Meshes for Better Compression," ACM Transactions in Graphics vol. 22, no. 4, pp. 982-996, 2003.

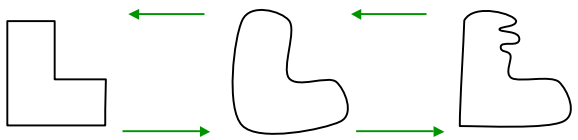


2/3 edges have length L (selected by the user).



134,074T	*13,642T, 3.5Tb*	*1505T, 5.2Tb*
WRL=4,100,000B	*678-to-1 compression*	*4000-to-1 compression*
	L_2 error 0.007%	*L_2 error 0.15%*

DIFFERENT SIMPLIFICATIONS



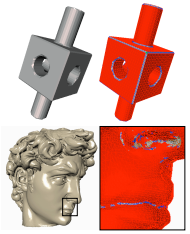
Decimation
Subsampling

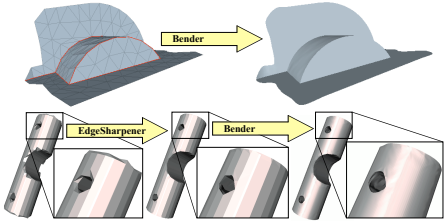
Smoothing
Denoising

Subdivision
Refinement

Sharpen
Straighten

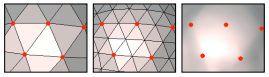
Sharpen&Bend

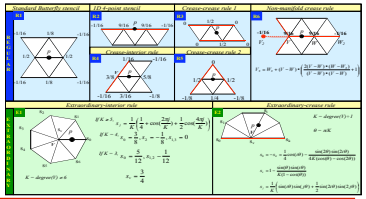




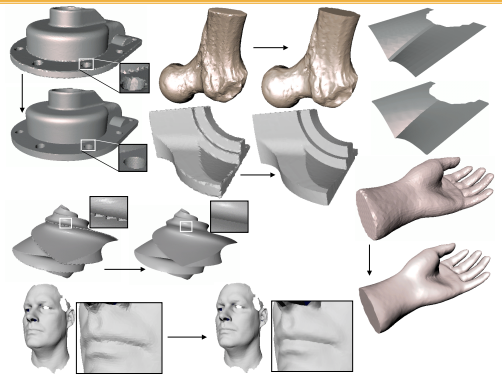
Butterfly subdivision for smooth parts

New subdivision mask to preserve sharpness of features





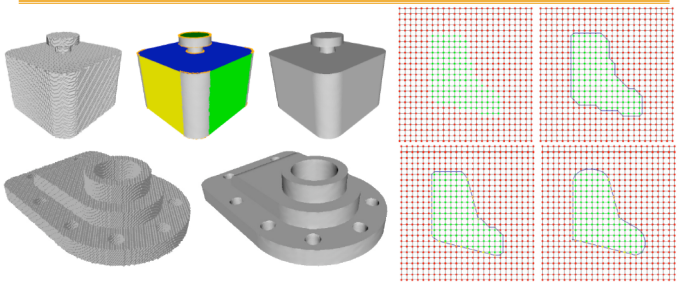
Sharpen&Bend examples



M. Attene, B. Falcidieno, J. Rossignac and M. Spagnuolo "**Edge-Sharpener**: A geometric filter for recovering sharp features in uniform triangulations," Eurographics Symposium on Geometry Processing 2003.

M. Attene, B. Falcidino, M. Spagnuolo, J. Rossignac, "**Sharpen&Bend**: Recovering curved edges in triangle meshes produced by feature-insensitive sampling", IEEE Transactions on Visualization and Computer Graphics, vol 11, no 2, pp 181-192, March/April 2005.

Pressing

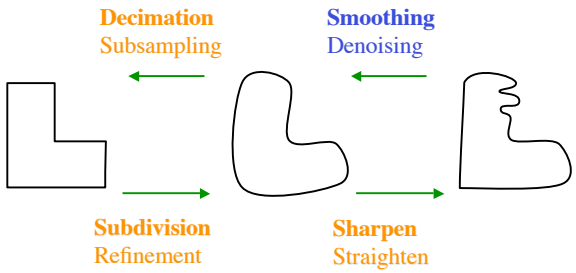


Recover planes, sharp features, and smooth blending surfaces from crude binary voxel data.

"Computing Maximal Tiles and Applications to Impostor-Based Simplification", Andujar, Brunet, Chica, Rossignac, Navazo, Vinacua. Eurographics, 2004.

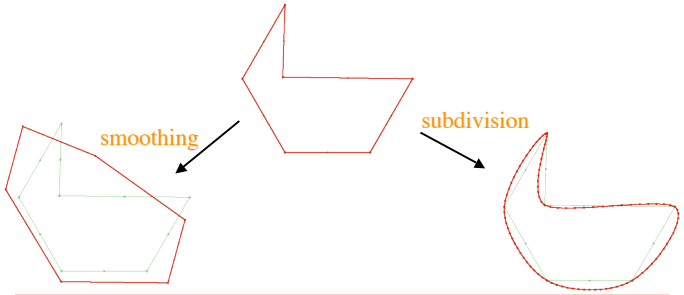
"Pressing: Smooth Isosurfaces with Flats from Binary Grids". A. Chica, J. Williams, C. Andujar, P. Brunet, I. Navazo, J. Rossignac, A.Vinacua, 2006.

DIFFERENT SIMPLIFICATIONS



Smoothing vs. subdivision

- **Smoothing** should be performed when the desired curve has been densely **sampled** but the resulting polyloop has **undesired sharp features** (acquisition noise, too many details, sharp changes in direction or curvature)
- **Subdivision** refines the polyloop (**adds more vertices**). It should be performed when the polyloop is a **crude** outline of the desired curve.



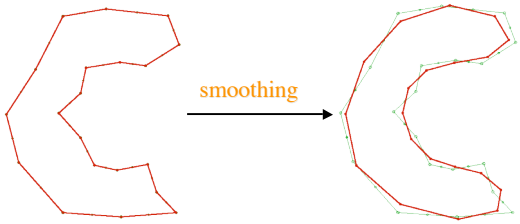
SMOOTHING

Adjust vertex position to eliminate sharp features

Try to avoid altering the shape where it is already smooth

- To eliminate acquisition errors or sampling artifacts
- To improve appearance

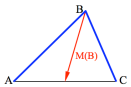
Move vertices towards average of their neighbors



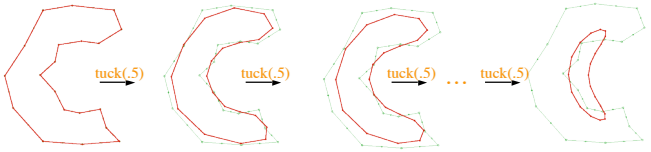
Keep tucking

tuck(s) moves the vertices by executing the following two steps:

- For each vertex B in {..A,B,C..} compute M(B)=(BA+BC)/2
- For each vertex B, move B to B+sM(B)

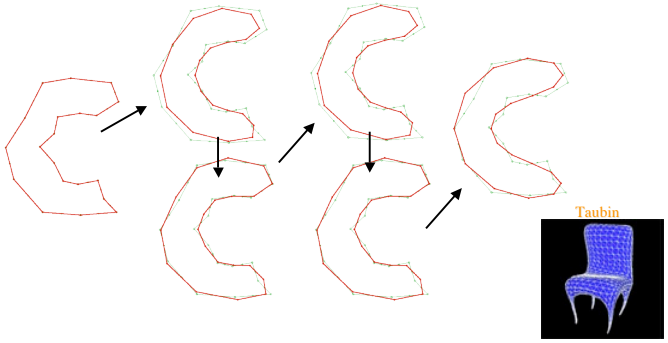


Repeating tuck(s) tends to eliminate sharp features, but **shrinks** the curve

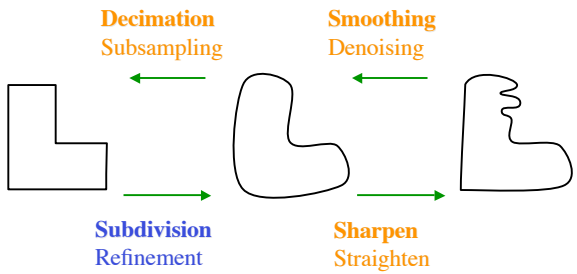


Avoid shrinking: **Tuck-tuck**

- Alternate **tuck(s)**, which shrinks, and **tuck(–s)**, which expands
  Smoothing: untuck cannot recover the sharp features lost during the tuck



DIFFERENT SIMPLIFICATIONS



SUBDIVISION

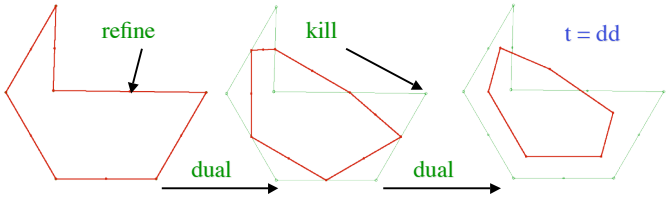
Refine the sampling automatically, converging to a smooth(er) shape

- Allow the designer to specify shmooth shapes using few control vertices
- Interpolates sparse sampling

Polyloop operators

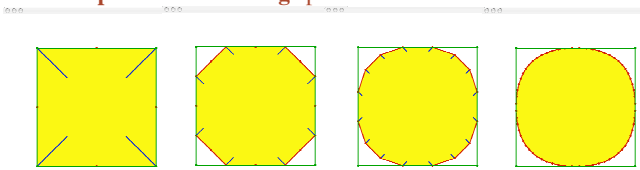
We will use the following operators:

- r (*refine*) introduces a new vertex in the middle of each edge
- k (*kill*) deletes the old vertices (those present before r)
- d (*dual*) performs (r, k)
- t_s (*parameterized tuck*) performs tuck(s)
- t (default *tuck*) performs tuck(1/2)



Quadratic uniform B-spline subdivision (r,d)

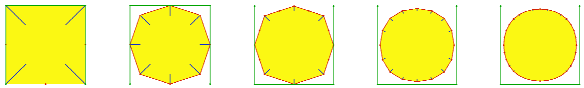
Repeating (r,d) converges to a **quadratic B-spline** curve that **interpolates** the **mid-edge** points



r rd rdrd rdrdrdrd

Cubic uniform B-spline subdivision (r,t)

Repeating (r,t), which is (r,d,d), converges to a **uniform cubic B-spline** curve that has second degree continuity (no jerk)



rt rtr rtrt rtrtrt

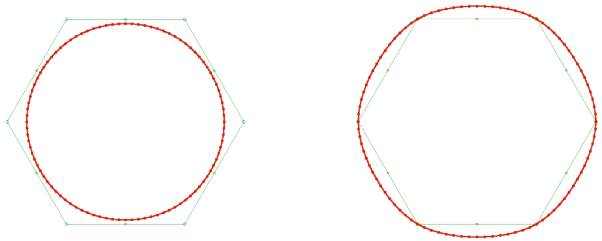
(r,d) gives you a **quadratic** (2nd degree) Bspline

(r,d,d) gives you a **cubic** (3rd degree) Bspline

Can you guess how to get a **quartic** (4th degree) Bspline?

B-spline curves are not interpolating

- The cubic B-spline is **approximating**
  - It does not go through the control vertices
- You may prefer the 4-point, which is **interpolating**



B-spline (approximating) 4-point (interpolating)

4-point subdivision

- We want to leave the original vertices where they are.
- Hence, we displace (bulge-out) each inserted mid-edge point C.
- By how much?

Fit a cubit $\mathbf{C}(t)=at^3+bt^2+ct+d$ through A, B, D, and E with 4 constraints

$B=\mathbf{C}(-1) = -a+b-c+d$

$D=\mathbf{C}(1) = a+b+c+d$

$A=\mathbf{C}(-3) = -27a+9b-3c+d$

$E=\mathbf{C}(3) = 27a+9b+3c+d$

Set $C=\mathbf{C}(0)=d$ and solve for d

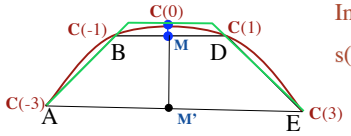
$M=(B+D)/2=b+d$

$M'=(A+E)/2=9b+d$

$M'-9M=-8d$

$8M+M-M'=8d$

$C=M+M'M/8$

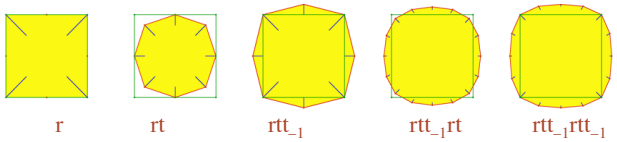


Implement as:

s(s(A,9/8,B) ,.5, s(E,9/8,D))

where
s(A,v,B)=A+vAB

4-point = (r,t,t_{-1})



Refine (r) puts a mid-edge point C at M.

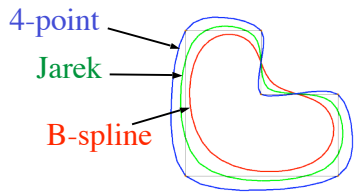
(t, t_{-1}) produces the cubic-fit bulge M'M/8

Verify by inspection…

4-pt = refine + tuck-tuck smoothing

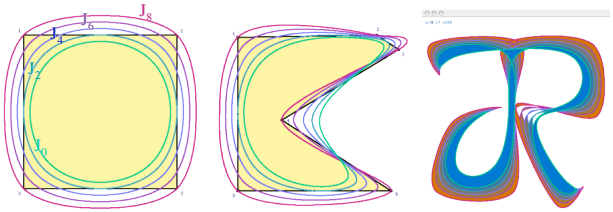
Jarek's compromise

- B-spline tucks-in, and thus shrinks a convex shape
- 4-points bulges the edges out, and hence grows a convex shape
- Both are relatively far from the initial control polygon
- We may want a compromise
  - Closer to the polygon
  - Bulge less
  - Cut less

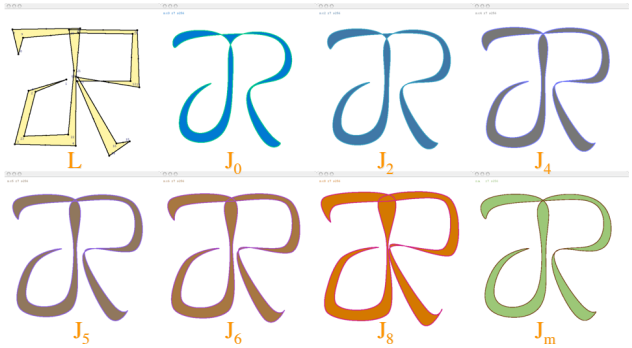


Generalized **Jarek** subdivision

- Jarek subdivision $J_s = (r , t , t_{-s/8})$
  - J_0 = cubic B-spline (C^2)
  - J_2 = through edge mid-points (C^2 smoother than quadratic B-spline)
  - J_4 = Jarek tends to preserve area (C^2)
  - J_5 = reduces discrepancy with previous level (C^2, good for LoD)
  - J_8 = 4-point interpolates (C^1)

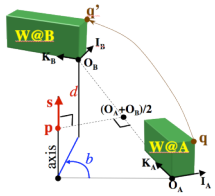


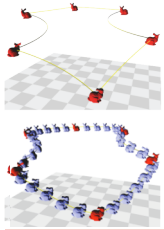
Comparing the aesthetics of J_s schemes

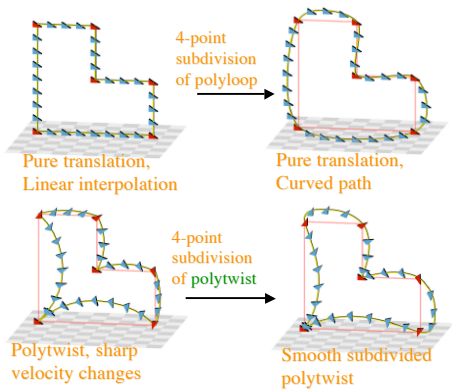


Demo curves

Smoothing polyscrew motions





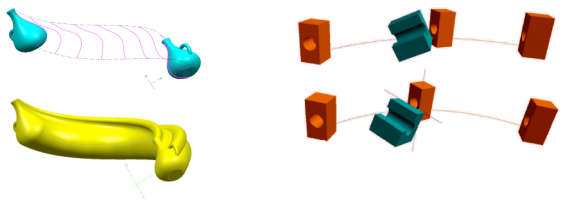


ScrewBender: Smoothing piecewise **helical** motions. A. Powell and J. Rossignac. IEEE Computer Graphics & Applications, 2007.

Demo Motion Smoothing

Applications to sweeps and collision

Arbitrary motions are approximated by smooth polyscrews, which simplify the calculation of boundaries swept by curved shapes and of collisions between polyhedra.

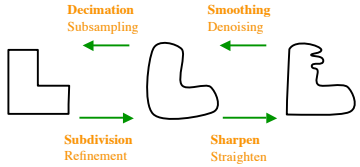


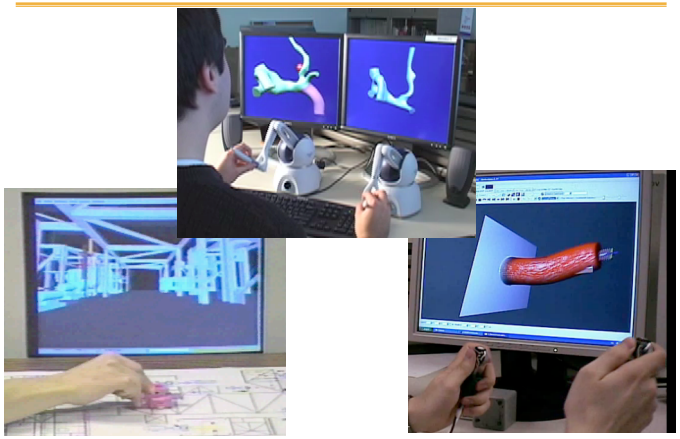
Boundary of the volume swept by a free-form solid in **screw** motion. J. Rossignac, J.J. Kim, S.C. Song, K.C. Suh, C.B. Joung. GIT-GVU-06-19.

Collision prediction for polyhedra under **screw** motions. ByungMoon Kim and Jarek Rossignac, ACM Symposium in Solid Modeling and Applications, pp. 4-10, June 2003.

Summary

- Compression: Reduce bit-count
- Decimation: Reduce vertex count
- Resampling: More regular surface sampling
- Smoothing: Move samples to reduce high frequency features
- Subdivision: Insert samples (and move old ones)



All these techniques deal with sampling and assume a sampling-based representation. What about other representations?

Videos of Human-Shape Interaction