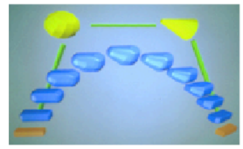
# Morphing



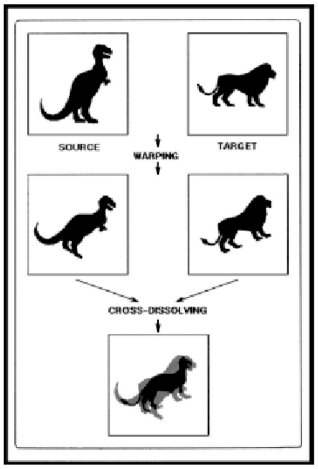
- Morphing vs deformation
- Image morphing
- Polygon-morphing
- Triangle-mesh morphing
- Morphing between animation frames (slow-motion)
- Mapping textures and parametric correspondence

Updated August 22, 2011

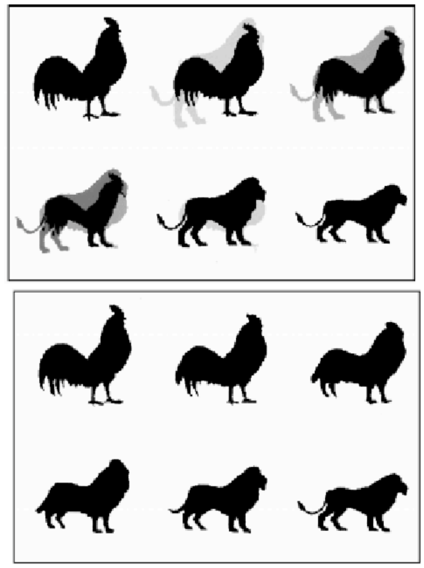
# Morphing vs deformation

- Deformation
  - Input: Initial shape A + warp (designed or simulated)
  - Result of computation: **Final shape**
  - Animation: Apply the warp progressively (bundle its parameters)
- Morphing
  - Input: Initial and final shapes A and B
  - Result of computation: **Warp**
  - Animation: Apply the warp progressively (bundle its parameters)
- Both can be combined

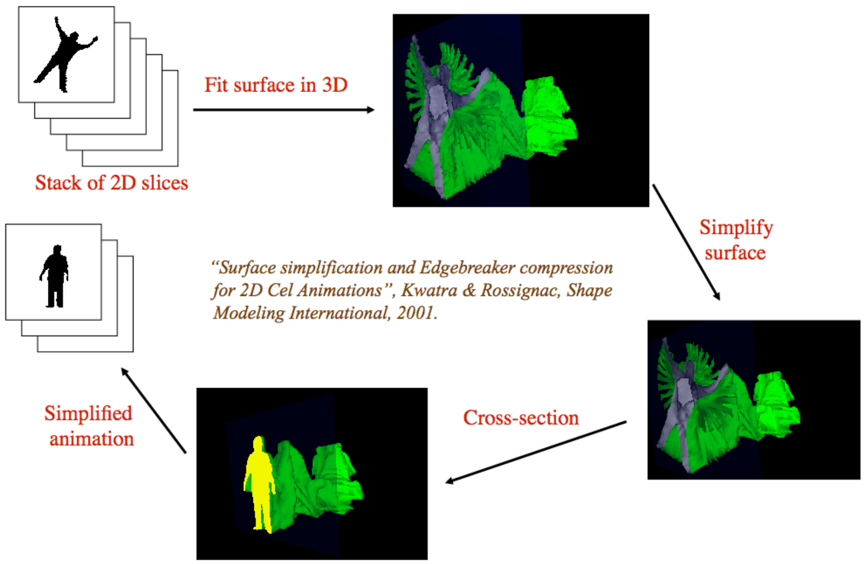
# Image morph: warp and blend




Cohen-Or et al.



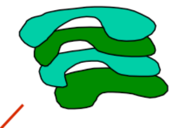
# Image morph by fitting a surface in 3D



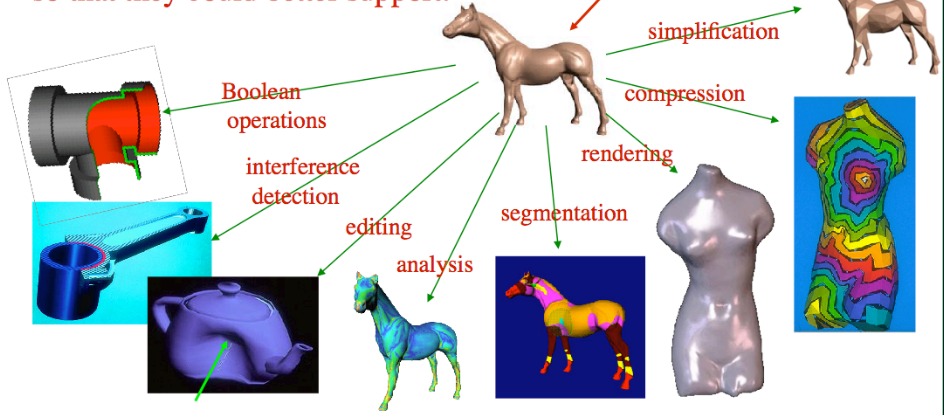

*"Surface simplification and Edgebreaker compression for 2D Cel Animations", Kwatra & Rossignac, Shape Modeling International, 2001.*

# 3D applications migrated from slices to 3D

Early 3D GIS, Medical, and M-CAD systems were based on stacks of **cross-sections**

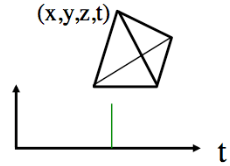


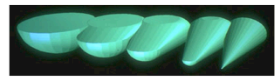
They migrated to **continuous** 3D models, so that they could better support:



# Need a similar migration for animations

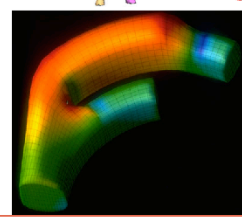
- Represent & slice **a** hyper-surface **in 4D**
  - Voxels or Tetrahedra in 4D:
    - (x,y,z,t)+connectivity?
  - Fast slice of hypercubes or tetrahedra
    - Addressed by Jack Snoeyink's CARGO project



- **Generate** interpolating **4D models**
  - 3D morph, fitting implicit hyper-surface

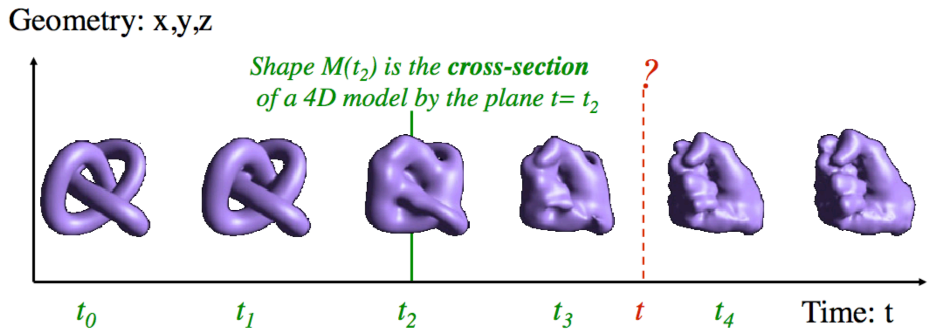

- **Use 4D model to build temporally** coherent segmentations **of the evolving shape into** features**?**
- **Use 4D model to build temporally** coherent parameterizations **of the evolving** features**?**



# A 4D model of the behavior of 3D shapes

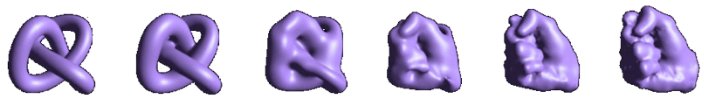
Many animation and simulation packages represent behavior as a series of independent **3D frames**

Yet, a continuous model is better suited for supporting slow-motion, geometric and topological **analysis**, and coherent **segmentation**, **texturing** and **visualization**

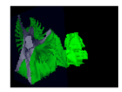


# How to generate a continuous 4D model?

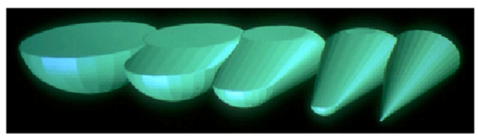
- Design by manipulating control points of B-spline S(u,v,t)
- Fit a hyper-surface to constraints



  – As we did by fitting a surface to a stack of contours

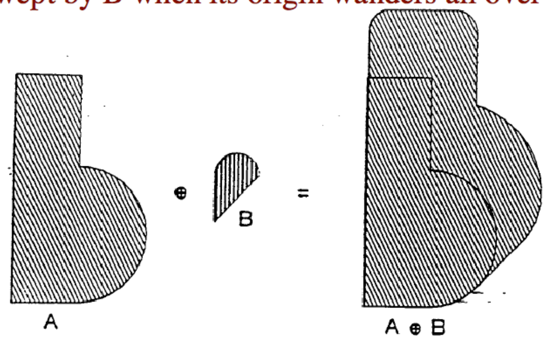


- Piecewise linear or polynomial morphs between 3D frames



# Minkowski sum

Region swept by B when its origin wanders all over A



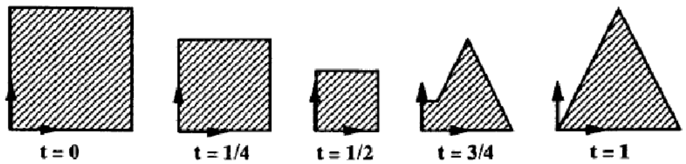

If B is small, then the result is similar to A (and vice versa)

Hence, we combine scaling with ⊕ : $(1-t)A \oplus tB$

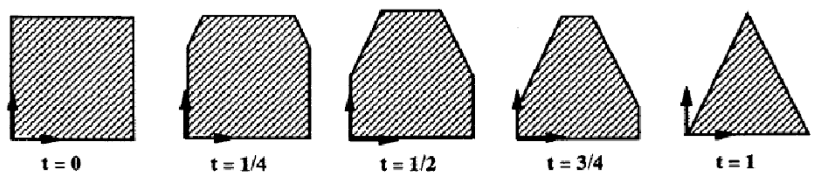
- When t=0, we obtain A. When t=1, we obtain B

The result is independent of the choice of origin

# Averaging superposition vs Minkowski



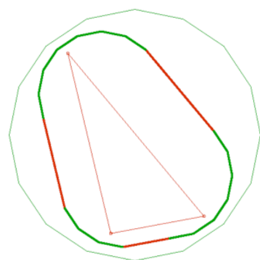

$(1 - t)*A \bigcup t*B$

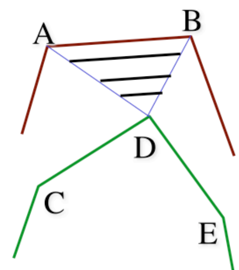



$(1 - t)*A \oplus t*B$

# 2D morph

- Each edge morphs to at least one vertex of the other shape
- Morph(A,B,D,t) produces an edge between (1–t)A+tD and (1 –t)B+tD
- Replace t by 1–t when morphing the edges of the other curve





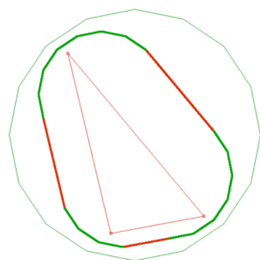
# Minkowski morph in 2D

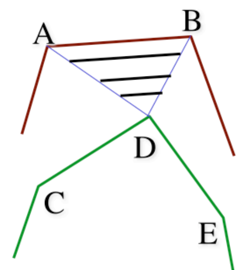
```
for (int i=0; i<nP; i++) {                                  // for all vertices A in P,
    for (int j=0; j<nQ; j++) {                                //for all vertices D in Q
     pt A = P[i]; pt B = P[in(i)];
    pt C = Q[jp(j)]; pt D = Q[j]; pt E = Q[jn(j)];
  if ( (dot(A.vecTo(B).left(),D.vecTo(C)) > 0)           // if AB.left * DC <0
      && (dot(C.vecTo(D).left(),C.vecTo(E)) > 0)       // skip concave vertices
      && (dot(A.vecTo(B).left(),D.vecTo(E)) > 0) )     // and if AB.left * DE <0
     morph(A,B,D,t);                                 // morph edge(A,B) of P with vertex D of Q
      };  };
```

…

swap P and Q

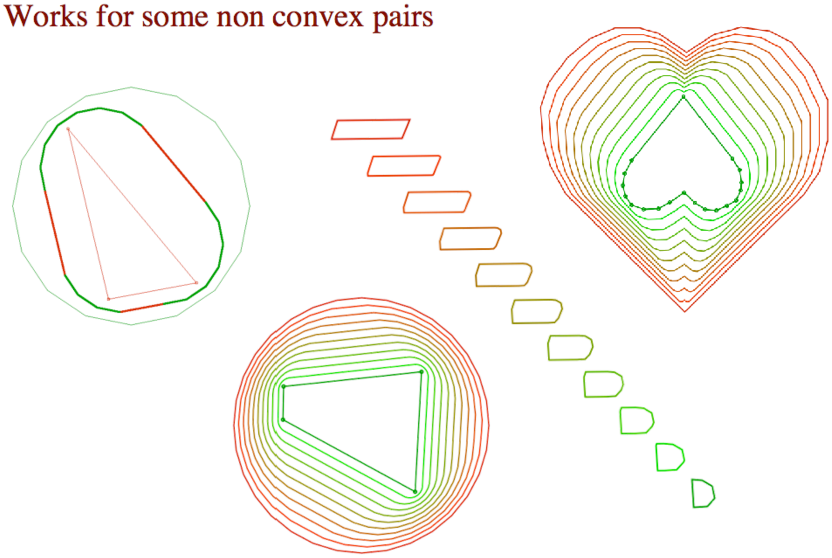
and redo





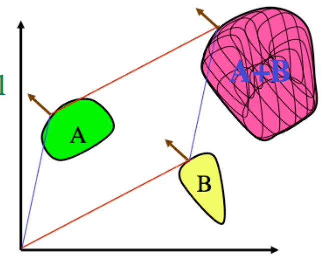
# Minkowski morph in 2D results

Works for some non convex pairs



# 3D morphing via Minkowski averaging

- A+B = {a+b: a∈A, b∈B}
  - Matches boundary points with same normal

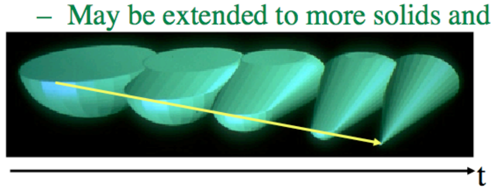



- M(t)=(1-t)A+tB

**"Solid-Interpolating Deformations: Construction and Animation of PIPs",**

**Kaul&Rossignac, C&G'92, 16(1)107-115.**

  - Constant connectivity, linear trajectory
  - Realtime animation
  - Vertices move on straight lines
  - May be extended to more solids and motions along Bezier curves

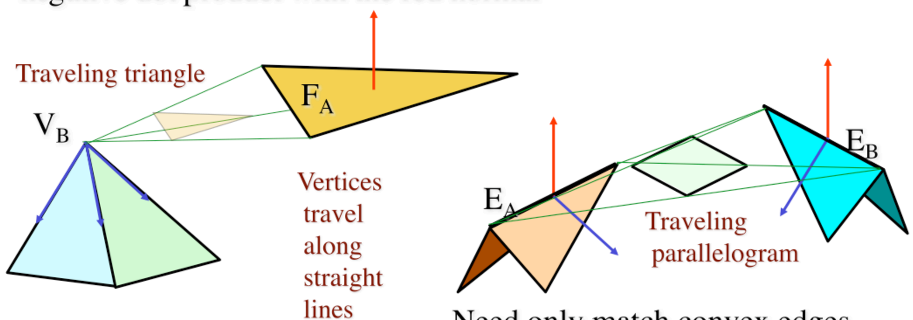


# The faces of a Minkowski morph

**Proposition 3** *Given any two arbitrary polyhedra A and B, the faces of* $A \oplus B$ *are subsets of the union of faces that can be written as* $V_A \oplus F_B$, $F_A \oplus V_B$ *or* $E_A \oplus E_B$. *where* $V_A$, $E_A$, $F_A$ *respectively denote a vertex, edge, face of A (similarly for B).*

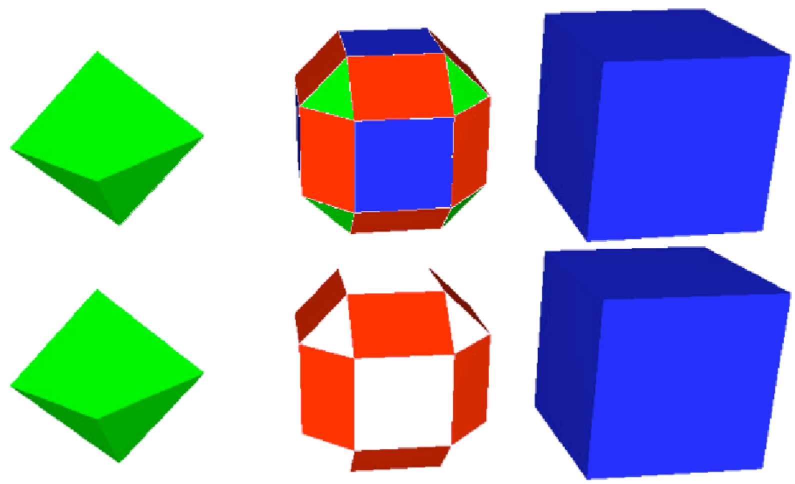
**Proposition 4** *Given two convex polyhedra A and B, any face of* $A \oplus B$ *may be written as* $V_A \oplus F_B$, $F_A \oplus V_B$ *or* $E_A \oplus E_B$,

Match face-to-vertex or edge-to-edge when each blue tangent has a negative dot product with the red normal



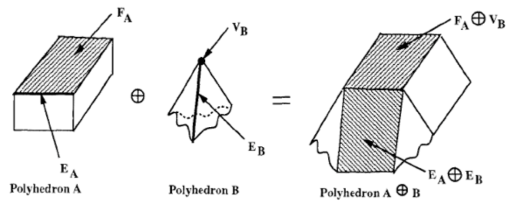

Need only match convex edges

# Why edge/edge matches are needed

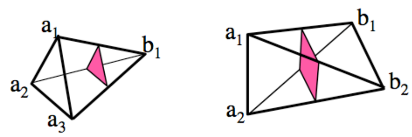


# Algorithm

- Establish mappings between pairs:
  Vertex-triangle, Edge-edge. Triangle-vertex



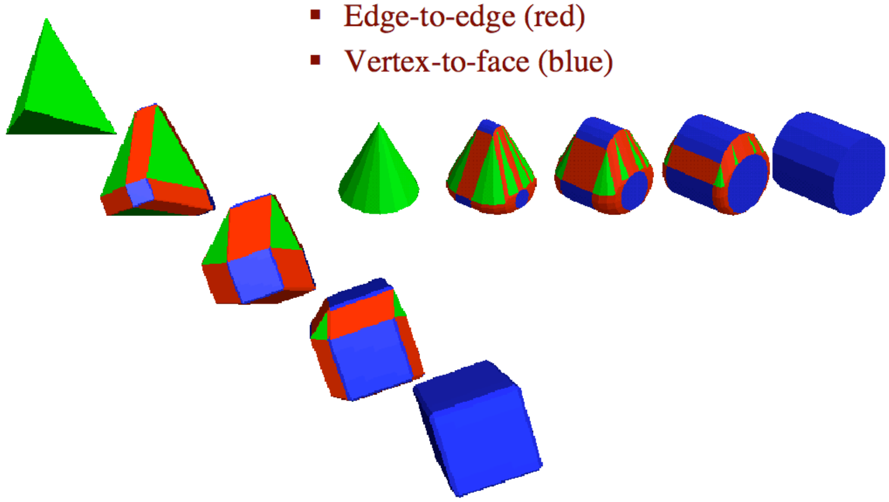

- Build a triangle list, whose vertices are each represented by a reference to a vertex of A and a reference to a vertex of B




- During animation, pre-compute $(1-t)a_i$ and $tb_j$, then render using +
  Each vertex of $M(t)=(1-t)A+tB$ linearly interpolates a vertex of A and a vertex of B

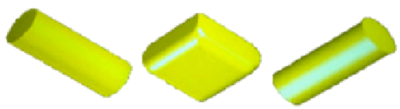
# Faces of a Minkowski morph

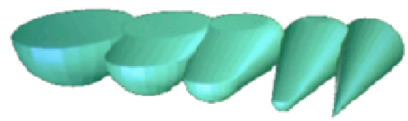
- Face-to-vertex (green)
- Edge-to-edge (red)
- Vertex-to-face (blue)



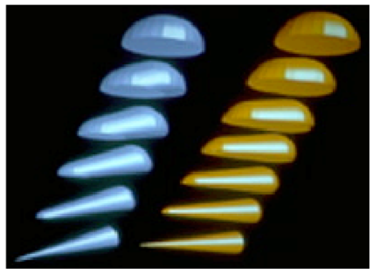
# Minkowski morphs for design

- Rounding
- Blending
- Creating new shapes



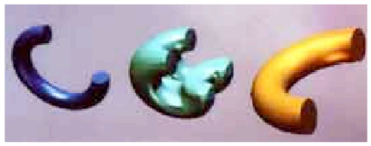


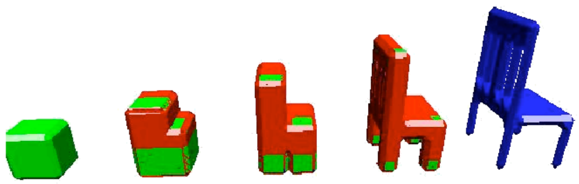
# Result depends on relative orientation

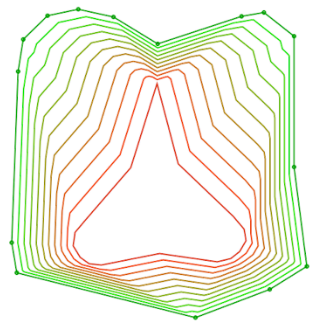


# Morphing non-convex shapes

**May** produce multiple branches and self-intersections

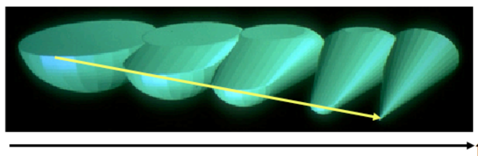


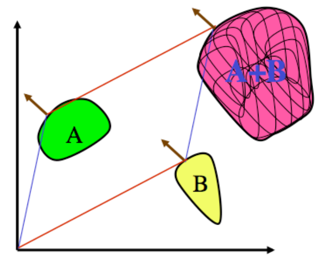




# Bezier morphs with 4 control shapes

- $A+B = \{a+b: a \in A, b \in B\}$
- $M(t)=(1-t)A+tB$

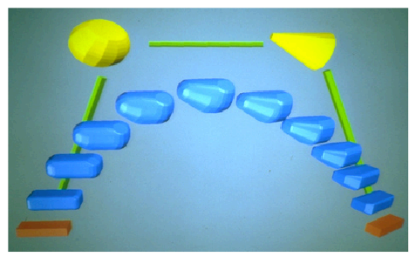





$M(t)= (1-t)((1-t)((1-t)A+tB)+t((1-t)B+tC))+t((1-t)((1-t)B+tC)+t((1-t)C+tD))$

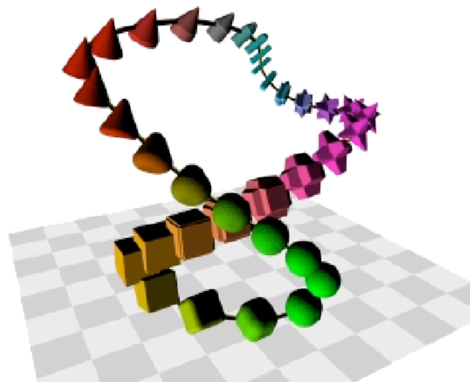
**"AGRELs and BIPs: Metamorphosis as a Bezier curve in the space of polyhedra", Rossignac&Kaul, CGForum'94, 13(3)179-184.**

– Vertices move on Bezier curves



# Combine morph with polyscrew motion

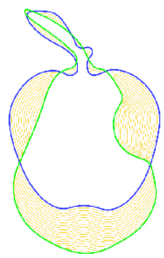
- Move coordinate system while performing the morph

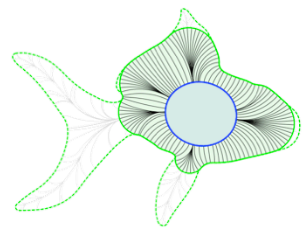


- Given 2 or more shapes, align them (registration) to define the motion, then compute the Minkowski morph of the aligned shapes

# Tangent Ball map & morph

- Distance between shapes
- Discrepancy measures (Hausdorff, Frechet, Tangent Ball)
- Closest Point (CP) and Tangent Ball (TB) map and morph
- Relative rounding





# Motivation

Dimension and tolerancing of machined parts

- Specify allowable discrepancy in shape

Manufacturing inspection and yield control

- Visualize areas where the process deviates from nominal

Bone grafting

- Ensure that the graft perfectly matches the desired shap

Shape Simplification

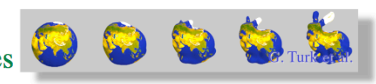
- Measure the discrepancy between original and approximatio

Animation

- Morph between two similar shapes

Rendering

- Map textures between different shapes



Analysis

- Track/transfer properties between (evolving) (iso) surfaces



The goal is to develop a precise mathematical formulation of what it means to simplify a shape.

We are focusing on morphological and topological simplifications that are independent of the particular representation used.

To be useful as a fundamental building tool for the theory of shapes, a simplification process cannot be described by a heuristic or algorithm.

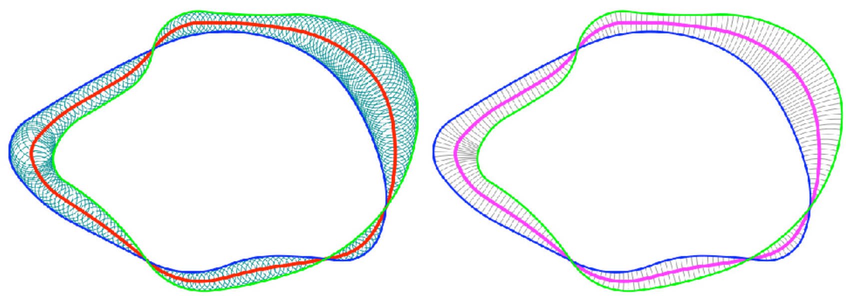
Hence, we must provide a formal, set-theoretic definition of the simplification process.

We hope to use such a formalism to develop new tools for multi-scale analysis of shapes and behaviors.

# Ball distance ✷new✷

$D_{Ball}$(**P**,**Q**) = Diameter of largest ball that fits in the **gap** and touches both shapes

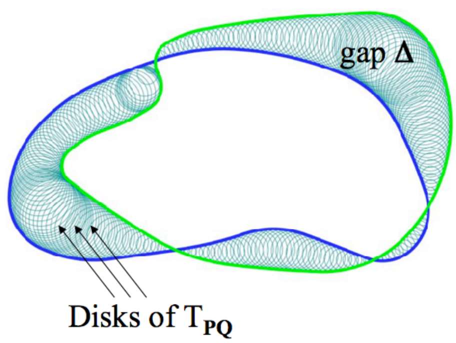
- Good measure if shapes are **b-compatible** (single point contacts)
  - Takes distance and orientation into account.
- Can be extended to incompatible shapes

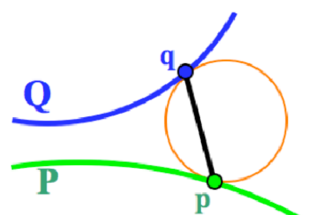


# Our “Tangent Ball” **b-map**

- $\Delta$= (**iP⊕iQ**) ∪ (**P∩Q**) is the **gap** between the two shapes
- $T_{PQ}$ is the set of **disks/balls** in **Δ** that touch both **P** and **Q**
- $B_{P\leftrightarrow Q}$ maps **S∩P** to **S∩Q** for each disk **S** of $T_{PQ}$

  Definition works in 2D (disks) and 3D (balls)






We propose to replace the c-map with the “tangent-ball” b-map.

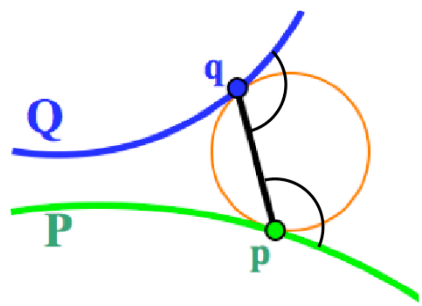
The gap between P and Q is the symmetric difference of their interiors, plus their intersection.

T is the set of spheres (in 3D) or circles (in 2D) that are in the gap and touch both P and Q.

The b-map takes each sphere of T and maps its intersection with P to its intersection with Q

# B-map advantages

- $B_{\mathbf{P}\leftrightarrow\mathbf{Q}}$ is symmetric: $B_{\mathbf{Q}\leftrightarrow\mathbf{P}}(B_{\mathbf{P}\leftrightarrow\mathbf{Q}}(\mathbf{p}))=\mathbf{p}$
- $B_{\mathbf{P}\leftrightarrow\mathbf{Q}}$ is sensitive to the orientation of both shapes






Our b-map does not have the draw-backs of the c-map:

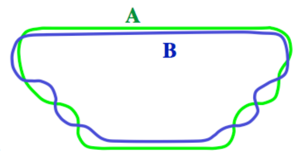
- It is symmetric (swapping P and Q produces the reverse map)
- It takes the orientation of both shapes into account in a symmetric fashion

In fact when the shapes are smooth and the b-map is homeomorphic, the segment pq reaches both shapes with the same incidence angle

# B-compatible curves

Two curves A and B are **b-compatible** if

$$H(A,B) < \max(LFS(A), LFS(B))$$




- $h < f \;\; \Rightarrow$ P and Q are b-compatible

  $h$ = **Hausdorff distance** between **A** and **B**.

  $h$ = smallest $r$ such that $\mathbf{A} \subset \mathbf{B}^r$ and $\mathbf{B} \subset \mathbf{A}^r$,

  where $\mathbf{X}^r$ is the set of points at distance $r$ or less from **X**.

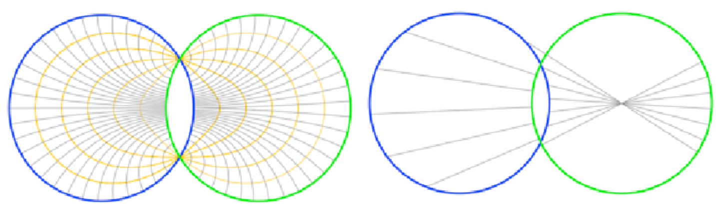
  $f$ = min( mfs(**A** ) , mfs( **B** ) )

  minimum feature size mfs(**X**) = largest $r$ such that $\mathbf{X}=\mathbf{F}_r(\mathbf{X})$
  where $\mathbf{F}_r(\mathbf{X})$ is the set not reachable by open r-ball not intersecting **X**

- *$h < f$ is less constraining than:* $h < 0.58\, f \;\; \Rightarrow$ **P** and **Q** are c-compatible

# Comparing compatibility conditions

- **P** and **Q** are **c-compatible** when both $C_{\mathbf{P}\to\mathbf{Q}}$ and $C_{\mathbf{Q}\to\mathbf{P}}$ are homeomorphisms
- c-compatibility implies b-compatibility
- **P** and **Q** are **b-compatible** when $B_{\mathbf{P}\leftrightarrow\mathbf{Q}}$ is a homeomorphism



b-compatible    not c-compatible



A homeomorphism is a continuous and bijective (one-to-one and onto) map.

We say that P and Q are b-compatible when the b-map is a homeomorphism and c-compatible when both c-maps are homeomorphisms.

Note that c-compatible shapes are necessarily b-compatible, but that the reverse is not true.

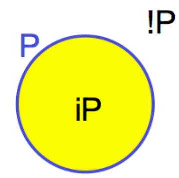
Hence many pairs of shapes that are not c-compatible may have a homeomorphic b-map.

# Restrictions for b-morphing

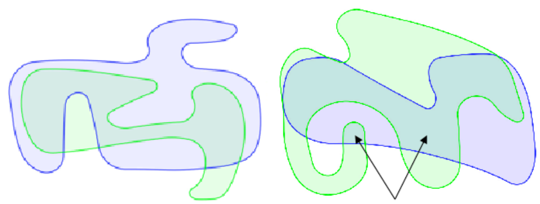
- **P** and **Q** are boundaries (curves in 2D, surfaces in 3D) of regularized regions (area, solid)

  **iP** is the finite interior (region) bounded by **P**

  **!P** is the exterior (complement of **P**∪**iP**)




- We assume that:
  - **P** and **Q** are connected and manifold
  - **iP**∩**iQ** is connected
  - **!P**∩**!Q** is connected



Good

Bad: disconnected

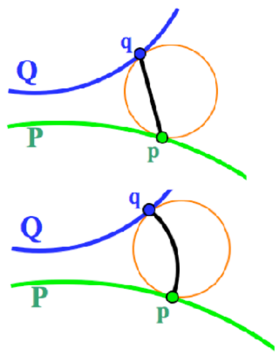


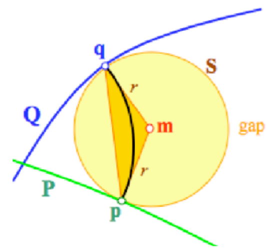
Therefore, we focus on situations where the two shapes are sufficiently similar and properly registered.

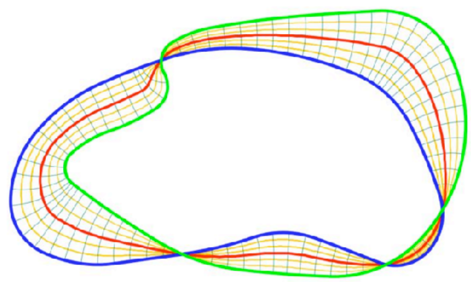
For simplicity, we assume that the intersections of their interior and of their exterior are both connected.

# B-morph with circular trajectories

- Circular trajectories
- Normal incidence on both shapes





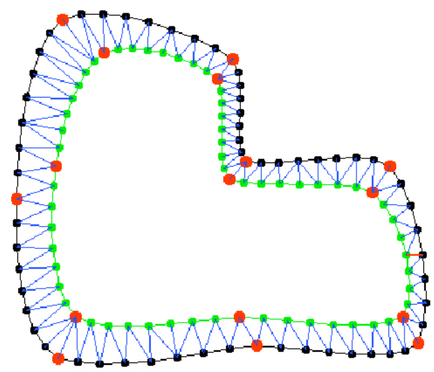
The b-map permits to define a b-morph between the two shapes during which each point travels along a circular arc trajectory.

The trajectory leaves point p along the surface normal and arrives at q along surface normal.

The circular trajectory is defined by its inscribing isosceles triangle (p,m,q))

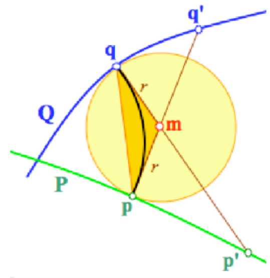
# Approximating the middle set

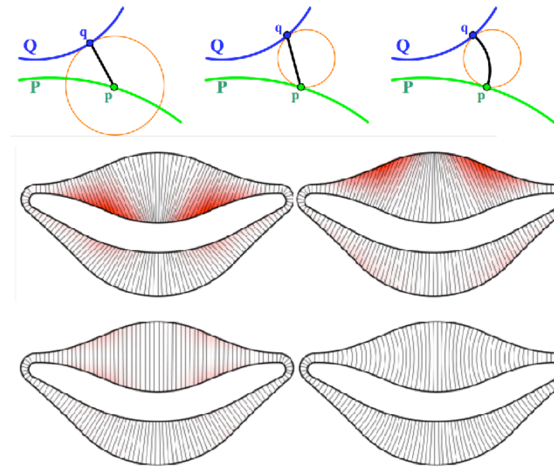
- Constrained Delaunay triangulation
- Join the centers of adjacent triangles (no bifurcation!)



# Average travel distance

b-morph requires less average travel than either c-morphs
when integrated over the median or over both shapes



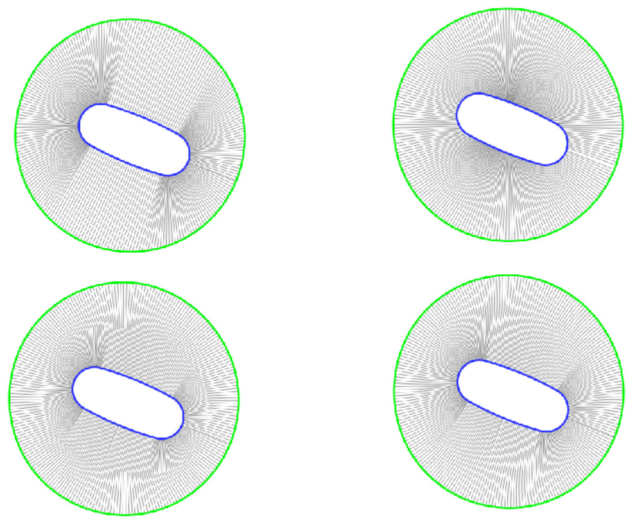

| | |
|---|---|
| c-morph from P to Q, ‖**p'q**‖: | 3555 |
| c-morph from Q to P, ‖**q'p**‖: | 3560 |
| Straight-line b-morph, ‖**pq**‖: | 3464 |
| Circular b-morph, arc-length: | 3492 |



Although it may appear surprising, the b-morph yields a lower average travel distance than a c-morph.

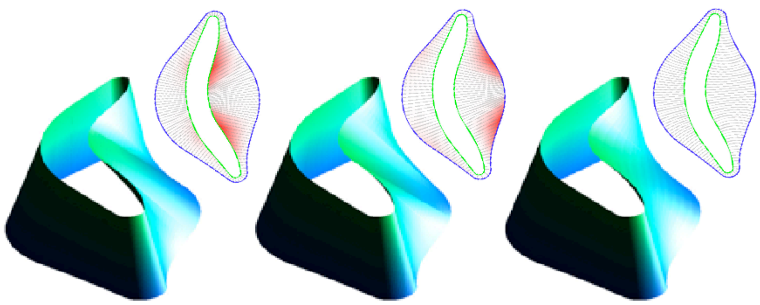
Integrating the travel distance over P would imply that C P to Q has lowest travel, but is incorrect, because it does not take into account the distortion.

# Average travel distance: case study



# Fitting surface between cross-sections

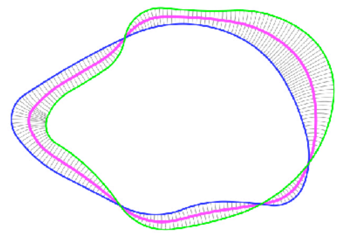
The b-morph yields an interpolating surface with a smaller surface area and a “nicer” shape



Two c-morphs and a b-morph. Which one is the “nicest”?

# Discrepancy between b-compatible shapes

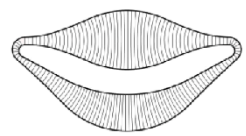
- Local measure: length of arc from **p**
- Global measure: Integral of arc-length trajectory
  - Integrate over **M**, **H**, or **P** & **Q**
  - $L_1, L_2, L_\infty$



- When $h<f$:

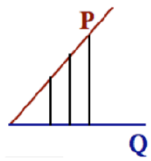
*length of longest arc* $= D_{Haus}(\mathbf{P},\mathbf{Q})$

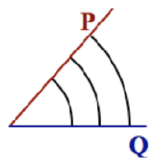
$= D_{Fre}(\mathbf{P},\mathbf{Q})$

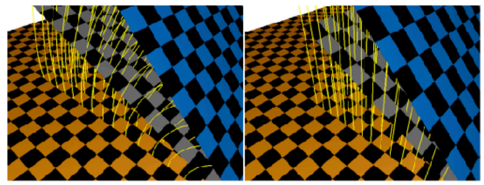


# Distortion

Ball morph between linear sections is distortion free

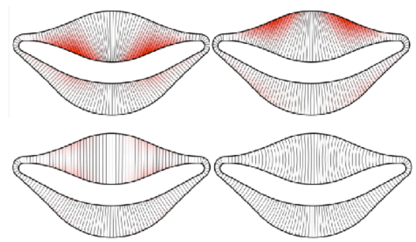


and in general is closer to normal (c-morph) flow

| | $C_{Q\to P}$ | $C_{P\to Q}$ | Linear $B_{P\leftrightarrow Q}$ | Circular $B_{P\leftrightarrow Q}$ |
|---|---|---|---|---|
| $\sum E(s,t)/S(s,t)$ | 2390 | 2373 | 2132 | **2118** |
| $\sum E(s,t)/\sum S(s,t)$ | 1.0378 | 1.0370 | 1.0090 | **1.0005** |



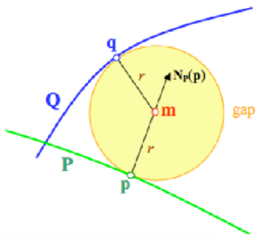
# b-map computation

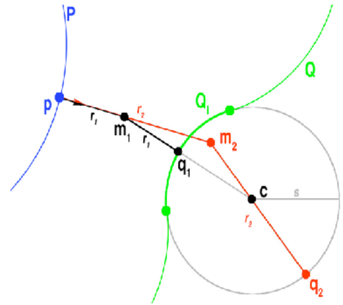
- Grow tangent ball at **p** in the gap until it touches **Q**.

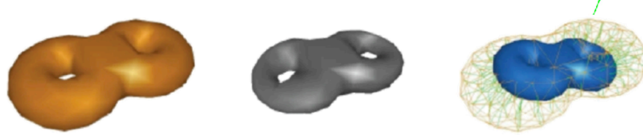
  **m** = **p** +$r$N$_{\mathbf{P}}$(**p**) find smallest r such that Dist(**m**,**Q**)=$r$

  Solve for all elements of **Q** (points, line segments, arcs, triangles…).

  Return smallest $r$>0



# Mid-edge to vertex b-morph

- E=(A+B)/2 // mid edge point
- N=(B–A).left.unit // normal to edge
- Find r such that $(E+rN–C)^2=r^2$

$(CE+rN)^2=r^2$

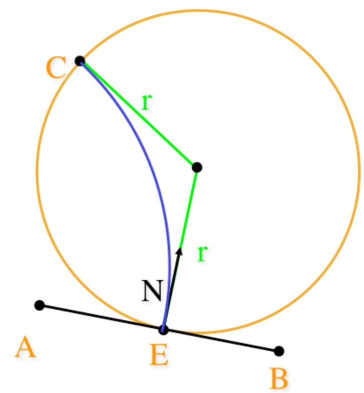
since $V^2=V•V$, distribute • over +

$CE^2+r^2N^2+2rCE•N=r^2$

since $N^2=1$, subtract $r^2$ on both sides

$CE^2+2rCE•N=0$

$r=–CE^2/(2CE•N)$

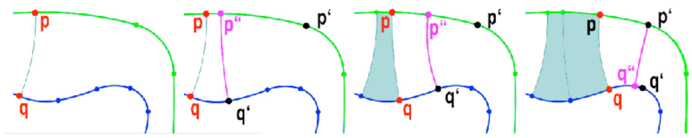


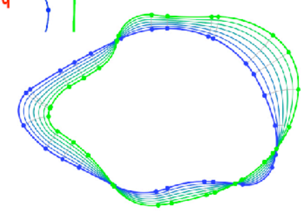
# Lacing PCCs

Linear cost algorithm  for b-compatible piecewise circular curves

- March on both curves
- Compute the images of the end-points of current edge
- Take the smallst of the two steps

Divides the gap into slabs, each bounded by 4 arcs



# B-offset reconstruction

Express **Q** as the b-offset of **P**

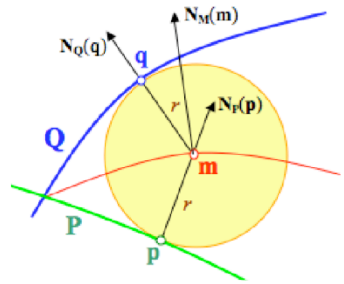
Represent it by **P** and the offset field **r**(p)

Reconstruct **Q** by computing:

**m**=**p**+$r$**N**$_P$(**p**)

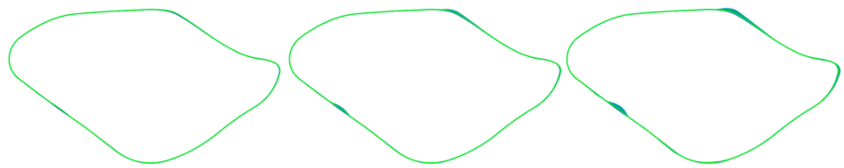
**N** = normal to **M** at **m** (from derivatives of r in tangent plane)

**q**=**p**+2(**pm**•**N**)**N**



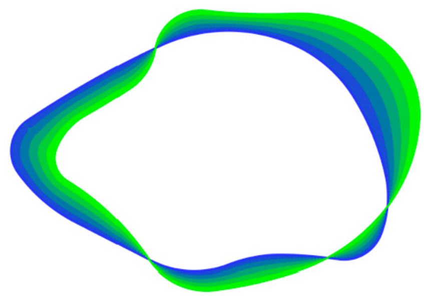
# Exaggeration

Barely noticeable discrepancies between b-compatible shapes may be exagerated by scaling the field r



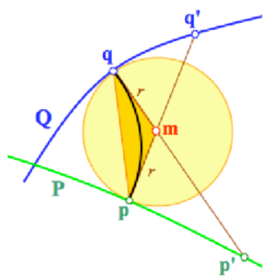
# Progressive offset

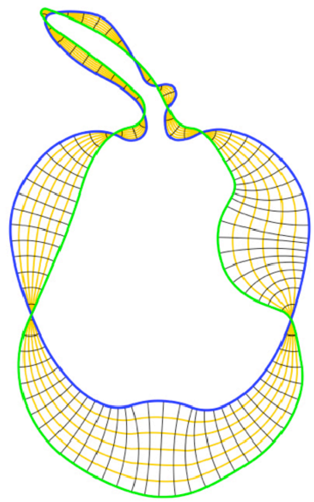
Increasing the scaling of r from 0 to 1 creates a morph that sweeps the gap



# B-morph

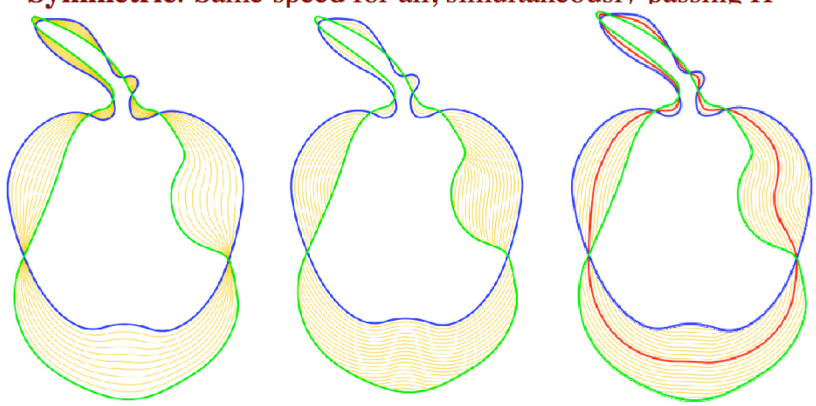
- Each sample travels on its circular arc from **p** to **q**



# B-morph synchronization

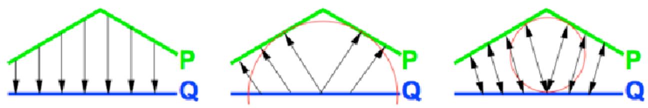
- **Regular**: Constant speed for each trajectory
  - Computed so that they all start and stop at the same time
- **Invading**: Same speed for all, starting at **P**
- **Symmetric**: Same speed for all, simultaneously passing H



# Non b-compatible shapes

Assume now that **P** and **Q** are not b-compatibile

- One of the c-maps can still be a homeomorphism
- $C_{\mathbf{P}\to\mathbf{Q}}$ can be a homeomorphism, while $C_{\mathbf{Q}\to\mathbf{P}}$ and $B_{\mathbf{P}\leftrightarrow\mathbf{Q}}$ are not





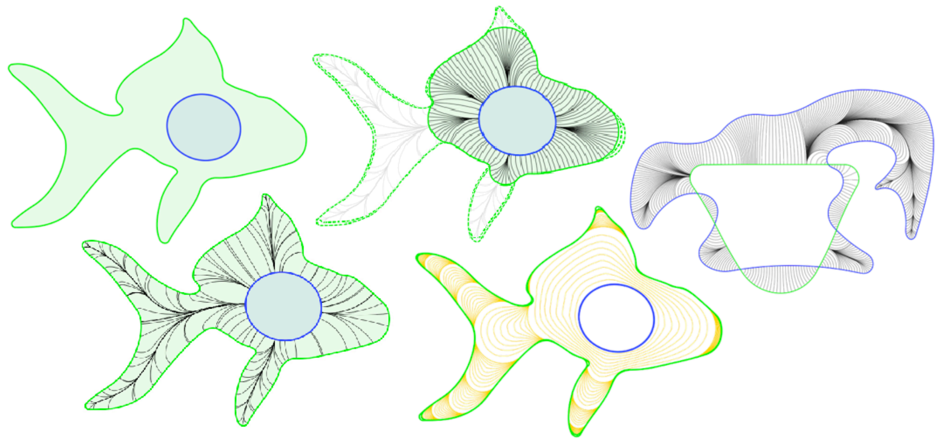

Let us now extend these ideas to pairs of shapes shapes that are not b -compatible

Which implies that they are not c-compatible,

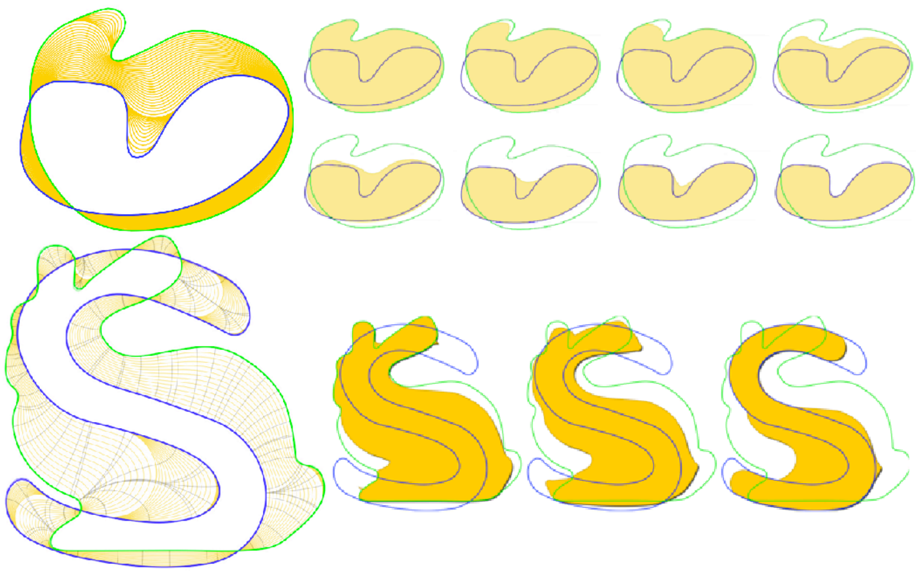
even though it is also possible for one of the c-maps to be homeomorphic , while the other and the b-map are not. This situation occurs near sharp features.

# Composite b-morph

- Compute b-roundings $\mathbf{P}_1$ =bRound($\mathbf{P}$,$\mathbf{Q}$), $\mathbf{Q}_1$=bRound($\mathbf{Q}$,$\mathbf{P}$)
- Then recursively: $\mathbf{P}_{i+1}$=bRound($\mathbf{P}$,$\mathbf{P}_i$), $\mathbf{Q}_{i+1}$=bRound($\mathbf{Q}$,$\mathbf{Q}_{i+1}$)
- Stop when $\mathbf{P}$=$\mathbf{P}_i$ or when $C_{\mathbf{P}\rightarrow\,\mathbf{P}_i}$ is a homeomorphism
- Note that trajectories are smoothly connected PCCs

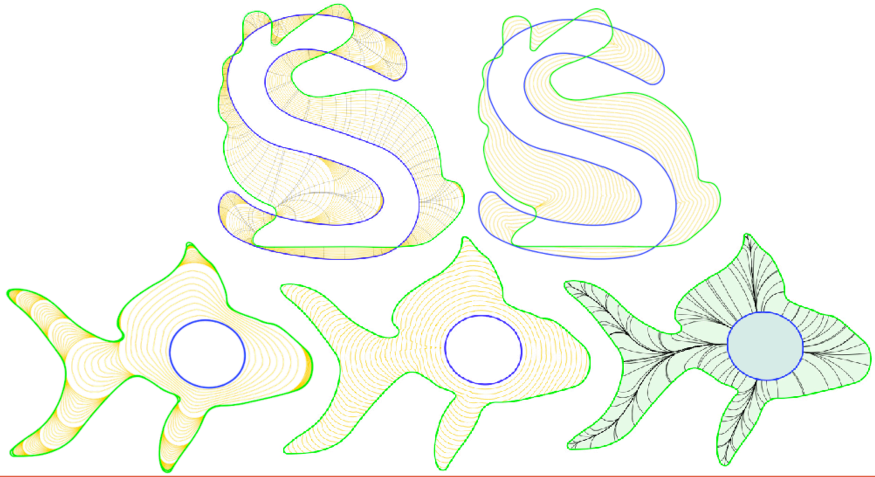


# Examples of composite b-morph



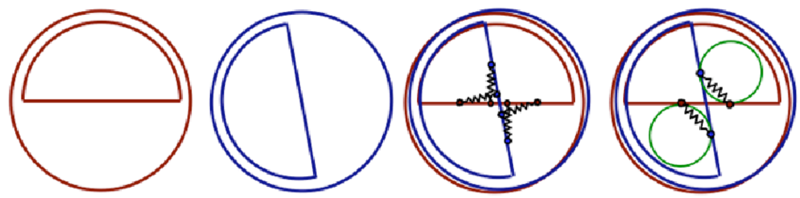
# Synchronization of composite b-morphs

**Regular**: Each morph has constant speed
**Invading**: All same speed staring at one shape
**Symmetric**: Same speed, synchronized at H



# Registration

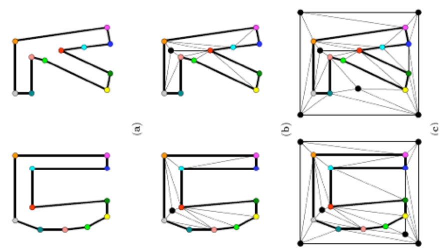
- The b-map is better than the c-map for ICP registration
  ICP = Iterative Closest Point



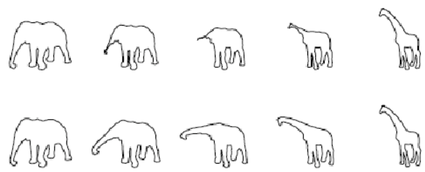
# Summary

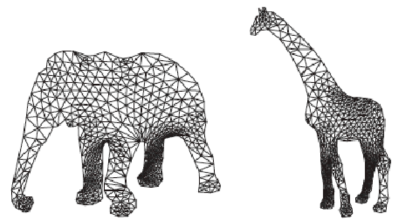
- B-morph is better than c-morph or parametric morph
  - symmetric
  - orientation sensitive
  - shorter travel and smaller surface area
  - less (or no) distortion
  - more compatible (except near sharp features)
  - implement Frechet distance (if h<f)

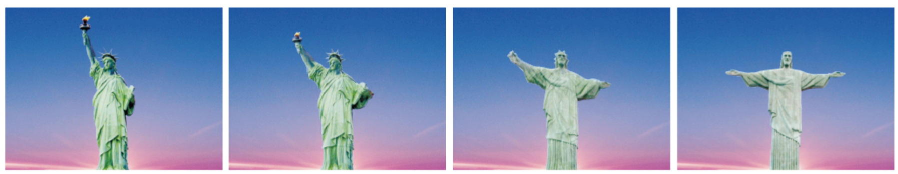
# Other approaches (Surazhsky & Gotsman)



# As rigid as possible warp (Alexa)

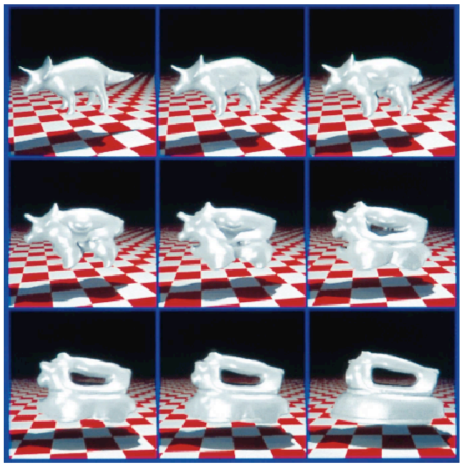


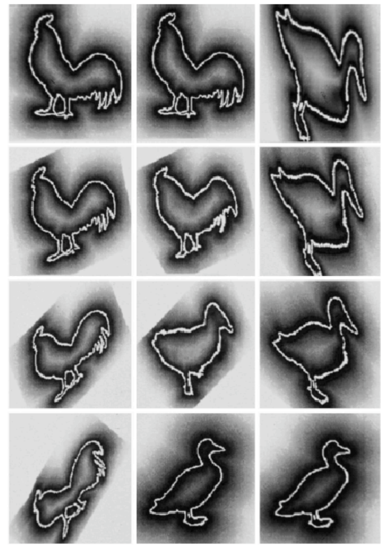




# Distance field blend (Cohen-Or…)

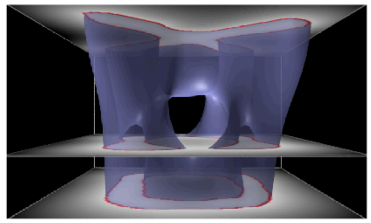
- Align and merge distance fields





# Fit hyper-surface to constraints [Turk&O'Brien 1999]

- 2D slices common (medicine)
- Implicit surface interpolates slices
- Slow
- Delicate



- Extended to 4D