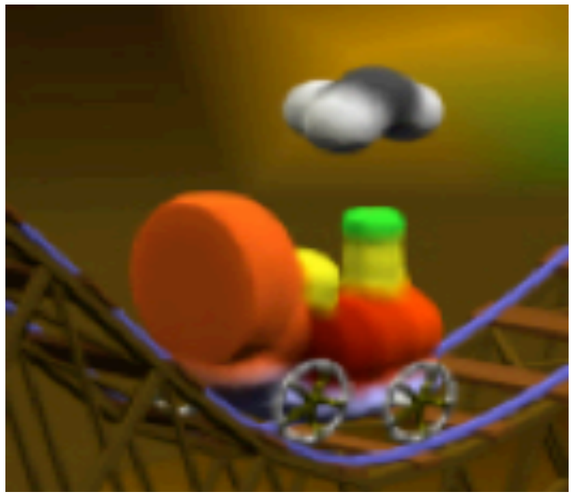
Advanced Computer Graphics

Object Representations Beyond Polygons (and Rendering them ...)



G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Quadrics (Quadric Surfaces/Forms)

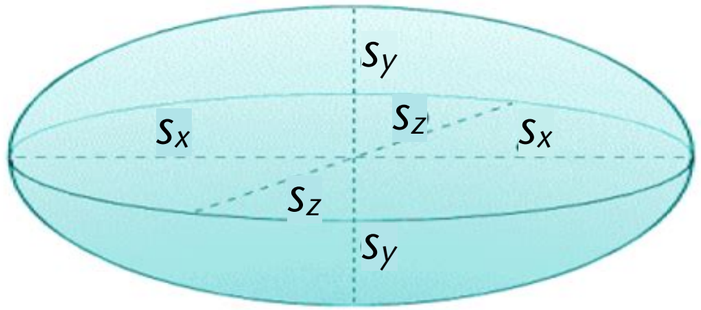
- Implicit form of a sphere (center at origin):

$$x^2 + y^2 + z^2 = r^2$$

$$\mathbf{x}^2 = \mathbf{x}^\mathsf{T}\mathbf{x} = r^2$$

- Generalization: ellipsoid

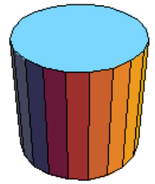
$$\left(\frac{x}{s_x}\right)^2 + \left(\frac{y}{s_y}\right)^2 + \left(\frac{z}{s_z}\right)^2 = r^2$$



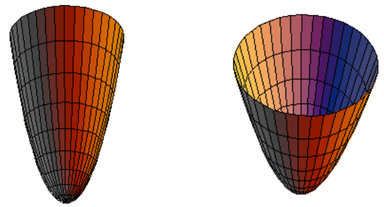
$$\mathbf{x}^\mathsf{T} \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & -d \end{pmatrix} \mathbf{x} = 0, \quad \text{with } \mathbf{x} = (x\, y\, z\, 1), \text{and } a, b, c, d > 0$$

- In the following: scaling factors will be omitted, but can always be introduced

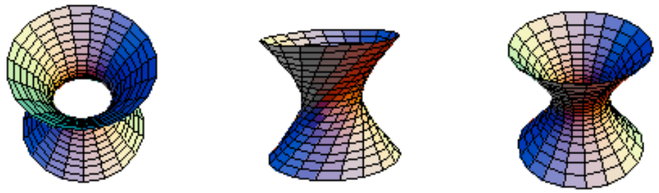
- Infinite cylinder: $x^2 + y^2 = 1$



- Paraboloid: $x^2 + y^2 - z = 0$



- Hyperboloid (*one sheet*): $x^2 + y^2 - z^2 = 1$

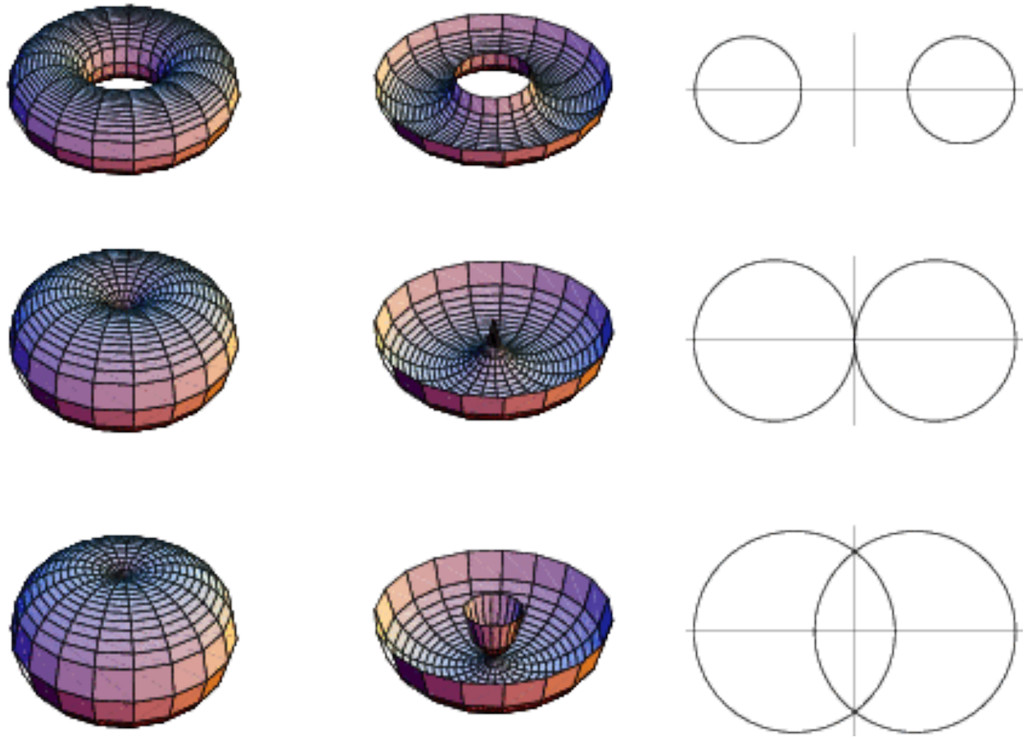


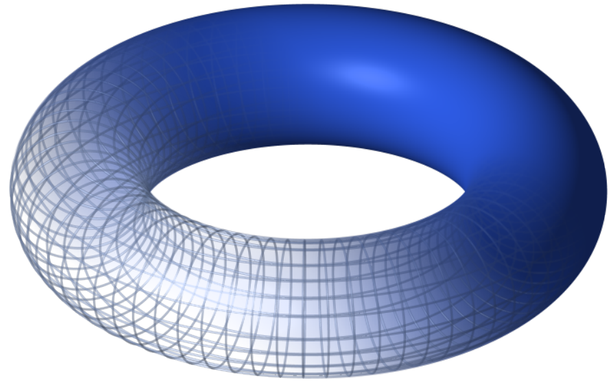
- All of these can be written as a quadratic form (hence the name):

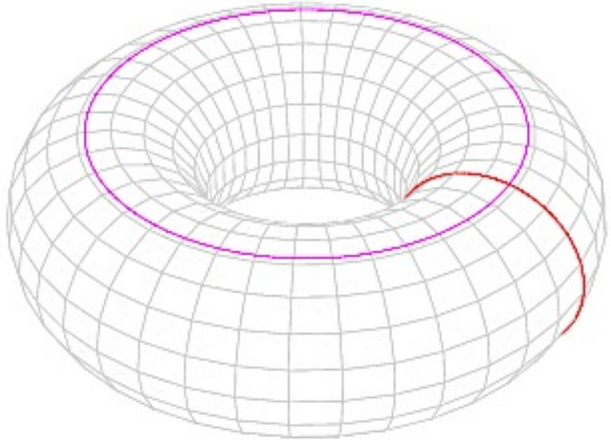
$$\mathbf{x}^T Q \mathbf{x} = 0, \quad \mathbf{x} \in \mathbb{R}^4, \; Q \in \mathbb{R}^{4\times 4}$$, with Q being symmetric

- Torus: (not really a quadric!)

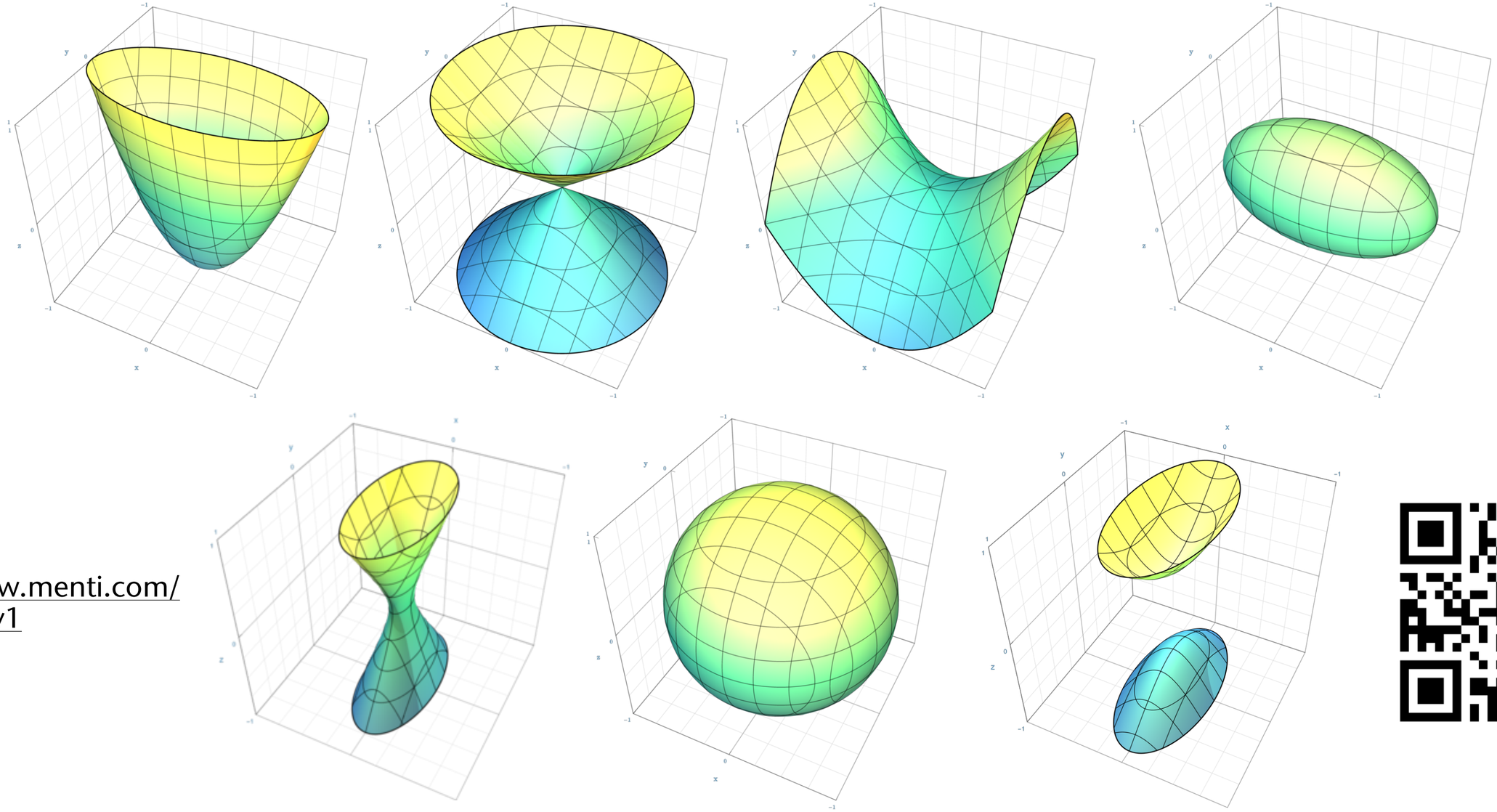
$$\left(c - \sqrt{x^2 + y^2}\right)^2 + z^2 = a^2$$







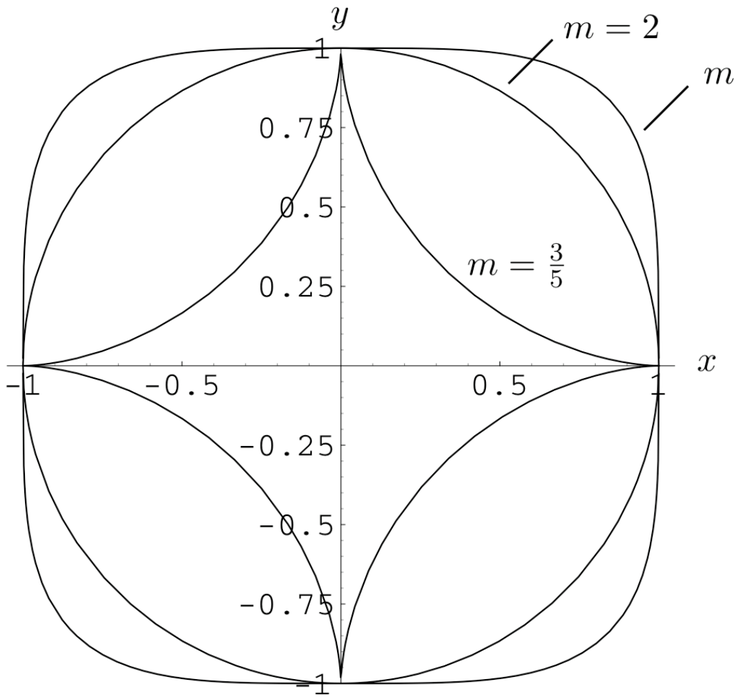
Quadric Surfaces Quiz: Which Picture Belongs to Which Equation?



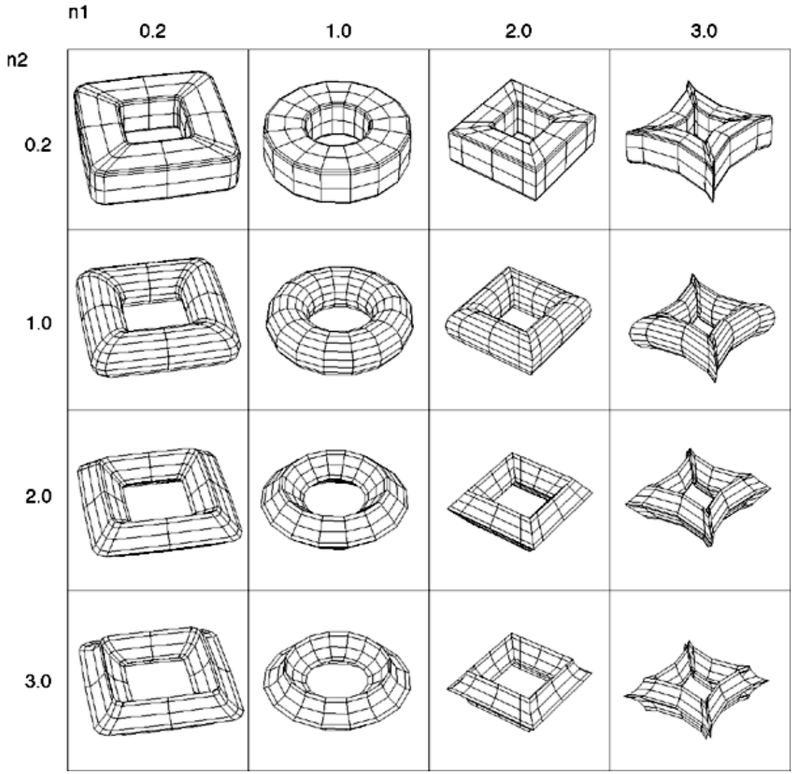
https://www.menti.com/ytms1d4mv1

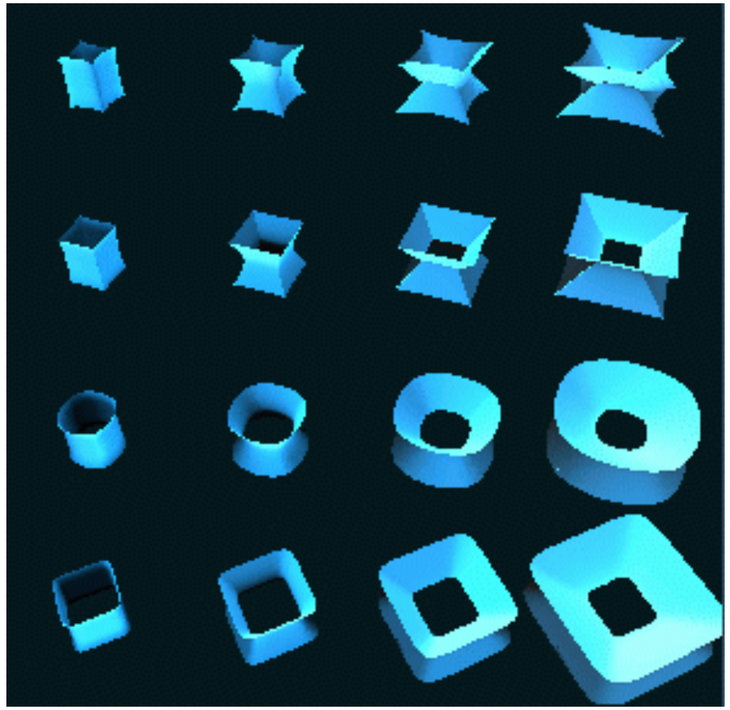
Generalization: Superquadrics

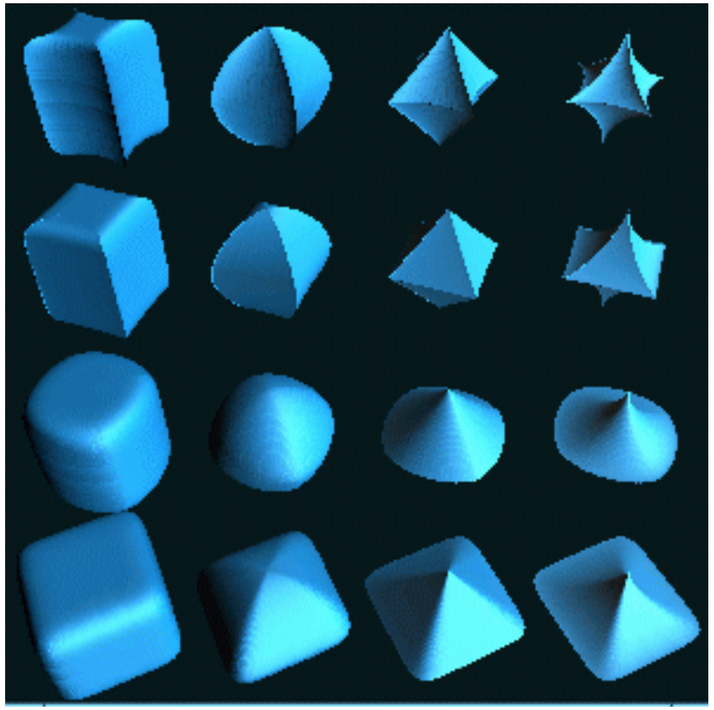
- Further generalization of quadrics
- Super-ellipsoid: $\left(\frac{x}{a}\right)^p + \left(\frac{y}{b}\right)^q + \left(\frac{z}{c}\right)^r = 1$
- Super-hyperboloid: $\left(\frac{x}{a}\right)^p + \left(\frac{y}{b}\right)^q - \left(\frac{z}{c}\right)^r = 1$
- Super-toroid: $\left(d - \left(\left(\frac{x}{a}\right)^m + \left(\frac{y}{b}\right)^n\right)^q\right)^r + \left(\frac{z}{c}\right)^p = e^2$
- Warning: in above equations, we always mean $|x|^p$!



Examples of Super-Quadrics







XScreenSaver demo "SuperQuadrics"
(www.jwz.org/xscreensaver)

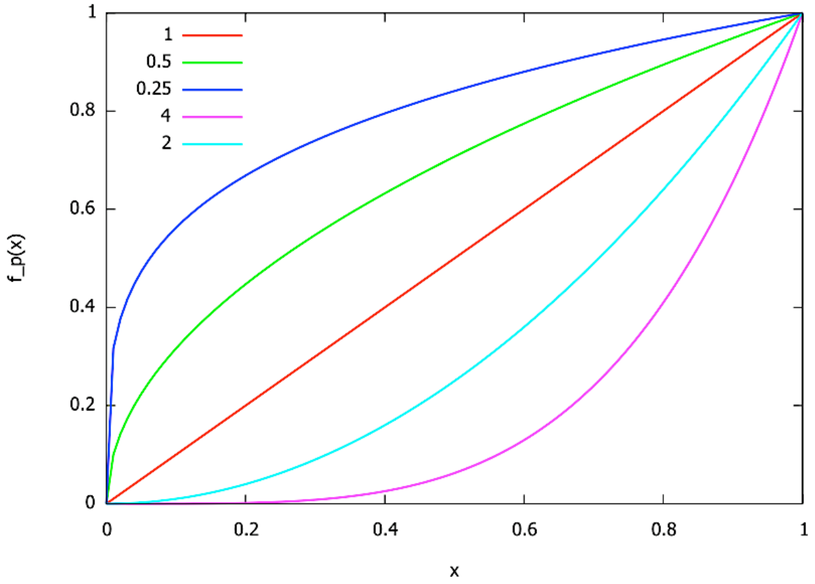
Ratioquadrics

Not Relevant for Exam

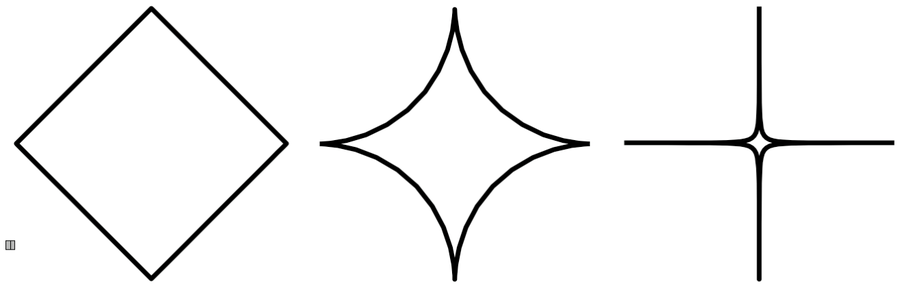
- Variant of superquadrics with somewhat better properties
- Idea of superquadrics can be rewritten like this:

$$F(x,y,z) = f_p\left(\frac{x}{a}\right) + f_q\left(\frac{y}{b}\right) + f_r\left(\frac{z}{c}\right) - d$$

$$f_p(x) = |x|^p \quad , p \in \mathbb{R}, p > 0$$



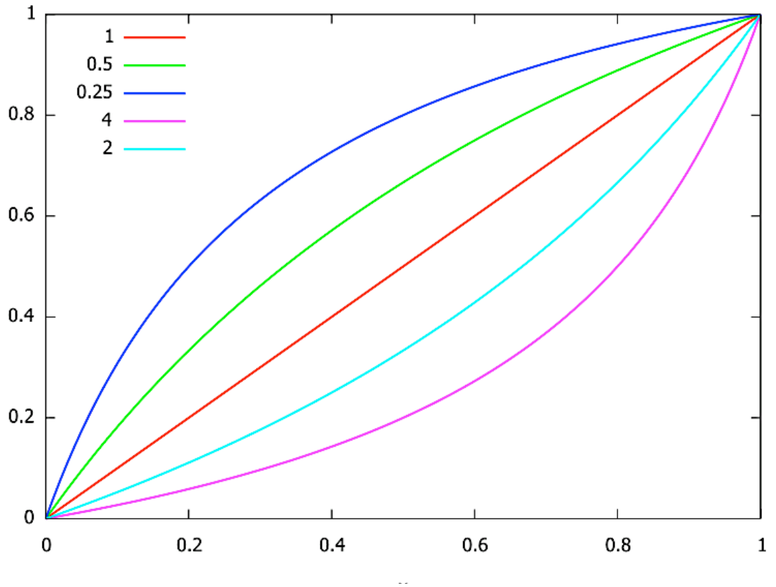
- Problem:
  - $f_p(x)$ is not differentiable at x=0 for $p \leq 1$
  - Thus, we get cusps, which might be unwanted
  - Besides, $f_p(x)$ is fairly expensive to evaluate



Not Relevant for Exam

- Simple idea: use different power functions
- E.g., the following pseudo-power function:

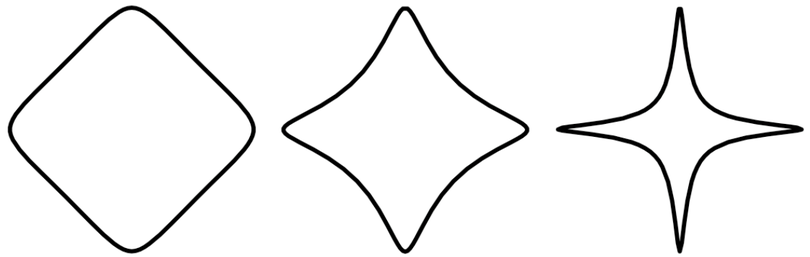
$$g_p(x) = \frac{x}{p + (1-p)x}$$



- With that, the ratioquadric for a "ratio-ellipsoid" is

$$F(x,y,z) = g_p\left(\frac{x}{a}\right) + g_q\left(\frac{y}{b}\right) + g_r\left(\frac{z}{c}\right) - 1$$

- Result:

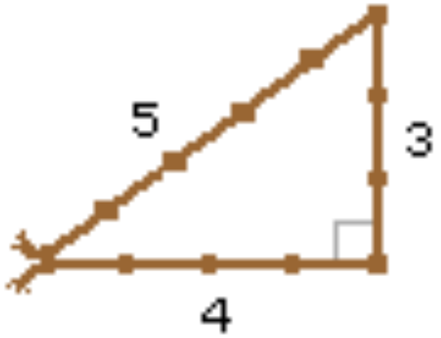


Not Relevant for Exam

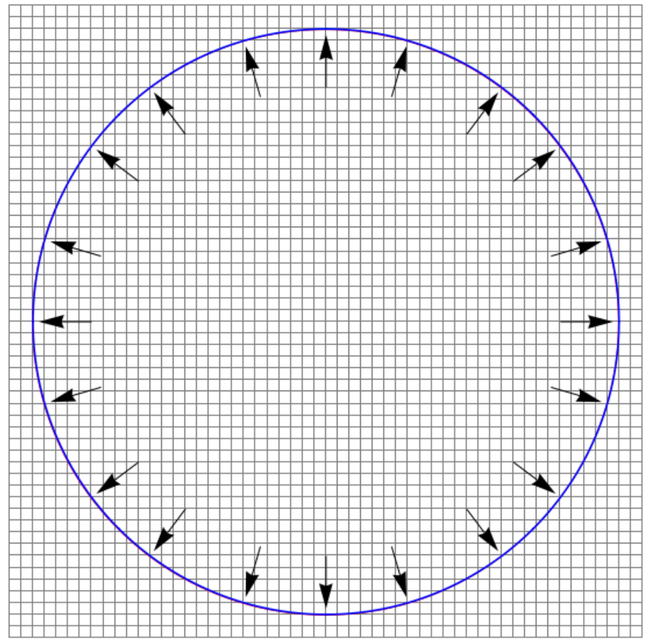
Digression: Fermat's Last Theorem and Super-Quadrics in 2D

- From Pythagoras, we know there are solutions for

$$a^2 + b^2 = c^2$$



- There are even *integer* solutions, the so-called Pythagorean Triples, e.g. {3, 4, 5}, {6, 8, 10}, and infinitely many more
- Visualize this in the plane on a lattice:
  - Radius of circle = $r = c^2$
    - Not every integer radius qualifies, of course
  - Solutions (a,b) = lattice points *on* the circle
    - Except ($\pm r$,0) and (0, $\pm r$) don't count, of course



Not Relevant for Exam

- Question: are there (non-trivial) integer solutions for

$$a^n + b^n = c^n\,, \quad n > 2$$

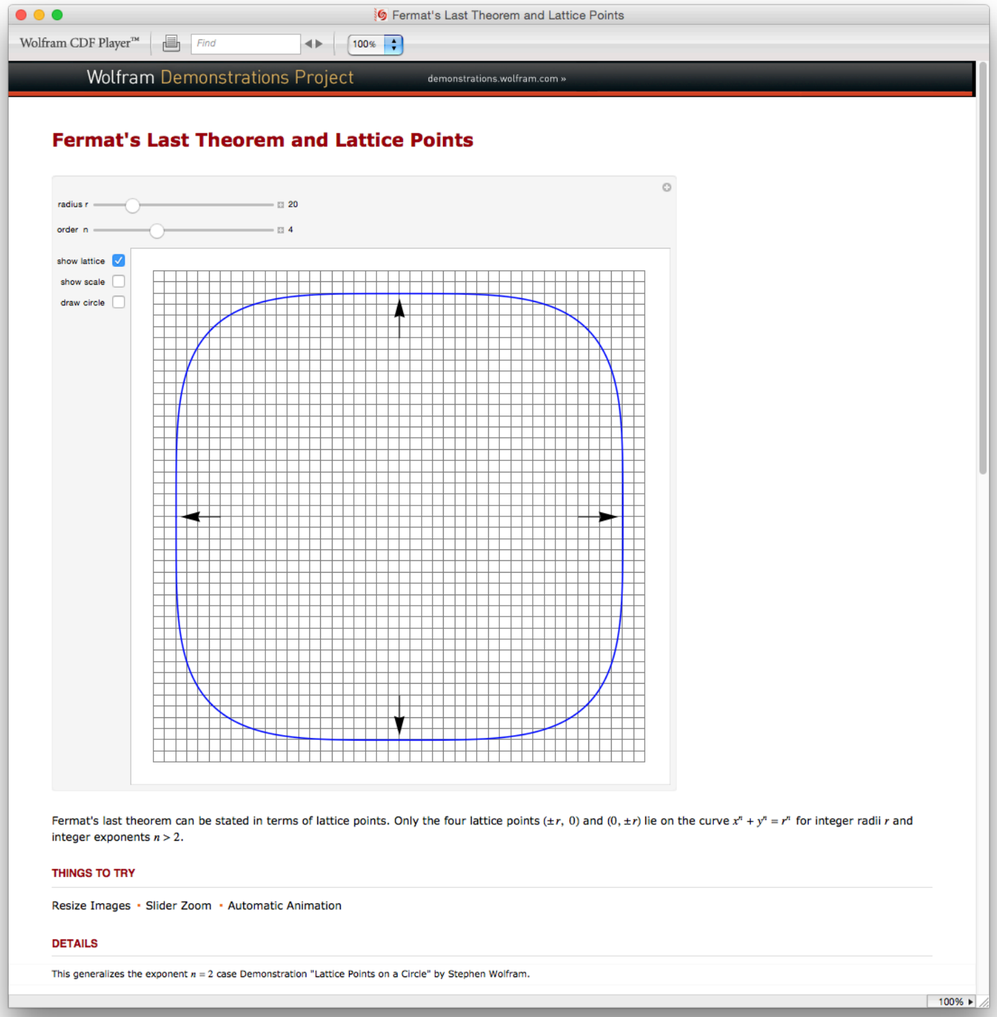
- Fermat's Last Theorem:

  No three positive integers *a, b, c* can satisfy the equation
  $a^n + b^n = c^n$ for any integer value $n > 2$

- Rephrased on the lattice:
  the curve $x^n + y^n = c^n$ never hits any lattice points (except the trivial ones),
  for all integer radii of the form c^n, where $n>2$.

Demo

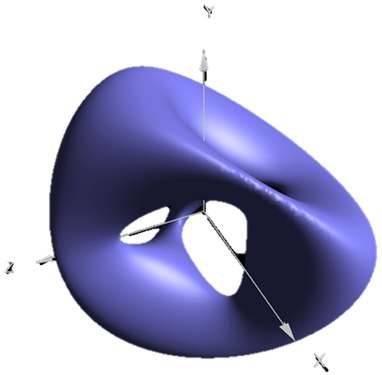
Not Relevant for Exam



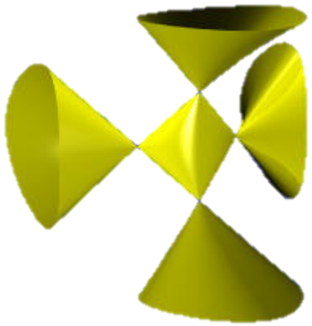
http://demonstrations.wolfram.com/FermatsLastTheoremAndLatticePoints/

Implicit Surfaces

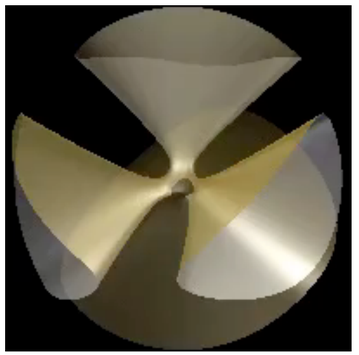
- Definition: an implicit surface is the set $\{\mathbf{x} \mid F(\mathbf{x}) = 0\,,\ \mathbf{x} \in \mathbb{R}^3\}$ of some function $F : \mathbb{R}^3 \to \mathbb{R}$
- Assumption in the following: $F \in \mathcal{C}^1$ or even $F \in \mathcal{C}^\infty$
- Example: all quadrics
- Another class: algebraic surfaces



$$(x^2+y^2+z^2-ak^2)^2-b((z-k)^2-2x^2)\,((z+k)^2-2y^2)^2$$



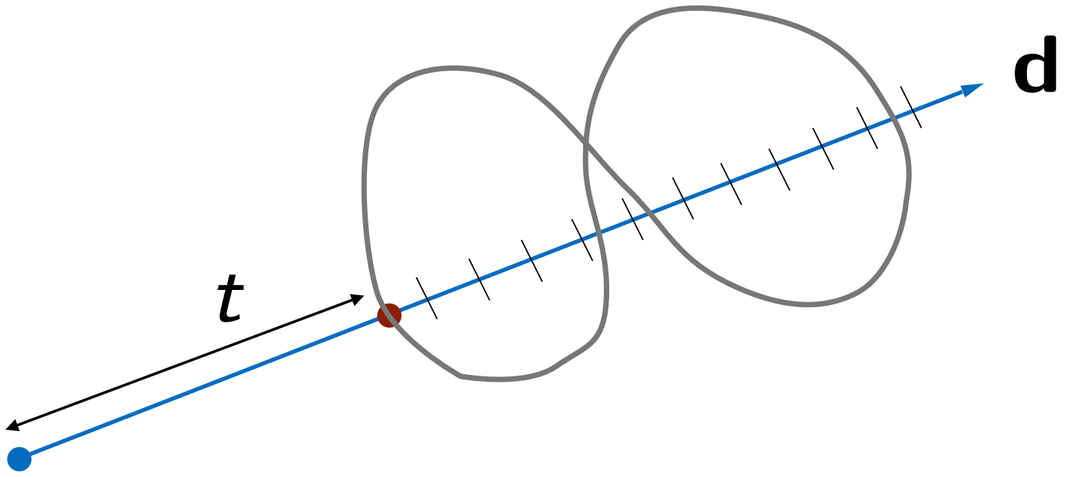
$$8x^2-xy^2+xz^2+y^2+z^2-8$$



$$(x+y+z)^3=x^3+y^3+z^3+1$$

Intersection of a Ray with an Implicit Surface

- Ray: $P(t) = O + t \cdot \mathbf{d}$
- Plugging ray into implicit function $F(\mathbf{x}) = 0$
  yields a 1D function in t: $F(\,P(t)\,) = 0$
- Find the roots:
  - If polynomial and degree < 5: solve for t analytically
  - Else: ...
  - Start values? ...



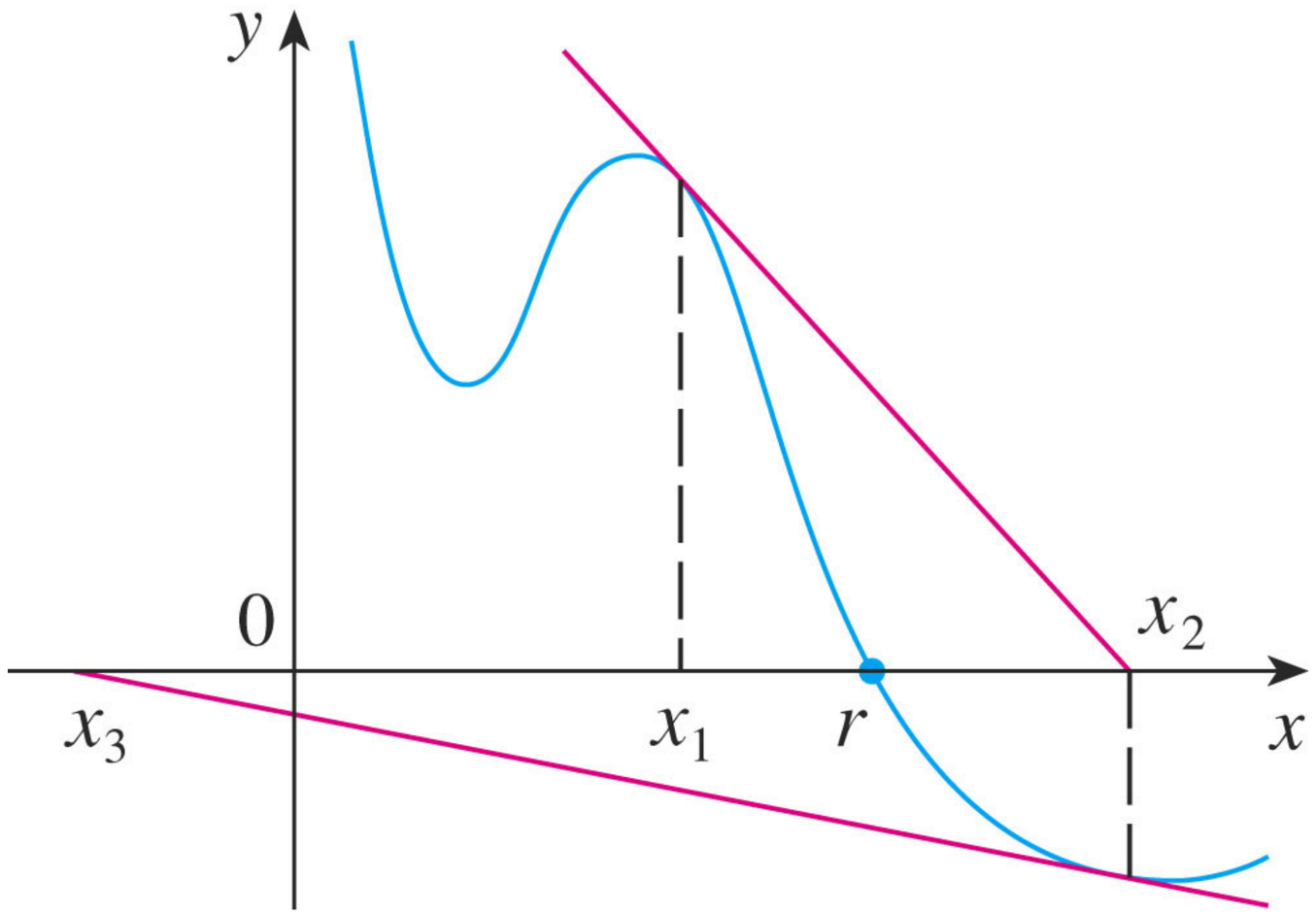
- Example with quadrics:

$$(O + t\mathbf{d})^{\mathsf{T}} Q (O + t\mathbf{d}) \stackrel{!}{=} 0$$

$$O^{\mathsf{T}} Q O + 2t O^{\mathsf{T}} Q\mathbf{d} + t^2 \mathbf{d}^{\mathsf{T}} Q\mathbf{d} \stackrel{!}{=} 0$$

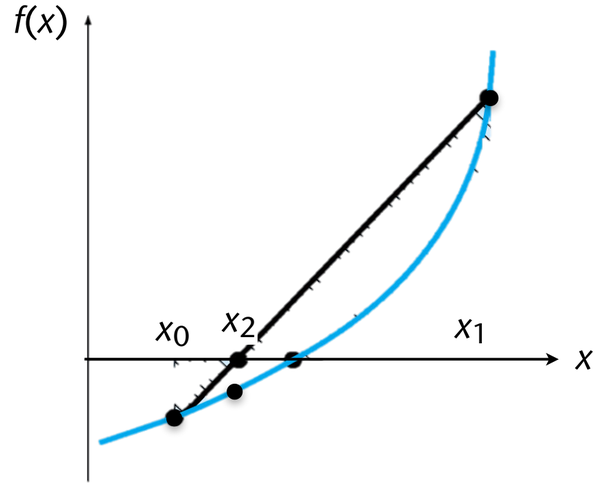
- Solve quadratic equation in t
- Cases:
  1. No solution $\longrightarrow$ no intersection
  2. Exactly one solution (i.e., value under sqrt = 0) $\longrightarrow$ exactly one intersection point, and ray is tangential to surface
  3. Two solutions:
     a) Both positive $\longrightarrow$ two intersections, take smaller t
     b) One negative $\longrightarrow$ ray starts in interior of quadric, use positive t

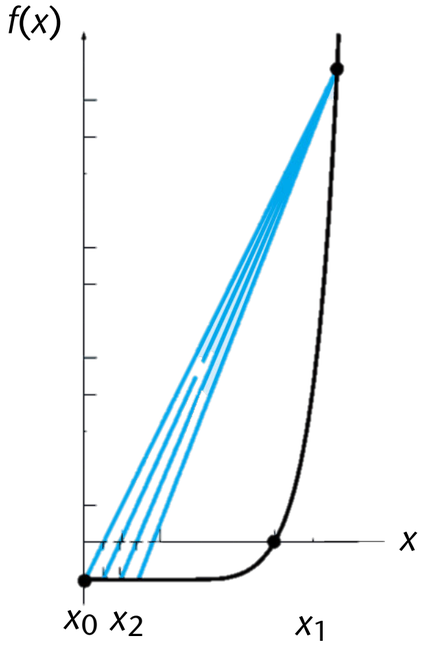
Root Finding – a Case Where Newton's Method Fails



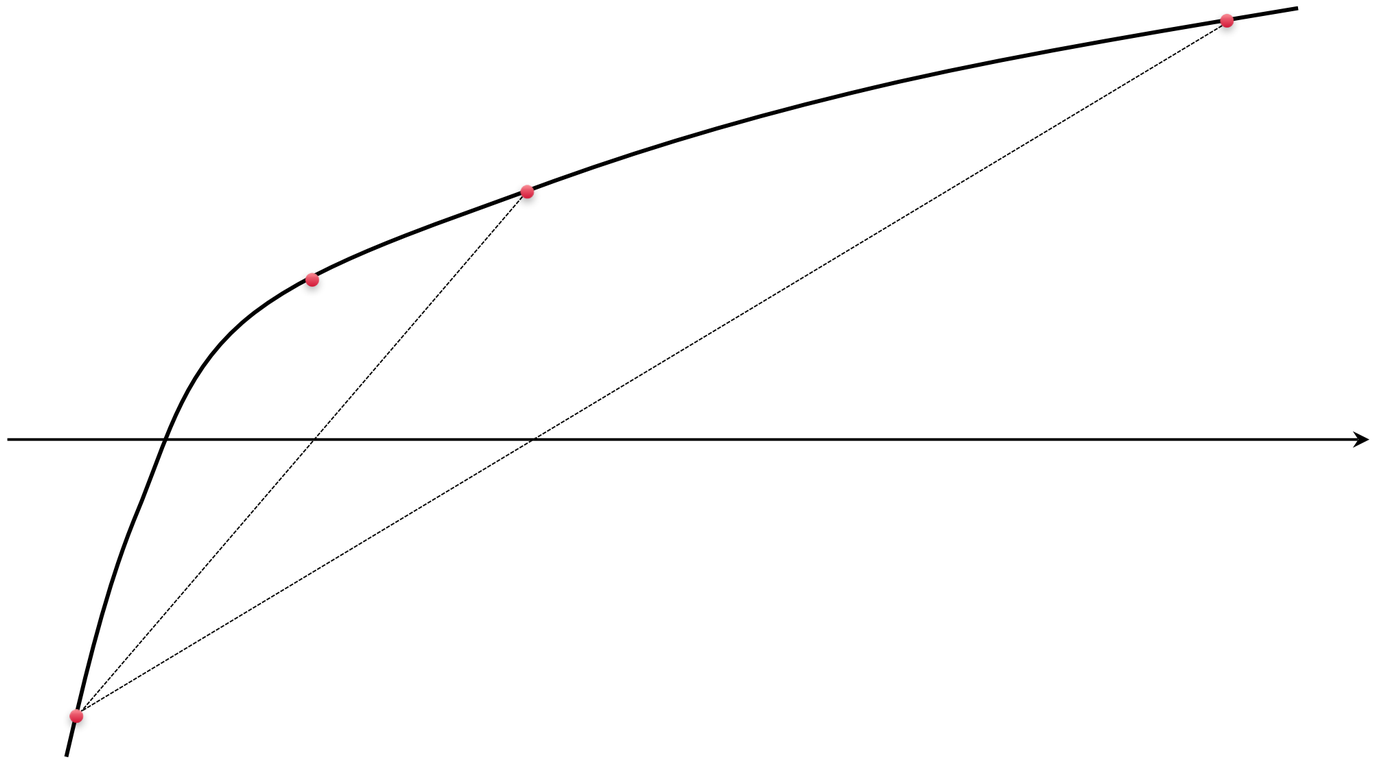
Simple, Sure-Fire Iterative Root Finding Methods

- Interval bisection:
  - Start with bracket $[x_0,x_1]$
  - $x_2 = \frac{1}{2}(x_0 + x_1)$
  - Keep $[x_0,x_2]$ or $[x_2,x_1]$
  - Convergence: 1 bit per iteration
- Regula falsi ("false position"):
  - As above, but with $x_2 = x_1 - f_1 \frac{x_1 - x_0}{f_1 - f_0} = \frac{x_0 \cdot f_1 - x_1 \cdot f_0}{f_1 - f_0}$
  - Converges always, usually faster than bisection
  - Detect "bad cases" $\longrightarrow$ switch to interval bisection for one step
- Hybrid method:
  - In each iteration, do bisection *and* regula falsi
  - Keep the smaller interval





- Problem of Regula falsi: once an interval is reached where *f* is convex (or concave), thereafter one of the end-points is always retained

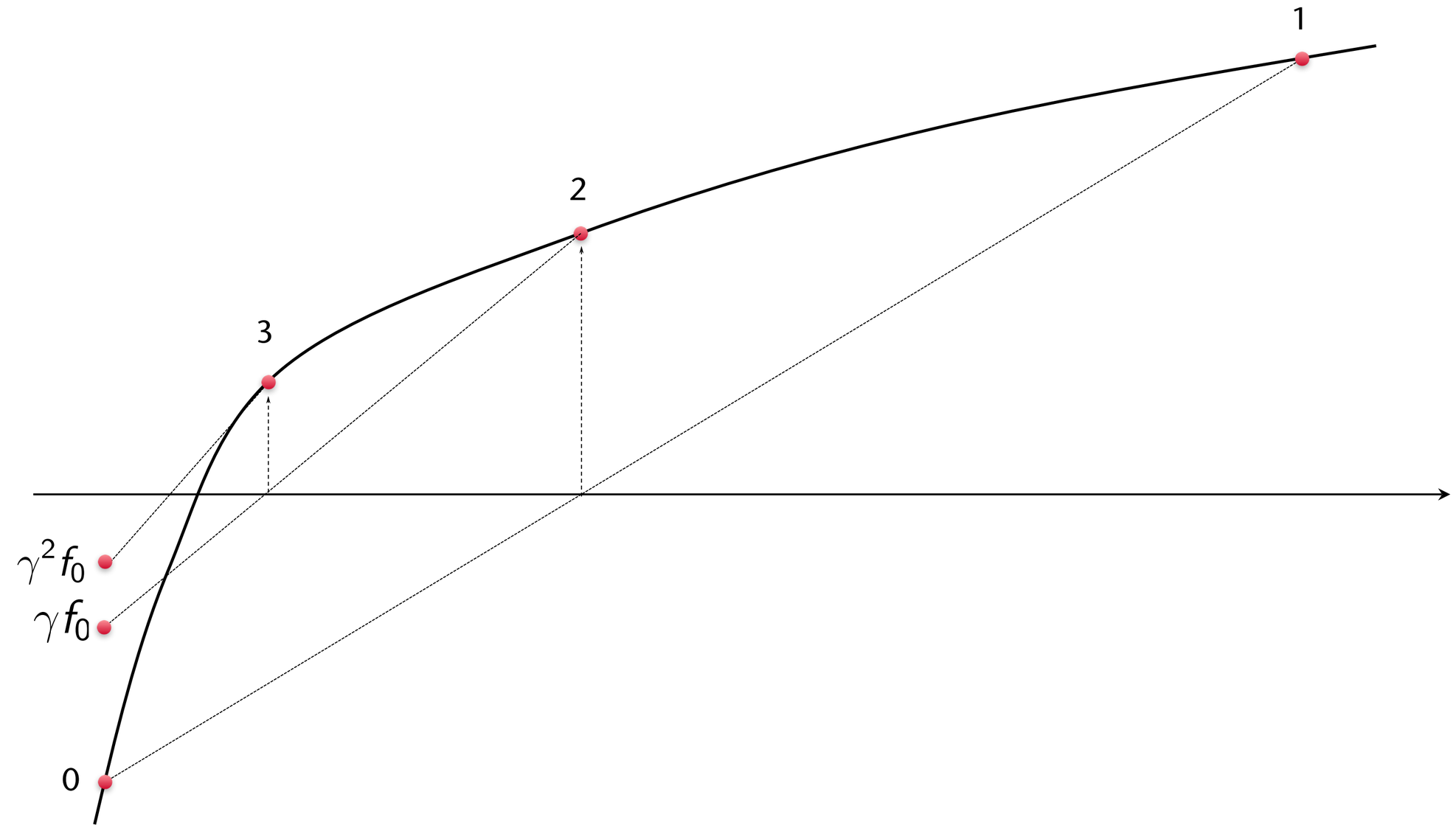


The Illinois Method

- Idea: try to detect cases of "end-point retention"
- Assume bracket $(x_0, f_0; x_1, f_1)$ with $f_i = f(x_i)$
  - x_0 is the "older" end-point
- Calculate $x_2 = \frac{x_0 f_1 - x_1 f_0}{f_1 - f_0}$ (standard regula falsi)
- If $\text{sign}(f_1) \neq \text{sign}(f_2)$:
  the new bracket is $(x_1, f_1; x_2, f_2)$;
  calculate $x_3 = \frac{x_1 f_2 - x_2 f_1}{f_2 - f_1}$ ("unmodified step")
- Else $\text{sign}(f_1) = \text{sign}(f_2)$:
  use new bracket $(x_0, \gamma f_0; x_2, f_2)$ (i.e., x_0 end-point is retained) ;
  calculate

$$x_3 = \frac{x_0 f_2 - x_2(\gamma f_0)}{f_2 - (\gamma f_0)} \quad \text{, with } \gamma = \frac{1}{2} \quad \text{("Illinois step")}$$

Visualization of the Illinois method



- Note to implementers: if the old end-point x_0 is retained again, i.e., if $\text{sign}(f_3) = \text{sign}(f_2)$, then the scaling with γ is applied again, i.e.,

$$x_4 = \frac{x_0 f_3 - x_3(\gamma^2 f_0)}{f_3 - (\gamma^2 f_0)}$$

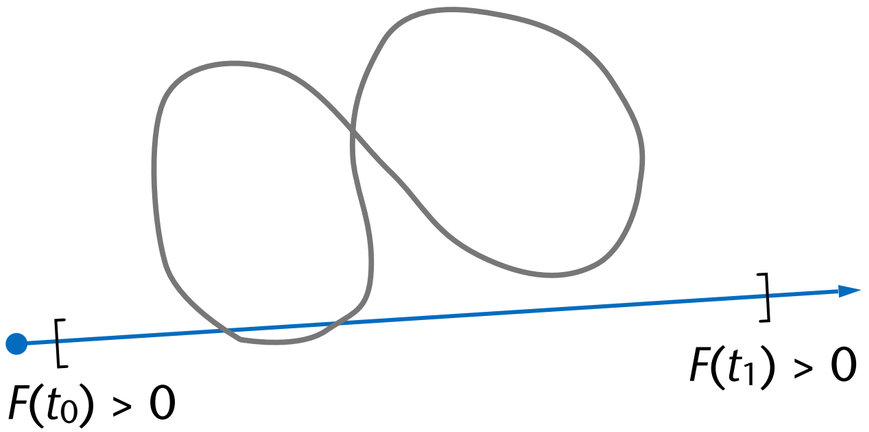
- Performance: empirical testing both methods (Hybrid and Illinois) with some functions shows that

$$\frac{\#\text{iterations(regula falsi + bisection)}}{\#\text{iterations(illinois)}} \approx 1.3 \ldots 2.0$$

- Other values of γ can lead to somewhat better convergence
  - E.g. "Pegasus method" or [Ford 1995]
  - But are they really faster? (on a real CPU)

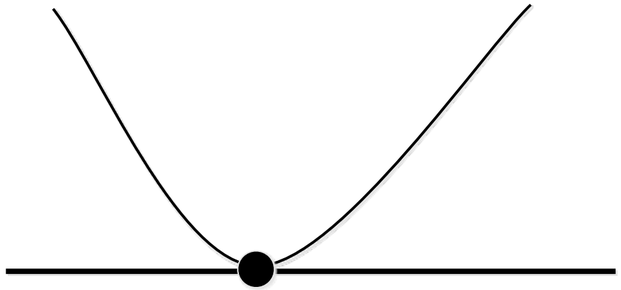
Note: the Initial Bracket is Tough to Find!

- The previous "sure fire "methods really only work, once you have a bracket!
- Approach: devise tests to *prevent* searching for roots unnecessarily
  - E.g., try co compute a bonding box
-

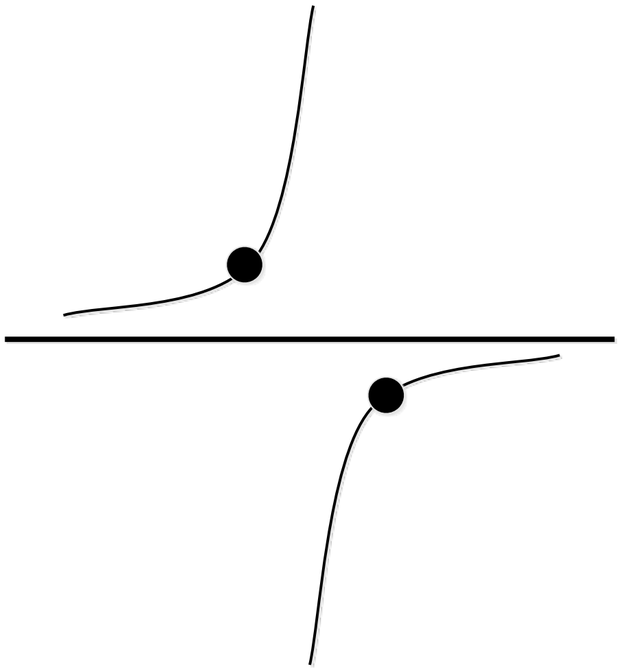


Beware of the Tough Cases

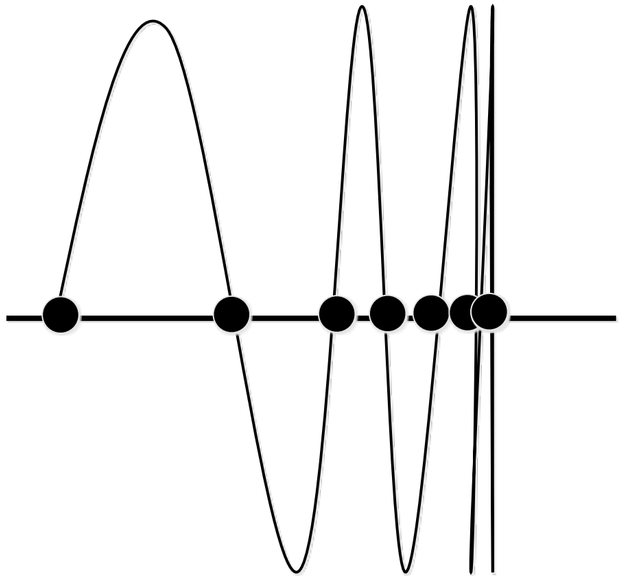
Multiple root:
Very difficult to find



Singularity:
Brackets don't
"bracket" a root



Pathological case:
Huge number of roots



Function f(x) = sin(1/x) ⟶ infinitely many roots

Root-Finding with Laguerre's Method

Not Relevant for Exam

- Advantage:
  - One of the very few "*sure-fire*" methods
  - An open method: does not require a bracket
- Limitations:
  - Works only for polynomials
  - Algorithm needs to perform calculations in complex numbers, even if all roots are real (and all coefficients)
  - Very little theory is known about its convergence behavior
    - If the root it converges to is a simple root, then the convergence order is (at least) 3
- Lots of empirical evidence that the algorithm (almost) always converges towards a root; and it does so from (almost) any starting value!

Not Relevant for Exam

Motivation for the Algorithm

- Given: the polynomial $$P(x) = (x - x_1)(x - x_2)\dots(x - x_n) \tag{0}$$

  where the x_i are the, possibly complex, unknown roots

- From that, we can derive the following equations:

$$\ln|P(x)| = \ln|x - x_1| + \ln|x - x_2| + \dots + \ln|x - x_n|$$

$$\frac{d}{dx}\ln|P(x)| = \frac{1}{x - x_1} + \dots + \frac{1}{x - x_n} = \frac{P'(x)}{P(x)} =: G \tag{1}$$

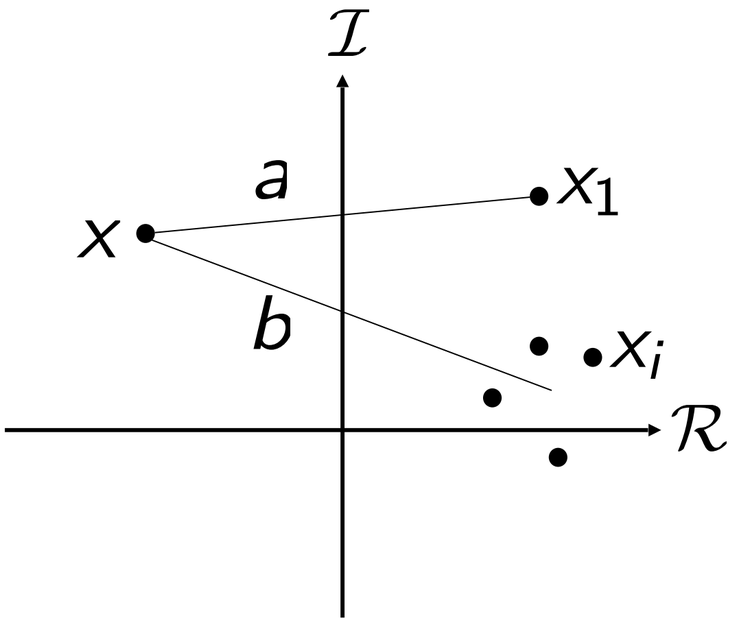
$$\frac{d^2}{dx^2}\ln|P(x)| = -\frac{1}{(x - x_1)^2} - \dots - \frac{1}{(x - x_n)^2}$$

$$= \frac{P''(x)}{P(x)} - \left(\frac{P'(x)}{P(x)}\right)^2 =: -H \tag{2}$$

- Let x be our current approximation of a root, w.l.o.g. root x_1

$$x - x_1 = a$$

- Denote distance



- Make a "drastic" assumption:
  - Assume, distance to all other roots is

$$x - x_i \approx b, \quad i = 2, 3, \ldots, n$$

- Then, we can write (1) & (2) like this:

$$G \approx \frac{1}{a} + \frac{n-1}{b} \qquad (3)$$

$$H \approx \frac{1}{a^2} + \frac{n-1}{b^2} \qquad (4)$$

Not Relevant for Exam

- Plug (4) into (3) and solve for a :

$$a \approx \frac{n}{G \pm \sqrt{(n-1)\left(nH - G^2\right)}} \tag{5}$$

  - Compute G and H from

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

$$P'(x) = a_1 + 2a_2 x + 3a_3 x^2 \ldots + n a_n x^{n-1}$$

$$P''(x) = 2a_2 + 3 \cdot 2 \cdot a_3 x \ldots + n \cdot (n-1) a_n x^{n-2}$$

  - Choose sign in front of sqrt such that $|a|$ becomes minimal
  - Remark: discriminant under sqrt can become negative
    $\longrightarrow a$ can become complex

- New approximation of root x_1 is

$$x_1 = x - a$$

The Algorithm

Not Relevant for Exam

- Strong recommendation: try to use code from *Numerical Recipes*
- For ray-tracing: have to compute all roots!
  - When first root is found, factor it out of polynomial
  - Find next root of smaller polynomial, repeat Laguerre n times

choose 0-th approximation $x^{(0)}$

repeat

compute $$G = \frac{P'(x^{(k)})}{P(x^{(k)})}$$

$$H = G^2 - \frac{P''(x^{(k)})}{P(x^{(k)})}$$

compute $$a = \frac{n}{G \pm \sqrt{(n-1)\,(nH - G^2)}}$$

let $x^{(k+1)} = x^{(k)} - a$

until ***a is*** **"small enough" or** ***k*** $\geq$ **max**

Metaballs

- Definitions:

  A potential field is described by a potential field function, e.g. $p(r) = \frac{1}{r^2}$ where $r = r_1(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_1\|$.

  An isosurface = set of all points that have the same potential value.

- The sphere's surface is thus

$$K = \{\mathbf{x} \mid p(r(\mathbf{x})) = \tau\}$$

  - τ is called threshold or isovalue

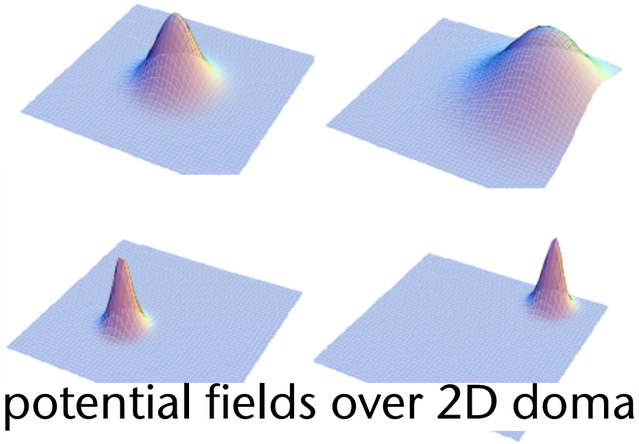
- More complex objects can be created by blending (superposition) of several potential fields
  - Simplest blending is (weighted) summation of the potential fields:

$$P(\mathbf{x}) = \sum_{i=1}^{n} a_i p_i(\mathbf{x})$$

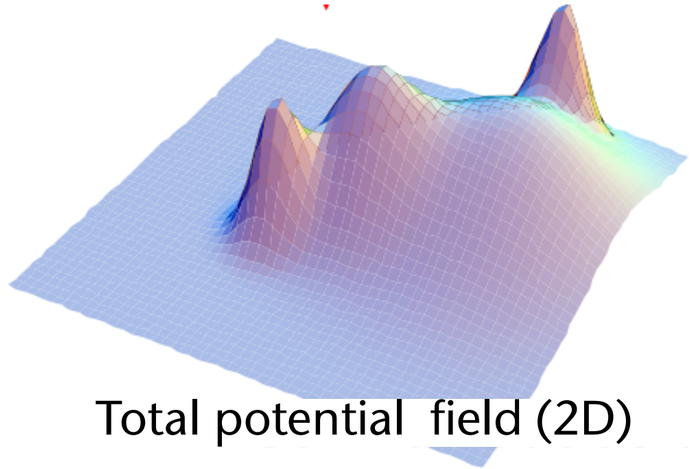
where p_i = potential field of i-th skeleton point $\mathbf{x}_i$

  - The set of points $\mathbf{x}_i$ is called the *skeleton*,
    P is the total potential, the a_i determine the influence (= kind of "field's force")
  - Negative a_i can "carve out" material (e.g., for making holes)
- Note: the potential field is, theoretically, defined throughout the *whole* space

Examples

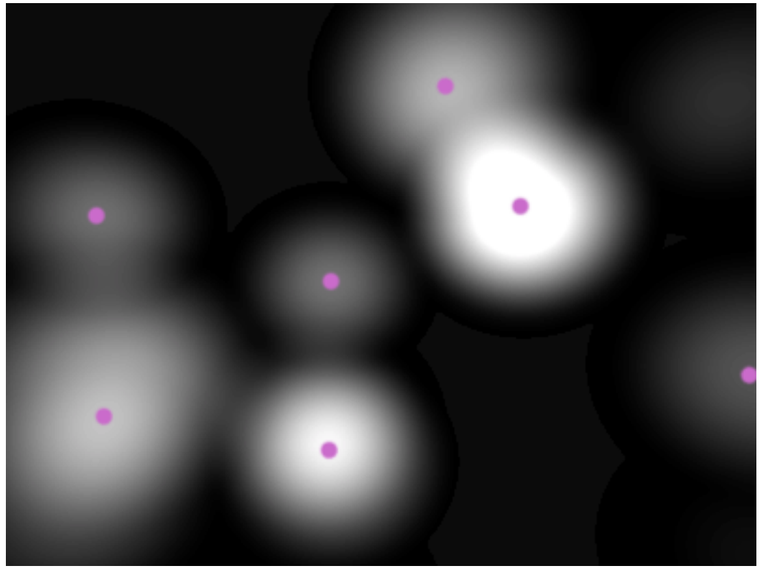


Individual potential fields over 2D domain

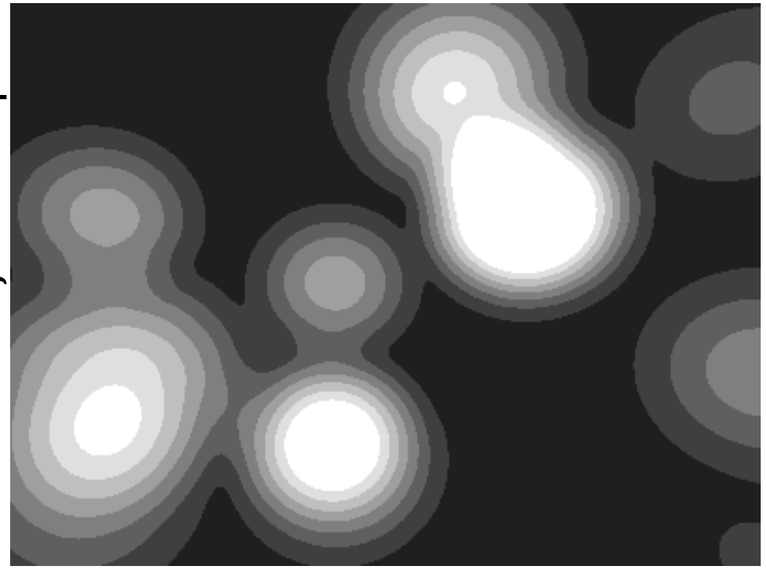


Total potential field (2D)

Total potential field (2D)



Potential objects shapes



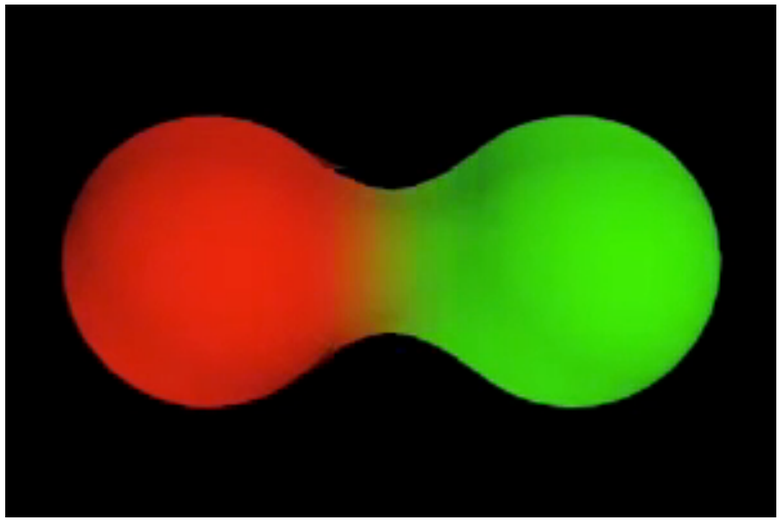
Generalized Metaballs

- Ingredients for definition of metaballs objects:
  distance function, potential function, skeleton points, weights
- Definition: a metaballs object is defined as the isosurface

$$\mathcal{S} = \{ \mathbf{x} \mid \mathbf{x} \in \mathbb{R}^3, \; P(\mathbf{x}) = \sum a_i p(d_i(\mathbf{x})) = \tau \}$$

  with p = some potential function,
  d_i = distance function to i-th skeletal point $\mathbf{x}_i$

- Examples for 2 skeleton points:



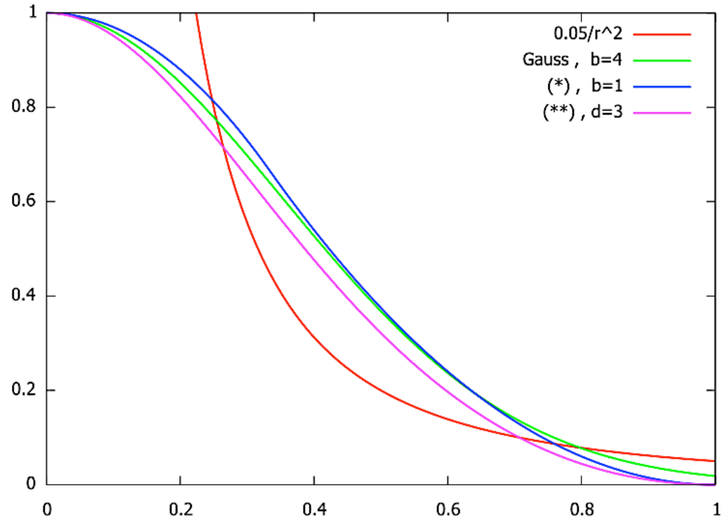
Other Functions for Radial Potential Fields (Transfer Functions)

$$p_i(r) = e^{-br^2} \tag{1}$$

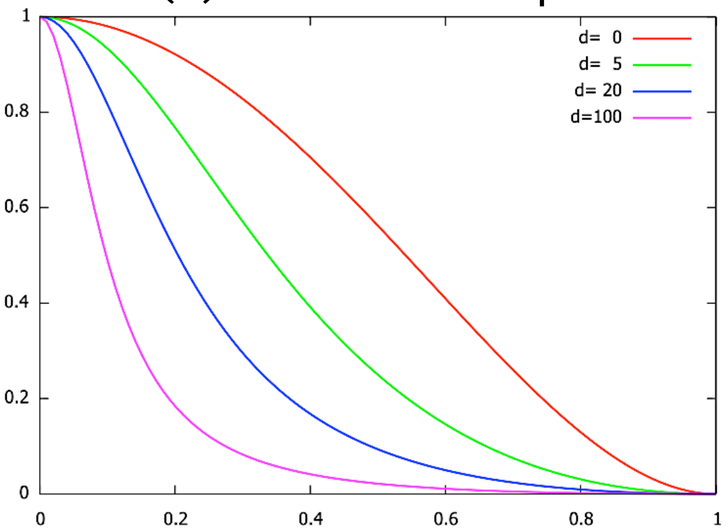
$$p(r) = \begin{cases} 1 - 3\frac{r^2}{b^2} & , \ r \leq \frac{1}{3}b \\ \frac{3}{2}(1 - \frac{r}{b})^2 & , \ \frac{1}{3}b \leq r \leq b \\ 0 & , \ r > b \end{cases} \tag{2}$$

$$p(r) = \begin{cases} \frac{r^4 - 2r^2 + 1}{1 + dr^2} & , \ r \leq 1 \\ 0 & , \ r > 1 \end{cases} \tag{3}$$

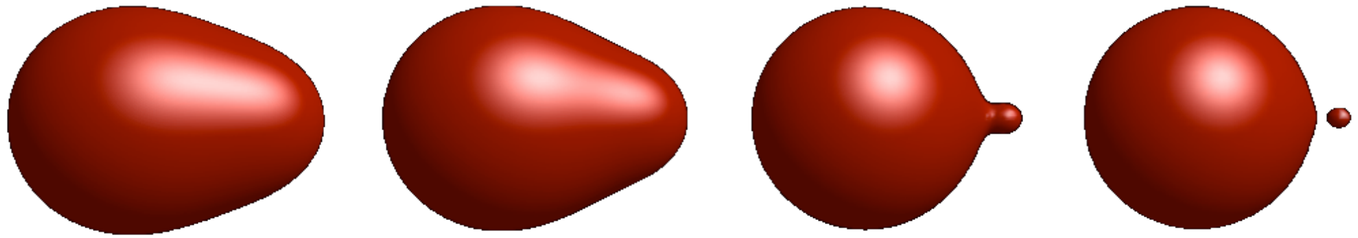
Comparison



Function (3) with different parameters

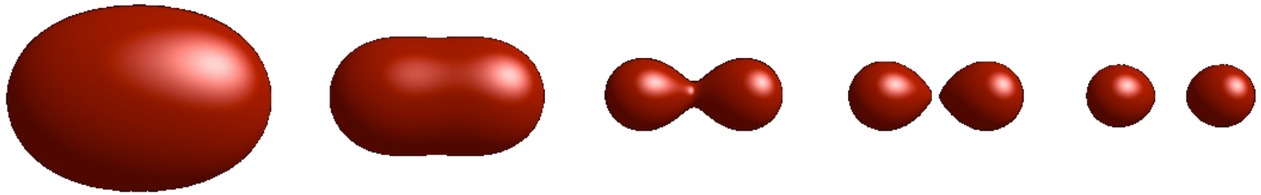


- Effect of the variation of the parameter *d* in function (3):



d is fixed for the left skeleton point, *d* = 10 ... 2000 for the right skeleton point

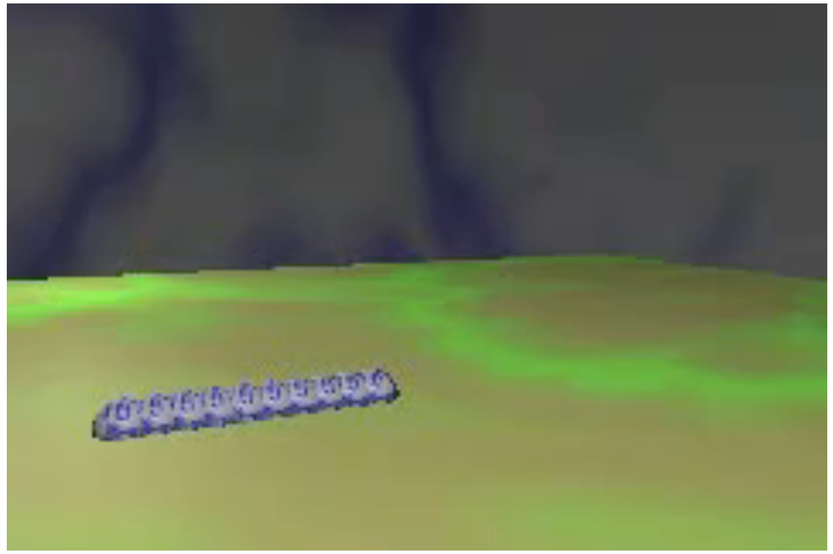
- Effect of varying the isovalue τ:



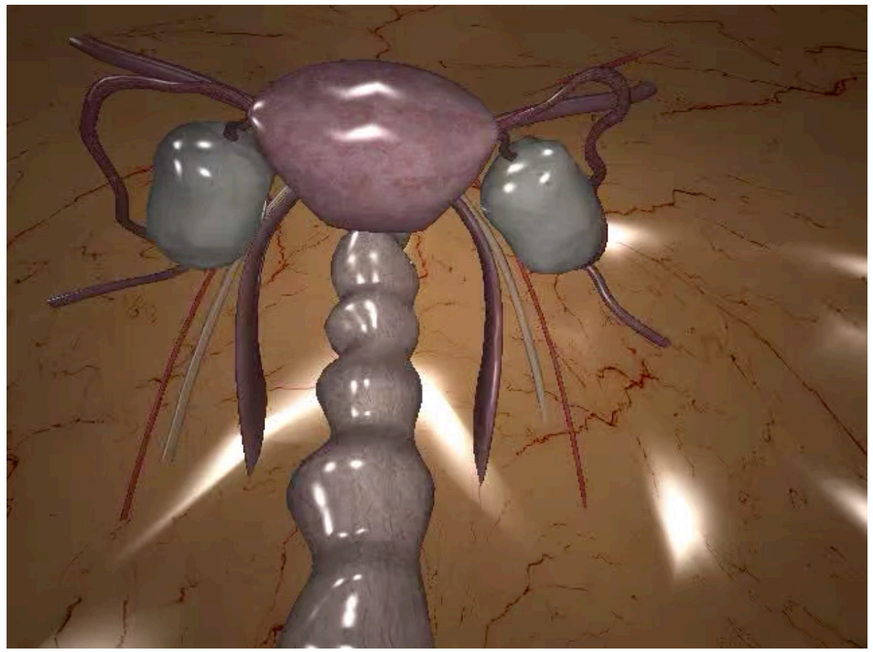
- Many names for this kind of modeling methodology: "metaballs", "soft objects", "blobs", "blobby modeling", "implicit modeling", ...

Deformable Models

- With implicit modeling (metaballs), it is easy to create and animate deformable "blob-like" objects:
  - Animate (move) the skeleton points
  - Modify parameters a_i, d, ...
  - Modify the iso-value τ



Brian Wyvill
http://pages.cpsc.ucalgary.ca/~blob/animations.html

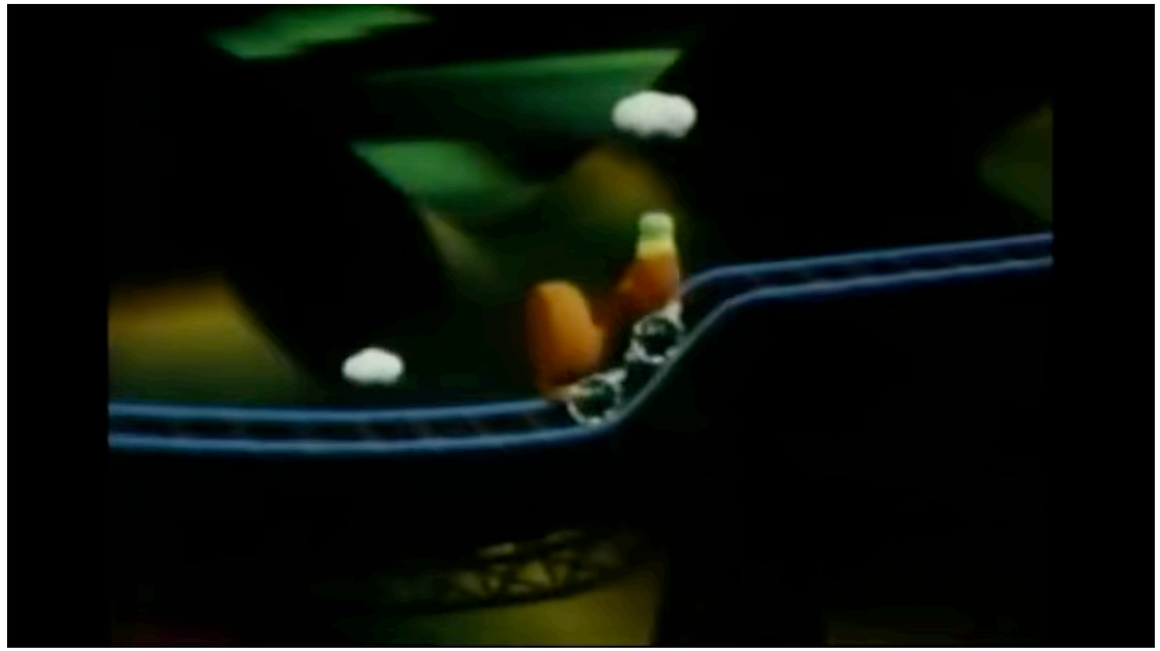


Frédéric Triquet
http://www2.lifl.fr/~triquet/implicit/video/

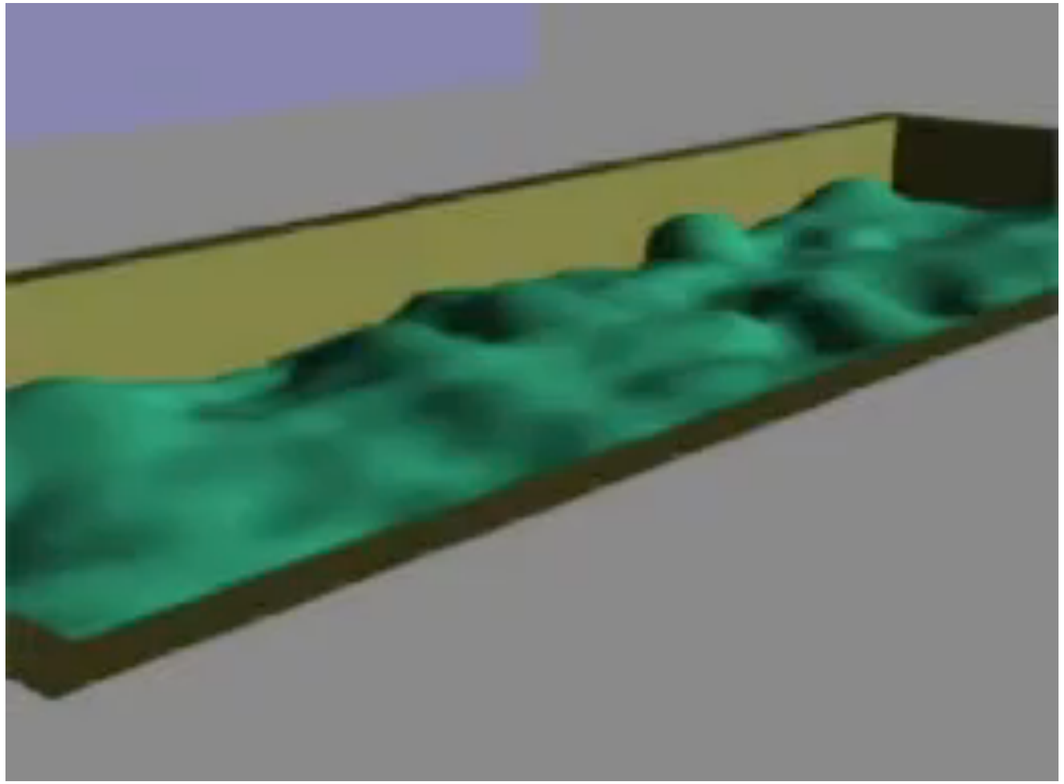
"The Wyvill Brothers"

Geoff

Brian



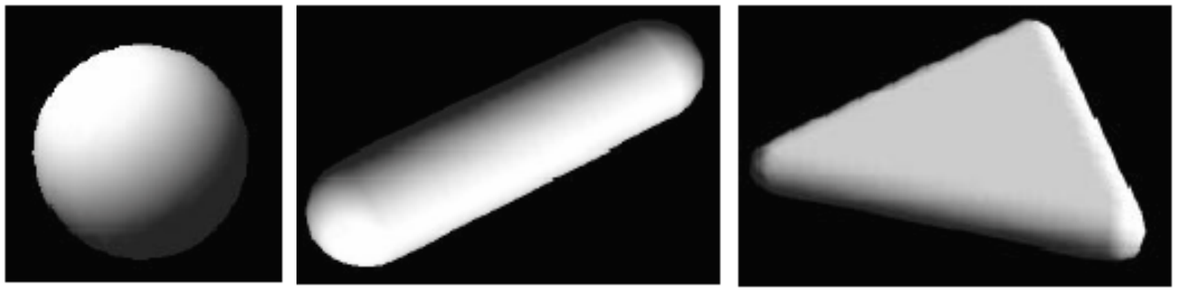
"The Great Train Rubbery" — Siggraph 1986

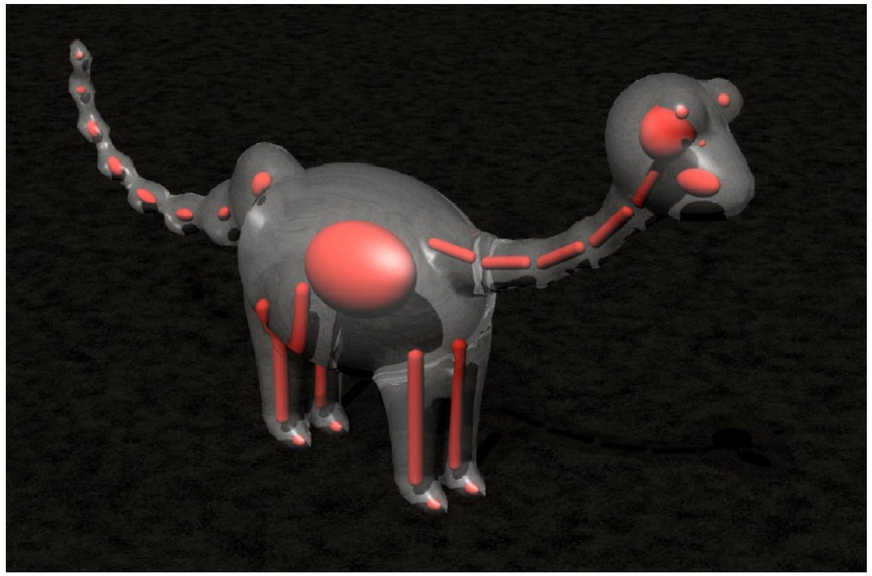


"Soft"

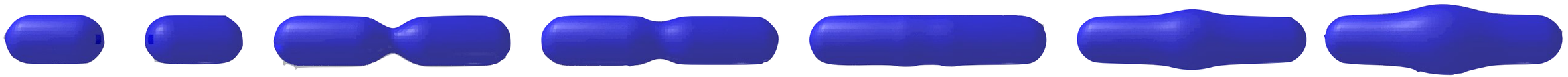
Generalization / Variants

- Points are the simplest kind of primitive for metaballs skeletons; analogously, we can use lines, polygons, ellipsoids, etc.:





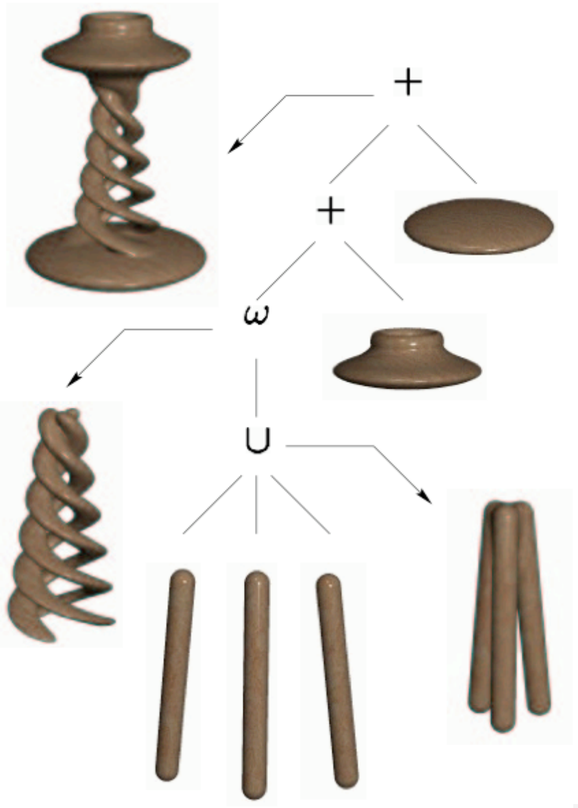
- Problem with higher primitives at junctions: bulges



- Other blending functions:

$$P_{\cup}(\mathbf{x}) = \max\{p_1(\mathbf{x}), p_2(\mathbf{x})\} \qquad P_{\cap}(\mathbf{x}) = \min\{p_1(\mathbf{x}), p_2(\mathbf{x})\}$$

- A tree of "blending" operations — the "BlobTree":
  - Combination of CSG and metaballs
  - Nodes can contain operations on pairs of children
  - Or, nodes contain a transformation of its only child



The Normal on Implicit Surfaces

- The normal in a point **x** on the implicit surface $f(\mathbf{x})$:

$$\mathbf{n}(\mathbf{x}) = \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x}(\mathbf{x}) \\ \frac{\partial f}{\partial y}(\mathbf{x}) \\ \frac{\partial f}{\partial z}(\mathbf{x}) \end{pmatrix} \approx \begin{pmatrix} f(x+\varepsilon, y, z) - f(\mathbf{x}) \\ f(x, y+\varepsilon, z) - f(\mathbf{x}) \\ f(x, y, z+\varepsilon) - f(\mathbf{x}) \end{pmatrix}$$

$$\approx \begin{pmatrix} f(x+\varepsilon, y, z) - f(x-\varepsilon, y, z) \\ f(x, y+\varepsilon, z) - f(x, y-\varepsilon, z) \\ f(x, y, z+\varepsilon) - f(x, y, z-\varepsilon) \end{pmatrix}$$

Note: Nabla is just a shorthand notation for

$$\nabla f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x})$$

- Warning: **n**(**x**) points to the *inside* of the implicit surface *P* !
  (assuming *P*(inside) > *P*(outside))
  Usually, you want **n** to point to the *outside*, so you want $\mathbf{n}(\mathbf{x}) = -\nabla f(\mathbf{x})$

- Gradient of metaballs functions:

$$\mathbf{n}(\mathbf{x}) = \nabla P(\mathbf{x}) = \sum_i a_i \nabla p_i(\mathbf{x})$$

  - The only tools you need is the chain rule (Kettenregel) and the fact that

$$\frac{\partial}{\partial \mathbf{x}} \|\mathbf{x} - \mathbf{x}_i\| = 2 \frac{(\mathbf{x} - \mathbf{x}_i)}{\|\mathbf{x} - \mathbf{x}_i\|}$$

Why does the gradient point in the direction of the normal?

- In other words: why is the gradient in point **x** perpendicular to the tangent plane at point **x** on the implicit surface?
- From math, you know: the directional gradient in an (arbitrary) direction **u** is

$$\frac{\partial}{\partial \mathbf{u}} f(\mathbf{x}_0) = \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_0) \cdot \mathbf{u}$$

- The scalar product of two vectors is proportional to the cosine, so

$$\frac{\partial}{\partial \mathbf{u}} f(\mathbf{x}_0) = \left| \frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_0) \right| \cdot |\mathbf{u}| \cdot \cos\theta$$

where Θ = angle between the gradient $\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_0)$ and **u**

- Now, assume $|\mathbf{u}| = 1$ and consider all possible $\mathbf{u}$'s, i.e., all vectors starting at $\mathbf{x}_0$ and pointing to the unit sphere around $\mathbf{x}_0$
- When does $\frac{\partial}{\partial \mathbf{u}} f(\mathbf{x}_0)$ attain its maximum value?

  $\longrightarrow$ when $\cos\Theta = 1$, i.e., $\mathbf{u}$ points in the direction of the gradient

  $\longrightarrow$ the gradient $\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}_0)$ is indeed the direction of maximal change of f
- When is $\frac{\partial}{\partial \mathbf{u}} f(\mathbf{x}_0) = 0$?

  $\longrightarrow$ if $\cos\Theta = 0$, i.e., when $\mathbf{u}$ is perpendicular to the gradient

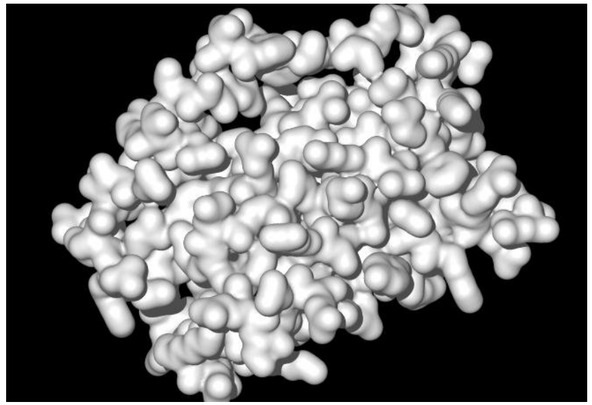
  $\longrightarrow$ all these $\mathbf{u}$'s denote directions where f does not change locally

  $\longrightarrow$ this is the tangent plane to the surface at point $\mathbf{x}_0$

  $\longrightarrow$ the gradient is indeed the *normal* at point $\mathbf{x}_0$

Applications of Metaballs

Digital art (Prof. Kawaguchi)



Molecule visualization

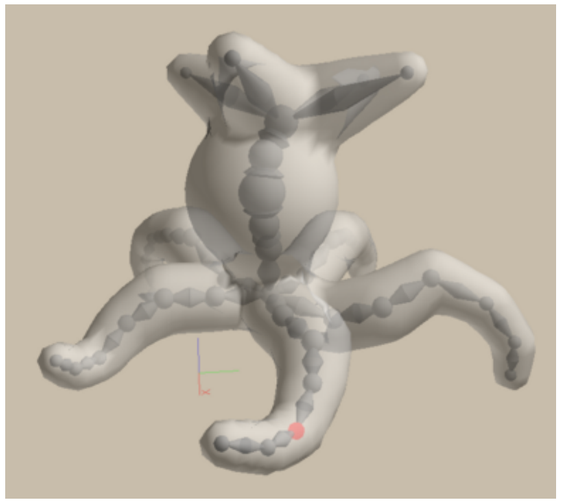
Rendering fluids
[Müller et al. 2003]

Application in Games: Metaballs in *Spore*

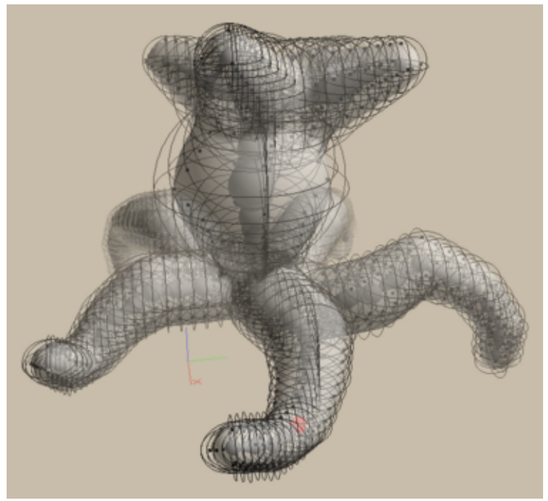
[Chris Hecker, 2014]

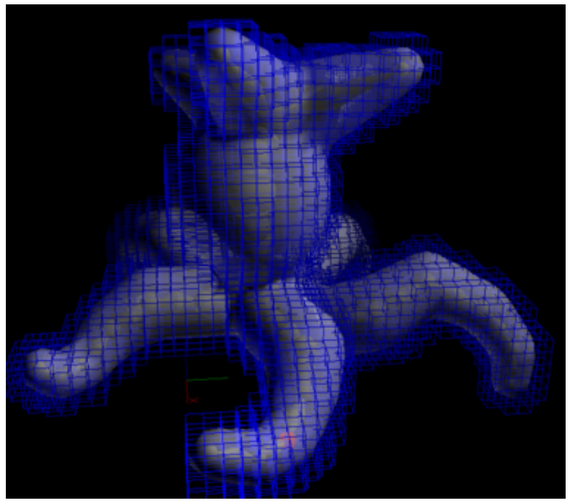
- Special challenge: all characters are designed by the player at runtime ⟶ no fixed building blocks, no parameterized characters

Skin and skeleton



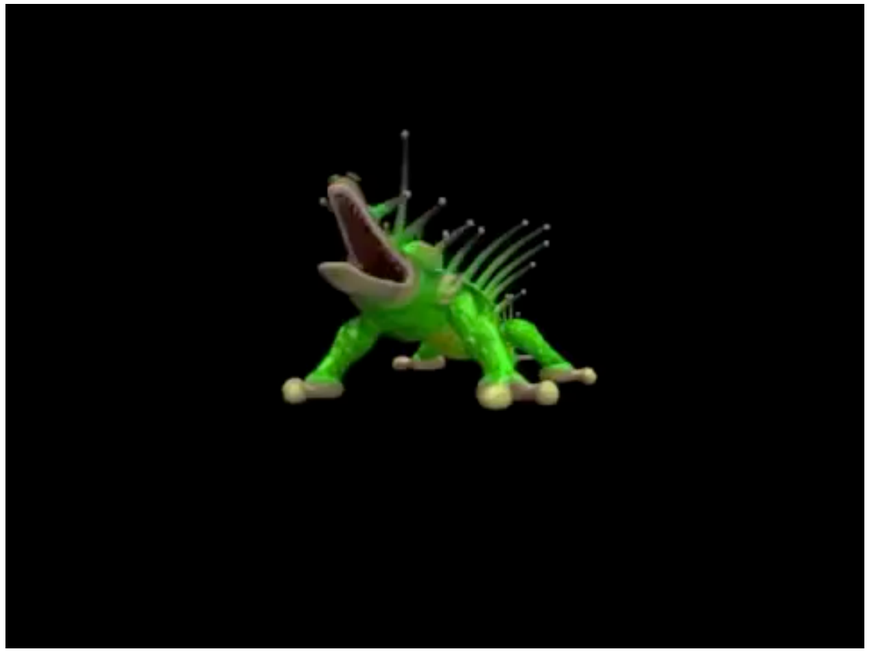
Metaballs





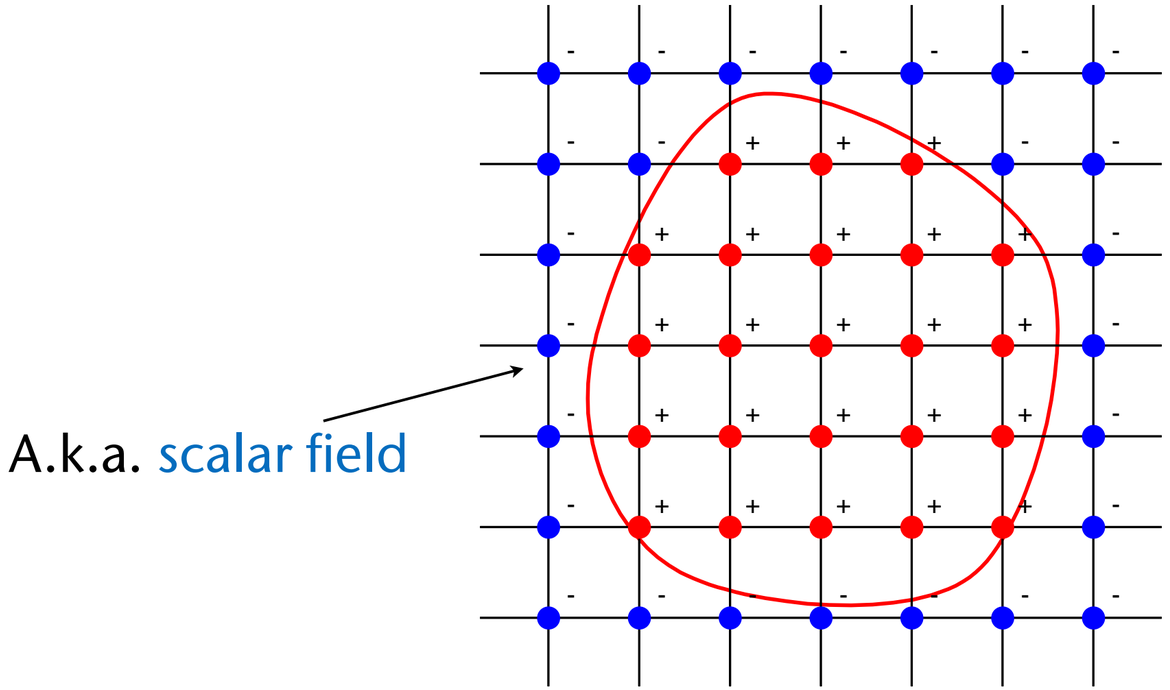
Voxels that contain the isosurface
(from marching cubes)

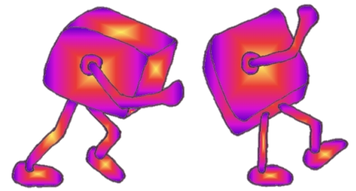
- Rendering uses a variant of Marching Cubes (later)
- Potential field function: $p(r) = \begin{cases} (\frac{r}{d} - 1)^4 & , \ r \leq d \\ 0 & , \ r > d \end{cases}$
- Some example creatures:



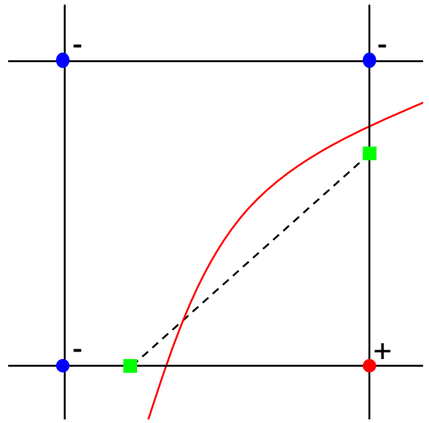
Marching Cubes: Polygonization of Implicit Surfaces

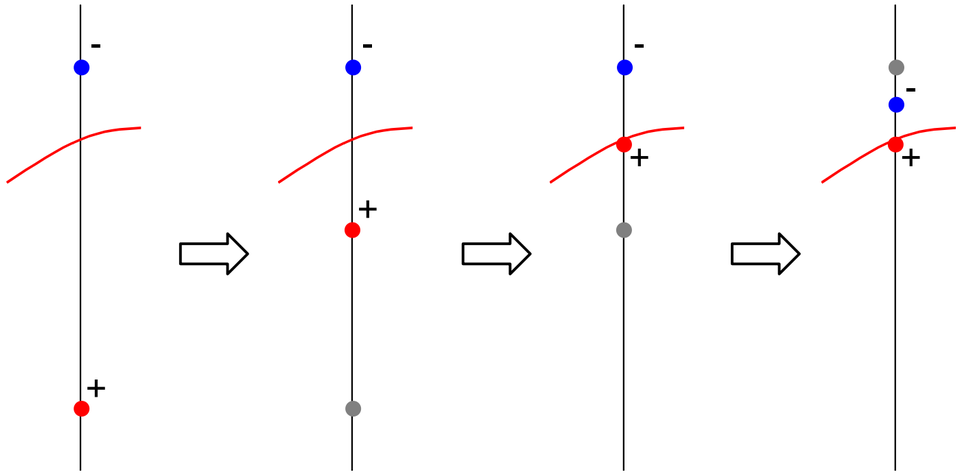
- Goal: polygonal mesh that closely "follows" the surface $F(x)=0$
- Idea: bracket roots by sampling space regularly
- Step 1: evaluate $F(x)$ at all grid node points
  - Note that (almost) every grid node is shared by 8 grid cells (voxels) $\longrightarrow$ store $F(x)$





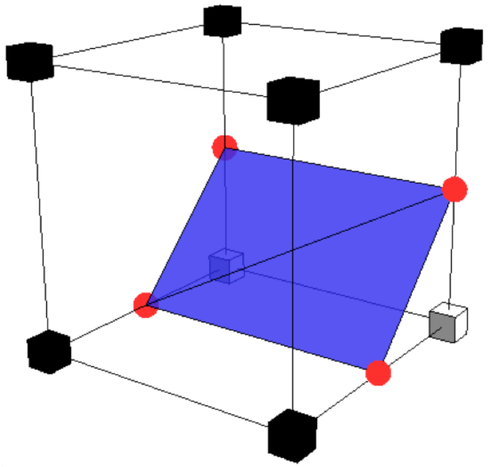
- Step 2: for all edges crossing the boundary, estimate the intersection point x where $F(\mathbf{x})=0$
  - Either use bilinear interpolation between the ⊕-node and the ⊖-node;
  - Or use root finding, if $F(\mathbf{x})$ is given analytically

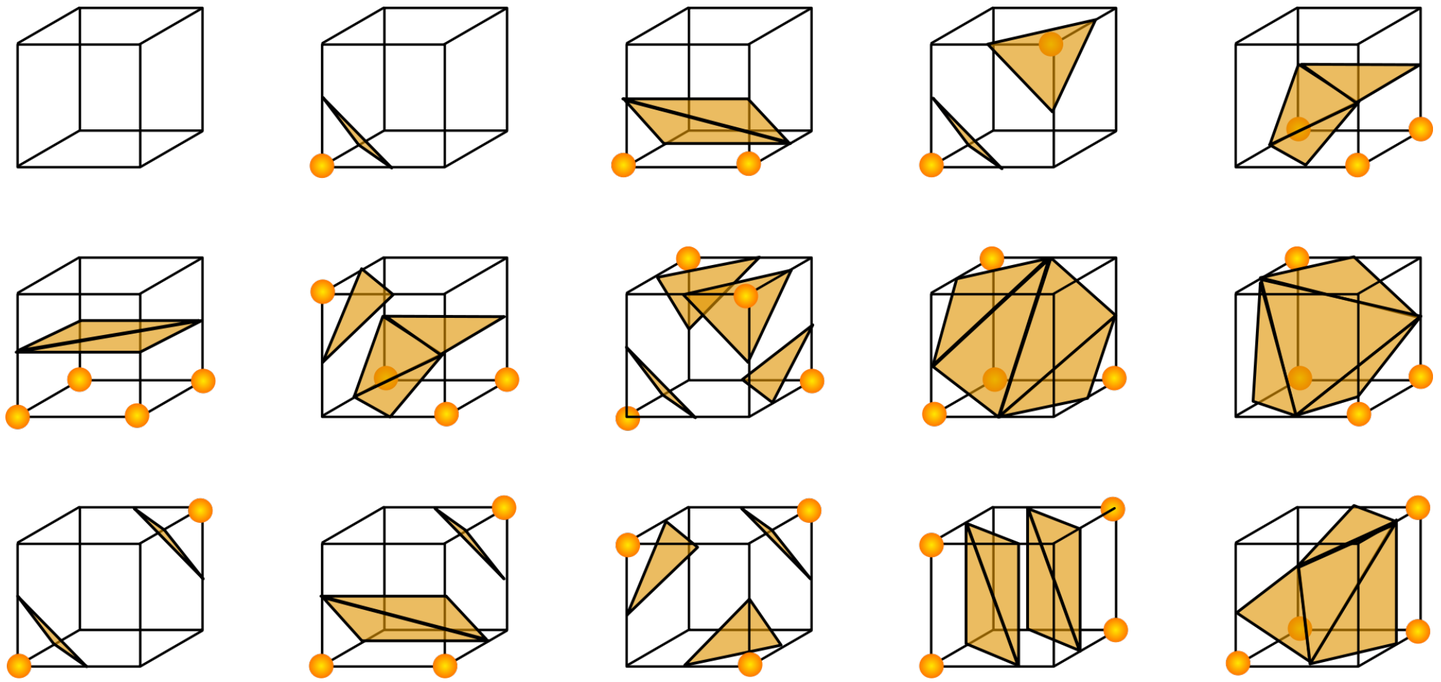




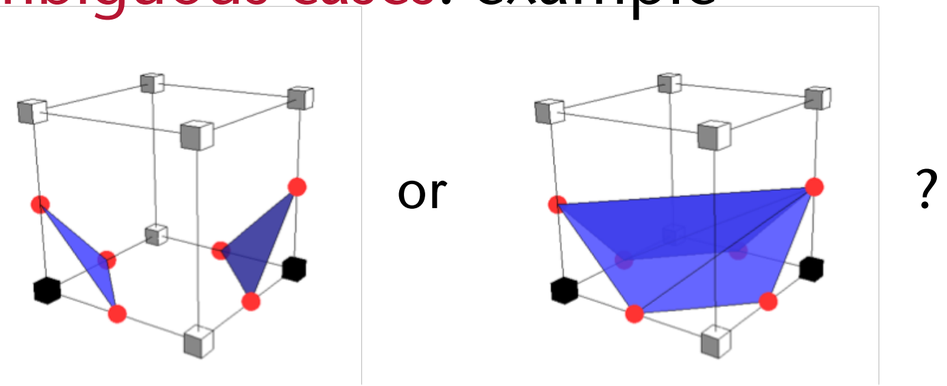
- Optim.: every edge is shared by 4 voxels
  ⟶ store the intersection pts with each edge

- Step 3: create polygons inside each "crossing" voxel
  - Voxel has 8 corners $\longrightarrow$ 28 cases of $\oplus$ and $\ominus$ corners
  - Vertex bit mask = index into LUT
  - There are 15 unique cases (mod rotation & sign flips):

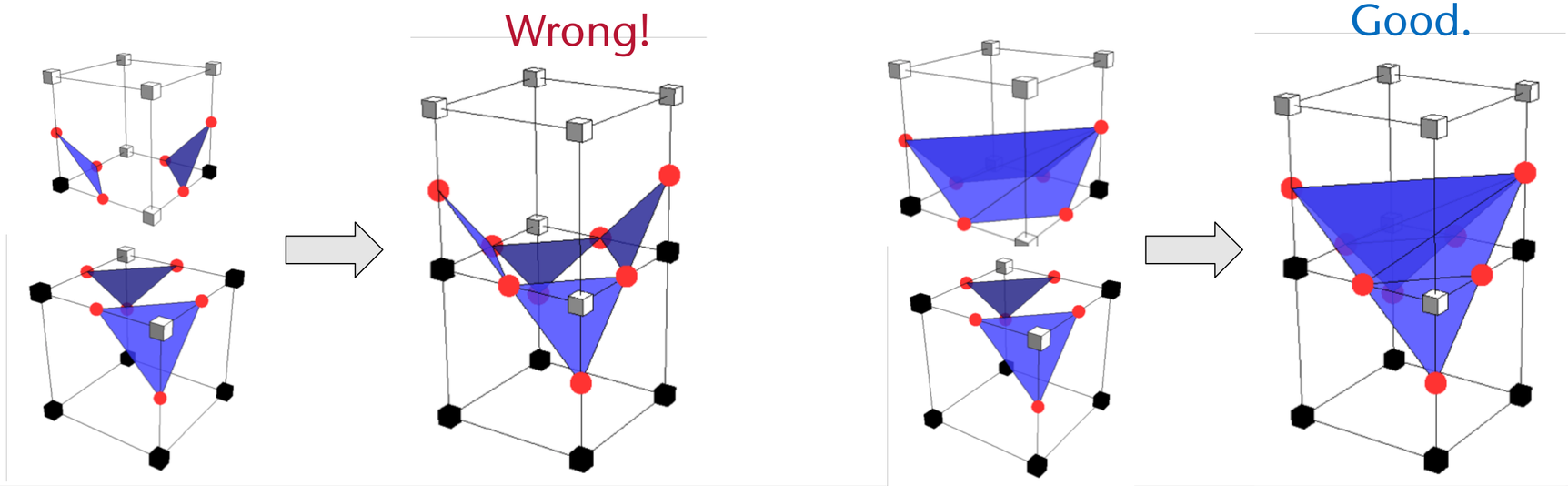




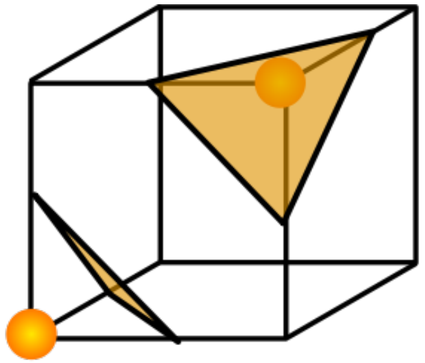
- There exist several ambiguous cases: example



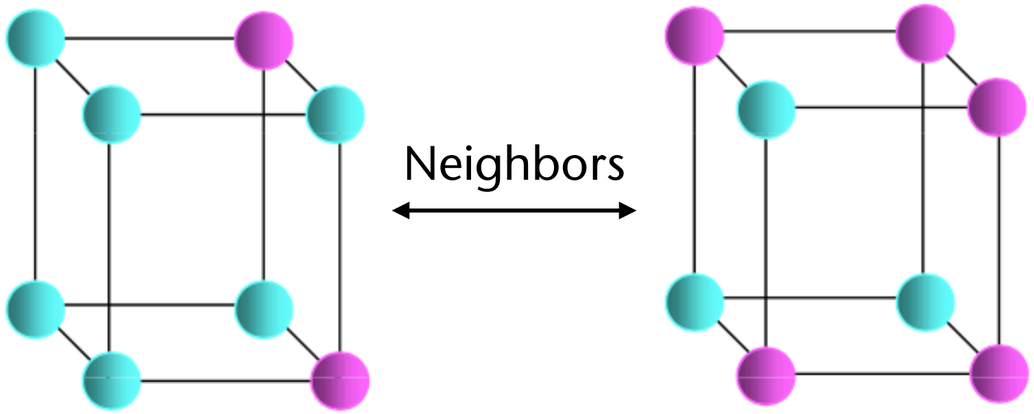
- Solution: note that final surface must be closed ⟶ look at neighbor cubes

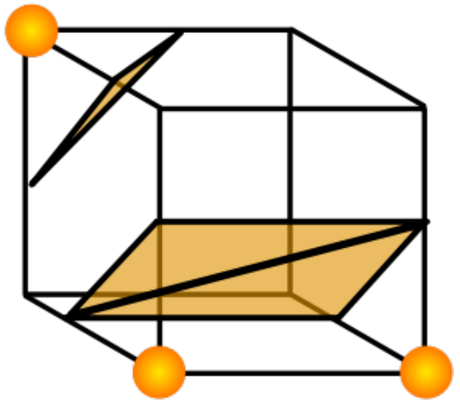


Another case

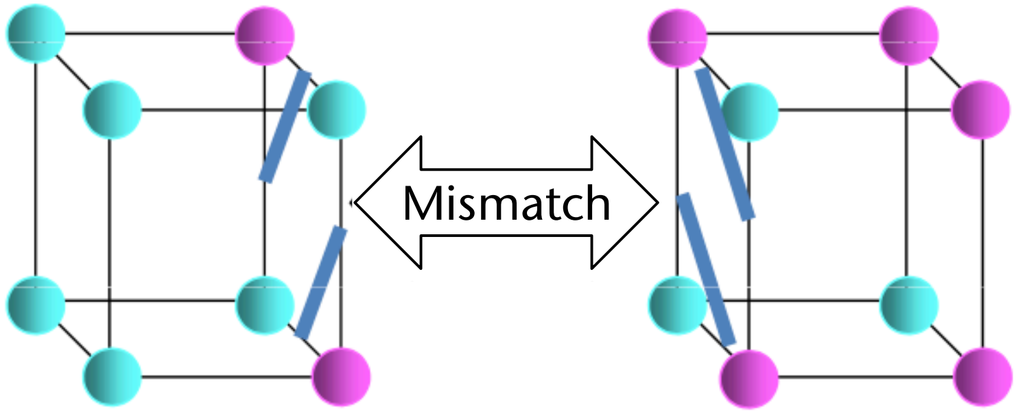


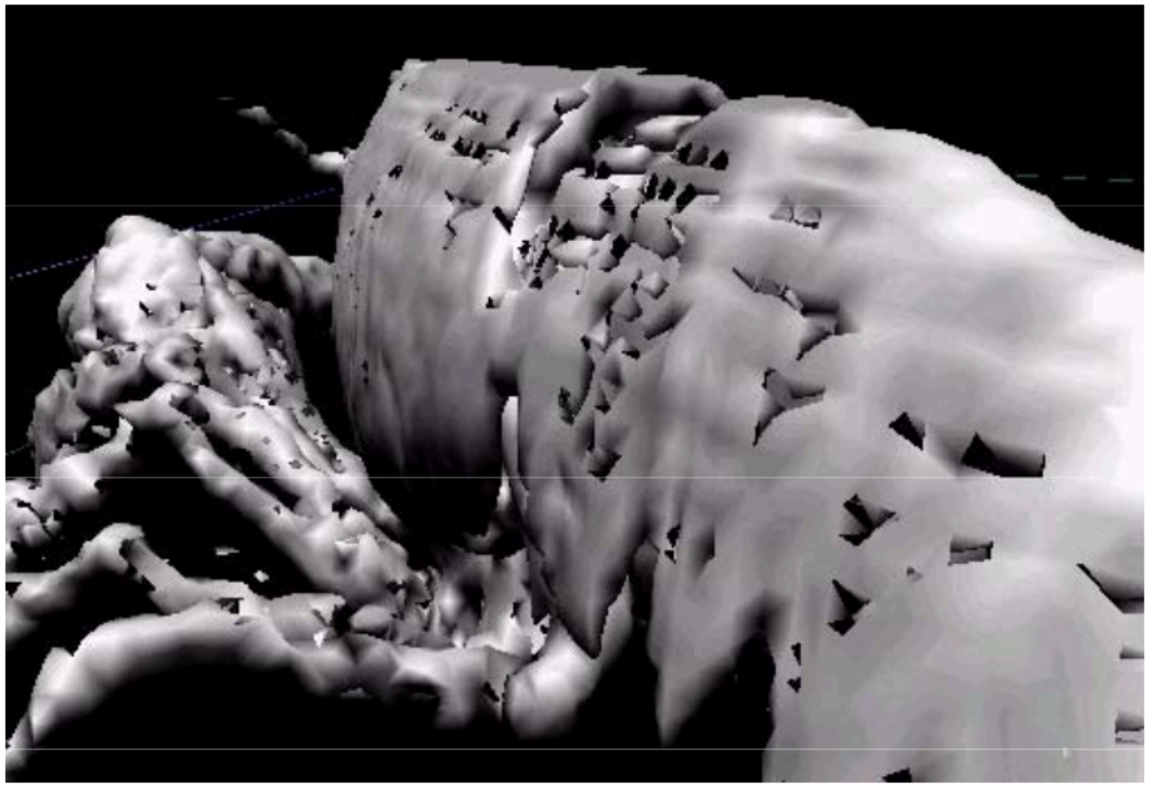
A template fitting the sign pattern



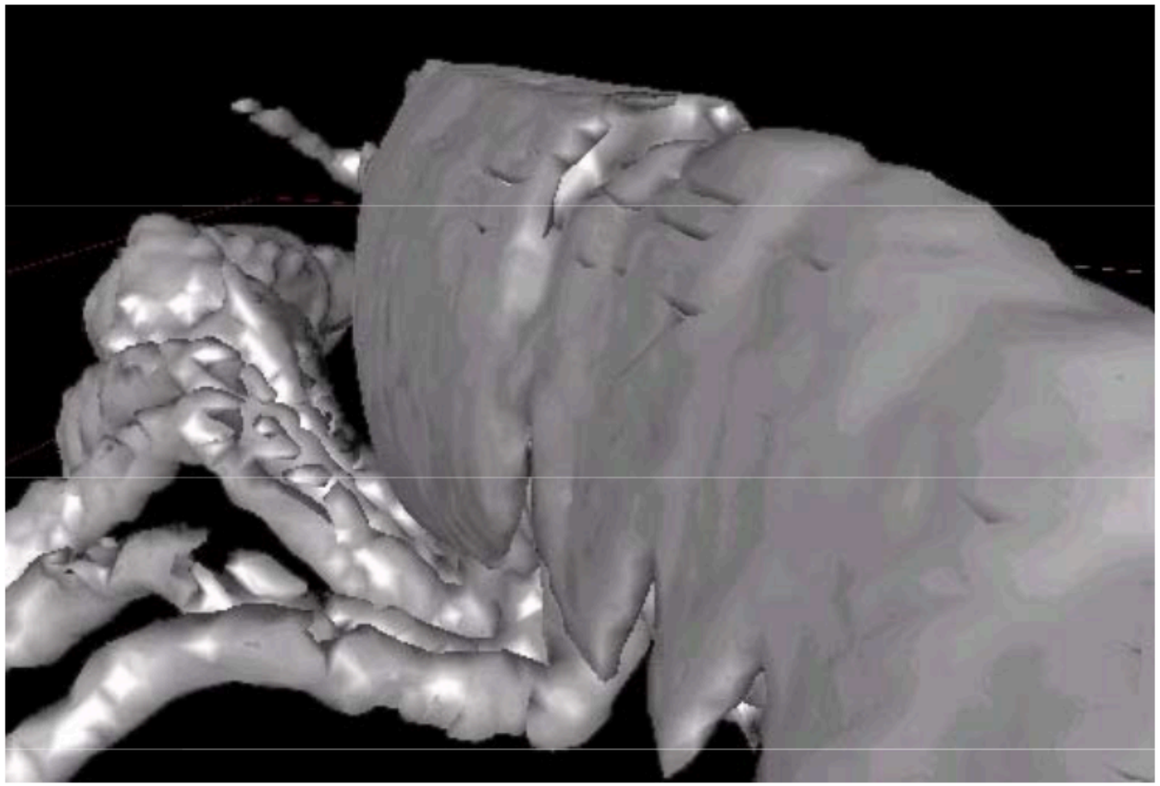


Template fitting the sign pattern (need to apply front/ back mirroring)





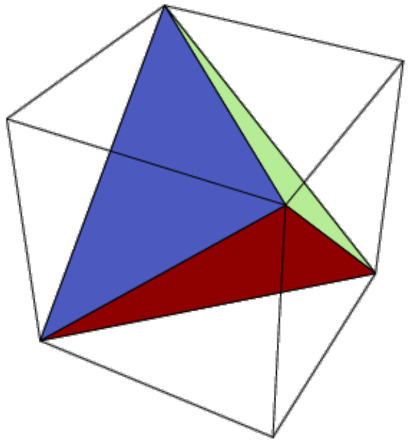
Ambiguity

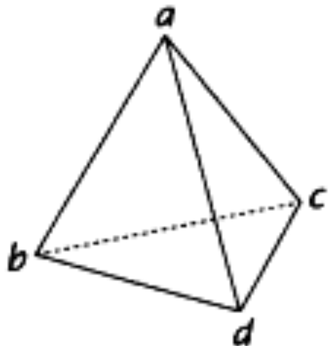


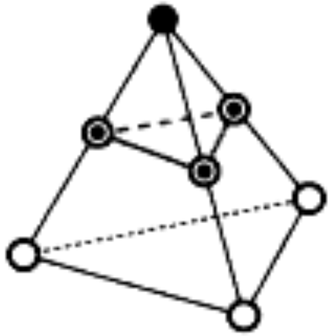
No ambiguity

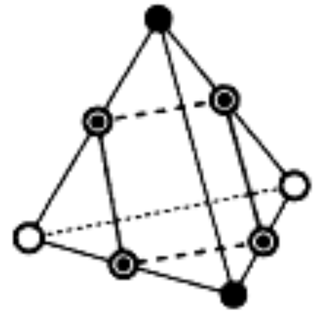
Marching Tetrahedra

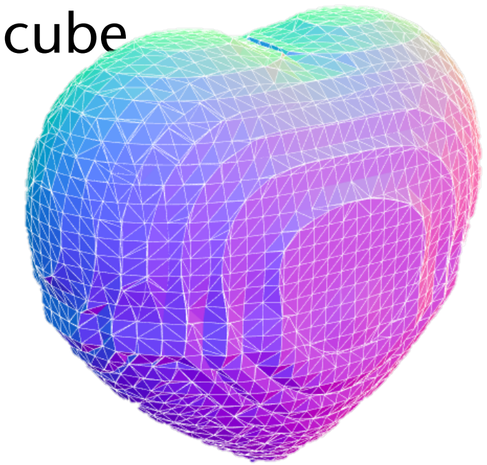
- A potential solution of the "ambiguity" problem:
  - Decompose the voxels into tetrahedra
  - Use tetrahedra instead of cubes for constructing the polygons
- A common decomposition is into 5 tetrahedra
  - Caveat: need to flip every other cube
- Marching Tetrahedra tends to produce meshes with lower quality (more long/thin triangles)



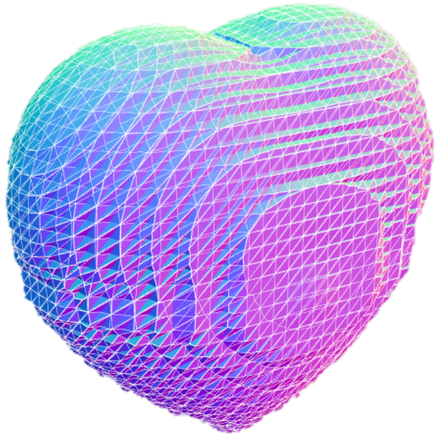






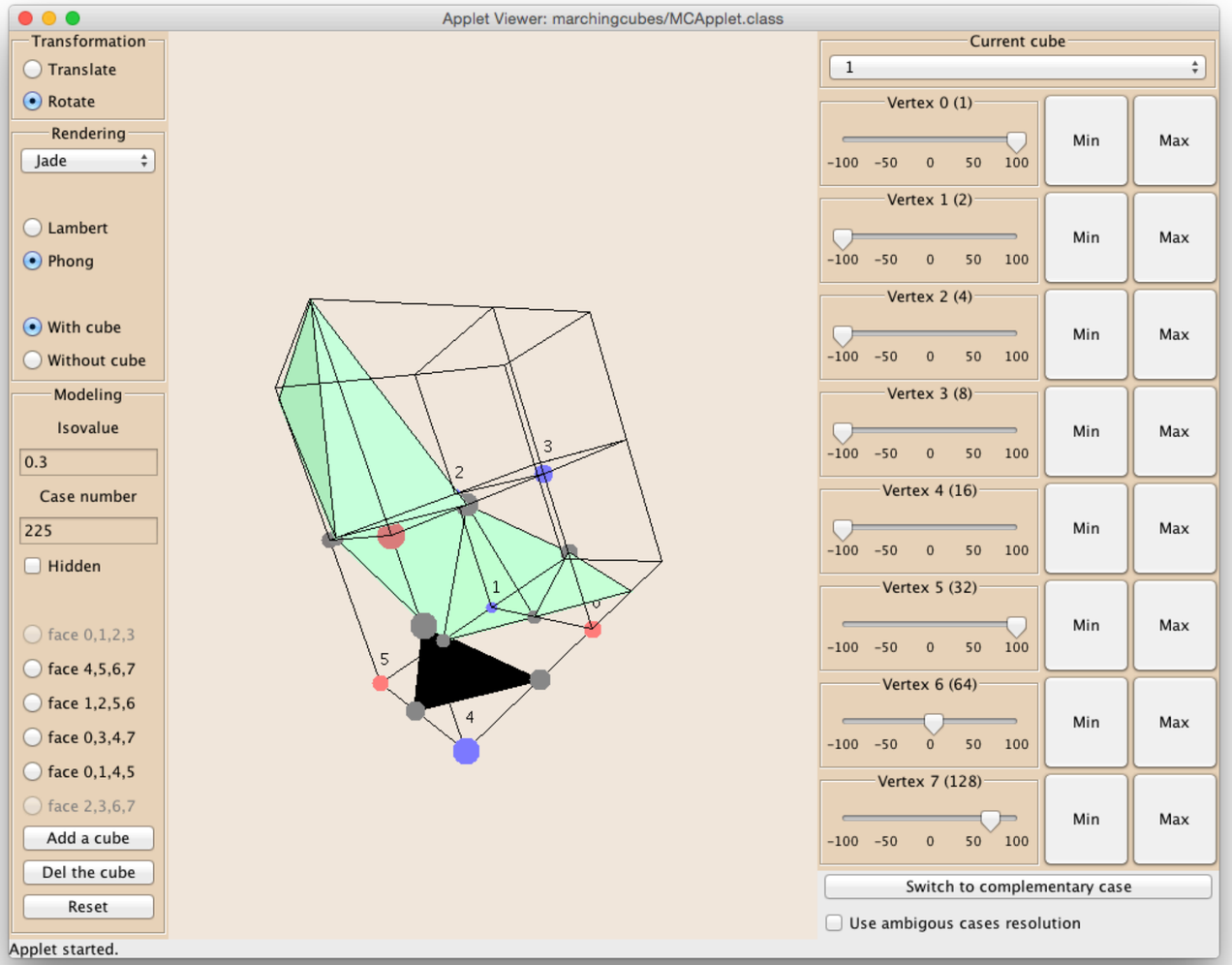


Marching Cubes



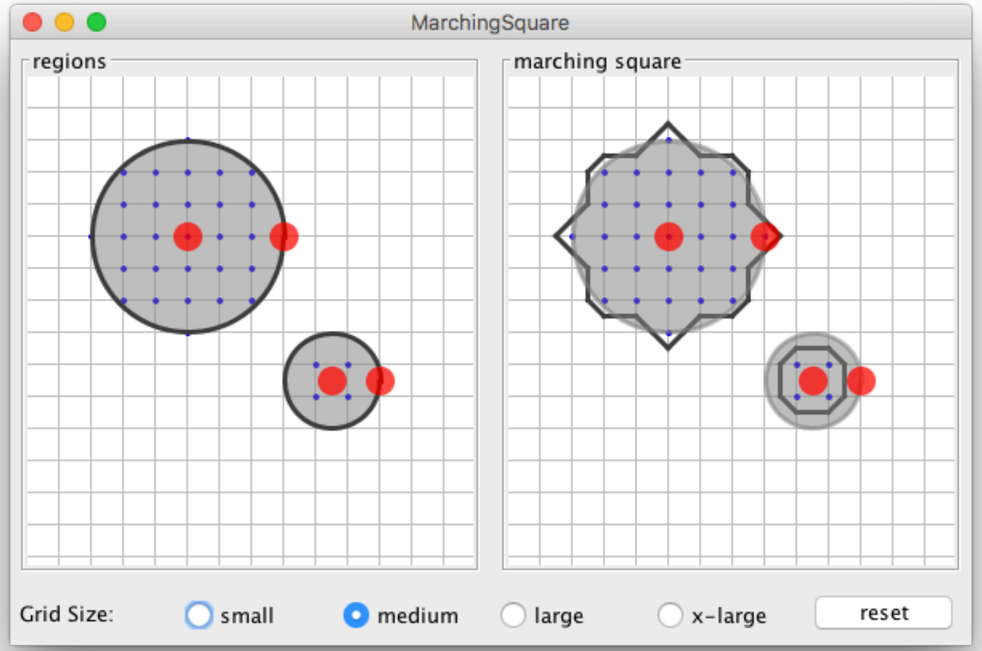
Marching Tetrahedra

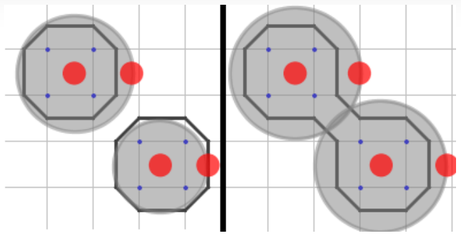
Demo



http://users.polytech.unice.fr/~lingrand/MarchingCubes/applet.html

Marching Squares Demo





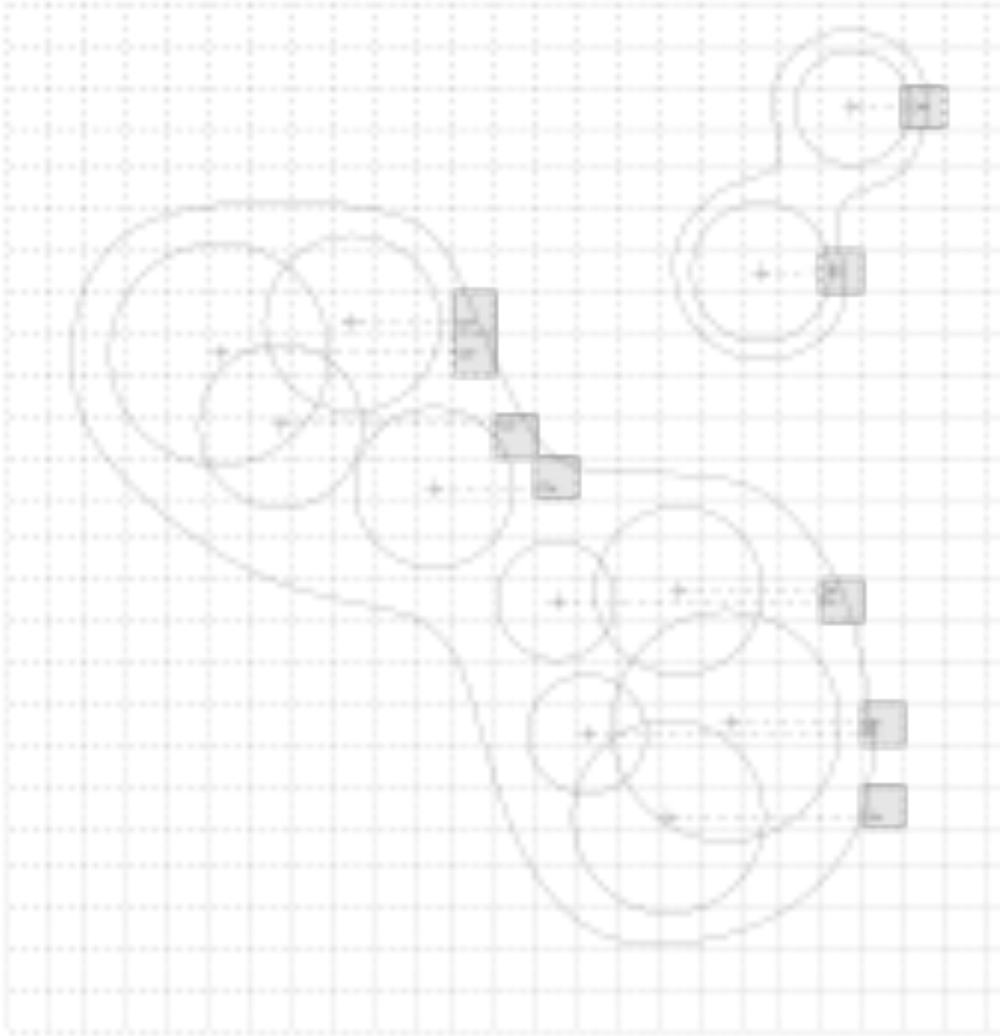
Ambiguous cases

Optimizations for Marching Cubes on Implicit Surfaces

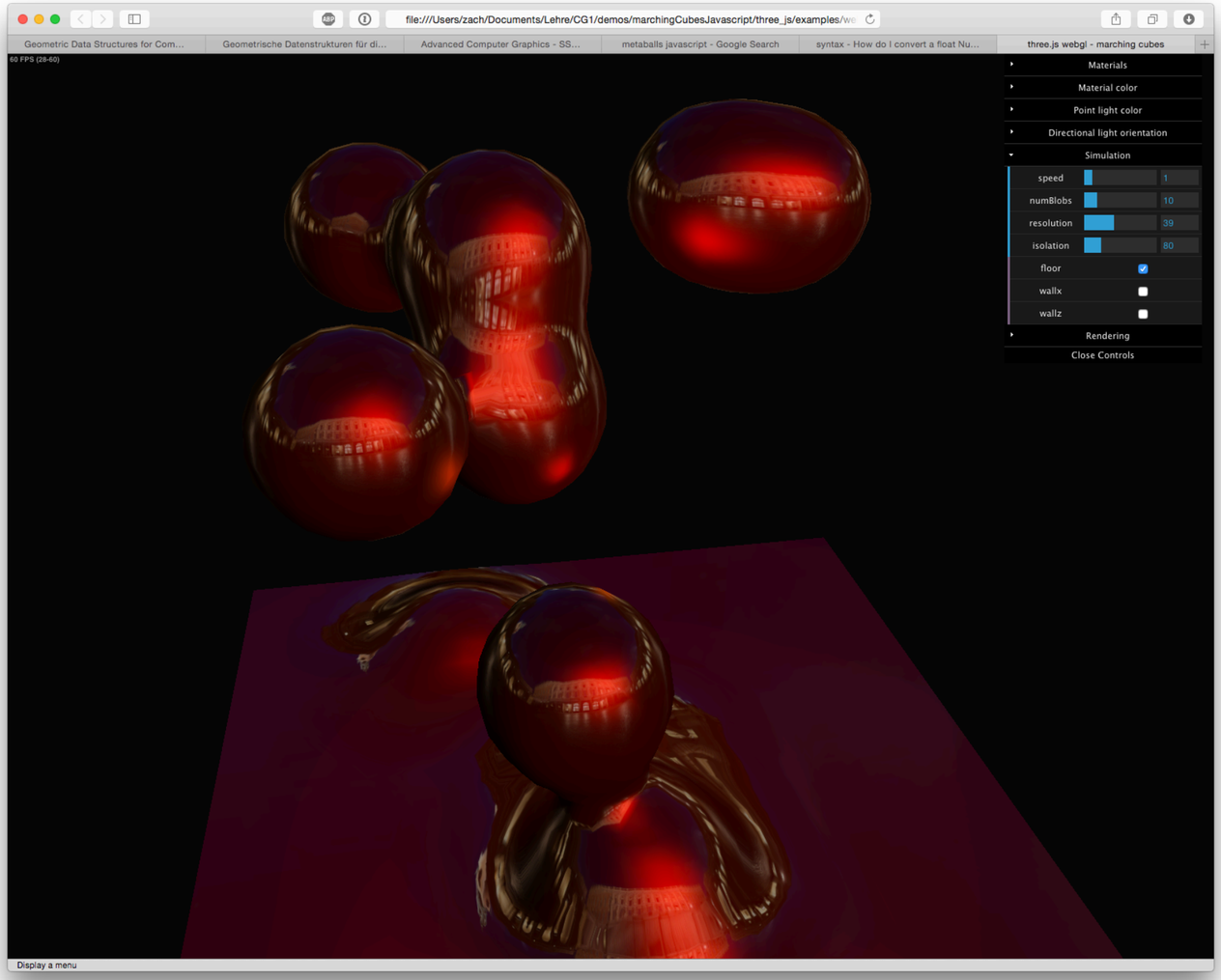
- Only use potential field functions with *finite* radius of influence ⟶ evaluation of overall implicit function *F* can omit all skeleton points whose distance from query point is larger than this radius
- Preprocessing: for all store nodes of 3D grid, store scalar *F*-values (3D array)
- Don't check every voxel, instead use octree to prevent visiting every voxel!
  - See "Computational Geometry" course ...
  - Works, of course, only for static metaballs skeletons (but isovalue is allowed to change!)

Tracking / Continuation Method

- Once a cube "on" the surface is known, there must be a neighbor cube also "on" the surface
- Algo: find a seed cube, follow/track the implicit surface
- Maintain list of cubes still to be visited
- Question: how to find seed & how to make sure no component of the implicit surface is missed!
- Solution: from every skeleton point, walk in x-direction until "crossing" cube is found; add these to list of cubes still to be visited



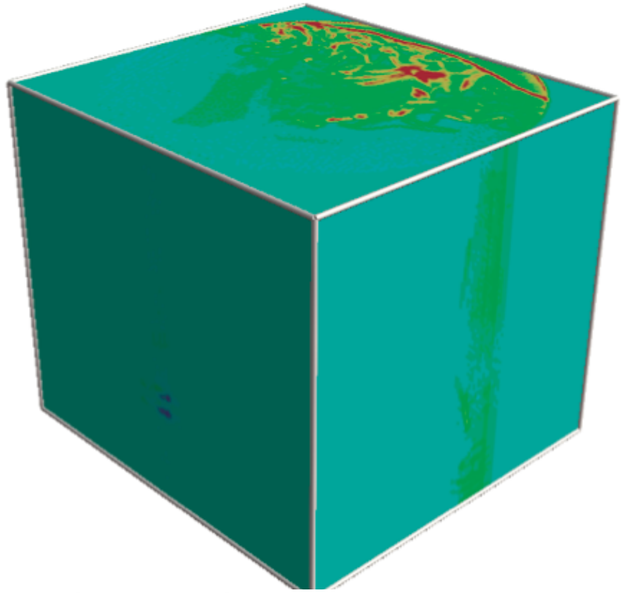
Another Metaballs Demo



http://threejs.org/

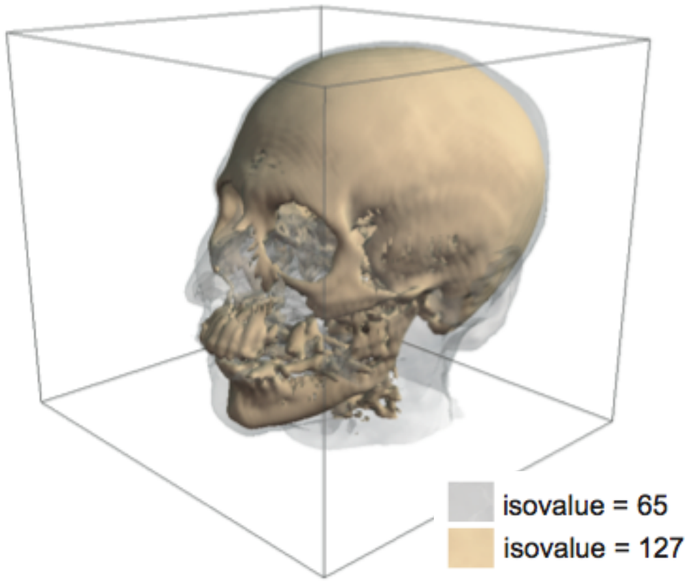
Remark: Marching Cubes for Volume Data

- Marching cubes is the standard method for segmentation of volume data (e.g., CT or MRT data)
- No distance field here, but densities at the voxel nodes
- Extension to continuous density field by trilinear interpolation



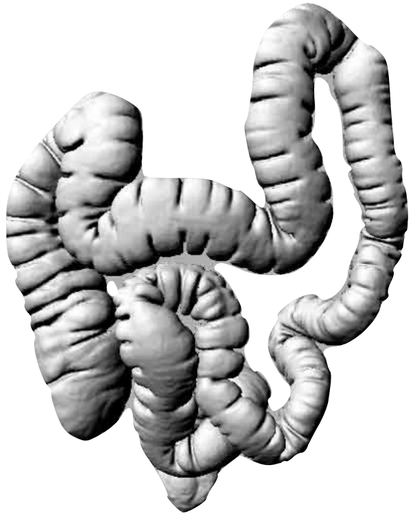
scalar CT volume (tissue density)

isosurfaces

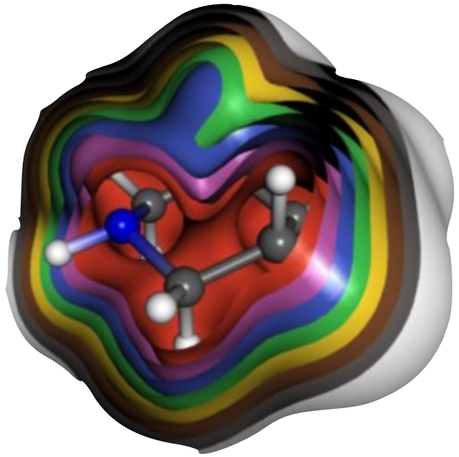


isosurfaces for skin and bone

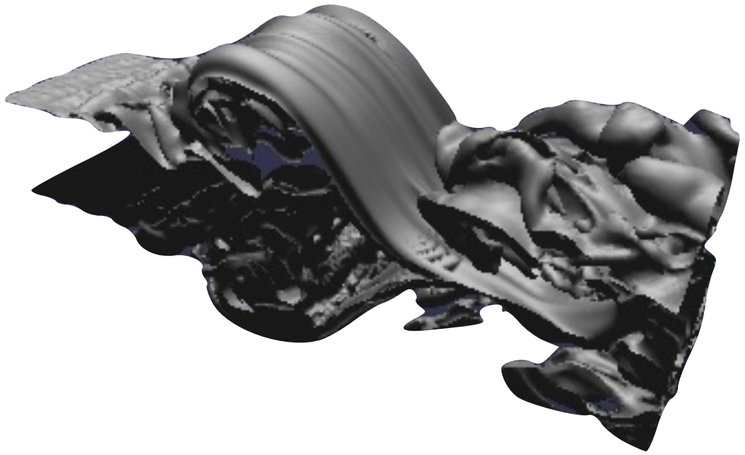
More Examples



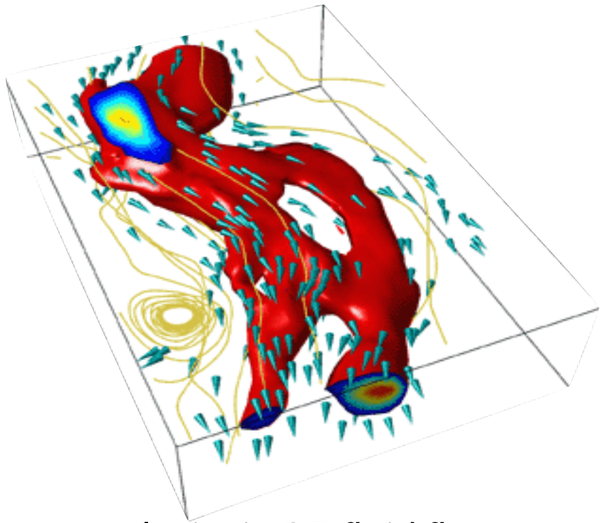
colon (CT dataset)



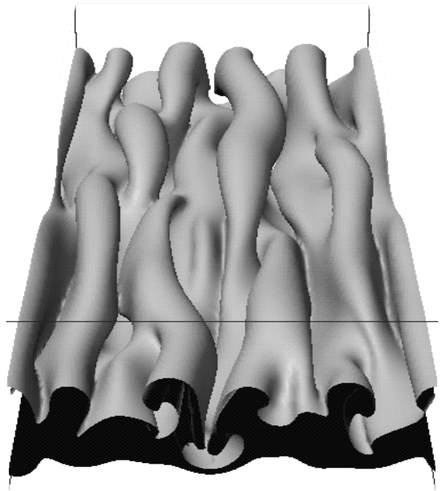
electron density in molecule



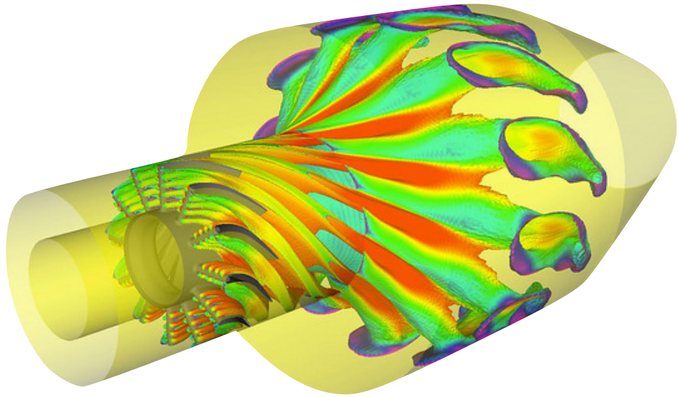
velocity in 3D fluid flow



velocity in 3D fluid flow



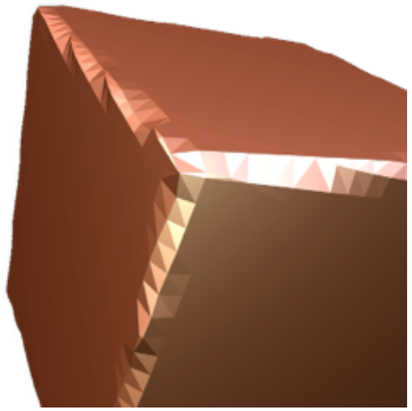
magnetic field in sunspots

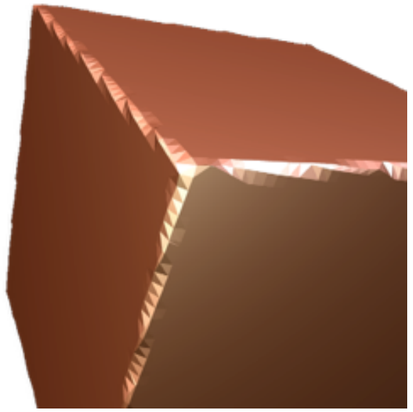


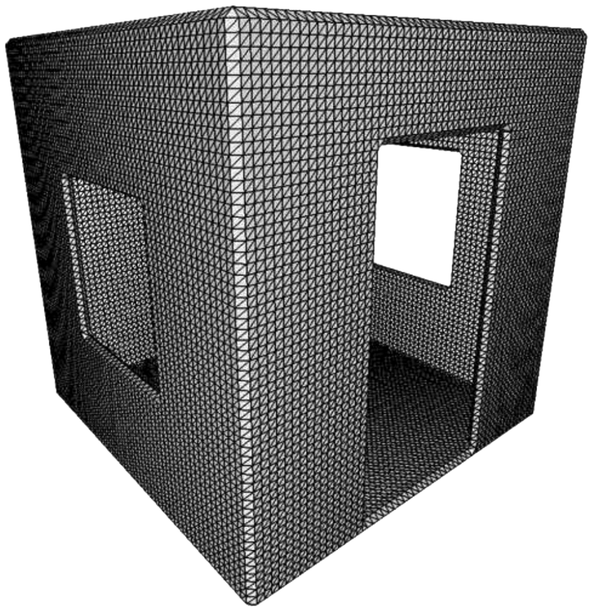
fuel concentration, colored by temperature in jet engine

Other Issues of Marching Cubes

- MC creates a huge number of polygons
  - MC's grid is not adaptive ⟶ many polygons spent for large features just to capture a few small features
- MC can represent only features at least as large as the grid voxels
- Features, e.g. corners and edges, are not preserved

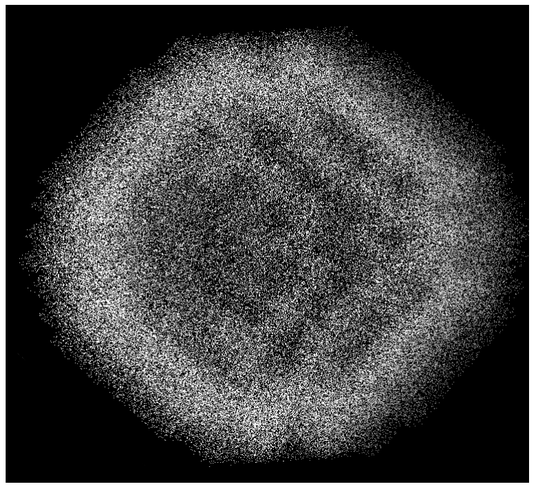


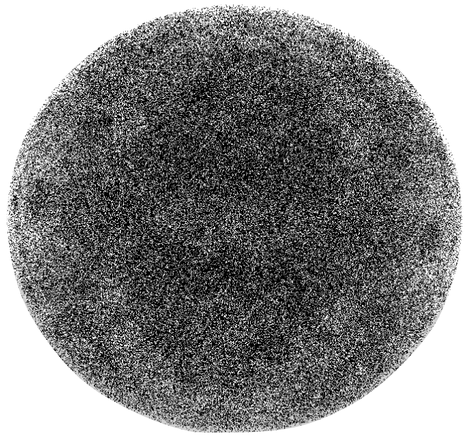


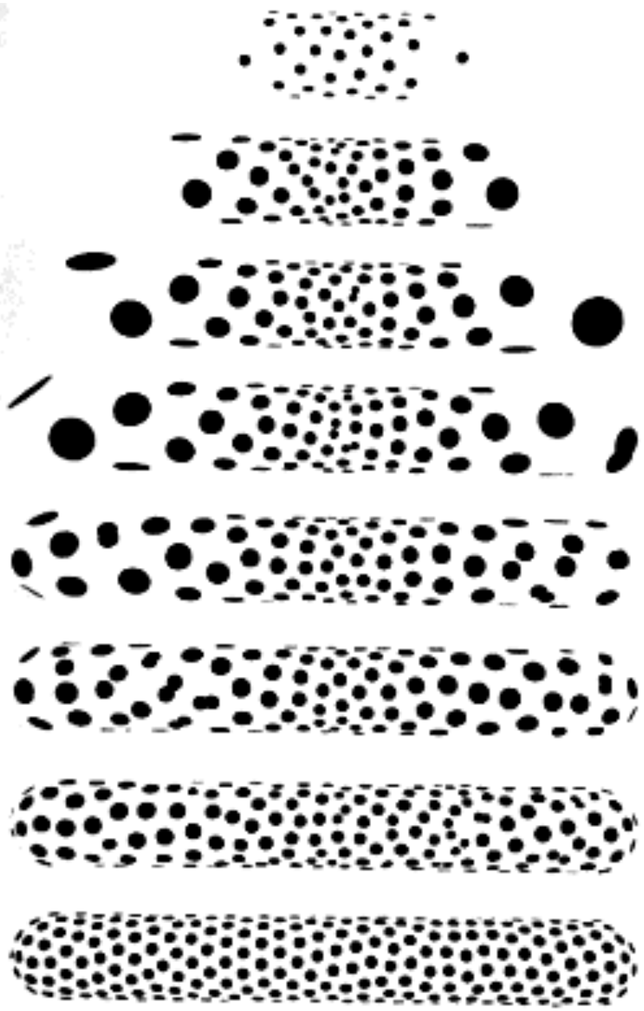


Polygonization of Implicit Surfaces Using Particle Systems

- Idea:
  - Start with set of particles at random positions
  - Move particles in direction of gradient of *F* (attraction towards *F*=0), and away from each other (repelling force)
  - Triangulate (e.g., Delaunay); or, render point cloud directly (e.g., splatting)

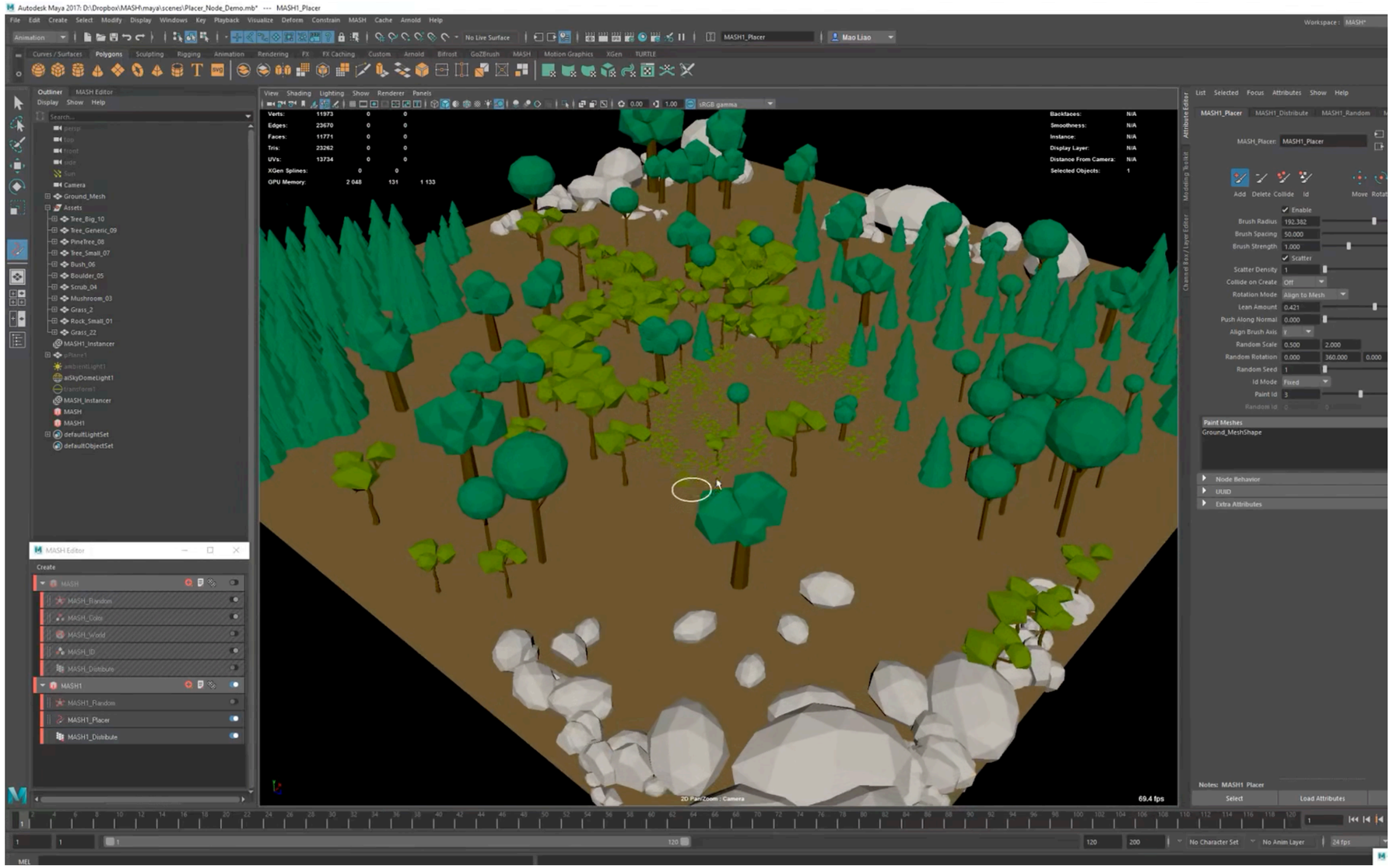




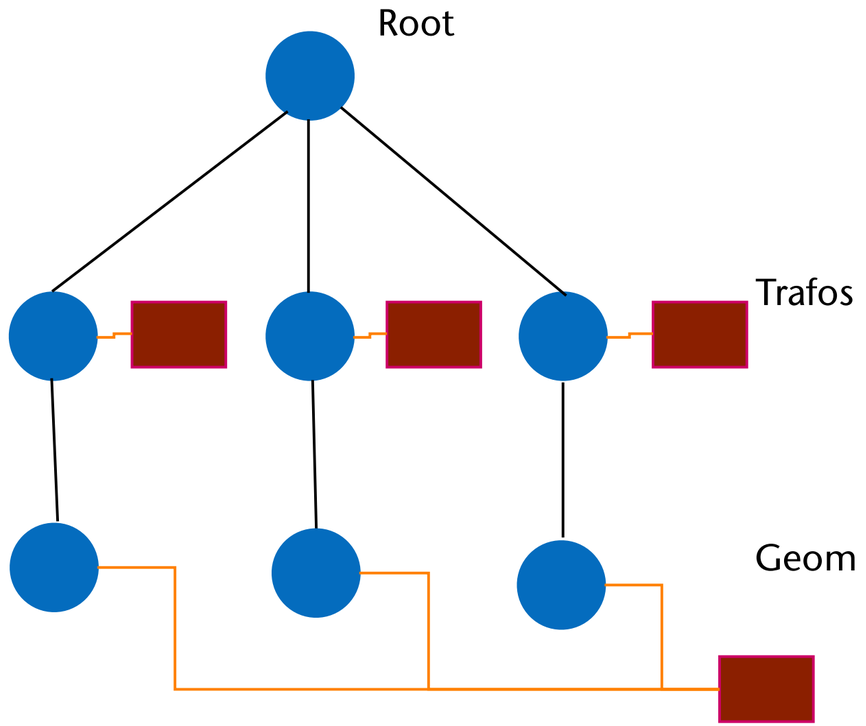


Instancing

- Often, a large number of copies of the same object (modulo affine transformation) need to be created in a scene
- In animation, *set/ environment dressing* is a frequent case



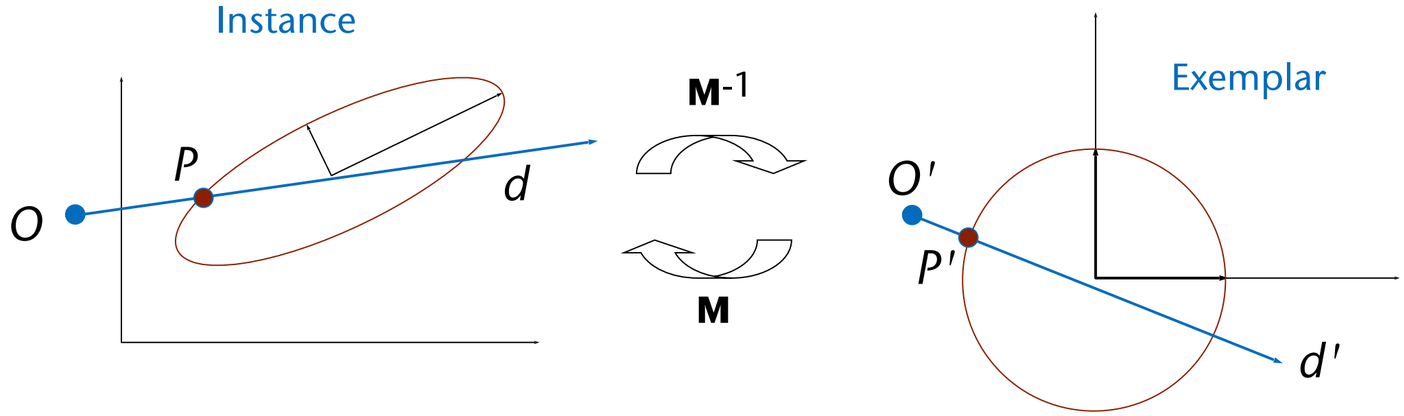
- Implementation in scenegraphs:
  several "copies" (instances) of
  the same object can be created
  by simply storing their
  transformations plus
  a pointer to the
  original geometry (exemplar)
  - Modern scenegraphs
    usually provide this feature



- "Complex" (transformed) shapes can often be created from simpler, canonical shapes
  - Example: non-axis-aligned ellipsoid from unit sphere

Raytracing Instances by Ray Transformation

- Approach: 1. transform ray into exemplar's model space; 2. compute intersection of ray and exemplar; 3. transform intersection point (and normal) back into world space



- The algorithm:

calc $P'(t) = \mathbf{M}^{-1}O + t\mathbf{M}^{-1}d$

intersect $P'(t)$ w/ unit sphere $\rightarrow P'$, $\mathbf{n}'$, t'

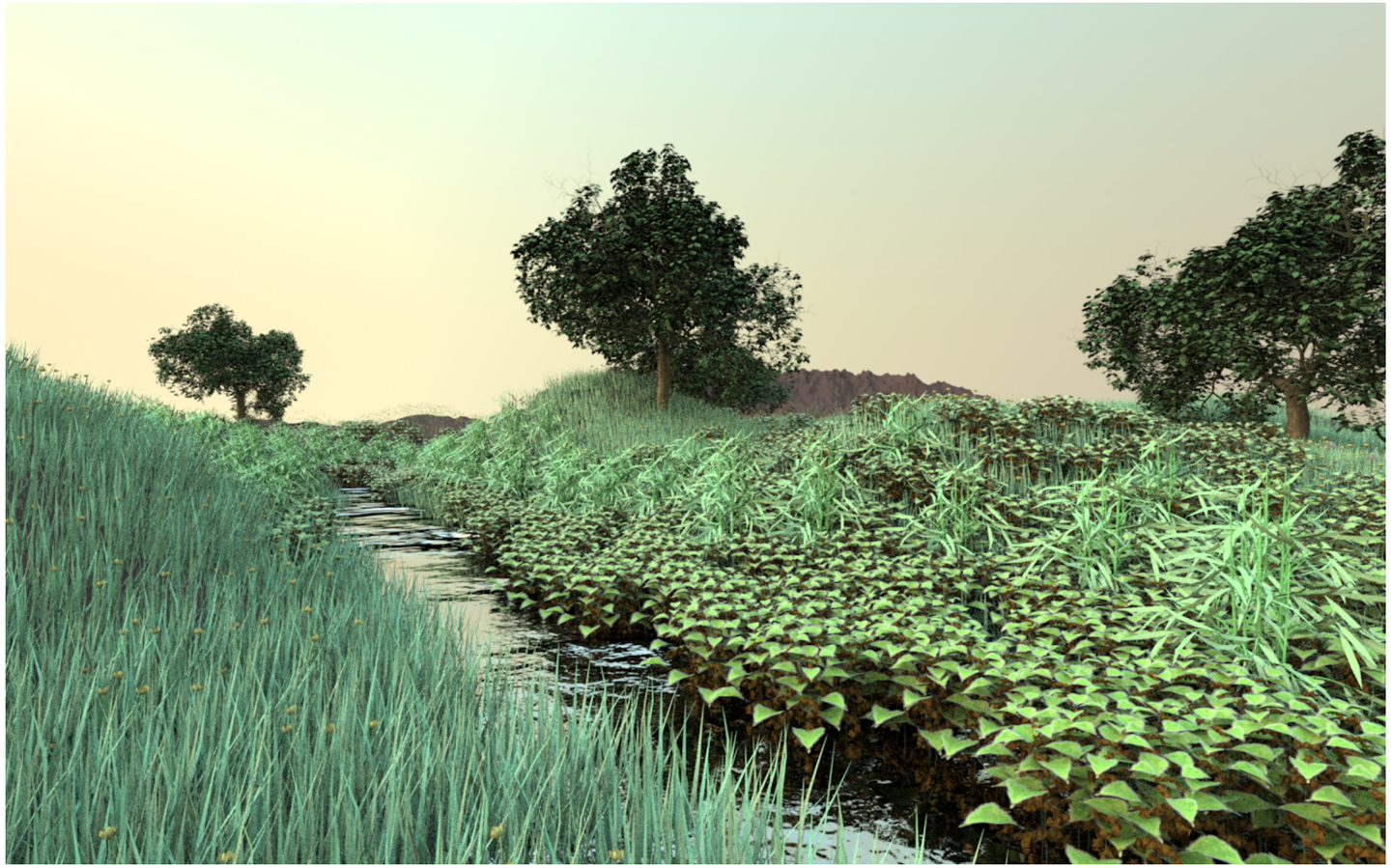
$P := \mathbf{M} \cdot P'$; $\mathbf{n} := (\mathbf{M}^{-1})^{\mathsf{T}} \cdot \mathbf{n}'$; $t :=?$

Why is it $(M^{-1})^T$ to transform the normal back?

https://www.menti.com/ytms1d4mv1

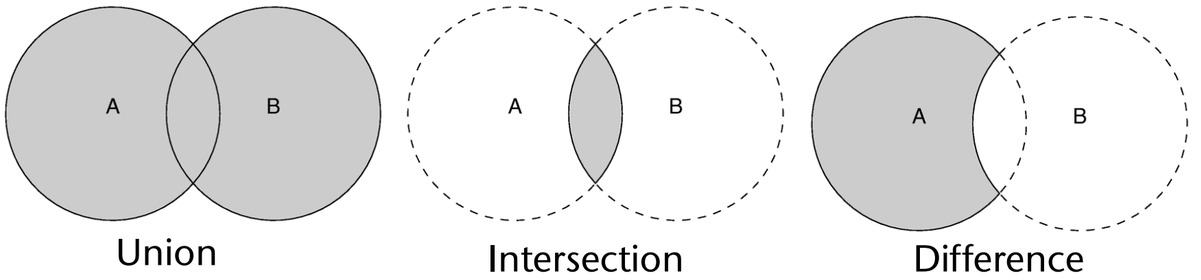
Example for Instancing

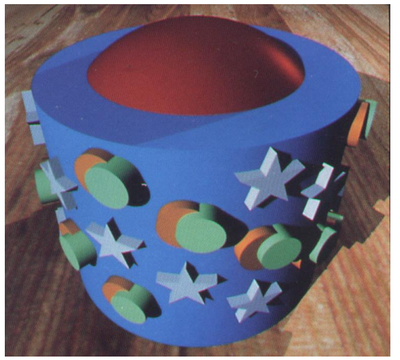
- Memory efficiency: only with instancing can you fit such huge scenes into main memory
- Example:
  - With instancing: 61 unique plant models, 1.1M unique triangles, 300MBytes
  - With explicit copies: 4000 instanced plants in the scene, 19.5M triangles

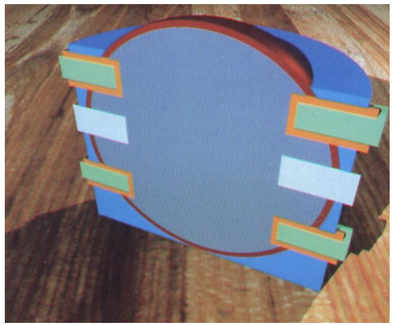


Constructive Solid Geometry (CSG)

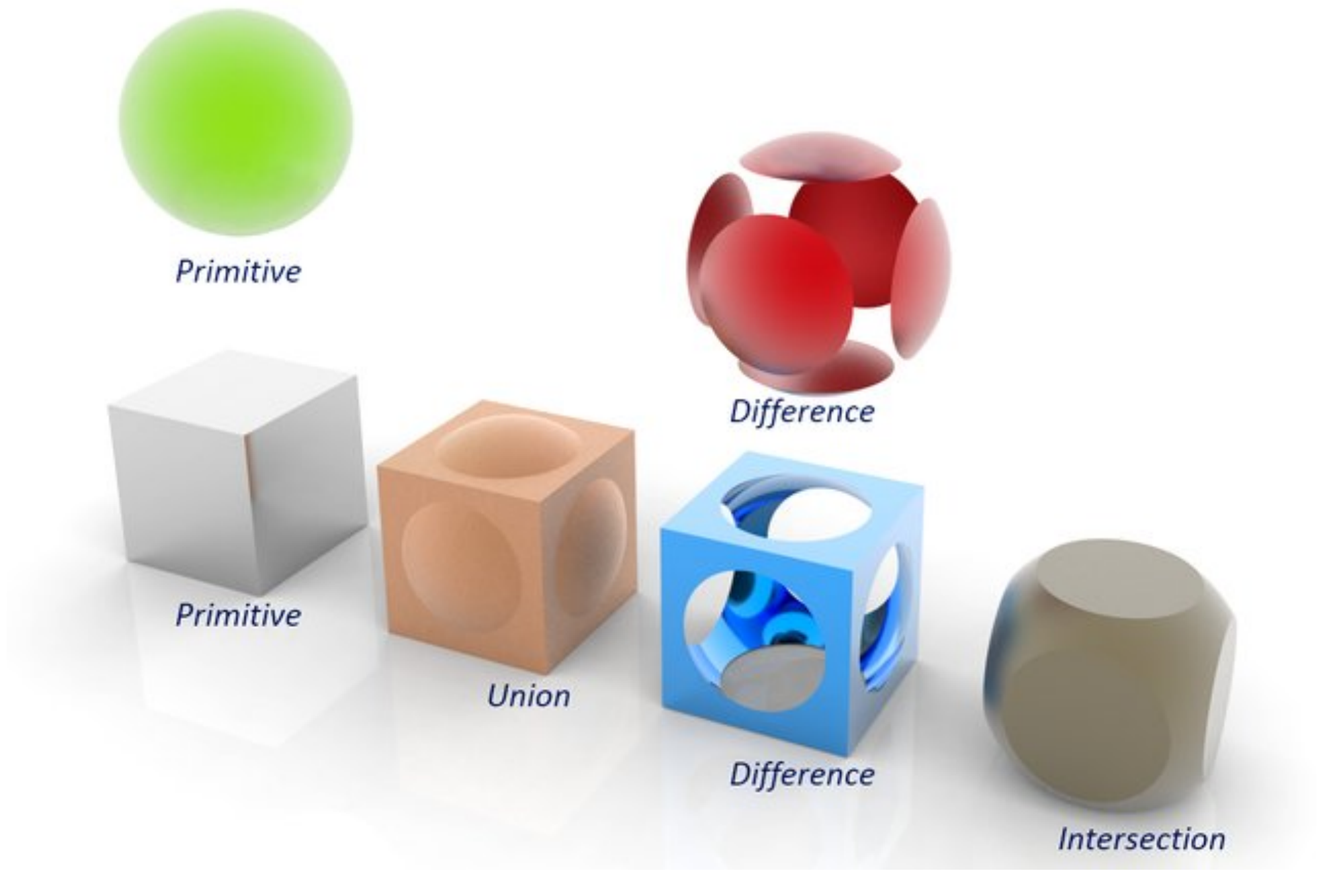
- Central idea: construct a new object by *set operations* performed on simpler, volumetric primitives → *constructive solid geometry* (CSG)
- Simple primitives = "simple" shapes *with volume*
  - E.g., sphere, box, cylinder, ...
  - "Simple" shapes are shapes with
    - Easy (= fast) intersection test with ray; or
    - Easy minimal distance computation dist(point,shape)
- Set operations on shapes:

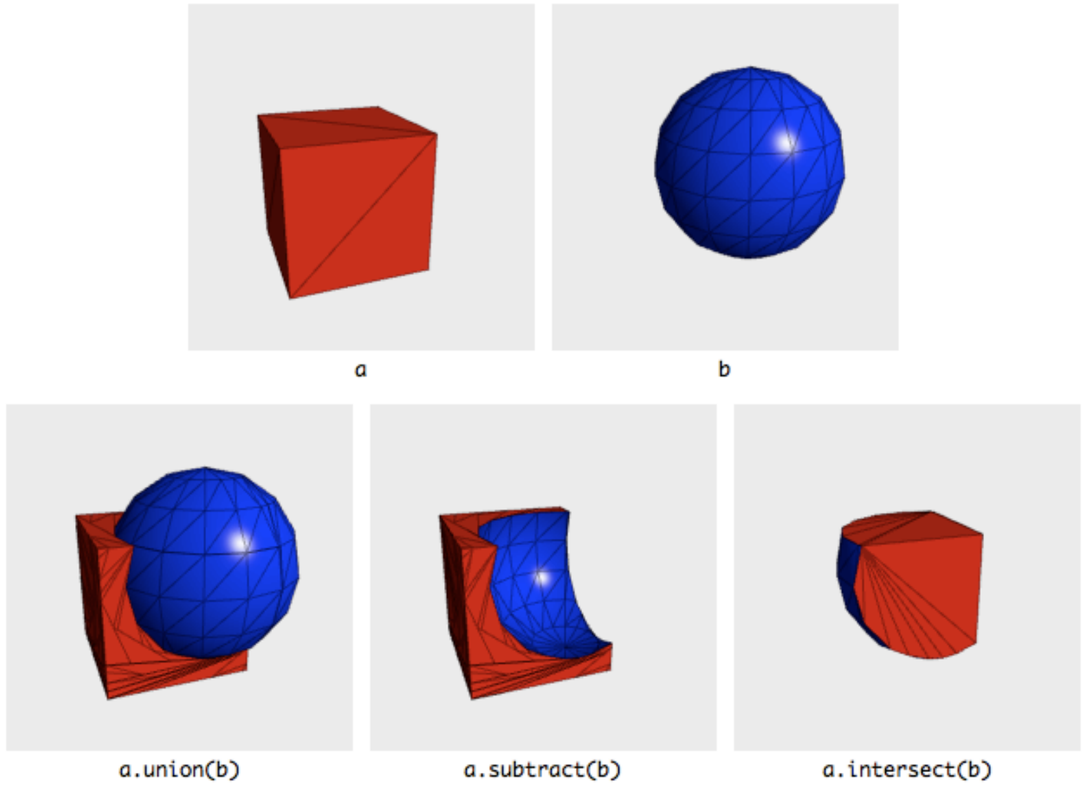






Examples of the Three Set Operations Applied to Sphere & Box

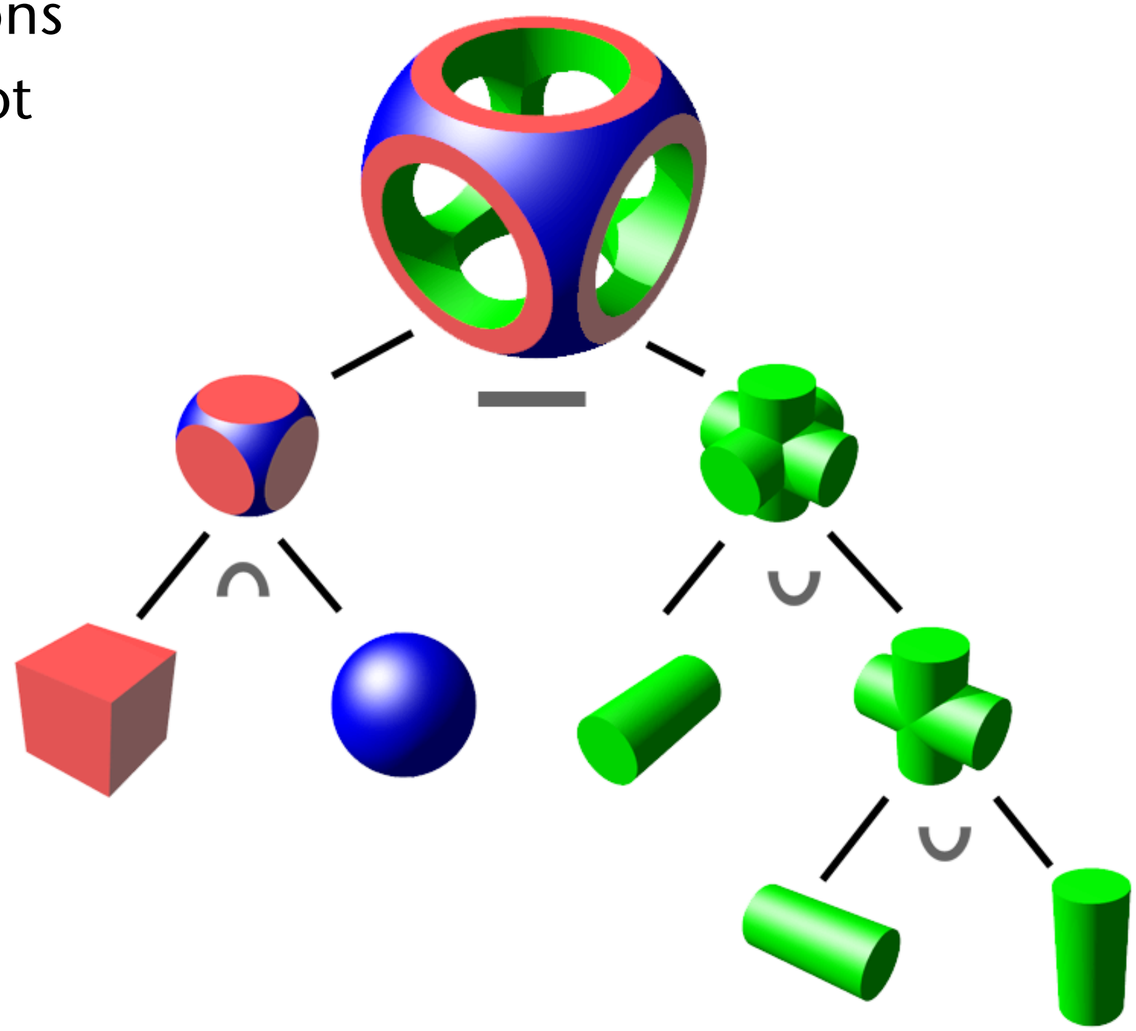




https://evanw.github.io/csg.js/

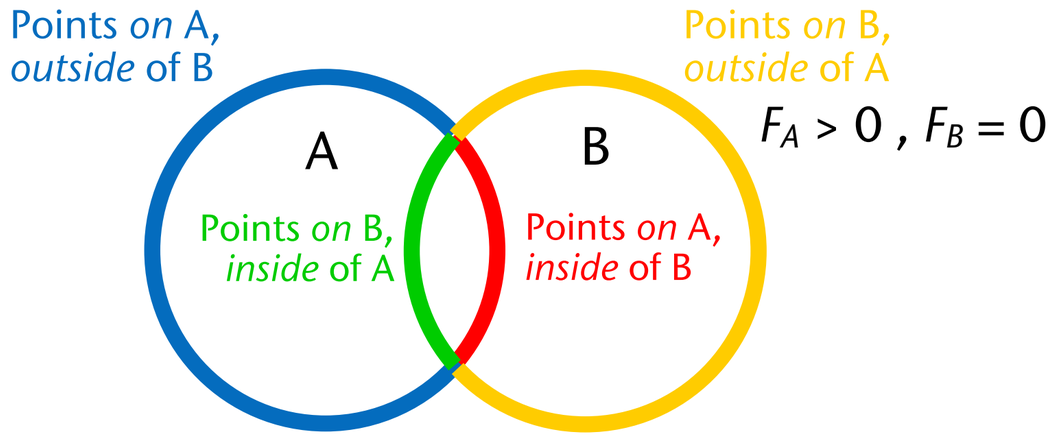
The CSG Tree by Way of an Example

- Recursive application of the set operations
  ⟶ CSG tree = one CSG object at the root
- Leaves = primitives
- Inner nodes = CSG operations
- Evaluation of CSG trees works similar to evaluation of arithmetic expression trees

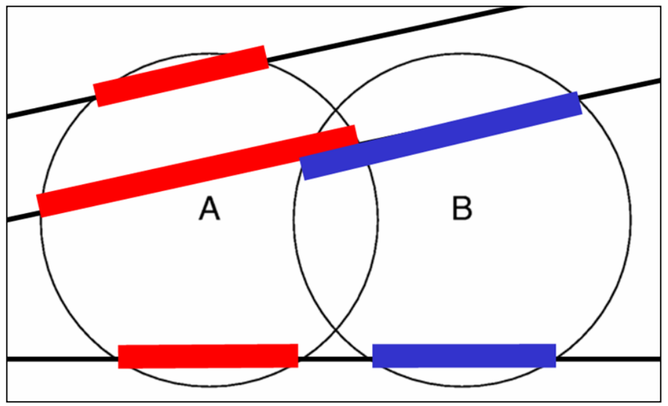


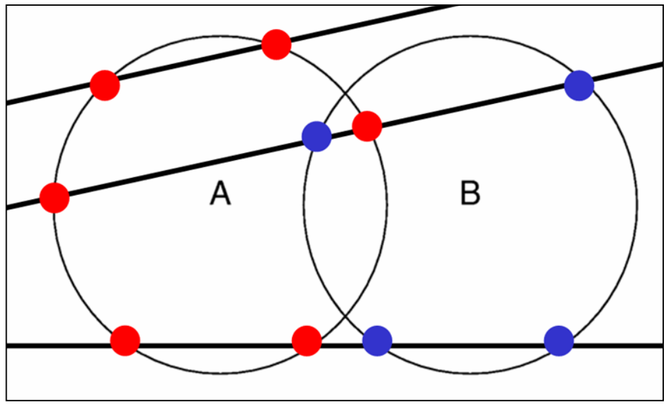
Rendering CSG Objects Using Raytracing

- Use implicit or explicit representation of the primitives:

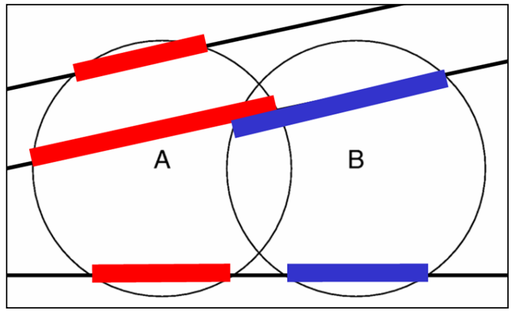


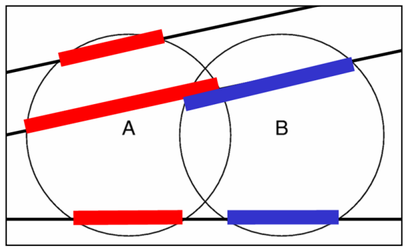
- Determine **all** intersection points of a ray with the 2 primitives
- If primitives are convex → one interval where ray is inside primitive



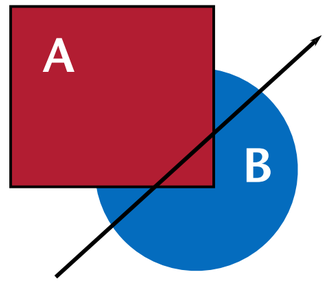


- Propagate intervals up through the CSG tree
  - At parent node, apply CSG operation to child intervals
- If interval is empty when reaching root → no intersection
- Else: choose closest interval and intersection point closest to viewpoint
- Warning:
  - During CSG operations on intervals, the resulting interval can become non-contiguous (i.e., several intervals need to be maintained during tree traversal)!
  - Also, pay attention to numerical robustness (e.g., kill very small intervals)





With A∪B, we get "one" non-contiguous interval in case of this ray!



Dito in this case with B - A !

Converting CSG's to Meshes Using Marching Cubes

- Prerequisite:
  - Given a CSG object, i.e., a CSG tree
  - A method to calculate minimal distance from any point in space to the CSG object
- Given the distance calculation method, the rest is straight-forward
  - Calculate bbox of CSG object, sample distances on grid inside bbox
  - Could even adapt resolution to highly detailed parts (using multi-grids)

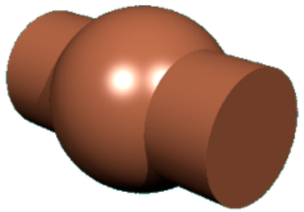
Algorithm for Calculating dist(q, CSG-obj)

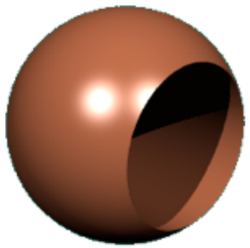
- Given: query point **q**
- Traverse the CSG tree down to all leaves *A* (= primitives), and calculate distance $d(\mathbf{q}, A)$
- Assume:

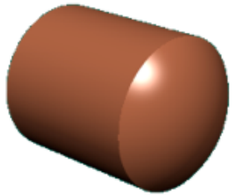
  $d < 0$ for **q** *inside* A, $d = 0$ for **q** *on* surface of A, $d > 0$ for **q** *outside* A
- Let $C = A \otimes B$ be an inner node of the CSG tree ($\otimes$ is either $\cup$, $\cap$, or $\setminus$)
- Now

$$d(\mathbf{q}, A \cup B) = \min(\, d(\mathbf{q}, A),\, d(\mathbf{q}, B)\,)$$

$$d(\mathbf{q}, A \cap B) = \max(\, d(\mathbf{q}, A),\, d(\mathbf{q}, B)\,)$$



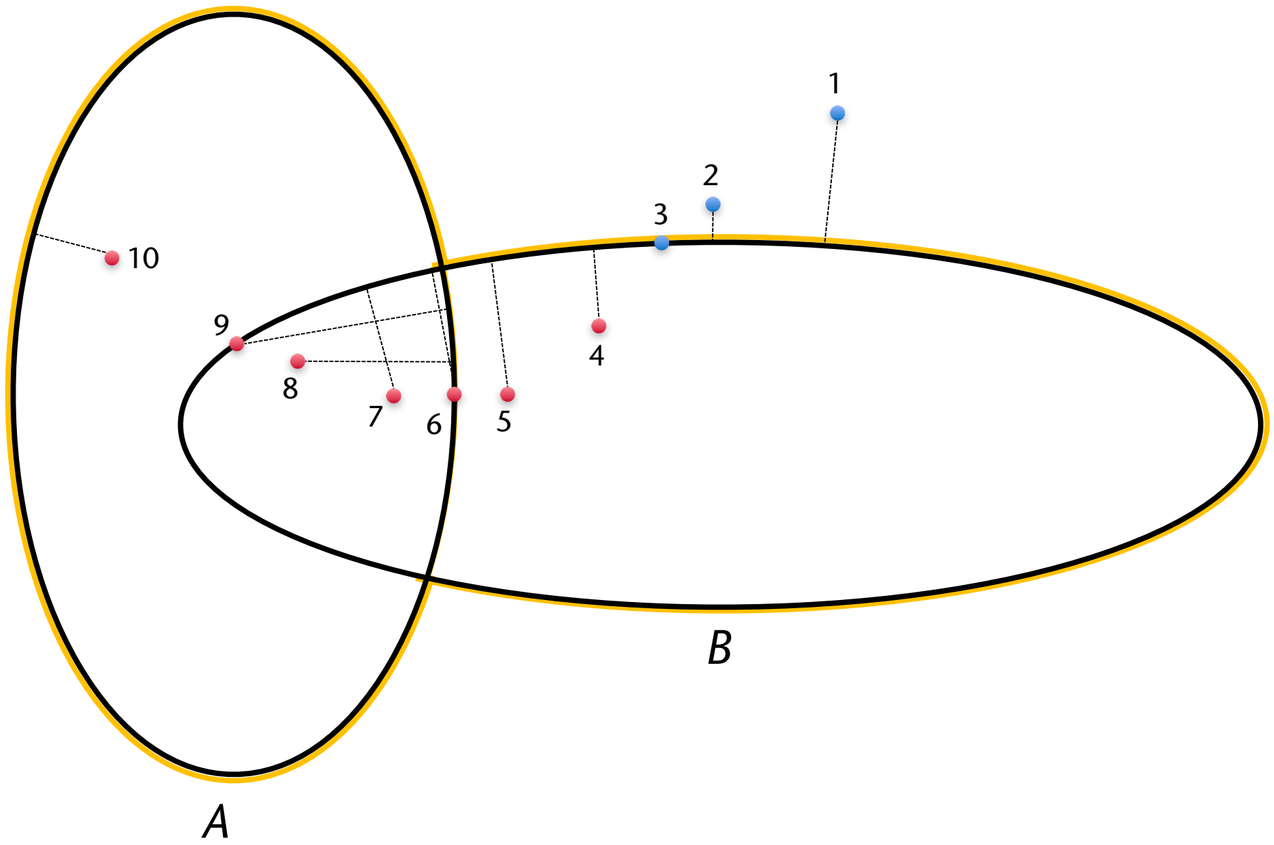




$$d(\mathbf{q}, A \setminus B) = \max(\, d(\mathbf{q}, A),\, -d(\mathbf{q}, B)\,)$$

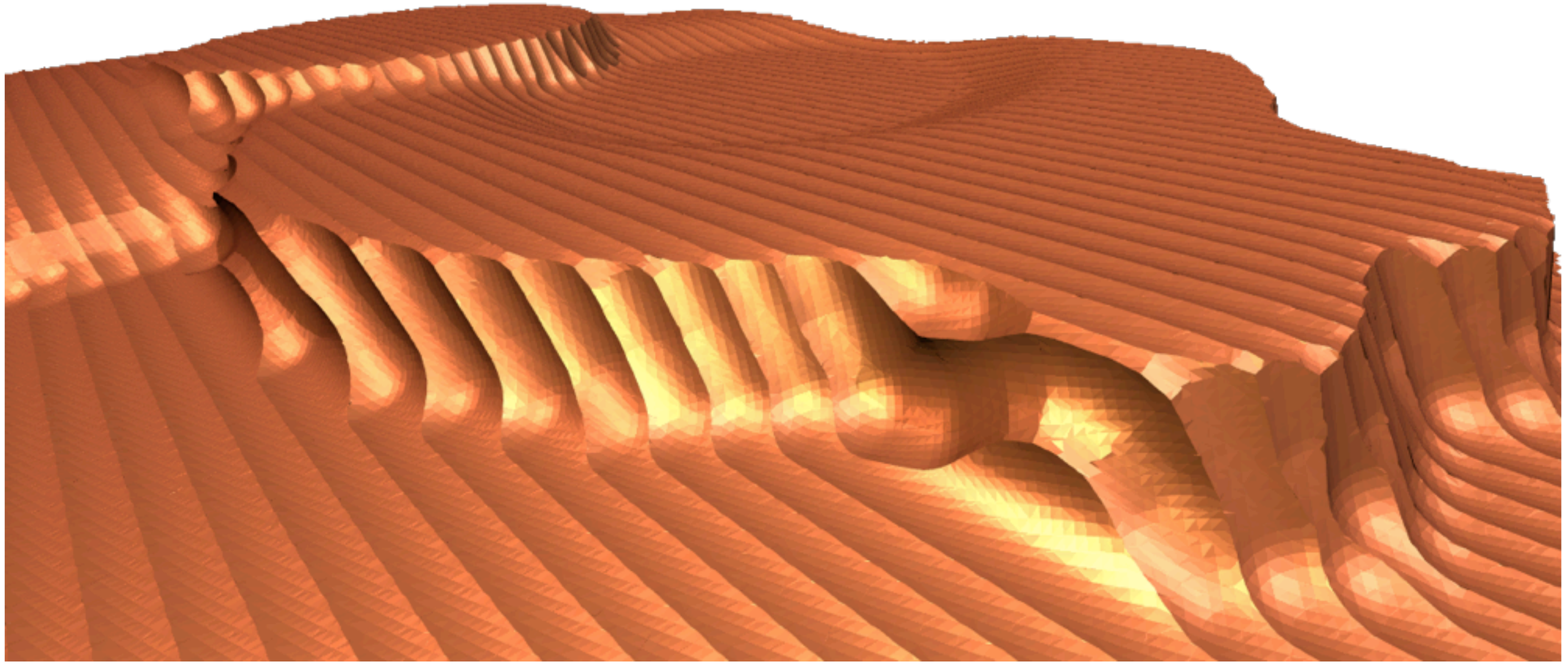
Motivation

$$C = A \cup B \quad \stackrel{?}{\equiv} \quad d(\mathbf{q}, A \cup B) = \min(\ d(\mathbf{q}, A),\ d(\mathbf{q}, B)\ )$$



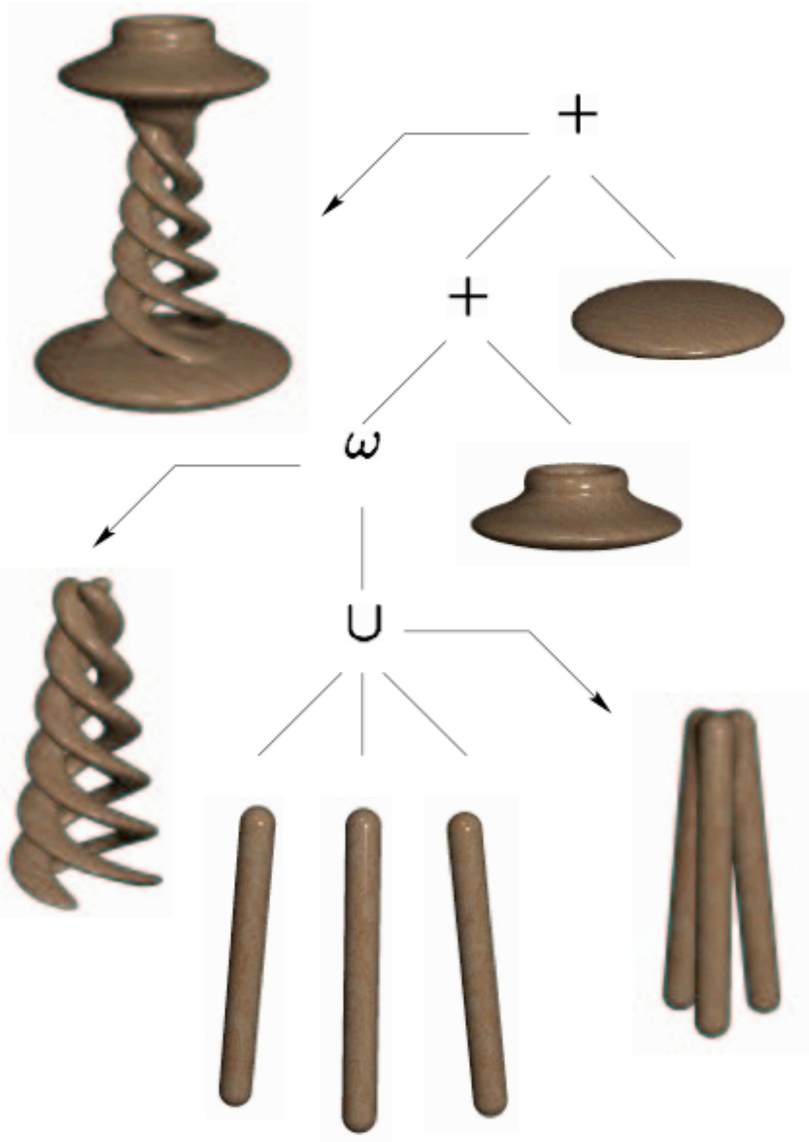
- Warning: this definition of the distance function does NOT produce a true distance field, i.e., for some **q**, it does NOT give the minimal distance between **q** and the CSG object !
- With the distance, you can propagate the "closest" point up the CSG tree
  - Similar warning: this is not always the true closest point on the resulting surface of the CSG object!
  - Can you make use of this closest point during the Marching Cubes algo?
- For query points (grid points) close to the final surface, the distances reported by *d*() are good enough

Example Application: Simulation of Milling



Combination of CSG and Metaballs

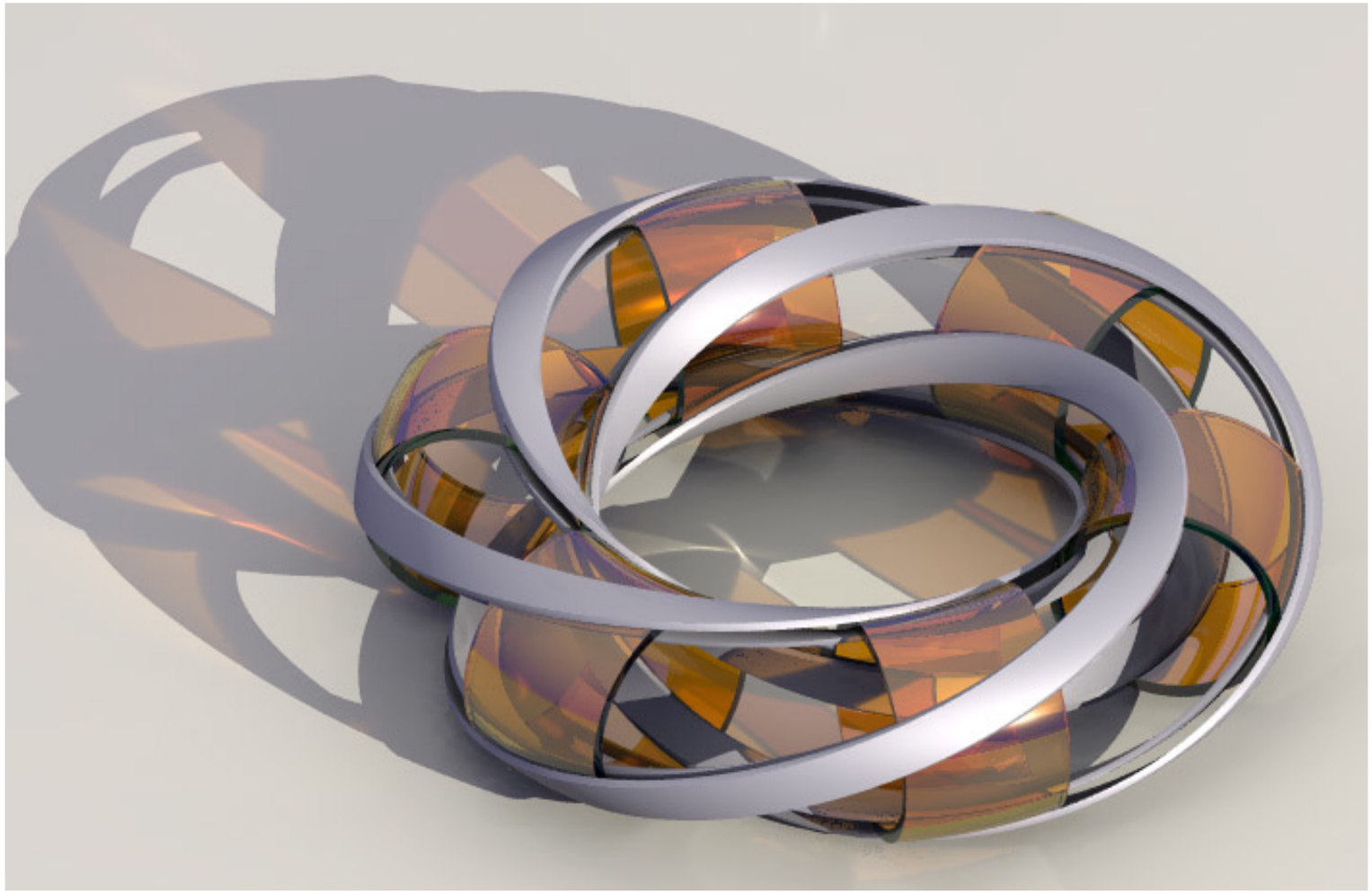
- Could combine metaballs and CSG
- Approach:
  - A tree of "blending" and set operations ("BlobTree")
  - Leaves are individual metaballs (isosurfaces of skeleton points/lines/ disks/...)
  -



CSG in Design (?)

Ferruccio Laviani

Solution to the Quizz



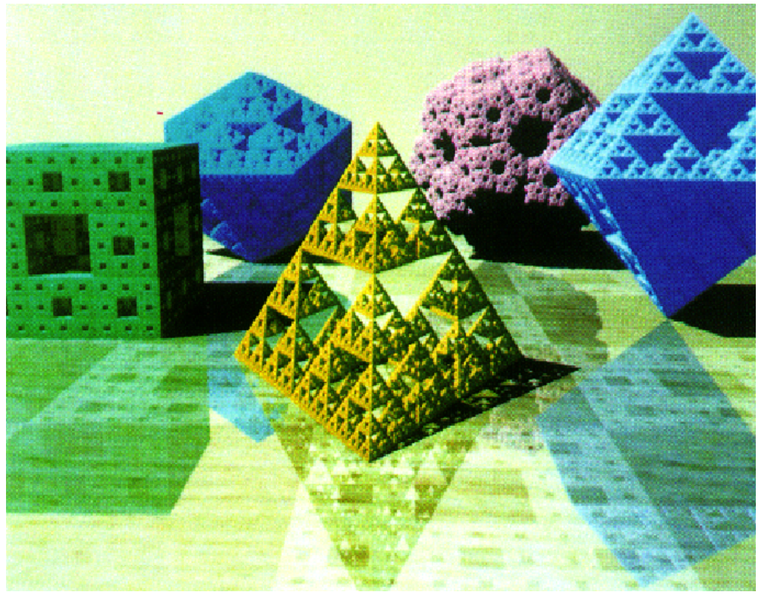
"Villarceau Circles" by Tor Olav Kristensen (2004)

For every point on a torus, one can draw four different circles through it that all lie on the surface of the torus. Two of these four circles are called Villarceau circles. The four narrow pairs of bands in this image follow such Villarceau circles. All the shapes in this image are made with Constructive Solid Geometry operations with tori only (except for the ground plane of course).

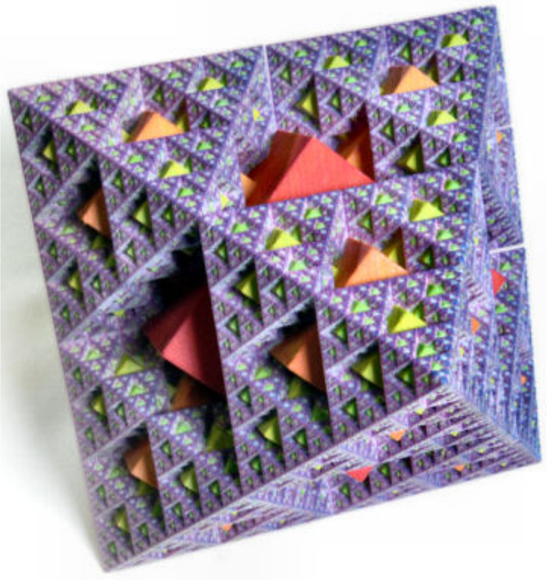
Procedurally Defined Fractals

- "Subtractive" recursive construction of fractals: recursively "punch out" a part of the bigger shape, such that the remains are smaller copies of the original object
- Ray-tracing them is trivial: just recurse "on demand" up to some predefined depth

Subtractive

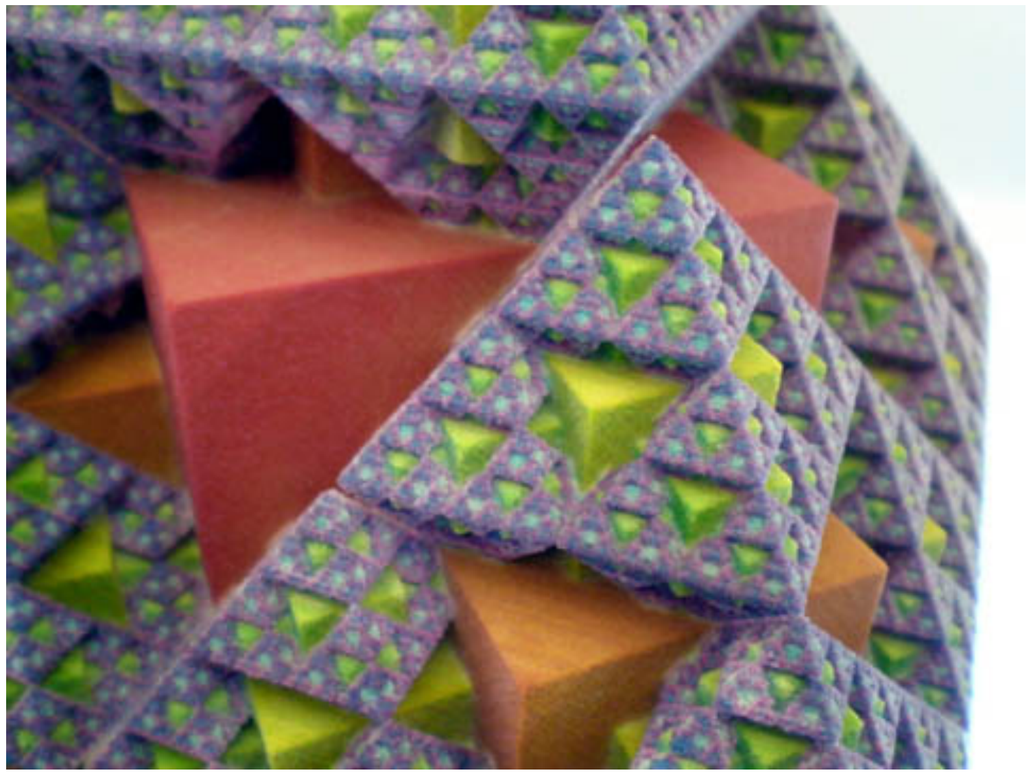


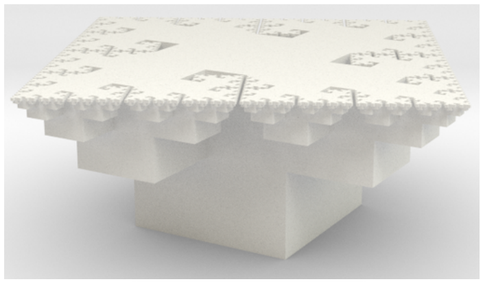
Additive

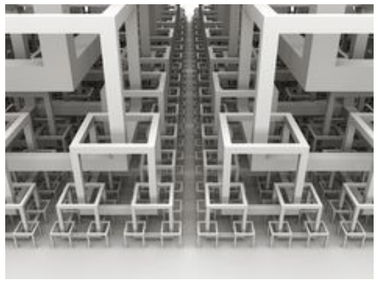


Fathauer Crystal

- "Additive" recursive construction of the Fathauer Crystal:
  - Start with a red cube
  - Place five half-sized orange cubes on its exposed faces
  - Put five smaller yellow cubes on the faces of each of those
  - Etc. ...
- Similarly, you can recursively generate many pleasing fractals:







An Early Application of Fractals

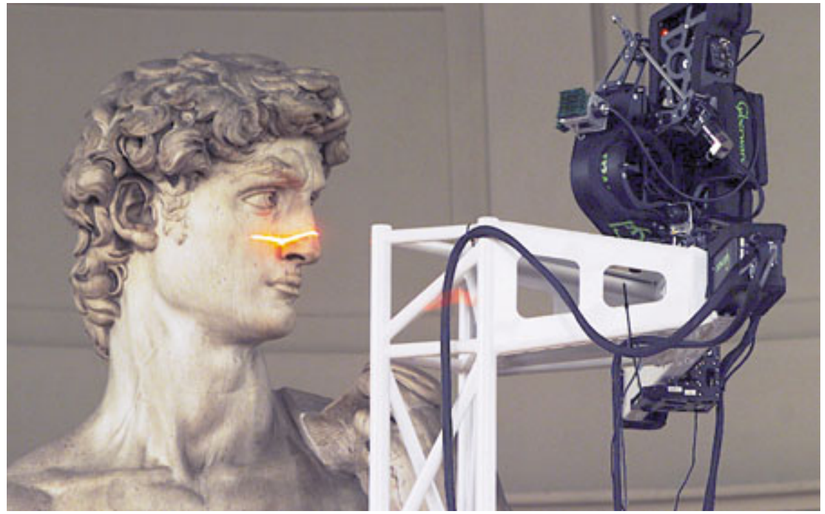
Fractals are very well-suited for modeling terrain

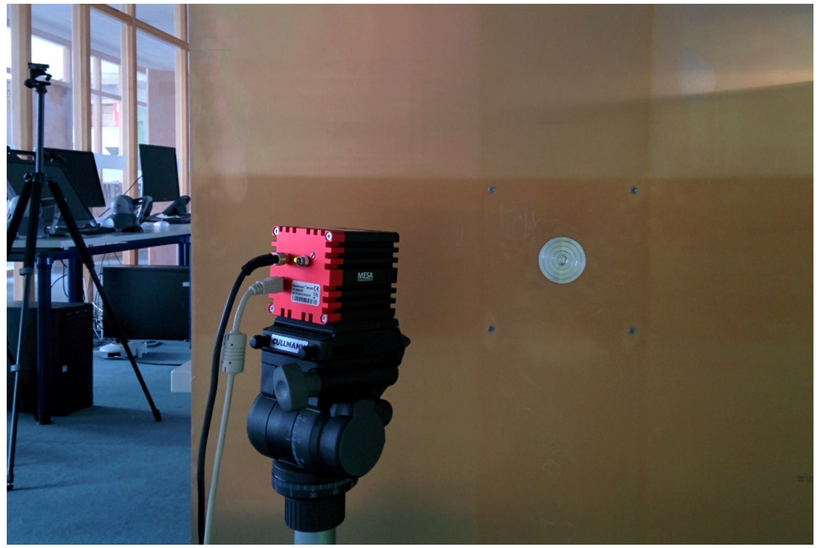
Loren C. Carpenter: Vol Libre, 1980

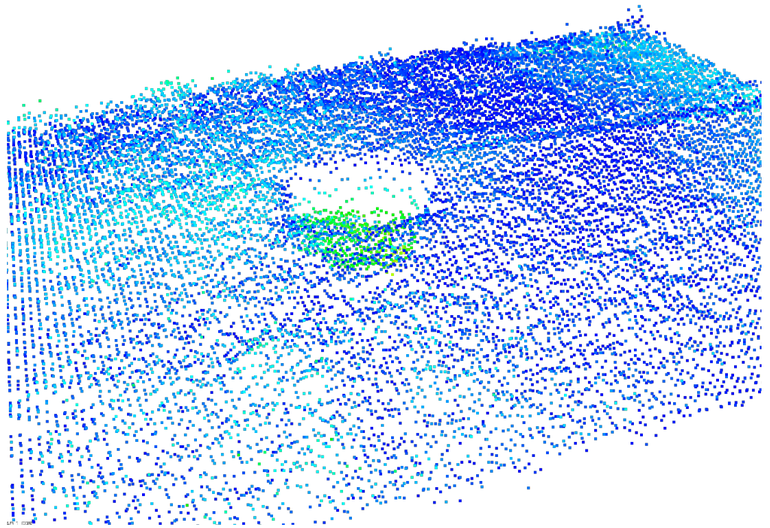
Point Cloud Surfaces

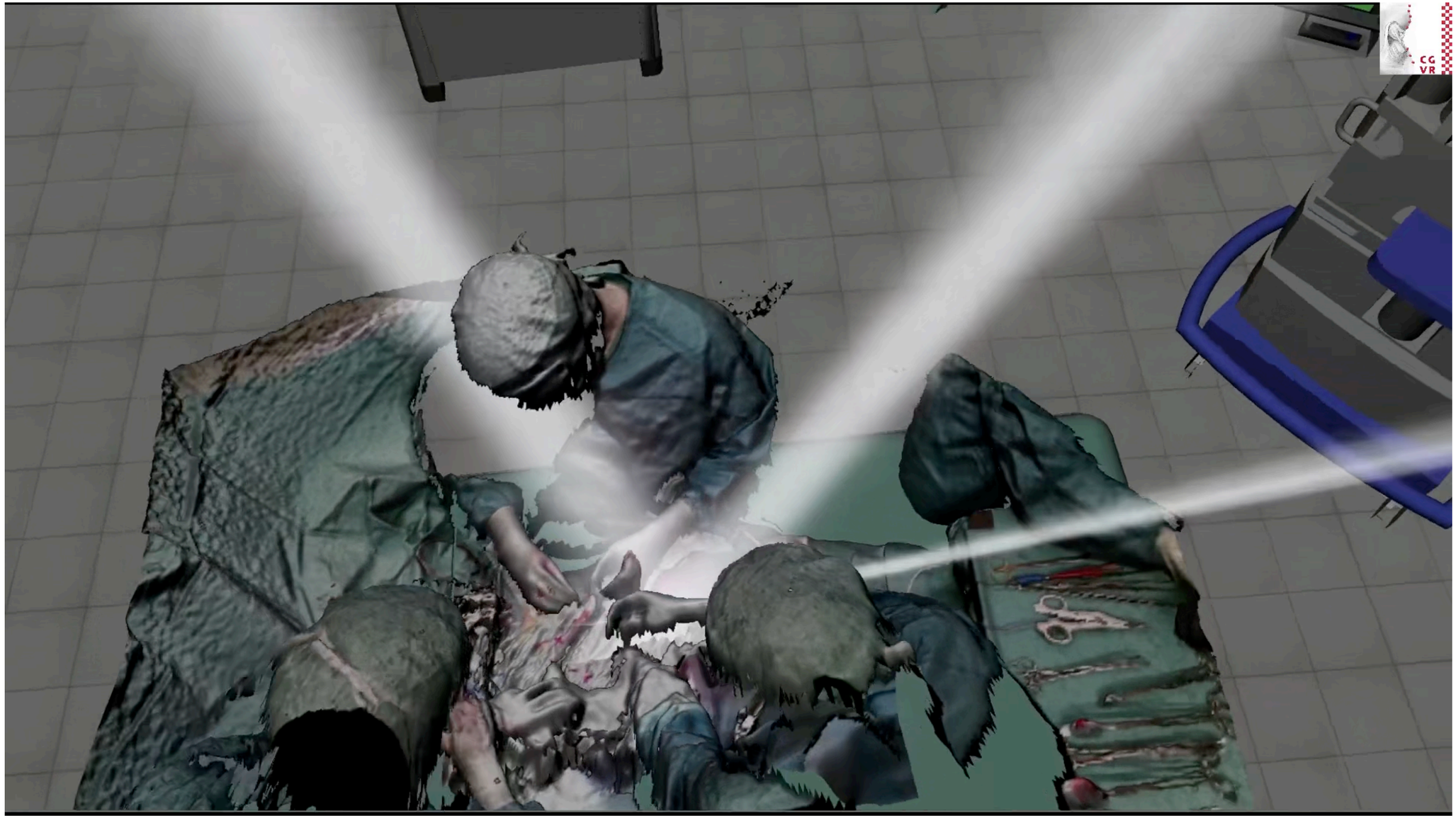
- Increasingly popular geometry representation
- Lots of sources of point clouds (laser scanners, Kinect et al., ...)

Applications





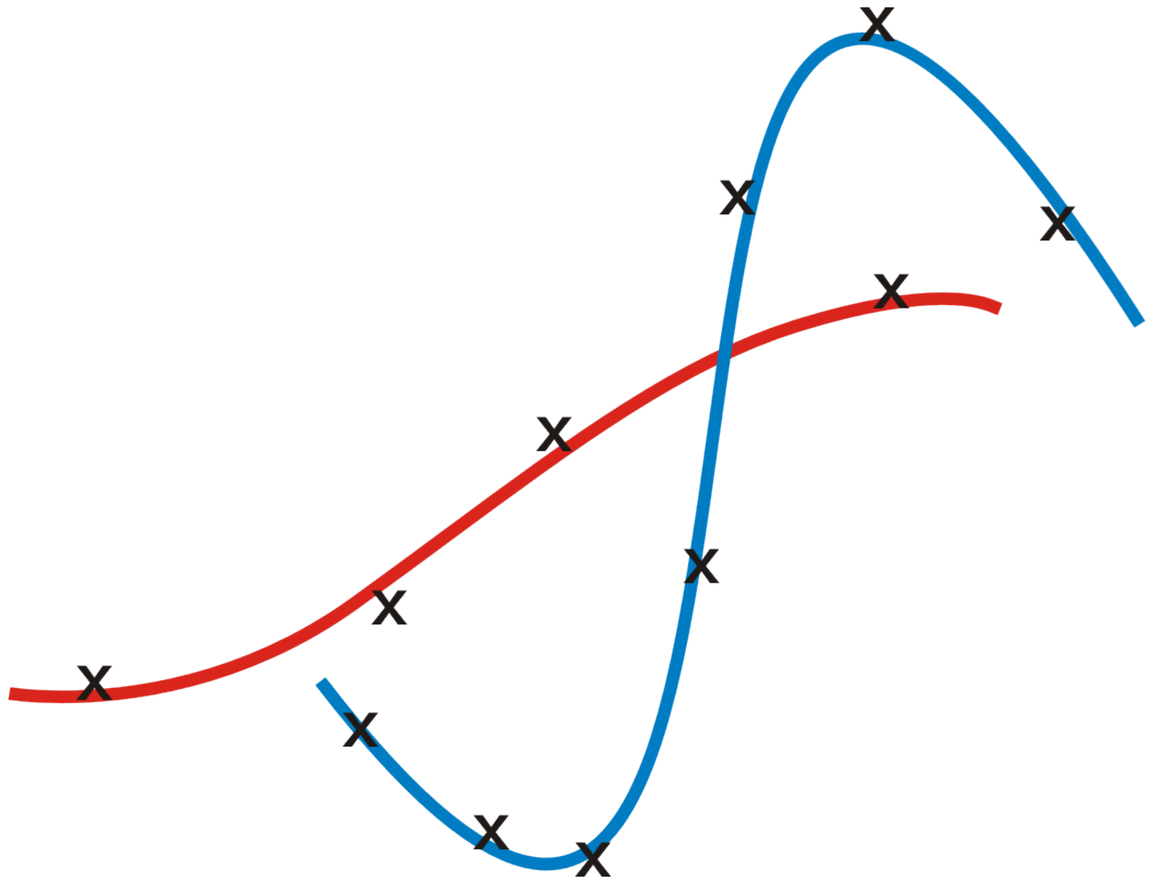




Project Smart-OT

Our Goal

- Surface definition that is ..
  - Quick to evaluate
  - Robust against noise
  - Smooth
- The surface definition / representation should be well suited for:
  - Ray tracing (rendering)
  - Collision detection (physics)

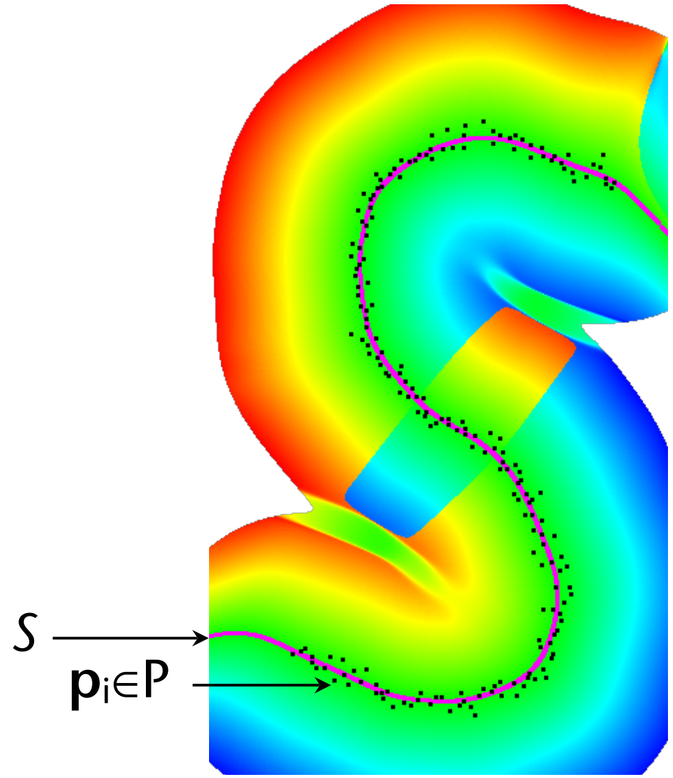


Implicit Surface Definition

- Consider a point cloud P as *noisy* sampling of a smooth surface
  - Consequence: reconstructed surface should *not interpolate* the points
- Define the surface as an implicit surface over a smooth distance function f, determined by the point cloud P:

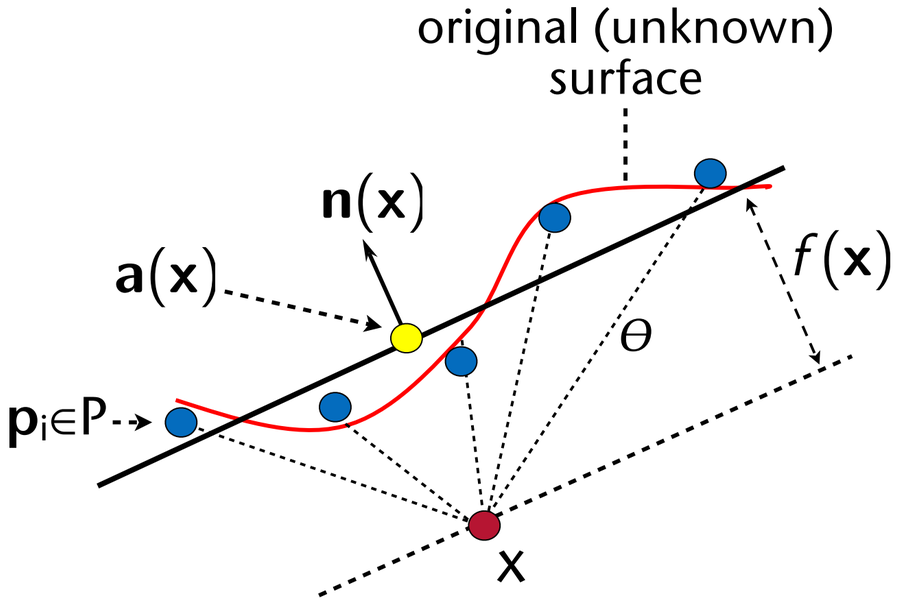
$$S = \{\mathbf{x} \mid f(\mathbf{x}; P) = 0\}$$

where f is the distance to the yet unknown surface S



Weighted Moving Least Squares Method (MLS)

- Define f using weighted moving least squares over k nearest neighbors
- Approximate the surface *locally* by a plane through



$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{k} \theta(\|\mathbf{x} - \mathbf{p}_i\|)\mathbf{p}_i}{\sum_{i=1}^{k} \theta(\|\mathbf{x} - \mathbf{p}_i\|)}$$

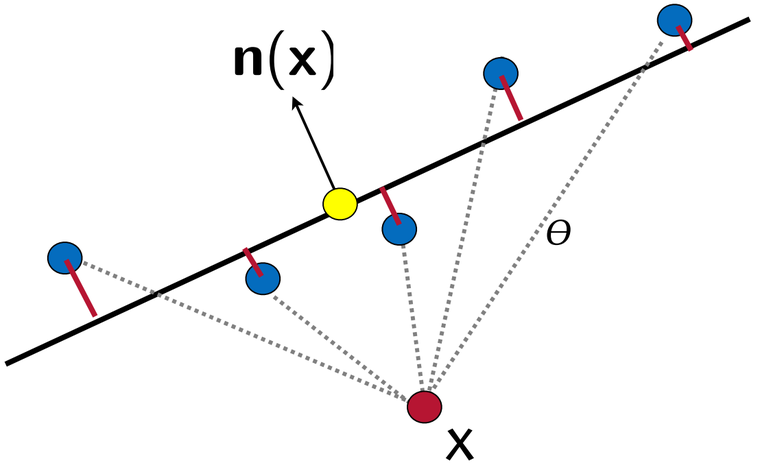
where Θ is an *appropriate* weight function based on distance from **x**, and k = number of nearest neighbors of **x** (fixed k, or determined by a max. radius)

- Overall:

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x})$$

- Choose **n** as

$$\min_{\mathbf{n},\|\mathbf{n}\|=1} \sum_{i=1}^{k} \big(\mathbf{n}\cdot(\mathbf{p}_i - \mathbf{a}(\mathbf{x}))\big)^2 \theta(\|\mathbf{x} - \mathbf{p}_i\|)$$



- From PCA we know: **n** happens to be the *smallest eigenvector* of the (weighted) covariance matrix $B = (b_{ij}) \in \mathbb{R}^{3x3}$ with

$$b_{ij} = \sum_{l=1}^{k} \theta(\|\mathbf{x} - \mathbf{p}_l\|)(p_{l,i} - a_i)(p_{l,j} - a_j)$$

- For the weight function Θ, use (e.g.) a Gaussian kernel

$$\theta(d) = e^{-d^2/h^2}, \quad d = \|\mathbf{x} - \mathbf{p}\|$$

with Euclidean distance (for now), where h is called bandwidth

- Possible weight functions (kernels), not exhaustive list:
  - Gauß kernel
  - The cubic polynomial

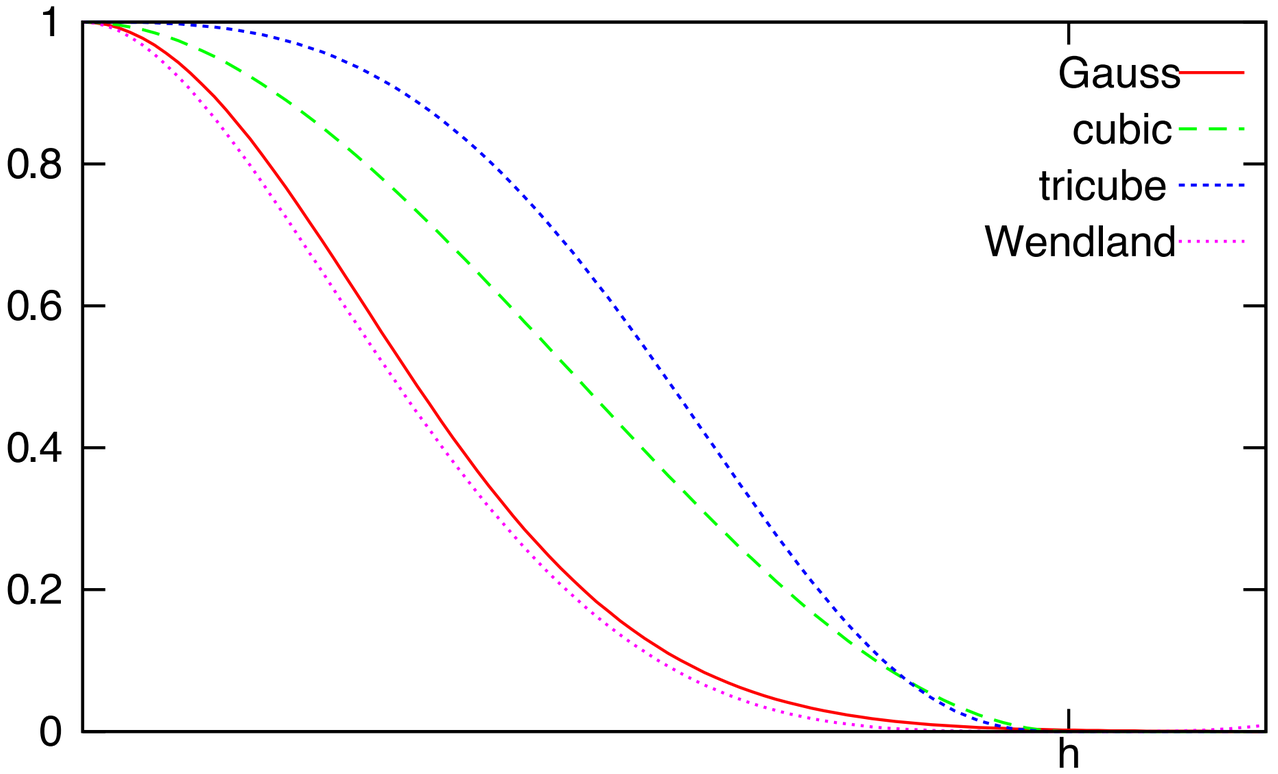
$$\theta(d) = 2\left(\frac{d}{h}\right)^3 - 3\left(\frac{d}{h}\right)^2 + 1$$

  - The tricube function

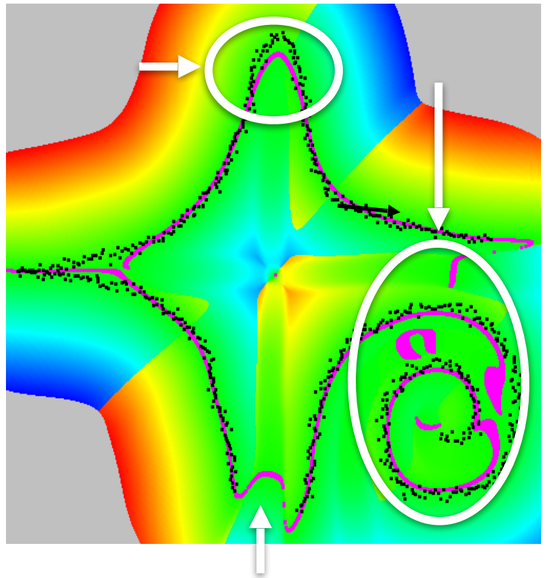
$$\theta(d) = \left(1 - \left|\frac{d}{h}\right|^3\right)^3$$

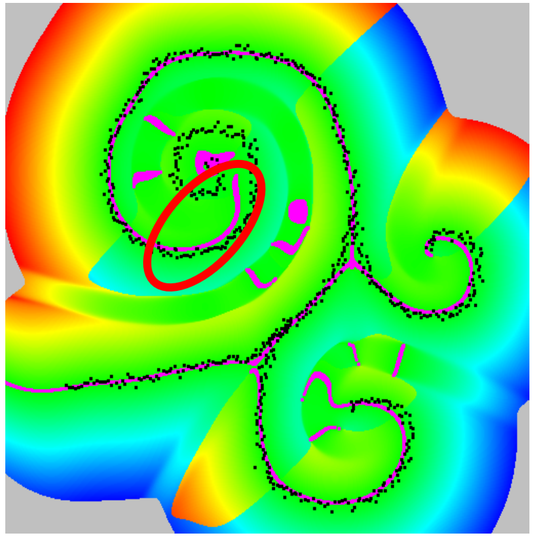
  - The Wendland function

$$\theta(d) = \left(1 - \frac{d}{h}\right)^4 \left(4\frac{d}{h} + 1\right)$$

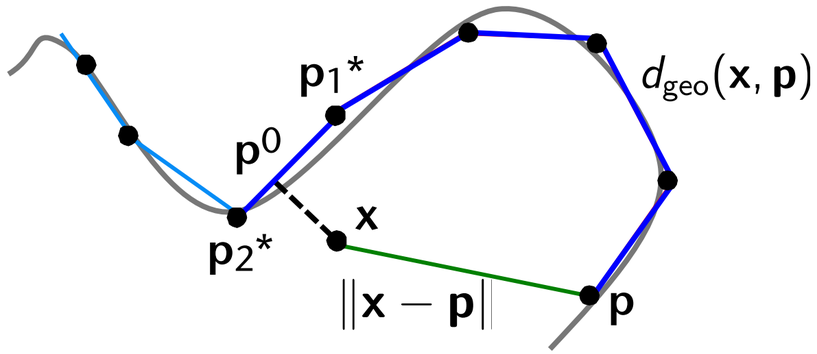


- Whatever kernel you use, it is fine to consider only "close neighbors" (NN) around **x** for the computation of **a**(**x**) and **n**(**x**)
  $\longrightarrow$ need lots of k-NN searches in P (see "Computational Geometry")
- More important: what distance measure to use in $\theta(\|\mathbf{x} - \mathbf{p}_i\|)$?
- *Euclidean* distance produces *artefacts* like this:





- Solution: use a topology-based distance measure
  - Try to mimic the *geodesic distance* on the surface
  - Except without knowing the surface yet
- Use a proximity graph over point cloud
- Define



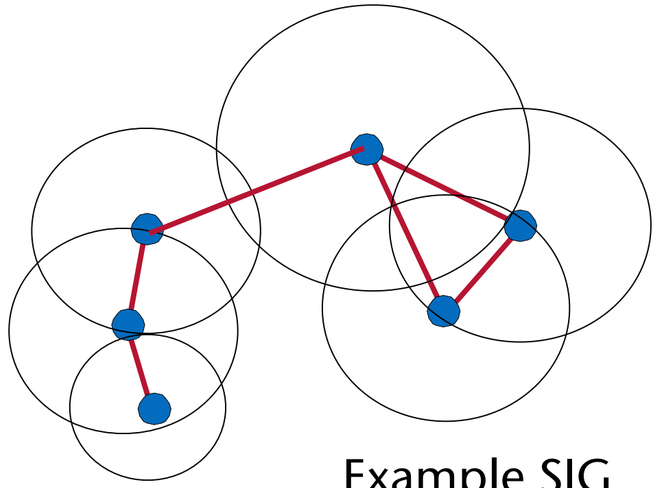
$$d_{\text{geo}}(\mathbf{x}, \mathbf{p}) = (1-a)\cdot\left(d(\mathbf{p}_1^*, \mathbf{p}) + \|\mathbf{p}^0 - \mathbf{p}_1^*\| \right) + a \cdot\left(d(\mathbf{p}_2^*, \mathbf{p}) + \|\mathbf{p}^0 - \mathbf{p}_2^*\| \right)$$

with $a = \|\mathbf{p}^0 - \mathbf{p}_1^*|$
and $d(\mathbf{p}_i^*, \mathbf{p})$ = length of shortest path through proximity graph

- Note: don't add $\|\mathbf{p}^0 - \mathbf{x}\|$ (for practical reasons)

Which Proximity Graph to Use

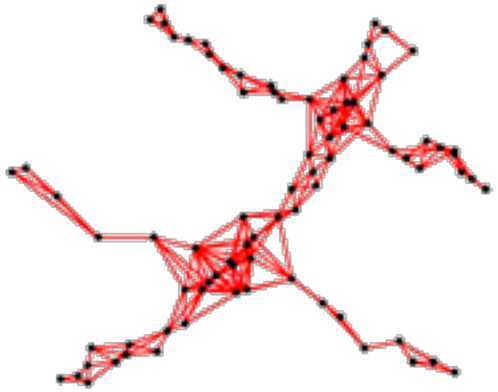
- Many kinds of proximity graphs exist
  - Delaunay graph (see "Computational Geometry")
    - Needs kind of a "pruning" because of "long" edges; has other problems, too
  - Most other proximity graphs are *subgraphs* of the Delaunay graph
  - Sphere-of-Influence graph (SIG): is *not* a subgraph of the Delaunay graph
- Definition of the SIG:
  - For each point $\mathbf{p}_i \in P$ define $r_i = \|\mathbf{p}_i - \mathrm{NN}(\mathbf{p}_i)\|$
  - Connect $\mathbf{p}_i$ and $\mathbf{p}_j$ by an edge iff $\|\mathbf{p}_i - \mathbf{p}_j\| \leq r_i + r_j$
- Extension: k-SIG
  - Define $r_i = \|\mathbf{p}_i - k\mathrm{NN}(\mathbf{p}_i)\|$



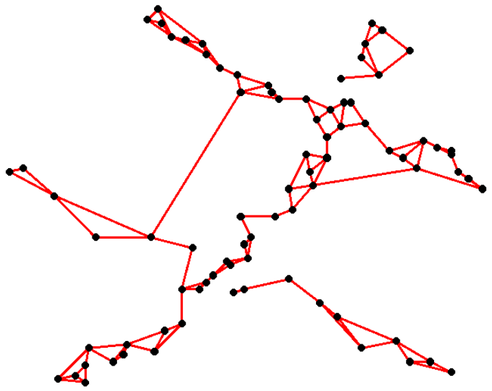
Example SIG

Results

Example sphere-of-influence graph (k-SIG)



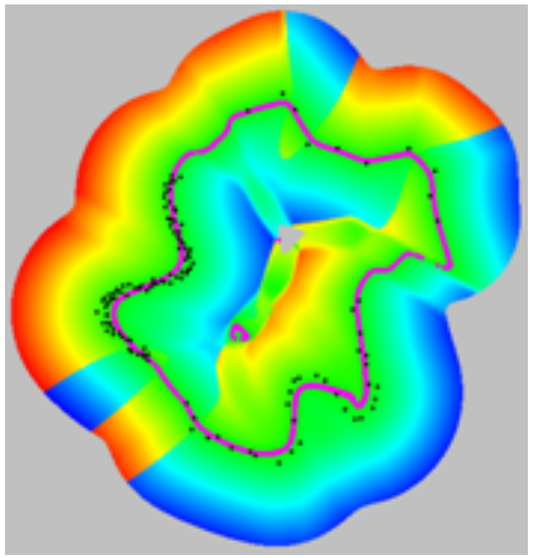
Delaunay graph with pruning

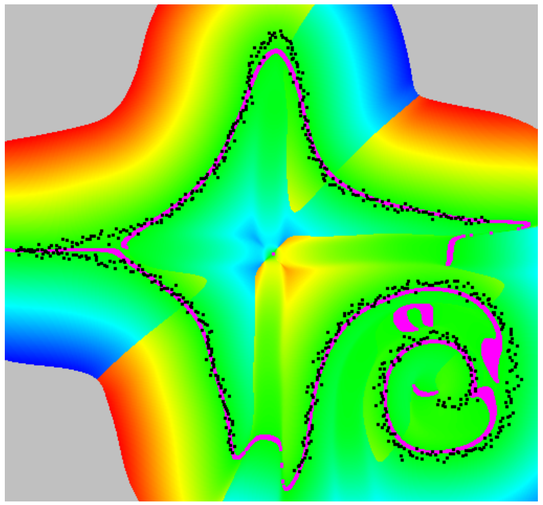


Weighted MLS surfaces using different k-SIGs for the geodesic distance

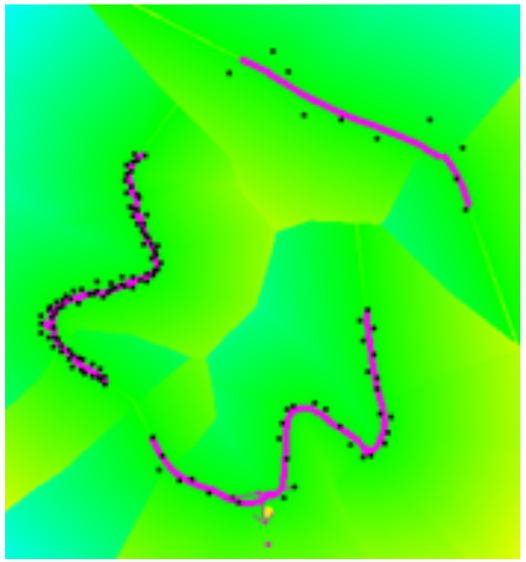
1-SIG 2-SIG 3-SIG

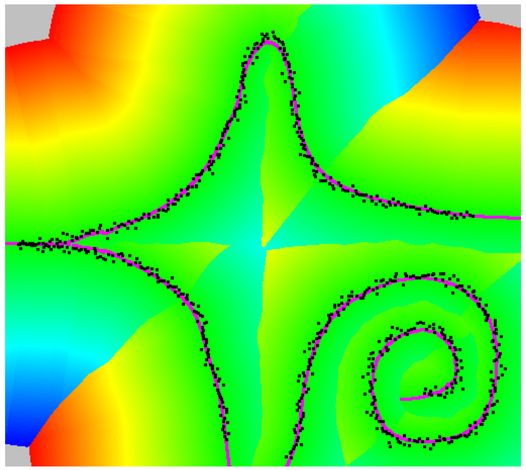
Weighted MLS surface with Euclidean distance and fixed bandwidth in kernel





Weighted MLS surface with proximity graph-based distance and automatic bandwidth estimation in kernel





More info in [Klein & Zachmann, 2004]
on http://cgvr.cs.uni-bremen.de/ → Publications

→ Master's Thesis!