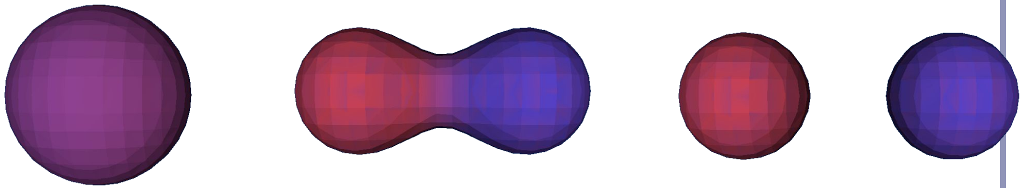
Further Graphics

Implicit Surfaces
Particle Systems



Left: Jose Maria de Espona, REM INFOGRAFICA, 1997 - Metaball model built with MetaReyes 3.0

Right: CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation, ACM Trans. Graph., Vol. 38, No. 4, Article 119. July 2019

Alex Benton, University of Cambridge – alex@bentonian.com

Supported in part by Google UK, Ltd

Implicit surfaces

Signed Distance Fields are just one example of the broad class of *implicit surfaces*.

An implicit surface is any description of a set of points which satisfy the equation

$$F(\mathbf{P}) = 0$$

where $P \in \mathbb{R}^3$ for a 3D surface.

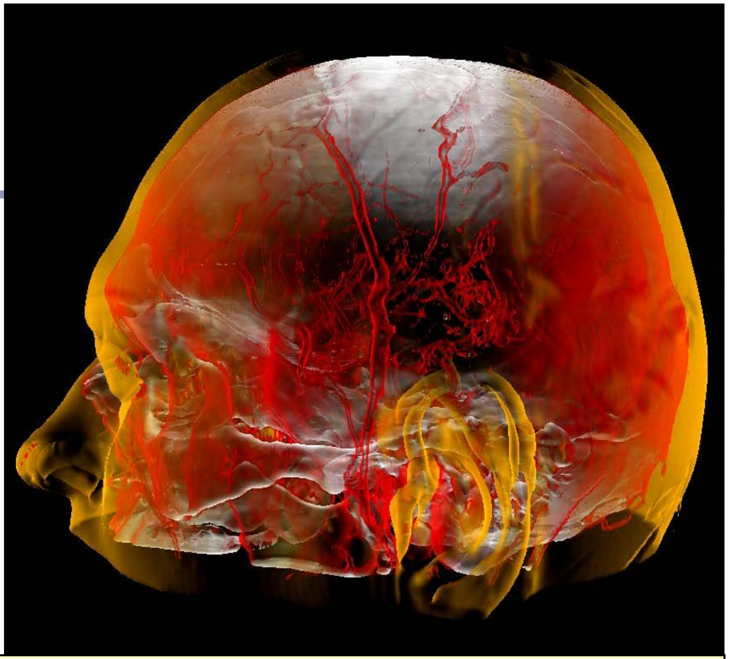
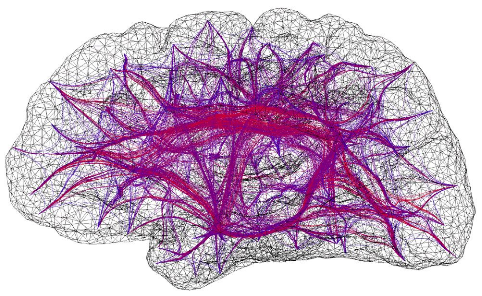


Image credit: Balázs Csebfalvi, Balázs Tóth, Stefan Bruckner, Meister Eduard Gröller **Illumination-Driven Opacity Modulation for Expressive Volume Rendering**, *Proceedings of Vision, Modeling & Visualization 2012*, pages 103-109. November 2012.



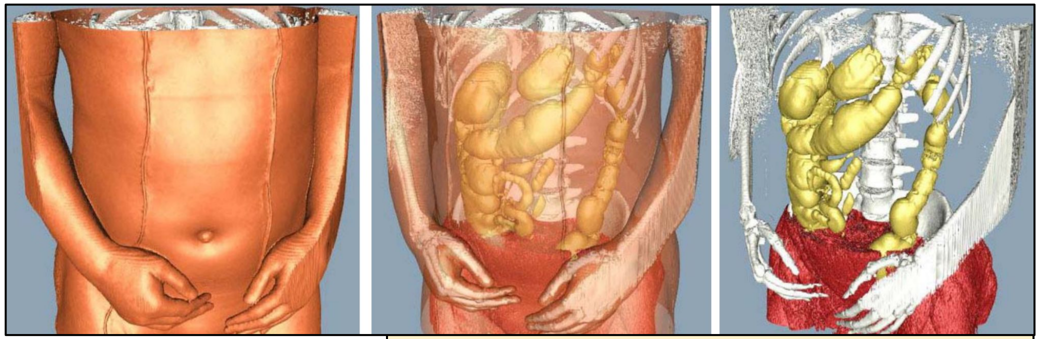


Image credit: W. Lorensen. *Marching Through the Visible Man*, 1995

Implicit surfaces in modern animation: *Metaballs* (sometimes called “metaball modeling”, “force functions”, “blobby modeling”…)

Metaballs are an early (1980s) technique for creating smooth, blobby, organic surfaces.

Metaballs leverage the fact that if two functions $F(P)=0$ and $G(P)=0$ describe implicit surfaces, then $F(P)+G(P)=0$ describes a surface blending both shapes.

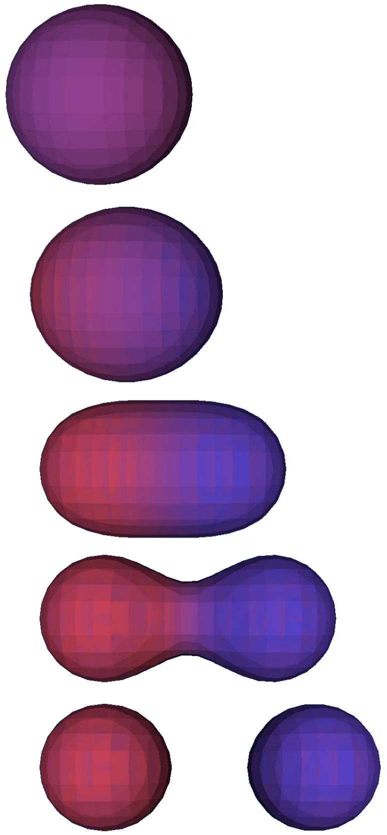
Metaball models are decsribed by a set of *control points*. Each control point p generates a ‘field’ of force, which drops off as a function $F(r)$ where r is the scalar radius from the control point.

The implicit surface is the set of all points in space where the sum of these field equals a chosen constant:

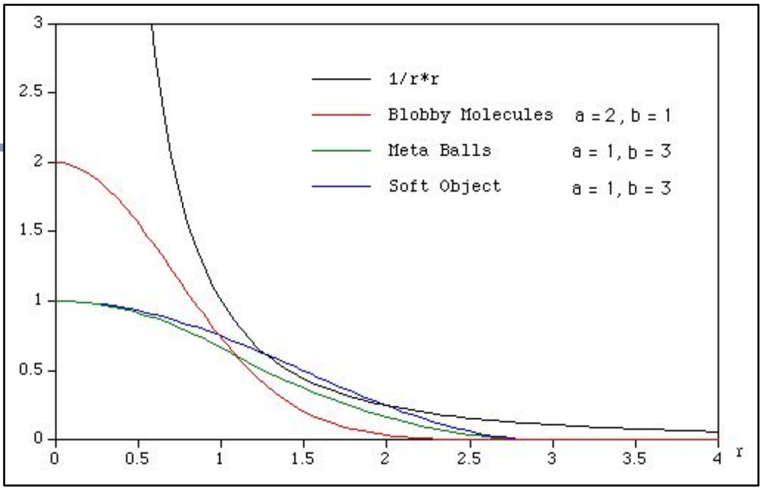
$$S = \{x \in \mathbb{R}^3 \mid \textstyle\sum_p F(|x-p|) = \tau\}$$

The surface thus solves the expression:

$$\textstyle\sum_p F(|x-p|) - \tau = 0$$



Metaball modeling force functions



Common force functions:

- "Blobby Molecules" – Jim Blinn

  $$F(r) = a\, e^{-br^{\wedge}2}$$

- "Metaballs" – Jim Blinn

  $$F(r) = \begin{cases} a(1 - 3r^2 / b^2) & 0 \le r < {}^{b}/_{3} \\ (3a/2)(1 - r/b)^2 & {}^{b}/_{3} \le r < b \\ 0 & b \le r \end{cases}$$

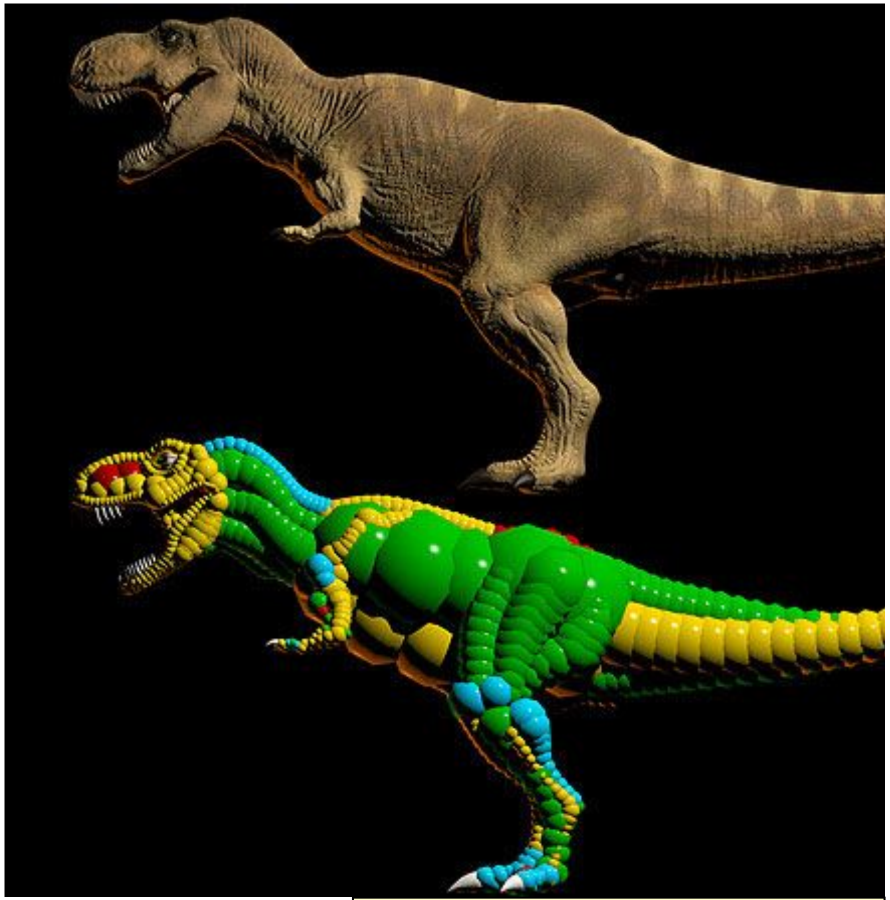
- "Soft Objects" – Wyvill & Wyvill

  $$F(r) = a(1 - 4r^6/9b^6 + 17r^4/9b^4 - 22r^2 / 9b^2)$$

Metaball modeling

Jim Blinn first used blobby models to animate electron orbital shells (1982).

Today animators and artists use blobby modeling to quickly create bumpy, organic surfaces.


Jose Maria de Espona, REM INFOGRAFICA, 1997

Polygonizing implicit surfaces: *Marching Cubes*

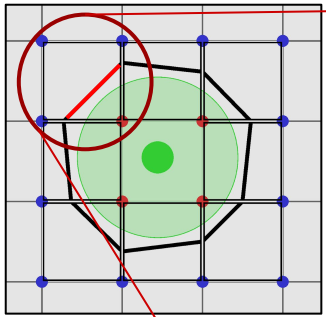
The *Marching cubes* algorithm (Lorensen & Cline, 1985) finds a set of polygons approximating a surface:

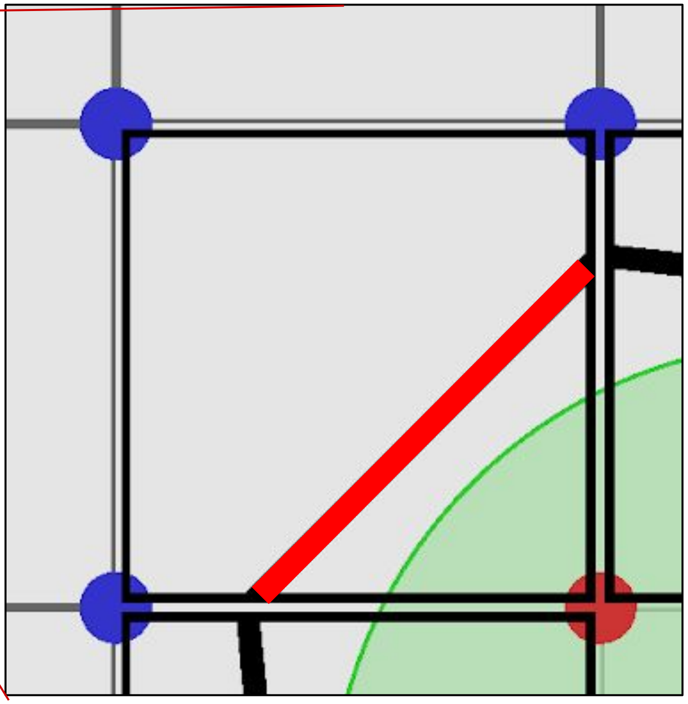
1. Fire a ray from any point known to be inside the surface.
2. Using Newton's method or binary search, find one place where the ray crosses the surface.
3. Place a cube centered at the intersection point: some vertices will be 'hot' (inside the surface), others 'cold' (outside).
4. While there exists a cube which has at least one hot vertex and at least one cold vertex on a side and no neighboring cube sharing that face, create a neighboring cube at that face.

Marching cubes is common in medical imaging such as MRI scans. It was first demonstrated (and patented!) by researchers at GE in 1984, modeling a human spine.

Cubes → Polygons

Each edge of the cube that has 1 hot and 1 cold corner, must be crossed by the isocline of the surface



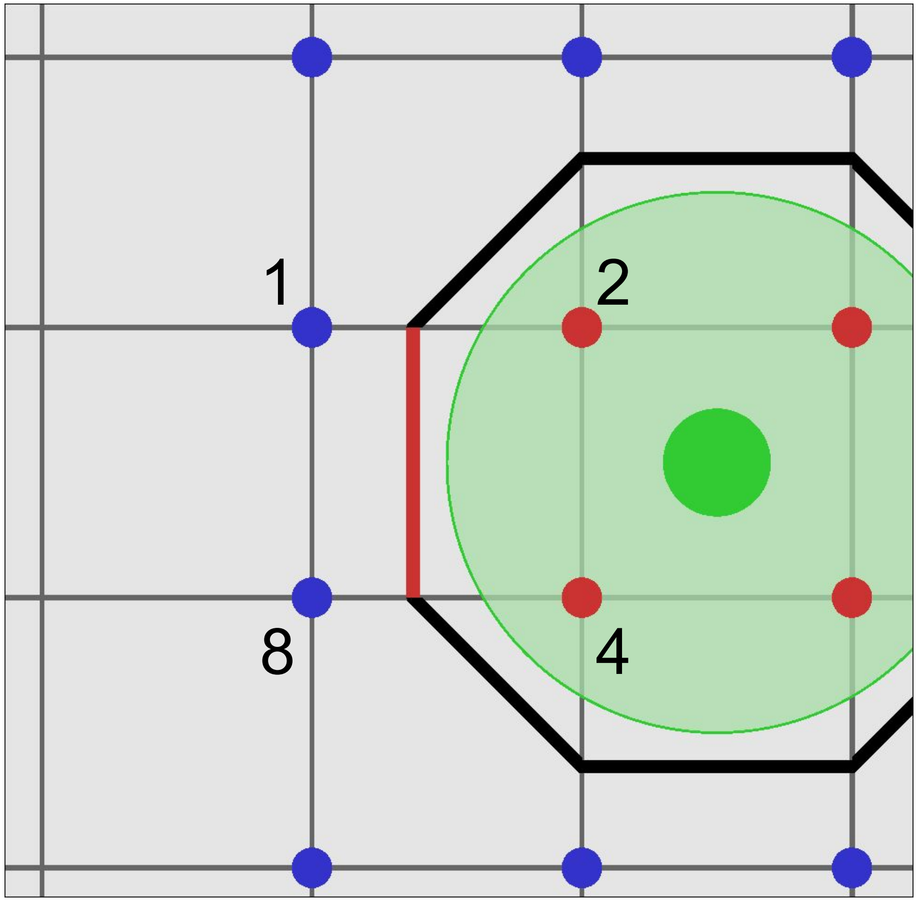


The simplest polygonization is to add a polygon face joining the midpoints of each crossed edge (but we can do better)

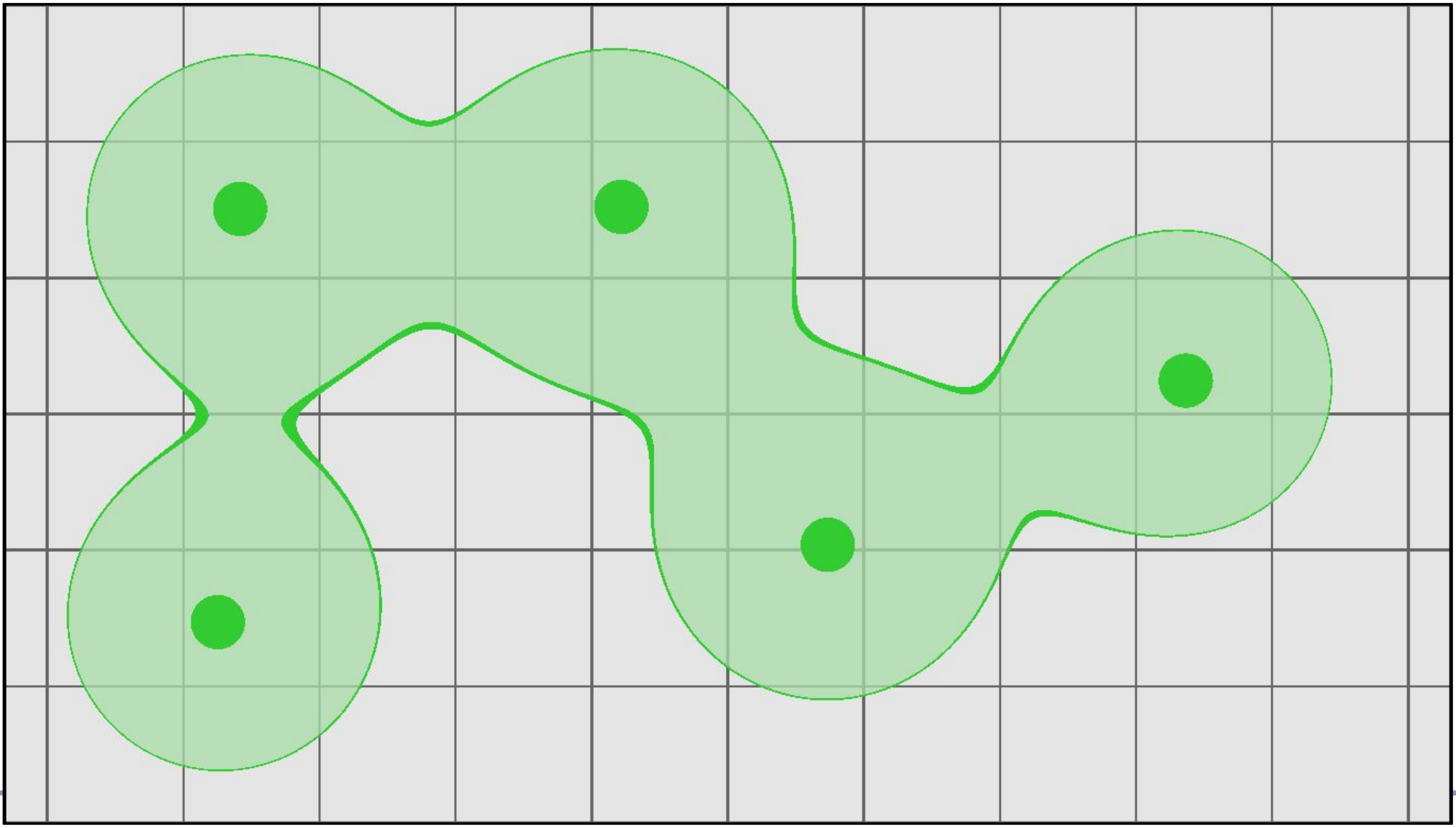
Cubes → Polygons

```
int flags =
   (isHot(T_L) ? 1 : 0) |
   (isHot(T_R) ? 2 : 0) |
   (isHot(B_R) ? 4 : 0) |
   (isHot(B_L) ? 8 : 0);

switch (flags) {
  case 1 :
    // left side ⟷ top;
  ...
  case 3 :
    // left side ⟷ right;
  ...
  case 10 :
    // top ⟷ right side,
    // AND,
    // bottom ⟷ left side
  ...
}
```



Marching ~~cubes~~ squares in action

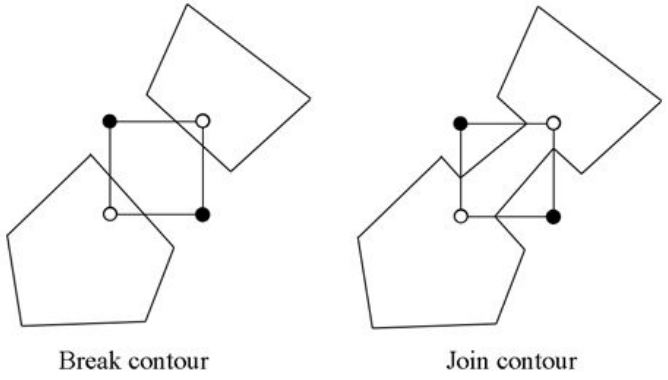


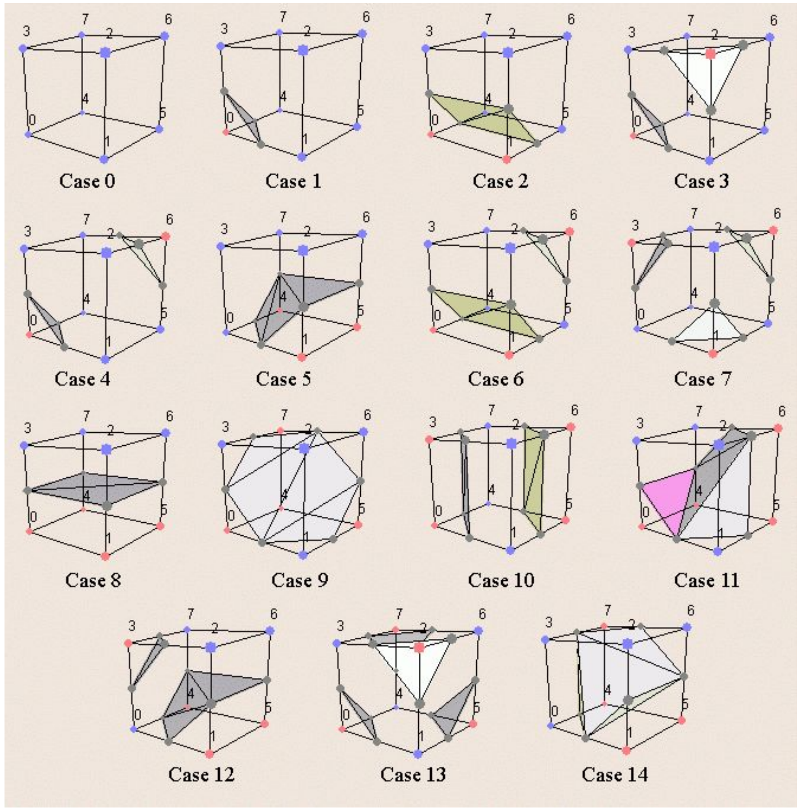
Cubes → Polygons

In 3D, there are fifteen possible configurations (up to symmetry) of hot/cold vertices in the cube. →

- With rotations, that’s 256 cases

Beware: there are *ambiguous cases* in the polygonization which must be addressed consistently. ↓





Images courtesy of Diane Lingrand

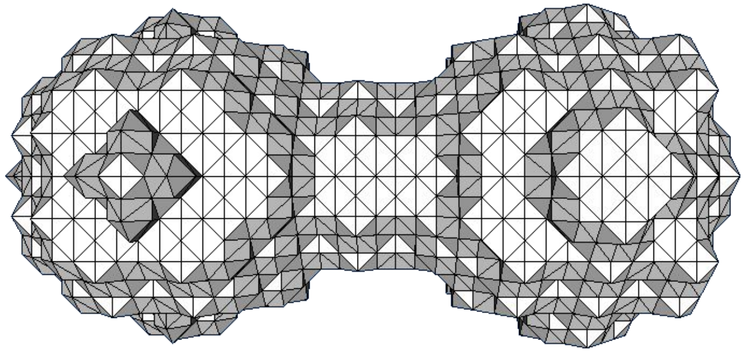
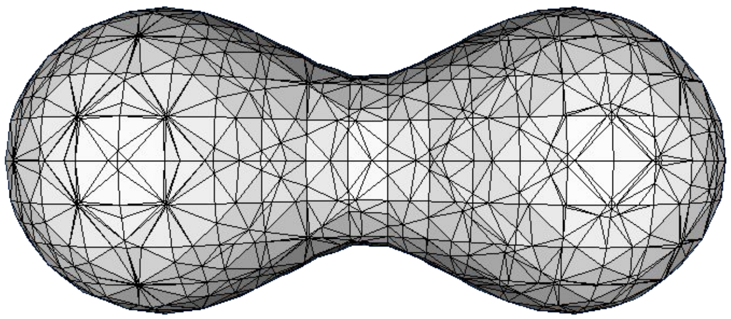
Smoothing the polygonization

The simplest polygonization uses a polygon face joining the midpoints of each crossed edge $P1 \rightarrow P2$:

- $P = P1 + ½ (P2 - P1)$

The implicit surface can be more closely approximated by linearly interpolating along the edges of the cube by the weights of the relative values of the force function:

- $t = (0.5 - F(P1)) / (F(P2) - F(P1))$
- $P = P1 + t (P2 - P1)$

Same implicit surface

Polygonizing implicit surfaces: *Octrees*

The *octree* is a recursive data structure which subdivides space to "home in" on an implicit surface. Each node of an octree is a cube, containing 0 or 8 child octrees.

- Each node of the tree occupies a cube in space
- Each node evaluates the force function F(*v*) at each of its vertices *v*
- Recursive definition: subdivide the cube into 8 equal-sized children for every node where at least 1 corner vertex is inside the surface ('hot') and at least 1is outside ('cold')

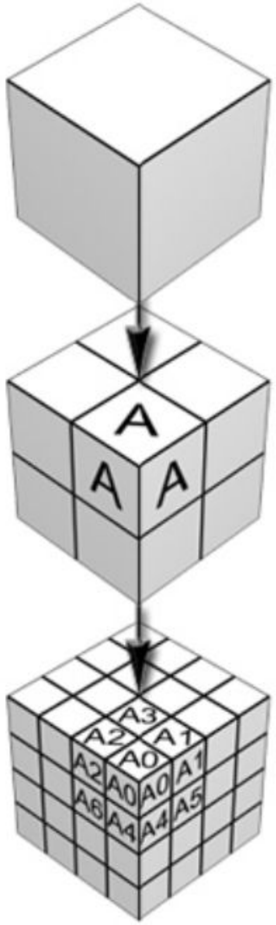


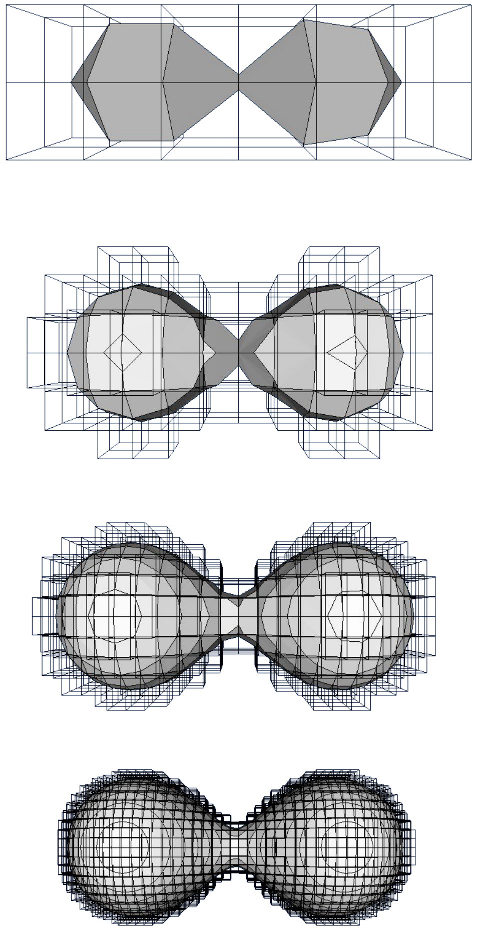
Image credit: J W Laprairie, Mark & Hamilton, Howard. (2018). *Isovox: A Brick-Octree Approach to Indirect Visualization*

Progressive refinement: Octrees

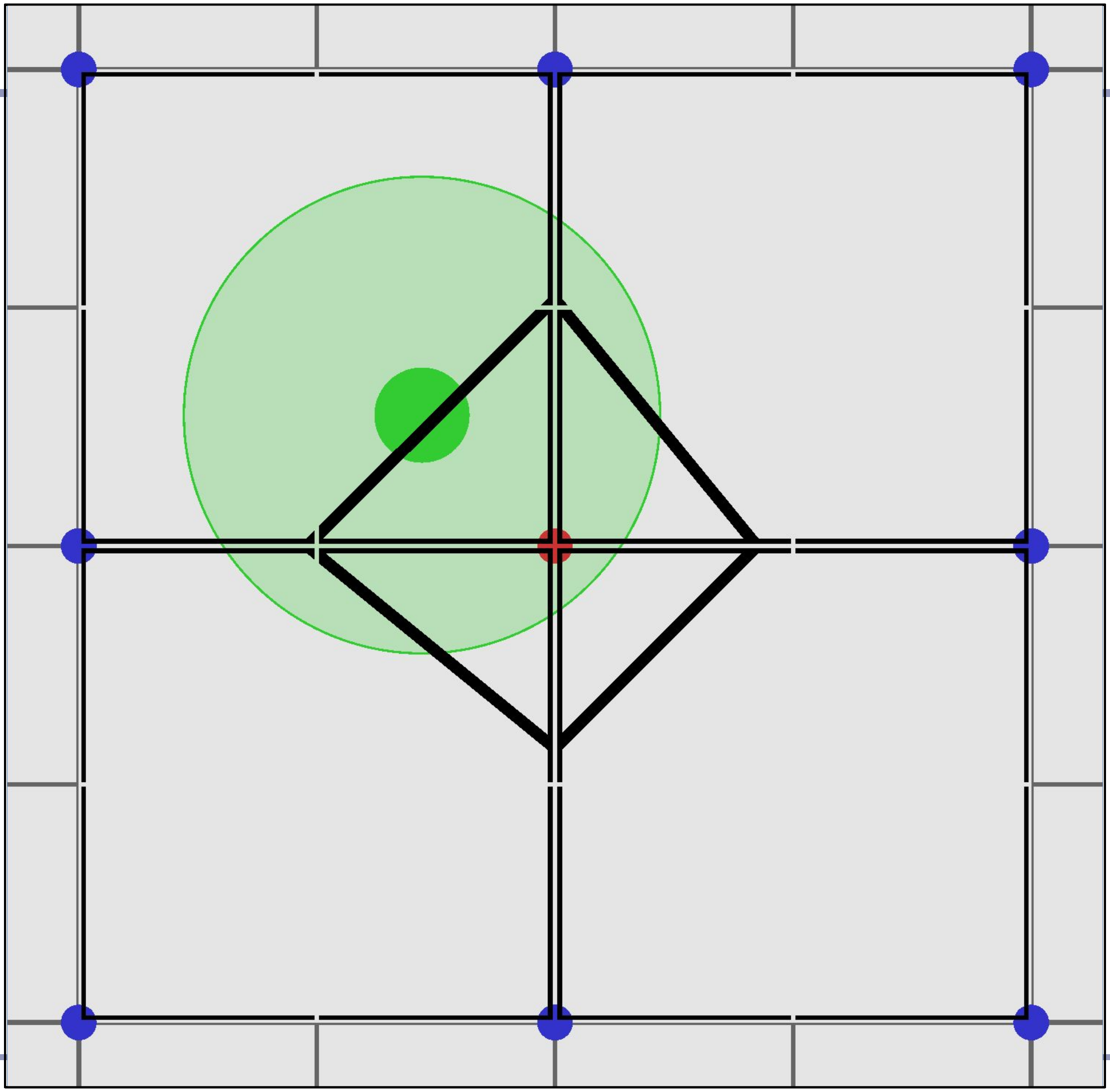
To display a set of octrees, convert the octrees into polygons.

- If some corners are "hot" (inside the surface) and others are "cold" (outside) then the isosurface must cross the cube edges in between.
- The set of midpoints of adjacent crossed edges forms one or more rings, which can be triangulated. The normal is known from the inside/outside direction on the edges.

To refine the polygonization, subdivide recursively; discard any child whose vertices are all inside or all outside.



Octree refinement in action



Particle systems

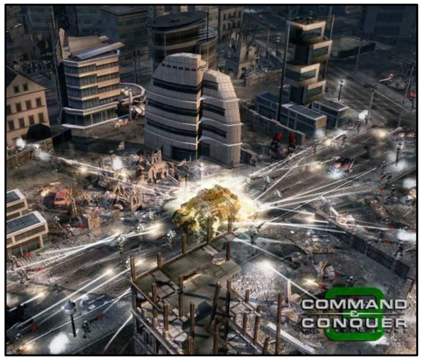
Particle systems are a monte-carlo style technique which uses thousands (or millions) of tiny finite elements to create large-scale structural and visual effects.

Particle systems are used for hair, fire, cloth, smoke, water, spores, clouds, explosions, energy glows, in-game special effects and much more.

The basic ideas:

- "Very simple procedural rules can create very deep visual effects"
- "If lots of little dots all do something coherent, our brains will see the thing they do and not the dots doing it."

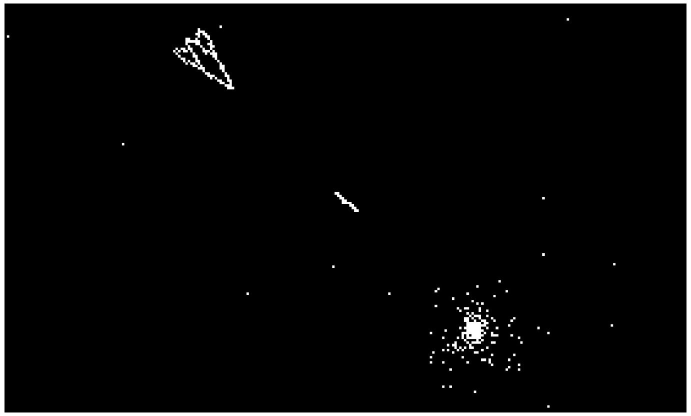
Still from *Large Steps in Cloth Simulation*, David Baraff, Andrew P. Witkin. Published in SIGGRAPH 1998

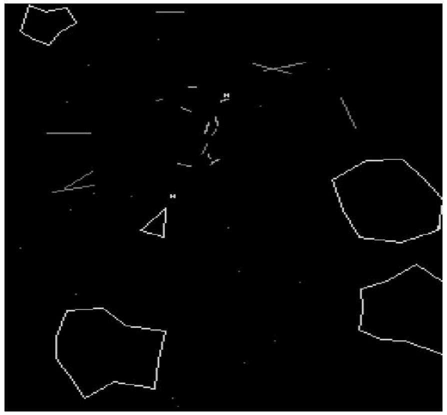


Screenshot from the game *Command and Conquer 3* (2007) by Electronic Arts; the "lasers" are particle effects.

Particle systems' honorable history

- *1962*: Ships explode into pixel clouds in "Spacewar!", the 2nd video game *ever*.
- *1978*: Ships explode into broken lines in "Asteroid"
- *1982*: The Genesis Effect in "*Star Trek II: The Wrath of Khan*"



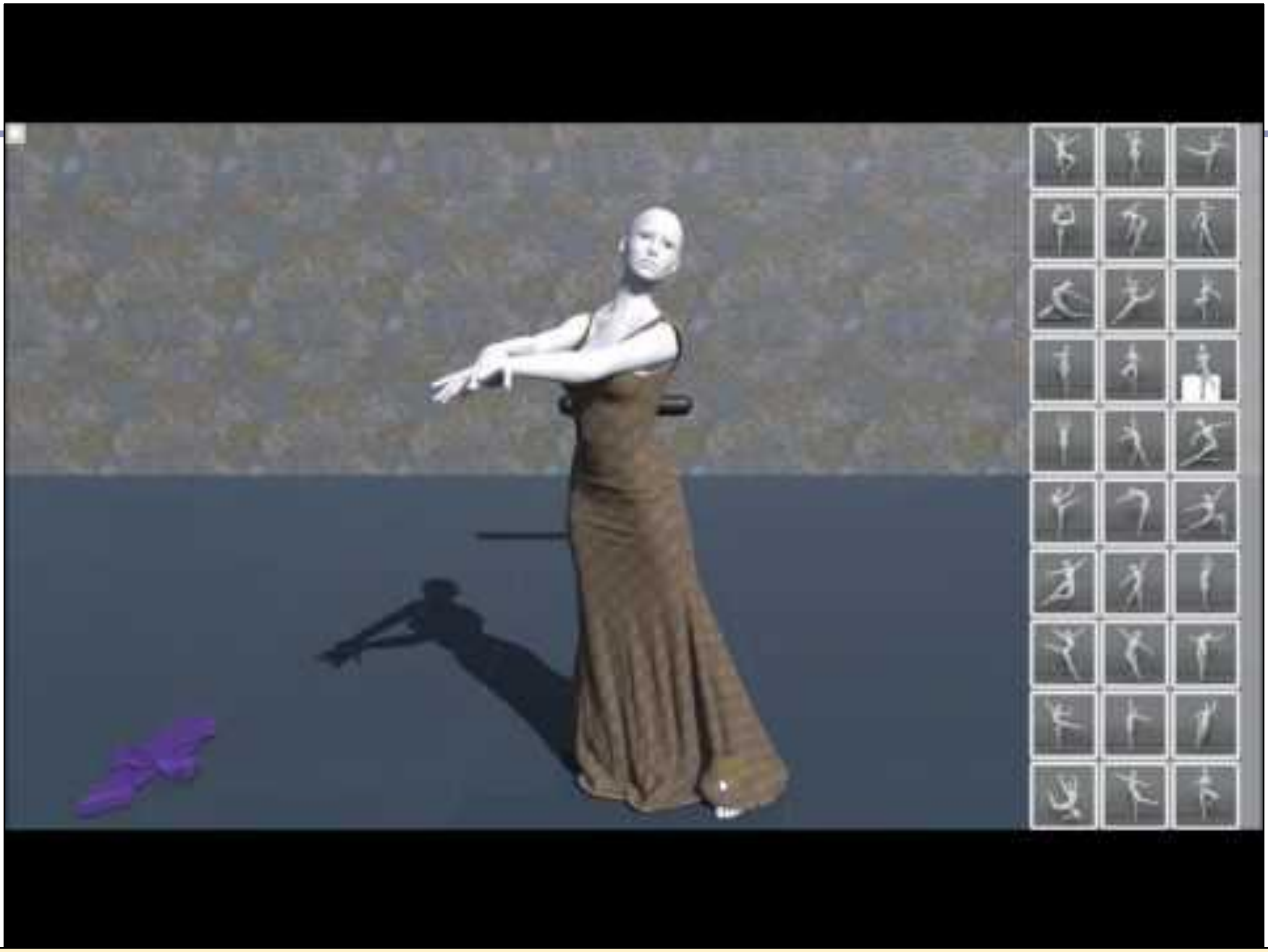


Fanboy note: You can play the original *Spacewar* at spacewar.oversigma.com/ -- the actual original game, *running in a PDP-1 emulator inside a Java applet*.

Particle systems: Fluid simulation

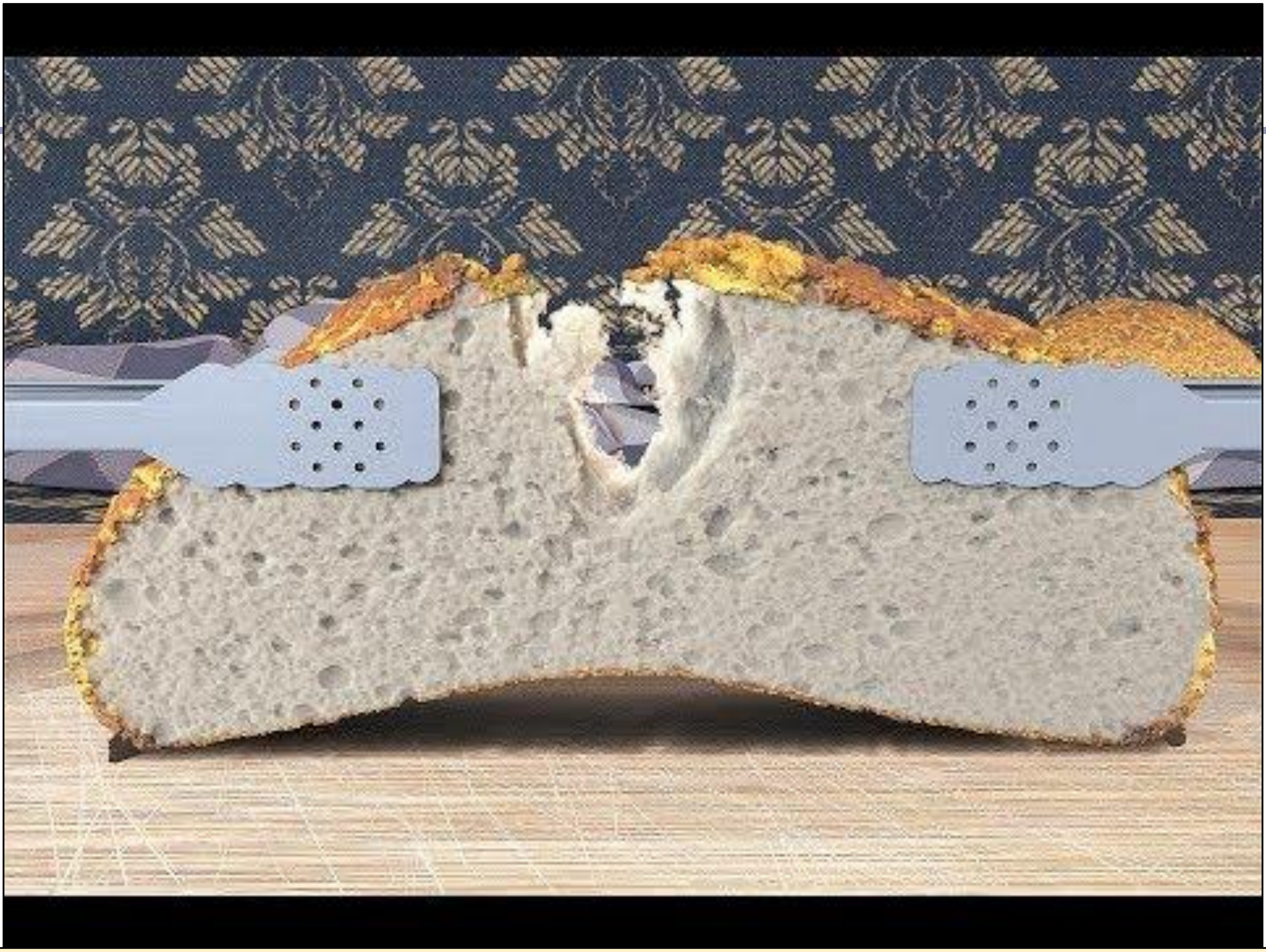
"Position Based Fluids", SIGGRAPH (2013) - Realtime fluid by Miles Macklin and Matthias Müller (NVIDIA)
Supporting material for *Position Based Fluids*, Miles Macklin, Matthias Müller, ACM TOG 32(4) (2013)

Particle systems: Cloth simulation



"Interactive Cloth Simulation", Jim Hugunin - Realtime GPU-driven cloth in Unity game engine
From his talk at Unite 2016, *GPU Accelerated High Resolution Cloth Simulation*

Particle systems: Fracture simulation



"CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation"
Video for SIGGRAPH 2019 for the paper *CD-MPM: Continuum Damage Material Point Methods for Dynamic Fracture Animation*, ACM Trans. Graph., Vol. 38, No. 4, Article 119. July 2019

How does it work?

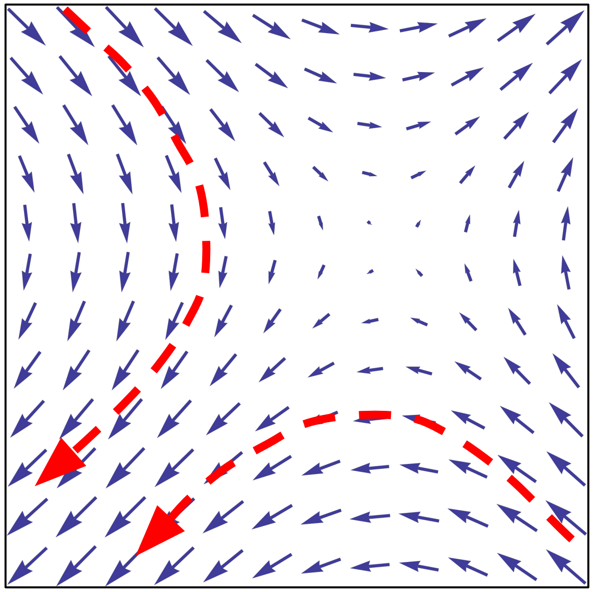
We want to ask,

- "A particle starts life with initial position and velocity. Given obstacles / forces / constraints, where will it wind up?"

or in other words…

- Solve this:
  - Given v=$dX/dt=f(X(t),t)$
  - Given $X(t_0) = X_0$
  - Find $X(t)$ for $t > t_0$

This is an ODE



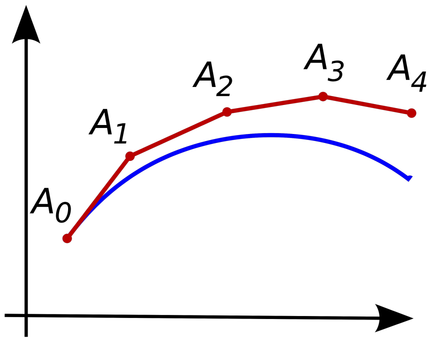
where $X(t)$ is the particle position, dX/dt is the particle velocity, X_0 is its initial position and $f(X(t),t)$ is a (complicated? time- and position dependent?) equation that changes particle velocity

Particle systems as Ordinary Differential Equations: Euler's Method

There are many ways to solve an ODE. The simplest (and most common in realtime graphics) is *Euler's Method*.

- "The forward difference method (Euler's Method) uses the rate values at the end of one timestep as though constant in the next timestep." --*Numerical Methods*

This is effective, albeit error-prone



← Each step tangent to the path *could* take us further from the true path

← But we will still approximate the integral 'reasonably', for small enough steps

← Error can be bounded by short particle lifetimes, damping, and other practical tricks

Figure from wikipedia

Example 1

A simple example--particles affected by gravity:

$$v(t) = v_0 + gt$$

(This has a known solution, because physics: $X(t) = X_0 + v_0 t + 1/2gt^2$)

Approximated with Euler's method:

```
For each frame:
  For each particle:
    velocity = velocity
          + timestep * gravity
    position = position
          + timestep * velocity
```

This generalizes nicely to array-multiply and array-add operations which scale well on modern GPU hardware, allowing you to update velocity and position in a single GPU raster operation

Animation: https://www.syncronorm.com/products/depence2/visualization/special-fx/

Example 2

A more complex example--particles affected by a position-dependent wind or force:

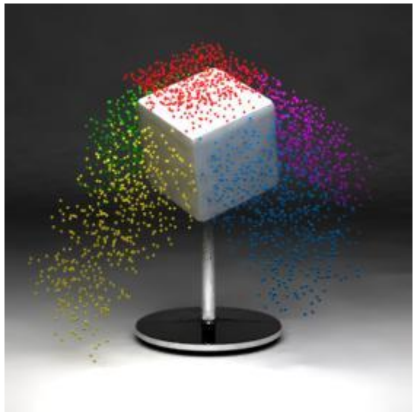
$$v(t) = v_0 + wind(x(t))$$

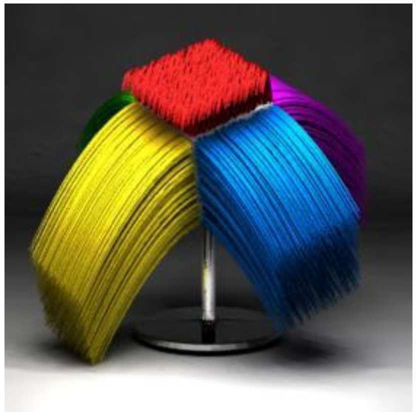
```
For each frame:
  For each particle:
    look up wind at position
    solve f=ma to find the
       acceleration of the wind on the
       mass of the particle
    velocity = velocity
         + timestep * wind_accel
    position = position
         + timestep * velocity
```

This still generalizes nicely to modern GPU hardware, although as complexity rises, more advanced GPU languages like CUDA may be more appropriate

Common particle system design

1. Particles are generated from an *emitter* with initial mass, position, velocity
   a. Emit rate, direction, flow, etc are often specified as a bounded random range (monte carlo)
2. Time ticks; at each tick, particles move by *dt * velocity*
   a. New particles are generated; expired particles are deleted
   b. Forces (gravity, wind, etc) accelerate the velocity of each particle
   c. Collisions and other interactions update velocity
      i. Ex: 'density' constraints for liquids
      ii. Ex: 'spring' constraints for cloth
   d. Velocity changes position
3. Particles are rendered





Transient vs persistent particles emitted to create a 'hair' effect (source: Wikipedia)

Particle systems—rendering

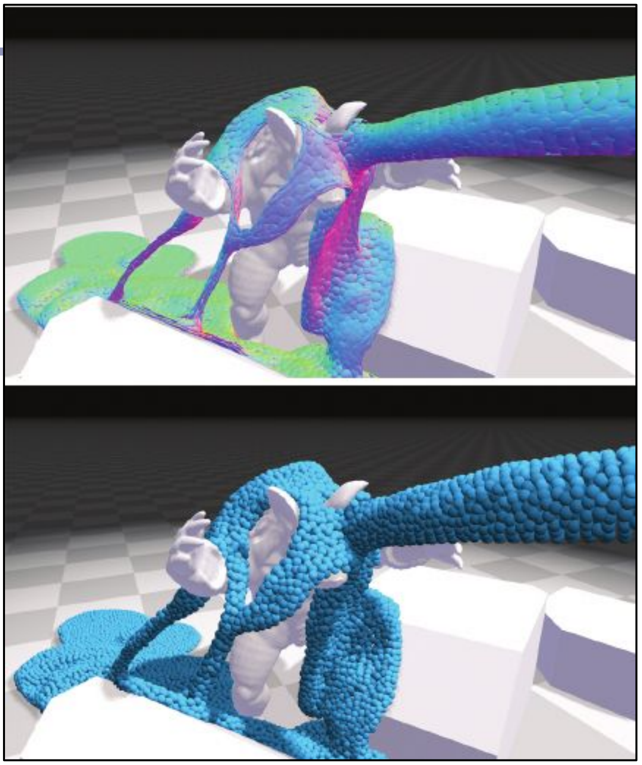
Particles can be rendered as points, textured polys, primitive geometry...

- Polygons with alpha-blended images make pretty good fire, smoke, etc

Transitioning one particle type to another creates realistic interactive effects

- Ex: a 'rain' particle becomes an emitter for 'splash' particles on impact

Implicit surfaces or *ellipsoid splatting* are popular algorithms for rendering particle system point clouds as liquid surfaces



Ihmsen, Markus & Orthmann, Jens & Solenthaler, Barbara & Kolb, Andreas & Teschner, Matthias. (2014). *SPH Fluids in Computer Graphics - Eurographics State-of-the-art report*

“The Genesis Effect” – William Reeves
Star Trek II: The Wrath of Khan (1982)

References

Implicit modelling

- D. Ricci, *A Constructive Geometry for Computer Graphics,* Computer Journal, May 1973
- J Bloomenthal, *Polygonization of Implicit Surfaces,* Computer Aided Geometric Design, Issue 5, 1988
- B Wyvill, C McPheeters, G Wyvill, *Soft Objects*, Advanced Computer Graphics (Proc. CG Tokyo 1986)
- B Wyvill, C McPheeters, G Wyvill, *Animating Soft Objects,* The Visual Computer, Issue 4 1986

Marching Cubes

- www.youtube.com/watch?v=M3iI2l0ltbE (very nice visualization)

Particle Systems

- William T. Reeves, "*Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*", Computer Graphics 17:3 pp. 359-376, 1983 (SIGGRAPH 83).
- David Baraff, Andrew Witkin, *Large Steps in Cloth Simulation*, SIGGRAPH 1998
- Leif Kobbelt and Mario Botsch, *A survey of point-based techniques in computer graphics*, Computers & Graphics Volume 28, Issue 6, December 2004, Pages 801-814
- Ihmsen, Markus & Orthmann, Jens & Solenthaler, Barbara & Kolb, Andreas & Teschner, Matthias. (2014). *SPH Fluids in Computer Graphics - Eurographics State-of-the-art report*
- www.gdcvault.com/play/1024344/D3D-Async-Compute-for-Physics (nice summary)
- nullprogram.com/webgl-particles/ (nice browser-based demo)