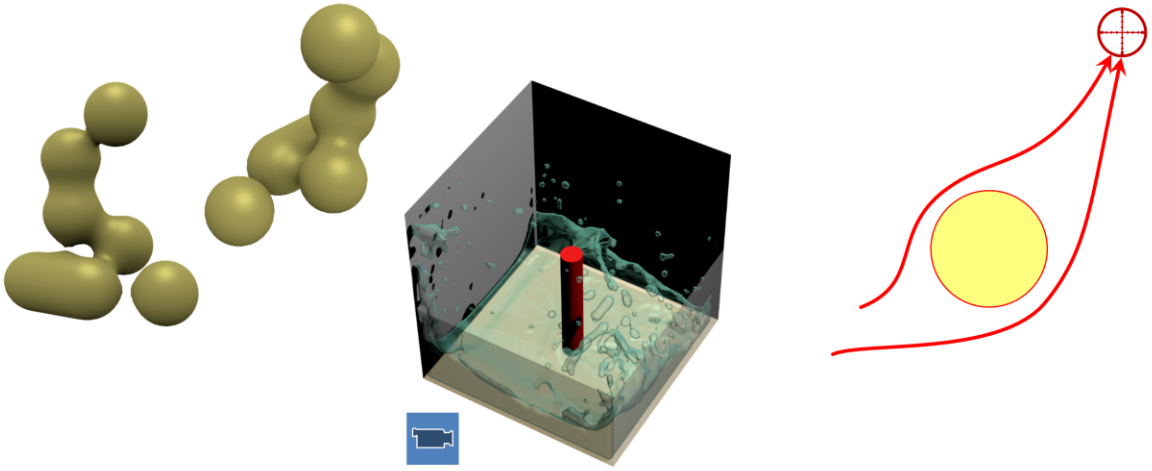
Implicit Geometry

Graham Rhodes
Applied Research Associates, Inc.

GAME DEVELOPERS CONFERENCE®
MOSCONE CENTER · SAN FRANCISCO, CA
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Examples



Left-most image: “metaballs” or “blobbies” merge together to generate joined blobs that resemble a lava lamp. Fun technique (at least for the first 5 minutes), and readily recognizable.d Not likely to be used for modeling general scene geometry. Underlying volumetric field may be used to model lighting.

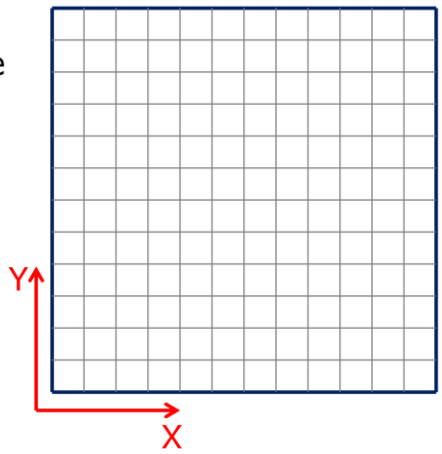
Middle image: in games, implicit surfaces commonly used to generate surfaces to render simulated fluids

- SPH simulation…Lagrangian approach. This is common for real time physics as it can leverage RBD collision system for fluid/solid interaction, requires no gridding, etc. Implicit surface generated using metaballs approach based on particle position with size based on the mass of fluid carried in that particle.
- Eulerian-based fluids solved on a grid, such as Lattice Boltzmann. Fluid properties such as density are known at each cell of a grid, and level set implicit geometry methods can be used to extract a surface mesh to render.

Right-most image: simple vector fields are sometimes used for path planning…let target point be a magnet, or “sink,” and obstructions a “source”. Set an initial velocity and let the source push the object (bot, character, particle) away from the source and pull it towards the sink. The source and sink increment the velocity to generate an obstacle avoiding path towards the target. Trace streamlines to generate implicit path that is pushed away from obstructions and pulled towards the target.

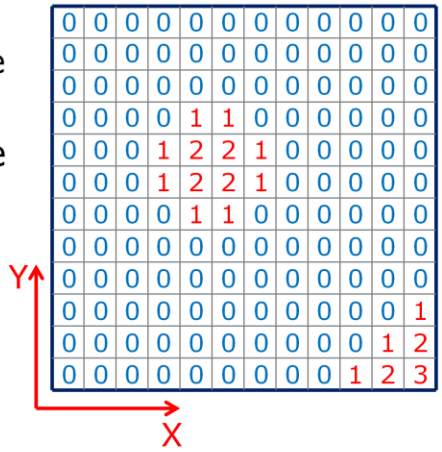
Fundamental Idea – Level Set

- Start with a scalar field function
  - Has a value everywhere in space



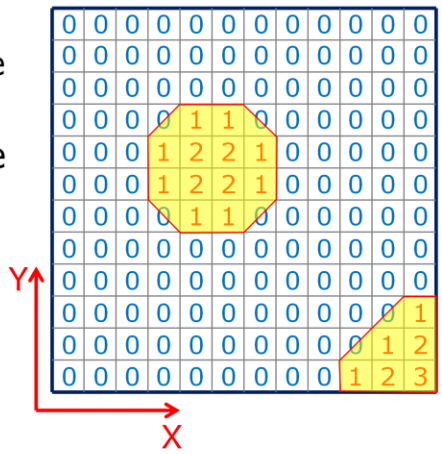
Fundamental Idea – Level Set

- Start with a scalar field function
  - Has a value everywhere in space
- Evaluated on a grid, each cell or cell corner holds the value of the field function



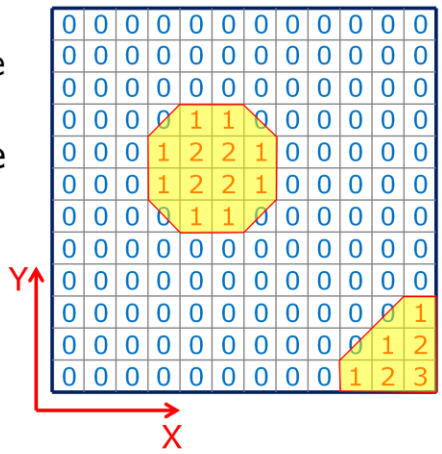
Fundamental Idea – Level Set

- Start with a scalar field function
  - Has a value everywhere in space
- Evaluated on a grid, each cell or cell corner holds the value of the field function
- Pick a threshold value
- Using magic, find the level set isosurface



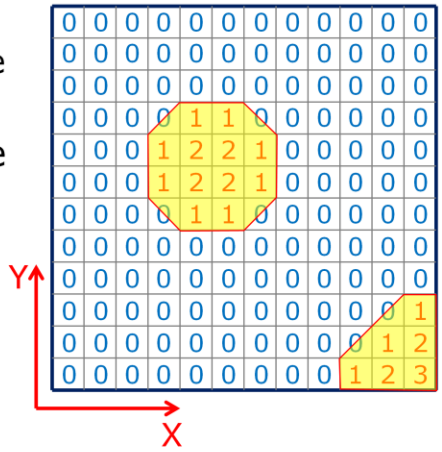
Fundamental Idea – Level Set

- Start with a scalar field function
  - Has a value everywhere in space
- Evaluated on a grid, each cell or cell corner holds the value of the field function
- Pick a threshold value
- Using magic, find the level set isosurface



Fundamental Idea – Level Set

- Start with a scalar field function
  - Has a value everywhere in space
- Evaluated on a grid, each cell or cell corner holds the value of the field function
- Pick a threshold value
- Use case 1: extract mesh of the level set isosurface using marching cubes/tetrahedrons
- Use case 2: ray tracing



For extraction of a mesh, or "polygonalization," marching cubes (squares in 2D) has ambiguities while marching tetrahedrons (triangles) avoids those ambiguities. See one of the many, many tutorials/references available on the Internet.

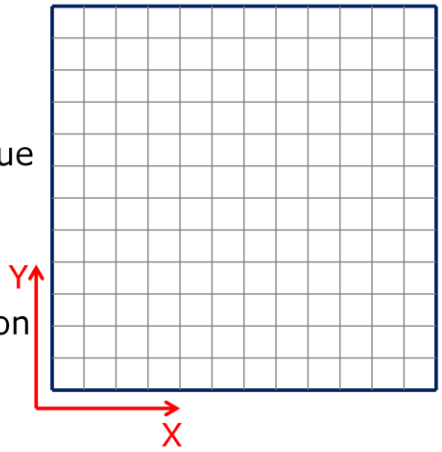
The mesh extraction use case might, for example, be used to draw the fluid surface illustrated on the first slide.

I’m Interested In Modeling With Implicit Functions

- Surfaces
- Containment volumes and light fields
- Animation Paths

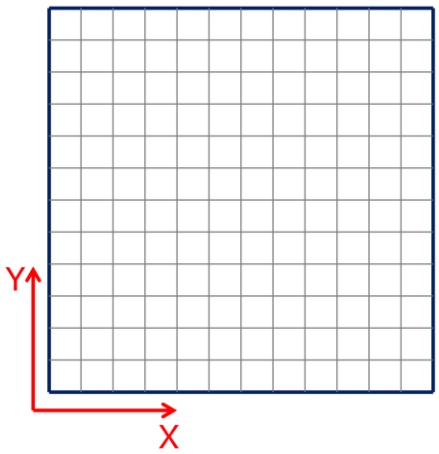
What Can the Function Be?

- Examples:
  - Fluid density from simulation
  - Noise function
  - Anti-aliased material density value from voxel modeling
  - Signed distance field
  - Texture or photo ready for computer vision feature extraction
  - Light field for volume lighting



What Can the Function Be?

- A superposition of kernel functions
- (Usually) continuous throughout the field
- Radial Basis Functions (RBF)



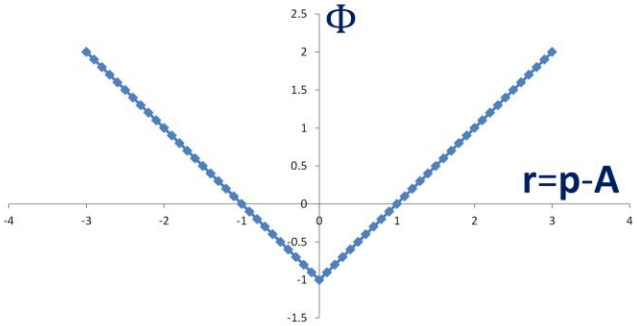
Idea is that you place these kernel functions at strategic locations, and when summed across the domain, the result is a smooth field function from which geometry can be extracted.

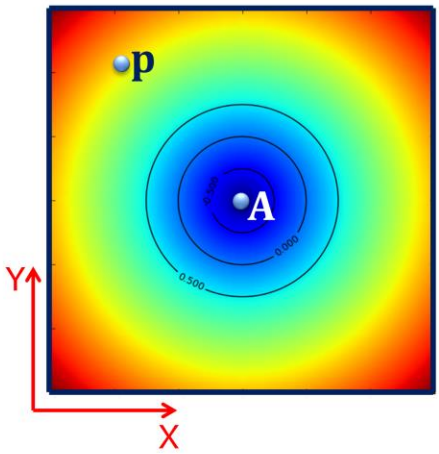
Radial Basis Functions are a class of kernel functions that are well studied for use in implicit modeling.

Radial Basis Functions: Examples

- Signed distance field for sphere

$$\Phi(\mathbf{p}) = |\mathbf{p} - \mathbf{A}| - r_{sphere}$$





Scatter plots are through centerline. Horizontal axis is distance from center, and vertical axis is the value of the RBF.

Blue is low value. Red is high value.

p is an arbitrary point in the field and A is the center of the RBF.

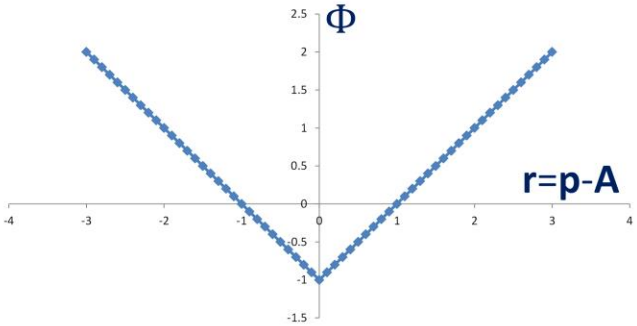
Here, the curve represents the signed distance field for a unit radius circle or sphere located at the origin. Notice that the value of the field is negative inside the sphere and positive outside the field, and the sphere itself is located at isosurface value 0.

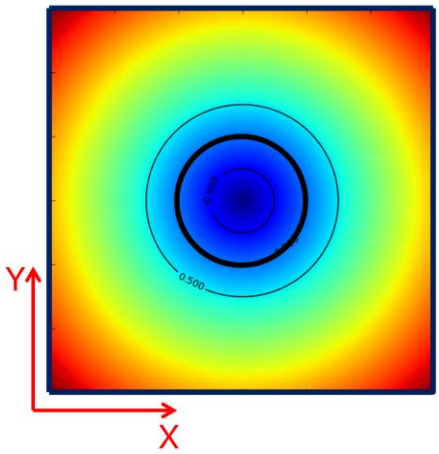
Obviously this could negated for positive values in the interior and negative values in the exterior.

Radial Basis Functions: Examples

- Signed distance field for sphere

$$\Phi(\mathbf{p}) = |\mathbf{p} - \mathbf{A}| - r_{sphere}$$



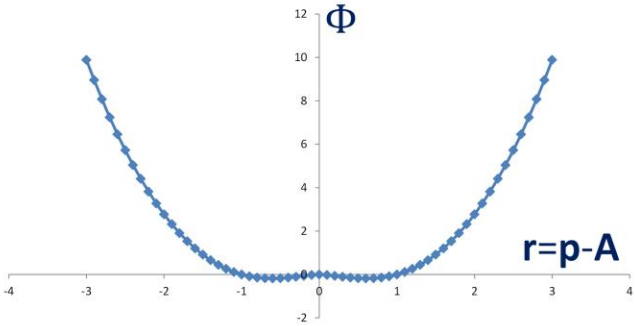


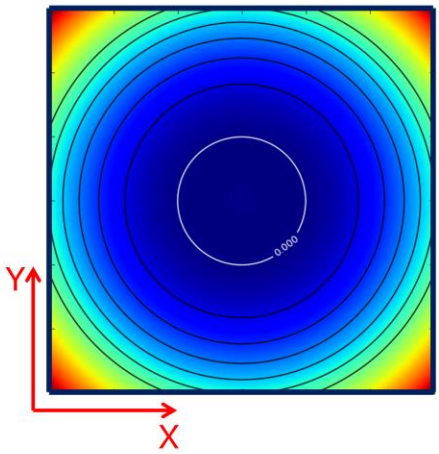
This is useful for rendering specific singular geometries, such as circles/spheres since we can define the function such that isosurface zero is on the surface. The thick dark circle in the picture is isosurface zero. The discontinuity at r = 0 makes this function less desirable for more sophisticated modeling.

Radial Basis Functions: Examples

- Biharmonic RBF

$$\Phi(\mathbf{p}) = |\mathbf{p} - \mathbf{A}|^2 \ln(|\mathbf{p} - \mathbf{A}|)$$





This is also called the thin plate spline RBF.

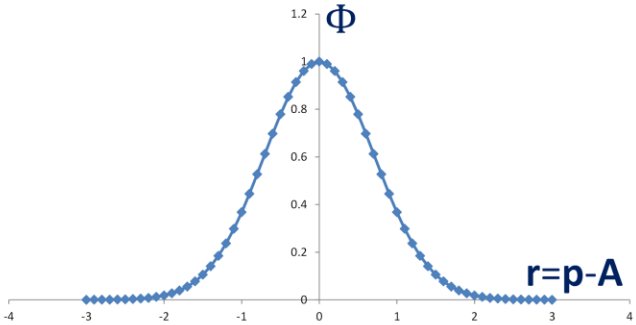
In this case, notice there are two zero points…one at the center and one at p-A = 1.

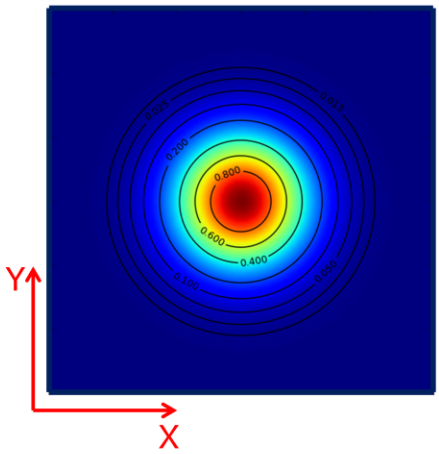
Also notice this function has a continuous first derivative. The second and higher derivatives have discontinuities when p = A. So, here we have C^1 continuity where the signed distance field gave us C^0. This function should enable modeling smoother implicit surfaces…we'll get into that in a few slides.

Radial Basis Functions: Examples

- Gaussian

$$\Phi(\mathbf{p}) = \exp(-\varepsilon^2(\mathbf{p}-\mathbf{A})^2)$$



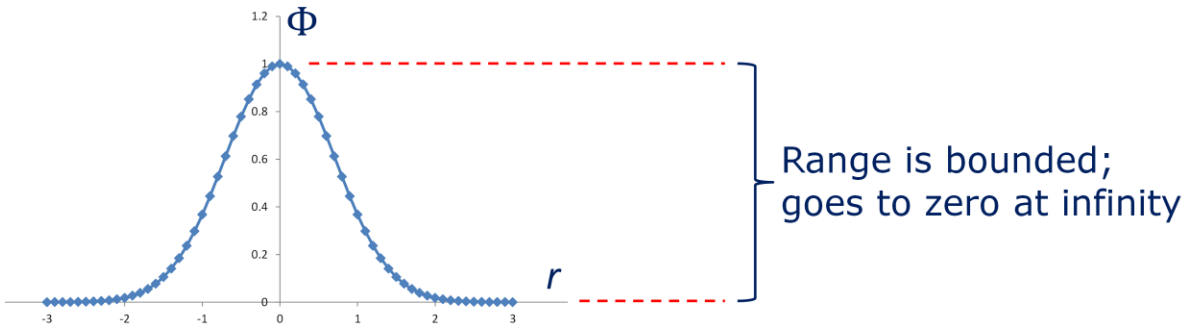


The Gaussian distribution is nice because it dissipates to zero at infinity. This is intuitive, and you can easily see what would happen if you superimpose several kernels across the field. Easier to mentally "see" what the field function will look like in combination with several such RBF kernels.

Radial Basis Functions: Selection

- There are many other choices
- Choice of function depends on use case
- **An important characteristic is the nature of the "support" of the RBF of choice**
  - Compact or finite
  - Non-compact or infinite

Compact/Finite Support

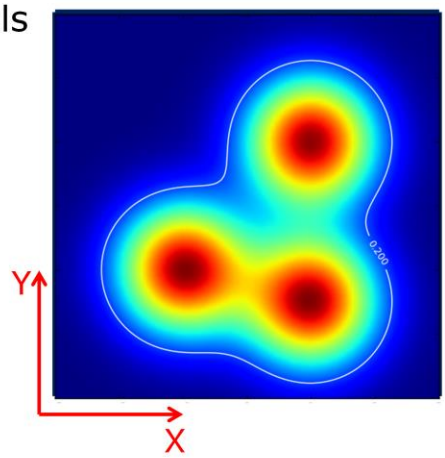


Notice that this RBF has a range that is bounded between 0 and 1. It has compact support, and goes to zero at infinity.

Compact/Finite Support

- Field is sum of n Gaussian kernels

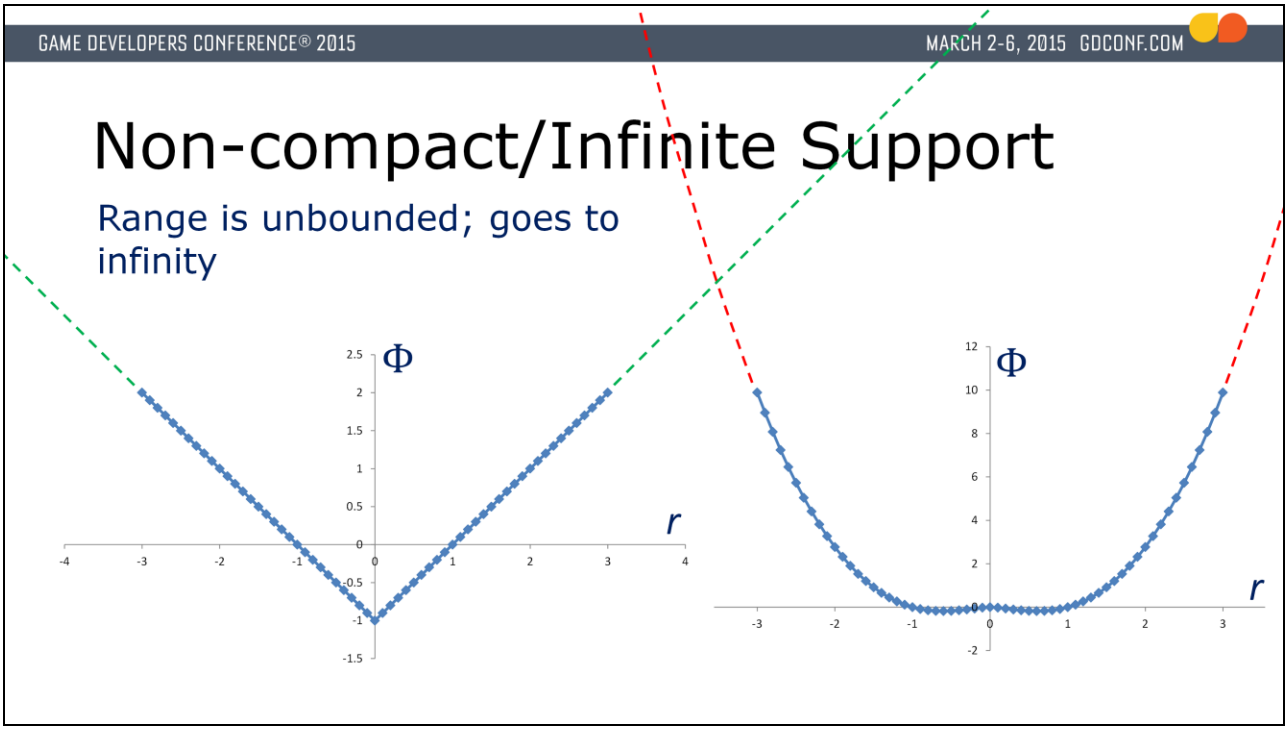
$$\Phi(\mathbf{p}) = \sum_{j=1}^{n} \exp(-\varepsilon_j{}^2(\mathbf{p} - \mathbf{A}_j)^2)$$



This looks like the function used to generate metaballs/blobby surfaces. It is intuitive. Value is interpolated between points, but fall off ensures there is a local maximum at each point; each point remains distinguishable. There is an obvious look to metaballs/blobbies that is impossible to disguise. For some applications, such as light field modeling in which the geometry of isosurfaces is never directly rendered, this may be suitable. For modeling game level geometry, this function yields geometry that has a distinctly procedural/generative look that may rarely be suitable.

Non-compact/Infinite Support

Range is unbounded; goes to infinity

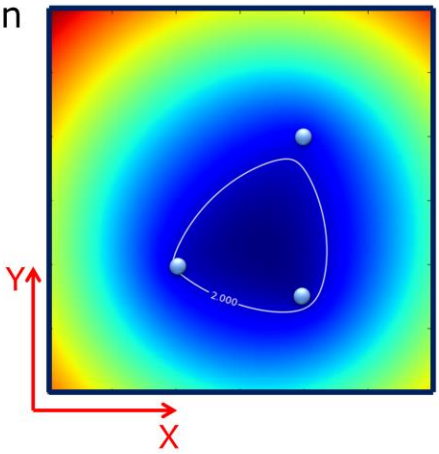


Both of these RBF's have non-compact support, with a range that goes to infinity.

Non-compact/Infinite Support

- Field is sum of n kernels based on signed distance field for sphere

$$\Phi(\mathbf{p}) = \sum_{j=1}^{n} (r_{sphere,j} - |\mathbf{p} - \mathbf{A}_j|)$$

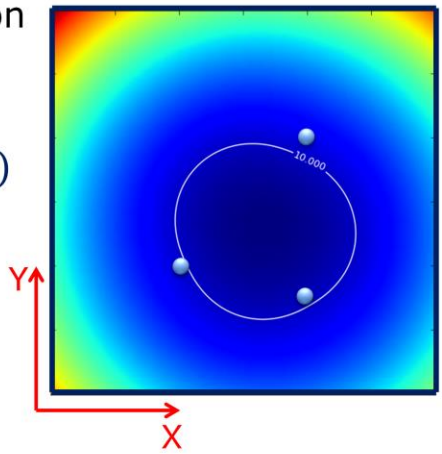


Back to the signed distance field. Notice that this function fills in the space between the kernel points, unlike the Gaussian kernel.

Non-compact/Infinite Support

- Field is sum of n kernels based on the biharmonic RBF

$$\Phi(\mathbf{p}) = \sum_{j=1}^{n} (|\mathbf{p} - \mathbf{A}_j|^2 \ln(|\mathbf{p} - \mathbf{A}_j|))$$

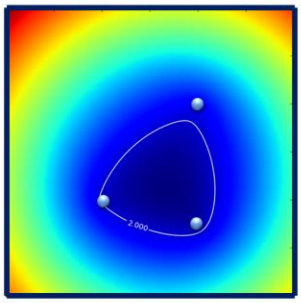


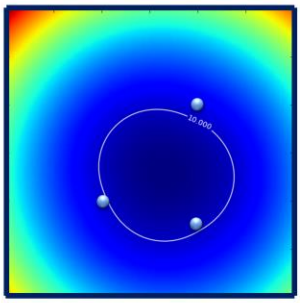
Similarly, the biharmonic RBF fills the space between the kernel points.

Infinite support is the thing that makes this happen. The effect of each kernel is to create a field that fills all space with nonzero values, unbounded at infinity.

Interpolating Scattered Point Data

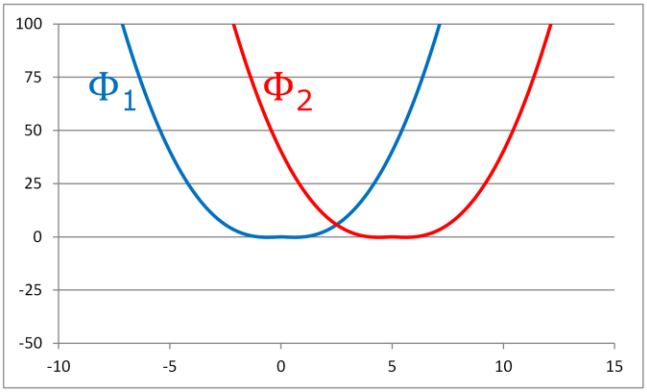
- Let's find a way to adjust the field to interpolate the kernel center points
- Then, we can use the kernel center points as control geometry for surface modeling





Interpolating Scattered Point Data

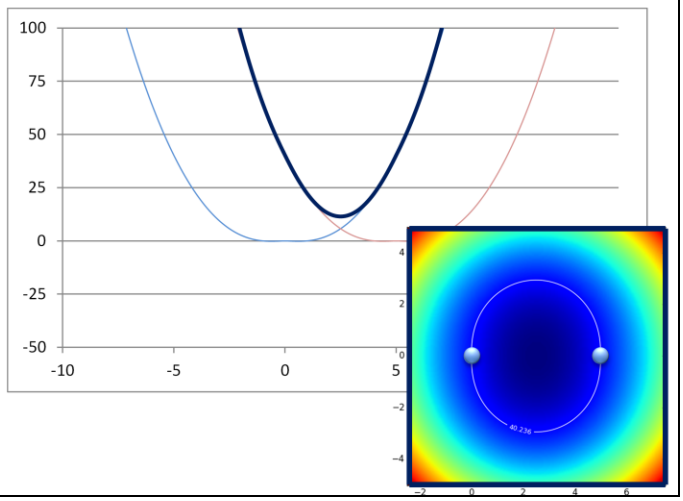
- Kernels at center points (0,0) and (5,0)
- Individually, each kernel has value 0 at its center



These are biharmonic RBF kernels

Interpolating Scattered Point Data

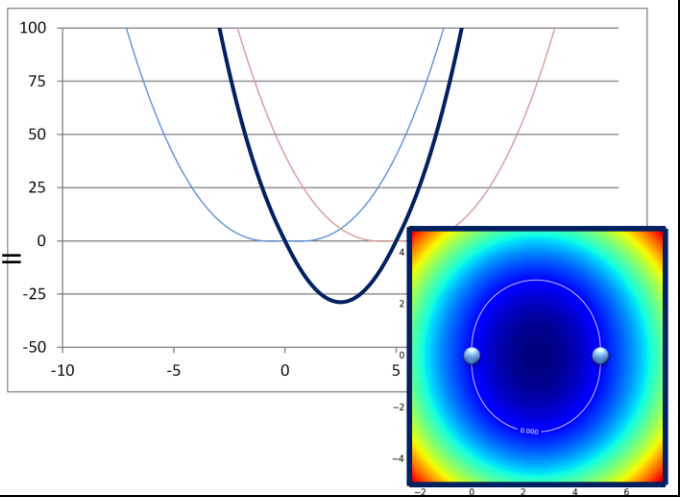
- Kernels at center points (0,0) and (5,0)
- Individually, each kernel has value 0 at its center
- Summed together, both points are on one isocurve, but…
  - We'd like isocurve value = 0
  - It isn't. Inconvenient
  - How can we change this?



There is an isocurve that interpolates the points, but it's value is not known without evaluating the field. Ideally, we'd like to control the value of the isocurve/surface that passes through our control points. How can we do that?

Interpolating Scattered Point Data

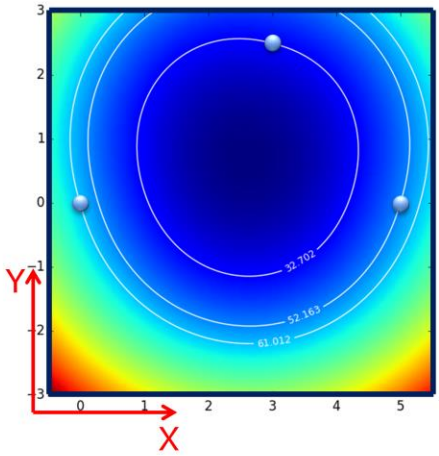
- Let $\Phi(\mathbf{p}) = \Phi(\mathbf{p}) - 40.23594781$
- 40.23594781 is not a magic number:
  - 40.23594781 = $\Phi_1(0,0)+ \Phi_2(0,0) = \Phi_1(5,0)+ \Phi_2(5,0)$
- Kernel now interpolates the two points at isocurve value = 0!
- *Idea here is that we can constrain the field to force zero isocurve through our points*



A constant offset to the total field value only works if all the points are on the same isocurve/surface. **This is only true when the kernels are spaced uniformly on a circle or sphere such that the field is symmetric!** Break this symmetry, and a constant offset no longer works, in general, to shift the value of the curve/surface passing through the points. We need a generalized method to adjust the field.

Interpolating Scattered Point Data

- Three kernels at control points (0,0), (5,0), and (3,2.5)
- *Points are all on different isocurves*
- A constant shift no longer puts them on the same curve!
- How can we adjust?



These are biharmonic RBF kernels

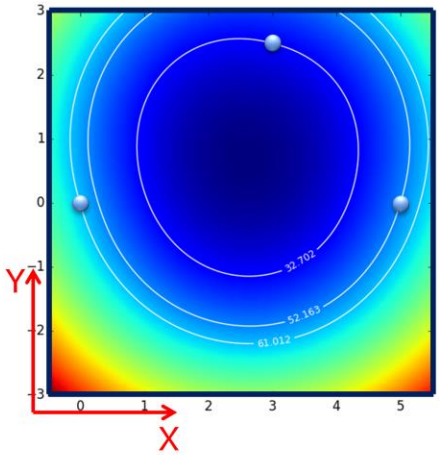
Interpolating Scattered Point Data

- We can write the local influence of a kernel at control point j on location i as:

$$\Phi_{ij} = \left|\mathbf{p}_i - \mathbf{A}_j\right|^2 \ln(\left|\mathbf{p}_i - \mathbf{A}_j\right|)$$

- And total field value at i as a *weighted sum* of n kernels at the control points:

$$\Phi(\mathbf{p}_i) = \lambda_1\Phi_{i1} + \lambda_2\Phi_{i2} + \cdots + \lambda_n\Phi_{in}$$

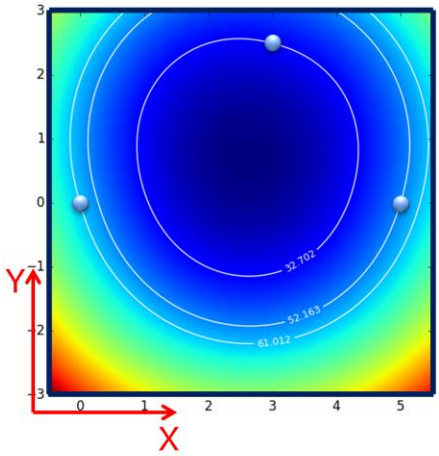


These are biharmonic RBF kernels

Interpolating Scattered Point Data

- Considering n control points that we wish to be on the same isocurve/surface with value f, we can write a system of n linear equations in matrix form:

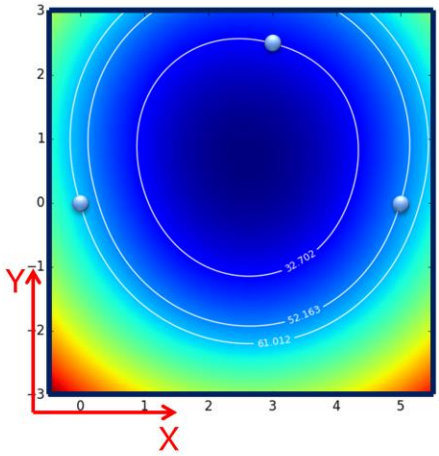
$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \cdots & \Phi_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \Phi_{n2} & \cdots & \Phi_{nn} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} f \\ f \\ \vdots \\ f \end{bmatrix}$$



These are biharmonic RBF kernels

Interpolating Scattered Point Data

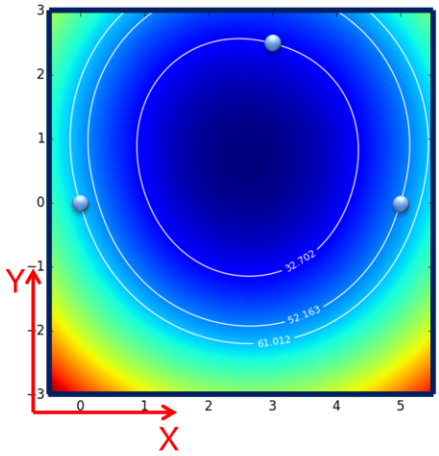
- Need to solve for the weights, λ_i
- Generally, we can solve a system of linear equations via:
  - Direct solvers such as Cholesky or SVD decomposition
  - Iterative solvers



These are biharmonic RBF kernels

Interpolating Scattered Point Data

- This problem requires a careful selection of solver
  - Diagonal is zero (for biharmonic)
  - Matrix is not positive definite
  - Condition number gets larger as kernels are added
- LU decomposition techniques are not ideal choices here
- SVD is a reasonable starting point, especially for small # of control points, but too slow and large memory requirements for large problems (thousands of kernel points)

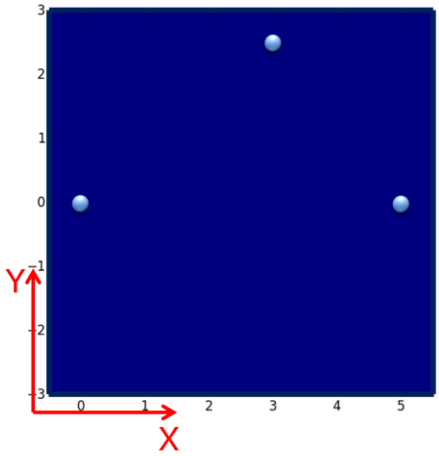


See Numpy, an open source Python add on library, for an implementation of an SVD solver.

Interpolating Scattered Point Data

- Back to the solution
- We want the zero isocurve, so set f = 0

$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \cdots & \Phi_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \Phi_{n2} & \cdots & \Phi_{nn} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



- Solver will find trivial solution:

$$\lambda_1 = \lambda_2 = \cdots = \lambda_n = 0$$

- Field is zero everywhere. That's no good.

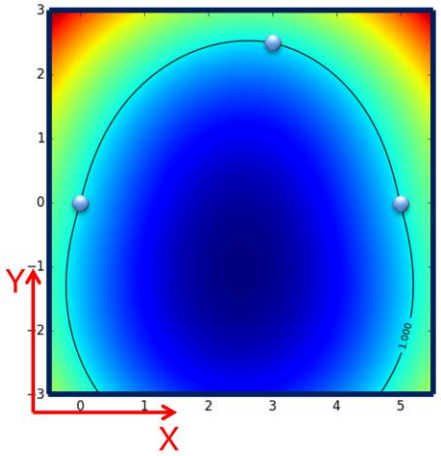
These are biharmonic RBF kernels

Interpolating Scattered Point Data

- Back to the solution
- What about f = 1

$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \cdots & \Phi_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \Phi_{n2} & \cdots & \Phi_{nn} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

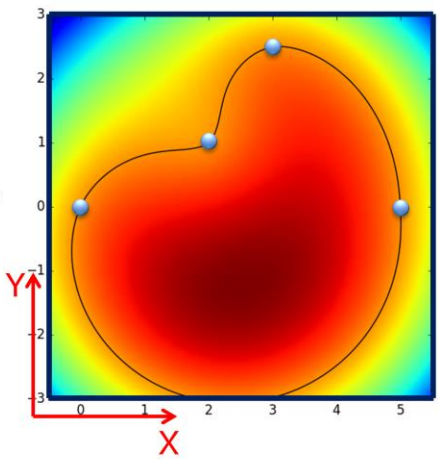
- Solver may find nontrivial values of λ_i
- But we wanted f = 0
- And may find difficult to control curvature if points are close together



These are biharmonic RBF kernels

Interpolating Scattered Point Data

- What if we add a surface point at (2,1)?
- Isocurve 0 nicely interpolates the new point
- BUT, notice that the bottom part of the curve moves significantly
  - The added point is modifying a part of the curve that we don't desire to be controlling
  - This is an example of curvature that we wish to keep under control



These are biharmonic RBF kernels

Interpolating Scattered Point Data

- Minimize curvature of solution by adding a planar offset, **F**, to the total field value…

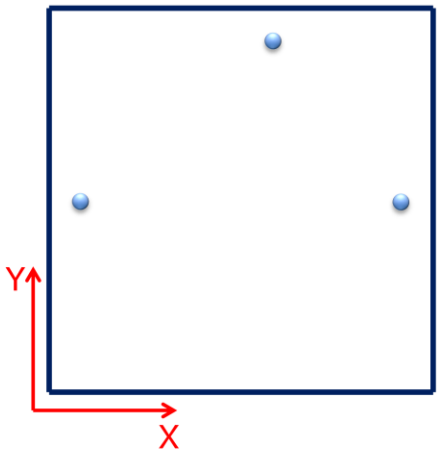
$$\Phi(\mathbf{p}_i) = \mathbf{F}(\mathbf{p}_i) + \sum_{j=1}^{n} \lambda_j \Phi_{ij}$$

where

$$\mathbf{F}(\mathbf{p}_i) = \boldsymbol{a} + \boldsymbol{b}\mathbf{p}_{i,x} + c\mathbf{p}_{i,y}$$

- …and requiring that

$$\sum_{j=1}^{n} \lambda_j = \sum_{j=1}^{n} \lambda_j \, \mathbf{p}_{j,x} = \sum_{j=1}^{n} \lambda_j \, \mathbf{p}_{j,y} = 0$$



These are biharmonic RBF kernels

Interpolating Scattered Point Data

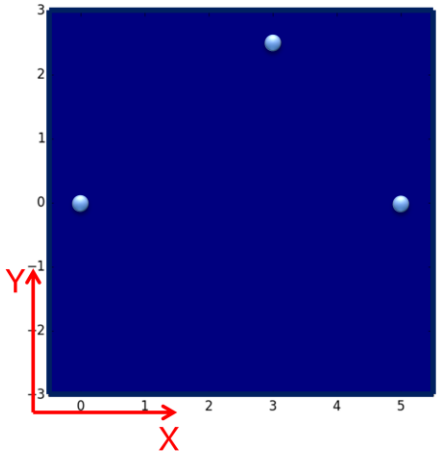
- System becomes the following, and solution includes λ_i, a, b, and c (and d for 3D)

$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \cdots & \Phi_{1n} & 1 & \mathbf{p}_{1,x} & \mathbf{p}_{1,y} \\ \Phi_{21} & \Phi_{22} & \cdots & \Phi_{2n} & 1 & \mathbf{p}_{2,x} & \mathbf{p}_{2,y} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \Phi_{n2} & \cdots & \Phi_{nn} & 1 & \mathbf{p}_{n,x} & \mathbf{p}_{n,y} \\ 1 & 1 & \cdots & 1 & 0 & 0 & 0 \\ \mathbf{p}_{1,x} & \mathbf{p}_{2,x} & \cdots & \mathbf{p}_{n,x} & 0 & 0 & 0 \\ \mathbf{p}_{1,y} & \mathbf{p}_{2,y} & \cdots & \mathbf{p}_{n,y} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ a \\ b \\ c \end{bmatrix} = \begin{bmatrix} f \\ f \\ \vdots \\ f \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

These are biharmonic RBF kernels

Interpolating Scattered Point Data

- Let's solve with f = 1
- Field = 0 again
- What happened?
  - Another trivial solution
  - a = 1, b = c = 0
  - λ_i = 0
- We can fix this by adding external control points, e.g., additional RBF kernels that are guaranteed to be in the region that is exterior to our curve

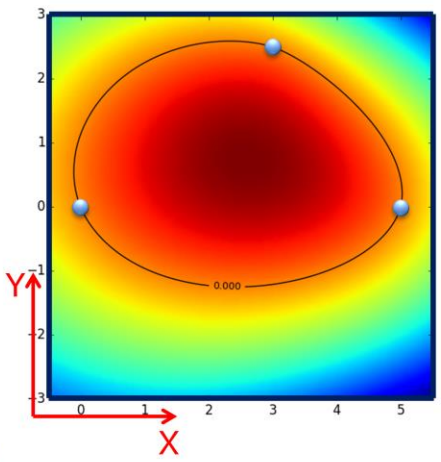


These are biharmonic RBF kernels

Interpolating Scattered Point Data

- Now we have the original 3 points with f = 0, which lie on the desired isocurve
- Plus 4 external points that lie somewhere known to be in the exterior area away from the isocurve
  - For example figure, external points* are at (-5,-5), (5,-5), (5,5), and (-5,5)
  - Those are somewhat arbitrary
  - Solve for f = 1 at these external points
- Now the solution looks better
- Isocurve is more tightly bound to the surface points

*see green points (●) near edges of page

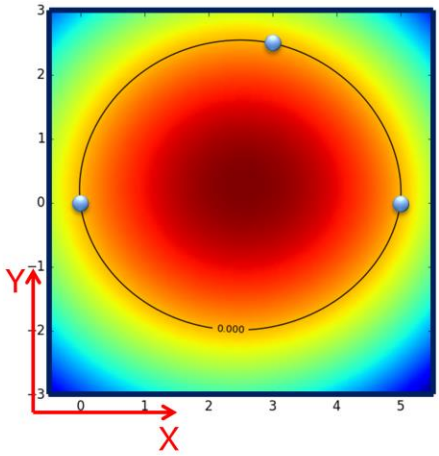


By adding the planar offset, which minimizes the curvature, we get a more reasonable curve/surface. This is a better choice for modeling implicit surfaces with RBF's

Do not have to have four external points. Need at least one. May get better results if you have more than one, and if they are uniformly distributed on a circle (2D) or sphere (3D).

Interpolating Scattered Point Data

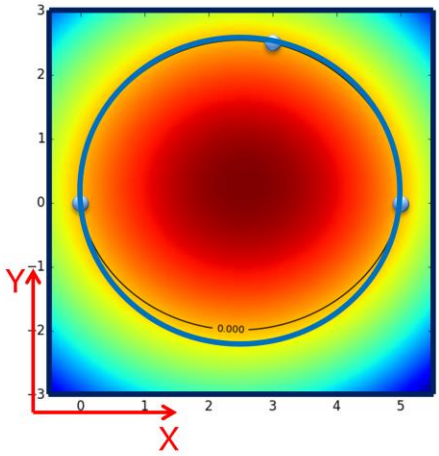
- What happens when we move our external points to (-25,-25), (25,-25), (25,25), and (-25,25)?



By adding the planar offset, which minimizes the curvature, we get a more reasonable curve/surface. This is a better choice for modeling implicit surfaces with RBF's

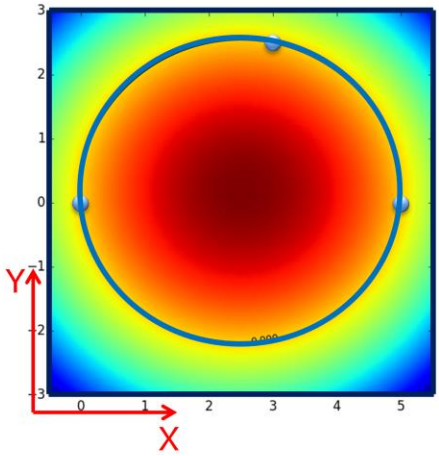
Interpolating Scattered Point Data

- What happens when we move our external points to (-25,-25), (25,-25), (25,25), and (-25,25)?
- Isocurve moves towards a circle that interpolates the 3 kernel points



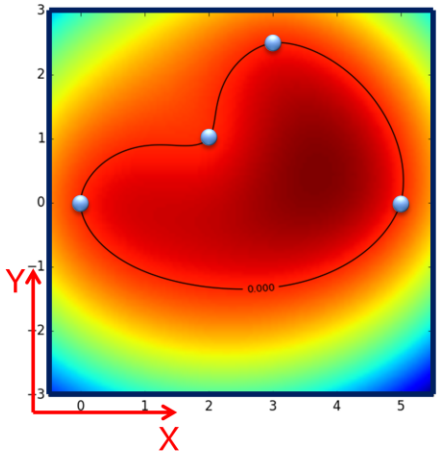
Interpolating Scattered Point Data

- What happens when we move our external points to (-250,-250), (250,-250), (250,250), and (-250,250)?
- Closer convergence to a circle



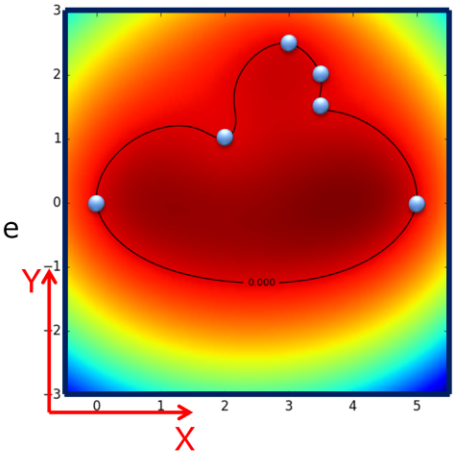
Interpolating Scattered Point Data

- What happens if we add a fourth surface point at (2,1)?
- Isocurve nicely interpolates the new point and the original 3
- Notice that with the planar offset, the added point has a strong local effect, and does not change the bottom of the curve as much as the solution without the planar offset.



Interpolating Scattered Point Data

- And if we add (3.5,1.5) and (3.5,2)?
- Isocurve nicely interpolates the new points
- Again, the change to the curve is fairly well localized
- Note that by careful placement of points, it is possible to locally control the surface normal, but this is a tedious operation



Extension to 3D

- Extension to 3D is natural, and methodology identical
- Consider using the triharmonic RBF instead of the biharmonic:

$$\Phi(\mathbf{p}) = |\mathbf{p} - \mathbf{A}|^3$$

- The planar offset adds a z axis term, which results in a fourth constraint equation:

$$\mathbf{F}(\mathbf{p}_i) = \boldsymbol{a} + \boldsymbol{b}\mathbf{p}_{i,x} + c\mathbf{p}_{i,y} + d\mathbf{p}_{i,z}$$

Takeaways

- Radial basis functions (RBF's) and other kernel functions can be used to interpolate/extrapolate volumetric scalar field functions given scattered control points
- Choice of RBF depends on application, and is a function of compact vs. non-compact support
- Especially for RBF's with non-compact support, a solver that is tolerant of non-positive definite matrices is required to solve for the RBF weight factors

Takeaways

- Example applications of RBF-based volumetric field and implicit surface modeling in games
  - Light field modeling for volumetric lighting
  - Generate polygon mesh for fluid rendering
  - Use gradient of scalar field to generate animation or navigation paths

References

- Witkin, Andrew P., and Paul S. Heckbert, "Using Particles to Sample and Control Implicit Surfaces," SIGGRAPH 1994
- Turk, Greg, and James F. O'Brien, "Modeling with Implicit Surfaces that Interpolate," ACM Transactions in Graphics, Volume 21, Number 4, October 2002
- Funkhouser, Thomas, "Implicit Surfaces," Princeton University, Fall 1999 (available on the Internet as PDF)

Contact

- Graham Rhodes
  - grhodes@nc.rr.com

Appendix:

An Incomplete/Failed Experiment in Modeling Curves/Surfaces with Simple Potential Flow Kernels

Modeling Implicit Surfaces using Fluid Flow

- When I presented this talk at GDC 2015, I spent some time attempting to explain an unfinished experiment
- Initial results of the experiment were unfortunately not promising
- Idea was to choose a basis function from classical potential field techniques that are historically used to simulate incompressible, irrotational fluid flow, e.g., in the preliminary design of wings, airplanes, etc.
- What follows is the original GDC slides plus one additional introductory slide to explain my motivation

Modeling Implicit Surfaces using Fluid Flow

- I was motivated to experimental with potential flow kernels in part because of my personal background in computational fluid dynamics and airplane design (years ago)
- I believed that the potential flow kernels might offer true benefits for geometric surface modeling, for the following reason:
  - Closed form equations exist for potential flow kernels based on linear and polygonal control geometry
  - Such edge and surface-based kernels may provide better control over local surface properties
  - Due to the basis in physics-based governing equations, gradient curves extracted from potential flow fields might be more pleasing than gradient curves extracted from other scalar fields

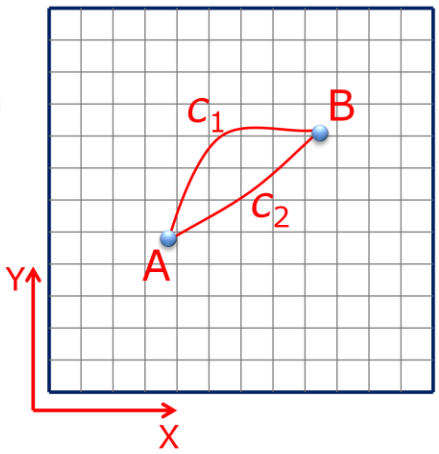
Modeling Implicit Surfaces using Fluid Flow

- I still believe that some interesting results can be obtained by using the linear and polygonal 2D and 3D kernels for potential fluid flow
- However, now I am convinced there would be no realized benefit since the cost of the linear and polygonal kernels is computationally too expensive, at least for evaluation at runtime in a game

Modeling Implicit Flow Surfaces

- "Potential" field
  - Green's Theorem
    - Intuitively, the sum of change over a volume is equal to the flux through the boundary surface
  - Analogous to the Fundamental Theorem of Calculus
- Laplace's Equation:

$$\nabla^2 \Phi = 0$$



1 min for intro to potential fields

Mention conservation laws

Laplace's equation as representation of conservation of mass of a fluid

Yes, we're getting into physics a bit, but it enables some pretty cool and useful implicit surface and curves

Why Potential Functions?

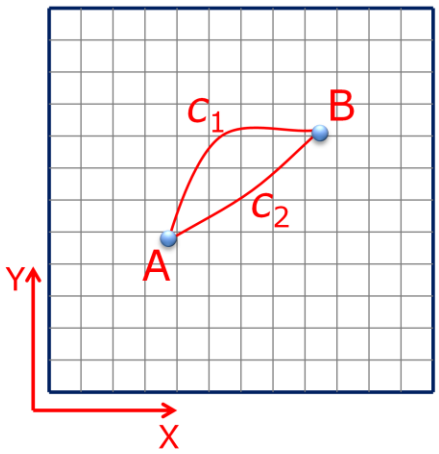
- Linear PDE:

$$\nabla = \boldsymbol{i}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{\partial}{\partial y} + \boldsymbol{k}\frac{\partial}{\partial z}$$

$$\nabla^2 = \nabla \cdot \nabla$$

$$\nabla^2 = \frac{\partial}{\partial x}\frac{\partial}{\partial x} + \frac{\partial}{\partial y}\frac{\partial}{\partial y} + \frac{\partial}{\partial z}\frac{\partial}{\partial z}$$

$$\nabla^2\Phi = \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\partial^2\Phi}{\partial z^2} = 0$$



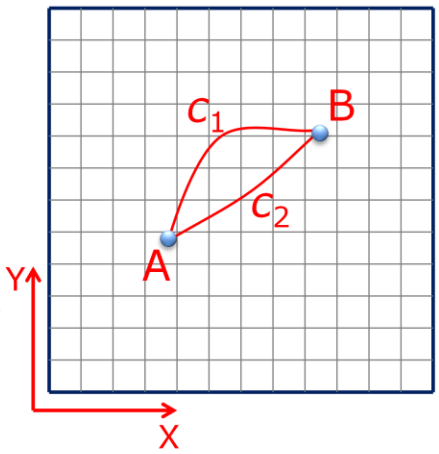
1 min for intro to potential fields

Mention conservation laws

Laplace's equation as representation of conservation of mass of a fluid

Why Potential Functions?

- Linear PDE:
  - Principle of superposition applies
  - If f and g are both solutions, then f+g is also a solution
- A number of basis functions automatically satisfy Laplace's Equation
- Laplace's Equation represents the continuity equation (conservation of mass), one of the governing conservation law equations of fluid flow

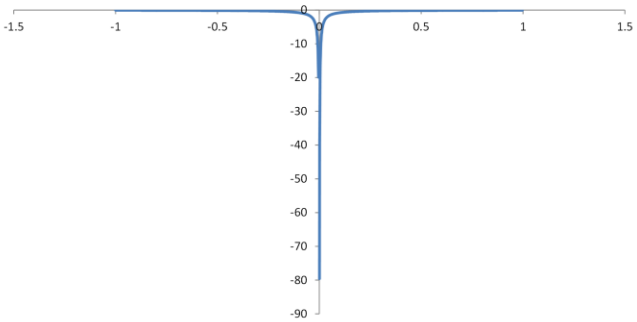


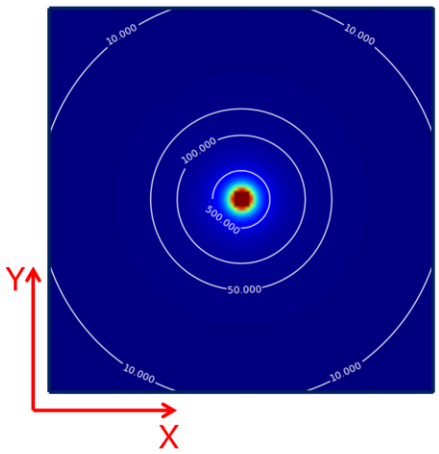
Principle of superposition allows us to generate a desirable field by introducing a set of primitive solutions that control the field in a meaningful way.

3D Point Source Basis Function

- Purpose: Attract/repel

$$\Phi = \frac{-\sigma}{4\pi r}$$





Don't freak out about inverse square root or discontinuity when r == 0!

We can evaluate anywhere. Doesn't have to be on a grid.

Infinite support, but at the location of the kernel….influence decays to zero at infinity. So this is not good for fitting surfaces to scattered fields.

(Think about whether or not to introduce the negative sign here…)

Notice that this is an RBF

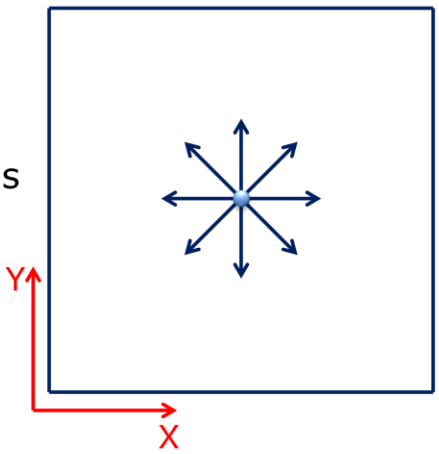
Relate to a Flow

- Velocity Potential

$$\vec{v} = \nabla\Phi$$

- Derivative of Φ in any direction is the velocity projection in the direction
- For 3D point source:

$$\vec{v} = \frac{\sigma}{4\pi r^2}\hat{r}$$



2D Constant Source

- Potential for single point

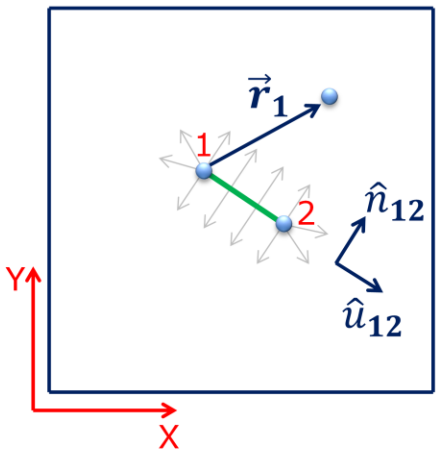
$$\Phi = \frac{\sigma}{2\pi}\ln(|\vec{r}|)$$

- Potential for a line segment

$$\Phi = \frac{\sigma}{4\pi}[(\vec{r}_1 \cdot \hat{u}_{12})\ln(|\vec{r}_1|) - (\vec{r}_2 \cdot \hat{u}_{12})\ln(|\vec{r}_2|) + 2z(\theta_2 - \theta_1)]$$

$$z = \vec{r}_1 - \hat{u}_{12}(\vec{r}_1 \cdot \hat{u}_{12})$$

$$\theta_i = \mathbf{arctan2}(z, \vec{r}_i \cdot \hat{u}_{12})$$



If we look at a 2D source, however…story is different. Infinite support! So we can use this to model scattered point clouds.

Ask me afterwards if you want to understand why this is ln(r) instead of 1/r!

There also exists a closed form equation for the source distributed over a polygon in 3D

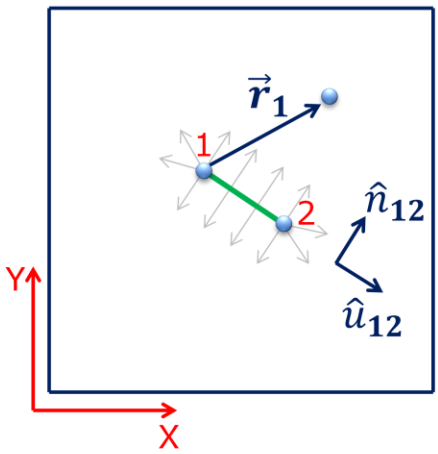
Notice that this resembles the biharmonic RBF. The single point version is an RBF, but the line segment version is not. There also exists a closed form kernel function for a polygonal potential source in 3D. It is quite ugly and computationally expensive.

2D Constant Source

- Velocity components

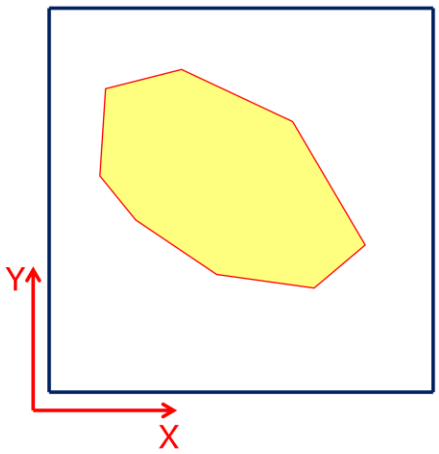
$$\vec{v} = \nabla\Phi$$

$$\vec{v} = \hat{u}_{12}\frac{\sigma}{4\pi}\ln\left(\frac{|\vec{r}_1|^2}{|\vec{r}_2|^2}\right) + \hat{n}_{12}\frac{\sigma}{2\pi}(\theta_2 - \theta_1)$$



Hand-crafted Mesh as Support

- Idea
  - Compute flow field based on source segments such that the total potential is constant on the boundary
  - Then, the model surfaces will be an isosurface in the potential field
  - If we use linear or polygonal sources, may be able to model surfaces that are perfectly flat and/or that contain slope discontinuities



We can use the 2D source (point or constant segment) to model point clouds, but also to model flow relative to closed surfaces.

We can use a constant distributed source to generate a smooth potential off of an area surface. We could generate an offset surface using point sources, but this would not be smooth.

Need to fill in details of boundary conditions (Dirichlet vs. Neumann), formulation of system, point to references to solve linear system.

Hand-crafted Mesh as Support

- Similar to the method used for traditional RBF-based implicit surface modeling, build system of equations

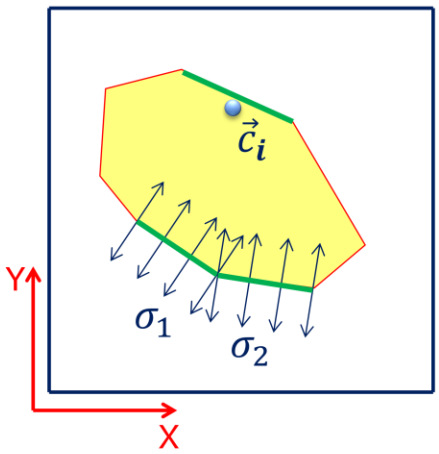
$$\Phi_1^* = \sigma_1 a_{11} + \sigma_2 a_{12} + \sigma_3 a_{13} + \cdots$$

$$\Phi_2^* = \sigma_1 a_{21} + \sigma_2 a_{22} + \sigma_3 a_{23} + \cdots$$

… where

Φ_2^* = total "internal" potential at panel i

$a_{ij} = \Phi_j(\vec{c}_i)$ using $\sigma_j = 1$



We can use a constant distributed source to generate a smooth potential off of an area surface. We could generate an offset surface using point sources, but this would not be smooth.

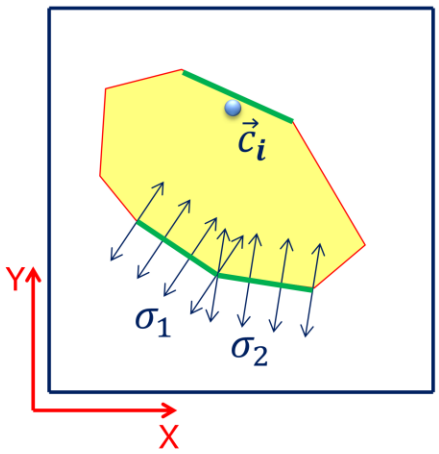
Need to fill in details of boundary conditions (Dirichlet vs. Neumann), formulation of system, point to references to solve linear system.

Hand-crafted Mesh as Support

- Dirichlet boundary condition

$$\begin{bmatrix} a_{11}a_{12}a_{13} \\ a_{21}a_{22}a_{23}\ \dots \\ a_{31}a_{32}a_{33} \\ \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Phi_1}^* \\ \boldsymbol{\Phi_2}^* \\ \vdots \\ \boldsymbol{\Phi_n}^* \end{bmatrix}$$

- Can't set RHS == 0
- Let $\boldsymbol{\Phi_1}^* = \boldsymbol{\Phi_2}^* = \cdots = \boldsymbol{\Phi_n}^* = \mathbf{1}$
- Solve for source strengths, σ_i



We can use a constant distributed source to generate a smooth potential off of an area surface. We could generate an offset surface using point sources, but this would not be smooth.

Dirichlet boundary conditions refer to the direct specification of the total potential. Note that the traditional RBF-based interpolated field technique, shown in the earlier slides, also specifies the total field value for each control point. This too is an example of Dirichlet boundary conditions.

Another type of boundary condition, the Neumann boundary condition, specifies the value of the derivative in the desired gradient direction instead. This is certainly used extensively for potential flow modeling and could be some benefit to doing this for implicit surface modeling. The velocity equations given on these pages provide enough information to formulate a system of equations based on Neumann boundary conditions.

Evaluating the surface

- Once the source strengths are known
  - Entire field can be evaluated at all points in space
  - Evaluate directly, as needed, or sample into a grid for polygonalization

Path tracing through potential field

- Evaluate local velocity directly using source strengths

$$\vec{v} = \nabla\Phi = \boldsymbol{i}\frac{\partial\Phi}{\partial x} + \boldsymbol{j}\frac{\partial\Phi}{\partial y} + \boldsymbol{k}\frac{\partial\Phi}{\partial z}$$

$$\vec{v}(\vec{p}) = \nabla\Phi^*(\vec{p}) = \sum_{i=1}^{n} \nabla\Phi_i(\vec{p})$$

Superimpose other control kernels

- Crowd path finding
  - Add a free stream to generate a field for entities moving in a general direction
  - Add point source/sink for entities with specific targeting goals

It is a slight hack to superimpose additional flow kernels without re-solving the equations for element strengths. The result will at best be approximate. Boundary conditions will be violated. But for visual effects this may be acceptable.

Attention to speed

- Accelerating computation of potential
  - Delegate to GPU
  - Use far field solution
  - Use approximate functions
  - Take advantage of similarity
    - Bake function into texture
    - Evaluate by texture lookup + transform

Far field potential is simply the potential far from the kernels. For all potential flow kernels based on linear and polygonal control geometry, the far field is asymptotic with an equivalent point element.

Additional Takeaways

- The potential flow basis functions that are based on linear and polygonal control geometry may be beneficial for modeling implicit surfaces with difficult features such as perfectly flat regions and surface discontinuities
- The cost of computing these basis functions makes them unlikely candidates for use at runtime in a game; however, they may be useful in digital content creation tools
- The curves (e.g., streamlines, pathlines) extracted using the velocity potential are consistent with physically based incompressible, irrotational fluid flows.
  - The physical nature of these curves is useful for engineering applications, but may not be particularly beneficial for games
  - The technique of using the gradient of the potential field to extract curves is not unique to the potential flow kernels. The gradient technique can be used with any scalar field. Any RBF-based field could produce interesting results.

Additional References

- Katz, Joseph, and Allen Plotkin, "Low-speed Aerodynamics: From Wing Theory to Panel Methods"