UNIVERSITY OF CAMBRIDGE

Level set methods

Stephen Millmore

Laboratory for Scientific Computing

Outline

- Implicit boundary methods
- Numerical methods for level sets
- Defining level set functions
- Reinitialisation

Representing a sharp interface

- Within this lecture we consider the underlying mathematical and numerical techniques required for **sharp interface methods**
- To achieve this, we need a technique which can identify the **location of a material interface,** and use this to split the computational domain into two or more regions
- This description suggests a distinction between the determining the location of the interface and the application of the multiphysics behaviour between the distinct regions
- This distinction does exist - here we cover the theory behind the representation of the interface
- However, this theory can extend to other applications, beyond sharp interface multiphysics methods

Outline

- Implicit boundary methods
- Numerical methods for level sets
- Defining level set functions
- Reinitialisation

How is a boundary represented?

- If we wish to use a sharp interface multiphysics method, then we need a technique which both describes and evolves the location of an interface
- This may seem straightforward - we simply map a function to the interfaces which exist between materials
- Though will this work with finite volume (or finite difference) approximations?
- Perhaps unsurprisingly, there are multiple approaches which can be used to deal with material interfaces

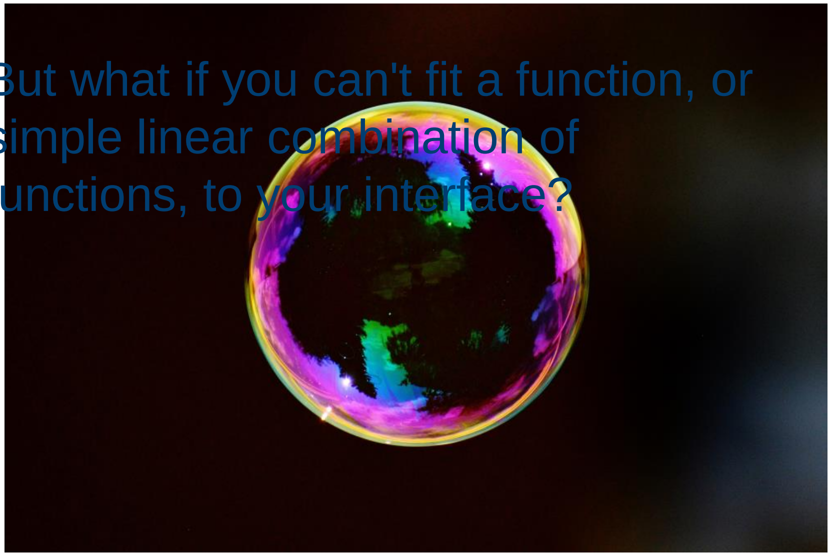
Explicit and implicit boundaries

- For numerical methods involving material interfaces, there are two classes of method – **explicit** or **implicit** boundary representation
- **Explicit boundaries:**
  - You always know exactly where your boundary is
  - In some ways, the edge of the computational domain is an explicit boundary, to which you must apply boundary conditions
- **Implicit boundaries:**
  - You don't know precisely where your boundary is, but you do know which side of the boundary you are on
  - In some ways, the discontinuities in Euler equation solutions are implicitly defined (ignoring the smearing)

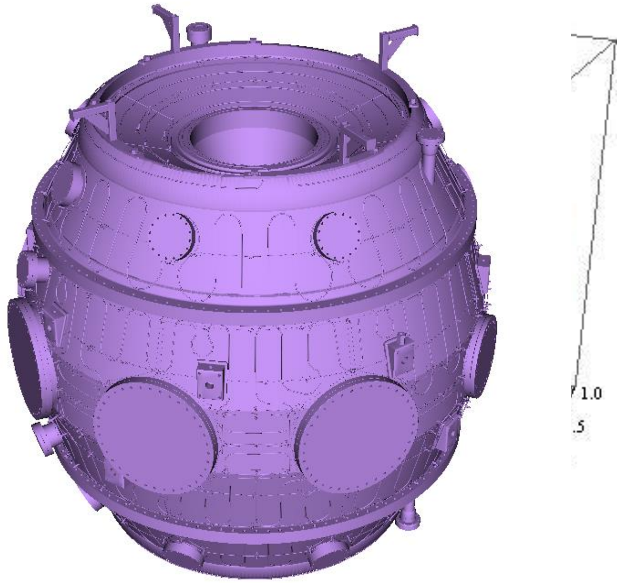
What is an explicit boundary?

- An explicit representation of an interface defines a surface, or curve, which exists only at the location of the interface
- For example, a bubble could have a boundary described by

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

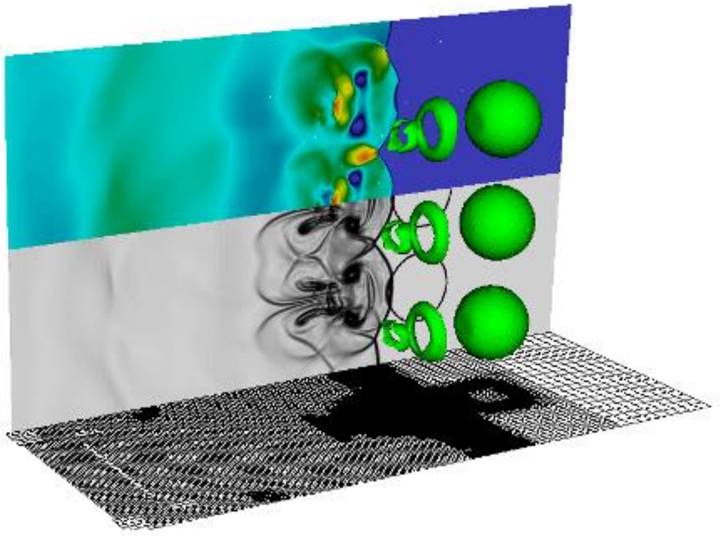


- But what if you can't fit a function, or simple linear combination of functions, to your interface?



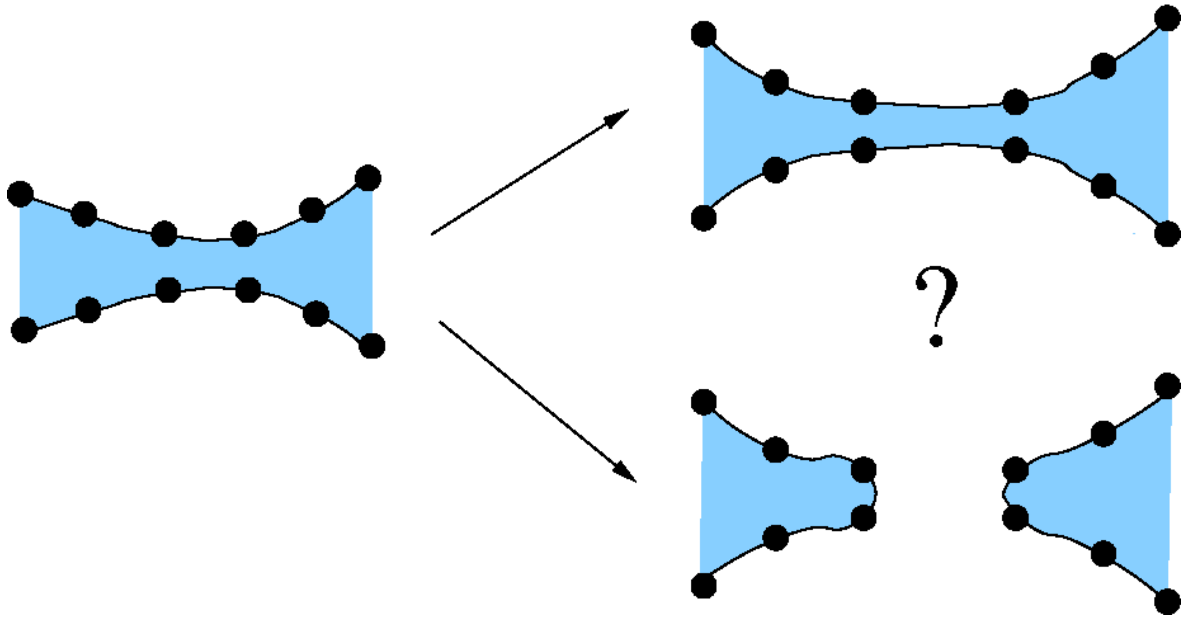
Challenges with explicit boundaries

- Even if it is difficult to map a function to a surface, that is not the main issue
- For multiphysics simulations, we need to be able to model how an interface moves
- This movement will be determined by the actual algorithm implemented for your model
- To understand these challenges, consider the interaction of a shock wave with one or more gas bubbles
- This interaction causes the bubble to break up; how do we ensure that our method knows when to do this?



Challenges with explicit boundaries

- It is difficult to accurately determine how this boundary representation should be ‘split’ in a physically realistic way
- **For an explicit boundary method, you would need a technique which could define this**

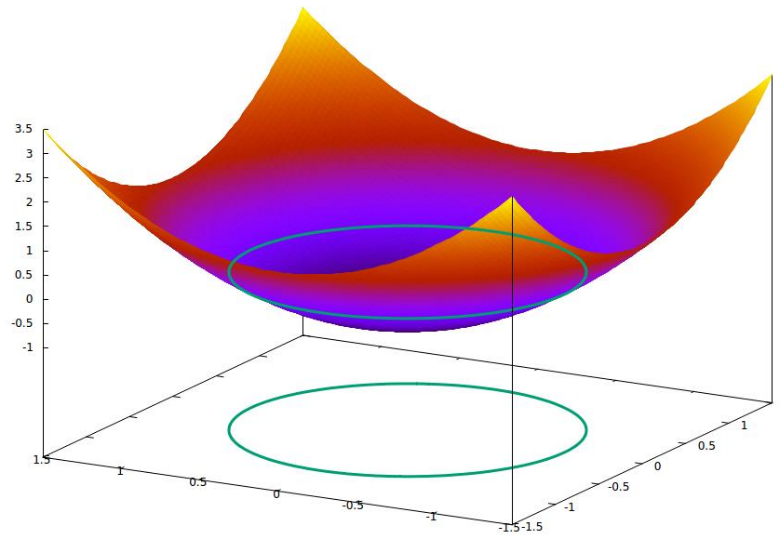


What is an implicit boundary

- Instead of treating the boundary as a surface, we can consider it the zero-contour of a scalar field
- Our previous example could now be defined through

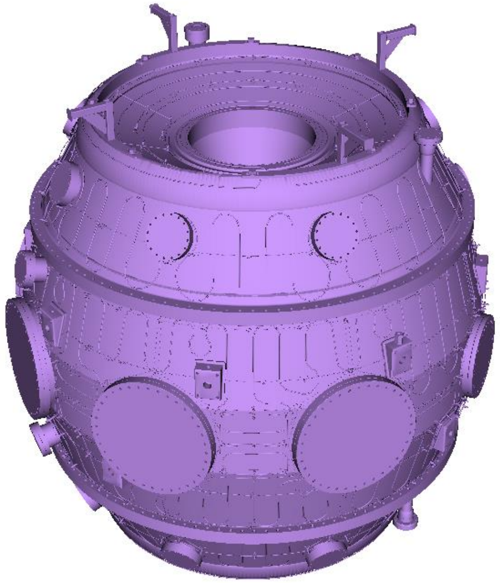
$$f(x,y,z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2$$

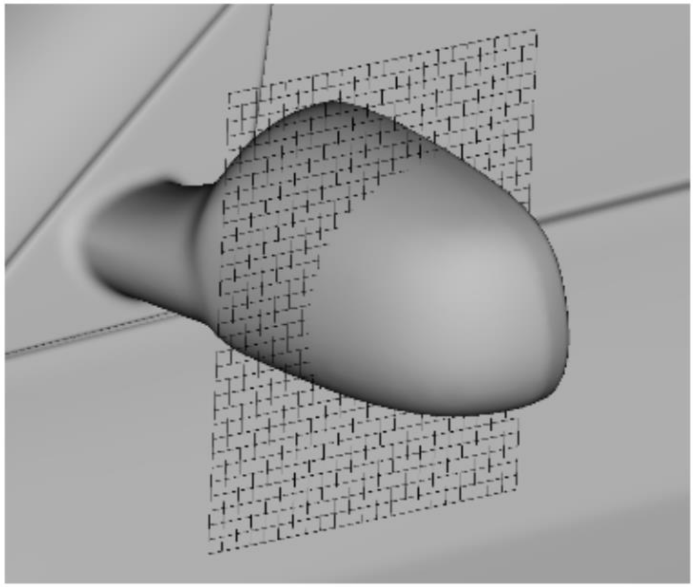
- Clearly this works well with our finite volume methods – we can provide this function at all cell-centred points
- We may never know where the interface is exactly, but we will know **which side of the interface our cell is on**



Challenges with implicit boundaries

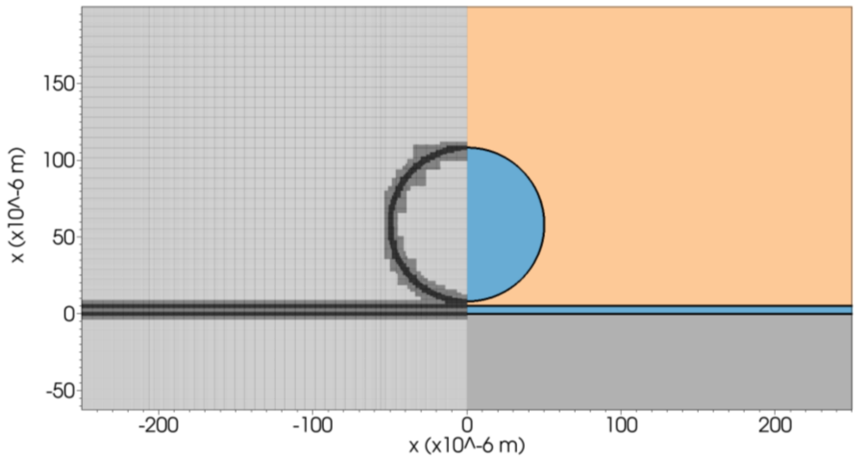
- Clearly the difficulties in determining the boundary of a complex object have not gone away
- In some ways, the key challenges have already been seen when cut cell methods were covered – how do you get information about where a boundary crosses a cell?

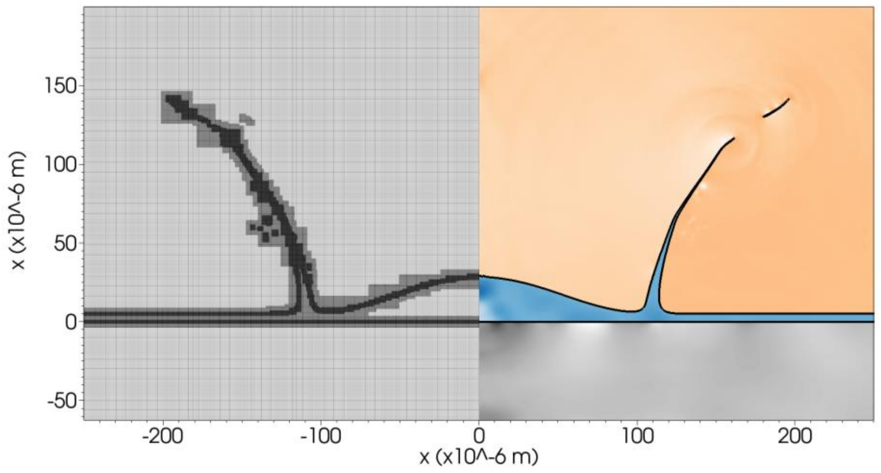




Advantages of an implicit boundary

- For many multiphysics methods, knowing what side of the interface you are is almost all you need (knowing how far away you are from it helps)
- The key, for our applications, is the fact that merging and splitting happens "trivially"
- Effectively, the finite volume evolution of an interface can cause it to merge or to split and we avoid having to worry about sub-cell scale behaviour





Level set methods

- We will now assume that we have an implicit representation of an interface at for some initial data for our simulation
- **Level set methods** are a common technique to track this interface as it evolves in time
- Although we are going to look at a specific application to multiphysics, level set methods themselves are a more general technique
- They can be used to model:

Level set methods

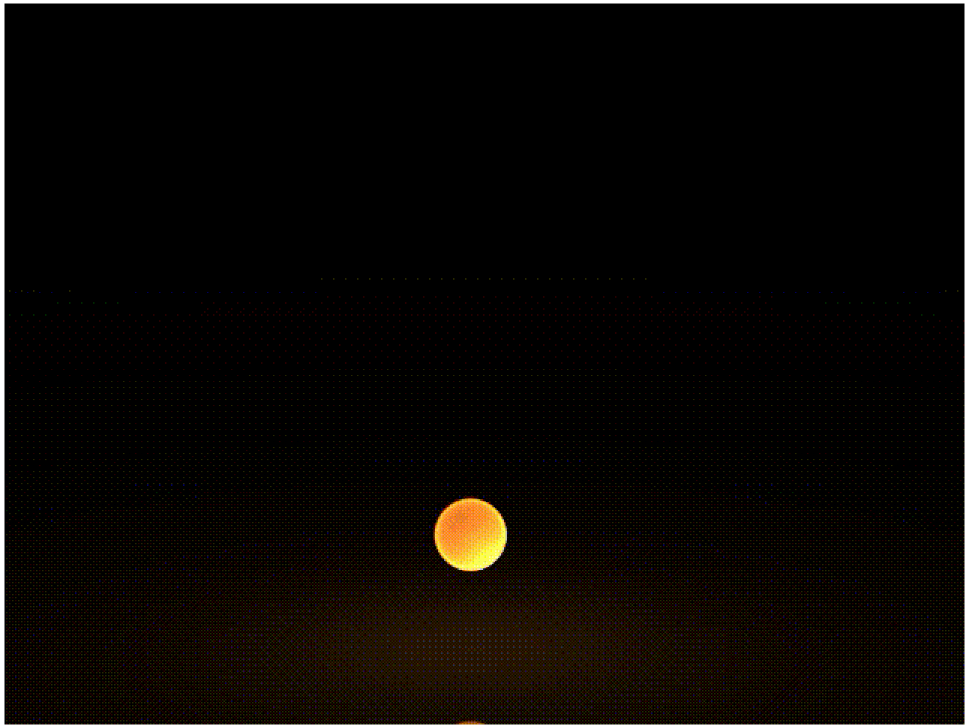
- They can be used to model:

1. Fluid dynamics (free surface flow)

Level set methods

- They can be used to model:

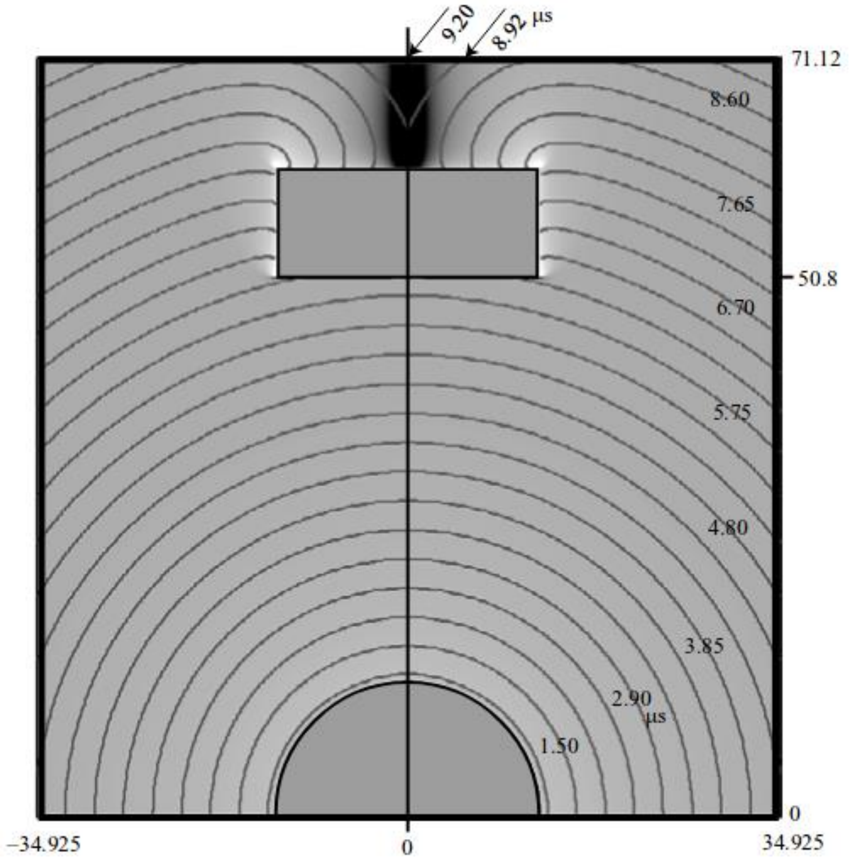
2. Flame fronts



Level set methods

- They can be used to model:

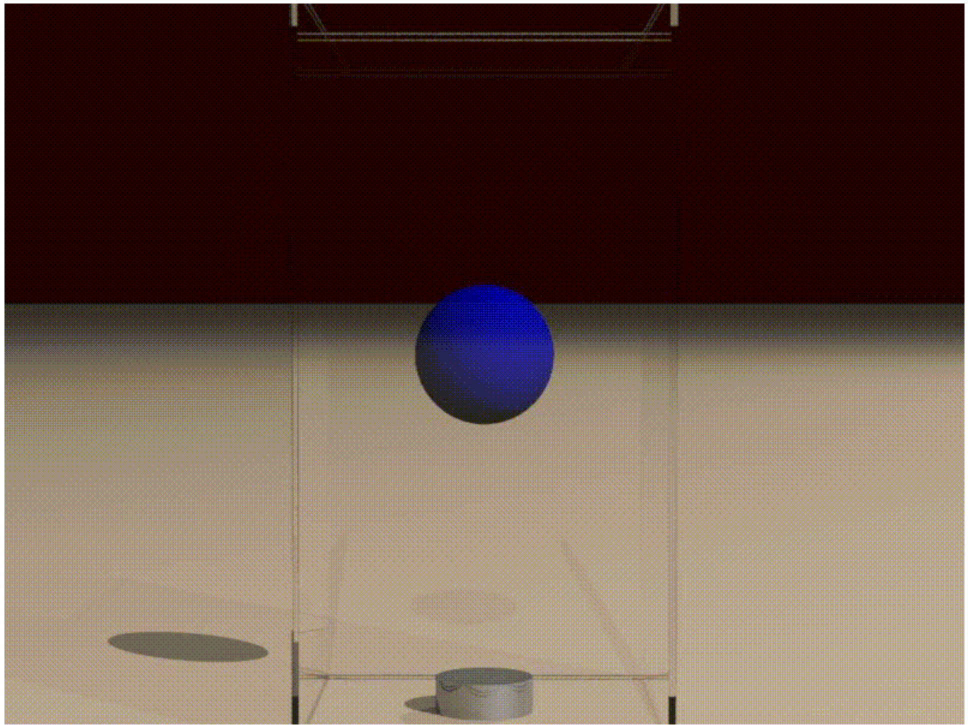
3. Detonation propagation



Level set methods

- They can be used to model:

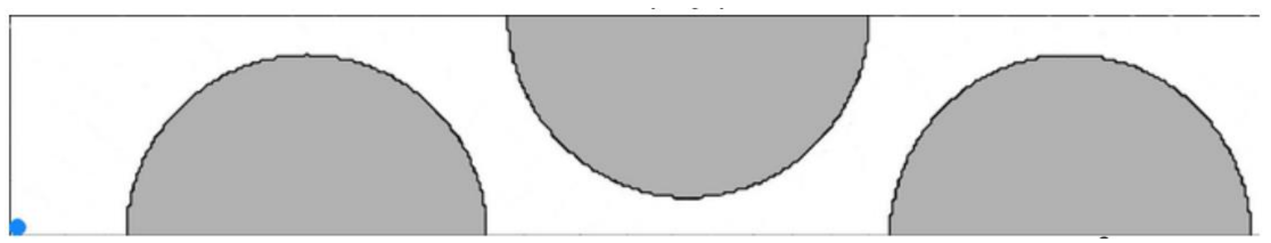
4. Density gradients in incompressible flow

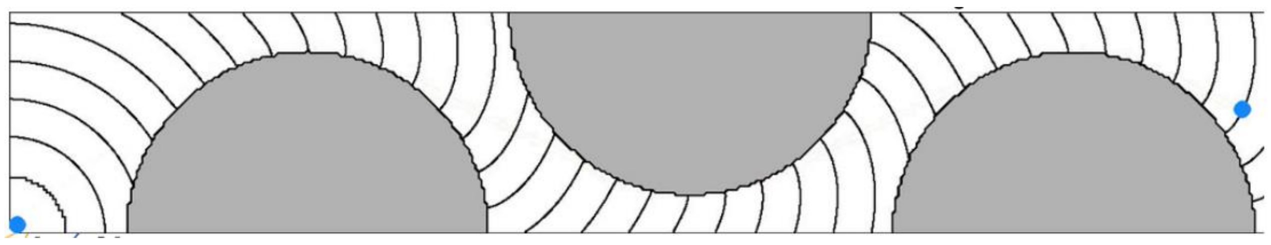


Level set methods

- They can be used to model:

5. Optimal path planning

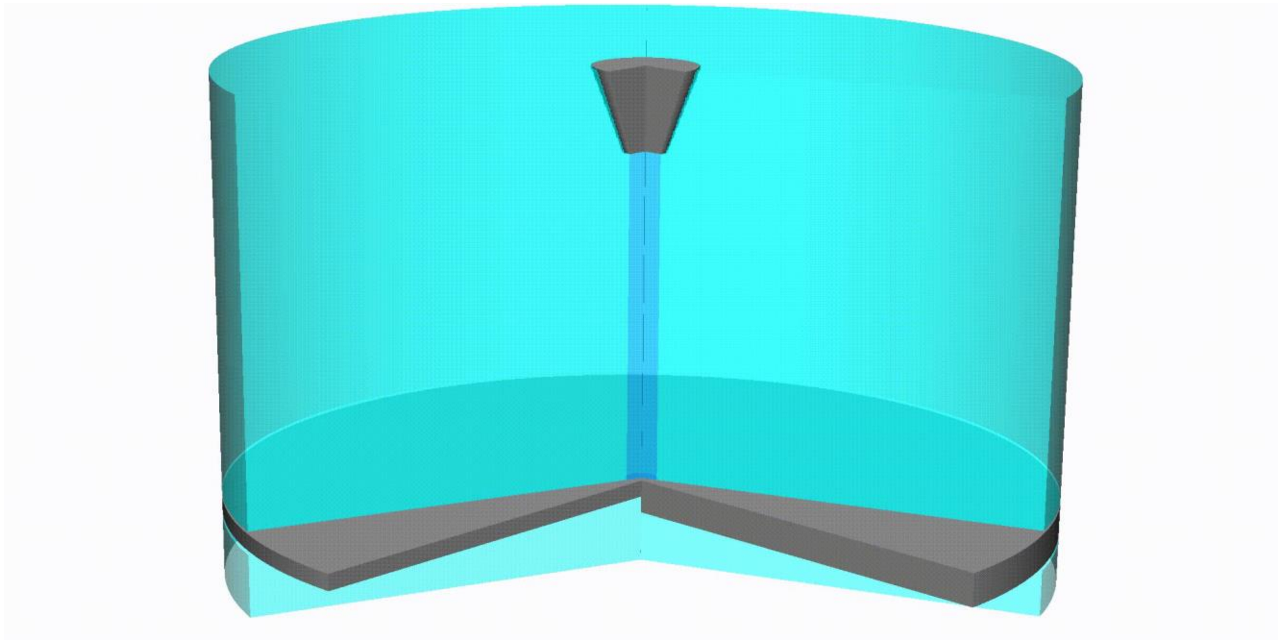




Level set methods

- They can be used to model:

6. Multiphysics (of course)

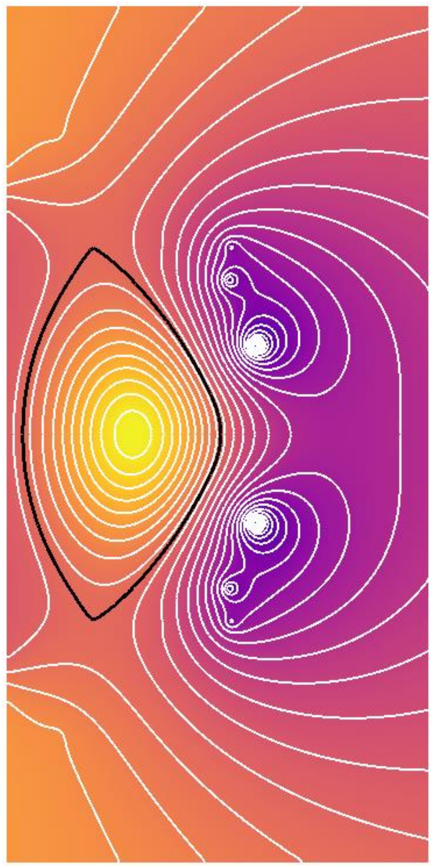


Outline

- Implicit boundary methods
- Numerical methods for level sets
- Defining level set functions
- Reinitialisation

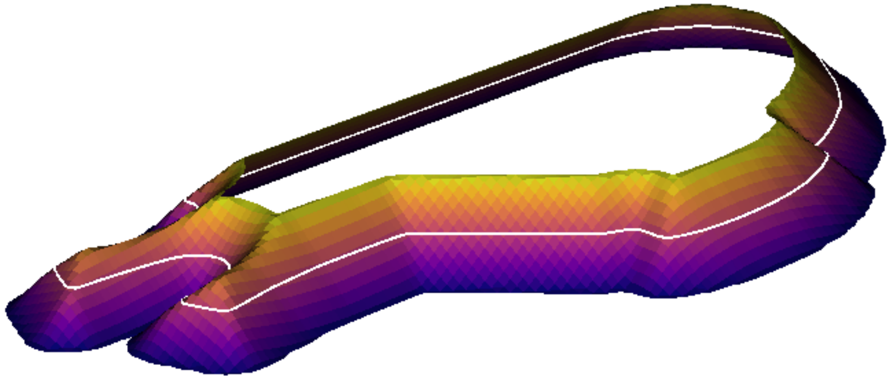
Level sets for modelling interfaces

- Level set methods are our method of choice for modelling sharp interfaces
- In many ways, **level set** is just a mathematical name for contour
- The level set is the set of points which all take the same value (i.e. are level)
- This is then referred to as the **level set function**, typically labelled ϕ
- It is often convenient to use the **zero-contour** of a function as the level set function, but not strictly necessary



Level sets for modelling interfaces

- Describing an interface is 'simply' a matter of choosing the correct function, ϕ , for which $\phi(\mathbf{x}) = 0$
- However, this is not all there is to a level set method - many of the examples shown require that the interface moves
- Therefore some form of evolution equation is needed for ϕ , and this must be formulated such that it describes the **physics of the interface**
- This does then depend on what you are using your level set method for

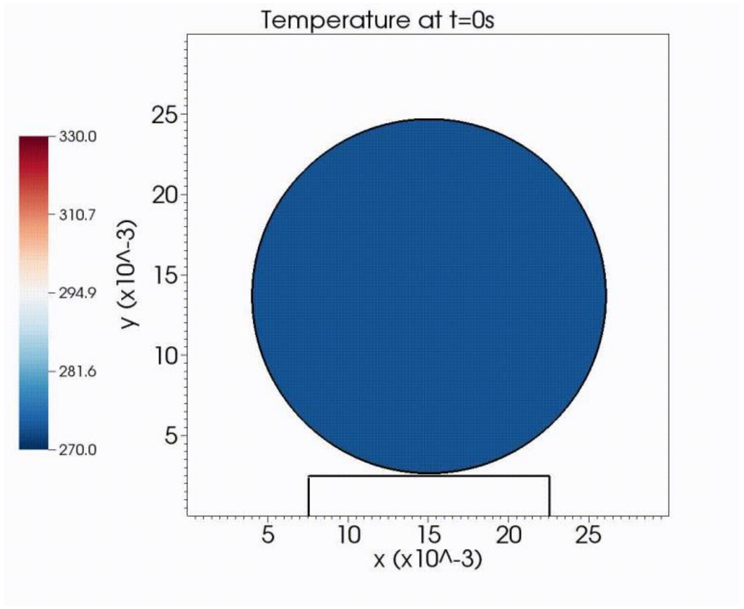


The level set equation

- **Level set equation** is a general term for the evolution equation that is being solved for a given level set function
- This must incorporate the physics of the system being modelled
- Commonly encountered level set equations are:

Advection:

$$\frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi = 0$$

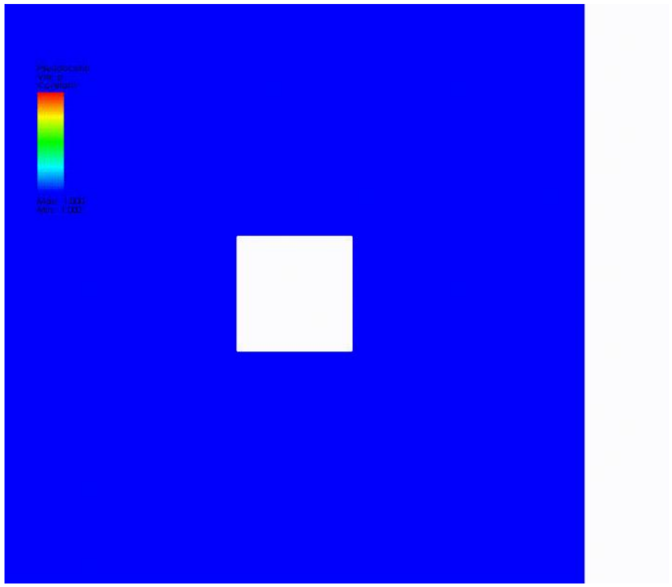


The level set equation

- **Level set equation** is a general term for the evolution equation that is being solved for a given level set function
- This must incorporate the physics of the system being modelled
- Commonly encountered level set equations are:

Normal motion:

$$\frac{\partial \phi}{\partial t} + V_n \left| \nabla \phi \right| = 0$$



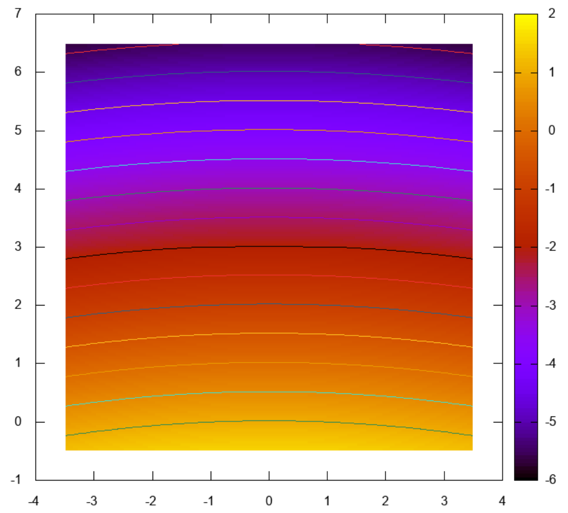
UNIVERSITY OF CAMBRIDGE

The level set equation

- **Level set equation** is a general term for the evolution equation that is being solved for a given level set function
- This must incorporate the physics of the system being modelled
- Commonly encountered level set equations are:

Curvature-driven motion:

$$\frac{\partial \phi}{\partial t} + \left[\nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|}\right)\right] |\nabla \phi| = 0$$



Hamilton-Jacobi equations

- Many level set equations are a type of **Hamilton-Jacobi** equation
- These equations have the form

$$\frac{\partial \phi}{\partial t} + H\left(\nabla \phi\right) = 0$$

- Here, the operator, H, acts upon the gradient (first derivatives) of the level set function

$$\frac{\partial \phi}{\partial t} + \boxed{\mathbf{v} \cdot \nabla \phi} = 0 \qquad \frac{\partial \phi}{\partial t} + \boxed{V_n \left|\nabla \phi\right|} = 0 \qquad \frac{\partial \phi}{\partial t} + \left[\nabla \cdot \left(\frac{\nabla \phi}{\left|\nabla \phi\right|}\right)\right] \left|\nabla \phi\right| = 0$$

$$H\left(\nabla \phi\right)$$

Second derivatives –
Not a HJ equation

- Note: Hamilton-Jacobi equations are not conservation laws

Relationship to conservation laws

- We have spent quite a lot of time making sure we solve conservation laws whenever we can
- This was due to the fact that discontinuities can easily occur for non-linear equations
- For Hamilton-Jacobi equations, we can get away with this non-conservative form because there is a relationship to conservation laws

$$\frac{\partial \phi}{\partial t} + H\left(\frac{\partial \phi}{\partial x}\right) = 0 \qquad \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} F\left(u\right) = 0$$

- This can be shown simply (but only in 1D)

Relationship to conservation laws

$$\frac{\partial \phi}{\partial t} + H\left(\frac{\partial \phi}{\partial x}\right) = 0 \qquad \frac{\partial u}{\partial t} + \frac{\partial}{\partial x}F\left(u\right) = 0$$

- To derive the relationship, we take the derivative of the H-J equation with respect to space
- We then define the variable $u = \frac{\partial \phi}{\partial x}$
- We can then express this as

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}H\left(u\right) = 0$$

- The spatial derivative of a Hamilton-Jacobi equation obeys a conservation law

Integral solution to a conservation law

$$\frac{\partial \phi}{\partial t} + H\left(\frac{\partial \phi}{\partial x}\right) = 0 \qquad \frac{\partial u}{\partial t} + \frac{\partial}{\partial x} F\left(u\right) = 0$$

- We can say that the Hamilton-Jacobi equation is the **integral solution to a conservation law**
- It is perfectly possible for the conservation law to have a discontinuity
- But, in order for the evolved level set function to have a discontinuity, the solution from the conservation law must be **a delta function**
- Whilst possible, when considering physical variables, we do not expect such behaviour to occur
- As a result, we can solve the Hamilton-Jacobi equation directly

Numerical methods for Hamilton-Jacobi equations

- Unfortunately, considering a new class of methods does require a few tweaks to numerical methods (at least, for more than first order accuracy)
- We do at least have a scalar equation
- A discretised Hamilton-Jacobi equation will often be written like

Time derivative

$$\boxed{\frac{\phi^{n+1} - \phi^n}{\Delta t}} + \boxed{\hat{H}^n}\left(\boxed{\phi^-_{x,y,z}, \phi^+_{x,y,z}}\right) = 0$$

A discrete representation of the function $H\left(\nabla\phi\right)$

An upwind and a downwind derivative approximation

Discrete representation

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \hat{H}^n \left(\phi^-_{x,y,z}, \phi^+_{x,y,z}\right) = 0$$

- The discrete representation, $\hat{H}$, must be carefully chosen such that the correct solution is obtained
- However, the methods which can be used are somewhat familiar
- For example, the Lax-Friedrichs scheme:

$$\hat{H} = H\left(\frac{\phi^-_{x,y,z} + \phi^+_{x,y,z}}{2}\right) - \alpha^{x,y,z}\frac{\phi^+_{x,y,z} - \phi^-_{x,y,z}}{2}, \quad \alpha^{x,y,z} = \max\left|H^{x,y,z}\left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right)\right|$$

- There are clear similarities (and differences) compared to the standard Lax-Friedrichs flux

$$f^n_{i+1/2} = \frac{1}{2}\frac{\Delta x}{\Delta t}\left(u^n_i - u^n_{i+1}\right) + \frac{1}{2}\left(f\left(u^n_{i+1}\right) + f\left(u^n_i\right)\right)$$

Discrete representation

$$\hat{H} = H\left(\frac{\phi^{-}_{x,y,z} + \phi^{+}_{x,y,z}}{2}\right) - \alpha^{x,y,z}\frac{\phi^{+}_{x,y,z} - \phi^{-}_{x,y,z}}{2}, \quad \alpha^{x,y,z} = \max\left|H^{x,y,z}\left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right)\right|$$

- Because we don't have a conservation law, the maximum information travel time is not as obvious
- Instead, we need to introduce some sort of wave speed through $H^{x,y,z}$, which is based on partial derivatives of H with respect to $\nabla\phi$
- This is not necessarily easy to compute, or nice to work with
- For normal motion, $H(\nabla\phi) = V_n|\nabla\phi|$:

$$H^{x} = \frac{V_n}{|\nabla\phi|}\frac{\partial\phi}{\partial x}$$

Level set equation for multiphysics

- It is clear that numerical methods for Hamilton-Jacobi equations could get complicated
- Fortunately, for material interfaces, we know that we are dealing with a feature governed by advection

$$\frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi = 0$$

- Here, $\mathbf{v}$ is the velocity of the physical materials at the interface, and this has no dependence on ϕ , which simplifies matters
- Firstly, for methods such as Lax-Friedrichs, the information travel speeds are easy to compute (and as expected)

$$H^x = v_x,\ H^y = v_y,\ H^z = v_z$$

Level set equation for multiphysics

$$\frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi = 0$$

- Secondly, the general case of Hamilton-Jacobi equations assumes the velocity is a function of the derivatives of the level set equation
- This is not the case for interface advection; we do not have to use the general centred difference to compute spatial derivatives
- Instead we can use more familiar upwind approaches

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + v_{x,i}^n \left(\frac{\partial \phi}{\partial x}\right)_i^n + v_{y,i}^n \left(\frac{\partial \phi}{\partial y}\right)_i^n + v_{z,i}^n \left(\frac{\partial \phi}{\partial z}\right)_i^n = 0$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i^n = \phi_{x,i}^{+} \text{ if } v_{x,i} < 0 \qquad \left(\frac{\partial \phi}{\partial x}\right)_i^n = \phi_{x,i}^{-} \text{ if } v_{x,i} > 0$$

UNIVERSITY OF CAMBRIDGE

Numerical methods for Hamilton-Jacobi equations

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + v_{x,i}^n \left(\frac{\partial \phi}{\partial x}\right)_i^n + v_{y,i}^n \left(\frac{\partial \phi}{\partial y}\right)_i^n + v_{z,i}^n \left(\frac{\partial \phi}{\partial z}\right)_i^n = 0$$

$$\left(\frac{\partial \phi}{\partial x}\right)_i^n = \phi_{x,i}^+ \text{ if } v_{x,i} < 0 \qquad \left(\frac{\partial \phi}{\partial x}\right)_i^n = \phi_{x,i}^- \text{ if } v_{x,i} > 0$$

- Numerical methods for Hamilton-Jacobi in this form are now 'simply' a matter of choosing approximations for $\phi_{x,y,z}^{\pm}$

- Since these are to be chosen based on the upwind direction, we can simply use first order derivatives

$$\phi_{x,i}^+ = \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x}, \qquad \phi_{x,i}^- = \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x}$$

- This is then very similar to the first numerical scheme we considered, the only difference is that velocity will be different in every cell

Numerical methods for Hamilton-Jacobi equations

$$\phi^{+}_{x,i} = \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x}, \qquad \phi^{-}_{x,i} = \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x}$$

- Higher order methods for computing $\phi^{\pm}_{x,y,z}$ exist, using more cells to compute these terms

- To express these, we use the **divided-difference** notation of Osher and Fedkiw, "Level Set Methods and Dynamic Implicit Surfaces" (2002)

$$D^0_i\phi = \phi_i \qquad \phi^{+}_i = D^1_{i+1/2}\phi = \frac{D^0_{i+1}\phi - D^0_i\phi}{\Delta x}, \qquad \phi^{-}_i = D^1_{i-1/2}\phi$$

$$D^2_i\phi = \frac{D^1_{i+1/2}\phi - D^1_{i-1/2}\phi}{2\Delta x}, \qquad D^3_{i+1/2}\phi = \frac{D^2_{i+1}\phi - D^2_i\phi}{3\Delta x}$$

- To save space, we consider only the 1D case

Hamilton-Jacobi ENO method

- Although slightly ugly notation, the divided-differences are useful for both fitting equations on slides, but also for coding – they are simple quantities which can be computed in a loop up to any order of accuracy
- And because they can be computed to high orders of accuracy easily, it is common to go beyond second order methods for these types of equation
- These form a class of ENO (Essentially Non-Oscillatory) method
- Here, the derivative terms are given by

$$\phi_x^{\pm}(x_i) = Q_1'^{\pm}(x_i) + Q_2'^{\pm}(x_i) + Q_3'^{\pm}(x_i)$$

- Where each term here is simply defined as:

Hamilton-Jacobi ENO method

One of 3 second derivatives

1 or -1

First order derivative

i-2, i-1 or i

2, -1 or 2

$$\boxed{Q'_1 = D^1_{k+1/2}\phi,} \qquad Q'_2 \boxed{= c}\boxed{[2(i-k)-1]}\Delta x \qquad Q'_3 = \boxed{c^\star}\boxed{\left[3(i-k^\star)^2 - 6(i-k^\star)+2\right]}(\Delta x)^2$$

$$\boxed{k = \begin{cases} i, \\ i-1 \end{cases}} \begin{matrix} v \le 0 \\ v > 0 \end{matrix} \qquad \boxed{(c}\boxed{k^\star)} = \begin{cases} \left(D^2_k\phi, \; k-1\right), & \left|D^2_k\phi\right| \le \left|D^2_{k+1}\phi\right| \\ \left(D^2_{k+1}\phi, \; k\right), & \left|D^2_k\phi\right| > \left|D^2_{k+1}\phi\right| \end{cases}$$

$$\boxed{c^\star} = \begin{cases} D^3_{k^\star+1/2}\phi, & \left|D^3_{k^\star+1/2}\phi\right| \le \left|D^3_{k^\star+3/2}\phi\right| \\ D^3_{k^\star+3/2}\phi, & \left|D^3_{k^\star+1/2}\phi\right| > \left|D^3_{k^\star+3/2}\phi\right| \end{cases}$$

This determines whether we use + or -

One of 4 third derivatives

Hamilton-Jacobi ENO method

$$Q_1' = D^1_{k+1/2}\phi, \qquad Q_2' = c\left[2\left(i-k\right)-1\right]\Delta x \qquad Q_3' = c^\star\left[3\left(i-k^\star\right)^2 - 6\left(i-k^\star\right)+2\right]\left(\Delta x\right)^2$$

$$k = \begin{cases} i, & v \leq 0 \\ i-1 & v > 0 \end{cases} \qquad (c, k^\star) = \begin{cases} \left(D^2_k\phi, \quad k-1\right), & \left|D^2_k\phi\right| \leq \left|D^2_{k+1}\phi\right| \\ \left(D^2_{k+1}\phi, \quad k\right), & \left|D^2_k\phi\right| > \left|D^2_{k+1}\phi\right| \end{cases}$$

$$c^\star = \begin{cases} D^3_{k^\star+1/2}\phi, & \left|D^3_{k^\star+1/2}\phi\right| \leq \left|D^3_{k^\star+3/2}\phi\right| \\ D^3_{k^\star+3/2}\phi, & \left|D^3_{k^\star+1/2}\phi\right| > \left|D^3_{k^\star+3/2}\phi\right| \end{cases}$$

- Not easy reading, but essentially this is just picking the smoothest third-order accurate approximation to $\left(\frac{\partial\phi}{\partial x}\right)^n_i$ from all available stencils
- And to code this, it is just a series of logic statements
- However, if the solution is smooth over the entire set of third-order stencils, why not combine them to get an even higher order approximation for very little extra effort?

Hamilton-Jacobi WENO method

- Combining the possible options for an ENO method gives you a WENO (Weighted Esentially Non-Oscillatory) method
- If we work out how the derivative approximations are computed for the ENO scheme, we get one of six possible values

$$\left(\phi_x^-\right)_i : \quad v_1 = D^-\phi_{i-2}, \quad v_2 = D^-\phi_{i-1}, \quad v_3 = D^-\phi_i, \quad v_4 = D^-\phi_{i+1}, \quad v_1 = D^-\phi_{i+2},$$

$$\left(\phi_x^+\right)_i : \quad v_1 = D^+\phi_{i+2}, \quad v_2 = D^+\phi_{i+1}, \quad v_3 = D^+\phi_i, \quad v_4 = D^+\phi_{i-1}, \quad v_1 = D^+\phi_{i-2},$$

$$\phi_x^{\pm,1} = \frac{v_1^\pm}{3} - \frac{7v_2^\pm}{6} + \frac{11v_3^\pm}{6} \quad \phi_x^{\pm,2} = -\frac{v_2^\pm}{6} - \frac{5v_4^\pm}{6} + \frac{v_4^\pm}{3} \quad \phi_x^{\pm,3} = \frac{v_3^\pm}{3} + \frac{5v_4^\pm}{6} - \frac{v_5^\pm}{6}$$

- Where, to save space, $D^-\phi_i = D^1_{i-1/2}$ and $D^+\phi_i = D^1_{i+1/2}$ are used

Hamilton-Jacobi WENO method

$$\phi_x^{\pm,1} = \frac{v_1^\pm}{3} - \frac{7v_2^\pm}{6} + \frac{11v_3^\pm}{6} \quad \phi_x^{\pm,2} = -\frac{v_2^\pm}{6} - \frac{5v_4^\pm}{6} + \frac{v_4^\pm}{3} \quad \phi_x^{\pm,3} = \frac{v_3^\pm}{3} + \frac{5v_4^\pm}{6} - \frac{v_5^\pm}{6}$$

- WENO schemes achieve higher order accuracy by taking a linear combination of the possible derivative approximations,

$$\phi_x^\pm = \omega_1 \phi_x^{\pm,1} + \omega_2 \phi_x^{\pm,2} + \omega_3 \phi_x^{\pm,3}$$

- In order to compute the three weights, three measures of **smoothness** are used

$$S_1 = \frac{13}{12}(v_1 - 2v_2 + v_3)^2 + \frac{1}{4}(v_1 - 4v_2 + 3v_3)^2 \quad S_2 = \frac{13}{12}(v_2 - 2v_3 + v_4)^2 + \frac{1}{4}(v_2 - v_4)^2 ,$$

$$S_3 = \frac{13}{12}(v_3 - 2v_4 + v_5)^2 + \frac{1}{4}(3v_3 - 4v_4 + v_5)^2$$

- Since all the v terms are first derivative approximations, if these are all identical, we have $S_{1,2,3} = 0$, the solution is completely smooth

Hamilton-Jacobi WENO method

$$\phi_x^{\pm} = \omega_1 \phi_x^{\pm,1} + \omega_2 \phi_x^{\pm,2} + \omega_3 \phi_x^{\pm,3}$$

- If we have a perfectly smooth solution, it can be shown that the optimal weights for this problem are $\omega_1^\star = 0.1$, $\omega_2^\star = 0.6$, $\omega_3^\star = 0.3$
- The actual weights then use this optimal solution do give weights

$$\alpha_i = \frac{\omega_i^\star}{(S_i + \epsilon)^2}, \qquad \omega_i = \frac{\alpha_i}{\alpha_1 + \alpha_2 + \alpha_3}$$

- So, the bigger the slope, the smaller the contribution from a given weight is
- The value $\epsilon = 10^{-6}$ ensures that in the case of perfect smoothness ($S_{1,2,3} = 0$), there is no division by zero

Time step with the WENO scheme

- This WENO scheme takes a third order ENO scheme, and using all the information available, results in a fifth order scheme
- With this level of spatial accuracy, we now worry about time accuracy
- And, if we do try using a first order time update, we often see oscillations
- It is difficult to say exactly why, since we no longer have any TVD properties
- However, in practice, a third-order Runge-Kutta time update (or higher) should be used

Why do we use non-conservative methods?

- For a material interface, it is possible to write the level set equation in conservative form

$$\frac{\partial \rho\phi}{\partial t} + \nabla \cdot \rho\phi\mathbf{v} = 0$$

- This could still be used for solving the level set equation
- However, because we know that for smooth initial data, discontinuities **will not arise** for the Hamilton-Jacobi form of the equation, we don't need to use this form
- When using level set methods to model an interface, there is (usually) a jump in density at the interface
- In this conservative formulation, this means a contact discontinuity (with associated smearing errors) at the **only** place we need the method to be accurate

Outline

- Implicit boundary methods
- Numerical methods for level sets
- Defining level set functions
- Reinitialisation

Why do we like level set methods?

- The main reason level set methods are so popular is because they allow for high accuracy at the material interface
- This is because they evolve a **smooth function**, rather than a discontinuous jump
- But there are a couple of additional advantages to using a function to describe the interface location
- Both the normal direction and curvature of the level set function are easy to compute

$$\hat{\mathbf{n}} = \frac{\nabla\phi}{|\nabla\phi|}, \qquad \kappa = \nabla\cdot\hat{\mathbf{n}} = \nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)$$

- Although these are generally computed at cell-centres, because the level set function is smooth, they are accurate enough close to the interface (or can be interpolated)

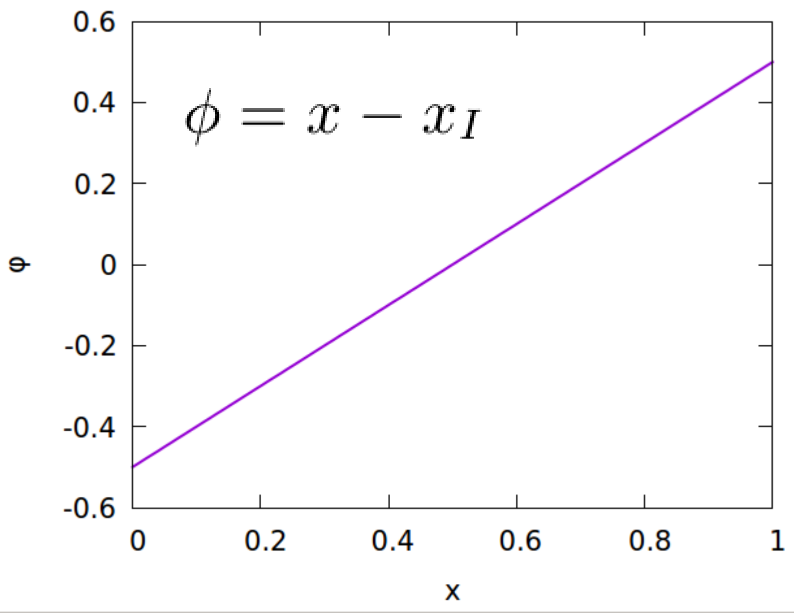
Signed distance functions

$$\hat{\mathbf{n}} = \frac{\nabla\phi}{|\nabla\phi|}, \qquad \kappa = \nabla\cdot\hat{\mathbf{n}} = \nabla\cdot\left(\frac{\nabla\phi}{|\nabla\phi|}\right)$$

- The normal vector (and to some extent the curvature) appear a lot in multiphysics methods, as well as other applications of level set methods
- There is one more feature, typically found in level set methods, and that is the **signed distance function**
- In this case, the magnitude of the level set function gives the **shortest distance to the interface**, where this distance will always be in the normal direction
- It can be useful to know how close we are to the interface, including for some multiphysics methods
- But also, a signed distance function has $|\nabla\phi| = 1$, 'halfway' between discontinuous and flat

Example level set functions

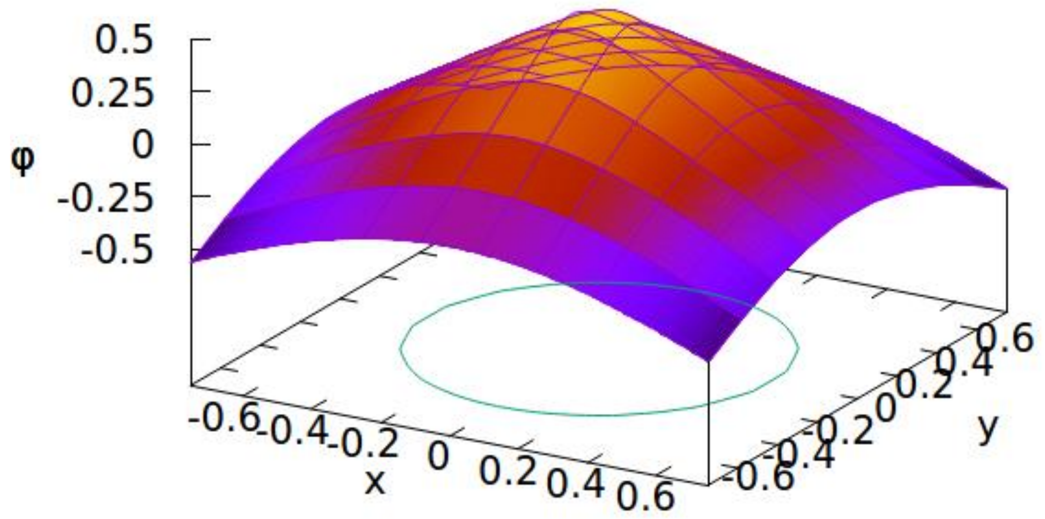
- When setting up initial data for a level set method, you need some initial signed distance function based on the initial material interfaces
- This might be complicated to define for complex shapes, though often simple functions are sufficient
- In 1D, signed distance functions are straight lines for single interfaces
- Multiple interfaces are also simple, using saw-tooth functions



Example level set functions

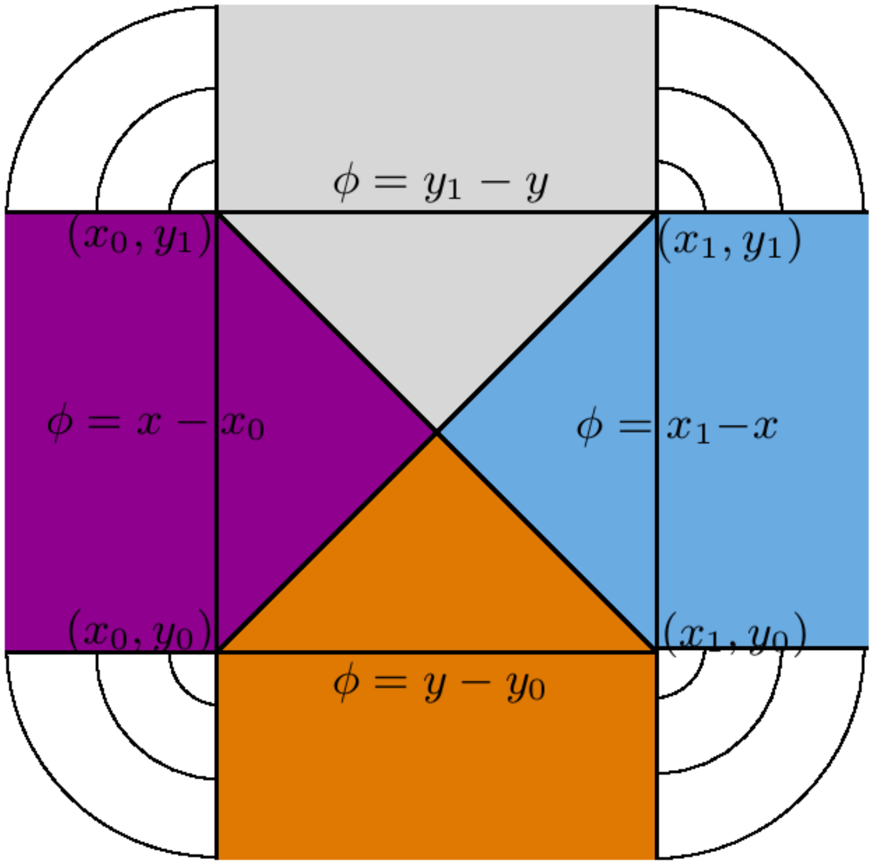
- Many applications in 2D use a circle, including shock-bubble interactions
- The level set function for this is a cone

$$\phi = r - \sqrt{(x - x_0)^2 + (y - y_0)^2}$$



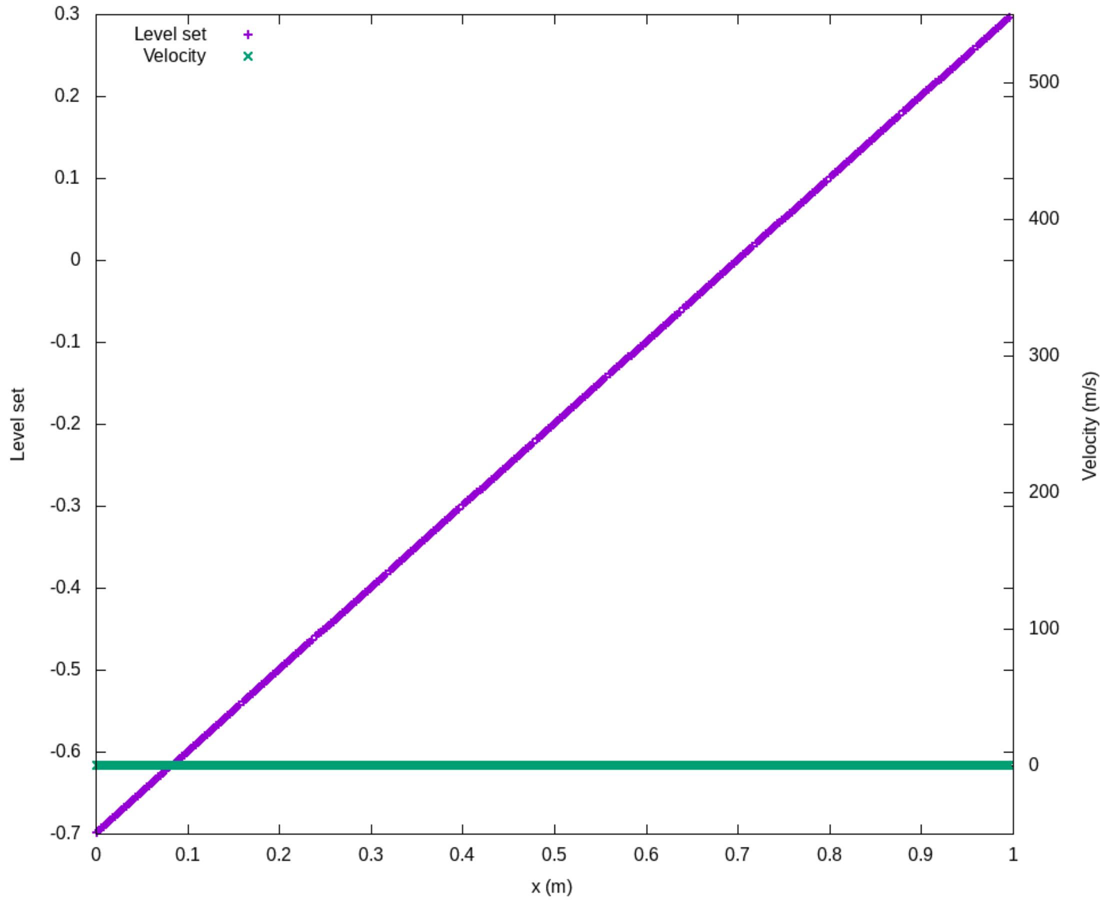
Example level set functions

- Polygons are conceptually simple, but can be a pain to code
- You are effectively minimising the distance from each line in the polygon, and to each end of each line
- These algorithms can be expensive, but they **only need to be done once**
- And algorithms like this extend to triangulated 3D geometries, e.g. STL files



Maintaining signed distance functions

- Constructing signed distance functions can take a bit of effort, but it is a well-understood problem
- However, evolution of the level set function does not necessarily maintain this signed distance nature
- This is because, although it follows an "advection equation", the velocity of the advection is **not constant** in space – this can lead to steepening



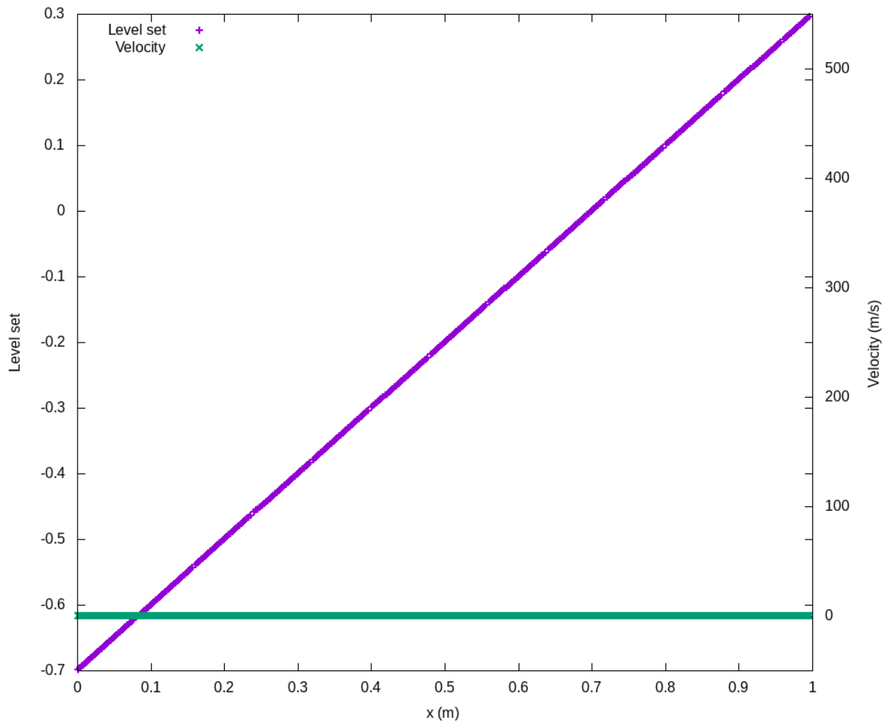
Outline

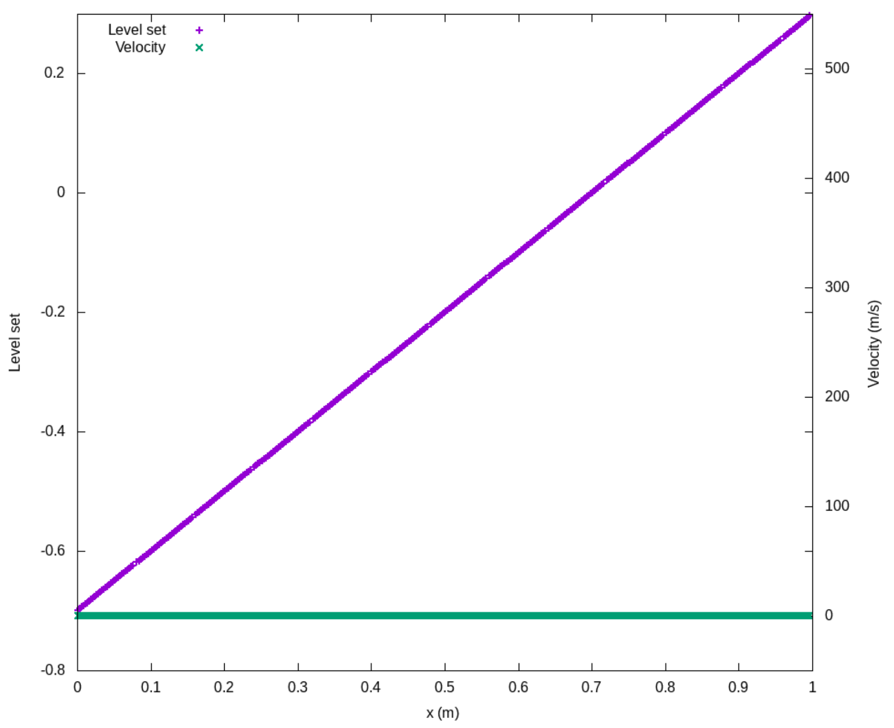
- Implicit boundary methods
- Numerical methods for level sets
- Defining level set functions
- Reinitialisation

Reinitialisation

- **Reinitialisation** is the process by which a level set can be returned to a signed distance function, whilst maintaining the location of the zero-contour
- Otherwise the level set function could become too smeared – easy to introduce errors into the interface location with small changes
- Or it could become too steep and resemble a discontinuity – at which point the numerical methods for level set equations would no longer be valid
- The idea behind reinitialisation is that as long as we don't change where the interface (zero contour) is, we could define our level set function as anything we want
- So if it starts to evolve away from a signed distance function, we can make it a signed distance function again

Reinitialisation example





UNIVERSITY OF CAMBRIDGE

Reinitialisation

- For an implicitly defined interface location, we have a major challenge – we do not know exactly where the interface is
- If we attempt to ensure that our level set function has $|\nabla \phi| = 1$ everywhere, then in more than 1D, we cannot achieve this without slightly moving the interface
- This movement tends to smear out the features along the interface, and we can lose a lot of information
- To maintain the position of the interface, we must ensure that any cell directly adjacent to the interface is not changed, i.e. it is held fixed during the reinitialisation procedure
- Although this could lead to local sharpening or steepening, we have found this to be much more accurate than allowing the interface to move

Reinitialisation in 1D (MODIFY ANIMATIONS!)

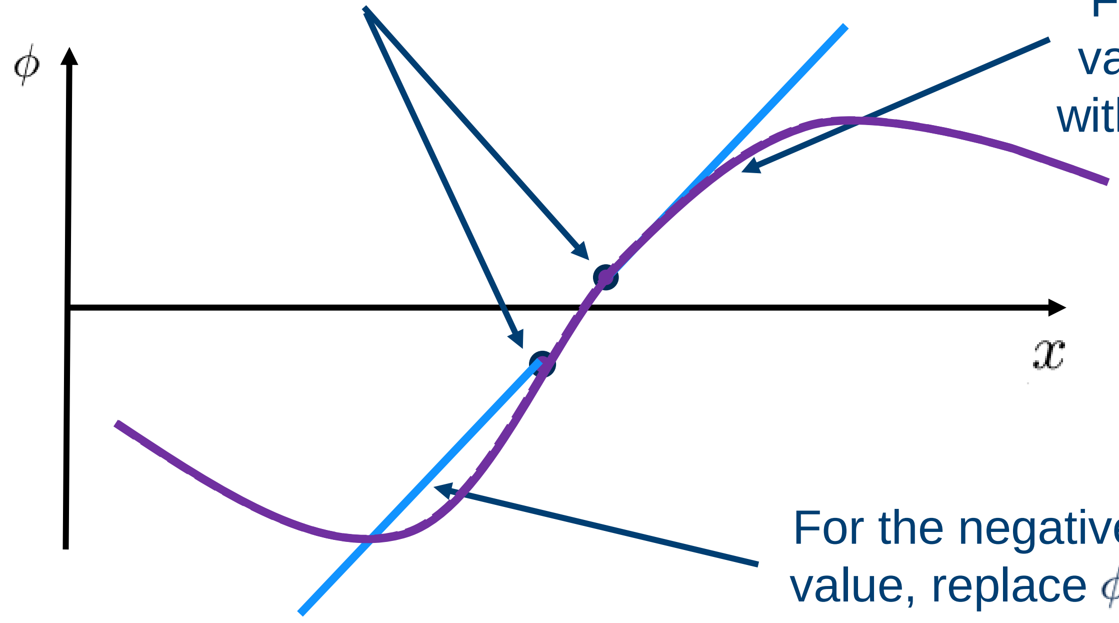
- In 1D, reinitialisation is straightforward

Identify points either side of the interface $\phi_{i+1}\phi_i < 0$

For the positive value, replace ϕ with a line of slope 1

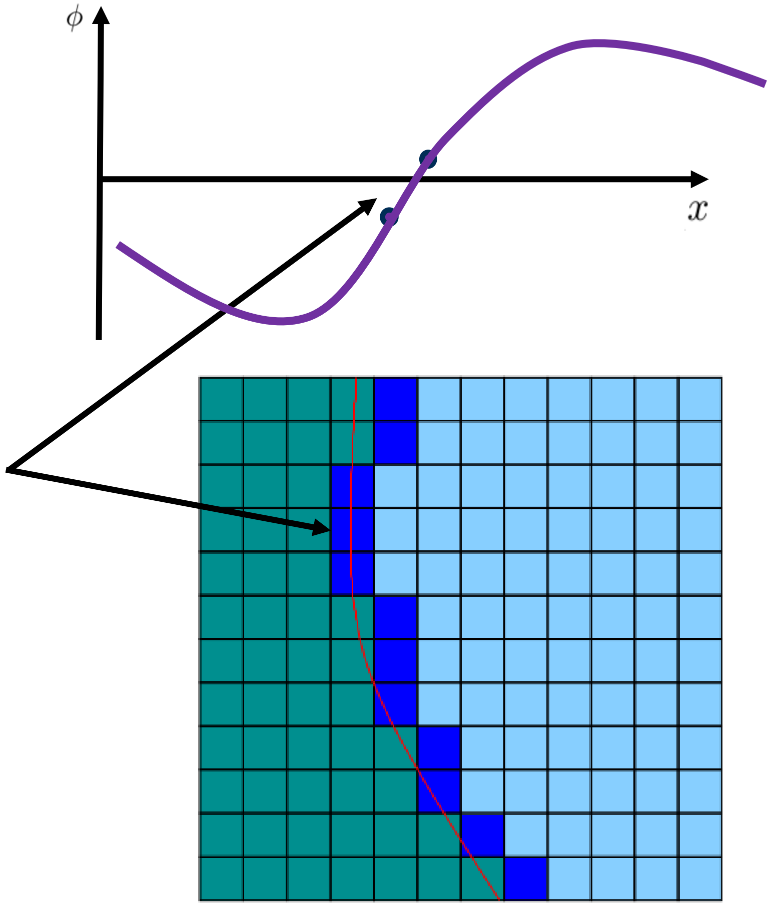
For the negative value, replace ϕ with a line of slope -1

All other level set information is unnecessary



Reinitialisation

- Conceptually, not a lot changes as we move beyond 1D
- We are still solving $|\nabla\phi| = 1$
- We just need a more complex algorithm than drawing a straight line
- We still have fixed points adjacent to the interface though (only shown on one side of the interface in 2D)
- And the rest of the data is still unnecessary for reinitialisation
- Effectively we are solving two steady-state problems with Dirichlet boundary conditions



Eikonal equation

- Reinitialisation involves solving, $|\nabla\phi| = 1$ which is part of a class of equations known as **Eikonal equations**
- They also have applications in describing electromagnetic beam attenuation and image reconstruction
- The Eikonal equation is an example of a **steady-state hyperbolic equation**, for level set equations, information propagates out from the boundary with speed 1
- Using naive numerical techniques would require every solution to evolve the propagation of the information from the boundary across the computational domain
- This would be an expensive iterative procedure, there could be the same number of steps as there are cells in the domain
- Because we may need to reinitialise frequently, we may need something better

Iterative reinitialisation

- Although it is not efficient, iterative reinitialisation is fairly straightforward, and can be made to work with the algorithms we have already implemented
- The idea is to consider the solution process to the Eikonal equation as **evolving a solution to a steady state**
- We introduce a **fictitious time** variable, τ, to construct a PDE

$$\frac{\partial \phi}{\partial \tau} + \operatorname{sgn}(\phi)\left(|\nabla \phi| - 1\right) = 0$$

- Provided that the slopes of ϕ are always calculated in the **upwind direction** (closer to the interface) then this equation will evolve to steady state
- At steady state, we have $\frac{\partial \phi}{\partial t} = 0$, hence we are satisfying the Eikonal equation

Iterative reinitialisation

$$\frac{\partial \phi}{\partial \tau} + \operatorname{sgn}(\phi)\left(|\nabla \phi| - 1\right) = 0$$

- Choosing the upwind direction for the derivatives is fairly straightforward
- Firstly, we can use first order derivatives, since we are aiming for a linear solution
- Secondly, the sign of the derivative tells us which direction upwind actually is
- For example, if $\operatorname{sgn}(\phi)\frac{\partial \phi}{\partial x} \geq 0$ then we should use, ϕ_x^- otherwise we use ϕ_x^+

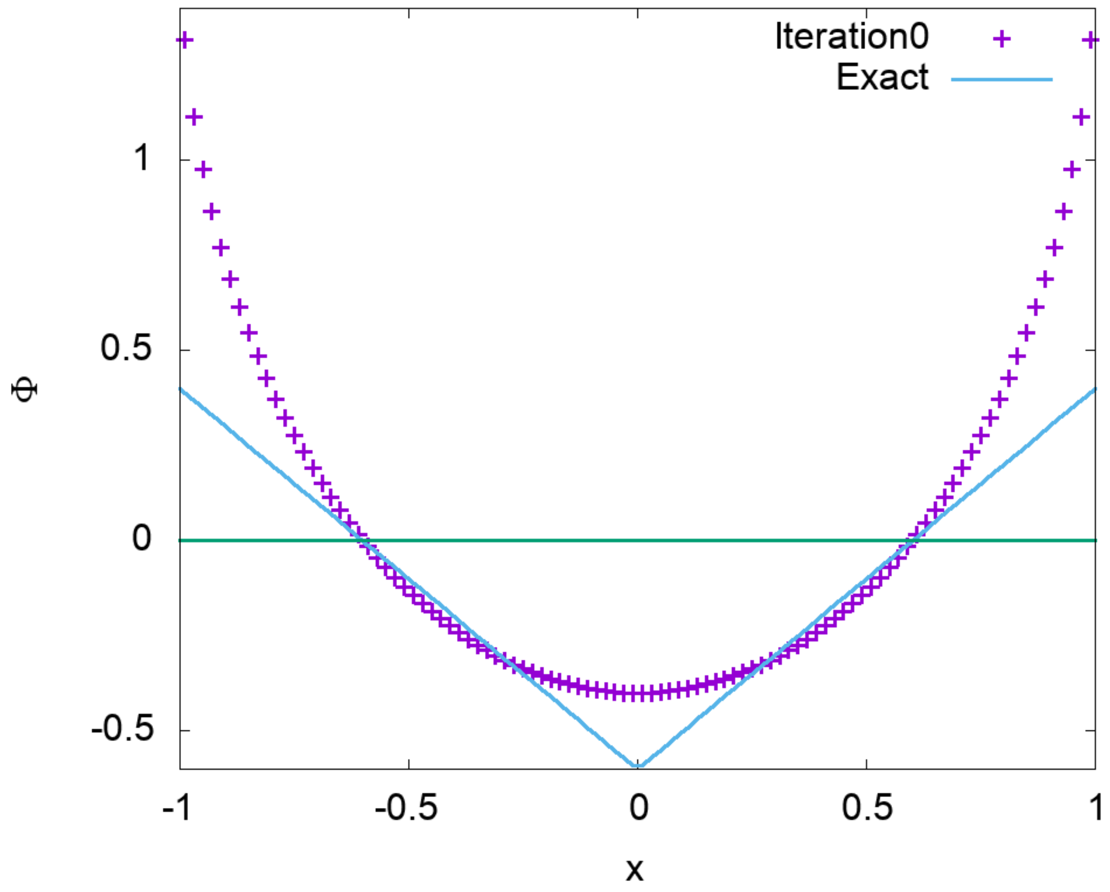
$$\phi_{x,i}^{+} = \frac{\phi_{i+1,j,k} - \phi_{i,j,k}}{\Delta x}, \qquad \phi_{x,i}^{-} = \frac{\phi_{i,j,k} - \phi_{i-1,j,k}}{\Delta x}$$

- This assumes that the derivative has the same sign in both directions; see Fedkiw et al. for limiting cases

Why use iterative reinitialisation?

- Full iterative reinitialisation is expensive; however it can still be used operationally for multiphysics
- Recall, we don't actually care about what the level set function looks like everywhere, all we need is to make sure it continues to look good (obey the Eikonal equation) close to the interface
- This will ensure the evolution of the level set function is accurate close to the interface
- If it is innacurate elsewhere, then we don't mind, future reinitialisation would take care of this if it gets close to the interface (providing no new interfaces are created...)
- Because the information is **propagated outwards from the interface**, then we achieve this relatively quickly – Fedkiw et al. say about 10 iterations is sufficient

Iterative reinitialisation example



Efficient reinitialisation

- Fortunately, there are plenty of more efficient reinitialisation algorithms around, and some are also not too complicated
- The idea is that each iteration sets one more cell to be a signed distance function
- So rather than updating the entire domain to achieve this, why not reinitialise this single cell
- The challenge is how do we identify this 'next cell' in more than 1D
- Two methods we consider which are inspired by this logic are the **fast sweeping** and **fast marching** methods

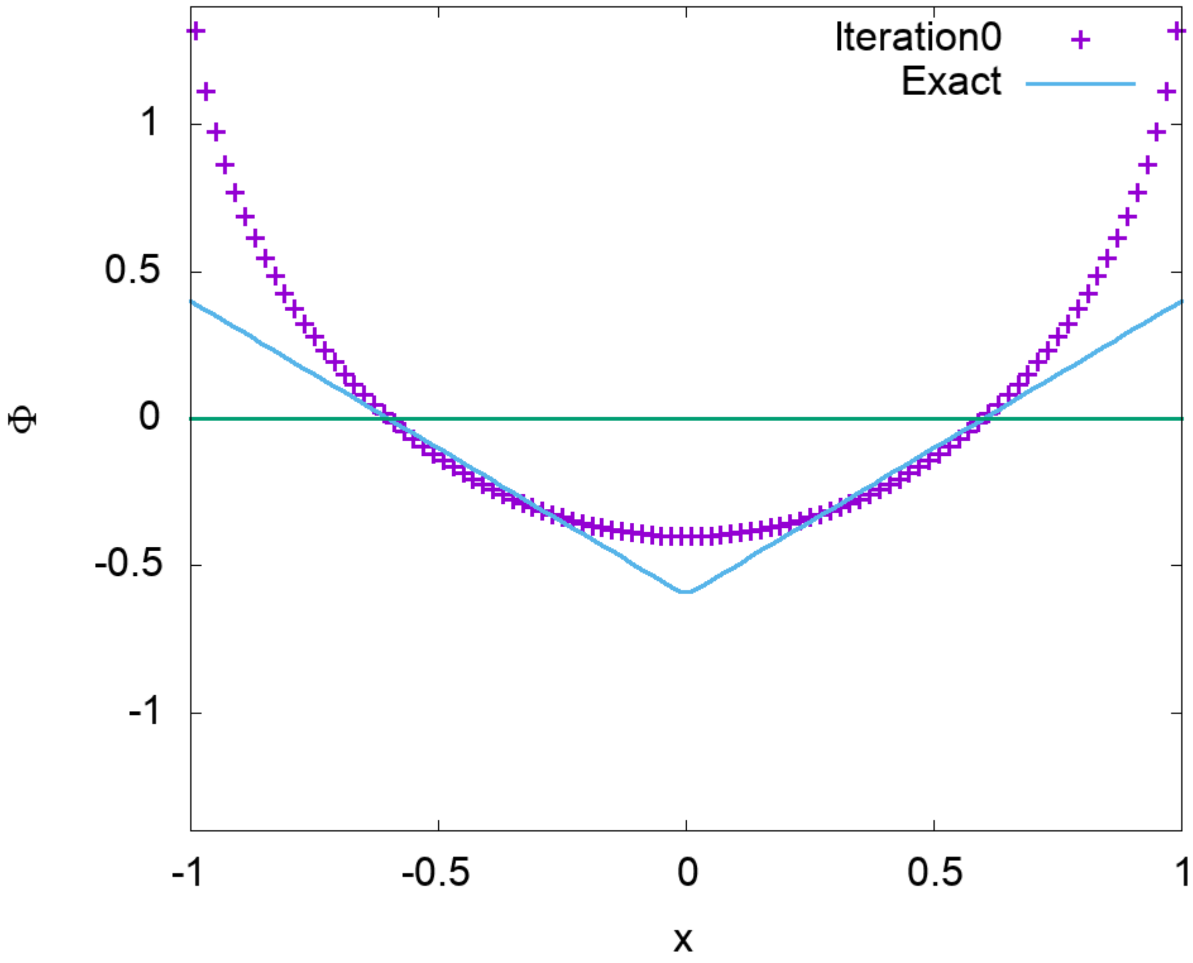
Fast marching method

- The principle behind the **fast marching method** is that there is a **correct** order in which we consider the cells when reinitialising
- If we start at the closest cell to the interface, and use this to compute reinitialised values for **all possible neighbours** (those not also adjacent to the interface, or in the wrong material)
- Once this has happened, this cell can no longer contribute anything to the solution, we can mark it as **fixed**
- We then chose the next-smallest cell, and carry on
- Quite a complex algorithm; every time we compute a new value, we have to work out where it sits in the list of smallest-to-largest cells (it will not necessarily be the largest value)
- Can use **multimaps** in C++, and is an order $O\left(N \log N\right)$ scheme

Fast sweeping method

- We shall not consider the exact details of the fast marching method; it is complex, care is needed to avoid circular dependencies, and a simple, and more efficient method exists (at least in non-parallel computation)
- The **fast sweeping method** of Zhao (2004) avoids this by considering a series of one-dimensional reinitialisation steps (**sweeps** of the computational domain)
- Sweeps are performed in different directions, along different coordinate axes, e.g. *x*-direction from 0 to N, then *y*-direction from 0 to N, then *x*-direction from N to 0, then *y*-direction from N to 0
- With each sweep, all available information is used to compute the level set function, regardless of whether it is accurate or not, but if it gives a better value, than a guess from a previous sweep, the new value is kept

Fast sweeping method example



Fast sweeping method

- In this example, the first thing the fast sweeping method did was 'remove' all values not adjacent to the interface
- What we actually did was set them to a **sufficiently large (positive or negative) value**, e.g. greater than the distance across the domain
- This way, if any newly-calculated level set quantity is smaller than the current guess, we know it is a better approximation
- Once we have done this, we are ready to begin the actual algorithm

Fast sweeping method – the algorithm

1. Starting with the first cell in the sweep, check if we are adjacent to an interface
2. If we aren't, move to the next cell. Repeat until we are adjacent to the interface, and then keep going until we are no longer adjacent to the interface, **but the previous cell in the sweep was adjacent**
3. Compute the **minimum** neighbours of our closest cell, ϕ_x, ϕ_y, ϕ_z e.g. for positive level set, $\phi_x = \min\left(\phi_{i+1,j,k}, \phi_{i-1,j,k}\right)$

   or negative $\phi_x = \max\left(\phi_{i+1,j,k}, \phi_{i-1,j,k}\right)$
4. Plugging these values into our Eikonal equation gives a quadratic to solve

$$\left(\frac{\boxed{\hat{\phi}_{i,j,k}} - \phi_x}{\Delta x}\right)^2 + \left(\frac{\boxed{\hat{\phi}_{i,j,k}} - \phi_y}{\Delta y}\right)^2 + \left(\frac{\boxed{\hat{\phi}_{i,j,k}} - \phi_z}{\Delta z}\right)^2 = 1$$

This is the cell whose value we are computing

Fast sweeping method – the algorithm

$$\left(\frac{\hat{\phi}_{i,j,k} - \phi_x}{\Delta x}\right)^2 + \left(\frac{\hat{\phi}_{i,j,k} - \phi_y}{\Delta y}\right)^2 + \left(\frac{\hat{\phi}_{i,j,k} - \phi_z}{\Delta z}\right)^2 = 1$$

5. We try to compute $\hat{\phi}_{i,j,k}$
   - If this is **smaller in magnitude** than the current guess for this cell, we keep it
   - For positive level set, this is always the positive root, and for negative level set, the negative root
   - If the solution is ill-defined (negative square root), remove the term largest ϕ_x, ϕ_y, ϕ_z and try again (until successful)
6. We now move on to the next cell, and repeat from step 3
7. If we reach another interface cell, we do not do anything until we have got back to a non-adjacent cell whose previous cell was adjacent

Fast sweeping method – the algorithm

8. Once we have reached the edge of the domain, move to the next cell at the start of the domain

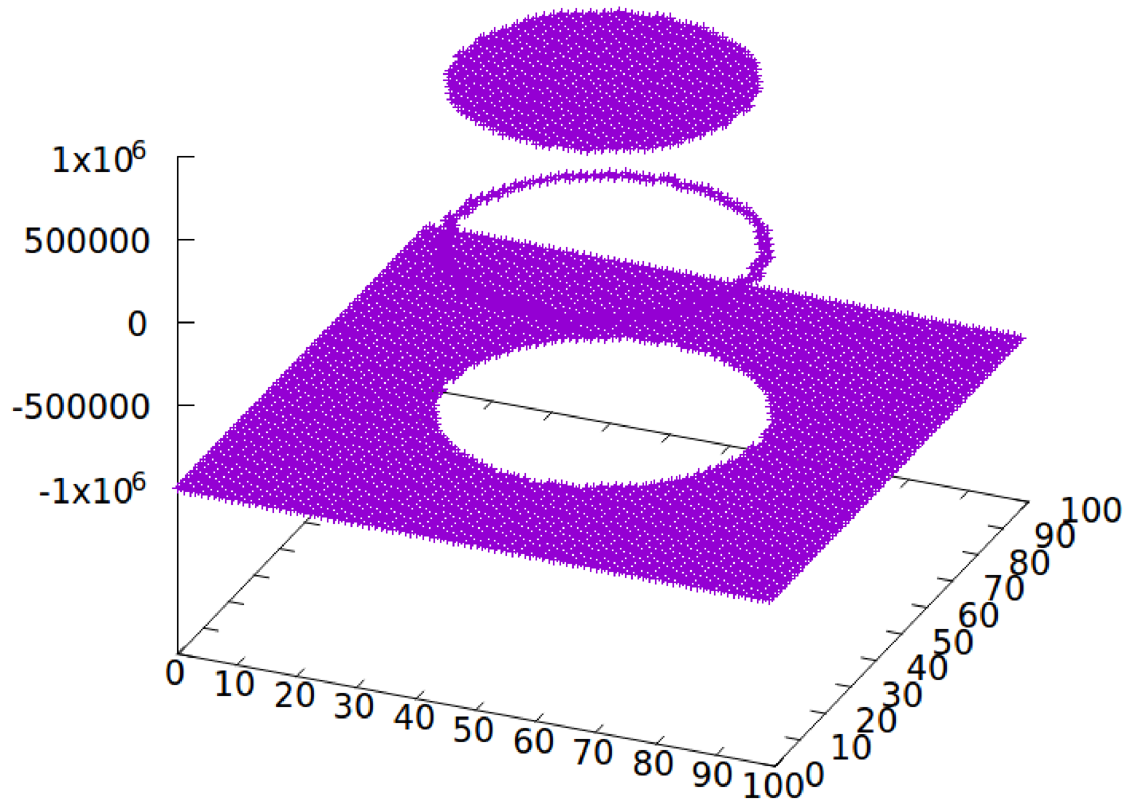
9. Once we have attempted to apply the algorithm to every cell in the domain, we have computed the sweep

10. Now we switch directions, as described previously, and begin the sweep algorithm all over again

- The efficiency of the algorithm comes from the fact that each sweep is just a **single loop over the domain**
- This then gives an algorithm of order $O(N)$

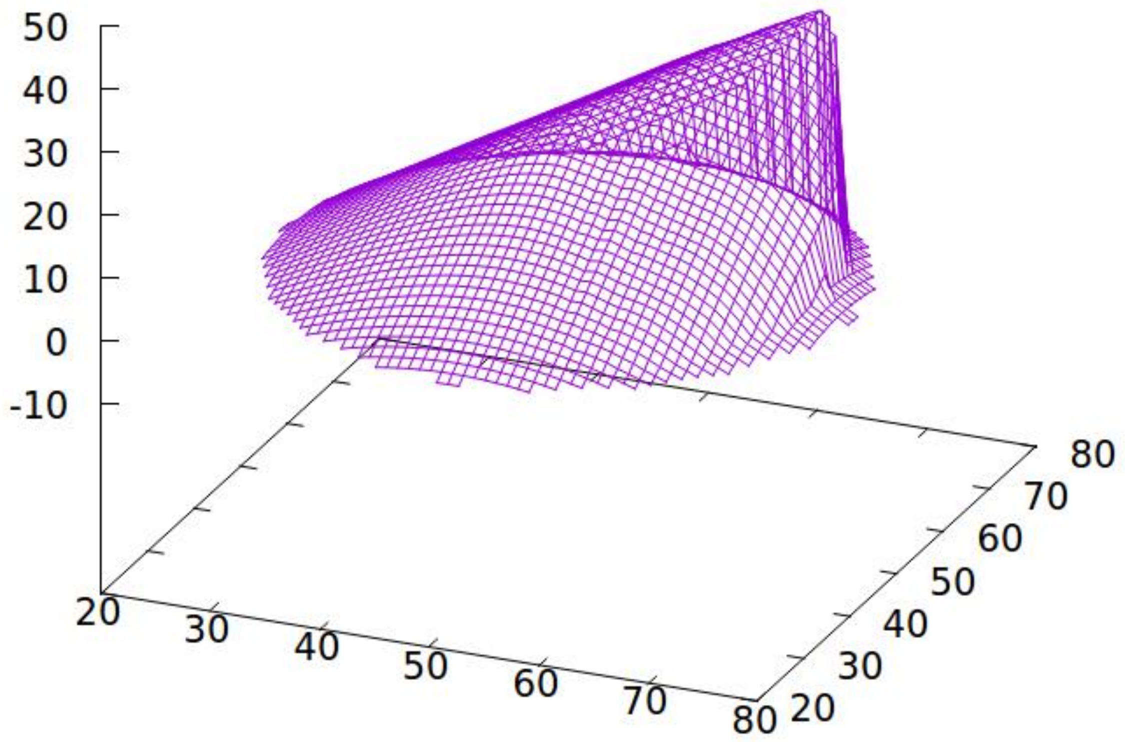
Fast sweeping method 2D example

- Initially we set all values not adjacent to the interface to a large number



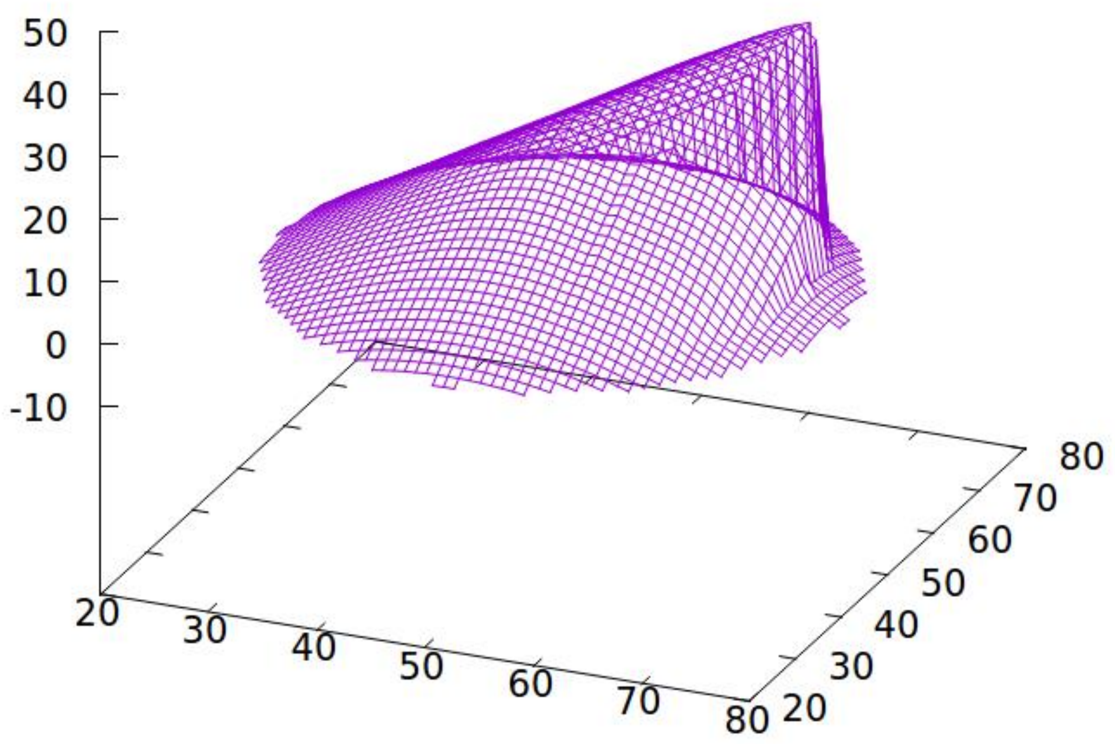
Fast sweeping method 2D example

- We now do a single sweep – note that it makes good use of interface cells



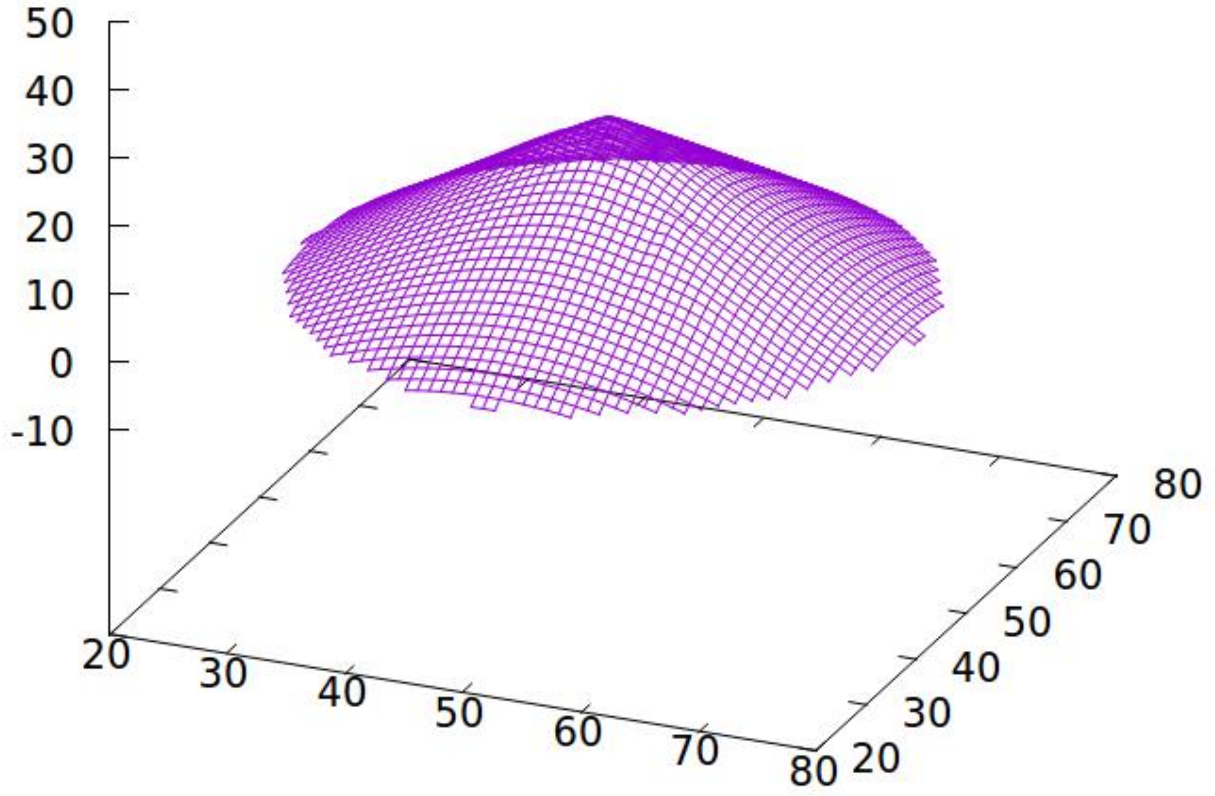
Fast sweeping method 2D example

- As a result, the second sweep doesn't look very different (in this example)

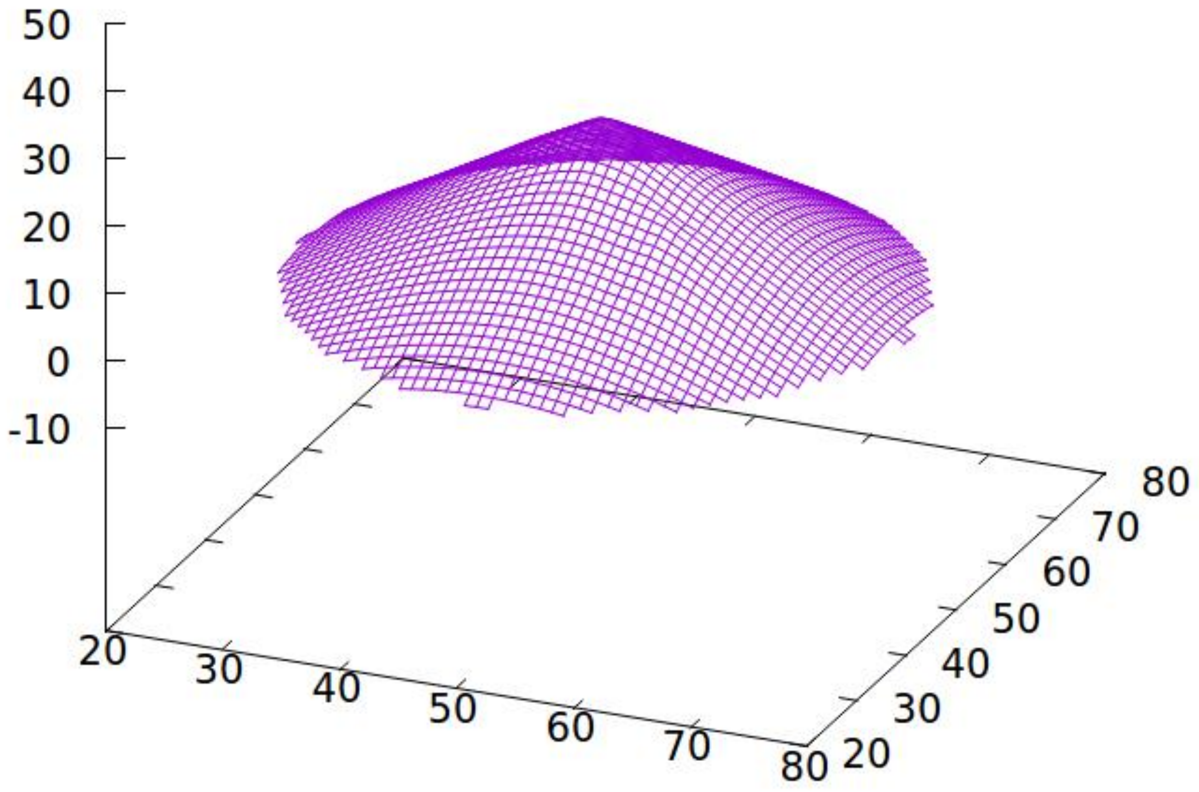


Fast sweeping method 2D example

- By the third sweep, we have nearly all the information now



- And the final sweep again doesn't do a lot



What method should we use?

- In 1D, none of these – just trivially reconstruct straight lines
- Otherwise, the fast sweeping method is generally the best choice, due to efficiency, but the simplicity of the iterative method might be advantageous when testing methods
- Although fast marching methods are $O\left(N\right)$, there can be cases where the absolute number which multiplies the error is large
- This is especially true in non-multiphysics applications, in such cases, the fast marching method may have advantages
- A single sweep in all directions (2xN) is enough for us, and the fast sweeping method is the obvious choice
- For parallel and GPU coding, additional techniques are available, including the **block fast iterative** and **block fast sweeping** methods