LEVEL-SET METHODS

MODELING AND ANIMATION, LAB 5

Mark Eric Dieckmann Erik Junholm Emma Broman Robin Skånberg

Wednesday 22nd March, 2023

Abstract

This lab targets the numerics involved in implementing a level set framework. A "level set" is an implicit surface that can be deformed by solving a set of *partial differential equations* (PDEs). The implicit representation provides several benefits, such as natural changes in topology and robust deformations. We discuss stable numerical schemes that can discretize the *parabolic* and *hyperbolic* PDEs covered in this lab. You will implement PDEs for performing dilation, erosion, advection and smoothing.

1 Introduction

In order to keep the instructions for this lab self contained, a thorough review of the level set method is presented in Section 2. We will start by defining a level set surface and then derive the necessary equations of motion which make it possible to deform the surface (move the surface in time). Then we present the discretization of these equations. It is important to realize that we deal with both a *temporal* and a *spatial* discretization. The first determines how we evolve the surface in time using discrete time steps, while the latter determines how we compute discrete differentials (gradients, curvature) in space. Furthermore, we emphasize that two different classes of PDEs (*hyperbolic* and *parabolic*) give rise to different spatial discretization strategies ("upwinding" and "central differencing").

Section 2.3 will describe why we want to keep the level set function as close as possible to a *signed distance function*. That is, a function where each point describes the *closest distance* to the surface, and where the *sign* of the distance determines whether we are inside or outside the surface. This is achieved through *reinitialization*, using algorithms such as *fast marching* or *fast sweeping*.

Finally, Section 2 will be ended with a presentation of localized *narrow band schemes*. We explain how to optimize computations by restricting the solution of the level set PDEs to a "band" around the surface. This construction is motivated by noting that we are only interested in the actual surface (zero-distance level set) rather than arbitrary points of the level set function.

After the theoretical review of level set methods, we will discuss how to construct PDEs for performing common operations such as dilation, erosion, advection and smoothing of the surface. We will also present how to implement a basic level set framework for performing these operations.

2 Level set methods

The level set method implicitly represents an interface as a level set S of the level set function ϕ as:

$$S = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}) = h\} \tag{1}$$

We define the set of points inside, S_{inside}, and the set of points outside, $S_{outside}$, as:

$$S_{inside} = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}) < h\} \tag{2a}$$

$$S_{outside} = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}) > h\} \tag{2b}$$

We consider interfaces with $d = 2$ (isolines) or $d = 3$ (isosurfaces). In what follows, we introduce some geometrical properties. It can be shown that the *normal* and curvature of any level set of ϕ is [3]:

$$\boldsymbol{n} = \frac{\nabla\phi}{|\nabla\phi|} \tag{3a}$$

$$\kappa = \nabla \cdot \boldsymbol{n} \tag{3b}$$

To make it possible to deform this implicit geometry, we derive equations of motion for the level set function by introducing time dependence to Equation 1. There are two distinct ways of doing this, the first of which varies the isovalue h over time, such that $S(t) = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}) = h(t)\}$. This is called the *static level set formulation*. It describes the evolution of level sets of a function as the isovalue changes. However, the level sets cannot intersect by definition, imposing a great constraint on the deformations possible. The second option of introducing time dependence is to vary the level set function itself over time, such that $S(t) = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}, t) = h\}$. To derive equations of motion for S we study a point $\boldsymbol{\alpha}(t)$ which is on S, and thus follows the motion of S. Since $\boldsymbol{\alpha}(t)$ is on S, we know that $\phi(\boldsymbol{\alpha}(t), t) \equiv h$. Differentiating this with respect to time, we get:

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \cdot \frac{d\boldsymbol{\alpha}}{dt} \tag{4a}$$

$$= -F\,|\nabla\phi| \tag{4b}$$

where F is referred to as the *level set speed function*:

$$F = \boldsymbol{n} \cdot \frac{d\boldsymbol{\alpha}}{dt} = \frac{\nabla\phi}{|\nabla\phi|} \cdot \frac{d\boldsymbol{\alpha}}{dt} \tag{5}$$

This can be interpreted as the speed $d\boldsymbol{\alpha}/dt$ at a point $\boldsymbol{\alpha}$ projected onto the normal at that point. That is, the speed in the normal direction. The key idea is that Equation 4 provides the user with the means of modifying the level set function, to achieve a particular motion of the surface (interface) S.

These PDE's are known as the *fundamental level set equations* and they yield the *dynamic level set formulation*. This allows for any deformations of the interface, making it the method of choice for most applications. In theory, the choice of h is arbitrary. However, it makes practical sense to use $h = 0$ as this allows the definition of an *inside* and an *outside* of the interface using a simple sign convention.

Equation 4 assumes that the level set function ϕ is continuous with well-defined gradients everywhere in the domain. However, to use these mathematical expressions on a computer, we need to discretize the function, only computing values at discrete points. More specifically, we have to discretize both the *temporal domain* and the *spatial domain*. Depending on the particular PDE used, this introduces a number of numerical implications, constraining the speed at which we can evolve the PDE. Next we will study the temporal and spatial discretizations separately.

2.1 Temporal discretization

The temporal discretization determines how we evolve Equation 4 in time. Since the time needs to be discrete, we evolve the PDE using discrete time steps of Δt, using either an *implicit* or *explicit* integration scheme. Implicit schemes are *unconditionally stable*, regardless of the time step Δt. However, since the computation of these schemes relies on solving a large system of equations, they are difficult and computationally demanding. Explicit methods are easy to implement but suffer from stability constraints imposed on the time step. Despite this constraint, explicit methods are often used for solving level set equations. A simple, first order accurate, explicit scheme is the *forward Euler* given by:

$$\frac{\partial\phi}{\partial t} \approx \frac{\phi^{n+1} - \phi^n}{\Delta t} \tag{6}$$

where ϕ^n denotes the values of ϕ at time instance t^n while ϕ^{n+1} denotes the values of ϕ at the time instance $t^n + \Delta t$. Although practical experience shows that this first order scheme usually suffices when solving level set equations, the accuracy can be improved by using the *total variation diminishing Runge-Kutta* (TVD RK) schemes proposed by Shu and Osher [12]. The first order TVD RK scheme is simply the forward Euler. The accuracy is increased by sequentially taking Euler steps and linearly combining the results. The second order TVD RK scheme, also known as the midpoint rule, advances the solution by two Euler steps to get ϕ^{n+2} at time instance $t^n + 2\Delta t$. The solution at ϕ^{n+1} is the average:

$$\phi^{n+1} = \frac{1}{2}\phi^n + \frac{1}{2}\phi^{n+2} \tag{7}$$

The third order TVD RK scheme proceeds by advancing the solution by Euler steps to ϕ^{n+2}, followed by a linear combination to get $\phi^{n+\frac{1}{2}}$. This solution is advanced to $\phi^{n+\frac{3}{2}}$, followed by a final linear combination to get ϕ^{n+1}:

$$\phi^{n+\frac{1}{2}} = \frac{3}{4}\phi^n + \frac{1}{4}\phi^{n+2} \tag{8a}$$

$$\phi^{n+1} = \frac{1}{3}\phi^n + \frac{2}{3}\phi^{n+\frac{3}{2}} \tag{8b}$$

2.2 Spatial discretization

Since the spatial discretization strongly depends on the behavior of the particular PDE used we will study two different versions of Equation 4, giving rise to two fundamental types of PDEs, *hyperbolic* type and *parabolic* type.

Hyperbolic advection

To introduce *hyperbolic advection*, two versions of Equation 4 can be stated as:

$$\frac{\partial \phi}{\partial t} = -\boldsymbol{V} \cdot \nabla\phi \tag{9a}$$

$$= -F\,|\nabla\phi| \tag{9b}$$

This corresponds to *advecting* (transporting) the interface in a vector field $\boldsymbol{V}$ (Equation 9a) or in the direction of the surface normal (Equation 9b). This type of equation could be used for erosion or dilation of a surface and is often found in computational physics describing a *flow* of information in a certain direction. Imagine a *numerical wave*, propagating information in a flow field as directed by Equation 9. The Courant-Friedrichs-Lewy (CFL) stability condition [2] states that only the sample points *behind*, or *up-wind* to, the wave should be used in the discretization. This makes intuitive sense, since sample points in front of the wave have not been "touched" by the flow field. Thus, they do not contain any reliable information needed to evaluate the partial derivatives. Considering Equation 9a, this results in the finite difference (FD) approximation given by:

$$\frac{\partial \phi}{\partial x} \approx \begin{cases} \phi_x^+ = \left(\phi_{i+1,j,k} - \phi_{i,j,k}\right)/\Delta x & \text{if } V_x < 0 \\ \phi_x^- = \left(\phi_{i,j,k} - \phi_{i-1,j,k}\right)/\Delta x & \text{if } V_x > 0 \end{cases} \tag{10}$$

This introduces some new notation. Without loss of generality, we assume a 3D level set function ϕ. Then, $\phi_{i,j,k}$ is the discrete sample point at position (i, j, k) in the grid, while Δx is the grid spacing along the x axis. The *stencil* defines the grid points which are used by the operator. In Equation 10, the stencil is two grid points wide, meaning the operator uses information from two sample points. For Equation 9b the direction of the flow is not known explicitly. For this we can use Godunov's method [10] to evaluate the partial derivatives:

$$\left(\frac{\partial \phi}{\partial x}\right)^2 \approx \begin{cases} \max\left[\max(\phi_x^-, 0)^2, \min(\phi_x^+, 0)^2\right] & F > 0 \\ \max\left[\min(\phi_x^-, 0)^2, \max(\phi_x^+, 0)^2\right] & F < 0 \end{cases} \tag{11}$$

These discretizations result in a *first order accurate* approximation of the partial derivatives. For higher order approximations, the *essentially non-oscillatory* (ENO) [8] or *weighted* ENO (WENO) [4] polynomial interpolation techniques can be used. The basic idea is to provide better approximations to ϕ_x^- and ϕ_x^+ by the smoothest combination of grid points in an up to six point wide stencil. These methods give a 3rd to 5th order accurate approximation depending on the smoothness around a grid point (i, j, k). For more information on the actual implementation, we refer the reader to [7].

When using an explicit method for the temporal discretization, the CFL condition defines the stability requirement for the time step Δt. Since information is propagated one grid point each time step the numerical wave speed along the x axis is $\Delta x/\Delta t$. This speed needs to be at least as fast as the physical wave speed given by $\boldsymbol{V}$ or F in Equation 9 to ensure stability. The time step of a three-dimensional level set function is given by:

$$\begin{aligned}\Delta t &< \min\left\{\tfrac{\Delta x}{|V_x|}, \tfrac{\Delta y}{|V_y|}, \tfrac{\Delta z}{|V_z|}\right\}, \\ \Delta t &< \frac{\min\{\Delta x, \Delta y, \Delta z\}}{|F|}\end{aligned} \tag{12}$$

Parabolic diffusion

To introduce *parabolic diffusion*, a version of Equation 4 can be stated as:

$$\frac{\partial \phi}{\partial t} = \alpha\kappa\,|\nabla\phi| \tag{13}$$

where α is a scaling parameter and κ is curvature. This equation is equivalent to the heat- or diffusion equation. It can be used to minimize (smooth) deformations of the level set surface [5]. This type of PDE is very different from the hyperbolic type in that the information flow has no particular direction. The solution at a point at a particular time depends on any other point in the domain at the previous time step. We need to use a second-order accurate central difference scheme for space:

$$\frac{\partial \phi}{\partial x} \approx \phi_x^{\pm} = \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2\Delta x} \tag{14}$$

The curvature κ is described by the divergence of the normal $\kappa = \nabla \cdot \boldsymbol{n}$ in 2D [5]. In 3D the mean curvature is the average of the two principal curvatures (minimium and maximium) found by evaluating the curvature of all curves passing through the point of interest on the surface.

$$\kappa = \frac{\kappa_1 + \kappa_2}{2} \tag{15}$$

Resolving the gradient operator in the orthonormal frame of the principal directions $\vec{e}_1, \vec{e}_2$ in the tangent plane and the normal vector $\vec{n}$ it is possible to write the principal curvatures using the gradient operator. Then, after some simplification, the mean curvature can be expressed as half the divergence of the level-set normal ($\hat{\boldsymbol{n}} = \nabla\phi / |\nabla\phi|$)

$$\begin{aligned}\kappa &= \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} \\ &= \frac{\phi_x^2(\phi_{yy} + \phi_{zz}) - 2\phi_y\phi_z\phi_{yz}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}} \\ &+ \frac{\phi_y^2(\phi_{xx} + \phi_{zz}) - 2\phi_x\phi_z\phi_{xz}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}} \\ &+ \frac{\phi_z^2(\phi_{xx} + \phi_{yy}) - 2\phi_x\phi_y\phi_{xy}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}}\end{aligned} \tag{16}$$

where $\phi_x = \partial\phi/\partial x$ and $\phi_{xy} = \partial^2\phi/\partial x\partial y$. This expression can be discretized using the following second order central difference scheme:

$$\frac{\partial^2 \phi}{\partial^2 x} \approx \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{\Delta x^2} \tag{17a}$$

$$\frac{\partial^2 \phi}{\partial x \partial y} \approx \frac{\phi_{i+1,j+1,k} - \phi_{i+1,j-1,k} + \phi_{i-1,j-1,k} - \phi_{i-1,j+1,k}}{4\Delta x\Delta y} \tag{17b}$$

When considering stability constraints for the time step Δt, we note that the information travels at an infinite speed. Thus, the CFL stability condition in Equation 12 is not applicable for parabolic PDE's. For a two-dimensional level set function, the stability constraint can be derived by a von Neumann stability analysis [13]. From the following expression we see that the timestep is only limited by the scaling parameter α and the grid cell spacing Δx.

$$\begin{aligned}&\Delta t < \left(\tfrac{2\alpha}{\Delta x^2} + \tfrac{2\alpha}{\Delta y^2} + \tfrac{2\alpha}{\Delta z^2}\right)^{-1}, \\ &\{\Delta x = \Delta y = \Delta z\} \Rightarrow \Delta t < \tfrac{\Delta x^2}{6\alpha}\end{aligned} \tag{18}$$

2.3 The signed distance property and reinitialization

We previously stated that ϕ needs to be continuous with well-defined gradients, in order to solve Equation 4. However, because of the spatial discretization, this requirement can be relaxed. More specifically, we require ϕ to be *Lipschitz continuous*, satisfying the following condition:

$$|\phi(\boldsymbol{x_0}) - \phi(\boldsymbol{x_1})| \leq C|\boldsymbol{x_0} - \boldsymbol{x_1}| \tag{19}$$

where $C \geq 0$. This means that the gradients of ϕ can be discontinuous, although the *rate-of-change* of ϕ must be bounded by the finite Lipschitz constant C. This constant is affected by the numerical schemes used to solve the discretized equations. In order to assure stability, $C \approx 1$ meaning that ϕ has to approximately satisfy the *Eikonal equation*:

$$|\nabla\phi| = 1 \tag{20}$$

As you walk in a direction orthogonal to the level set interface, the distance increases by one unit each step. Combined with Equation 2, this means that ϕ is a *signed distance function*. This constraint can be motivated by considering Equation 9b and Equation 13. If $|\nabla\phi|$ would drift away from one, the speed of the *information flow* would increase, implying that we need to evolve the PDE with smaller time steps. However, it is often more efficient to keep the time step fixed while making sure ϕ approximately satisfies the Eikonal equation.

This process is called *reinitialization* and should be performed more or less frequently depending on how sensitive the discretization of the particular PDE is to numerical fluctuations. There are a number of ways of doing this, the first of which solves the *reinitialization equation* to steady state [14]:

$$\frac{\partial\phi}{\partial t} = S(\phi)(1 - |\nabla\phi|) \tag{21}$$

where $S(\phi)$ is the *sign* of ϕ, with a smooth numerical approximation given by [9]:

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi|^2 \, \Delta x^2}} \tag{22}$$

At steady state $\partial\phi/\partial t = 0$, implying $|\nabla\phi| = 1$. Since Equation 21 is an hyperbolic type PDE, $\nabla\phi$ needs to be discretized using an up-wind scheme as discussed in Section 2.2. This approach has a runtime complexity of $O(N^2)$, where N is the number of grid points.

Another way of maintaining the signed distance property is to solve the Eikonal equation directly, using for example the *fast marching method* as proposed by Sethian in [11]. The idea behind this method can be described by sequentially adding "layers" to the interface. It walks around the interface, computing the *time-of-arrival* at each grid point immediately adjacent to the current layer. When a layer is completed, a second layer is added with a later time-of-arrival. Using a heap data structure, this algorithm has a runtime complexity of $O(N \log N)$. A faster, and simpler, algorithm called *fast sweeping* was later proposed by Zhao [15]. This is based on a number of sequential sweeps through the grid, following the characteristics of the Eikonal equation in a certain direction in each sweep. The complexity is optimally $O(N)$ and in d dimensions, 2^d sweeps are necessary. However, this method is not easily adapted to narrow-band schemes, which we will present next.

2.4 Narrow band schemes

The level set method solves the level set equation in the entire domain of ϕ. However, for most applications, the method is used to track the motion of a particular interface, or level set, of ϕ with isovalue zero. Therefore, the computations can be restricted to a neighborhood around the zero crossing, with the same results as solving the equations on the entire domain. Using such a *narrow band* scheme, the computational cost is proportional to the size of the interface and reduces drastically the needed computing time, especially when propagating surfaces in three dimensions or higher. This idea was first introduced by Adalsteinsson and Sethian in [1] and further improved by Peng *et al.* in [9]. We will describe the scheme by Peng *et al.* further.

We define two narrow band *tubes*, T_β and T_γ as:

$$T_\beta = \{\boldsymbol{x} \in \Re^d : |\phi(\boldsymbol{x})| < \beta\} \tag{23a}$$

$$T_\gamma = \{\boldsymbol{x} \in \Re^d : |\phi(\boldsymbol{x})| < \gamma\} \tag{23b}$$

where $0 < \beta < \gamma$. This is depicted in Figure 1. The level set equations will now be solved only within these tubes. However, to avoid numerical oscillations at the tube boundary, we introduce a cut-off

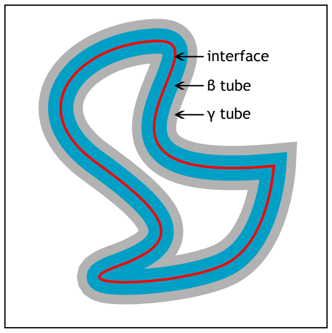

Figure 1: Narrow band displaying the interface, the β tube and the γ tube by Peng *et al.* [9]

function:

$$c(\phi) = \begin{cases} 1 & \text{if } |\phi| \leq \beta \\ \frac{(2|\phi|+\gamma-3\beta)(|\phi|-\gamma)^2}{(\gamma-\beta)^3} & \text{if } \beta < |\phi| \leq \gamma \\ 0 & \text{if } |\phi| > \gamma \end{cases} \tag{24}$$

that changes Equation 4 into:

$$\frac{\partial\phi}{\partial t} = -c(\phi)\ \nabla\phi \cdot \frac{d\boldsymbol{x}}{dt} \tag{25a}$$

$$= -c(\phi)\ |\nabla\phi|\, F(\boldsymbol{x}, \boldsymbol{n}, \phi, ...) \tag{25b}$$

Thus, the level set equation is solved exactly only within T_β. In order to rebuild the tubes, we note that the numerical scheme can only move the interface at a maximum of one grid point each time step. Assuming a grid spacing of Δx, we use this to define a tube $N = \{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x} + \boldsymbol{x_0}) < \gamma, |\boldsymbol{x_0}| \leq \Delta x\}$ in which we perform the reinitialization described in the previous section. In other words, we reinitialize in a tube corresponding to T_γ dilated by one grid point. After the reinitialization, we have the correct signed distance function in N which is required to contain all $\{\boldsymbol{x} \in \Re^d : \phi(\boldsymbol{x}) < \gamma\}$. Using N, we can now rebuild the tubes T_β and T_γ. The actual choice of β and γ depends on the stencil used for the spatial discretization. For a first order scheme, $\beta = 2\Delta x, \gamma = 3\Delta x$ could prove sufficient, while the 3rd to 5th order WENO scheme could require $\beta = 4\Delta x, \gamma = 6\Delta x$.

This approach is *computationally efficient* since it only computes the level set equations and reinitializes in a narrow band around the interface of principal interest. The implementation is simple and the rebuilding of the tubes can be optimized to give an overall runtime of $O(N)$ where N is the size of the interface. However, it is not *memory efficient* since it stores the full level set function. Moreover, since the narrow band is stored scattered in memory, the CPU cache can not be efficiently used.

An approach which is both computationally and memory efficient is the *dynamic tubular grid* (DT-grid) proposed by Nielsen and Museth [6]. It is based on the method by Peng *et al.* but only stores the actual narrow band in memory. Not only does this reduce storage requirements, but it also very efficiently uses the CPU cache for sequential memory access. Moreover, since the narrow band can be seen as “floating” in free space, it is not constrained within any grid boundaries and can expand freely in an *out-of-the-box* like behavior.

3 Common level set operators

This section will present some common level set equations, describe how to solve them and what geometrical operations they represent.

3.1 Advection

In many applications we want to *advect* (“move”) the level set surface in a vector field. The vector field can describe physical events such as velocity (winds, fluids) or arbitrary deformations used in modeling (twists, skews). The level set equation used for advecting the surface given a velocity field (Equation 9a) is of hyperbolic type and was presented in Section 2.2:

$$\frac{\partial\phi}{\partial t} = -\boldsymbol{V} \cdot \nabla\phi$$

As discussed in Section 2.2, this equation has to be discretized by an upwind scheme, which means that the finite difference is evaluated on points that have been “touched” by the numerical wave. In other words, the direction of $\boldsymbol{V}$ in each point plays an important role when computing the gradient of ϕ. A first order upwind scheme was presented in Equation 10, and reads:

$$\frac{\partial \phi}{\partial x} \approx \begin{cases} \phi_x^+ = \left(\phi_{i+1,j,k} - \phi_{i,j,k}\right) / \Delta x & \text{if } V_x < 0 \\ \phi_x^- = \left(\phi_{i,j,k} - \phi_{i-1,j,k}\right) / \Delta x & \text{if } V_x > 0 \end{cases}$$

Note that if $\boldsymbol{V}$ travel toward the negative x direction ($V_x < 0$) we use values from the positive side (ϕ_x^+) since these points have been "visited" by the flow, and vice versa for positive directions. Given a level-set function representing the next time step, ϕ^{new}, and a vector field $\boldsymbol{V}$ that drives the motion, we can now start to move the interface. For each grid point (i, j, k) in the level-set:

1. Evaluate the vector field, $\boldsymbol{V}_{i,j,k}$
2. Compute the gradient, $\nabla \phi_{i,j,k}$, using an upwind scheme
3. Calculate the rate of change ($\frac{\partial \phi_{i,j,k}}{\partial t}$) as the scalar product between the vector field and the gradient (Equation 9a)
4. Finally update $\phi^{new}_{i,j,k}$, using for example forward Euler time integration (Equation 6)

3.2 Motion in the normal direction

For motion in the normal direction (Equation 9b) you follow the same steps as described in the previous section, except for the evaluation of the speed (F) and the calculation of the gradient. Now we are interested in the *magnitude* of the gradient and we can use Godunov's method from Equation 11. The rate of change ($\frac{\partial \phi_{i,j,k}}{\partial t}$) is now described by the product of the speed function (speed in the normal direction) denoted by $F(\boldsymbol{x})$, and the magnitude of the gradient, $|\nabla \phi|$.

1. Evaluate the speed function, $F(i, j, k) = F_{i,j,k}$
2. Compute the magnitude of the gradient, $\left|\nabla \phi_{i,j,k}\right|$, using the Godunov scheme
3. Calculate the rate of change ($\frac{\partial \phi_{i,j,k}}{\partial t}$) as the product between the speed function and the magnitude of the gradient (Equation 9b)
4. Finally update $\phi^{new}_{i,j,k}$, using for example forward Euler time integration (Equation 6)

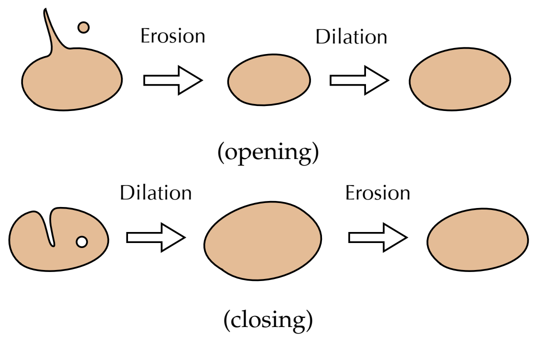

Figure 2: Morphological operators.

3.2.1 Dilation and erosion

The speed function $F(\boldsymbol{x})$ which gives the speed in the normal direction can be different for each point on the surface. However, if we set the speed function to the same value for all points we can achieve a dilation (advection outwards) or erosion (advection inwards) of the surface, depending on the sign of $F(\boldsymbol{x})$. Combining erosion and dilation we can achieve morphological processing using level set methods. Morphological opening is an operation that removes small spikes on the surface or small free floating shells and can be implemented by erosion followed by dilation for the same amount of time. Morphological closing - which removes small inside voids or gaps - is done the other way around, as dilation followed by erosion, see Figure 2.

Consider Equation 9b, which moves the interface in the normal direction. Using this we can dilate the interface 1 unit distance by setting the speed function ($F(\boldsymbol{x})$) to a positive value and advect for 1 time unit. Inversely we can achieve erosion by setting the speed to a negative value. Dilation and erosion are scalar speed functions and the resulting PDE can be solved using the scheme in Section 3.2

above with the following speed definitions

$$\begin{array}{lll} \text{Dilation :} & F(\boldsymbol{x}) = c_1, & c_1 > 0 \\ \text{Erosion :} & F(\boldsymbol{x}) = c_2, & c_2 < 0 \end{array} \qquad (26)$$

where c_1 and c_2 are constants.

3.3 Smoothing

As briefly mentioned in Section 2.2, the parabolic diffusion equation (Equation 13) can be used for geometrical smoothing of a surface. The equation reads:

$$\frac{\partial \phi}{\partial t} = \alpha\kappa\,|\nabla\phi|$$

where κ is curvature and α is a scaling parameter. Unlike the hyperbolic equations used for advection, the information flow in this parabolic diffusion equation has no particular direction. Therefore, we use central differencing schemes to evaluate the differentials in this equation. Recall the derived equation for the curvature (Equation 16) and the associated finite differences (Equation 17):

$$\begin{aligned} \kappa &= \frac{1}{2}\nabla\cdot\frac{\nabla\phi}{|\nabla\phi|} \\ &= \frac{\phi_x^2(\phi_{yy}+\phi_{zz}) - 2\phi_y\phi_z\phi_{yz}}{2(\phi_x^2+\phi_y^2+\phi_z^2)^{3/2}} \\ &+ \frac{\phi_y^2(\phi_{xx}+\phi_{zz}) - 2\phi_x\phi_z\phi_{xz}}{2(\phi_x^2+\phi_y^2+\phi_z^2)^{3/2}} \\ &+ \frac{\phi_z^2(\phi_{xx}+\phi_{yy}) - 2\phi_x\phi_y\phi_{xy}}{2(\phi_x^2+\phi_y^2+\phi_z^2)^{3/2}} \end{aligned}$$

$$\begin{aligned} \frac{\partial^2\phi}{\partial^2 x} &\approx \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{\Delta x^2} \\ \frac{\partial^2\phi}{\partial x\partial y} &\approx \frac{\phi_{i+1,j+1,k} - \phi_{i+1,j-1,k} + \phi_{i-1,j-1,k} - \phi_{i-1,j+1,k}}{4\Delta x\Delta y} \end{aligned}$$

Propagating the surface according to the diffusion equation is done in the same manner as for the advection. For each grid point (i, j, k) in the level-set:

1. Compute the curvature, $\kappa_{i,j,k}$, according to Equation 16 and Equation 17
2. Compute the norm of the gradient, $|\nabla\phi|_{i,j,k}$, using a central difference scheme (Equation 14)
3. Calculate the rate of change ($\frac{\partial\phi_{i,j,k}}{\partial t}$) as the product between the scaling parameter, the curvature and the norm of the gradient of the level set (Equation 13)
4. Finally update $\phi^{new}_{i,j,k}$, using for example forward Euler time integration (Equation 6)

By solving this equation, high curvature "spikes" in the surface will disappear, and the effect will be a smoother surface. Again, note the difference in solving this diffusion equation (parabolic) and an advection equation (hyperbolic) numerically. We use central differencing for the first, but upwind schemes for the latter.

4 Assignments

4.1 Grading

The assignments marked with (3) are mandatory to pass the lab, grade 3. Completing assignment (4) gives you the grade 4, while completing (4,5a) or (4,5b) gives you a grade 5.

Implement differential calculations (3)

In the text we have investigated many different differentials, $\phi_x^+, \phi_x^-, \phi_x^\pm, \phi_{xx}^\pm, \phi_{xy}^\pm$. Now it's time to implement them in code. The functions in the class `LevelSet` are called `Diff + {X,Y,Z} + {p,m,pm}` where {X,Y,Z} denotes dimension along which the derivative should be taken and p=plus (e.g. ϕ_x^+), m=minus (e.g. ϕ_x^-), and pm=central difference (e.g. $\phi_x^\pm$). There are in total 15 function implementations to write. When working with this (and later labs) keep in mind that we use the convention of representing space coordinates (usually world space) with (x, y, z) and grid coordinates (indices) by (i, j, k). Recall that the level set function is sampled in a regular cubic grid (the `mGrid` member variable), which means that only values at grid points are stored. Arbitrary points in the level set function have to be interpolated. The distance between grid points Δx, is stored in the variable `mDx`.

You can verify that your code works by looking at cut planes. Add an implicit surface to the scene and convert it to a levelset object using the menu option "Convert implicit to levelset". Remember that what we call a "levelset" is simply an implicit surface with a bunch of math for doing deformations. The conversion simply samples the implicit function and stores the values in a grid which we can edit. Then, select the levelset and add a "Vector cut plane" from the menu. The vector cut plane gets the gradients from the `LevelSet::GetGradient()` function, which initially just uses `Implicit::GetGradient()` which you might have implemented in the previous lab. You can change `LevelSet::GetGradient()` and compute the gradient using each of the first order differentials in turn and verify that the normals look correct. Remember to convert from space coordinates (x, y, z) to grid coordinates (i, j, k) when accessing the grid, see `LevelSet::GetValue()` for an example. What is the gradient in the center of an implicit sphere using one-sided difference schemes?

The signed distance property (3)

Add a levelset object like in the previous task and add a scalar cut plane using the corresponding menu option. This cut plane maps the scalar values of the levelset function (or any implicit function) to colors using the currently selected colormap. For this assignment the "Iso contour" colormap is useful. It maps green to red in a wrap-around fashion so that each green-red cycle represents a change in the levelset function by 0.1. This visualizes *all* level sets of the "levelset" function - not only the zero level set which is rendered as a surface (use the opacity slider to reveal the inside of the surface mesh). Run the reinitialize operator on the level set and see what happens. Note that the lines of the grid plane in the GUI has a distance of 0.1. Study both the scalar cut plane and the vector cut plane. How should the gradients and the level sets look when the levelset function is properly reinitialized? What exactly does reinitialization mean and why is it necessary?

Implement erosion and dilation (3)

Implement an operator which does erosion and dilation in `OperatorDilateErode.h` (study `OperatorReinitialize.h` to see a working example of an operator). You need to compute a stable timestep in `ComputeTimestep()` by using Equation 12. The speed function for erosion and dilation is explained in Section 3.2.1 which should be implemented in `Evaluate()`. This function returns the rate of change ($\frac{\partial \phi}{\partial t}$) for a grid point (i, j, k). Remember to use upwinding by the Godunov scheme (`LevelSetOperator::Godunov`) to calculate the squares of the gradients. Verify your implementation by adding a levelset object and do dilation and erosion. Be sure to study the `Propagate()` function to make sure you understand how the propagation loop is executed.

Implement advection from an external vector field (3)

Implement an operator that does advection in `OperatorAdvect.h`. The time step for stability is given in Equation 12. The ideas behind implementing advection is summarized in Section 3.1. As in the previous assignment, the functions `ComputeTimestep()` and `Evaluate()` require implementation. In `Evaluate()`, you sample an external velocity field which determines the actual advection. This sampling should be done in world coordinates, so remember to convert your grid coordinates (i, j, k) to world coordinates (x, y, z). Test your code by running advection on a levelset object (try the quadric plane). Currently `FrameMain::LevelsetAdvect()` uses the `VortexVectorField`. Feel free to experiment with other types of fields, or to modify the `VortexVectorField`.

Implement mean curvature flow (4)

Implement an operator which does mean curvature flow in `OperatorMeanCurvatureFlow.h`. This can be used for smoothing an implicit surface. As for the previous operators, carefully choose a stable time step. The implementation is outlined in Section 3.3. Add an interesting model (not a sphere) and try out your operator. You can load a volume in `Data/Vols` for more complex models.

Implement morphing (5a)

Implement an operator which does morphing in `OperatorMorph.h`. There are several ways of doing this, but the simplest approach is to consider the target model as a speed function (speed in normal direction) used to deform the morphing model. Select two implicits in the GUI (where at least one is a levelset) and execute morph to verify your implementation. Note: There is currently a bug that shows up when trying to morph a loaded volume (such as Horse_small.vol) into another loaded volume. To get around this, select your loaded volume objects and do "Convert implicit to level set". This will simply resample the level set function, so the object won't change.

Inspect the narrow band optimization (5b)

How do the operators currently loop over the grid in `LevelSetOperator::IntegrateEuler()`? Study how the narrow band structure is implemented in `LevelSetGrid` with a simple bit mask and iterators. By default, the narrow band contains the entire bounding box of the level set, so all computations are fairly slow. You can enable narrow band computations by specifying the width of the band and hit the "Enable" button in the GUI. What happens to the level set function when enabling the narrow band (study the scalar and vector cut planes)? What happens if you vary the width of the band? Time your dilate/erode and advect operators with and without the narrow band optimization (hint: use class `Util/Stopwatch`). Increase the resolution of the levelset by changing the first parameter to the levelset constructor when adding levelset objects in `FrameMain::ConvertToLevelset()` to note the asymptotic complexity with and without narrow band.

5 Acknowledgements

The lab scripts and the labs were originally developed by Gunnar Läthén, Ola Nilsson, Andreas Söderström and Stefan Lindholm.

References

[1] David Adalsteinsson and James A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[2] R. Courant, K. Friedrichs, and H. Lewy. Uber die partiellen differenzengleichungen der mathematischen physik. *Mathematische Annalen*, 100:32–74, 1928.

[3] Ron Goldman. Curvature formulas for implicit curves and surfaces. *Computer Aided Geometric Design*, 22(7):632 – 658, 2005.

[4] Xu-Dong Liu, Stanley Osher, and Tony Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115:200–212, 1994.

[5] Ken Museth, David E. Breen, Ross T. Whitaker, Sean Mauch, and David Johnson. Algorithms for interactive editing of level set models. *The International Journal of Eurographics Assoc.: Computer Graphics Forum*, 24(4):821–841, December 2005.

[6] Michael B. Nielsen and Ken Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, pages 1–39, February 2006.

[7] Stanley Osher and Ronald Fedkiw. *Level Set and Dynamic Implicit Surfaces*. Springer-Verlag New York Inc., 2003.

[8] Stanley Osher and Chi-Wang Shu. High-order essentially nonsoscillatory schemes for hamilton-jacobi equations. *SIAM Journal on Numerical Analysis*, 28(4):907–922, 1991.

[9] Danping Peng, Barry Merriman, Stanley Osher, Hong-Kai Zhao, and Myungjoo Kang. A pde-based fast local level set method. *Journal of Computational Physics*, 155(2):410–438, 1999.

[10] Elisabeth Rouy and Agnes Tourin. A viscosity solutions approach to shape-from-shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, June 1992.

[11] J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of*

the National Academy of Science, volume 93, pages 1591–1595, 1996.

[12] Chi-Wang Shu and Stanley Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, August 1988.

[13] John C. Strikwerda. *Finite difference schemes and partial differential equations*. Wadsworth Publ. Co., Belmont, CA, USA, 1989.

[14] Mark Sussman, Peter Smereka, and Stanley Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, September 1994.

[15] Hong-Kai Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation*, (74):603–627, 2005.

A Deriving the curvature expression

The curvature expression from section Section 2.2 looks a bit scary at first. Let's go through it step by step. We have stated that our interest is in the mean curvature

$$\kappa = \frac{\kappa_1 + \kappa_2}{2}$$

And proposed that it can be found as half the divergence of the normal.

$$\kappa = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} \tag{28}$$

This is the expression we want to simplify. In the analytical case it is simple, we know that our scalar function ϕ is a signed distance function and thus the magnitude of the gradient is one.

$$|\nabla\phi| \equiv 1$$

Then the mean curvature can be found as

$$\begin{aligned} \kappa &= \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{1} \\ &= \frac{1}{2}\nabla \cdot \nabla\phi \\ &= \frac{1}{2}\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right) \\ &= \frac{1}{2}\left(\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}\right) \end{aligned} \tag{29}$$

The double application of the *del* operator (∇) to a scalarfield is often called the laplace operator and sometimes also denoted ∇^2. When ∇ is applied once we get a vector field (the gradient field, $\nabla\phi$), and when applied again we take the divergence of this vector field, which is a scalar field.

When dealing with discrete mathematics it is important to sometimes take certain "truths" with a bit of scepticism. The function ϕ is only sampled on a discrete grid, thus we can expect to have (at least) round-off errors. The error analysis also tells us that no matter what the accuracy of our scheme, it will always have a trailing error term[1]. That means that as long as our grid is not refined to infinitely small grid cells there will still be an error term that is proportional to the grid size and the scheme we are using. So it is a lot safer to assume that the signed distance property is only approximately true

$$|\nabla\phi| \approx 1$$

This, in turn, means that the derivative above (Equation 29) gets a lot more complicated since we have to use the complete definition

$$\kappa = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}$$

To shorten the notation we will now use subscript notation for the partial differentials when applied to a function, $\frac{\partial\phi}{\partial x} \equiv \phi_x$, and similiarly $\frac{\partial^2\phi}{\partial x^2} \equiv \phi_{xx}$.

Remember that $\nabla\phi$ is a vector function in three variables, in short (ϕ_x, ϕ_y, ϕ_z), also note that the magnitude of the gradient is a scalar function.

$$|\nabla\phi| = \sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}$$

This gives us

$$\kappa = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}} \tag{30}$$

[1]Also called truncation error, often denoted by $O()$.

To differentiate this we will make use of quite a few differentiation rules from calculus. We will need

- Divergence operator: $\nabla \cdot f(x,y,z) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} + \frac{\partial f}{\partial z}$. The notation is resembling the vector dot product. Note that it is different differentials that are applied to the different vector components and that the result is a scalar.
- The qoutient rule: $D\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}$
- Square root derivative: $D\left(\sqrt{x}\right) = \frac{1}{2\sqrt{x}}$
- The chain rule: $D\left(f(g(x))\right) = f'(g(x))g'(x)$
- Equality of mixed partial derivatives: $\frac{\partial^2 \phi}{\partial xy} = \frac{\partial^2 \phi}{\partial yx}$

Of course all our derivatives are partial so we can't denote them with the $'$ symbol. Instead we write them either using the subscript notation (ϕ_x) or full partial notation ($\partial\phi/\partial x$). To get a better overview we will split our expression, Equation 30, into its three vector components and treat each of them seperately

$$\begin{aligned}
\kappa &= \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} \\
&= \frac{1}{2}\nabla \cdot \frac{(\phi_x, \phi_y, \phi_z)}{|\nabla\phi|} \\
&= \frac{1}{2}\nabla \cdot \left(\frac{\phi_x}{|\nabla\phi|}, \frac{\phi_y}{|\nabla\phi|}, \frac{\phi_z}{|\nabla\phi|}\right) \\
&= \frac{1}{2}\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot \left(\frac{\phi_x}{|\nabla\phi|}, \frac{\phi_y}{|\nabla\phi|}, \frac{\phi_z}{|\nabla\phi|}\right) \\
&= \frac{1}{2}\left(\frac{\partial}{\partial x}\left(\frac{\phi_x}{|\nabla\phi|}\right) + \frac{\partial}{\partial y}\left(\frac{\phi_y}{|\nabla\phi|}\right) + \frac{\partial}{\partial z}\left(\frac{\phi_z}{|\nabla\phi|}\right)\right)
\end{aligned} \tag{31}$$

Let us start with the x component, and calculate $\frac{\partial}{\partial x}\left(\phi_x / |\nabla\phi|\right)$. We first apply the qoutient rule and get

$$\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\phi_x}{|\nabla\phi|}\right) &= \frac{\frac{\partial}{\partial x}(\phi_x)\,|\nabla\phi| - \phi_x \frac{\partial}{\partial x}(|\nabla\phi|)}{|\nabla\phi|^2} \\
&= \frac{\frac{\partial}{\partial x}(\phi_x)\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2} - \phi_x \frac{\partial}{\partial x}\left(\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}\right)}{\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}^2}
\end{aligned} \tag{32}$$

The expression $\frac{\partial}{\partial x}(\phi_x)$ is easily found as the second derivative in x, ϕ_{xx}. The square root is a bit more complicated, since it involves succesive applications of the chain rule. Lets look at it seperately

$$\begin{aligned}
\frac{\partial}{\partial x}\left(\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}\right) &= \frac{1}{2\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}}\frac{\partial}{\partial x}\left(\phi_x^2 + \phi_y^2 + \phi_z^2\right) \\
&= \frac{1}{2\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}}\left(2\phi_x\frac{\partial}{\partial x}(\phi_x) + 2\phi_y\frac{\partial}{\partial x}(\phi_y) + 2\phi_z\frac{\partial}{\partial x}(\phi_z)\right) \\
&= \frac{1}{2\sqrt{\phi_x^2 + \phi_y^2 + \phi_z^2}}\left(2\phi_x\phi_{xx} + 2\phi_y\phi_{xy} + 2\phi_z\phi_{xz}\right)
\end{aligned} \tag{33}$$

For each of the lines in the Equation 33 above we have expanded the inner derivatives using the chain rule. Shortening the expresion again with $|\nabla\phi|$ for the square roots gives us, after back substitution into Equation 32, the following

$$\frac{\partial}{\partial x}\left(\frac{\phi_x}{|\nabla\phi|}\right) = \frac{\phi_{xx}\,|\nabla\phi| - \phi_x \frac{1}{\cancel{2}\,|\nabla\phi|}\cancel{2}\left(\phi_x\phi_{xx} + \phi_y\phi_{xy} + \phi_z\phi_{xz}\right)}{|\nabla\phi|^2} \tag{34}$$

If we multiply the left hand side of the numerator with $\frac{|\nabla\phi|}{|\nabla\phi|}$ then it is possible to write the expression as one fraction

$$\begin{aligned}
\frac{\partial}{\partial x}\left(\frac{\phi_x}{|\nabla\phi|}\right) &= \frac{\frac{|\nabla\phi|}{|\nabla\phi|}\phi_{xx}\,|\nabla\phi| - \phi_x \frac{\left(\phi_x\phi_{xx} + \phi_y\phi_{xy} + \phi_z\phi_{xz}\right)}{|\nabla\phi|}}{|\nabla\phi|^2} \\
&= \frac{\phi_{xx}\,|\nabla\phi|^2 - \phi_x\left(\phi_x\phi_{xx} + \phi_y\phi_{xy} + \phi_z\phi_{xz}\right)}{|\nabla\phi|^3} \\
&= \frac{\phi_{xx}(\cancel{\phi_x^2} + \phi_y^2 + \phi_z^2) - \phi_x\left(\cancel{\phi_x\phi_{xx}} + \phi_y\phi_{xy} + \phi_z\phi_{xz}\right)}{|\nabla\phi|^3} \\
&= \frac{\phi_{xx}(\phi_y^2 + \phi_z^2) - \phi_x\phi_y\phi_{xy} - \phi_x\phi_z\phi_{xz}}{|\nabla\phi|^3}
\end{aligned} \tag{35}$$

Following the exact same procedure for the y and z components we get

$$\begin{aligned}
\kappa &= \frac{1}{2}\left(\frac{\partial}{\partial x}\left(\frac{\phi_x}{|\nabla\phi|}\right) + \frac{\partial}{\partial y}\left(\frac{\phi_y}{|\nabla\phi|}\right) + \frac{\partial}{\partial z}\left(\frac{\phi_z}{|\nabla\phi|}\right)\right) \\
&= \frac{1}{2}\Bigg(\frac{\phi_{xx}(\phi_y^2 + \phi_z^2) - \phi_x\phi_y\phi_{xy} - \phi_x\phi_z\phi_{xz}}{|\nabla\phi|^3} + \\
&\quad \frac{\phi_{yy}(\phi_x^2 + \phi_z^2) - \phi_x\phi_y\phi_{xy} - \phi_y\phi_z\phi_{yz}}{|\nabla\phi|^3} + \\
&\quad \frac{\phi_{zz}(\phi_x^2 + \phi_y^2) - \phi_x\phi_z\phi_{xz} - \phi_y\phi_z\phi_{yz}}{|\nabla\phi|^3}\Bigg) \quad \square
\end{aligned} \tag{36}$$

It is left to the reader to verify that this is the same as Equation 16.