INSTITUTE FOR MICROELECTRONICS

360.246 Simulation of Semiconductor Device Fabrication Assignment 1 - Surface Representations

Submission is due on Monday, April 28th 2019, 10am (submit to **simfab-submit@iue.tuwien.ac.at**)

- Use C++ 11 or Python 3.6+ for all implementations
- Binaries/Scripts must be callable with the command line parameters specified below
- Submit your report as a single PDF file. There is no set length required for the report.
- Submit everything (Sources, Binaries, PDF) together as one zip-file to **simfab-submit@iue.tuwien.ac.at**

General information

- You may use any compiler of your liking, however we recommend additional warning output. When using GCC, at least the following flags should be used: `-std=c++11 -Wall -Wextra -pedantic`
- Make sure you structure your programs in a readable fashion. It will make you more efficient and help us identify possible errors easily, i.e.: save you marks if there are problems.

1 Dense Grid Signed Distance Function

The signed distance function $\Phi(\vec{x})$ is a method by which a surface can be implicitly defined. It gives the shortest distance of the point $\vec{x}$ to the surface it describes. The sign of the distance relates to the point being above or below the surface, which translates to outside or inside for closed surfaces. Although there is no strict convention, most simulation frameworks use negative values to describe points inside the surface, which is the convention you should follow.

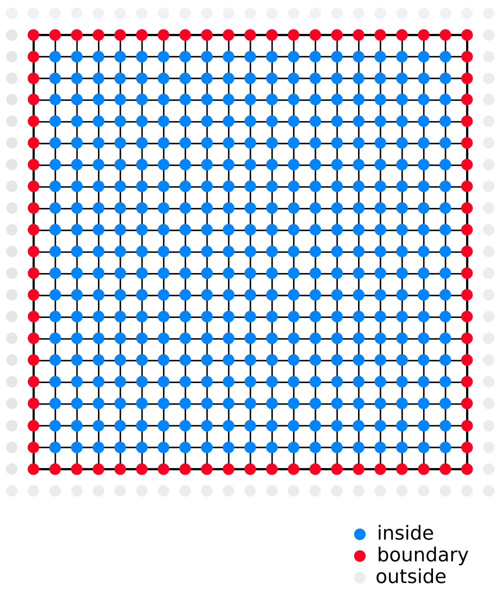

Figure 1: Dense grid with highlighted boundary points.

1.1 Task

Implement a dense rectangular grid on which to store the signed distance function of a surface. The grid should fulfil the following:

- The grid extent in each dimension, as well as the grid spacing(Δx), should be variable
- Both reflective and periodic boundary conditions should be implemented
- Block allocated memory should be used for your data structures

Apply this grid to represent the following surfaces by filling it with the respective signed distance function:

- Sphere (centre, radius)
- Axis-parallel rectangle (minimum Corner, maximum Corner)

In your report, describe how you generated each surface and how the grid resolution of the implicit representation as a level set influences the accuracy of the surface description. Your program should be callable using:

```
./grid x-size y-size spacing [Sphere | Rectangle] [parameters]
```

Hint: Use helper functions to translate neighbour points outside of the grid to indices in your grid according to the chosen boundary condition. If the boundary condition is reflective, a point at $x_{max} + d$ should be mapped to $x_{max} - (d - spacing)$. If it is periodic, the same point should be mapped to $x_{min} + (d - spacing)$.

Hint: In order to analyse your results, you may use the provided vtkOutput.hpp file to generate a VTK Rectilinear Grid. The data can then be visualised using ParaView, which also provides functionality for analysing the data, e.g.: generating the explicit surface (zero level set) under *Filters− > Alphabetical− > Contour* using the contour value 0.

2 Geometric Variables of a Signed Distance Function

During physical simulations, the geometric parameters of a surface, such as the surface normal or the curvature, are often required. In implicit surface representations, these need to be generated from the signed distance function.

2.1 Numerical Derivatives

Due to the properties of signed distance functions, geometric variables can be approximated using its derivatives. Numerically, they are calculated using the forward D_i^+ and backward D_i^- differences

$$D_i^{\pm}(\Phi(\vec{x})) = \pm\frac{\Phi(\vec{x} \pm \vec{e}_i) - \Phi(\vec{x})}{\Delta x} \quad , \tag{1}$$

where $\Phi(\vec{x})$ is the signed distance function at $\vec{x}$, $\vec{e}_i$ the vector to the next grid point in direction i and Δx the grid spacing. The numerical derivative is then given by the central difference, which is the average of the two:

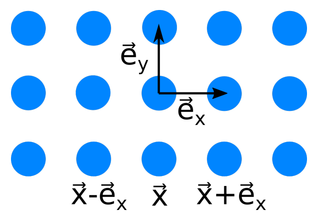

Figure 2: Next neighbours of point at $\vec{x}$ in a dense grid.

$$D_i(\Phi(\vec{x})) = \frac{D_i^+(\Phi(\vec{x})) + D_i^-(\Phi(\vec{x}))}{2} = \frac{\Phi(\vec{x}+\vec{e}_i) - \Phi(\vec{x}-\vec{e}_i)}{2\Delta x} \quad . \tag{2}$$

2.2 Surface Normal

Since the signed distance function increases linearly away from the surface, the direction of maximum change(i.e. the derivative) gives the normal vector to the level set:

$$\vec{n}(\vec{x}) = \frac{\nabla\Phi}{\mid \nabla\Phi \mid} \quad . \tag{3}$$

Therefore, the i-th component of the normal vector can be approximated numerically using central differences:

$$n_i(\vec{x}) \approx \frac{D_i(\Phi(\vec{x}))}{\sqrt{\sum_{j=1}^{D} D_j(\Phi(\vec{x}))^2}} \quad . \tag{4}$$

2.3 Curvature

Since the curvature is essentially the second derivative, it is given by

$$\kappa(\vec{x}) = \nabla \cdot \frac{\nabla\Phi(\vec{x})}{\mid \nabla\Phi(\vec{x}) \mid} = \nabla \cdot \vec{n}(\vec{x}) \quad . \tag{5}$$

Therefore, the simplest approximation to the curvature is the sum of central differences of the normal vectors around the current point:

$$\kappa(\vec{x}) \approx \sum_i D_i(\vec{n}_i(\vec{x})) = \sum_i \frac{n_i(\vec{x}+\vec{e}_i) - n_i(\vec{x}-\vec{e}_i)}{2\Delta x} \tag{6}$$

2.4 Task

Implement functions to calculate the normal vector and curvature for any given point in the dense signed distance grid described earlier. The program should output the normal vector and the curvature at point (x,y) when invoked as follows:

```
./grid x-size y-size spacing [Sphere | Rectangle] [parameters] x y
```

3 Moving the Zero Level Set to Advance a Surface

Since the explicit surface is defined by the zero level set, the surface can be moved by changing the signed distance function. In microelectronic process simulations this would correspond to the passage of time during a fabrication step which etches or deposits on a surface. The simplest way to do this is by adding a constant value to the signed distance function, which will simply shift the interface along the surface normals. Therefore, the time evolution is given by

$$\frac{\partial \Phi(\vec{x})}{\partial t} = -V(\vec{x}) \quad , \tag{7}$$

where $V(\vec{x})$ is a scalar velocity field describing the movement of the surface at any point in the domain.

However, for curved surfaces this would lead to the signed distance function losing its normalisation, which is why the scalar field has to be normalised using the gradient of the signed distance function:

$$\frac{\partial \Phi(\vec{x})}{\partial t} = -V(\vec{x}) \mid \nabla\Phi(\vec{x}) \mid \quad . \tag{8}$$

Therefore, the change of the values in the signed distance function in one time step is

$$\Delta\Phi(\vec{x}) = -V(\vec{x}) \mid \nabla\Phi(\vec{x}) \mid \Delta t \quad . \tag{9}$$

Since the right-hand side of Eq. (8) is a Hamiltonian, many numerical schemes exist to solve it, the simplest being the Engquist-Osher scheme

$$\mid \nabla\Phi(\vec{x}) \mid = \begin{cases} for\ V(\vec{x}) < 0 : \sqrt{\sum_{i=1}^{D} [max(-D_i^-(\Phi), 0)]^2 + [min(-D_i^+(\Phi), 0)]^2} \\ for\ V(\vec{x}) > 0 : \sqrt{\sum_{i=1}^{D} [max(D_i^-(\Phi), 0)]^2 + [min(D_i^+(\Phi), 0)]^2} \end{cases} \quad , \tag{10}$$

which will result in a properly normalised signed distance function in the whole domain, as long as the time steps taken are small enough to satisfy the Courant-Friedrichs-Lewy (CFL) condition:

$$\max \mid V(\vec{x}) \mid \ \leq \ \frac{\Delta x}{\Delta t} \tag{11}$$

3.1 Task

Investigate how advancing the surface changes the signed distance function. Compare the following ways of advancing the surface:

- Simply subtracting a velocity value from every grid point as in Eq. (7)
- Several steps of the Engquist-Osher scheme Eq. (8) - Eq. (11)

Compare the results for $V(\vec{x}) = const$ for a positive velocity greater than several grid spacings for the following shapes:

- Sphere (radius of 10) for $V = 10$ at the resolutions $\Delta x = (1.0, 0.25)$
- Rectangle (with side lengths 5, 20) for $V = 10$ at resolutions $\Delta x = (1.0, 0.25)$

Describe what effect different advection schemes have on the result and how the grid spacing influences the results of each scheme.

Use the normal vectors at each point to introduce a vector velocity function $\vec{V}(\vec{x})$, which can move the surface directionally. The scalar velocity value used in all algorithms, $V(\vec{x})$, can be generated easily by taking the dot product of the vector velocity and the normal vector at that point:

$$V(\vec{x}) = \vec{V}(\vec{x}) \cdot \vec{n}(\vec{x}) \tag{12}$$

Use the normal vectors and curvatures generated earlier, to investigate the following using the Engquist-Osher scheme:

- How does the rectangle behave over many time steps when it is moved laterally by the vector velocity function $\vec{V}(\vec{x}) = (1, 0)$?
- Investigate what happens when the curvature is used as the velocity,
  i.e. $V(\vec{x}) = -\kappa(\vec{x})$. What could this be used for?