Meldshape Evolution Details

Neil Birkbeck

October 18, 2010

1 Introduction

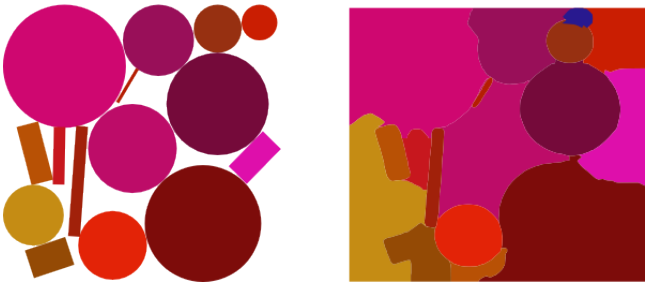

Figure 1: Left: a set of input shapes. Right: a deformation of these shapes such that the result occupies the plane and that some of the shapes retain their original shape.

This document is a description of the implementation of the Meldshape application.

Meldshape is an application for deforming a set of shapes, with some of the shapes retaining their initial structure, and some moving more or less than others (Fig. 1). Ideally, the resulting shapes should occupy the entire domain (a rectangle) and should be non overlapping.

With these constraints, it seemed that an explicit polygonal representation of the shapes during their evolution would be problematic. Instead, an implicit level-set representation was used.

2 Problem Definition

Assume that we are given a set of N input shapes, $\Phi_0^{(N)}$, contained in a rectangular domain A. Specifically, let $\Phi_0^{(N)}$ be the signed distance function to the original shape. The objective is to deform each shape through a shape-dependent motion function such that the rectangular domain, A, is completely occupied.

Consider only the evolution of Φ^1. Let $T_i(\mathbf{x})$ be the signed distance to all shapes other than i, where H is a regularized heaviside function. The Φ_i that minimizes the following objective function should balance the constraints to *occupy* the domain and *preserve* the original shapes.

$$E_i(\Phi_i) = \int_A w^i(\mathbf{x}) \underbrace{\|H(\Phi^i(\mathbf{x})) - H(T_i)\|^2}_{Occupy} + \lambda^i(\mathbf{x}) \underbrace{\psi(\|\Phi^i(\mathbf{x}) - \Phi_0^i(\mathbf{x})\|^2)}_{Preserve} dA$$

The λ_i is a user defined term that allow random motion of the shapes, and w^i is a weighting term used to give more weight in regions that are far from other shapes:

$$w^i(\mathbf{x}) = \max(1, T_i(\mathbf{x})^{0.05})$$

The $\psi(s^2) = \sqrt{s^2 + e^2}$ behaves like a differentiable absolute value function ($e = 0.5$)

The Euler-Lagrange equation of the above equation suggests the following evolution equation:

$$\begin{aligned}\frac{\partial}{\partial t}\Phi^i = \quad & -2w^i(\mathbf{x})(H(\Phi^i(\mathbf{x})) - H(-T_i))H'(\Phi^i) \\ & -2\lambda^i(\mathbf{x})\psi'((\Phi^i(\mathbf{x}) - \Phi_0^i(\mathbf{x}))^2)(\Phi^i(\mathbf{x}) - \Phi_0^i(\mathbf{x}))\end{aligned}$$

The above equation assumed that shapes other than i were constant, which implies that T_i is constant. To allow for motion of multiple shapes, we perform the evolution of each shape independently and update the T_i before each iteration. With a small enough time step, the shapes have enough time to interact with one another without conflicting.

2.1 Motion alternative

An alternative to the energy-based derivation above is to simply derive a speed function that gives the desired results.

One approach that seems promising is to allow the difference in shape to come by modifications to the speed function. In the following speed funcion, points internal to the shape retain that preservation speed, whereas points outside move according to the noise function.

$$\frac{\partial}{\partial t}\Phi^i = \begin{cases} -(\lambda_i(\mathbf{x}) - \tau_i) & \text{if } \Phi_0^i(\mathbf{x}) > 0 \\ 0.2\lambda^i(\mathbf{x})\psi'(s^2)s & \text{otherwise} \end{cases}$$

with $s = (\Phi^i(\mathbf{x}) - \Phi_0^i(\mathbf{x}))$. Here τ_i is a threshold, and λ_i should be interpreted as the speed function.

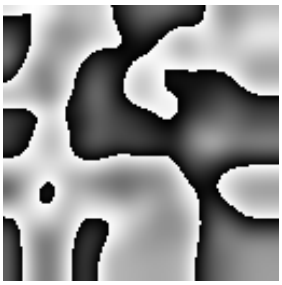

Figure 2: An example of the noise function with intensity wrapping.

Figure 3: Left: the simulation may leave some holes. Middle: large holes are filled with extra shapes. Right: the small gaps are filled by polygon growing and polygon differencing.

2.2 Noise Functions

The noise functions, λ_i, control either the speed or the *stiffness* of the generated shapes. Currently, the noise functions are based on perlin noise.

In order to obtain hard shapes, the noise function is contrast adjusted with the following transform:

$$\begin{cases} 1 - (I(\mathbf{x}) - e_{min})/(e_{max} - e_{min}) & \text{if } I(\mathbf{x}) < e_{min} \\ 1 - (I(\mathbf{x}) - e_{min})/(e_{max} - e_{min}) & \text{if } I(\mathbf{x}) > e_{max} \\ (I(\mathbf{x}) - e_{min})/(e_{max} - e_{min}) & \text{otherwise} \end{cases} \quad (1)$$

where e_{min} (resp. e_{max}) is slightly less (resp. greater) than the min (resp. max) value of in I. Entries, l, less than 0 map to $1 - l$, and entries greater map to $g - 1$. This transformation takes what is typically a smooth noise function into one with hard edges (Fig. 2)

2.3 A complete tiling

Both the evolution functions try to move the shapes in such a way that they completely occupy the simulation domain. However, in practice, the random λ_i images may keep some parts unoccupied, and the approximation of the simulation may leave some space between the space.

The first problem is easily fixed by adding extra shapes in the empty space and restarting the simulatoin. To fix the second problem, we go back to the explicit (polygonal) representation and use polygonal union/difference operations. Each shape starting is traversed in order of most likely to deform. The shape is then grown several pixels. The neighboring shapes are then subtracted from this grown shape to give the final shape. Figure 3 illustrates this filling process. The final result is a set of polygons that tile the plane.

3 Implementation

Efficiency and miscellaneous details regarding implementation.

There are several issues regarding the efficient implementation of the evolution. One of the most important decisions affecting both memory and time is the level set representation. But the evolution needs to access the noise function for every shape, which can occupy a large amount of memory when there are many shapes and a large simulation domain.

For the level set representation, the first attempt used a full level set, with dimensions equal to the full simulation domain (one for each shape). This approach limits both the number of shapes that can be simulated and additionally the resolution of the simulation domain. An alternative is to use a sparse level set (i.e., a narrow band method). However, we found that since many of the shapes in a simulation only occupy a small region, it was sufficient to use a windowed level set representation.

In the windowed level set representation, the level set is maintained and evolved over a small rectangular subset of the domain. When the zero level set interface (contour) approaches the bounds of this domain, the domain is increased. This approach is both simple, and helps eliminate unnecessary memory and computation.

For the noise functions, we also use a similar rectangular window representation. At the beginning of the evolution, the entire noise function for each shape, λ_i is generated. This image is then written to disk, and only a subimage is retained in memory. To keep memory usage to a minimum, the subimage in memory grows dynamically only when required.

4 Results

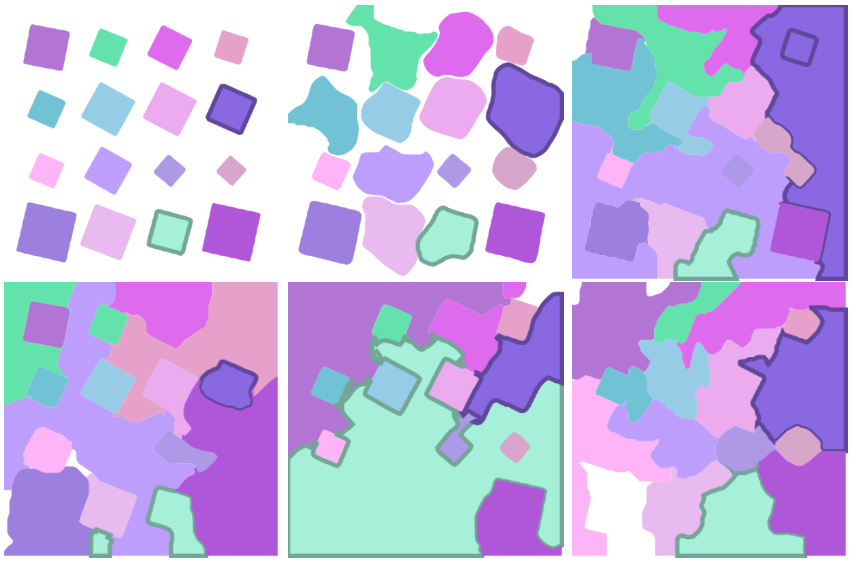

Figure 4: Top: the evolution of a set of squares. Bottom: the same squares evolved with different λ_i. The last one was evolved with the speed-based method.

Figure 5: Some more results obtained with different parameters and different initial configurations