Contents

List of Figures

Chapter 1

Level Set Method

1.1 Introduction

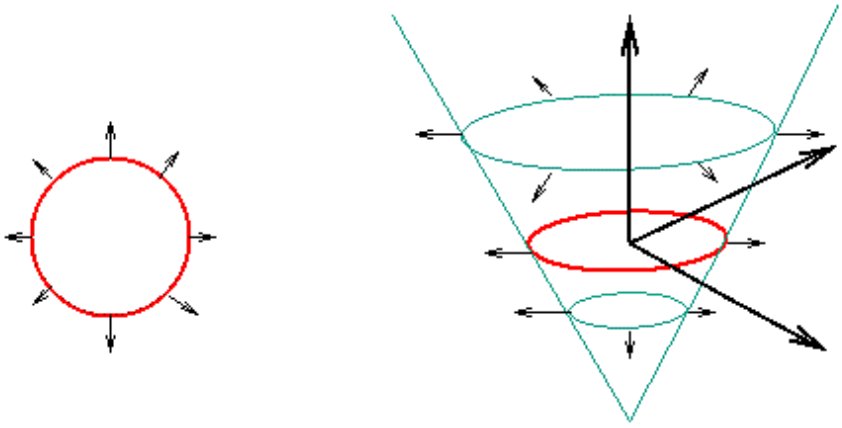
Surfaces that evolves over time can be difficult to represent parametrically. When a surface deforms, the model have to be reparemeterized, which, due to the computional overhead (especially in 3D) add limitations to what kind of shapes a parameterical model can represent effectively. Topological changes, such as splitting or merging parts during the deformation is difficult to represent using parametric models. Sharp corners, distant edges blending together and the complexity of representing boundaries in higher dimensions are some other reasons why an evolving (deforming) surface is difficult to represent parametrically. The level set method is, like Active Shape Model, an extension based on deformable models, and was first introduced by Osher and Sethian in 1988[**?**]. The main idea behind the level set method is to represent the boundary/interface of a surface implicitly by using a higher dimensional function. Adding an extra dimension simplifies the problems mentioned above significantly. This higher dimension function is called the level set function, and represents a 2D model as

$$\phi(x, y, t) \tag{1.1}$$

where the additional dimension t represents time. The level set can represent any model in any dimension by adding this extra dimension. At a given time step, the evolving surface/model can be represented as a closed curve by the boundary of the level set at that time step. This representation of the model is called the zero level set and is defined as the set of points where the level set is zero:

$$\Gamma(x, y, t) = \{\phi(x, y, t) = 0\}. \tag{1.2}$$

The initial curve is at the xy-plane, that is, at $\phi(x, y, 0)$. As an example, figure 1.1a depicts a circle with arrows pointing in the direction it is evolving, and figure 1.1b is the cone that represents the corresponding level set function with the initial/start-position in red.



(a) (b)

Figure 1.1: (a): Circle with arrows pointing in direction of movement, (b): Corresponding level set function

Assuming that the zero level set moves in a direction normal to the speed F, then ϕ satisfies the level set equation

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| F \tag{1.3}$$

which is used to update the level set at each time step (iteration). Here $|\nabla \phi|$ represents the gradient of ϕ, and the speed function F describes how each point in the boundary of the surface evolves. The level set method is applied in many different contexts, such as image processing, fluid dynamics and other simulations, and the speed function F depends on the type of problem being considered.

An often used speed function for image segmentation that combines a data term and the mean curvature of the surface is [**?**, **?**]

$$F = \alpha D(I) + (1 - \alpha) \nabla \frac{\nabla \phi}{|\nabla \phi|} \tag{1.4}$$

where $\nabla \cdot (\nabla \phi / |\nabla \phi|)$ is the normal vector that represents the mean curvature term which keeps the level set function smooth. $D(I)$ is the data function

that forces the model towards desirable features in the input data. The free weighting parameter $\alpha \in [0, 1]$ controls the level of smoothness, and I is the input data (the image to be segmented). The smoothing term α restricts how much the curve can bend and thus alleviates the effect of noise in the data, preventing the model from leaking into unwanted areas[**?**]. This is one of the big advantages the level set method has over classical flood fill, region grow and similar algorithms, which does not have a constraint on the smoothness of the curve.

A simple data function for any point (pixel, voxel) based solely on the input intensity I at that point[**?**, **?**] is:

$$D(I) = \epsilon - |I - T| \tag{1.5}$$

Here T is the central intensity value of the region to be segmented, and ϵ is the deviation around T that should also considered to be inside the region. This makes the model expand if the intensity of the points are within the region $T \pm \epsilon$, and contract otherwise. The data function is gradual, thus the effects of $D(I)$ diminish as the model approaches the boundaries of regions with gray-scale levels within the $T \pm \epsilon$ range [**?**]. This results in the model expanding faster with higher values of ϵ and slower with lower values.

The level set algorithm is initialized by placing a set of seed points that represents a part inside the region to be segmented. These seed points are represented by a binary mask of the same size as the image to be segmented. This mask is used to compute the signed distance function which ϕ will be initialized to.

1.2 Signed Distance Transform

A distance function $D : \mathbb{R}^3 \to \mathbb{R}$ for a set S is defined as

$$D(r, S) = min(r - S) \quad for\ all\ r \in \mathbb{R}^3 \tag{1.6}$$

If a binary image have one or more objects, a distance function can be used to assign a value for every pixel (or voxel in 3D) that represents the minimum distance from that pixel to the closest pixel in the boundary of the object(s). That is, the pixels in the boundary of an object are zero valued, and all other pixels represent the distance to the boundary as a value. Using a distance transform was the idea of how to initialize ϕ in [**?**], where it was initialized as $\phi = 1 \pm D^2$. But in [**?**] it was showed that initializing ϕ to a signed distance function gives more accurate results. Signed distance transforms (SDT) assign for each pixel a value with a positive or negative

sign that depend on whether the pixel is inside or outside the object. The values are usually set to be negative for pixels that are inside an object, and positive for those outside. The pixels of the model, which represents the boundary (the zero level set), have values 0. A binary image containing an object is shown in figure 1.2a (the numbers in this image represent intensity values). Figure 1.2b is the signed distance transform of 1.2a where city-block (manhattan) distance have been used, and figure 1.2c is the signed Euclidean distance transform (SEDT).

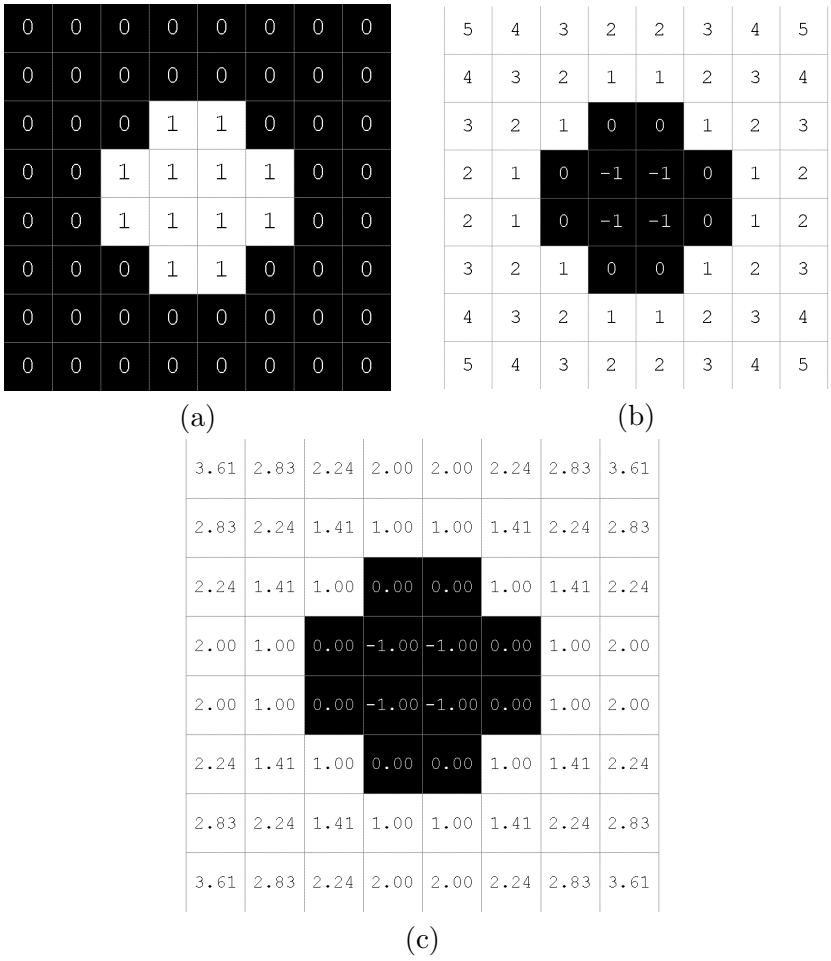


Figure 1.2: (a): Binary image, (b): SDT based on city-block distance, (c): SDT based on euclidean distance

As can be seen from the figures above, using different kind of functions

for the SDT can result in different distances. These differences effects the accuracy of the level set function, which may leads to different end-results of the segmentation, hence, the function used to represent the distance have to be carfully chosen. However, sometimes a less accurate SDT have to be used as a tradeoff for faster computation time.

1.3 Narrow Band

A drawback with the originally proposed level set method is the computional inefficiency due to computing over the whole domain of ϕ. The narrow band introduced in [**?**] *is a method to decrease the computational labor of the standard level set method for propagating interfaces.* Unlike the original level set method, which describe the evolution of an embedded family of contours, the narrow band works with only a single surface model[**?**]. This method ignores points that are far away from the zero level set at each iteration and only looks at the points within a narrow band. This is possible because points far away from the zero level set do not have any influence on the result. That is, only the area of ϕ where $\phi \approx 0$ is important for accurate representation of the level set. The narrow band method restrict the computation to a thin band of points by extending out approximately k points (see figure 1.3) from the zero level set and calculates the SDT for only these. This reduces the number of operations at each iteration from $O(n^{d+1})$ to $O(nk^d)$ [**?**] where d is the number of dimensions (in the image) and n is the (average) number of points in one dimension. As the zero level

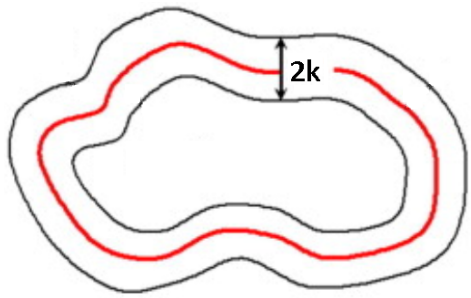


Figure 1.3: The narrow band extending out with a width of k from the level set.

set evolves, ϕ will get further and further away from its initialized value as signed distance. As this happens ϕ must be ensured to stay within the band. This can be accomplished by reinitializing ϕ when the model gets close to

the end of the band. The band have to be reset from the current position of the zero level set, and then ϕ can be reinitialized. Reinitialize ϕ at every iteration takes too much time. But finding out if any of the pixels in the zero level set are getting close to the edge of the band (for every iteration) also takes time. Hence, ϕ is usually just reinitialized after a fixed number of iterations. This keeps ϕ approximately equal to the SDT. As metioned in the section about signed distance transforms, different SDTs can lead to slightly different end-results and must be carefully chosen. If the technique used to approximate ϕ to a signed distance function is too sensitive, ϕ needs to be reinitialized accurately and often. If it is less sensitive, it does not have to be initialized so often and a less accurate method can be used, but this may lead to noisy features [**?**].

The narrow band, despite its improvements over the original level set method, is not optimal. The band used being too wide is the main reason. The band have to be of a certain width (k=12 was used in the test of topological changes in [**?**]) because of two reasons[**?**]. The first is the cost of computing the position of the curve and the SDT, and reset the band. The second is the cost of computing the evolution process over the entire band.

1.4 Sparse Field

The narrow band method assumes that the computation of the SDT is so slow that it cannot be computed for every iteration (time step). The sparse field method introduced in [**?**] uses a fast approximation of the distance transform that makes it feasible to compute the neighborhood of the level set model for each iteration. In the sparse field method the idea of using a thin band is taken to the extreme by working on a band that is only one point wide. The points adjacent to the level set are called active points, and all of them together are referred to as the active set. At each iteration only a thin layer of points near the active set are visited and updated. Using only the active points to compute the derivatives would not give sufficient accuracy. Because of this, the method extends out from the active points in small layers to create a neighborhood that is precisely the width needed to calculate the derivatives needed.

Several advantages to this approach are mentioned in [**?**]. No more than the precise number of calculations to find the next position of the zero level set surface is used. The number of points being computed is so small that a linked-list can be used to keep track of them. This also results in that only those points whose values control the position of the zero level set surface are visited at each iteration. A disadvantage of the narrow band method is

that the stability at the boundaries of the band have to be maintained (e.g. by smoothing) since some points are undergoing the evolution while other neighbouring points remain fixed. The sparse field method avoid this by not letting any point entering or leaving the active set affecting its value. A point enters the active set if it is adjacent to the model. As the model evolves, points in the active set that are no longer adjacent to the model are removed from the active set. This is done by defining the neighborhoods of the active set in layers and keeping the values of points entering or leaving the active set unchanged. A layer is a set of pixels represented as L_i where i is the city-block distance from the active set. The layer L_0 represents the active set, and $L_{\pm 1}$ reprsents pixels adjacent to the active set on both sides. Using linked lists to represents the layers and arrays (matrices) to represent distance values makes the algorithm very efficient. The exact steps of the sparse field algorithm can be found in [**?**].

The sparse field algorithm is based on an important approximation, it assumes that points adjacent to the active points undergoes the same change in value as their nearby active set neighbors. But despite this, the errors introduced by the sparse field algorithm are no worse than many other level set algorithms. Since only those grid points whose values are changing (the active points and their neighbors) are visited at each time step the growth computation time is d^{n-1}, where d is the number of pixels in along one dimension of the image. This is the same as for parameterized models where the computation times increase with the resolution of the domain, rather than the range.

1.5 Discretization by upwinding and difference of normals

To use the level set method in image processing it have to be discretized. This is accomplished using the upwinding differencing scheme. Let ϕ^n and F^n represent the values of ϕ and F at some point in time t^n. The updating process consist of finding new values for ϕ at each point after a time interval Δt. The forward Euler method is used to get a first-order accurate method for the time discretization of equation 1.3, given by (from [**?**])

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + F^n \cdot \nabla \phi^n = 0 \tag{1.7}$$

where ϕ^{n+1} is ϕ at time $t^{n+1} = t^n + \Delta t$, and $\nabla\phi^n$ is the gradient at time t^n. This equation is expanded as follows (for three dimensions):

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + u^n\phi_x^n + v^n\phi_y^n + w^n\phi_z^n = 0, \tag{1.8}$$

where the techniques used to approximate the $u^n\phi_x^n$, $v^n\phi_y^n$ and $w^n\phi_z^n$ terms can be applied independently in a dimension-by-dimension manner [**?**]. When looking at only one dimension (for simplicity), the sign of u^n would indicate whether the values of ϕ are moving to the right or to the left. The value u^n can be spatially varying, hence by looking at only one point x_i in addition to only look at one dimension, equation 1.8 can be written as

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} + u_i^n(\phi_x)_i^n = 0, \tag{1.9}$$

where $(\phi_x)_i^n$ denotes the spatial derivative of ϕ at point x_i at time t^n. The values of ϕ are moving from left to right if $u_i > 0$, thus the points to the left for x_i are used to determine the value of ϕ at point x_i for the the next time step. Similarly, if $u_i < 0$ the movement is from right to left, and the points to the right of x_i are used. As a result, ϕ_x is approximated by the derivative function D_x^+ when $u_i < 0$ and D_x^- when $u_i > 0$. When $u_i = 0$ the term $u_i(\phi_x)_i$ equals zero, and approximation is not needed. Extending this to three dimensions, the derivatives used to update the level set equation are

$$\begin{aligned}
D_x &= \frac{\phi_{i+1,j,k} - \phi_{i-1,j,k}}{2} \quad D_y = \frac{\phi_{i,j+1,k} - \phi_{i,j-1,k}}{2} \quad D_z = \frac{\phi_{i,j,k+1} - \phi_{i,j,k-1}}{2} \\
D_x^+ &= \phi_{i+1,j,k} - \phi_{i,j,k} \quad D_y^+ = \phi_{i,j+1,k} - \phi_{i,j,k} \quad D_z^+ = \phi_{i,j,k+1} - \phi_{i,j,k} \\
D_x^- &= \phi_{i,j,k} - \phi_{i-1,j,k} \quad D_y^+ = \phi_{i,j,k} - \phi_{i,j-1,k} \quad D_z^+ = \phi_{i,j,k} - \phi_{i,j,k-1}
\end{aligned} \tag{1.10}$$

which is taken from the appendix of [**?**]. This is a *consistent* finite difference approximation to the level set equation in 1.3, because the approximation error converges to zero as $\Delta t \to 0$ and $\Delta x \to 0$ [**?**]. In addition to being consistent, it also have to be *stable* in order to get the correct solution. Stability guarantees that small errors in the approximations are not amplified over time. The stability can be enforced using the Courant-Friedreichs-Lewy (CLF) condition which says that the numerical wave speed $\frac{\Delta x}{\Delta t}$ must be greater than the physical wave speed $|u|$,

$$\Delta t = \frac{\Delta x}{max\{|u|\}}, \tag{1.11}$$

where $max\{|u|\}$ is the largest value of $|u|$ on the model.

The gradient $\nabla\phi$ is approximated to either $\nabla\phi_{max}$ or $\nabla\phi_{min}$ depending on whether the speed function for a given point $F_{i,j,k}$ is positive or negative,

$$\nabla\phi = \begin{cases} ||\nabla\phi_{max}||_2 & F_{i,j,k} > 0 \\ ||\nabla\phi_{min}||_2 & F_{i,j,k} < 0 \end{cases} \tag{1.12}$$

where $\nabla\phi_{max}$ and $\nabla\phi_{min}$ is given by (from [**?**])

$$\nabla\phi_{max} = \begin{bmatrix} \sqrt{max(D_x^+, 0)^2 + max(-D_x^-, 0)^2} \\ \sqrt{max(D_y^+, 0)^2 + max(-D_y^-, 0)^2} \\ \sqrt{max(D_z^+, 0)^2 + max(-D_z^-, 0)^2} \end{bmatrix} \tag{1.13}$$

$$\nabla\phi_{min} = \begin{bmatrix} \sqrt{min(D_x^+, 0)^2 + min(-D_x^-, 0)^2} \\ \sqrt{min(D_y^+, 0)^2 + min(-D_y^-, 0)^2} \\ \sqrt{min(D_z^+, 0)^2 + min(-D_z^-, 0)^2} \end{bmatrix} \tag{1.14}$$

The curvature term $\nabla \cdot (\nabla\phi/|\nabla\phi|)$ of the speed function F is discretized using the difference of normals method. The second order derivatives are computed first:

$$\begin{aligned} D_x^{+y} &= (\phi_{i+1,j+1,k} - \phi_{i-1,j+1,k})/2 \quad D_x^{-y} = (\phi_{i+1,j-1,k} - \phi_{i-1,j-1,k})/2 \\ D_x^{+z} &= (\phi_{i+1,j,k+1} - \phi_{i-1,j,k+1})/2 \quad D_x^{-z} = (\phi_{i+1,j,k-1} - \phi_{i-1,j,k-1})/2 \\ D_y^{+x} &= (\phi_{i+1,j+1,k} - \phi_{i+1,j-1,k})/2 \quad D_y^{-x} = (\phi_{i-1,j+1,k} - \phi_{i-1,j-1,k})/2 \\ D_y^{+z} &= (\phi_{i,j+1,k+1} - \phi_{i,j-1,k+1})/2 \quad D_y^{-z} = (\phi_{i,j+1,k-1} - \phi_{i,j-1,k-1})/2 \\ D_z^{+x} &= (\phi_{i+1,j,k+1} - \phi_{i+1,j,k-1})/2 \quad D_z^{-x} = (\phi_{i-1,j,k+1} - \phi_{i-1,j,k-1})/2 \\ D_z^{+y} &= (\phi_{i,j+1,k+1} - \phi_{i,j+1,k-1})/2 \quad D_z^{-y} = (\phi_{i,j-1,k+1} - \phi_{i,j-1,k-1})/2 \end{aligned} \tag{1.15}$$

Then these derivatives are used to compute the normals n^+ and n^- in equation 1.16, which is used to compute the mean curvature H in equation 1.17 taken from [**?**].

$$n^+ = \begin{bmatrix} \frac{D_x^+}{\sqrt{(D_x^+)^2+(\frac{D_y^{+x}+D_y}{2})^2+(\frac{D_z^{+x}+D_z}{2})^2}} \\ \frac{D_y^+}{\sqrt{(D_y^+)^2+(\frac{D_x^{+y}+D_x}{2})^2+(\frac{D_z^{+y}+D_z}{2})^2}} \\ \frac{D_z^+}{\sqrt{(D_z^+)^2+(\frac{D_x^{+z}+D_x}{2})^2+(\frac{D_y^{+z}+D_y}{2})^2}} \end{bmatrix}$$

$$n^- = \begin{bmatrix} \frac{D_x^-}{\sqrt{(D_x^-)^2+(\frac{D_y^{-x}+D_y}{2})^2+(\frac{D_z^{-x}+D_z}{2})^2}} \\ \frac{D_y^-}{\sqrt{(D_y^-)^2+(\frac{D_x^{-y}+D_x}{2})^2+(\frac{D_z^{-y}+D_z}{2})^2}} \\ \frac{D_z^-}{\sqrt{(D_z^-)^2+(\frac{D_x^{-z}+D_x}{2})^2+(\frac{D_y^{-z}+D_y}{2})^2}} \end{bmatrix} \tag{1.16}$$

$$H = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} = \frac{1}{2}[(n_x^+ - n_x^-) + (n_y^+ - n_y^-) + (n_z^+ - n_z^-)] \tag{1.17}$$

Finally, the level set equation is updated as

$$\phi(t + \Delta t) = \phi(t) + \Delta t F|\nabla\phi|. \tag{1.18}$$

1.6 Parallel implementation in GPU

A huge disadvantage with the level set method for segmentation is that it is very slow when working with big data volumes in 3D space. Implementations of level set algorithms for 3D in the graphical processing unit (GPU) parallelizes the level set method and makes it much faster. One of the first GPU based 3D implementations of the level set method was by Lefohn et al. in [**?**] in 2003. In this paper a modified sparse field level set method was implemented for the GPU using graphic APIs such as OpenGL and DirectX. In the past few years general purpose GPUs have made implementing

level set methods and other non-graphical tasks in GPUs much easier. In [**?**] some simple medical segmentation algorithms was implemented using NVIDIAs CUDA technology, and in [**?**] CUDA was used to implement the level set method.

Chapter 2

blablabla

2.1 asdf

test

2.2 cc

test

Chapter 3

blabla

3.1 asdfg

3.2 bb

Chapter 4

bla

4.1 asd

4.2 aa

Bibliography

[1] S. Osher & James A. Sethian, *Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulation.* Journal of computational physics 79.1, 1988.

[2] David. Adalsteinsson & James A. Sethian, *A fast level set method for propagating interfaces.* Journal of Computational Physics, 1994.

[3] Stanley Osher & Ronald Fedkiw, *Level set methods and dynamic implicit surfaces.* Vol. 153. Springer, 2002.

[4] W. Mulder & S. Osher & James A. Sethian, *Computing interface motion in compressible gas dynamics.* Journal of Computational Physics 100.2, 1992.

[5] Ross T. Whitaker, *A level-set approach to 3D reconstruction from range data.* International Journal of Computer Vision 29.3, 1998.

[6] Aaron E. Lefohn & Joe M. Kniss & Charles D. Hansen & Ross T. Whitaker, *A streaming narrow-band algorithm: Interactive computation and visualization of level sets.* IEEE Transactions on Visualization and Computer Graphics, 2004.

[7] Aaron E. Lefohn & Joshua Cates & Ross T. Whitaker, *Interactive, GPU-based level sets for 3D segmentation.* Medical Image Computing and Computer-Assisted Intervention, 2003.

[8] Lei Pan & Lixu Gu & Jianrong Xu, *Implementation of medical image segmentation in CUDA.* Information Technology and Applications in Biomedicine, 2008.

[9] M. Roberts & J. Packer & Mario C. Sousa & Joseph R. Mitchell, *A work-efficient GPU algorithm for level set segmentation.* Proceedings of the Conference on High Performance Graphics, pp. 123-132, Eurographics Association, 2010.