Image Segmentation on Cell-Center Sampled Quadtree and Octree Grids

Byungmoon Kim[a] and Panagiotis Tsiotras[b]

[a]NVIDIA, USA;
[b]Georgia Institute of Technology, Atlanta, USA

ABSTRACT

Geometric shapes embedded in 2D or 3D images often have boundaries with both high and low curvature regions. These boundaries of varying curvature can be efficiently captured by adaptive grids such as quadtrees and octrees. Using these trees, we propose to store sample values at the centers of the tree cells in order to simplify the tree data structure, and to take advantage of the image pyramid. The difficulty with using a cell-centered tree approach is the interpolation of the values sampled at the cell centers. To solve this problem, we first restrict the tree refinement and coarsening rules so that only a small number of local connectivity types are produced. For these connectivity types, we can precompute the weights for a continuous interpolation. Using this interpolation, we show that region-based image segmentation of 2D and 3D images can be performed efficiently.

Keywords: Adaptive Grid, Quadtree, Octree, Level Set, Image Segmentation

1. INTRODUCTION

Partial differential equation (PDE)-based image segmentation performed on high resolution 2D or 3D images requires typically high resolution grids. These grids can be dense, often facing memory or computational limitations. In contrast, adaptive grids can dramatically reduce memory and computational requirements, since the grid can be adaptively refined close to the sharp features of the image and coarsened at flat boundaries and regions far from the image boundary.

Such adaptivity is trivially obtained using quadtree and octree grids since their hierarchical structure allow us to refine or coarsen the grids. Among various quadtree or octree implementations, storing values at the cell corners provides an easy interpolation of the data,[12] but complicates the tree data structure. Storing sample values at the center of the tree cell on the other hand has a simple tree data structure, parent-child relationship between samples, and often has significant benefits such as image pyramid levels trivially mapped to the octree. However, the cell-centered sampling approach comes with a cost of continuous interpolation that requires triangulation or tetrahedralization.[3] As a result, the cell-centered approach has not been broadly used in the literature thus far. We address the problem of continuous interpolation for cell-centered sampling by developing efficient continuous bilinear and tri-linear interpolation methods. To facilitate the interpolation, we develop a special quadtree and octree grid structure. Since the proposed tree structure produces only a small number of local connectivity types, we can pre-compute the interpolation rule for each type. We first identify the interpolation box that is made of the neighboring cells of the same depth. Then, the interpolation box is divided by smaller sub-grid boxes, whose corner values are interpolated from neighboring cell center values. We pre-compute the interpolation rules at each sub-grid points as a set of weight and cell pairs, and store them in a look-up table. These pre-computed interpolation look-up table values greatly reduce the interpolation cost. We also use this table to differentiate the values and build the iso-surfaces. By using the new tree structure and the continuous interpolation, we show that large 2D or 3D images can be efficiently segmented with a quadtree or an octree.

2. AN OVERVIEW ON REGION-BASED LEVEL SET SEGMENTATION

Since their introduction almost two decades ago,[4–6] level sets have been used in a large number of applications, including image segmentation applications. The latter are performed either using edge-based level set methods,[7,8] or region-based level set methods.[9] While edge-based methods are used to segment a feature that has a boundary edge, region-based methods can be used when a feature in an image has statistics that are different from the background. Examples of such statistics are intensity,[9] texture,[10] or motion.[11] Region-based level set methods can also be used to segment vector-valued images,[12] or to segment an image into more than two regions using multi-phase level sets.[10,13] In particular, region-based

level set segmentations are suitable to adaptive tree grids since a tree cell can contain statistics inside the cell. Therefore, in this work we choose to work with region-based level set segmentation methods. In this section, we provide an overview of the region-based level set segmentation method.

Given an image $u_0(\mathbf{x})$, where $\mathbf{x}$ is the location in the image, Chen et al[9] defined the image segmentation problem as a problem of minimizing the following functional:

$$\begin{aligned} F(c_1,c_2,C) &= \mu\,\text{Length}(C)+\nu\text{Area}(\text{inside}(C)) \\ &\quad +\lambda_1\int_{\text{inside}(C)}|u_0(\mathbf{x})-c_1|^2\,\mathrm{d}\Omega+\lambda_2\int_{\text{outside}(C)}|u_0(\mathbf{x})-c_2|^2\,\mathrm{d}\Omega, \end{aligned} \tag{1}$$

where C is the unknown curve boundary of image segment to be found, and c_1 and c_2 are average values inside and outside the curve C, respectively. As we integrate over time, the shape of C and the values of c_1 and c_2 will be updated. In (1) μ and ν are length and area weights, and λ_1, and λ_2 are weights for the user to choose.

Region-based segmentation solves (1) by first representing the curve C as the zero level set of a scalar function $\phi(\mathbf{x})$. The value of $\phi(\mathbf{x})$ is the signed distance from $\mathbf{x}$ to C. Using this level set function $\phi(\mathbf{x})$, and the Heaviside function $H(\phi)$, we can write $F(c_1,c_2,C)$ as

$$\begin{aligned} F(c_1,c_2,C) &= \mu\int_\Omega \delta(\phi(\mathbf{x}))|\nabla\phi(\mathbf{x})|\mathrm{d}\Omega+\nu\int_\Omega H(\phi(\mathbf{x}))\mathrm{d}\Omega \\ &\quad +\lambda_1\int_\Omega|u_0(\mathbf{x})-c_1|^2H(\mathbf{x})\mathrm{d}\Omega+\lambda_2\int_\Omega|u_0(\mathbf{x})-c_2|^2(1-H(\mathbf{x}))\mathrm{d}\Omega. \end{aligned} \tag{2}$$

We first regularize δ and H as δ_ε and H_ε, and then compute the Euler-Lagrange equation for ϕ that minimizes F as

$$\frac{\partial\phi}{\partial t}=\delta_\varepsilon\left[\mu\nabla\cdot\frac{\nabla\phi}{|\nabla\phi|}-\nu-\lambda_1(u_0-c_1)^2+\lambda_2(u_0-c_2)^2\right]. \tag{3}$$

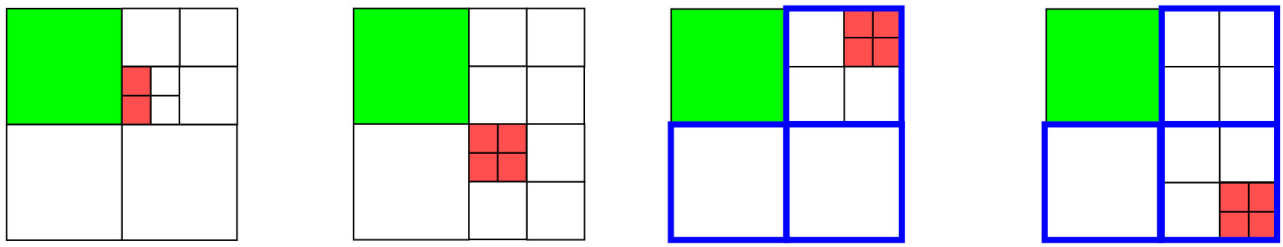
We can segment images by solving (3). First, the user specifies an initial shape, say, a big circle. This will define an initial level set to start with. We then evaluate the right hand side of (3) and perform the explicit time integrations.

3. INTERPOLATION ON THE QUADTREE AND OCTREE GRIDS

Continuously interpolating the level set values $\phi(\mathbf{x}_i)$ sampled at tree cell centers $\mathbf{x}_i$ is an important ingredient to implement the previous approach. A continuous interpolation provides a continuous field $\phi(\mathbf{x})$. This allows us to define the continuous interface line $\{\mathbf{x}|\phi(\mathbf{x})=0\}$. In addition, to solve a PDE on the grid, a continuous interpolation is also necessary. For example, the advection type PDE $\dot{\phi}=\nabla\cdot\mathbf{u}$, where $\mathbf{u}$ is the velocity, can be stably solved using semi-Lagrangian methods, e.g., the CIR method, which requires interpolation. If this interpolation is not continuous, significant noise on the interface may result. Thus, continuous interpolation is necessary to solve advection-type PDEs.

Continuous interpolation is trivially implemented on a regular grid, or even in quadtree or octree grids by sampling values at the corners of a tree cell. However, in cell-centered quadtrees or octrees, continuous interpolation is not trivial. In this section, we develop a new method to continuously interpolate the center-sampled values $\phi(\mathbf{x}_i)$.

Previously, center-sampled values were interpolated by using a triangulation (or tetrahedralization) of the center locations.[3] This operation is relatively slow (i.e., of order $O(n\log n)$), and required an additional triangle or tetrahedralization data structure. The first observation is that this operation can be accelerated if we restrict the tree connectivity so that only a small number of local tree connectivity types exist, and then pre-compute the interpolation weights for each connectivity type. Another problem in the triangulation or tetrahedralization-based interpolation is the fact that a triangle or a tetrahedron-inside test is more expensive than bilinear or trilinear interpolations in axis-aligned boxes. Therefore, we develop an interpolation method that is based on axis-aligned interpolation boxes.



(a) (b) (c) (d)

Figure 1. Invalid quadtree structures: (a) and (b) are not valid since the green cell and the red cells are neighbors, but their depth difference is greater than one. (c) and (d) are not valid, since the green cell's same-depth neighbors are the three blue cells, one of which contains a non-leaf child cell (lower right child is not a leaf, but has four red children).

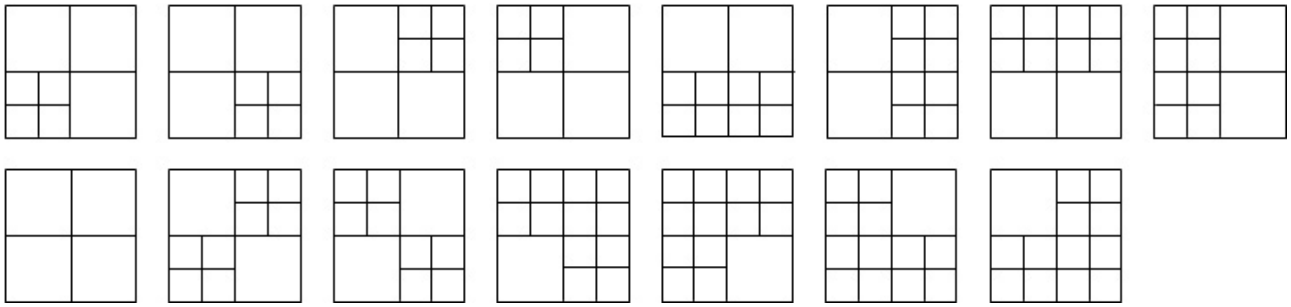


Figure 2. Only 15 quadtree connectivity types (even without considering symmetries) satisfy the quadtree constraints.

3.1 Construction of Octree/Quadtree Constraints

With the goal in mind to limit the number of local cell connectivity types, we develop the following constraints:

1. We allow only one resolution difference between adjacent grid cells. In Fig. 1, (a) and (b) show the green and red cells violating this condition, since the neighbor in a green cell and a red cell have depth difference greater than one.

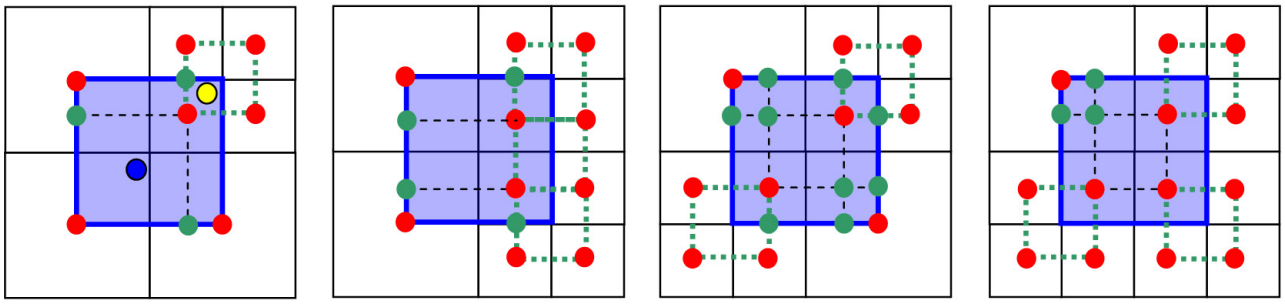
2. All the same-depth neighbors are either leaves or have leaf children only. In Fig. 1, (c) and (d) show the green cell's same-depth neighbors (blue cells) containing a non-leaf children (the red cell).

These two constraints produce a small number of connectivity types illustrated in Fig. 2 for a quadtree. In all quadtree cells, we can find the four neighboring same-depth cells that are either leaf cells or have four leaf children. Since each of the four cells can have children, there can be 16 different cases. However, if all the four cells have leaves, then the connectivity is equivalent to the case in which all the four cells are leafs. Therefore, there are 15 different local connectivity types. Similarly, on an octree, we can always find eight cells that are either leaf cells or have leaf children. Therefore, there exist 255 different connectivity types. The number of connectivity types can be further reduced by removing symmetric cases, but we do not pursue this reduction, since the number of 255 cases is still a small number that does not require much memory space, even if we precompute the interpolation weights for all of the 255 cases. Thus, only a small number of connectivity types are produced, and therefore, we can pre-compute all interpolation operations.

The above constraint makes the tree resolution decay slowly as we move away from a high curvature interface region. At first glance, this appears to be inefficient, since the lower the resolution we can make, the greater savings we obtain. Note, however, that the amount of reduction is often insignificant since the number of grid cells is dominated by the number of the fine grid cells in the high curvature interface region.

3.2 Interpolation

In an octree or a quadtree with the previous constraints applied, we now consider how to interpolate cell-centered data. We prefer to use axis-aligned interpolation boxes since identifying such boxes is easier. For example, in Fig. 3 (a), suppose we want to interpolate values at the yellow marker. By comparing the coordinates of the yellow marker and the coordinates of



(a) (b) (c) (d)

Figure 3. Various interpolation boxes and their sub-boxes. At the yellow point, bilinear interpolation is performed in the small box enclosed by the green dotted line. At the blue point, bilinear interpolation is performed in the lower-left sub-box of the blue box.

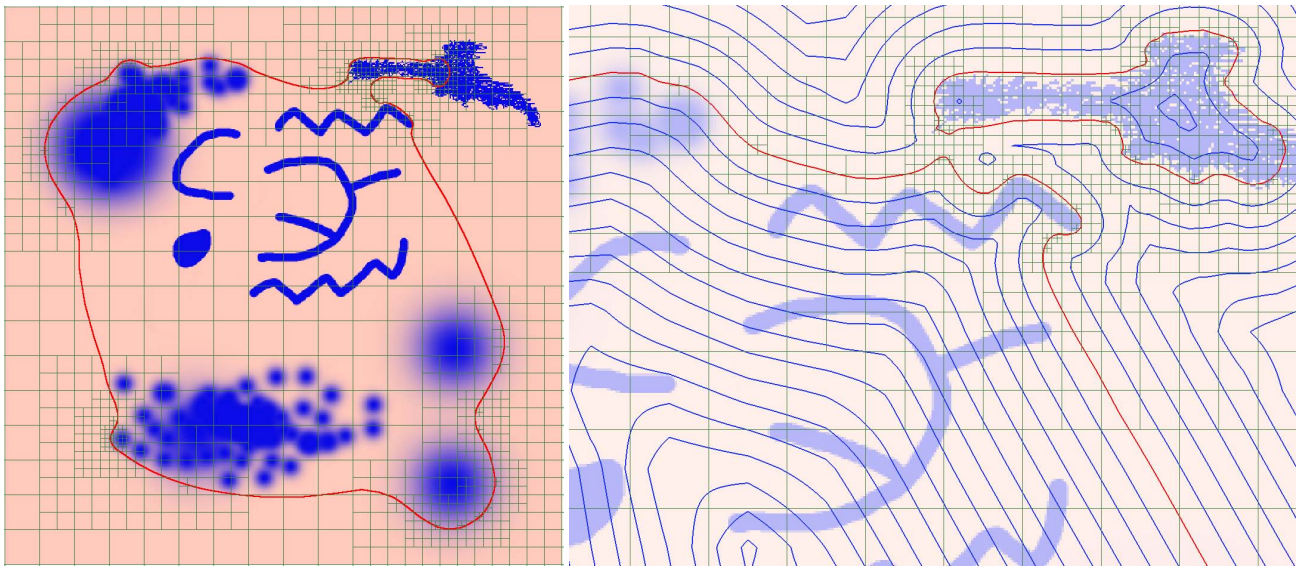


Figure 4. Level set values sampled at cell centers are interpolated continuously. This continuous interpolation produces the smooth red interface line (left), or the contour lines (right), which are continuous across differently-sized tree cells .

the neighboring cell centers, we immediately know that bilinear interpolation can be done in the box enclosed by the dotted green lines, and we can perform bilinear interpolation with the four samples at the corners of the box (the red samples). Similarly, if we want to compute the value at the blue marker, we can quickly identify the blue box. In this case, however, interpolation is not obvious since the upper right corner of the blue box does not contain a valid sample.

To interpolate at the blue marker in Fig. 3 (a), we split the blue box as shown by the black dotted lines. This step produces the three sub-boxes shown in Fig. 3 (a). Again, by comparing the coordinates of the blue point and the dotted lines, we can trivially identify the sub-box that contains the blue marker. However, in order to perform bilinear interpolation in this sub-box, we must compute the interpolated value at the green markers as weighted sums of the values at the red markers. We simply pre-compute these interpolation weights for green markers for all the 15 or 255 different box types for a quadtree or a octree, respectively. This way, we can quickly compute the values at the green markers. Now, we have all values at the four corners of all sub-boxes. Finally, we can perform the bilinear interpolation in each sub-box. This interpolation is continuous, since interpolation (sub-)boxes do not overlap, and interpolations are consistent at the (sub-)box boundaries. See Fig. 4 for interpolation examples.

3.3 Computing Derivatives

In order to evaluate the right hand side of (3), we must compute the partial derivatives $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$, and $\frac{\partial}{\partial z}$ at the center of each grid cell. In Fig. 5, suppose that Δx is the size of the cell that contains ϕ_0. Then, $\frac{\partial \phi}{\partial x}$ at ϕ_0 can be approximated as the

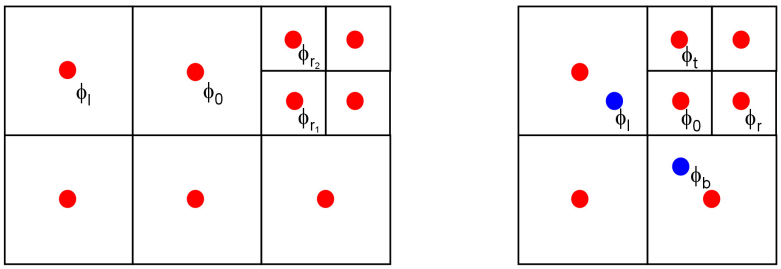
average value between the left- and right-hand side derivatives. The left-hand side derivative is simply $(\phi_0 - \phi_l)/\Delta x$. The right-hand side derivative is computed as $(\frac{1}{2}(\phi_{r_1} + \phi_{r_2}) - \phi_0)/(\frac{2}{3}\Delta x)$. Finally, we obtain

$$\frac{\partial \phi}{\partial x} = \frac{1}{2}\left[\frac{\phi_0 - \phi_l}{\Delta x} + \frac{\frac{1}{2}(\phi_{r_1} + \phi_{r_2}) - \phi_0}{\frac{2}{3}\Delta x}\right]. \tag{4}$$

From the Taylor series expansion, we see that this derivative has error $O(\Delta x)$. Now, consider computing $\frac{\partial \phi}{\partial x}$ in the case (b) in Fig. 5. In this case, we first compute ϕ_l and then take the centered difference, i.e.,

$$\frac{\partial \phi}{\partial x} = \frac{\phi_r - \phi_\ell}{2\Delta x}. \tag{5}$$

Since ϕ_ℓ is computed by bilinear interpolation, ϕ_ℓ contains an error of order $O(\Delta x^2)$. Therefore, $\frac{\partial \phi}{\partial x}$ computed by (5) contains an error of order $O(\Delta x)$. Note that in image segmentation problems, derivatives with accuracy higher than $O(\Delta x)$ are not necessary since ϕ will converge to a static field, and therefore, most of time integration accuracy issues (such as numerical diffusion) during the transient has no adverse effect on the final result.



(a) (b)

Figure 5. Differentiation across different resolution levels.

4. MESH REFINEMENT

As the level set is updated, the interface moves. As a result, some cells may move away from the interface, while other cells may come closer to the interface. In addition, the curvature of the interface may change. Therefore, the mesh must be adaptively refined by subdividing a cell into smaller cells, or coarsened by merging child cells. In this section, we discuss the criteria for coarsening and refining the cells.

The mesh is refined if the distance to the interface is close enough, and the curvature is high enough. Conversely, if the distance to the interface is large, the mesh is coarsened, regardless of the curvature. In addition, if the curvature is not high, the mesh is coarsened, regardless of the distance to the interface. In order to facilitate the discussion below, we use *coarseness* to refer to the degraded mesh resolution. Where the coarseness is zero, the mesh is fully refined, and where the coarseness is, say, three, the mesh has tree depth smaller than the maximum depth by three. Note that coarseness is a function of the curvature $\kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}$, and the distance to the interface $|\phi|$.

We propose to compute the coarseness of the cell as

$$C(\phi, \kappa) = \frac{|\phi|}{\sigma d_{\max}} + c_1 \tan^{-1}\left(\frac{c_2 \kappa_C}{\kappa}\right), \tag{6}$$

where $d_{\max}$ is the maximum depth, σ is the thickness of the interface where maximum resolution is maintained when the curvature is high, $C_{\max}$ is the maximum allowed coarseness at the interface, κ_C is the curvature at which the mesh is coarsened by one, $c_1 = 2C_{\max}/\pi$, and $c_2 = \tan(1/c_1)$. When $\phi \to 0$ and $\kappa \to \infty$, the coarseness is reduced, i.e., $C(\phi, \kappa) \to 0$. In smooth regions, the curvature κ is small. In this case, the coarseness is increased, i.e., when $\kappa \to 0$, $C(\phi, \kappa) \to |\phi|/(\sigma d_{\max}) + C_{\max}$. When the curvature is $\kappa = 1/r_C$, the coarseness takes the value $C(\phi, \kappa) = |\phi|/(\sigma d_{\max}) + 1$.

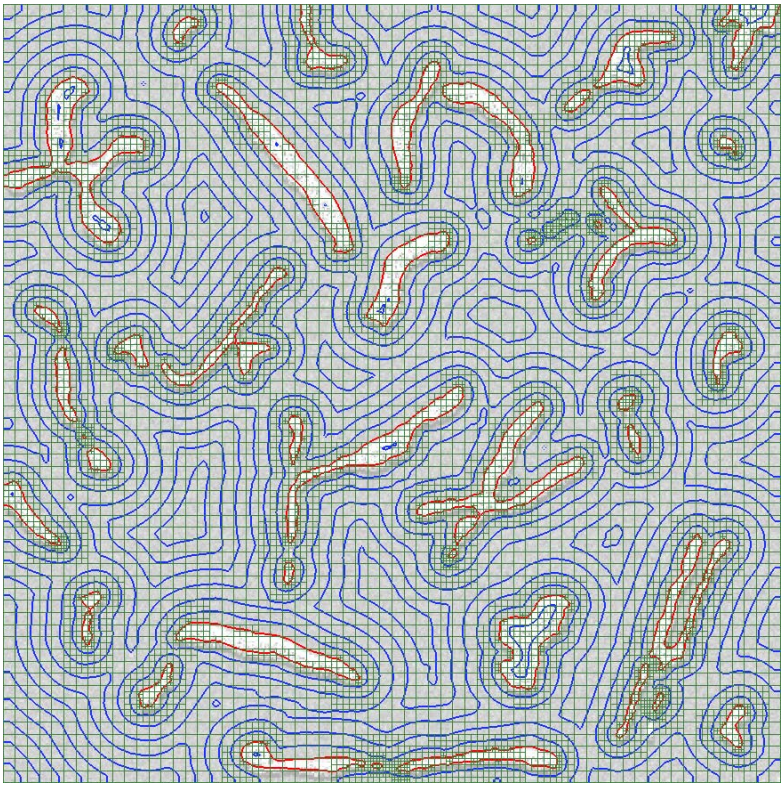


Figure 6. Segmentation of a 512×512 image. Brute force segmentation without using other speed up methods, such as narrow band level set, results in a segmentation within 4 seconds on an Intel Q6600 (3.6GHz), DDR2 800 system. Significant speedup can be achieved by using the image pyramid. See Fig. 7 for an example.

Once $C(\phi,\kappa)$ is computed, we can compute the desired tree depth as $d_{\text{desired}} = d_{\max} - C(\phi,\kappa)$. We then compare d_{desired} and the current depth d. If $d_{\text{desired}} > d$, we can merge the tree. However, since we must satisfy the tree connectivity constraint, we cannot always merge to enforce d_{desired}. In contrast, if $d_{\text{desired}} < d$, we always refine the cell, and refine the neighboring cells to meet the constraint.

5. RESULTS AND DISCUSSION

Figure 6 shows a segmentation result on a 2D image. Using an explicit time integration of (3) with a constant time step, we obtained the segmented image in about 4 seconds (400 time steps). As shown in Fig. 4, and 6, the level set interfaces remained smooth during the image segmentation processes, without developing any artifacts.

Figure 7 shows the result of the segmentation of a synthetic 3D test image. With this image, even naive time integration with a small constant time step converged in 34 seconds, although the interface is moving only by a small amount at each time step. The computation time is greatly reduced to 0.5 second by first taking eight time steps in a $64\times64\times64$ grid (i.e., we set $d_{\max}=6$), and then taking eight more time steps in a $128\times128\times$ grid ($d_{\max}=7$). To compute the level set interface mesh, we used the marching cube algorithm with a modification to irregular grids.

Figure 8 shows the segmentation result for a volumetric cell image taken from the CCDB site (http://ccdb.ucsd.edu). Since the cell shape has high curvature regions, cells are refined to high depths. As a result, naive integration of (3) with $d_{\max}=9$ took a very long time (30min). Again, by segmenting in lower resolution first, and then moving on to higher resolution, the segmentation computation time is reduced to approximately 40 seconds.

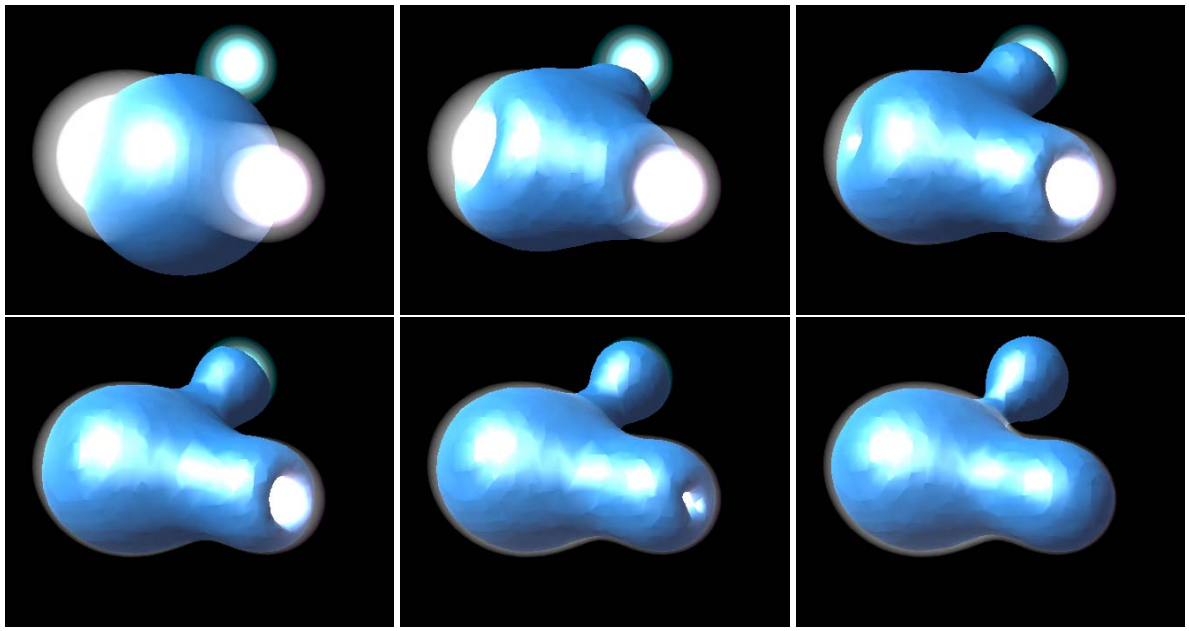


Figure 7. Segmentation of a volumetric $128 \times 128 \times 128$ set of data. Naive segmentation takes approximately 34 seconds on a Intel Q6600 CPU. However, if we use very large time steps (i.e., k-mean clustering[14]) in a coarse image pyramid level and move on to finer level with refined time steps, the level set is converged within about 0.5 second (0.2 sec for eight time steps in $64 \times 64 \times 64$ image pyramid level (we simply set $d_{\max} = 6$), and 0.3 sec for eight time steps in the $128 \times 128 \times 128$ level ($d_{\max} = 7$).

6. CONCLUSION AND FUTURE WORK

The main weakness of using cell-centered octrees is the complexity of interpolation. We show that this complexity can be resolved by the proposed tree connectivity constraints and by precomputing the interpolation values using interpolation rules on a small number of local connectivity types. In addition, we demonstrate that the cell-center-sampled quadtree and octree allows us to accelerate segmentation by using the image pyramid. We plan to further explore the accommodation of several common speed-up techniques, such as narrow-band or freezing the converged region.

7. ACKNOWLEDGEMENT

Partial support for the second author has been provided by NSF through award no. CMS-0510259.

REFERENCES

[1] Losasso, F., Gibou, F., and Fedkiw, R., "Simulating water and smoke with an octree data structure," in [*ACM SIGGRAPH*], 457–462 (2004).

[2] Bai, Y., Han, X., and Prince, J. L., "Octree grid topology preserving geometric deformable model for three-dimensional medical image segmentation," in [*Information Processing in Medical Imaging (IPMI 2007)*], (July 2007).

[3] Strain, J., "Fast tree-based redistancing for level set computations," *Journal of Computational Physics* **152**(2), 648–666 (1999).

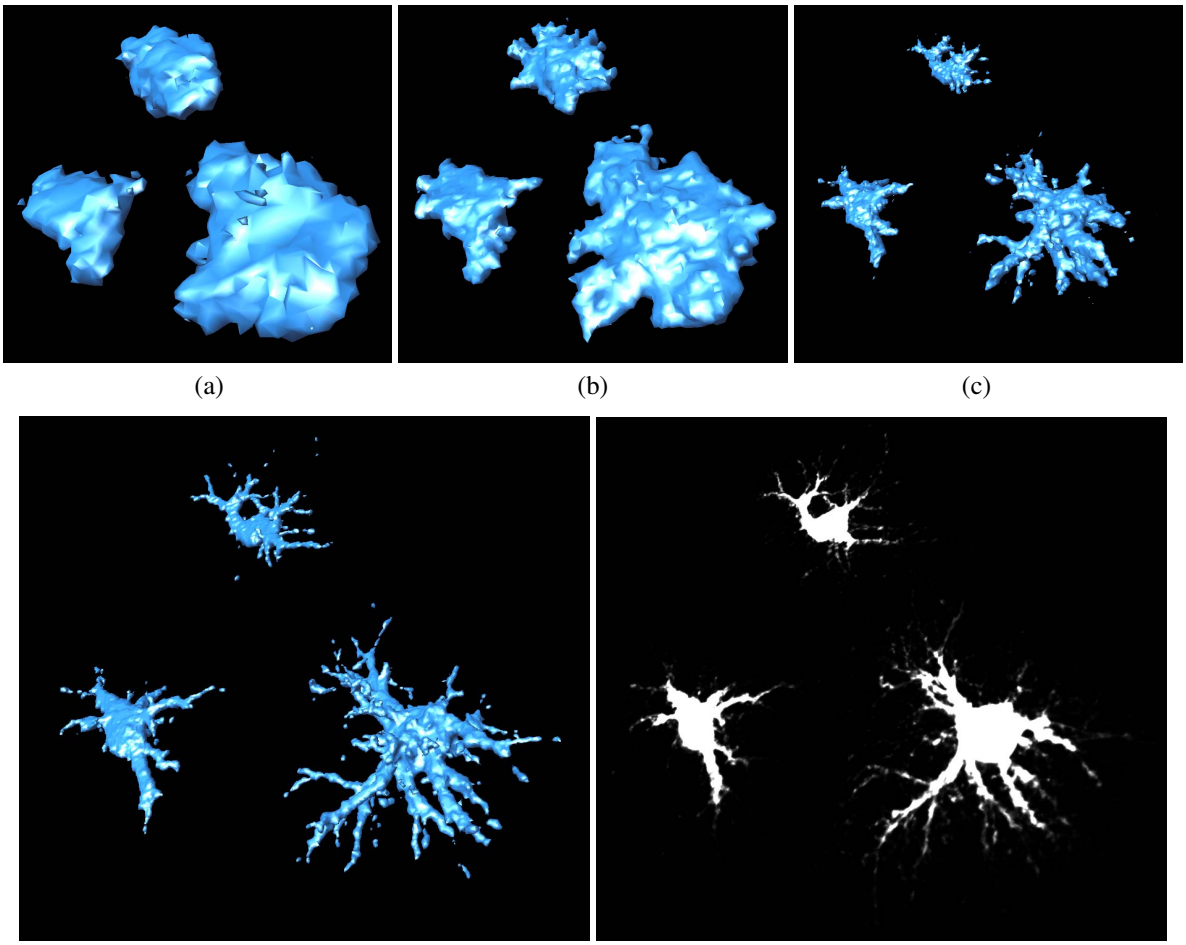
[4] Osher, S. and Sethian, J. A., "Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations," *Journal of Computational Physics* **79**, 12–49 (1988).

[5] Osher, S. J. and Fedkiw, R. P., [*Level Set Methods and Dynamic Implicit Surfaces*], Springer-Verlag (2002).

[6] Sethian, J. A., [*Level Set Methods and Fast Marching Methods*], Cambridge University Press (1999).

[7] Malladi, R., Sethian, J. A., and Vemuri, B. C., "Evolutionary fronts for topology-independent shape modeling and recovery," in [*Proceedings of the third European conference on Computer vision*], 1–13 (1994).

[8] Caselles, V., Kimmel, R., and Sapiro, G., "Geodesic active contours," in [*International Conference of Computer Vision (ICCV)*], 694–699 (1995).



(a) (b) (c)

(d) (e)

Figure 8. Segmentation of a cell image data of size $512 \times 512 \times 64$ took about 30 minutes by naively solving the PDE. The segmentation time can be reduced to 40 seconds by first segmenting in a low resolution and then moving on to higher resolutions. Images (a),(b),(c), and (d) are segmentation results in $64 \times 64 \times 8$, $128 \times 128 \times 16$, $256 \times 256 \times 32$, and $512 \times 512 \times 64$ resolutions respectively. Volume data shown in (e) is courtesy of CCDB (http://ccdb.ucsd.edu).

[9] Chan, T. F. and Vese, L. A., "Active contours without edges," *IEEE Transactions on Image Processing* **10**(2), 266–277 (1999).

[10] Brox, T. and Weickert, J., "Level set segmentation with multiple regions," *IEEE Transactions on Image Processing* **15** (October 2006).

[11] Cremers, D., "A variational framework for image segmentation combining motion estimation and shape regularization," in [*IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*], 53–58 (2003).

[12] Chan, T. F., Sandberg, B. Y., and Vese, L. A., "Active contours without edges for vector-valued images," *Journal of Visual Communication and Image Representation* **11**(2), 130–141 (2000).

[13] Vese, L. A. and Chan, T. F., "A multiphase level set framework for image segmentation using the mumford and shah model," *IEEE Transactions on Image Processing* **50**(3), 271–293 (2002).

[14] Gibou, F. and Fedkiw, R., "A fast hybrid k-means level set algorithm for segmentation," in [*4th Annual Hawaii International Conference on Statistics and Mathematics*], (2002).