Robust Volume Segmentation using an Abstract Distance Transform

Márton Tóth
Dávid Dvorszki
*Supervised by: Balázs Csébfalvi**

Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest / Hungary

Abstract

In this paper, a novel segmentation technique is introduced that is based on an abstract distance transform. The user can select a voxel as a seed point, from which the other voxels can be reached along different paths. For each voxel, we determine a path that is of minimal cost. Taking the density of the seed point as a reference, the cost of a path is calculated as an aggregate deviation of the densities corresponding to voxels visited along the given path. After having the cost of the cheapest path assigned to each voxel, a simple thresholding is used to obtain a segmentation mask. We demonstrate that this approach is competitive to the popular level set segmentation, but can be implemented more efficiently on recent Graphics Processing Units (GPU). Previously, using high-level shader languages for direct programming of the graphics pipeline, the implementation of different segmentation methods was difficult. However, new languages such as OpenCL or CUDA provide a more flexible environment for GPGPU (General Purpose computing on the GPU) programming. We show that, utilizing this new technology, our algorithm can be easily realized.

Keywords: Segmentation, GPGPU programming, Volume rendering

1 Introduction

Segmentation is a fundamental task in 3D medical image processing [13]. The goal is to identify different organs or tissues in the data. Usually a binary classification of the voxels is not possible, instead a probability is obtained that expresses how much the given voxel belongs to a certain organ. Furthermore, a simple thresholding [16] of the density values generally cannot be applied for soft tissue segmentation, since due to the noise contained in the data it often leads to under or oversegmentation. Therefore, many more sophisticated segmentation algorithms have been proposed that utilize secondary information, such as gradient or curvature [13, 11, 7, 15], distance or connectivity [13, 14, 11], or a priori anatomical information [3]. Most of these methods are computationally expensive, therefore their traditional CPU implementation is rather inefficient.

Recently, the conventional GPUs are more and more used as general-purpose coprocessors exploiting their parallel computing capacity. Originally, the GPUs were designed for fast incremental image synthesis, which is the core of many computer graphics applications. GPUs are in fact optimized for rendering huge polygonal meshes. The standard computer graphics pipeline includes vertex processing, rasterization, fragment (or pixel) processing, and compositing. The first technological milestone was reached when the vertex and pixel processing became programmable, since it allowed the redefinition of the standard pipeline for general-purpose computation [18]. However, using direct shader programming for the implementation of complex segmentation techniques was still cumbersome. For example, volumetric data represented by texture maps could not be read and written at the same time, and the number of instructions in the vertex and pixel shaders was also limited. Therefore, several rendering passes were required for mapping the segmentation process onto the graphics hardware.

OpenCL and CUDA represent the second milestone in GPGPU programming [17]. Using these languages for developing GPU applications, the programmer can abstract from the graphics pipeline considering the GPU as a general multi-processor architecture. In this paper, we show that this higher flexibility can be very well exploited in GPU-based segmentation.

In Section 2, we briefly review the previous methods that are related to our work. In Section 3, the GPU implementation of the popular Level Set (LS) method is described. Our new segmentation technique called Abstract Distance Transform (ADT) is introduced in Section 4. In Section 5, ADT is compared to LS on a brain segmentation task. Finally, in Section 6, the contribution is summarized.

*cseb@iit.bme.hu

2 Related Work

Medical image segmentation has a wide literature. Although there exist general segmentation techniques [13], practical methods are often designed and optimized for a very specific segmentation task. In this section, we overview only those previous techniques that are the most closely related to our method.

The simplest segmentation technique is thresholding [16]. It requires only two parameters, a lower and a higher threshold. If the intensity of a voxel is between these two threshold values then it belongs to the segmentation mask. Thresholding works fine if the given material is well defined by an intensity interval like the bone in a CT scan, for instance. Soft tissue structures, however, are much more difficult to segment. Especially in MRI data sets, the data values are not so coherent as in CT data because of the higher noise-to-signal ratio and the so called global signal fluctuation [12]. Therefore, the connectivity information plays a crucial role in the segmentation process. Region-growing techniques try to find voxel regions that are connected and show a nearly homogeneous density distribution. This is an iterative approach started from a user-selected seed point. In each iteration step, the boundary voxels of the current segmentation mask are investigated and it is decided whether their neighboring voxels should be added to the mask or not. Variants of the region-growing method differ in the criteria that are used for this decision [11, 14, 13]. For example, edge detection operators are usually applied to find the material boundaries [11]. In practice, pure region-growing often leads to so called segmentation leakage [7], where the binary mask unexpectedly leaks through the boundaries of the organs to be segmented. This problem can be somehow handled by starting a region growing from anti seed points, which fills those voxels that should not be added to the segmentation mask. However, an appropriate placement of the anti seed points might be difficult or time-consuming for the physician. In order to complete the segmentation without significant user intervention, the level set method was proposed [15]. This approach is also based on region growing, but it defines additional constraints on the boundary surface of the segmentation mask. Usually, the iteration steps are performed in such a way that the total curvature of the boundary surface is kept under a reasonable level. As we compare our new graph-based method to the level set technique, in Section 3, we describe its GPU implementation in detail.

There exist other graph-based segmentation techniques that are commonly used in practice, such as intelligent scissors [10] or the graph-cut method [2]. Both of them consider the image as a weighted graph. With intelligent scissors, the user can define a cut in a 2D image by giving two points and the algorithm calculates a path between them trying to follow edges as far as possible. The weights are calculated by edge detection methods and the path is constructed by a shortest path method. However, intelligent scissors cannot be easily extended to 3D [4]. In the graph-cut technique, the user can define a foreground and a background point and the algorithm calculates a border between them separating the two regions based on the max-flow min-cut theorem. The weights can be estimated by using region and edge properties. This approach can be applied in 3D as well. Our method is based on aggregate deviation calculation and uses the Bellman-Ford algorithm to determine a shortest path between pairs of voxels.

3 GPU-Accelerated Level Set Segmentation

The LS algorithm consists of an initialization and an iterative region growing. In the initialization step, the user specifies a spherical level set inside the region of interest. Based on the position and the radius a 3D signed distance map is created. The voxels in this distance map represent the distance of the nearest surface point. The sign indicates whether the given voxel is inside or outside the spherical surface. During the region growing, in fact, the distance values are modified in each iteration step, so the zero-crossing level set surface is evolving until the desired segmentation mask is obtained. The distance values are modified depending on the voxel intensities and the curvature of the level set surface.

The data term of the distance update equation is defined as follows

$$D(\vec{p}) = \varepsilon - |I(\vec{p}) - T|, \tag{1}$$

where $\vec{p}$ is the voxel position, $I(\vec{p})$ is the voxel intensity at $\vec{p}$, and T is the target intensity. Parameter ε is set by the user during initialization. The data term is shown in Figure 1 as a function of the intensity.

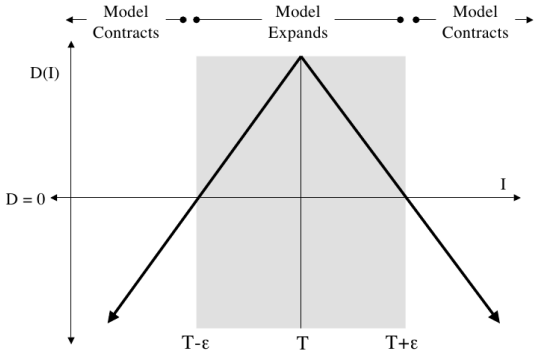

Figure 1: The data term depending on the voxel intensity [7].

The curvature term can be defined as

$$C(\vec{p}) = \nabla \cdot \frac{\nabla \phi(\vec{p})}{|\nabla \phi(\vec{p})|}, \tag{2}$$

where $\phi(\vec{p})$ represents the distance to the nearest surface point at position $\vec{p}$. Using Equations 1 and 2, the distance update equation is constructed as

$$\frac{\partial \phi(\vec{p})}{\partial t} = |\nabla \phi(\vec{p})| \Big[\alpha D(\vec{p}) + (1-\alpha) C(\vec{p}) \Big], \qquad (3)$$

where α is also a user defined parameter of the algorithm. Without the curvature term, the algorithm would be a simple flooding based on intensity, but using curvature calculations helps avoiding the leakage of the segmentation in ambiguous regions.

Since the update process locally modifies the distance values $\phi(\vec{p})$, it is necessary to recalculate $\phi(\vec{p})$ from time to time in order to consistently represent the distance to the evolved surface. The Fast Iterative Method (FIM) [6] is a convenient way to accomplish this, since it can be parallelized as well. Using FIM, the voxels are divided into three sets: source voxels, active voxels and far-away voxels, and their distance values are iteratively updated until they converge. Figure 2 illustrates the propagation of the distance calculation frontwave, where the black grid points are the source voxels with fixed distance value, the blue grid points are the active ones being updated, and the white points are considered as far-away voxels.

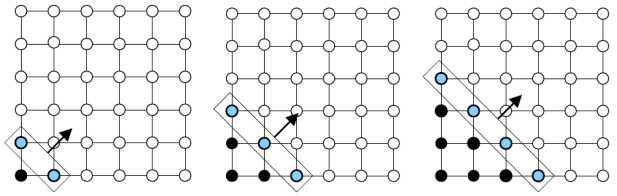

Figure 2: FIM frontwave propagation [6].

3.1 Implementation

The equations introduced previously need to be discretized in time and space as well. The implementation of the data term calculation is the easier. Since the user sets all parameters during initialization, we need to calculate the data term values only once. If we store these precalculated values in an array, we can access them later during the update steps. However, the curvature term calculation must be approximated. The applied approximation method is known as "difference of normals" [9].

Having both terms calculated, we need to update the distance values according to Equation 3. This needs to be discretized in time, thus we need to perform the following update steps:

$$\begin{aligned}
\phi(\vec{p})^{(t+1)} &= \phi(\vec{p})^{(t)} + \Delta\phi(\vec{p})^{(t)} \\
&= \phi(\vec{p})^{(t)} + \Delta t |\nabla \phi(\vec{p})^{(t)}| \Big[\alpha D(\vec{p}) + (1-\alpha) C(\vec{p}) \Big] \\
&= \phi(\vec{p})^{(t)} + \Delta t |\nabla \phi(\vec{p})^{(t)}| \left[\alpha D(\vec{p}) + (1-\alpha) \nabla \cdot \frac{\nabla \phi(\vec{p})^{(t)}}{|\nabla \phi(\vec{p})^{(t)}|} \right],
\end{aligned}$$

It is important to apply an appropriate "bravery" factor, to ensure accuracy and avoid having large steps that could ruin the segmentation. In this implementation we set $\Delta t = 5$ and the whole $\Delta\phi(\vec{p})^{(t)}$ can have a value of maximum 0.6.

The distances are recalculated after every third iteration of the level-set update. A distance recalculation consists of two parts. First, the positive distance values are set to infinity and classified as far-away voxels, and the ones with negative (or zero) distance are preserved and classified as source voxels. During the second part of the FIM, distance values are updated in several steps. Before an update step, we find active voxels, that is, a voxel having a source voxel as one of its first neighbors becomes active. Then the distance of each active voxel is updated based on the other active and source voxels in its neighborhood. A voxel becomes a source voxel if its distance has converged (has not changed significantly during the last update step). Therefore, in the next update step, we may find other active voxels.

3.2 Optimization

There are several ways to optimize the segmentation process. First of all, it can be implemented on the GPU exploiting its parallel computational power in the data term and curvature term calculations, and in the distance recalculation as well. Using narrow-band and sparse-field methods, we can even limit the calculations to the voxels that are near the surface[8].

Our implementation is based on a similar optimization. Since the computational bottleneck of the algorithm is the distance recalculation, the volume is divided into cubes of size $4 \times 4 \times 4$. Before each FIM update step, we can determine which cube contains voxels close to the surface. In these voxels accurate distance values are required. Since the update process of the FIM is performed multiple times, these cubes need permanent updating to decide whether they contain active voxels or not. This checking does not produce any significant overhead, but the distance recalculation becomes four to five times faster without the need to change the storage scheme.

4 Abstract Distance Transform

Our new method is designed to avoid segmentation leakage, but without expensive curvature calculations. The key idea is to calculate an abstract distance of each voxel from a user-defined seed point. Assume that a path of length N between the seed point $\vec{p}_0$ and an arbitrary voxel passes through voxels located at $\vec{p}_i$, where $i \in \{1,2,...,N\}$. The cost of the path is defined as

$$\sum_{i=1}^{N} |I(\vec{p}_i) - I(\vec{p}_{i-1})|, \qquad (4)$$

where $I(\vec{p})$ is the intensity of the voxel at position $\vec{p}$. In each voxel, we calculate the cost of the cheapest path from

the seed point. These abstract distance values represent the accumulated deviation along the path and not some kind of approximation of the Euclidean distances as in case of traditional distance transforms. Having the cheapest paths determined for each voxel, a 3D Abstract Distance Map (ADM) is obtained. In order to produce a binary segmentation mask, a simple thresholding is applied on the ADM.

The calculation of the ADM is, in fact, the classical problem of finding the shortest path in a weighted graph. Given two nodes u and v in a graph G, the length of a directed path from u to v is the sum of the weights of the edges along the path. The shortest path from u to v is a directed path which has minimal length among the possible paths from u to v in G. Now we can consider the distance $d(u,v)$ between u and v. This distance is 0 if $u = v$, it is ∞ if there is no path from u to v in G, otherwise the distance can be measured as the length of the shortest path from u to v. In general $d(u,v) \neq d(v,u)$ because the path is directed. This distance measurement is not always meaningful. Consider a situation when u and v is placed on a directed cycle, which has negative sum of weights. In this case we could achieve an arbitrarily small distance between the two nodes by going through the cycle multiple times. So it is required, that the given graph G does not contain a directed cycle with negative sum of weights.

Assume that $G = (V,E)$ is a directed graph, $c : E \to R^+$ is a cost function with non-negative values, and $s \in V$ is the source node. Our goal is to determine the value of $d(s,v)$ for each $v \in V$. In our application, V is the set of all the voxels, E is the set of edges between the adjacent voxels, and c is the difference between the intensities of two adjacent voxels.

The shortest path can be found using the classical Bellman-Ford algorithm [5]. This algorithm is described by the following pseudo code:

```
Step 1: initialize distances

  for each node v in V

    if v is seed
    then v.distance := 0
    else v.distance := infinity

Step 2: relax distances repeatedly

  for each node v in V

    for each edge uv in E

      u := uv.source
      v := uv.destination
      if u.distance + uv.weight < v.distance
      then v.distance := u.distance + uv.weight
```

This algorithm runs in $O(|V| \cdot |E|)$ time, where $|V|$ is the number of nodes and $|E|$ is the number of edges. $|E|$ is proportional to $|V|$ because of the geometrical structure of the volume, so the overall computational complexity is $O(|V|^2)$.

4.1 GPU Implementation

Because of the high amount calculations, it is worthwhile to implement our method on the GPU. An iteration of the distance relaxation can be calculated in a parallel way. We store the distance of each voxel in an array, and for each array element calculate the new distance by considering the distances to the adjacent voxels as the difference between the intensity values. In each iteration, it is decided whether the distance values have to be modified. If the distance value of a voxel is already minimal, the algorithm does not change it, but the adjacent voxels can potentially get a new, smaller value. If the distances do not change anymore, the iteration is terminated.

An iteration is calculated by the following code:

```
/*
input:  input array of distances
result: output array of distances
c:      termination condition
dimx, dimy, dimz: dimensions of the volume
*/
__global__ void cuda_BellmannFord(
  unsigned short* input,
  unsigned short* result, unsigned short* c,
  int dimx, int dimy, int dimz)
{
  long i = __umul24(blockIdx.x, blockDim.x)
           + threadIdx.x;
  long z = i / (dimx * dimy);
  long y = (i % (dimx * dimy)) / dimx;
  long x = (i % (dimx * dimy)) % dimx;

  if(x < dimx - 1 &&
     y < dimy - 1 &&
     z < dimz - 1 &&
     x > 0 && y > 0 && z > 0)
  {
    unsigned short u =
      input[x + y * dimx + z * dimx * dimy];
    unsigned short vv =
      tex3D(volumeTexture, x, y, z);

    unsigned short tmp = USHRT_MAX;

    unsigned short v;
    unsigned short w;

    int dd = 1;

    for(int dz = -dd; dz < dd+1; dz++)
      for(int dy = -dd; dy < dd+1; dy++)
        for(int dx = -dd; dx < dd+1; dx++)
        {
          if((z + dz) >= 0 && (z + dz) < dimz &&
             (y + dy) >= 0 && (y + dy) < dimy &&
             (x + dx) >= 0 && (x + dx) < dimy)
          {
            if(dx != 0 || dy != 0 || dz != 0)
            {
              v = input[(x + dx) + (y + dy) *
                dimx + (z + dz) * dimx * dimy];
              w = abs(vv - tex3D(volumeTexture,
                x + dx, y + dy, z + dz));
              tmp = v + w < tmp ? v + w : tmp;
            }
          }
        }
```

```
    if(tmp < u) c[0] = 1;

    result[x + y * dimx + z * dimx * dimy] =
      tmp < u ? tmp : u;
  }
}
```

In worst case, $|V|-1$ iteration steps have to be performed (the longest path in G can not be longer then $|V|-1$, an iteration updates each voxel, so the aggregated number of the started threads is $O(|V|^2)$), but the iteration can be terminated if the distance map does not change. The segmentation mask is obtained by an appropriate thresholding of the obtained distance map.

5 Results

We have tested our method for brain segmentation from CT and MRI data, and compared the results to that of the classical LS method. Figure 3 shows the obtained segmentation masks rendered by direct volume visualization. Note that the two algorithms provide almost the same results. However, our method is significantly faster to evaluate on the GPU. The running times measured on an NVIDIA GT335M graphics card are summarized in Table 1.

5.1 Accuracy of ADT

To evaluate the accuracy of our method, we compared our segmentation results with fifteen manually segmented MRI images. These segmentations were created by professional physicians and the data were provided by the IBSR project of the Center for Morphometric Analysis at Massachusetts General Hospital [1]. We used the conventional overlap metric for the comparisons, which is defined as $A/(B+C-A)$, where A is the number of voxels that are classified to belong to the segmentation mask by both the expert and the algorithm, B is the number of voxels assigned to the mask by the expert, while C is the number of voxels assigned to the mask by the algorithm. As a reference, the IBSR project provides results of other segmentation techniques and also compares the segmentations of two different experts. The accuracy measurements are summarized in Table 2.

6 Conclusion

In this paper, we have introduced a new and robust method for segmenting noisy CT and MRI data. Compared to the popular LS segmentation, our method provides similar results for a significantly lower computational cost. Using a fast GPU implementation, the calculation of our ADM is an order of magnitude faster that the GPU-accelerated iterative region growing based on the LS approach.

	CT Brain	MRI brain
Resolution	$256 \times 256 \times 159$	$256 \times 256 \times 60$
LS	2790 sec	1265 sec
ADT	151 sec	34 sec

Table 1: Running times of the brain segmentation.

	Average overlap	Description
ADT	0,562	based on 15 images
Expert	0,854	based on 4 images
Other	0,544	based on 20 images and 6 methods

Table 2: Overlap of different techniques.

So far we have tested our method only on a brain segmentation task. In our future work, we would like to make experiments on kidney and liver segmentation as well.

Acknowledgements

This work was supported by the project TÁMOP-4.2.2.B-10/1–2010-0009 and OTKA 101527. The first author is a grantee of the Öveges József Scholarship funded by GE Healthcare Hungary.

References

[1] The internet brain segmentation repository. `http://www.cma.mgh.harvard.edu/ibsr/`.

[2] Yuri Y. Boykov and Marie-Pierre Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *International Conference on Computer Vision*, pages 105–112, 2001.

[3] G.E. Christensen, S.C. Joshi, and M.I. Miller. Volumetric transformation of brain anatomy. *IEEE Transactions on Medical Imaging*, 16(6):864–877, 1997.

[4] L. Grady. Minimal surfaces extend shortest path segmentation methods to 3D. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(2):321–334, 2010.

[5] G. Ivanyos, L. Rónyai, and R. Szabó. *Algoritmusok*. TYPOTEX ELEKTRONIKUS KIADÓ KFT., 2008.

[6] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for a class of Hamilton-Jacobi equations on parallel systems. Technical Report UUCS-07-010, University of Utah, April 2007.

[7] A. Lefohn, J. Cates, and R. Whitaker. Interactive, GPU-based level sets for 3D brain tumor segmentation. In *Medical Image Computing and Computer Assisted Intervention*, pages 564–572, 2003.

[8] A. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):422–433, 2004.

[9] A. Lefohn and R. Whitaker. A GPU-Based, Three-Dimensional Level Set Solver with Curvature Flow. Technical Report UUCS-02-017, University of Utah, December 2002.

[10] Eric N. Mortensen and William A. Barrett. Interactive segmentation with intelligent scissors. In *Graphical Models and Image Processing*, pages 349–384, 1998.

[11] T. Pavlidis and Y.T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3):225–233, 1990.

[12] M.P. Persoon, I.W.O. Serlie, F.H. Post, R. Truyen, and F.M. Vos. Visualization of noisy and biased volume data using first and second order derivative techniques. In *Proceedings of IEEE Visualization*, pages 379–385, 2003.

[13] D. L. Pham, C. Xu, and J. L. Prince. Current methods in medical image segmentation. *Annual Review of Biomedical Engineering*, 2:315–337, 2000.

[14] J. B. T. M. Roerdink and A. Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41:187–228, 2000.

[15] James Albert Sethian. *Level set methods and fast marching methods: Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1999.

[16] M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1):146–165, 2004.

[17] L. Szirmay-Kalos and L. Szécsi. General purpose computing on graphics processing units. In A. Iványi, editor, *Algorithms of Informatics*, pages 1451–1495. MondArt Kiadó, Budapest, 2010. http://sirkan.iit.bme.hu/˜ szirmay/gpgpu.pdf.

[18] L. Szirmay-Kalos, L. Szécsi, and M. Sbert. *GPU-Based Techniques for Global Illumination Effects*. Morgan and Claypool Publishers, San Rafael, USA, 2008.

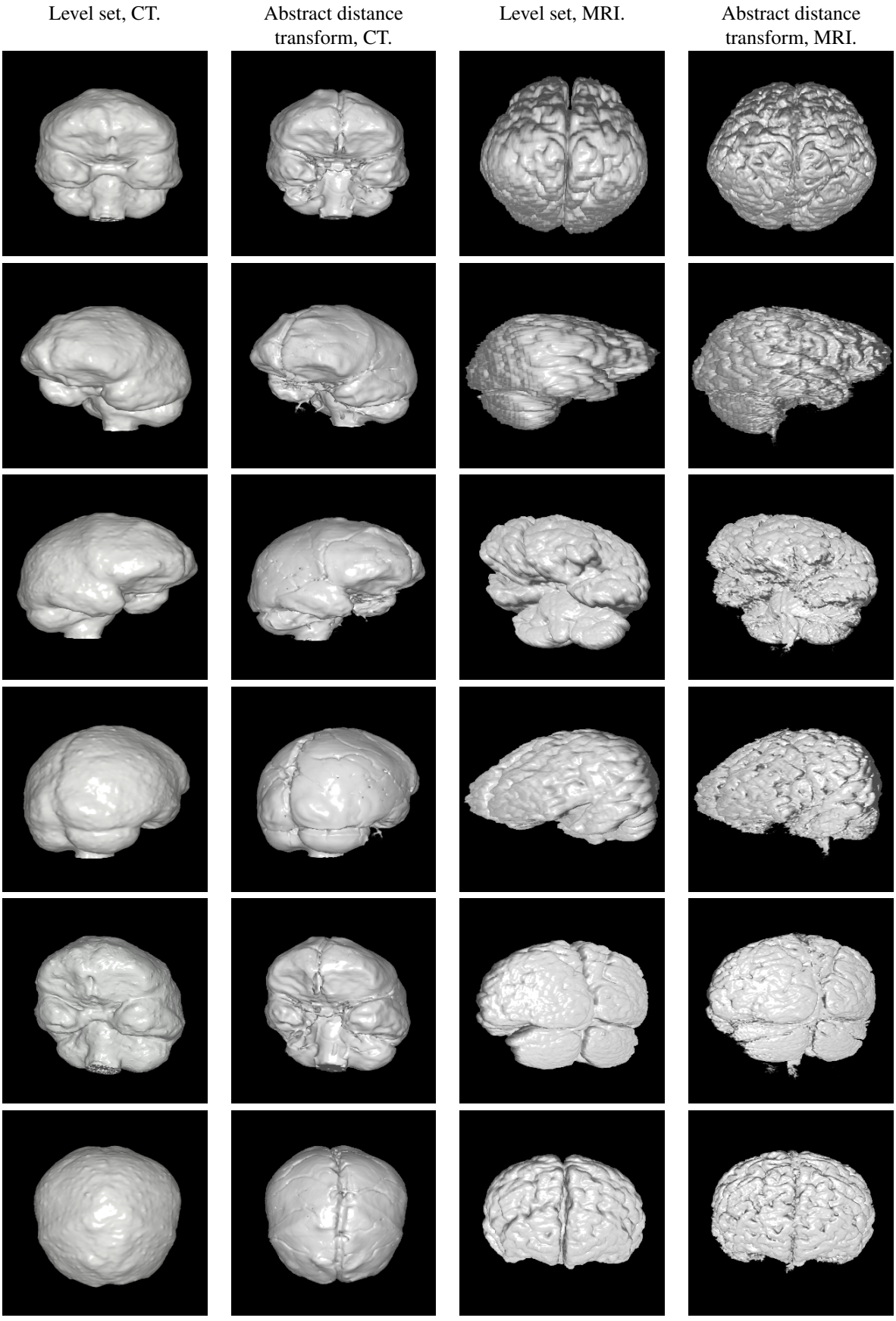

Figure 3: Brain segmentation from CT and MRI data using our abstract distance transform method and the classical level set method.