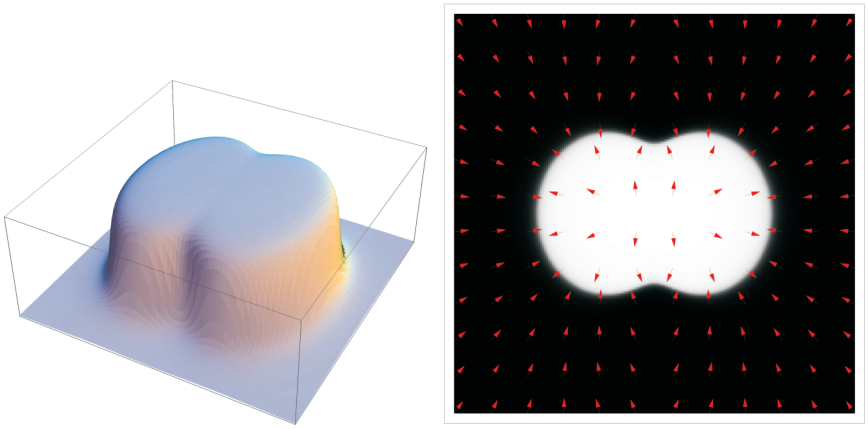

Figure 1. Left: A graph of the image intensity function $I^0(x)$. **Right**: Image given by the intensity $I^0(x)$ plotted together with arrows representing the vector field $-\nabla g(|\nabla I^0(x)|)$.

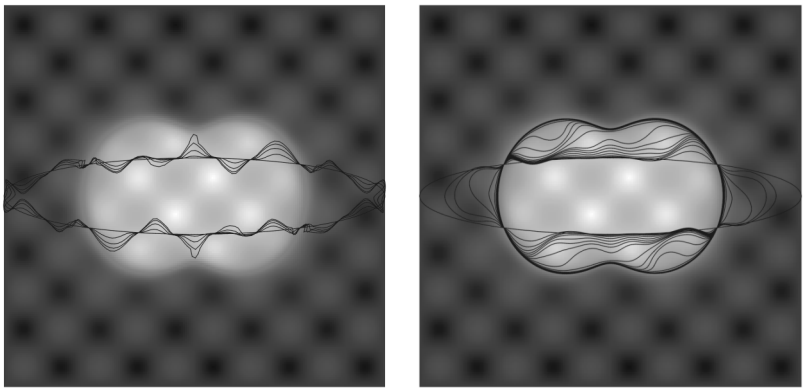

Figure 2. Evolution only by advection leads to attracting a curve (initial ellipse) to spurios edges, but adding a regularization term related to the curvature of evolving curve, the edge is found smoothly also in the case of a 2D noisy image (right).

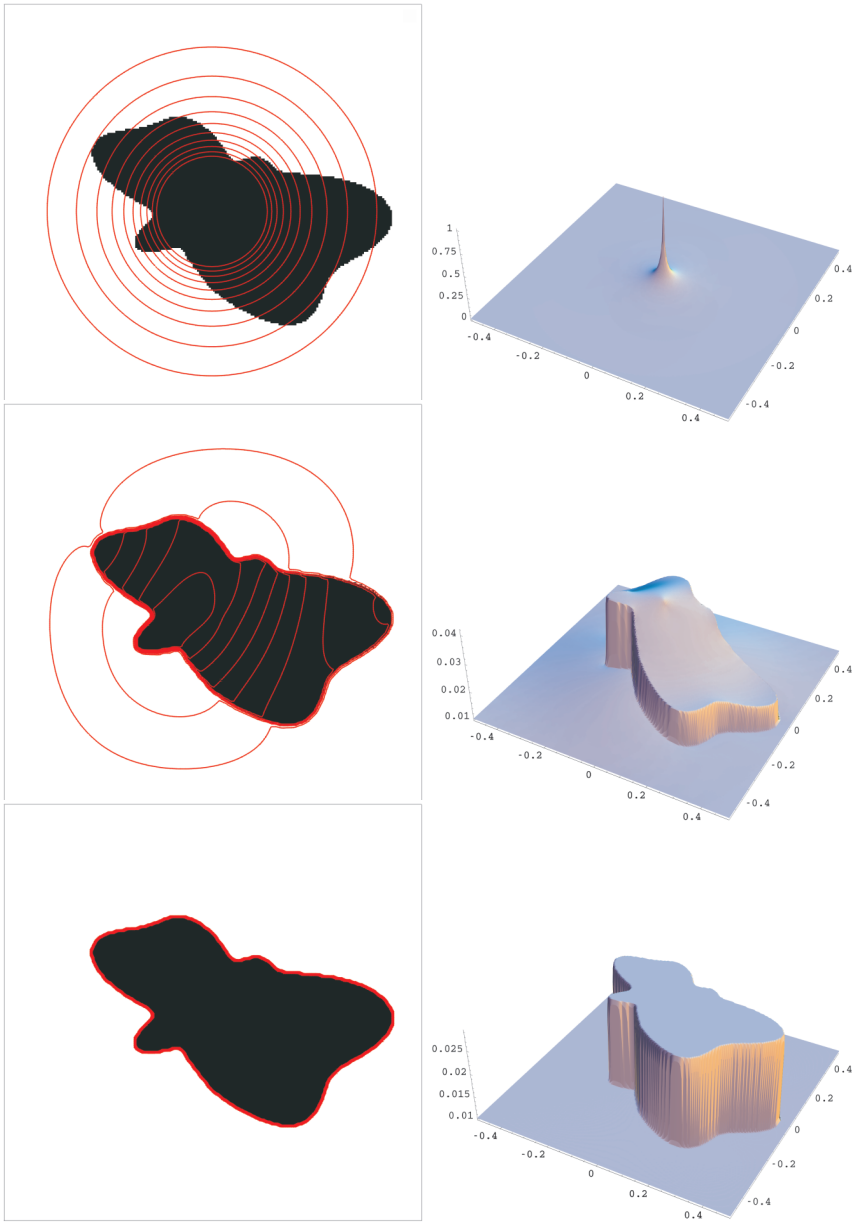

Figure 3. Subjective surface-based segmentation of a "batman" image. In the left column we plot the black-and-white image to be segmented together with isolines of the segmentation function. In the right column there is the shape of the segmentation function. The rows correspond to time steps 0, 1, and 10, which gives the final result. The regularization parameter $\varepsilon = 1$ is used in this example.

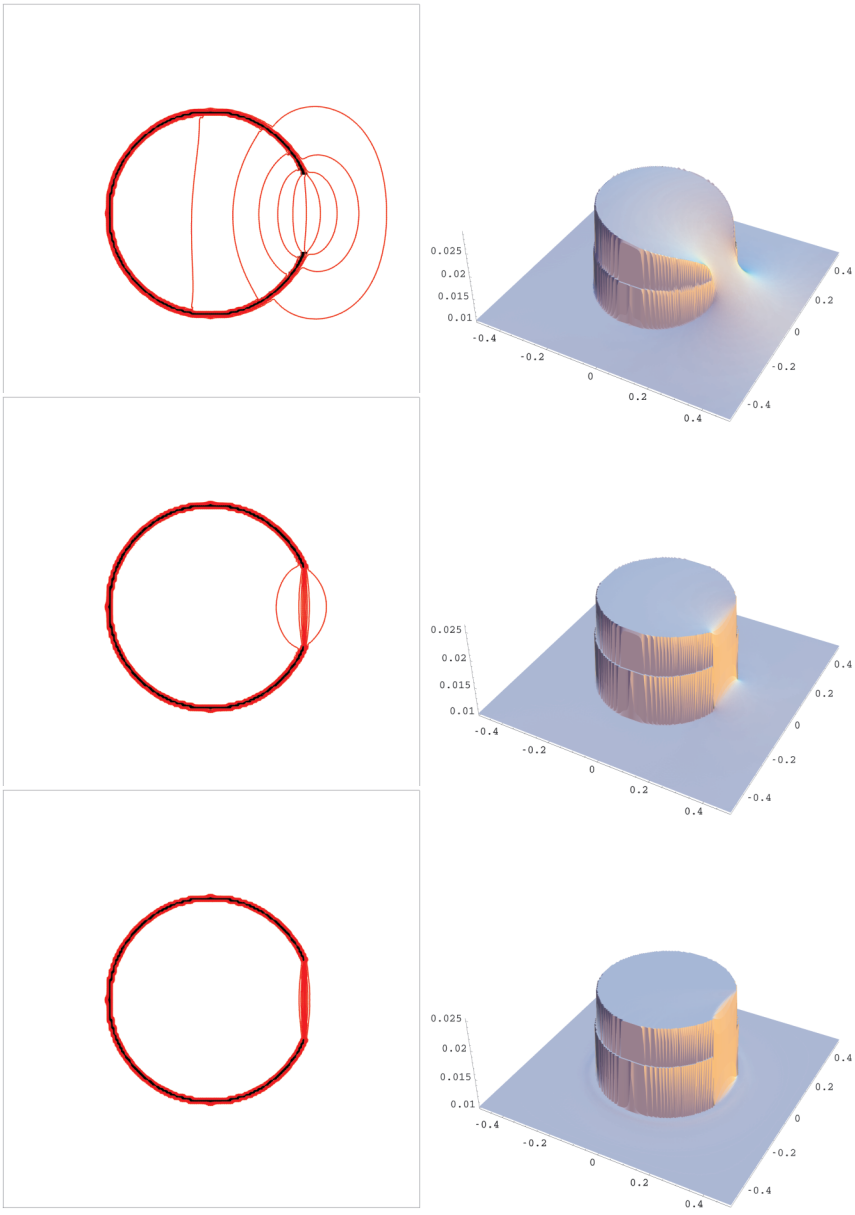

Figure 4. Segmentation of the circle with a big gap using $\varepsilon = 1$ (top), $\varepsilon = 10^{-2}$ (middle), and $\varepsilon = 10^{-5}$ (bottom). For a bigger missing part, a smaller ε is desirable. In the left column we see how closely to the edges the isolines are accumulating and closing the gap; on the right we see how steep the segmentation function is along the gap.

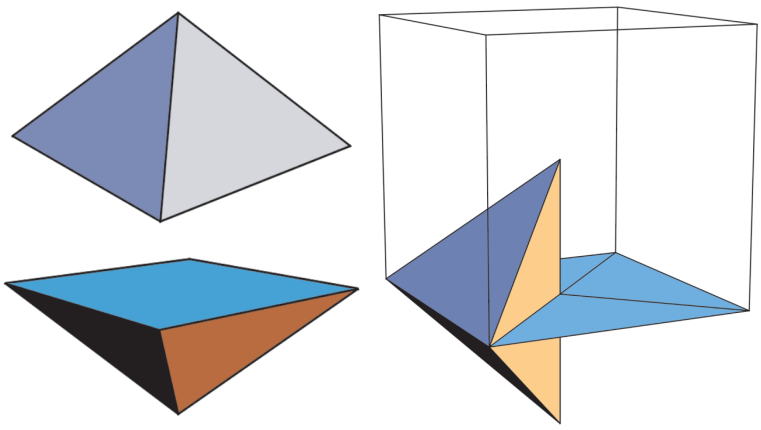

Figure 5. Neighbouring pyramids (left) that are joined together and which, after splitting into four parts, give tetrahedra of our 3D grid. We can the see intersection of one of these tetrahedra with the bottom face of the voxel co-volume (right).

```
#include <mpi.h>
int main(int argc, char *arvg[])
{
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
 .
 .
 MPI_Finalize():
}
```

Figure 6. Typical structure of an MPI parallel program.

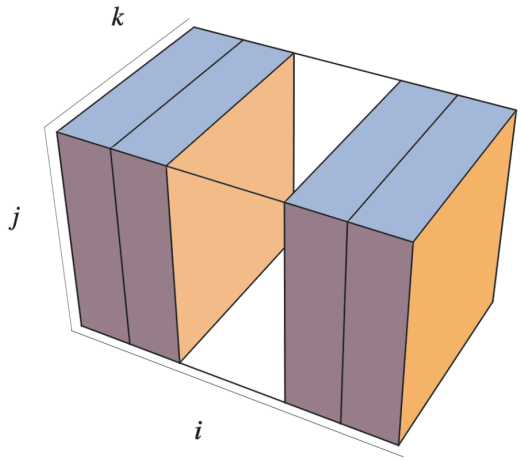

Figure 7. Splitting of a 3D image to n_{procs} 3D rectangular subareas, where n_{procs} corresponds to the number of processes involved in parallel execution.

```
#include <mpi.h>

int main(int argc, char *argv[])
{
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
 if (myid==0)
   {
   printf("tau, number of time steps:\n");
   scanf("%lf %d",&tau,&nts);
   }
 MPI_Bcast(&tau, 1, MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast(&nts, 1, MPI_INT,0,MPI_COMM_WORLD);
 ReadingImage();
 Coefficients0();
 InitialSegmentationFunction();
 for (k=1;k<=nts;k++)
   {
    change=EllipticStep();
    if (change < delta)
      {
       WritingSegmentationResult();
       break;
      }
   }
MPI_Finalize();
}
```

Figure 8. Main structure of our MPI parallel program for the 3D semi-implicit co-volume subjective surface segmentation method.

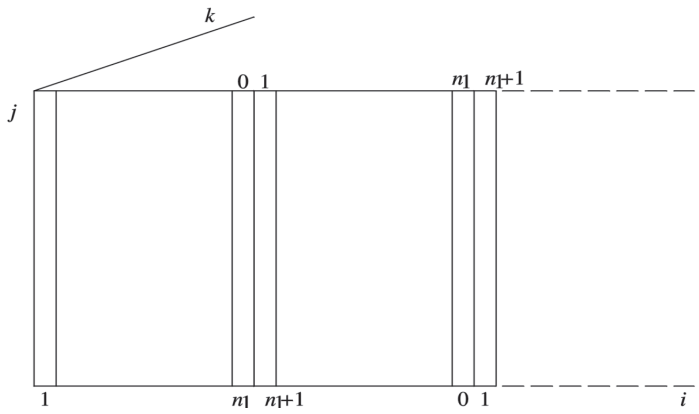

Figure 9. Data distribution and its overlap over parallel processes.

```
void ReadingImage()
{
 input=fopen("3Dimage.dat","r");

 for (n=0;n<nprocs;n++)
  {
    if (myid==n)
    {
     if(myid==0)
      {
       fseek(input,0,SEEK_SET);
       for(i=p;i<=n1+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    if((myid>0)&&(myid<nprocs-1))
      {
       fseek(input,(n1*myid-1)*(n2+1)*(n3+1),SEEK_SET);
       for(i=p-1;i<=n1+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    if (myid==nprocs-1)
      {
       fseek(input,(n1*myid-1)*(n2+1)*(n3+1),SEEK_SET);
       for(i=p-1;i<=n1last+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    }
  }
fclose(input);
}
```

Figure 10. Parallel reading of a 3D image.

```
for (i=p+1;i<=N1+p-1;i++)
for (j=p+1;j<=N2+p-1;j++)
for (k=p+1;k<=N3+p-1;k++)
  {
  z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
     ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
     an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
     at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
  u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);
  }
```

Figure 11. One iteration of the standard serial SOR method.

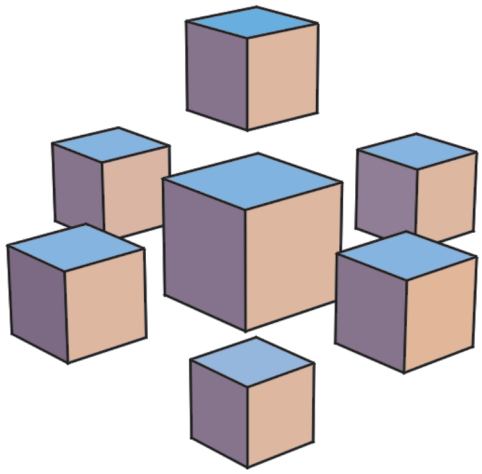

Figure 12. The RED element is in the middle, and its 6 neighbors (west, east, south, north, bottom, and top) have the sum of indices different by 1 from the middle one, so they are all BLACK elements.

```
if (myid==0)
{
 for (i=p+1;i<=n1+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
   {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
       ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
       an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
       at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
    u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
if ((myid>0)&&(myid<nprocs-1))
{
 for (i=p;i<=n1+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
  {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
      ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
      an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
      at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
   u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
if (myid==nprocs-1)
{
 for (i=p;i<=n1last+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
  {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
      ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
      an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
      at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
   u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
```

Figure 13. One iteration for RED elements in the parallel RED-BLACK SOR method. One iteration for BLACK elements differs only in the Boolean condition of the **if** command, which has the form $(i + j + k)\ \%\ 2\ ==\ 1$.

```
if (myid==0)
{
 MPI_Isend(&(u[n1][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req1);
 MPI_Irecv(&(u[n1+p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req2);
}
if ((myid>0)&&(myid<nprocs-1))
{
 MPI_Isend(&(u[n1][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req1);
 MPI_Isend(&(u[p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req2);
 MPI_Irecv(&(u[n1+p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req3);
 MPI_Irecv(&(u[0][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req4);
}
if (myid==nprocs-1)
{
 MPI_Isend(&(u[p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req1);
 MPI_Irecv(&(u[0][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req2);
}
MPI_Wait(&req1, &status);
MPI_Wait(&req2, &status);
if (myid > 0 && myid < nprocs-1)
{
 MPI_Wait(&req3, &status);
 MPI_Wait(&req4, &status);
}
```

Figure 14. Point-to-point communication after one RED-BLACK SOR iteration for RED elements. The same exchange of data is done also after one iteration for BLACK elements.

```
deltain=0;
if (myid==0)
  {
   for (i=p+1;i<=n1+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
if ((myid>0)&&(myid<nprocs-1))
  {
   for (i=p;i<=n1+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
if (myid==nprocs-1)
  {
   for (i=p;i<=n1last+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
MPI_Allreduce(&deltain,&tmp,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
deltain=tmp;
```

Figure 15. Computing the initial residual in parallel.

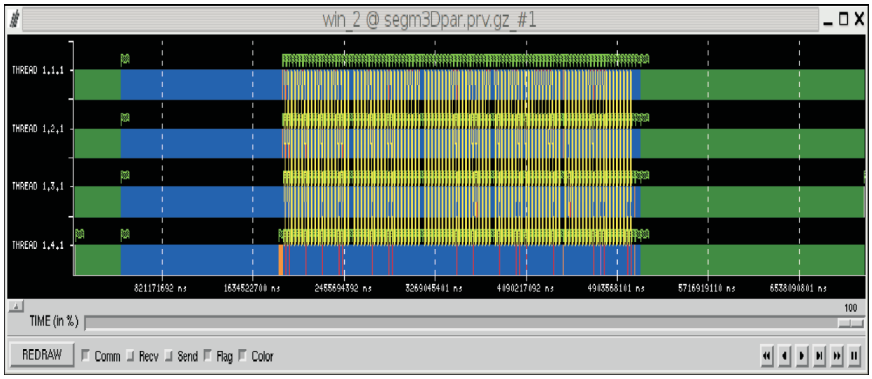

Figure 16. Visualization of the parallel run on 4 processors.

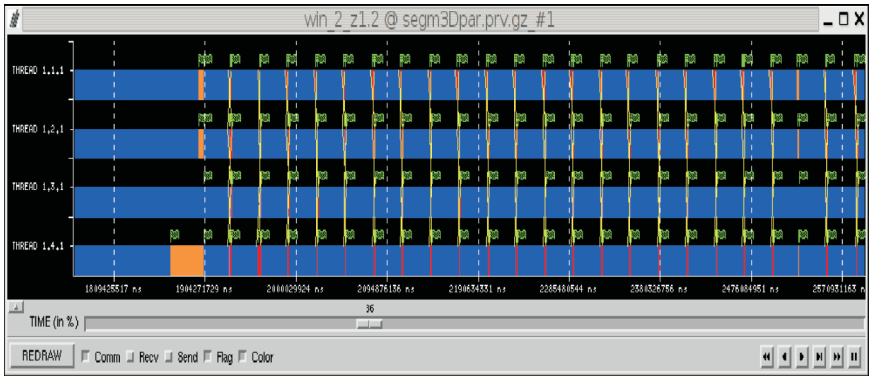

Figure 17. Zoom of the parallel run.

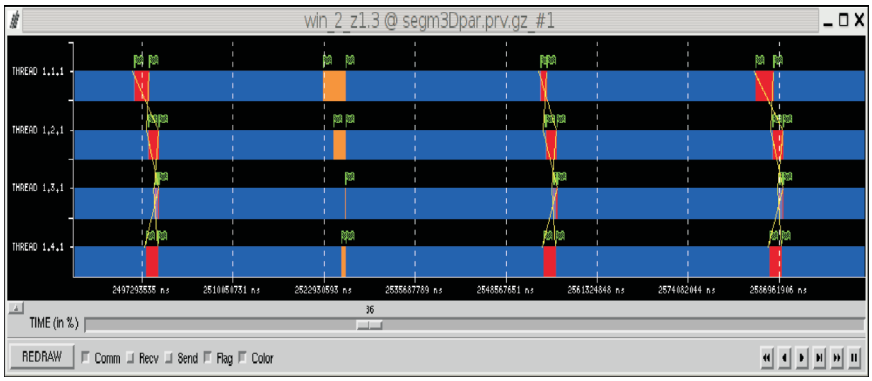

Figure 18. Further zoom of the parallel run.

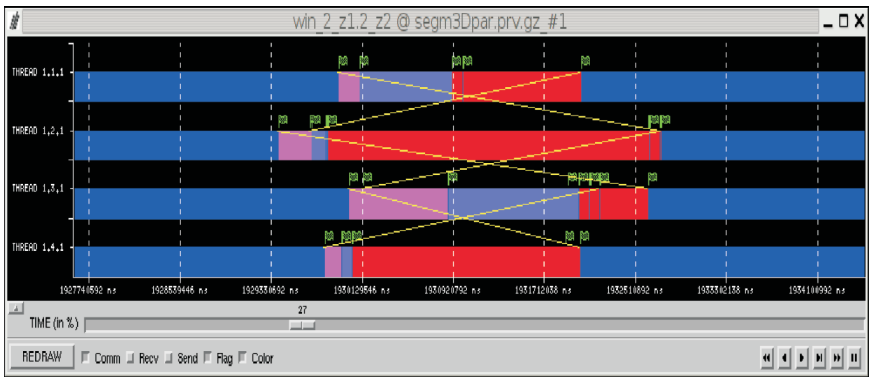

Figure 19. Final zoom of the parallel run.

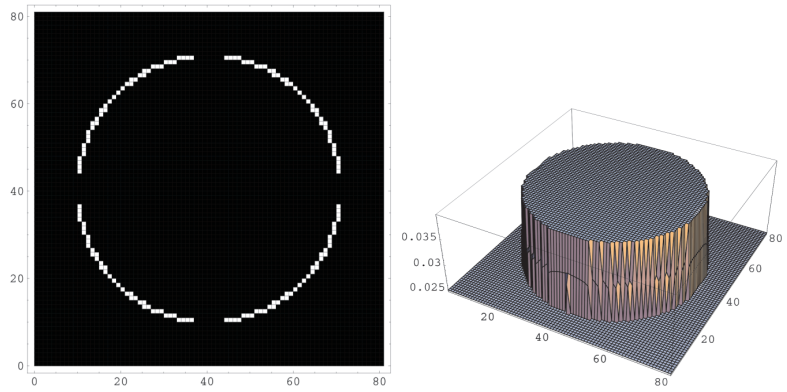

Figure 20. Subjective surface segmentation of 3D sphere with four holes.

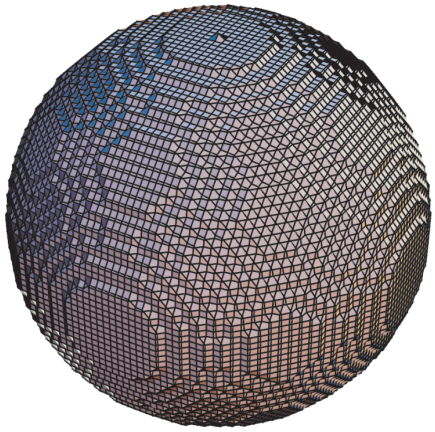

Figure 21. Result of subjective surface segmentation of a 3D sphere with four holes.

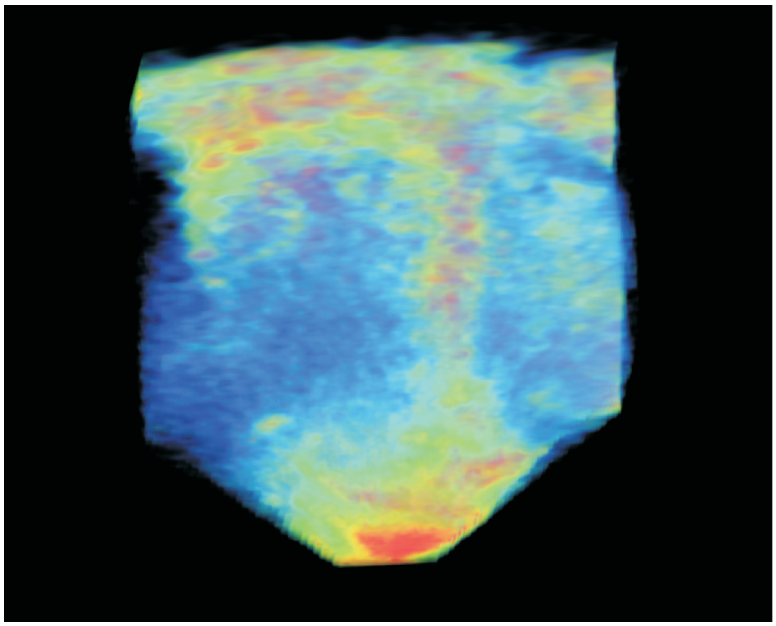

Figure 22. Volume rendering of the original 3D data set.

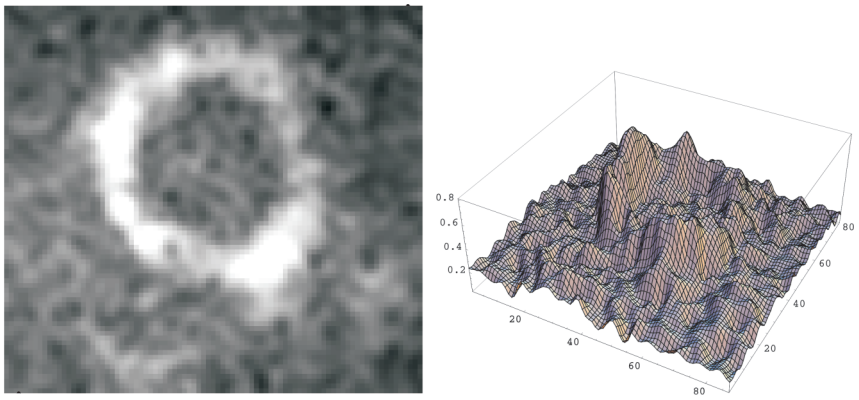

Figure 23. Plot of image intensity in slice $k = 130$ (left), and its 3D graphical view (right).

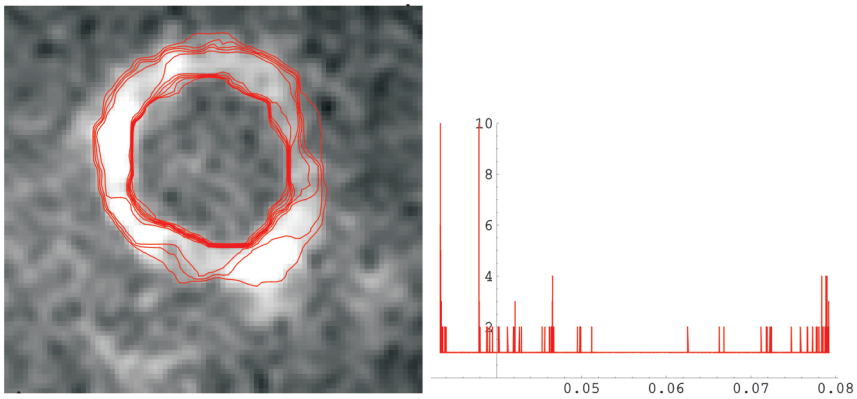

Figure 24. Plot of accumulated level sets in slice $k = 130$ (left); the histogram of the segmentation function in this slice (right).

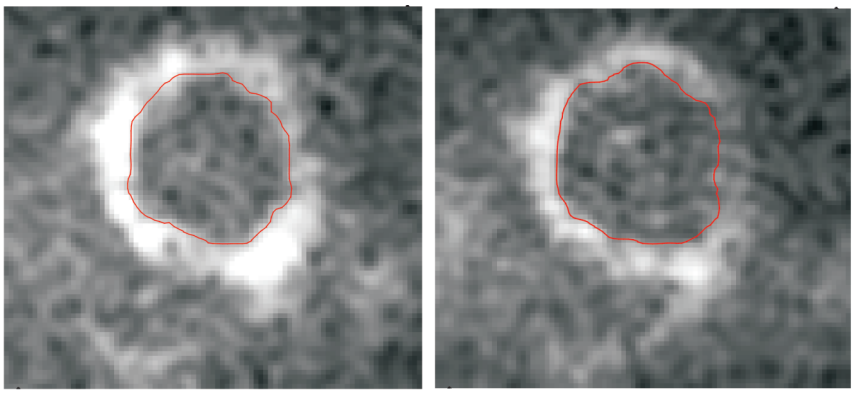

Figure 25. Plot of image intensity together with level line 0.052 in slices $k = 130$ (left) and $k = 125$ (right). Visualization of 3D surface in Figure 22 is done with the same level set.

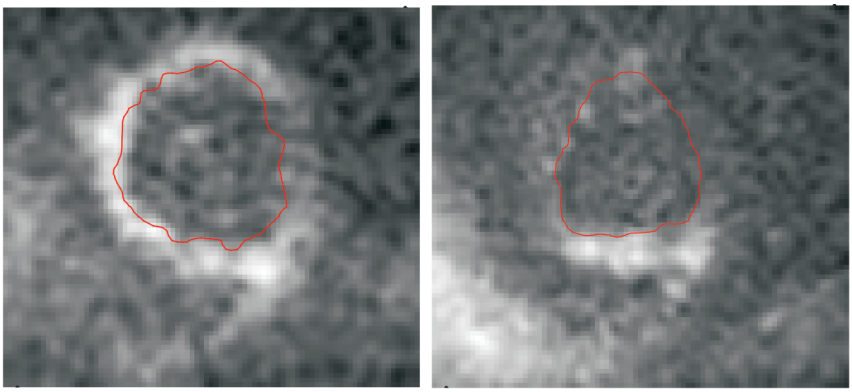

Figure 26. Plot of image intensity together with level line 0.052 in slices $k = 115$ (left) and $k = 100$ (right). Visualization of 3D surface in Figure 22 is done with the same level set.

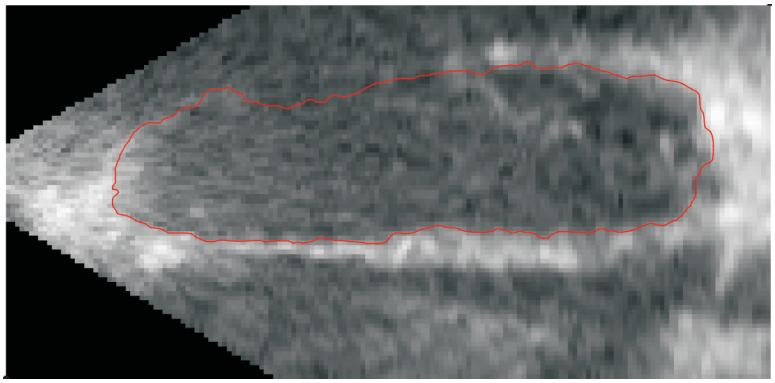

Figure 27. Plot of image intensity together with level line 0.052 in two other slices, $j = 40$. Visualization of 3D surface in Figure 22 is done with the same level set.

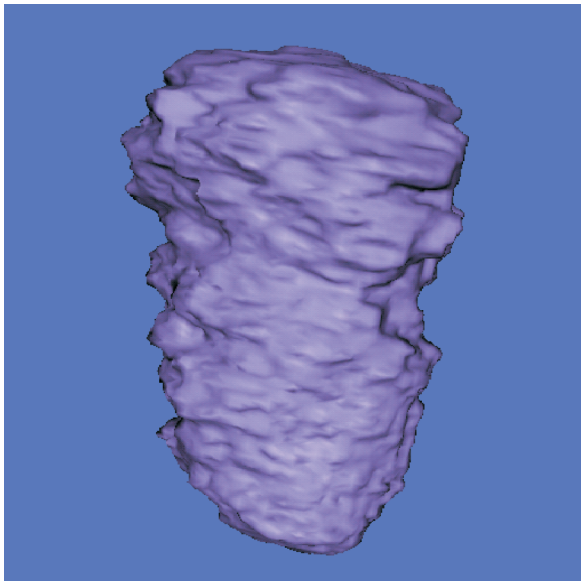

Figure 28. Isosurface visualization of the segmentation result for the left ventricle.