ECE 5470 Computer Vision Lab6 report

Three-Dimensional Image Processing

Weiran Wang

NetID: ww463

Three-dimensional Filtering

As instructed in the lab handout, I get started by running t1make script. The vgen3d command is used for generating 3D shapes and objects. The vop command combines with an or operation and we use the v3p command to convert the image to 3D polygons. Fig1 shows the outcome of running the t1make script. Then we can change the configuration of the 3d image by pressing c to bring up the option menu. Fig2 is the outcome by changing the render option to light mode.

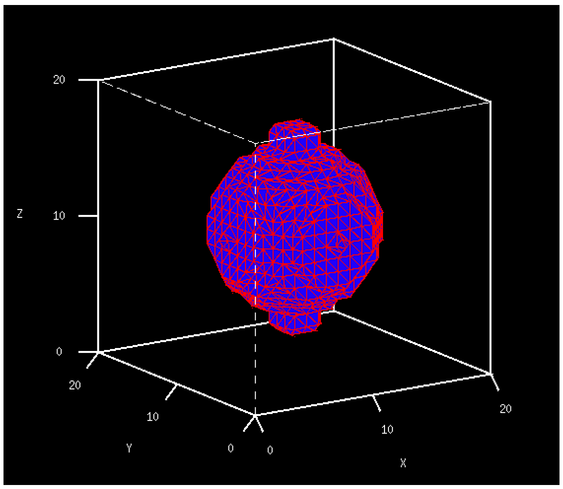

Fig1 t1.vd 3D object polygon

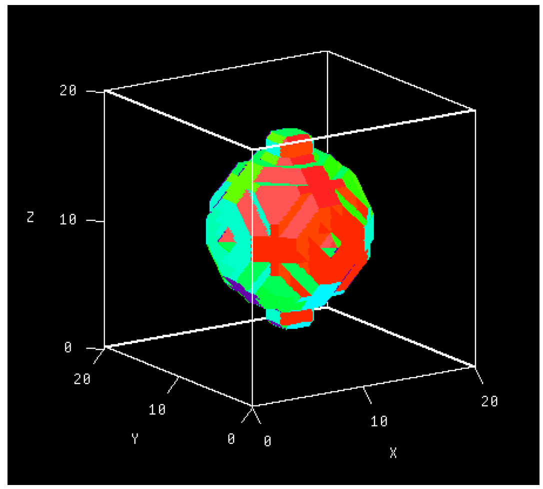

Fig2 t1.vd in light mode

The appearance of the polygon we generated is coarse, we can smooth the polygon by using v3tfilt command. Fig3 shows what it looks like after smoothing. By changing the color configuration to grey, we can better see the smoothing effect.

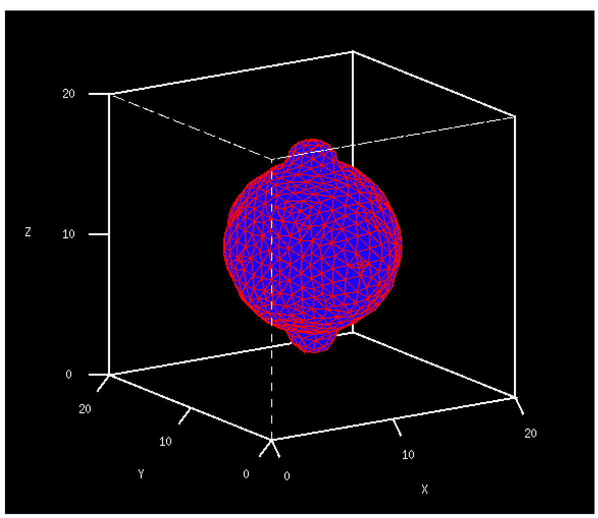

Fig3 t1f.vd 3D object polygon

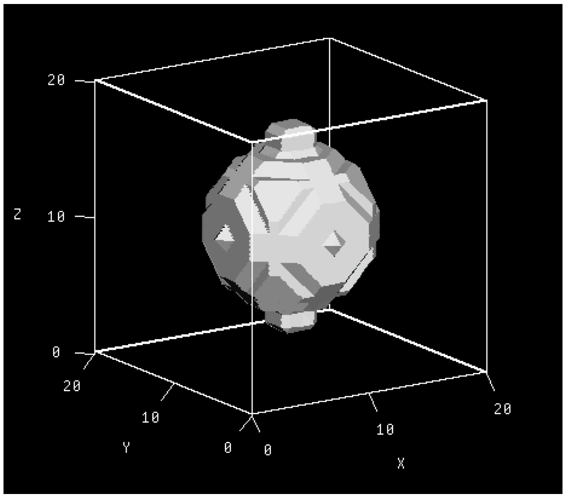

Fig4 t1.vd 3D object polygon in grey

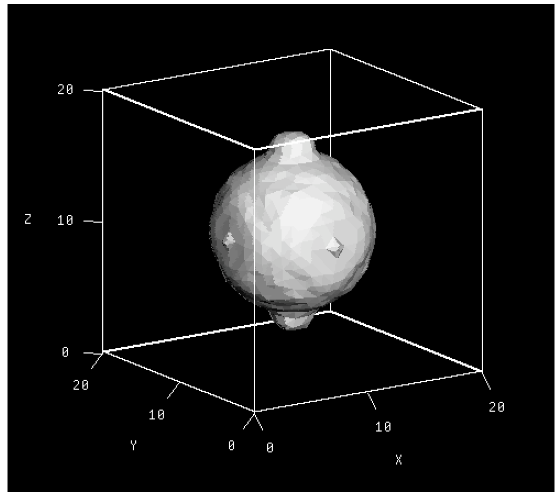

Fig5 t1f.vd 3D object polygon in grey

After the smoothing, the object loses sharp edges and looks more rounded. The bumps on the sphere look smaller than they were. We only want to perform this kind of action when we want to get rid of the sharp appearance of the object. If the intension is to preserve the details of the object, then smoothing should not be performed.

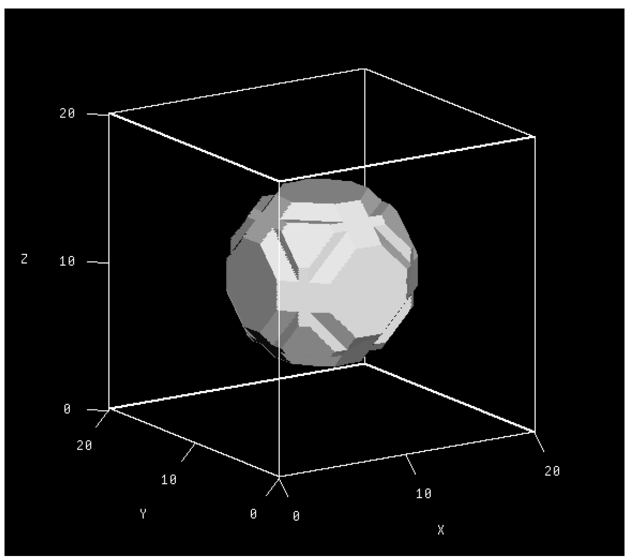

Fig6 t1.vd after protrusions removal

I tried to remove the protrusions on the 3D object by using the vmorph command. I set parameter s to be 3,3,3 and the result can be seen in fig6. This set of parameter works well for protrusion removal, however, the problem is it removes too much of the 3D object. It would better if we can remove only the protrusion and save as many details as possible.

Three Dimensional Edge Detector

In this section, we will be programming a 3d image filtering and edge detection algorithm. I get started by running t2make script to add some gaussian noise to the image t1n. The effect of adding Gaussian noise can be seen in Fig8. To handle the gaussian noise, I first apply a median filter before doing edge detection. For each pixel in the 3d image, the median filter take the median value of the a 3*3*3 cube. The edge detection was achieved by applying a kernel to to cube, the kernel looks like the following.

$$\underset{x-1}{\begin{bmatrix}-1,-2,-1\\-2,-4,-2\\-1,-2,-1\end{bmatrix}}\ \underset{x}{\begin{bmatrix}0,0,0\\0,0,0\\0,0,0\end{bmatrix}}\ \underset{x+1}{\begin{bmatrix}1,2,1\\2,4,2\\1,2,1\end{bmatrix}}\ \underset{y-1}{\begin{bmatrix}1\ ,2\ ,1\\0\ ,0\ ,0\\-1,-2,-1\end{bmatrix}}\ \underset{y}{\begin{bmatrix}2\ ,4\ ,2\\0\ ,0\ ,0\\-2,-4,-2\end{bmatrix}}\ \underset{y+1}{\begin{bmatrix}1\ ,2\ ,1\\0\ ,0\ ,0\\-1,-2,-1\end{bmatrix}}\ \underset{z-1}{\begin{bmatrix}-1,0,1\\-2,0,2\\-1,0,1\end{bmatrix}}\ \underset{z}{\begin{bmatrix}-2,0,2\\-4,0,4\\-2,0,2\end{bmatrix}}\ \underset{z+1}{\begin{bmatrix}-1,0,1\\-2,0,2\\-1,0,1\end{bmatrix}}$$

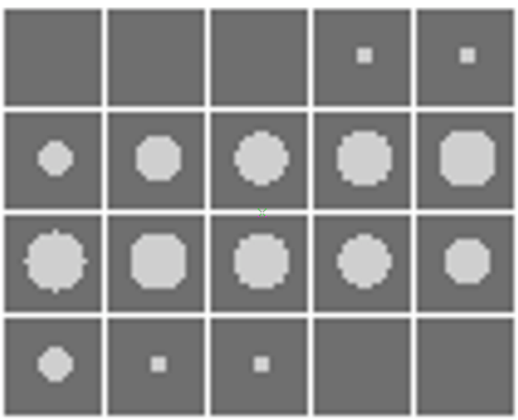

Fig6 t1g.vd

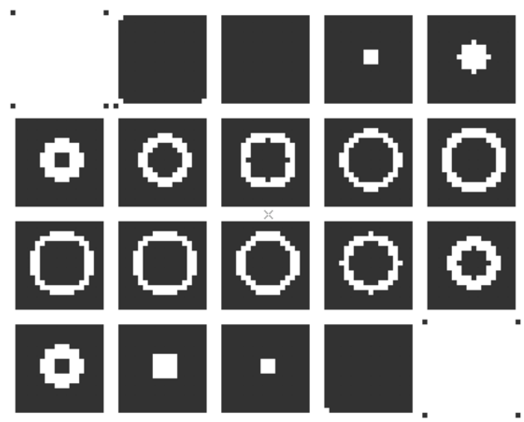

Fig7 edge detection output of t1g

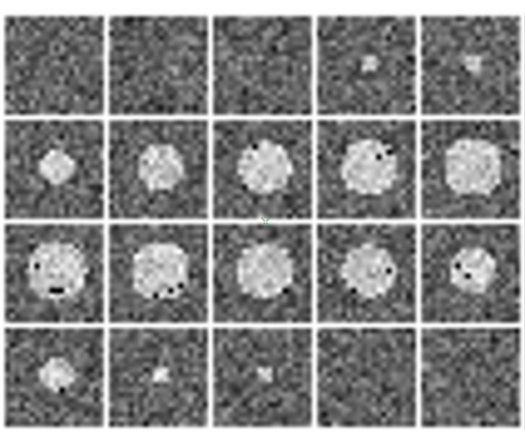

Fig8 t1n.vd

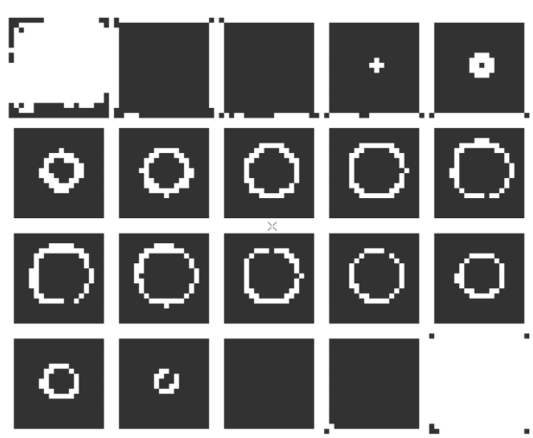

Fig9 edge detection output of t1n

As we can see from the results above, the edge detection result of t1g.vd is pretty good. The edges of the object are nice and clear. When it comes to the image t1n.vd, which has gaussian noise, the edge detection operator still works pretty good. The median filter is very important in this case, it's critical that we performed median filtering before the edge detection. For the edge detection algorithm, the idea is to compute the gradient of all three directions(x,y,z) and evaluate the total gradient magnitude, if the magnitude exceeds a certain threshold, we'll take it as the edge. In this case, I set the threshold to 70.

Program:

```
#include "VisXV4.h"          /* VisionX structure include file       */
#include "Vutil.h"           /* VisionX utility header files         */

VXparam_t par[] =            /* command line structure               */
{
{    "if=",    0,   " input file  v3dmean: compute local mean"},
{    "of=",    0,   " output file "},
{    "-v",     0,   " visible flag"},
{     0,       0,    0}
};
#define  IVAL   par[0].val
#define  OVAL   par[1].val
#define  VFLAG  par[2].val

int
main(argc, argv)
int argc;
char *argv[];
{
V3fstruct (im);
V3fstruct (tm);
int        x,y,z;              /* index counters                    */
int        xx,yy,zz;           /* window index counters             */
int        sum;
int        temp;
float grad;
```

```c

int gy[3][3][3], gx[3][3][3], gz[3][3][3];
int hx[3] = {1,2,1}, hy[3] = {1,2,1}, hz[3] = {1,2,1};
int hpx[3]={1,0,-1}, hpy[3]={1,0,-1},hpz[3]={1,0,-1};
int sumx,sumy,sumz;
int m,n,k;

    VXparse(&argc, &argv, par); /* parse the command line         */

    V3fread( &im, IVAL);        /* read 3D image                  */
    if ( im.type != VX_PBYTE || im.chan != 1) { /* check  format  */
       fprintf (stderr, "image not byte type or single channel\n");
       exit (1);
    }

    V3fembed(&tm, &im, 1,1,1,1,1,1); /* temp image copy with border */
    if(VFLAG){
       fprintf(stderr,"bbx is %f %f %f %f %f %f\n", im.bbx[0],
                 im.bbx[1],im.bbx[2],im.bbx[3],im.bbx[4],im.bbx[5]);
    }

for(m=0; m<=2; m++)
   for(n=0; n<=2; n++)
      for(k=0; k<=2;k++){
     gx[m][n][k] = hpx[m]*hy[n]*hz[k];
     gy[m][n][k] = hx[m]*hpy[n]*hz[k];
     gz[m][n][k] = hx[m]*hy[n]*hpz[k];
}
```

```
    for (z = im.zlo; z <= im.zhi; z++) {/* for all pixels */
      for (y = im.ylo; y <= im.yhi; y++) {
        for (x = im.xlo; x <= im.xhi; x++) {
             int arr[26];
       int i = 0;
             int j=0;
             for (zz = -1; zz <= 1; zz++) {/* compute the function */
               for (yy = -1; yy <= 1; yy++) {
                 for (xx = -1; xx <= 1; xx++) {
                     arr[i] = tm.u[z+zz][y+yy][x+xx];
         i++;
                 }
               }
             }
             for(i=0; i<27; i++){
    for(j=0; j<27; j++){
        if(arr[i] > arr[j]){
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
         }
    }
     }
    tm.u[z][y][x] = arr[13];

        }
      }
   }
```

```
    for (z = im.zlo; z <= im.zhi; z++) {/* for all pixels */
      for (y = im.ylo; y <= im.yhi; y++) {
        for (x = im.xlo; x <= im.xhi; x++) {
             sumx=0, sumy=0, sumz=0;
             for (zz = -1; zz <= 1; zz++) {/* compute the function */
               for (yy = -1; yy <= 1; yy++) {
                 for (xx = -1; xx <= 1; xx++) {
                     sumx += gx[zz+1][yy+1][xx+1]*tm.u[z-zz][y-yy][x-xx];
        sumy += gy[zz+1][yy+1][xx+1]*tm.u[z-zz][y-yy][x-xx];
                     sumz += gz[zz+1][yy+1][xx+1]*tm.u[z-zz][y-yy][x-xx];

                 }
               }
             }
             sumx/=16; sumy/=16; sumz/=16;
       temp = sqrt(sumx*sumx + sumy*sumy + sumz*sumz);
       im.u[z][y][x] = temp > 70?255:50;
        }
      }
   }
   V3fwrite (&im, OVAL);
   exit(0);
}
```

Three Dimensional Segmentation

In this section, I get started by marking the boundary of the tumor in the image sequence, the analseg script reads the boundary and generates a series of images. dcompare.vx, nodseg.vd, mregion.vd are shown in figure 10, 11, 12 respectively

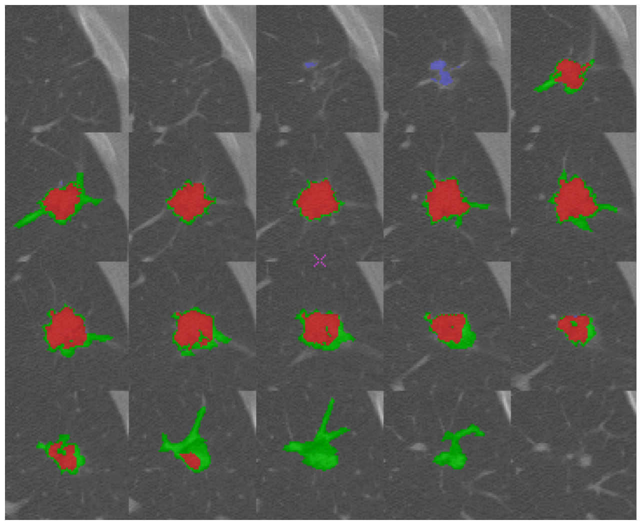

Fig10 dcompare.vx

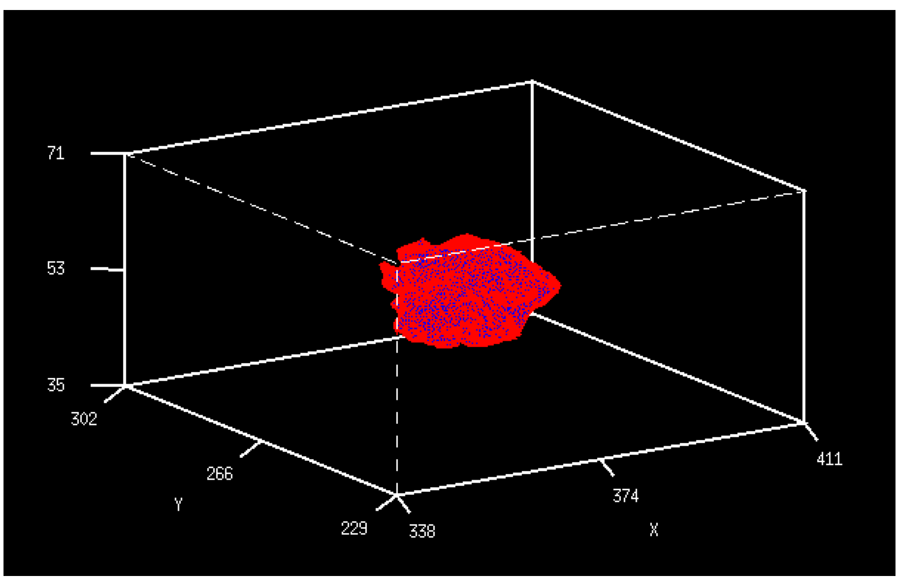

Fig11 nodseg.vd

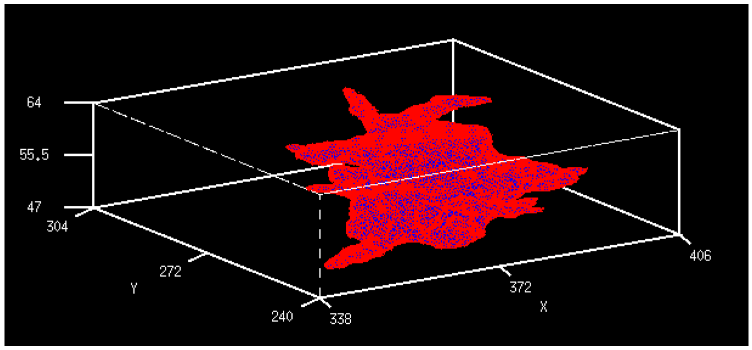

Fig12 mregion.vd

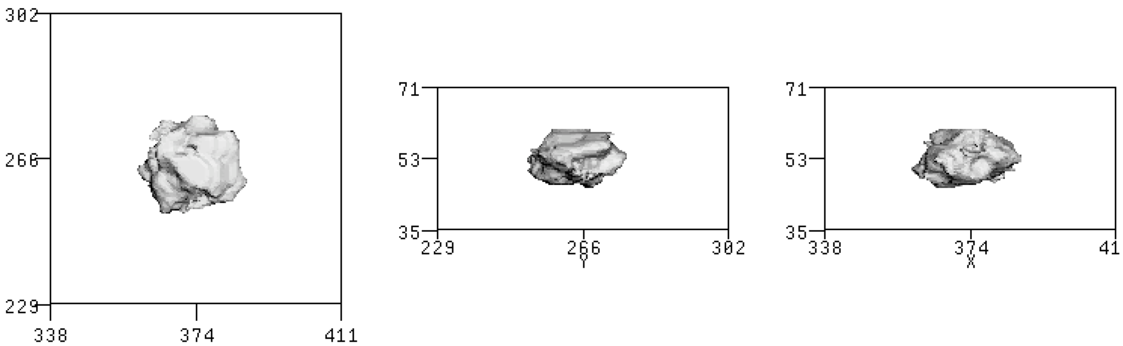

Fig13 dnodseg.vx

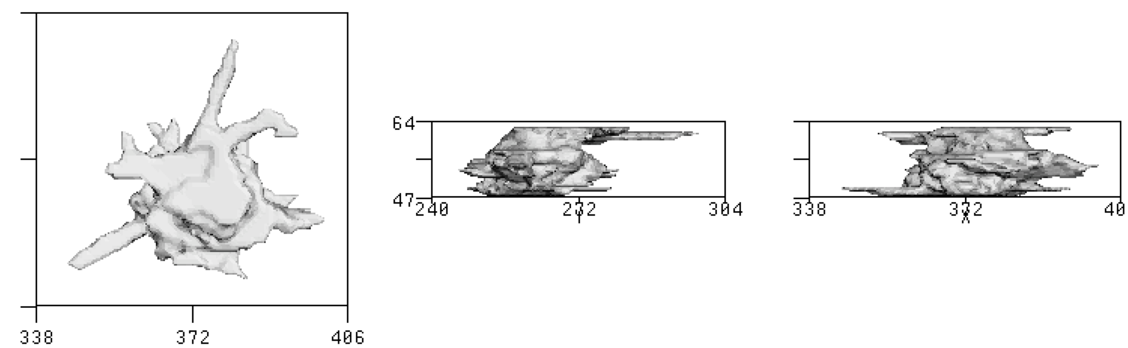

Fig14 dmregion.vx

```
apcnt   6886
bpcnt   3849
overlap 3733
anotb   3153
bnota   116
union   7002
diff    3269
tp      3733
tn      187470
fp      3153
fn      116
fpc     0.15069
sim     0.53313
sens    0.96986
spec    0.98346
jin     0.53313
dsc     0.69548
```

Fig15 compare.txt

From compare.txt we can evaluate the quality if the image segmentation by comparing the result with the ground. There are areas that I under-covered and overcovered, the segmentation result is not really good, the fraction of pixels correct is 0.15, which is the best I achieved in several times of boundary marking, the Jaccard similarity coefficient is 0.53, the Dice similarity coefficient is 0.69 which can be considered good.