Molecular illustration in black and white

David S. Goodsell and Arthur J. Olson

The Scripps Research Institute, La Jolla, CA 92037, USA

Two-dimensional image processing techniques are used to create black and white illustrations from conventional z-buffer images. The illustrations appear to be composed of lines—outlines, contour lines, and hatched shading—but are calculated as grey-scale raster images. They have the advantage of faithful reproduction in publication and ease of display on workstations and laser printers. A FORTRAN source code is provided.

Keywords: molecular illustration, publication graphics, 2-dimensional image processing, line drawing

INTRODUCTION

With the growing availability and simplicity of raster-based hardware, we have seen the steady decline of plotter images of molecules in the scientific literature. This is unfortunate, as a carefully produced ORTEP drawing really cannot be beat for the publication of a ball-and-stick model. Line-based images reproduce faithfully, unlike the gamble you take when publishing a photographed image, especially a colored image. Also, line-based drawings are easily comprehended, as we are familiar with black and white illustrations in our everyday life, in newspapers and books.

In this paper, we describe the application of two-dimensional (2D) image processing techniques to create black and white molecular illustrations. The illustrations appear to be made of lines, with outlines, contour lines, and hatched shading, but are calculated as gray-scale raster images, so they may be displayed on a workstation and printed on a laser printer. The images in this paper and in a previous paper[1] were published directly from laser printer hard copies. You may be the judge of the printed result.

Image processing starts with several conventional images of the molecular subject: a shaded surface image, a z-buffer, and if available, a shaded image with shadows. The simple FORTRAN program in Appendix 1 extracts outlines, contour lines, and hatch lines from these raster images.

Address reprint requests to Dr. Arthur J. Olson at The Scripps Research Institute, La Jolla, CA 92037, USA.
Received 22 July 1991; accepted 7 January 1992

METHODS

The recent publication of Saito and Takahashi[2] and the picture enhancement work of Namba *et al.*[3] give much of the necessary theory for the 2D image processing. We will briefly describe the processes, leaving most of the mathematical details to their paper and to the source code provided in Appendix 1.

Outlines and lines describing cusps and creases are extracted using the z-buffer information. A first order differential (g) is used for outlines:

$$g = (|A + 2B + C - F - 2G - H| + |C + 2E + H - A - 2D - F|)/8$$

where A–H are the depth values of the eight pixels surrounding pixel X:

$$\begin{matrix} A & B & C \\ D & X & E \\ F & G & H \end{matrix}$$

Using depth values in pixel units, this differential value ranges from 0 at the (flat) background, to values about 1 at the surface of typical molecules, to values much greater than 1 at edges. A simple threshold function is used to determine how much of the pixel is covered by an outline. If g is greater than a given upper limit, the pixel is completely covered and the pixel is colored black. If g is less than a given lower limit, it is uncovered and left white. For intermediate values, the amount covered is interpolated and the pixel is colored the proper grey. Allowing for fractional coverage provides some antialiasing; the final image is a gray-scale image, not strictly an image in black and white.

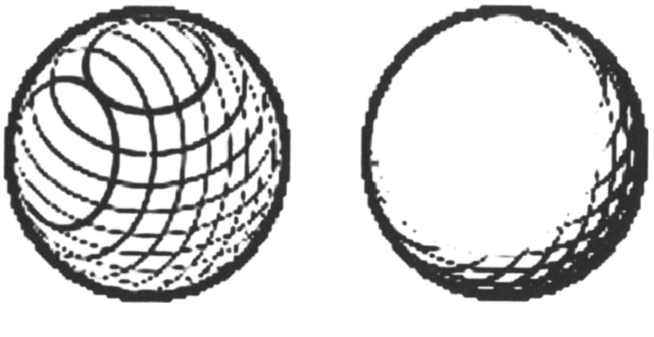
Cusps and creases in the surface are extracted with a second derivative of the depth:[1]

$$1 = |8X - A - B - C - D - E - F - G - H|/3$$

Again, a threshold is used to determine if the pixel is covered by a line. This function, however, forms two lines side-by-side for each feature. We apply a second step to fill in these lines. If more than five of the pixels around the central pixel have second derivative values greater than the threshold, the pixel is given the average value of the surrounding nonzero pixels.

Contour lines can add a great deal of shape information to an image of a surface. In Saito and Takahashi,[2] contour lines are drawn using information about the geometric model, such as radial lines around the center of rotation of a torus. This is not always convenient for the arbitrary shapes of molecules. We present a somewhat more general approach.

We draw contour lines at fixed intervals along the $x = -z$ line and the $y = -z$ line, then modulate the thickness of the lines by the local value of the shading of the surface—thick black lines for dark areas and thin black lines for light:



If the light source is placed in the upper left corner and contour lines are drawn only on the darker half of the sphere, a nice set of contour lines is obtained, removing the distracting circles on the upper left faces of the spheres.

Shadows and depth cuing are added using diagonal hatch lines. For shadows, lines are only drawn in parts where a shadowed image is darker than a similar shaded image without shadows. This also provides a simple way to highlight regions of a molecule, such as the heme in hemoglobin: one region is hatched and one left clear. Depth cuing is added with hatch lines perpendicular to the shadow lines, getting thicker on deeper surfaces.

APPLICATIONS

Black and white illustrations are effective in descriptive applications, for presenting the shape and form of molecules. The four illustrations included here show some of the range available with this simple technique.

Figure 1 shows the alpha subunit of hemoglobin. Tubes defining the α-carbon backbone and the molecular surface were calculated using MCS,[4] from coordinates in the Brookhaven Protein Data Bank listing 2HHB.[5] The heme is shown in a ball-and-stick model, slotted into the binding crevice. In the illustration, outlines define the shapes of surfaces and tubes, and contour lines enhance their three-dimensionality. Hatch lines, normally used for shadows, are used here to darken the inside of the molecular surface. (Depth cuing is not used.)

Figure 2 shows a 300-Å cube of a red blood cell, with the physiological concentration of molecules: 80 hemoglobin molecules, 1300 small molecules, and 3200 ions.[6,7] The only thing missing is 530,000 water molecules, each about the size of one of the ions, filling all of the gaps. The picture is drawn with outlines defining each molecule. Shadows and depth cuing are used to enhance the three-dimensional character of the image. (Contour lines are not used.)

Figure 3 shows an electron microscope reconstruction of HIV, data provided by Ulf Skoglund at Karolinska Institute and Stefan Hoglund at University of Uppsala. The starting shaded images were calculated using direct volume-rendering techniques.[8] Outlines bound the significant regions of density, contour lines create a dramatic shading, and depth cuing lines darken the background. (Shadow lines are not used.)

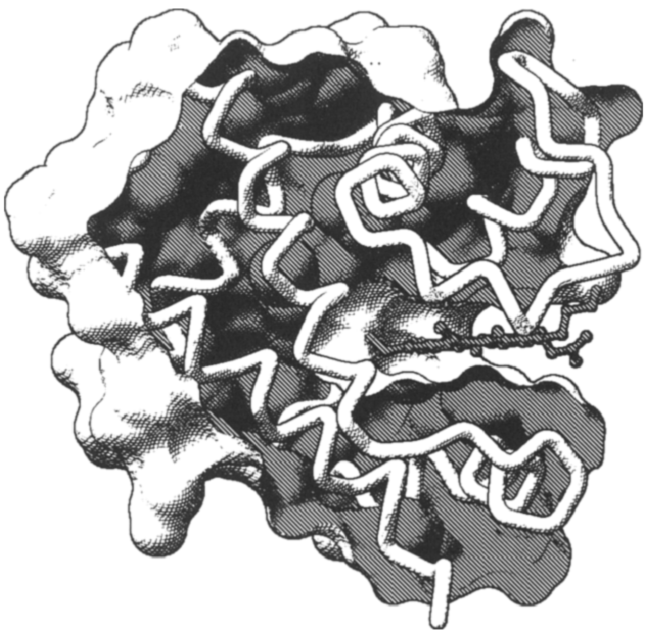


Figure 1. Alpha carbon backbone and molecular surface of the alpha subunit of hemoglobin.

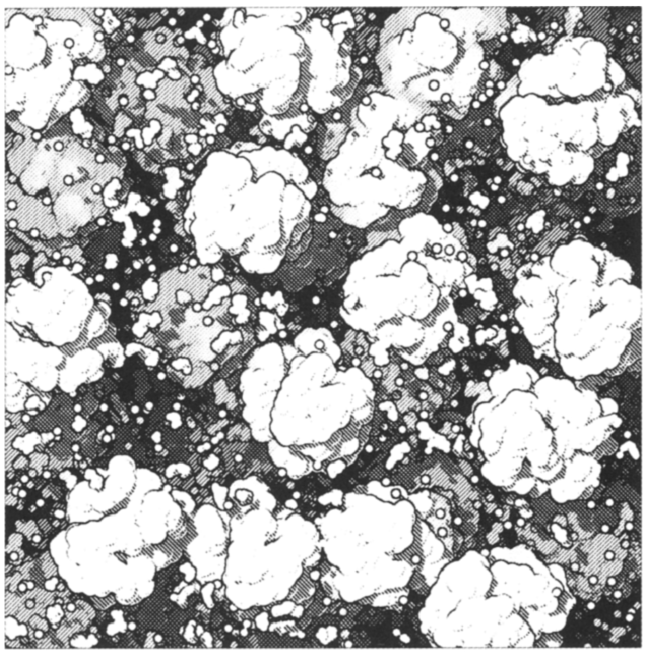


Figure 2. Interior of a human red blood cell, showing all molecules except water.

Figure 4 shows the electrostatic potential around superoxide dismutase. The Coulomb potential was calculated with a constant dielectric from coordinates in PDB listing 2SOD.[9] Shaded isocontour surfaces were calculated using direct volume rendering techniques.[8] Outlines are drawn around atoms and potential surfaces, contour lines are drawn on potential surfaces, and shadow hatch lines darken the negative potential surfaces. (Depth cuing is not used.)

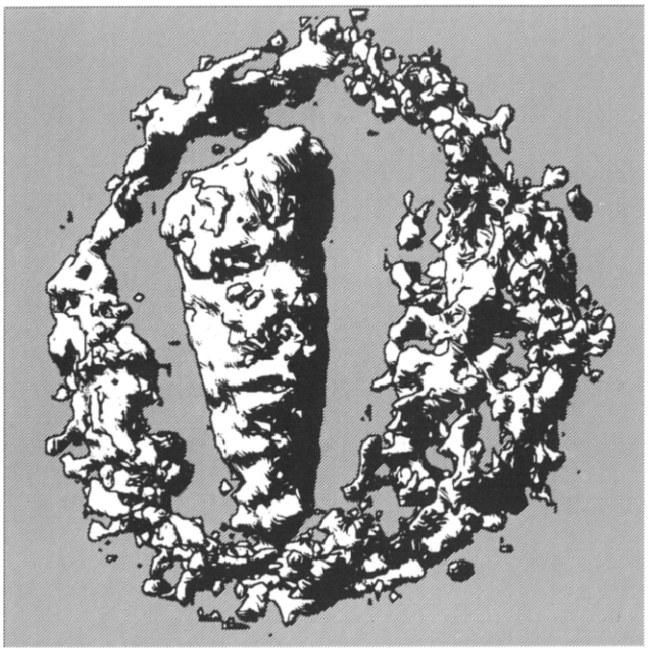


Figure 3. Electron microscope reconstruction of HIV, courtesy of Stefan Hoglund and Ulf Skoglund.

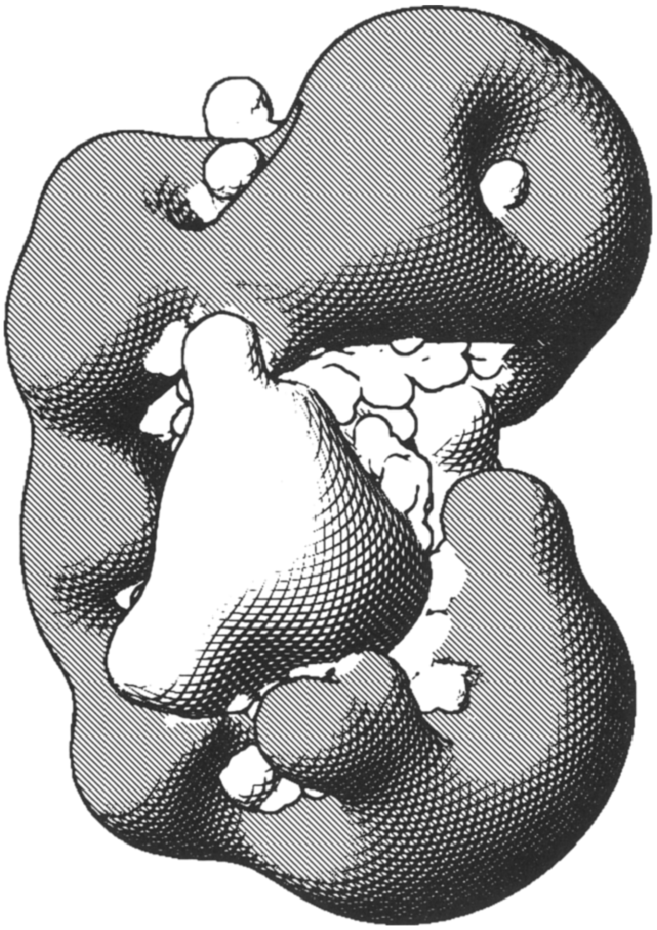


Figure 4. Electrostatic potential around superoxide dismutase.

IMPLEMENTATION

The figures were calculated at a resolution on the order of 900 pixels square and were printed on a Canon color laser copier. (This copier prints grey-scale images; a dithered laser printer hardcopy will be of slightly lower quality.) Image processing requies 1–2 minutes on a Convex C240 computer. The FORTRAN program is provided as Appendix 1. You will need to add routines to input the shaded image, the shadowed image and the *z*-buffer information, as well as writing out the final image. The parameters used to create the figures are described in Appendix 2.

ACKNOWLEDGMENTS

We wish to thank the reviewers for their helpful comments, and particularly to thank them for taking the time to implement and test the accompanying FORTRAN code. This work was funded in part by NIH Grant P01 GM38794-04 and a grant from Johnson and Johnson.

REFERENCES

1 Goodsell, D.S. and Olson, A.J. Molecular Illustration. *Pixel* 1991, **2**, 36–41

2 Saito, T. and Takahashi, T. Comprehensible Rendering of 3-D Shapes. *Comp. Graphics* 1990, **24**, 197–206

3 Namba, K., Caspar, D.L.D. and Stubbs, G. Enhancement and Simplification of Macromolecular Images. *Biophys. J.* 1988, **53**, 469–475

4 Connolly, M.L. Depth-buffer Algorithms for Molecular Modelling. *J. Mol. Graphics* 1985, **3**, 19–24

5 Fermi, G., Perutz, M.F., Shaanan, B. and Fourme, R. The Crystal Structure of Human Deoxyhaemoglobin at 1.74 Angstroms Resolution. *J. Mol. Biol.* 1984, **175**, 159–174

6 Ruch, T.C. and Fulton, J.F. *Medical Physiology and Biophysics*. W.B. Saunders Co., Philadelphia, 1961, 530–532

7 Best, C.H. and Taylor, N.B. *The Physiological Basis of Medical Practice*. Williams and Wilkins Co., Baltimore, 1961, 1–12

8 Goodsell, D.S., Mian, I.S. and Olson, A.J. Rendering of Volumetric Data in Molecular Systems. *J. Mol. Graphics* 1989, **7**, 41–47

9 Tainer, J.A., Getzoff, E.D., Richardson, J.S. and Richardson, D.V. Electrostatic Recognition Between Superoxide and Cu, Zn Superoxide Dismutase. *Nature* 1983, **306**, 284–287

APPENDIX 1

```
      real*4 back_intensity, line_intensity
      real*4 c_limit, c_spacing, c_width, s_spacing, s_width
      real*4 d_spacing, d_width_low, d_width_high, d_width_spread
      real*4 g_low, g_high, l_low, l_high
      real*4 value, distance, shade, width_pixel
      real*4 x_opacity, y_opacity, s_opacity, d_opacity
      real*4 g_opacity, l, l_opacity_ave, line_opacity
      real*4 shade_image(900,900), shadow_image(900,900), image(900,900)
      real*4 zl(900,900), l_opacity(900,900),zl_min,zl_max,zl_spread
c     --- read in parameters -------------------------------------------------
c     parameters are described in Appendix II
      read(5,*) back_intensity          !background intensity (0.=black,1.=white)
      read(5,*) line_intensity                !line intensity (0.=black,1.=white)
      read(5,*) c_limit                       !shade limit for contours (0. to 1.)
      read(5,*) c_spacing,c_width             !contour width and spacing (pixels)
      read(5,*) s_spacing,s_width         !shadow line width and spacing (pixels)
      read(5,*) d_spacing,d_width_high,d_width_low !depth cue parameters (pixels)
      d_width_spread=d_width_high-d_width_low
      read(5,*) g_low,g_high                                !outline parameters
      read(5,*) l_low,l_high                        !cusp and crease parameters
c***************************************************************************
c     Insert your routines for reading the shaded image into shade_image, shadowed
c     image into shadow_image, and z-buffer into zl. The images should have real
c     values with 0. = black and 1. = white, and the light source should be in the
c     upper left hand corner. The z information should have negative values away
c     from the viewer, in units of pixels, and the background set at z = -10000.
c***************************************************************************
c     --- find maximum and minimum z levels ----------------------------------
      zl_max = -100000.
      zl_min = 100000.
      do ix = 1,ixr
      do iy = 1,iyr
      zl_max = max(zl(ix,iy),zl_max)
      if (zl(ix,iy).ne.-10000.) zl_min = min(zl(ix,iy),zl_min)
      enddo
      enddo
      zl_spread = zl_max-zl_min
c     --- calculate second derivative for cusps and creases --------------------
      do ix = 2,ixr-1
      do iy = 2,iyr-1
     & l = abs(-zl(ix-1,iy-1)-zl(ix-1,iy)-zl(ix-1,iy+1)-
     &         zl(ix,iy-1)+8.*zl(ix,iy)-zl(ix,iy+1)-
     &         zl(ix+1,iy-1)-zl(ix+1,iy)-zl(ix+1,iy+1))/3.
      l_opacity(ix,iy) = min((l-l_low)/(l_high-l_low),1.)
      l_opacity(ix,iy) = max(l_opacity(ix,iy),0.)
      enddo
      enddo
c     *** begin 2-D processing ************************************************
      do ix = 2,ixr-1
      do iy = 2,iyr-1
c     --- initialize opacities -------------------------------------------------
      g_opacity = 0.        !opacities are 1. for completely opaque, black pixels
      l_opacity_ave = 0.
      x_opacity = 0.
      y_opacity = 0.
      s_opacity = 0.
      d_opacity = 0.
```

```
c  --- calculate first derivative for outlines ------------------------------
 & g = (abs(zl(ix-1,iy-1)+2.*zl(ix-1,iy)+zl(ix-1,iy+1)-
 &          zl(ix+1,iy-1)-2.*zl(ix+1,iy)-zl(ix+1,iy+1))+
 &      abs(zl(ix-1,iy-1)+2.*zl(ix,iy-1)+zl(ix+1,iy-1)-
 &          zl(ix-1,iy+1)-2.*zl(ix,iy+1)-zl(ix+1,iy+1)) )/8.
   g_opacity = min((g-g_low)/(g_high-g_low),1.)
   g_opacity = max(g_opacity,0.)
c  --- averaging cusps and creases -----------------------------------------
   l_opacity_ave = 0.
   rl = 0.
   do i = -1,1
   do j = -1,1
   l_opacity_ave = l_opacity_ave+l_opacity(ix+i,iy+j)
   if (l_opacity(ix+i,iy+j).gt.0.) rl = rl+1.
   enddo
   enddo
   if (rl.ge.6.) then
    l_opacity_ave = min(1.,l_opacity_ave/6.)
   else
    l_opacity_ave = l_opacity(ix,iy)
   endif
c  ==========================================================================
   if (zl(ix,iy).ne.-10000.) then   !contours calculated if not background ***
c  --- contour lines --------------------------------------------------------
   value = zl(ix,iy)-float(ix)                 !contours perpendicular to z = -x
   distance = abs(mod(value,c_spacing))             !distance to nearest contour
   distance = min(distance,c_spacing-distance)
   shade = min(shade_image(ix,iy)/c_limit,1.)       !shade determines line width
   x_opacity = (1.-shade)*c_width/2.-distance       !draw black if inside a line
c
   value = zl(ix,iy)-float(iy)                                  !same for z = -y
   distance = abs(mod(value,c_spacing))
   distance = min(distance,c_spacing-distance)
   shade = min(shade_image(ix,iy)/c_limit,1.)
   y_opacity = (1.-shade)*c_width/2.-distance
c  --- shadow hatch lines ---------------------------------------------------
c  this test determines which parts of the image are hatched:
   if (shadow_image(ix,iy).lt..95*shade_image(ix,iy)) then
    value = float(ix)-float(iy)                            !diagonal lines x = -y
    distance = abs(mod(value,s_spacing))
    distance = min(distance,s_spacing-distance)
    s_opacity = s_width/2.-distance
   endif
   endif                                             ! end of background test ***
c  --- depth hatch lines ----------------------------------------------------
   value = float(ix)+float(iy)                              !diagonal lines x = y
   distance = abs(mod(value,d_spacing))
   distance = min(distance,d_spacing-distance)
   if (zl(ix,iy).ne.-10000.) then
    d_opacity = ((zl(ix,iy)-zl_min)/zl_spread*d_width_spread+
 &              d_width_low)/2.-distance
   else
    d_opacity = d_width_low/2.-distance
   endif
c  ==========================================================================
   --- calculate intensity of pixel ---------------------------------------
   line_opacity = max(g_opacity,l_opacity_ave,x_opacity,
  &                     y_opacity,s_opacity,d_opacity,0.)
     line_opacity = min(line_opacity,1.)
     image(ix,iy) = back_intensity-(back_intensity-line_intensity)*line_opacity
     image(ix,iy) = min(image(ix,iy),1.)
     enddo
     enddo
c    *** end 2-D processing ************************************************
c    ***********************************************************************
c    Insert your routine for writing the final image. The values in image are
c    reals, with 0. = black and 1. = white.
c    ***********************************************************************
     end
```

APPENDIX 2

Parameter No.		Description of Parameters
1.	back__intensity	Intensity of the background, 0. for black, 1. for white. Normally 1.
2.	line__intensity	Intensity of lines, 0. for black to 1. for white. Normally 0.
3.	c__limit	This parameter determines how the shaded image is translated into line widths. The shaded image has values of 1. for white and 0. for black. Lines will be drawn with maximal thickness where the shaded image is black (0.) and will taper to zero width at shade values of c__limit. For spheres that shade from black to white, this value is typically in the range of .5 to .8.
4.	c__spacing	The distance, in pixels, between contour lines. Typically in the range of 6. to 20., larger if the surfaces are large and smooth, smaller for rapidly changing surfaces. Should not be set to zero; to remove contours, set c__width to 0.
5.	c__width	Maximal thickness of lines, in pixels. Typically one-half to one times c__spacing.
6.	s__spacing	Spacing of the hatch lines used for shadows, in pixels. Typically 5. to 10. Should not be set to zero; to remove shadows lines, set s__width to zero.
7.	s__width	Width of shadow hatch lines, in pixels. Typically one half of the s__spacing.
8.	d__spacing	Spacing of the depth-cuing lines, in pixels. Typically 5. to 10. Should not be set to 0; to remove depth cue lines, set both d__widths to 0.
9.	d__width__high	Width of the depth-cuing lines, at the surfaces closest to the viewer, in pixels. Typically set to 0.
10.	d__width__low	Width of the depth cuing lines, at the most distant surfaces, in pixels. Typically set equal to d__spacing, making the background completely black.
11., 12.	g__low, g__high	Parameters for drawing outlines. Pixels with values for *g* (the first derivative) greater than g__high will be black; those with values lower than g__low will be white; and those in between will be levels of grey. With depths in units of pixels, these are typically on the order of 2000. and 3000. If you are getting lots of extraneous lines, increase these parameters, and the *l* parameters below.
13., 14	l__low, l__high	Parameters for drawing cusps and creases from values of the second derivative. Similar to g parameters above. Typical values are 2. and 7.

Parameters for Figures

Parameter No.	Figure 1	Figure 2	Figure 3	Figure 4
1.	1.0	1.0	1.0	1.0
2.	0.0	0.0	0.0	0.0
3.	0.6	0.1	0.7	0.7
4., 5.	6.0, 6.0	5.0, 0.0	20.0, 22.0	10.0, 10.0
6., 7.	6.0, 4.0	6.0, 4.0	6.0, 0.0	5.0, 3.0
8., 9., 10.	4.0, 0.0, 0.0	6.0, 0.0, 12.0	4.0, 0.0, 2.0	10.0, 0.0, 0.0
11., 12.	16000., 17000.	1., 2000.	20000., 21000.	20000., 21000.
13., 14.	5000., 10000.	5., 10.	10., 20.	1., 10.