Behav Res
DOI 10.3758/s13428-011-0127-3

Probing pictorial relief: from experimental design to surface reconstruction

Maarten W. A. Wijntjes

Abstract The perception of pictorial surfaces has been studied quantitatively for more than 20 years. During this time, the "gauge figure method" has been shown to be a fast and intuitive method to quantify pictorial relief. In this method, observers have to adjust the attitude of a gauge figure such that it appears to lie flat on a surface in pictorial space. Although the method has received substantial attention in the literature and has become increasingly popular, a clear, step-by-step description has not been published yet. In this article, a detailed description of the method is provided: stimulus and sample preparation, performing the experiment, and reconstructing a 3-D surface from the experimental data. Furthermore, software (written in PsychToolbox) based on this description is provided in an online supplement. This report serves three purposes: First, it facilitates experimenters who want to use the gauge figure task but have been unable to design it, due to the lack of information in the literature. Second, the detailed description can facilitate the design of software for various other platforms, possibly Web-based. Third, the method described in this article is extended to objects with holes and inner contours. This class of objects have not yet been investigated with the gauge figure task.

Keywords Depth perception · Pictorial space · 3D shape

Introduction

Almost 20 years ago, Koenderink, van Doorn, and Kappers (1992) published a key study on quantifying the perceived 3-D structure of a pictorial surface. Their experimental task was to adjust the attitude of a gauge figure probe. The gauge figure consists of a circle with a rod that sticks out perpendicularly from the middle. Observers are instructed to manipulate the 3-D orientation of this probe such that the disk appears to lie flat on the pictorial surface, while the rod consequently sticks out in the normal direction. Since then, numerous researchers have used this paradigm to study visual perception or 3-D shape. Yet the community of scientists using this method has been limited to those who understand the underlying mathematics and are able to experimentally implement this method. Detailed documentation has never been published in a complete fashion. This article will explain in detail all steps of the procedure. Furthermore, software written for PsychToolbox (Brainard, 1997; Pelli, 1997) is made available that covers all of these steps and should lead to an easily usable procedure, by means of which any user of PsychToolbox can conduct gauge figure experiments.

M. W. A. Wijntjes (✉)
Perceptual Intelligence Lab, Faculty of Industrial Design,
Delft University of Technology,
Landbergstraat 15,
2628 CE, Delft, The Netherlands
e-mail: m.w.a.wijntjes@tudelft.nl
URL: www.maartenwijntjes.nl/gaugefigure

Method

The procedure for running an experiment is visualized in Fig. 1. Each of the steps can be described as follows.

Contour selection After selecting a stimulus image, the experimenter needs to select which part of the pictorial surface is to be used for the experiment.

Triangulation Within the contour, measurement samples need to be defined. This is done using a triangulation grid.

Experiment After it is set up, the experiment can be conducted. The gauge figure probe should be rendered in the picture, and the observer should be able to manipulate its attitude.

Published online: 26 July 2011

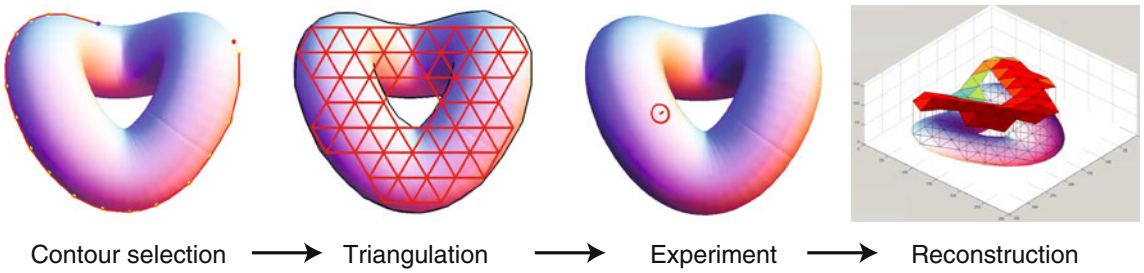

Fig. 1 Illustration of the four procedural steps

3-D reconstruction On the basis of the observers' settings, the 3-D surface can be reconstructed. These data are the final result; further analysis will depend on the specific research question, and should thus be designed by the experimenter.

These four steps will now be explained in detail.

Contour selection

First, an image is needed. Some shape should be visible, preferably a smooth one. In previous research, only an outer contour was defined. The method presented here also allows for defining a hole and inner contours. These three types of contours are visualized in Fig. 2. The experimenter can define the contour manually by selecting contour sample points that form a polygon that approximates the actual contour. The distance between the contour sample points can have approximately the same size as the triangulation faces. A sampling of contour points, as shown by the red dots in Fig. 3, is thus sufficiently detailed.

The output of the contour procedure consists of three sets of coordinates, for the outer, inner, and hole contours, which can be written in n-by-2 arrays. A point on the contour can be written as $\mathbf{c}_j^q = \left(x_j^q, y_j^q\right)$, where q defines the contour type by the letter o, h, or i (for outer, hole, or inner contour), and j is the index. For example, the first inner contour point is $\mathbf{c}_1^{\mathrm{i}} = \left(x_1^{\mathrm{i}}, y_1^{\mathrm{i}}\right)$. For the outer and hole contours, the last element (say, n and m, respectively) equals the first: $\mathbf{c}_1^{\mathrm{o}} = \mathbf{c}_n^{\mathrm{o}}$ and $\mathbf{c}_1^{\mathrm{h}} = \mathbf{c}_m^{\mathrm{h}}$.

Triangulation

Based on the contour data, the triangulation can be defined. To this end, a triangular grid is used that in principle covers the whole screen. However, only points within the outer contour that are not within the hole contour should be used and displayed (at this stage, inner contours are neglected). This requires an algorithm to test whether a point is between the outer and hole contours.

Contour-enclosed points filtering As can be seen in Fig. 3, a simple rule can be defined to assess whether a point $\mathbf{p}_i$ is within the closed contours: When a horizontal line is drawn in the positive x direction, it intersects a number of times with the closed contour. If this number is odd, $\mathbf{p}_i$ is within the closed contours. Thus, the number of intersections needs to be calculated.

First, we need to select contour point pairs whose y-coordinates enclose the y-coordinate of the point. In Fig. 3, these pairs are $\{\mathbf{c}_i, \mathbf{c}_{i+1}\}$ and $\{\mathbf{c}_j, \mathbf{c}_{j+1}\}$. Now we can define straight lines through these point pairs. A straight line through two subsequent contour points $\mathbf{c}_i = (x_i, y_i)$ and $\mathbf{c}_{i+1} = (x_{i+1}, y_{i+1})$ can be defined as $y = ax + b$, with $a = (y_{i+1} - y_i) / (x_{i+1} - x_i)$ and $b = y_i - ax_i$. The x-coordinate of the intersection point $\mathbf{p}_1'$ can now be calculated (the y-coordinates of the points $\mathbf{p}_1$ and $\mathbf{p}_1'$ are obviously the same): $p_1'^x = (p_1^y - b)/a$. Since the rule states that only intersections from the point in the positive x direction are to be counted, only $p_1'^x - p_1^x > 0$ is counted as an intersection.

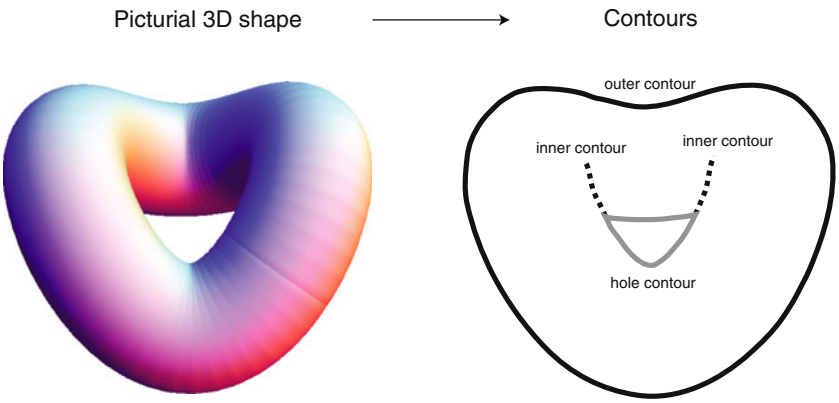

Fig. 2 Definition of the three types of contours

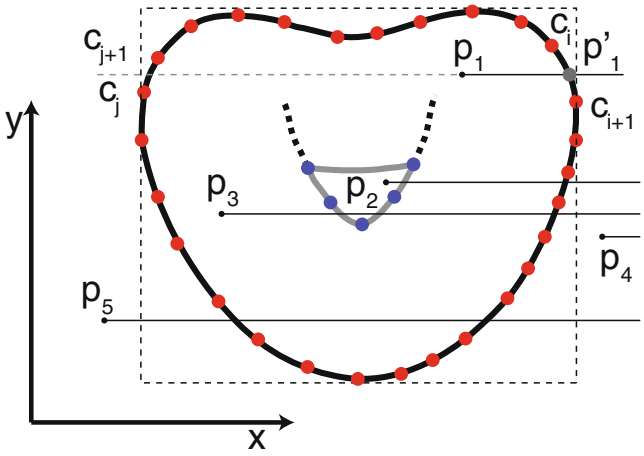

Fig. 3 Visualization of the algorithm that tests whether a point is within the closed contours

When this procedure is performed for all points, we get one intersection for $\mathbf{p}_1$ (odd, and thus this point is inside the closed contours), two intersections for $\mathbf{p}_2$ (outside), and so forth. Testing whether a point is within the closed contours is computationally laborious. Therefore, the procedure may include an initial selection of the triangulation positions that are within the rectangle defined by the width and height of the outer contour, as is shown by the outer dotted line in Fig. 3.

Adjusting the triangulation grid size and position It can be useful for the experimenter to adjust the grid size and position of the triangulation in real time. This can be done using the procedures described above, which are also implemented in the supplemental software. In the software, the position can be adjusted with the mouse and grid size by the arrow keys, as is illustrated in Fig. 4a, but other types of implementations may be equally user friendly.

Calculating faces and barycentres and performing final point filtering Up to now, only points that span the triangular grid have been used, without explicit face numbering. However, the reconstruction algorithm requires explicit information about the vertex numbers that constitute the triangles. Each face (triangle) is defined by three vertices, so a definition of all faces comprises an m-by-3 matrix, with a set of three numbers in each row that refer to the vertices. Note that the number of faces does not equal the number of vertices.

The faces can be calculated with a brute-force method, which is why this algorithm is not used during the real-time adjustment of grid size and location. Having defined the individual triangles, we can calculate the barycentres of the faces, which are the actual sample points where the gauge figure will be rendered.

Lastly, triangles crossing the inner contours need to be filtered. To this end, a line intersection algorithm is needed. The basic question is, do the lines between two point pairs $\{\mathbf{p}_1, \mathbf{p}_2\}$ and $\{\mathbf{p}_3, \mathbf{p}_4\}$, as shown in Fig. 5, coincide? This can simply be calculated by parameterizing vectors through these lines $\mathbf{v}_{12}(t) = t(\mathbf{p}_2 - \mathbf{p}_1) + \mathbf{p}_1$ and $\mathbf{v}_{34}(s) = s(\mathbf{p}_4 - \mathbf{p}_3) + \mathbf{p}_3$. When $0 < t < 1$, $\mathbf{v}_{12}(t)$ lies between $\mathbf{p}_1$ and $\mathbf{p}_2$, and the equivalent holds for the parameter s. The intersection parameters can be found by solving $\mathbf{v}_{12}(t) = \mathbf{v}_{34}(s)$. If both intersection parameters are between zero and one, the lines intersect.

As can be seen in Fig. 4, it can happen that a triangle line crosses the hole, which is unwanted. To overcome this problem, the hole contour should be included in the last filtering procedure. When the experimenter is satisfied with the triangulation parameters, the points (vertices), faces, and barycentres should be saved. A screen shot of the final result from the supplemental software is also exported, for later reference (see Fig. 4b).

The experiment

During the experiment, a gauge figure is subsequently presented at all barycentres, in random order. In Fig. 6, a

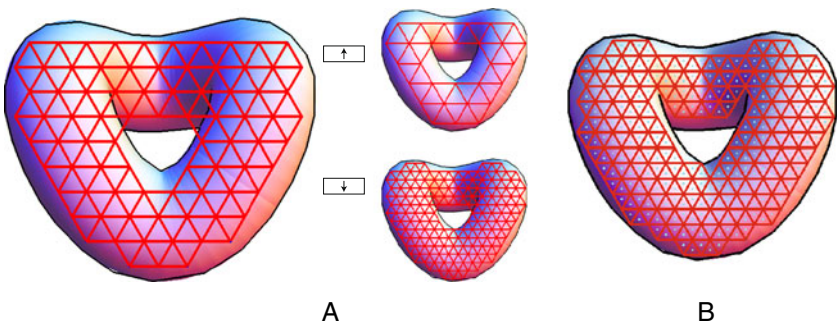

Fig. 4 **a** An initial triangulation is shown at startup. The experimenter can adjust the position with the mouse and can increase (up arrow) or decrease (down arrow) the triangle size. **b** When the experimenter is satisfied with the settings, the final version can be shown with barycentres and additional cuts in the mesh by the inner contour (see the Calculating Faces and Barycentres and Performing Final Point Filtering section)

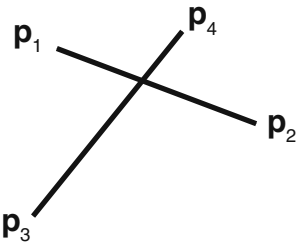

Fig. 5 Illustration of two intersecting lines

screen shot of the experiment is shown. The 3-D rotation of the gauge figure can be implemented as follows. The circle of radius r is defined by a polygon of n points that lie in the (x, y) plane centered at the origin. The rod has length r and sticks out in the z direction. Although the circle and rod are defined in 3-D coordinates, only the x- and y-coordinates are used for the actual rendering (orthographic projection). Changing the slant and tilt of the gauge figure can be achieved by rotating all coordinates around the y- and z-axes, respectively (the order is evidently important, since rotations do not commute). Rotations can be implemented by using rotation matrices.

The observer needs control over the attitude—that is, the slant and tilt of the gauge figure. A simple interface for this control is to use the mouse position, as is illustrated in Fig. 6. The location (x, y) of the mouse is tracked with respect to the middle of the screen. The tilt τ is defined by $\arctan(y / x)$, and the slant σ is defined by the distance $g\sqrt{x^2 + y^2}$, where g denotes a gain to tune the sensitivity of the mouse. To prevent confusion about where the mouse position starts, it is recommended that the gauge figure start at the $(\tau, \sigma) = (0, 0)$ position.

Surface reconstruction

The data from the experiment are attitudes that can be interpreted as depth gradients: the local change of depth in the x and y directions. This can be easily seen by noting the following. The gauge figure can be regarded as the normal vector on a local plane (the triangle is a small plane) and can be written as a function of slant and tilt: $\mathbf{n}(\tau, \sigma) = (\cos\tau \sin\sigma, \sin\tau \sin\sigma, \cos\sigma)$. This normal vector defines a plane, as is illustrated in Fig. 7. The equation for this plane is $n_x x + n_y y + n_z z = d$, where d is some unknown depth offset. It can be readily understood from this equation that the relative depth difference between two points is $(z_2 - z_1) = -[n_x(x_2 - x_1) + n_y(y_2 - y_1)] / n_z$. Similarly, the depth difference $(z_3 - z_1)$ can be calculated, while the depth difference $(z_3 - z_2)$ evidently follows from the other two depth differences and is thus omitted. Thus, one experimental setting results in two depth differences.

A system of linear equations from these depth differences is needed. The basic idea is that a matrix M should be constructed that fulfills the equation $M\mathbf{z} = \Delta\mathbf{z}$, in which $\Delta\mathbf{z}$ is a vector with all of the depth differences and $\mathbf{z}$ are all depth values. This can be done by using the matrix in Eq. 1.

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \dots \\ 1 & 0 & -1 & 0 & \dots \\ 1 & 0 & 0 & -1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \dots \\ 0 & 1 & -1 & 0 & \dots \\ 0 & 1 & 0 & -1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 1 & 1 & 1 & \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ \vdots \end{pmatrix} = \begin{pmatrix} z_1 - z_2 \\ z_1 - z_3 \\ z_1 - z_4 \\ \vdots \\ z_2 - z_3 \\ z_2 - z_4 \\ \vdots \\ 0 \end{pmatrix} \quad (1)$$

To reconstruct a continuous surface, one needs to constrain all triangles to be connected with their neighbours. This means that triangle edges that connect two triangles [e.g., (3, 5) in Fig. 7] result in two equations with possibly different depth differences, one resulting from the depth gradient of triangle (2, 3, 5), and the other from triangle (3, 5, 6). This needs to be accounted for in the system of equations from Eq. 1. To ensure that the reconstructed surface has zero average depth (depth offset is irrelevant, anyway), add a last row to the matrix consisting of all 1 s. This boundary condition implies that $\sum z_i = 0$. The reconstruction can now be calculated by taking the pseudo-inverse of the matrix $\mathbf{M}$: $\mathbf{z} = \widetilde{\mathbf{M}}^{-1}\Delta\mathbf{z}$. The output of the software gives a 3-D plot of the reconstructed surface (as shown in Fig. 8) and the 3-D vertices as a text file, which can be used for the analysis. A different version of the reconstruction algorithm has previously been published by Nefs (2008).

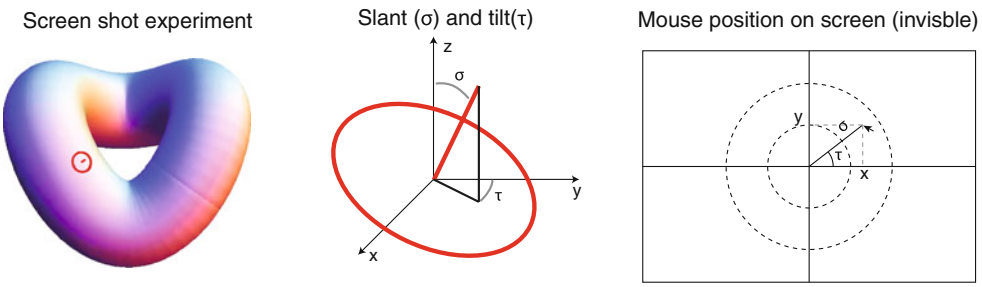

Fig. 6 The experiment. On the left, a typical setting of the gauge figure during the experiment is shown. In the middle, the slant and tilt are defined. On the right, the relation between the mouse position and the slant/tilt parameters is explained

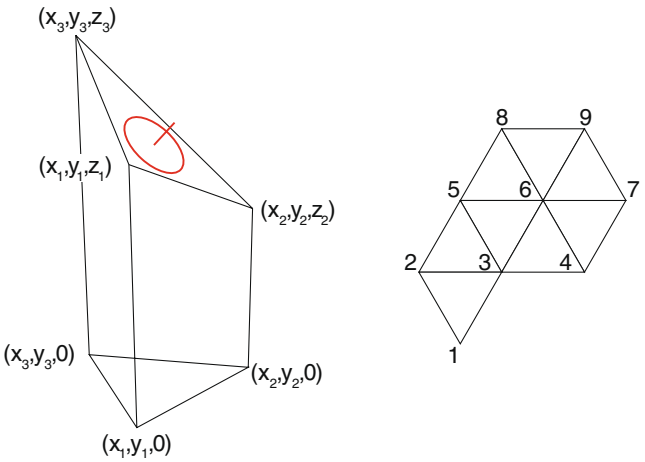

Fig. 7 Left: A single triangle with depth difference that is based on the depth gradient from the gauge figure. Right: Vertex numbering of the triangle faces

Discussion

This article gives a complete description of the implementation of the gauge figure method. This does not mean that it does not need some additional input by the user. To start with, users should be cautious using large holes or inner contours in the stimuli. As may have become clear, the reconstruction algorithm integrates the depth gradients, and the stability of this integration depends on the sampling. Some noise from observer settings is automatically filtered, because the reconstruction algorithm imposes an integrable surface:[1] The shared edges of the faces are joined—that is, the surface is "continuous." An example of a shared edge is (5, 3) from Fig. 7, while (5, 2) is an outer edge. When there are relatively few shared edges, such as in the arm of the bodybuilder in Fig. 9, the integration may get unstable. Instead of averaging out, the observer noise is integrated and may yield unstable results.

Two potential issues have been raised by researchers using the gauge figure method. Firstly, "the gauge orientation task suffers from several limitations. First, there is no way to distinguish between errors due to misunderstanding the orientation of the shape and errors due to misunderstanding the orientation of the gauge" (Cole et al., 2009). These authors imply that the mental image of the stimulus contains attitude information that is matched to the attitude information of the mental image of the gauge figure. However, the mental image is (evidently) in the mind, and therefore whether it contains attitude information is unknown. As Koenderink, van Doorn, Kappers, and Todd (2001) pointed out, "pictorial relief can only be defined operationally"; that is, the perceived attitude can only be defined through a *task*. In physics, a ruler measures distance, and a clock measures time. There is no way of distinguishing the time on the clock and the actual time, because time is operationally defined by the clock. Similarly, the perceived attitude is operationalized by the gauge figure probe. Hence, the point raised by Cole et al. (2009) does not seem to be an issue, in the context of Koenderink et al. (2001). The second issue was raised by an anonymous reviewer: "When confronted with a reconstruction of the shape that they [the observers] reported using the gauge figures, they quickly identify locations where the reported shape does not correspond to the shape that they actually intended to report." Indeed, this can be an issue, but only with respect to the reconstruction itself, and not with respect to the task. It is debatable whether one should let observers view a real-time reconstruction of the pictorial relief while doing the task. One may as well let observers use 3-D modeling software to reproduce pictorial shapes, or perform a shape discrimination task. All of these tasks may be useful to study vision, but they ignore the concept of operationalizing pictorial relief by using a (i.e., any) *gauge* figure probe.

It should also be noted that there exist more experimental tasks to measure pictorial surface structure (Koenderink et al., 2001). However, the gauge figure task is the most used and seems to be the most intuitive task for the observers. Using different methods may be worthwhile, since the gauge figure task is essentially based on first-order (orientation) estimates. Using tasks that probe zeroth- (Koenderink et al., 2001) or second-order information may give different insights into pictorial relief. Furthermore, pictorial surfaces are just one geometric facet of pictorial space. Recently, a method to quantify the spatial layout of objects in pictorial space has been developed (Wijntjes & Pont, 2010). In this method, observers were instructed to use a pointer to point from one object to another. These data were then used to reconstruct the relative depth differences in pictorial space. The combination of pictorial surface and spatial layout methods should increase our understanding of pictorial space.

The analysis of the results is to be defined by the user of the method described in this article. There is much literature that can serve as background material, and essentially, the way that the data are analysed depends on the specific research question. The method described in this report merely provides the means to start with a picture and arrive at the 3-D reconstructed relief.

[1] This assumption is met when the curl of the normal vector field vanishes identically (Koenderink et al., 1992). In most (if not all) studies, the integrability assumption was justified, but this evidently depends on the stimulus. It is not unthinkable that for some visual stimuli, the vector field will be nonintegrable.

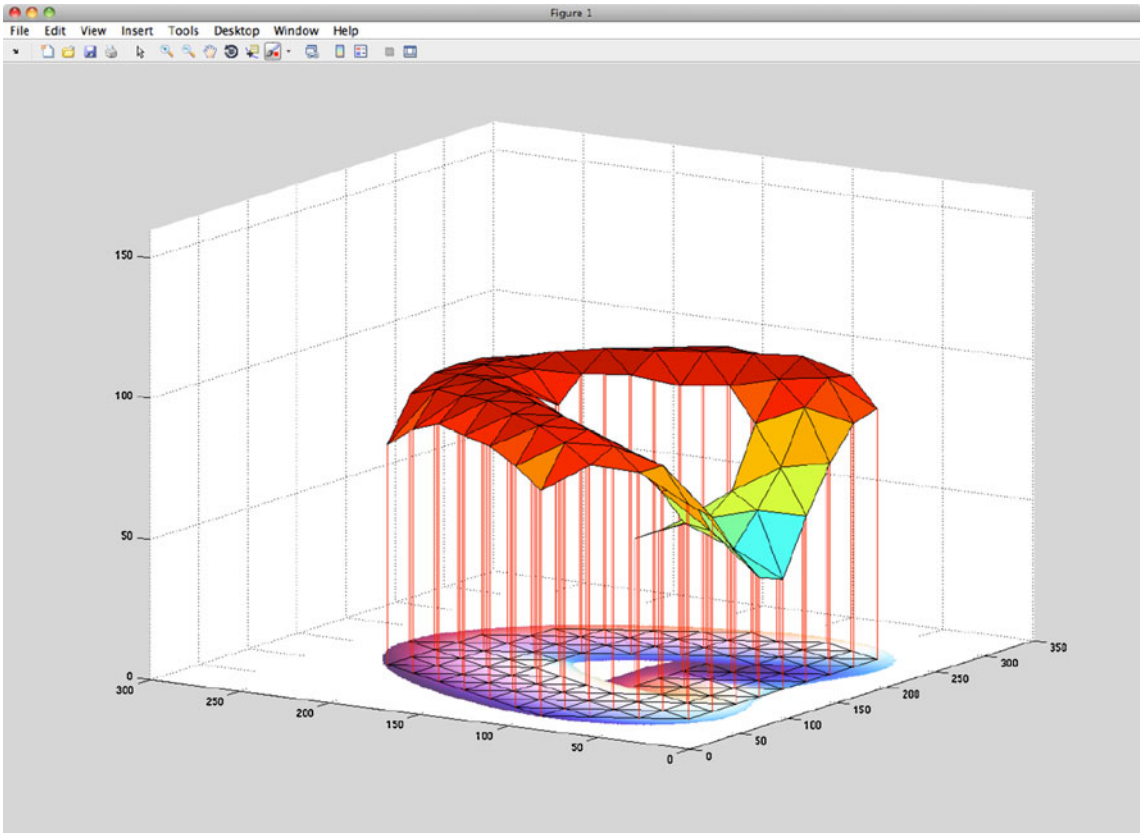

Fig. 8 Resulting 3-D relief superimposed on the picture

Author Note This work was supported by a grant from the Netherlands Organization for Scientific Research (NWO). Supplemental materials may be downloaded from the author's website: www.maartenwijntjes.nl/gaugefigure.

Appendix

The algorithms described above have been implemented in ready-to-use software written with PsychToolbox for MATLAB. Visit the PsychToolbox webpage (http://psychtoolbox.org) for installation instructions. It is recommended that new users familiarize themselves with the PsychToolbox demos to understand the general structure of the toolbox, although it should not be necessary to use the gauge figure software that is described here. The supplemental software package, an instruction movie, and the documented instructions can be downloaded from www.maartenwijntjes.nl/gaugefigure. For practical instructions, the instruction document is recommended.

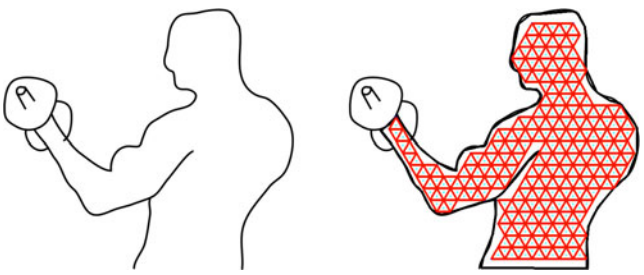

Fig. 9 Example of a stimulus in which the integration may get unstable

The software package consists of four main .m files, one for each of the four procedural steps shown in Fig. 1.

Contour creation

The contour can be manually defined my registering mouse clicks on the contour. The mouse position is tracked by `[mx, my, buttons] = GetMouse();`. After clicking the mouse, the positions are appended to the `locations0` matrix (0 stands for outer contour, in this case):

```
if find(buttons)
            locationsO=[locationsO;[mx my]];
            [...]
            if (sqrt( (locationsO(3,1)-mx)^2+(locationsO(3,2)-my)^2 ...
                )<DISTANCETHRESHOLD)&&length(locationsO)>4
             ncount=1;
            end
            WaitSecs(.5);
end;
```

The `[...]` stands for irrelevant code. The nested `if` loop checks whether the new location is within a threshold distance from the start location, which is indexed as the third location (`locationsO(3,:)`). If this is true, the `ncount=1` means that the outer contour is finished, and the program will continue with the other contours.

Triangulation

The triangulation starts by generating a triangular grid, which is done by the auxiliary .m file genPoints.m (stored in the `functions` folder):

```
function points=genPoints(w,h,dx,transX,transY)
%creates a triangular grid along the whole screen. These are just points,
%without any connection, i.e. no faces or triangles

dy=sqrt(3)*dx/2;
nw=round(w/dx);
nh=round(h/(sqrt(3)/2*dx));
dots=zeros(nw,nh,2);
dotsx=zeros(nh*nw,2);

for j=1:nh
for i=1:nw
    dotsx(i+nw*(j-1),:)=[(i-1)*dx+dx*mod(j,2)/2+transX (j-1)*dy+transY];
end
end

points=dotsx;
```

The input parameters are the width (`w`) and height (`h`) of the screen in pixels, the latter of which is given by `wRect` from `[w, wRect]=Screen('OpenWindow',screenNumber)` in triangulation.m. Furthermore, the grid distance is defined by `dx`, and a translation is supplied through `transX` and `transY`. This function is called in triangulation.m by `triangdots0=genPoints(wRect(3),wRect(4),dx,mx,my);` where `mx` and `my` are the mouse positions, and can be adjusted with the arrow keys:

```
if KeyCode(KbName('UpArrow'))
     dx=dx+1;
else if KeyCode(KbName('DownArrow'))
     dx=dx-1;
     end
end
```

The triangulation points have to be filtered for being inside the contour. This is done by the auxiliary file pointsInContour.m. This file first filters the points from the smallest rectangle that encloses the outer contour `dl` and puts the contour points in pairs (`cpairs`). Then it does the final filtering to `inside`:

```
[...]
inside=[];
for j=1:length(dl)
    point=dl(j,:);
    linepos=[];
    for i=1:length(cpairs)
        if betweenY(cpairs(i,:),point)==1
            linepos=[linepos ; i];
        end
    end
    dx=[];
    for i=1:length(linepos)
        xyxy=cpairs(linepos(i),:);
        a=(xyxy(4)-xyxy(2))/(xyxy(3)-xyxy(1));
        b=xyxy(2)-a*xyxy(1);
        x=(point(2)-b)/a;
        dx=[dx ; 0<x-point(1)];
    end
    %Here it is tested whether the number of intersections are odd.
    if mod(sum(dx),2)==1;
        inside=[inside ; j];
    end
end
pointsInContour=dl(inside,:);
```

The first nested `for` loop selects the contour pairs that enclose the y value of the `point` using betweenY.m. The second `for` loop it performs the intersection test described in the Contour-Enclosed Points Filtering section. Lastly, this program tests whether the number of intersections is odd, with the code `mod(sum(dx),2)==1`.

After the filtering, the inner points should be connected by mesh lines, which is done by meshit.m, which basically searches for point pairs that have a distance equal to the grid distance defined by `dx`. Up to now, point filtering has been done in real time, so that the experimenter can see the triangulation while adjusting the grid size and translation manually. When the experimenter is satisfied with the triangulation, the faces are calculated by facit.m. Then these

faces are filtered for triangles that intersect with the inner contours, as described in the Calculating Faces and Barycentres and Performing Final Point Filtering section:

```
denominator=-x3*y1 + x4*y1 + x3*y2 - x4*y2 + x1*y3 - x2*y3 - x1*y4 + x2*y4;
t = -(x3*y1 - x4*y1 - x1*y3 + x4*y3 + x1*y4 - x3*y4)/denominator;
s = -(-x2*y1 + x3*y1 + x1*y2 - x3*y2 - x1*y3 + x2*y3)/denominator;
if (0<t)&&(t<1)&&(0<s)&&(s<1)
    lineintersect=1;
else
    lineintersect=0;
end
```

Experiment

In the experiment, the gauge figure has to be rendered at (randomly ordered) barycentres. The gauge figure is defined in three dimensions, which makes it easy to rotate. Here, the disk (called `ellips`) and rod are defined:

```
    %The ellips basis, in three dimensions which makes it easy to rotate
    %the whole thing using rotation matrices.
    n=45;                   %The ellips is rendered using n line elements
    t=0:2*pi/n:2*pi;
    r=GAUGE_FIGURE_RADIUS;  %radius and length of the ellips and rod, reps.
    ex=r*cos(t);
    ey=r*sin(t);
    ez=0*t;
    ellips=[ex; ey; ez]';
    %The rod basis, similar rationale as the ellips basis
    rx=0;
    ry=0;
    rz=r;
    rod=[rx ry rz]';
```

During a trial, the mouse position (`mx,my`) defines the slant (`phi`) and tilt (`theta`):

```
phi=sqrt((mx-wRect(3)/2)^2+(my-wRect(4)/2)^2)/phigain;
if phi>pi/2
    phi=pi/2;
end
theta=atan2(my-wRect(4)/2,mx-wRect(3)/2);
```

The sensitivity of the slant can be tuned by `phigain`. These parameters then define the rotation matrix `rotmat`:

```
rotmat=vrrotvec2mat([0 0 1 theta])*vrrotvec2mat([0 1 0 phi]);
pos=[loc(1) loc(2) 0];
for i=1:n+1
    ellips2(i,:)=(rotmat*(ellips(i,:))')'+pos;
    %So, the basis ellips is positioned at the origin,
    %which is why the rotation works. Afterwards, the
    %coordinates get translated to the psoition "pos"
end
rod1=rotmat*rod;
```

The location of the barycenter is `loc`. Since the gauge figure is still three-dimensional, `pos` is used instead of `loc`. In the `for` loop, the `ellips` is rotated and translated to the barycenter. The same is done for the `rod`.

Reconstruction

The experimental data (n trials) are in the form of an n-by-4 matrix, with the first two columns for the barycentres and last two columns for the slant and tilt. The first thing that reconstruction.m does is transform these data into normal vectors, and then into gradients:

```
%convert slant/tilt settings in a normal vector
normalvectors=[];
for i=1:length(data)
    normalvectors=[normalvectors ; [cos(data(i,4))*sin(data(i,3)) ...
        sin(data(i,4))*sin(data(i,3)) cos(data(i,3))]];
end

%convert normal vectors in depth gradients
gradients=[];
for i=1:length(data)
    gradients=[gradients ; [normalvectors(i,1) ...
        normalvectors(i,2)]/normalvectors(i,3)];
end
```

These depth gradients need to be converted to relative depth differences, as described in Eq. 1. For each triangle, the depth difference between the first vertex and the other two vertices is calculated.

```
relativedepths=[];
for i=1:length(faces)
    f=faces(i,:);
    x1=vertices(f(1),1);
    y1=vertices(f(1),2);
    x2=vertices(f(2),1);
    y2=vertices(f(2),2);
    x3=vertices(f(3),1);
    y3=vertices(f(3),2);
    gx=gradients(i,1);
    gy=gradients(i,2);
    relativedepths=[relativedepths ; gx*(x2-x1)+gy*(y2-y1) ];
    relativedepths=[relativedepths ; gx*(x3-x1)+gy*(y3-y1)] ;
end
relativedepths=[relativedepths ; 0];
```

Note that a zero is appended to the `relativedepths` vector to satisfy the boundary condition $\sum z_i = 0$ (overall depth is zero). Now, the matrix $\mathbf{M}$ has to be defined. This is done as follows:

```
x=[];
for i=1:length(faces)
    x=[x; [faces(i,1) faces(i,2)]];
    x=[x; [faces(i,1) faces(i,3)]];
end

m=[];
for i=1:(length(relativedepths)-1)
    row=zeros(1,length(vertices));
    row(x(i,1))=1;
    row(x(i,2))=-1;
    m=[m;row];
end
m=[m;ones(1,length(vertices))];
```

Again, note the appending of a row of ones to satisfy $\sum z_i = 0$. Finally, the depths are calculated through the pseudo-inverse of **M**: `depths=pinv(m)*relativedepths;`. The remainder of reconstruction.m writes the 3-D vertices to a data file and produces a visualization of the results.

References

Brainard, D. H. (1997). The Psychophysics Toolbox. *Spatial Vision, 10,* 433–436. doi:10.1163/156856897X00357

Cole, F., Sanik, K., DeCarlo, D., Finkelstein, A., Funkhouser, T., Rusinkiewics, S., et al. (2009). How well do line drawings depict shape? *ACM Transactions on Graphics, 28*(3). doi:10.1145/1531326.1531334

Koenderink, J. J., van Doorn, A. J., & Kappers, A. M. L. (1992). Surface perception in pictures. *Perception & Psychophysics, 52,* 487–496. doi:10.3758/BF03206710

Koenderink, J. J., van Doorn, A. J., Kappers, A. M. L., & Todd, J. T. (2001). Ambiguity and the "mental eye" in pictorial relief. *Perception, 30,* 431–448. doi:10.1068/p3030

Nefs, H. T. (2008). Three-dimensional object shape from shading and contour disparities. *Journal of Vision, 8*(11), 11:1–16.

Pelli, D. G. (1997). The VideoToolbox software for visual psychophysics: Transforming numbers into movies. *Spatial Vision, 10,* 437–442. doi:10.1163/156856897X00366

Wijntjes, M. W. A., & Pont, S. C. (2010). Pointing in pictorial space: Quantifying the perceived relative depth structure in mono and stereo images of natural scenes. *ACM Transactions on Applied Perception*, *7*(4), 24:1–8. doi:10.1145/1823738.1823742