Displaying 3D Images: Algorithms for Single Image Random Dot Stereograms

Harold W. Thimbleby,[†] *Stuart Inglis,*[‡] *and Ian H. Witten*[§*]

Abstract

This paper describes how to generate a single image which, when viewed in the appropriate way, appears to the brain as a 3D scene. The image is a stereogram composed of seemingly random dots. A new, simple and symmetric algorithm for generating such images from a solid model is given, along with the design parameters and their influence on the display. The algorithm improves on previously-described ones in several ways: it is symmetric and hence free from directional (right-to-left or left-to-right) bias, it corrects a slight distortion in the rendering of depth, it removes hidden parts of surfaces, and it also eliminates a type of artifact that we call an "echo".

Random dot stereograms have one remaining problem: difficulty of initial viewing. If a computer screen rather than paper is used for output, the problem can be ameliorated by shimmering, or time-multiplexing of pixel values. We also describe a simple computational technique for determining what is present in a stereogram so that, if viewing is difficult, one can ascertain what to look for.

Keywords: Single image random dot stereograms, SIRDS, autostereograms, stereoscopic pictures, optical illusions

† Department of Psychology, University of Stirling, Stirling, Scotland. Phone (+44) 786–467679; fax 786–467641; email hwt@compsci.stirling.ac.uk

‡ Department of Computer Science, University of Waikato, Hamilton, New Zealand. Phone (+64 7) 856–2889; fax 838–4155; email singlis@waikato.ac.NZ.

§ Department of Computer Science, University of Waikato, Hamilton, New Zealand. Phone (+64 7) 838–4246; fax 838–4155; email ihw@waikato.ac.NZ.

* Please address all correspondence to Ian H. Witten

Introduction

The perception of depth arises in various ways, including perspective, shadows, color and intensity effects, hazing, and changes in size. These effects are well known by artists and can be manipulated in pictures to give an impression of depth—indeed, pictures of solid objects drawn without perspective look odd.

In three dimensions, additional methods are available for perceiving depth. Because the eyes are separated (on each side of the nose) each one has a slightly different view of a distant object, and the angular difference varies with distance. When both eyes look at an object, they turn toward it (convergence) and the retinas receive slightly different images (stereo disparity). Also the eyes focus differently for objects at different distances.

The fusing of the two eyes' images to produce the effect of depth is called "stereopsis." Stereopsis requires the cooperation of both eyes to construct a single mental picture. This paper analyses how to use stereopsis to recreate the appearance of depth from a purely flat picture. To understand how to do so, we need only consider the geometry of stereo vision, not its physiological or psychological basis.

Stereopsis occurs naturally when viewing solid objects with both eyes. It is only possible from 2D pictures when each eye receives a separate image, corresponding to the view it would have had of the 3D object depicted. There are various ways to produce stereo images, but special equipment is normally required, either for making the image or for viewing it. In a stereoscope (Wheatstone, 1838, 1852), an optical instrument similar to binoculars, each eye views a different picture, and can thereby be given the specific image that would have arisen naturally. An early suggestion for a color stereo computer display involved a rotating filter wheel held in front of the eyes (Land & Sutherland, 1969), and there have been many successors.

In contrast, the present paper describes a method that can be viewed on paper or on an ordinary computer screen without any special equipment. Although it only displays monochromatic objects,[1] interesting effects can be achieved by animation of video displays. The image can easily be constructed by computer from any 3D scene or solid object description.

---

1 The image can be colored (e.g., for artistic reasons), but the method we describe does not allow colors to be allocated in a way that corresponds to an arbitrary coloring of the solid object depicted.

Julesz and Miller (1962) were the first to show clearly that a sense of depth could arise purely from stereopsis, without relying on other cues such as perspective or contours. They used random patterns of dots which, although meaningless to single eye viewing, nevertheless created a depth impression when viewed in a stereoscope.

It might seem that stereopsis necessarily requires two separate pictures, or at least some method of splitting a single picture into two to give each eye a separate view (using, for instance, red/green filters, polarized light, or interference, as in holograms). Recently, however, Tyler and Clarke (1990) realized that a pair of random dot stereograms can be combined together, the result being called a "single image random dot stereogram" (SIRDS) or, more generally, an autostereogram. Essentially, one overlays the two separate random dot patterns, carefully placing the dots so that each one serves simultaneously for *two* parts of the image. All that is necessary is to constrain the dot pattern suitably. Ordinary stereograms, such as photographs or wire frame models, cannot be combined into a single image, since one merely obtains a double picture.[2]

It turns out that very convincing images with vivid depth can be constructed in this way, and the advantage of this ingenious approach is that no special viewing equipment is required. It does take a little practice to see depth in the pictures, but the experience is very satisfying when first achieved.

Tyler and Clarke (1990) described a simple but asymmetric algorithm, which meant, for example, that some people can only see the intended effect when the picture is held upside-down. This paper presents a new, simple, and symmetric algorithm for generating single image stereograms from any solid model.

There is a vast literature on the psychology of stereopsis. For example, Marr and Poggio (1976, 1979) discuss computational models of the visual processes that are involved in interpreting random dot stereograms. Gulick and Lawson (1976) offer an excellent general survey of the psychological processes and history of stereopsis. Although these references provide useful general background, they do not bear directly on the technique described in the present paper.

Stereo vision and autostereograms

Figure 1 shows an image plane placed between the eyes and a solid object. Imagine that it is a sheet of glass. (In fact, it could be a computer screen, or a piece of paper.) Light rays

---

[2] An effect called the "wallpaper illusion" occurs when lines of horizontally repeating patterns are perceived to lie at different depths; however, since the patterns repeat monotonously in wallpaper, they convey no useful information.

are shown coming from the object, passing through the image plane, and entering each eye. So far as the eyes are concerned, two rays pass through each point in the image plane, one for each eye. If both rays are the same color and intensity, they can be conveniently reproduced by a single light source in the image plane. Hence, although the object can be seen stereoscopically, there need only be *one* image in the image plane, not two, and it can be shared by both eyes. This solves the problem of seeing a stereoscopic picture without any special equipment.

The problem of generating the autostereogram amounts to illuminating the screen in such a way that it simulates a pattern of light that could have come from a solid object lying behind it. In general, each point of the object will map into two points on the image plane. Now, if two locations on the solid object are chosen carefully, as shown in Figure 1, and are both black dots, then it can be arranged that they generate just three black images on the plane, two of the images coinciding. Notice that the distance between each pair of dots is different: the further the corresponding point on the object is behind the image plane, the further apart are its two image points.

The central dot shown on the image plane in the Figure represents two separate locations on the object. Therefore these two locations must have the same color. In turn, then, the other two dots shown in the image plane must be the same color. Overall, of course, some dots must be different colors, or else the image plane would appear uniform and not present any useful information about the object lying behind it. Such considerations constrain the surface coloring of the object. It is sufficient to use only two colors (for example, black and white), and there is considerable flexibility in choosing them.

Figure 1 also illustrates the task of viewing autostereograms, of seeing depth in the initially meaningless arrangement of random dots. Suppose the image plane is transferred to a sheet of opaque paper. If the eyes converge to view the paper in the normal way, then they are not converged to be able to reconstruct the solid image. The same effect occurs when you look at a mark on a window: objects behind the window appear double. In the case of random dot stereograms, seeing the solid image “double” is tantamount to not seeing it at all. To view it correctly one must deliberately deconverge one’s eyes as explained in Box 1.

A program for generating single-image random dot stereograms

Referring to Figure 1, it can be seen that constraints only affect points along a line that lies in the same plane as the two eyes. This gives a clue to making the algorithm efficient: it can construct the image line by line. The inevitable disadvantage is that there are no constraints at any other angle, and therefore the stereo effect is only achievable when the picture is upright (or upside down). Tyler and Clarke (1990) briefly discuss the possibility of having orthogonal lines of constraints, but we will not pursue it here.

Our program is based on the geometry shown in Figure 2. The object to be portrayed lies between two planes called the "near" and "far" planes. The latter is chosen to be the same distance D behind the screen as the eyes are in front. This is a convenient value because when viewing the autostereogram the eyes should converge on the far plane, and you may be able to catch your reflection in the screen and use this to assist the convergence process. Since initially you don't know what you are looking for, it helps if the eyes can initially converge on a large equidistant target. The near plane is a distance μD in front of the far plane, and in the program μ is set to 1/3. The separation between the near and far planes determines the depth of field (not the depth of focus—for all dots actually lie on the image plane and both eyes should focus there). Increasing the depth of field by increasing μ brings the near plane closer to the screen and causes greater difficulty in attaining proper convergence of the eyes.

It is convenient to define the "image stereo separation" of a point on the surface of the solid object viewed in 3D to be the distance between its image points lying in the image plane. This quantity relates directly to the conventional measure of stereo disparity, which is the difference in angle subtended at the eyes. Since the background is taken to be the same distance behind the screen as the screen is from the eyes, the separation for the far plane is half the distance between the eyes.

Figure 2 shows the stereo separation s for a point with specified z-coordinate. The range of z-values is from 0, which corresponds to the far plane, to 1, which corresponds to the near plane. Thus the point is a distance $\mu z D$ in front of the far plane, or $(1-\mu z)D$ from the image plane. By similar triangles,

$$s = \frac{1-\mu z}{2-\mu z} E,$$

giving stereo separation, s, as a function of z. This is the fundamental relationship on which the program is built.

THE BASIC ALGORITHM

The first lines in the program of Figure 3 set the scene. The screen is `maxX` by `maxY` pixels, and the object's z-value is `Z[x][y]`. The depth of field μ is chosen to be `1/3`. Neither the eye separation nor the screen resolution are critical: approximate values will do in both cases. The image stereo separation corresponding to the far plane is called `far`.

The program processes one scan line at a time (using the large loop in lines 16–62 of Figure 3). The key to solving the constraint equations is to record what the constraints are in an initial pass (lines 26–56), and follow this with a second pass that allocates a random pixel value (black or white) whenever there is a free choice, and otherwise obeys the relevant constraint (lines 57–61).

Pixel constraints are specified by the `same[]` array. In general, each pixel may be constrained to be the same color (black or white) as several others. However, it is possible to arrange that each element of the array need only specify a *single* constraint on each pixel. The `same[]` array is initialized by setting `same[x]=x` for every pixel, representing the fact that, in the absence of any depth information, each pixel is necessarily constrained to be the same as itself (line 24).

Then the picture is scanned giving, at point `x`, a separation `s` between a pair of equal pixels that corresponds to the image stereo separation at that point. Calling these two pixels `left` and `right`, then `left` is at `x-s/2` and `right` at `x+s/2`, but just in case `s` is odd, we can more accurately position `right` at `left+s` (lines 28–29). The pair of pixels, `left` and `right`, must be constrained to have the same color. This is accomplished by recording the fact in the `same[]` array. (However, there may be geometric reasons why the corresponding point on the solid object is not visible along both lines of sight; this is checked in lines 35–39 and is discussed in detail below.)

To ensure that the pixel at `left` is recorded as being the same as `right`, should we set `same[left]=right`, or set `same[right]=left`, or both? In fact, it is unnecessary to set them both. When the time comes to draw the pixels, the line will be scanned either left-to-right or right-to-left, and in the latter case (for example) it is only necessary to ensure that `same[left]=right`. There is no significance in which of the two directions is used. We choose right-to-left (line 57), set `same[left]=right`, and require, as an invariant, that `same[x]` ≤ `y` whenever `x` ≤ `y`.

Now `same[left]` may have already been set in the course of processing a previous constraint. We have already decided that `same[x]` should record which pixel to the right of `x` is the same as it. If `same[left]` is already constrained, that constraint is followed rightwards (lines 43–46, using variable `l` to follow the `same` links) to find a pixel that is not otherwise constrained. In following the constraints, the variable `l` may “jump” over `right`. If this happens (line 43), lines 48–52 preserve the assumption that `left<right` by swapping them.

Once the constraints have been set for all pixels in a scan line, the line is re-scanned in decreasing `x` order, that is, from right to left as promised above, allocating pixel values (lines 57–61). When a pixel is unconstrained (`same[x]=x`), a value is chosen for it randomly, and stored in `pix[x]`. Otherwise its value must be constrained to be the same as some pixel further to the right, and so `pix[x]` is set to `pix[same[x]]`. The routine `Set_Pixel()` then sets the pixel to black or white on the screen as soon as its color is known. (If desired, the number of calls to `Set_Pixel()` can be halved by first drawing a white line across the scan line, and then only setting the black pixels.)

The code is followed by drawing two circular marks with their centers separated by an appropriate amount, near the bottom of the screen (lines 63–64). The convergence marks are given a separation of `far`, which places them on the background of the 3D picture. Some viewers find it easier to see the stereograms by converging their eyes on the near plane, and this is facilitated by separating the marks by `separation(1)` instead of `far= separation(0)`. Initial viewing may be slightly easier if the marks are placed at the center of the screen, though for aesthetic reasons we prefer to place them near the bottom.

HIDDEN SURFACE REMOVAL

It may be that, because of a transition in the object (from the near to the far plane, say), a surface in the foreground obscures one eye's view of a more distant point. Hidden surface removal is a technical detail, and very few objects make it visibly worthwhile. Yet the advantage is that for any part of the object that is strictly hidden, there is one fewer constraint to process. In turn, this gives the algorithm greater flexibility in allocating pixel colors, which reduces the problem of artifactual "echoes" as discussed in the next section. The hidden surface removal code is in lines 35–39 of Figure 3.

Figure 4 shows a point on the object that would cause two pixels of the image to be linked together, were it not for an obscuring object that interrupts one eye's view. The crucial inequality for hidden surface removal is that $z_1 \geq z_t$, where z_1 is the z-coordinate of a point on the obscuring object and z_t is the z-coordinate of a point on the ray from the eye to the original object. The x-coordinate is governed by the distance t, and if such an interruption occurs for any value of t greater than 0 and up to the point where $z_t = 1$, then the original point is no longer visible by this eye. The small shaded triangle is similar to the much larger one with the same apex but based on half of the line joining the eyes, and so

$$\frac{t}{(z_t - z_0)\mu D} = \frac{E/2}{(2 - \mu z_0)D}$$

(the relevant distances in the denominators are marked in Figure 4), from which it follows that:

$$z_t = z_0 + \frac{2(2 - \mu z_0)t}{\mu E}.$$

This is calculated in line 36 of the program in Figure 3. The crucial inequality $z_1 \geq z_t$ appears on line 37, and at the same time a test is made for the other eye's view being obscured. The whole operation is repeated for different values of t, from 1 up to the maximum possible value which occurs when $z_t = 1$.

This algorithm checks for obscuring surfaces at each point of the image plane, but if a geometric object were being rendered rather than the `Z[x][y]` array, a more efficient implementation could be devised using surface intersections.

Geometrical limitations

This section describes some of the geometrical limitations of the approach. In practice, none of these detract from the depth effect.

GEOMETRICAL DISTORTION

As Figure 2 illustrates, the object being viewed will have a slight lateral distortion in the x direction. Suppose a value of `x` has been reached (Figure 3, line 26) that corresponds with point a on the image plane. In order to determine the stereo separation, the program will look along the line aA to find the appropriate depth, which in this case locates a point (marked A) on the far plane (line 27). However, the stereo separation should really be governed by point A' instead. There might be something intervening between the eyes and A', but not between the eyes and A. While it would be possible to modify the program to correct the problem, this could only be done for a fixed viewing position.

There is an analogous distortion in the vertical or y direction, perpendicular to the plane containing the eyes. Hidden-surface decisions are made by projecting backwards from each scan line in a direction orthogonal to the viewing plane, and this is only accurate when the object subtends a small vertical angle at the eyes. Neither the horizontal nor the vertical distortion affects the depth geometry, and neither are noticeable at normal viewing distances. For distant viewing both errors disappear (the lateral one disappears because points A and A' of Figure 2 coincide).

The amount of depth resolution that is available is the same as the distance $far–near+1$, in pixels, where far and $near$ are the stereo separations corresponding to the far and near planes, that is, `separation(0)` and `separation(1)` in the program. This is because points on the far plane are separated by far pixels on the screen, points on the near plane by $near$ pixels, and intermediate values are interpolated. For the parameters in the program $far = 90$ and $near = 72$, giving a total of 19 separate depth planes. The number of depth planes increases in proportion with the output device resolution.

However, the depth planes are not spaced exactly the same distance apart, because the stereo separation is not a linear function of z. Previous schemes for calculating single image random dot stereograms (see Box 2) interpolate the z-values linearly into the space between the near and far planes, creating a small depth distortion. The program of Figure 3 corrects this problem by using the true non-linear separation function (line 7).

Finally, a very small bias is created by always rounding down when the stereo separation is halved in line 28 of Figure 3, which tends to shift the image very slightly rightwards. This is negligible in practice, but could be corrected if desired by randomizing the direction of rounding to remove the consistent bias towards larger values of `left` (and, consequently, `right`).

FALSE FUSION: ECHOES

Suppose there are constraints on pixels such that a=b and b=c. It follows that a=c. This last constraint, interpreted geometrically, corresponds to a third point in 3D space, and if pixel b lies between pixels a and c the new point lies further from the eyes than the points constraining a=b or b=c. There may be no such point on the object! This phenomenon, which follows inexorably from the transitivity of equality, can produce visible artifacts that we call "echoes".

Most transitive pixel constraints do not cause a visible echo. If two pixels are constrained to be equal but are separated by more than `far`, the echo is conceptually behind the object—indeed if the pixels are separated by more than E the echo is impossible to see anyway since it corresponds to a virtual image behind the viewer's eyes. However, for regular objects such as plane figures, echoes may combine in complex but systematic ways right across the image. Periodicities in constraints may result in echoes which appear in the same depth plane as does a legitimate part of the object. Such echoes can be very distracting, and of course may be quite misleading when they occur in unknown or unfamiliar objects.

Fortunately, hidden surface removal reduces constraints for plane figures to the point where echoes in the same plane as the figure cease to be a noticeable problem. For example, Figure 5a shows the echoes that arise from the test pattern of Figure B1 if the hidden-surface code is suppressed. It was generated as follows. First, a version of Figure B1 was created without hidden surface removal. Then it was scanned for pairs of pixels with the same value (black or white) that have a separation corresponding to the near plane, that is, `separation(1)`. Whenever this occurred, the midpoint was marked black. Of course, it frequently happens by chance, so the resulting image is mottled, but the visible echoes appear as the large connected black regions that can be seen in Figure 5a. If the procedure is repeated with the hidden-surface code in place, the corresponding picture shows just the single large annulus, without any echoes at all. Visible echoes are also produced by Tyler and Clarke's (1990) scheme for calculating single image random dot stereograms described in Box 2, and these appear in Figure 5b. The asymmetry of this scheme is clearly evident in this figure.

Often, a visible echo creates such an implausible distortion that it remains unnoticed—certainly when the object is familiar, such as a sphere or text written in relief. Nevertheless, once an echo has been noticed, it may be hard to ignore it again! Once seen, the echoes often persist, although for some people they come and go of their own accord (like depth reversal in the Necker cube illusion). This is probably because *both* the background plane and the echoes in the foreground are simultaneously present in the autostereogram, and the brain must choose which to see.

FALSE FUSION: ARTIFACTS

Even though there may be no geometrical constraint that $a=b$, it can happen that a and b are given the same color purely by chance. This will result in a visual *artifact* that may be distracting to the eye.

With a computer screen display, artifacts can be effectively eliminated by displaying different versions of the picture in rapid succession. This can be done efficiently by storing all the constraints in an array, and then repeating lines 57–61 but with different random numbers. This means that artifacts appear only intermittently and are therefore hardly noticed. If the display can be recomputed fast enough, or if several versions are generated in advance and cycled through quickly, a striking “shimmering” effect is obtained. This makes it very much easier to see the depth illusion because the eyes cannot latch on to chance patterns or focus on individual dots—the only constant is the 3D image.

It is also possible to ameliorate artifacts by using gray levels. Rather than color the pixels black and white, which are equal half the time, they can be given gray levels (or colors) from a larger palette, say 8 gray levels. Then chance artifacts become less likely.

FALSE FUSION: INCORRECT CONVERGENCE

False fusion can also occur simply because the eyes are incorrectly converged. The most common cause of this, in our experience, is having the eyes converged at far too great a distance and seeing the virtual solid object that is imaged by alternate image pixels in the image plane. Figure 6 shows the effect: instead of seeing the annulus in Figure B1 one sees two interlocking rings. In fact, if the eyes really are separated by 2.5 inches as assumed in the program, this effect will be achieved precisely when the gaze is directed at infinity. The risk of perceiving 3D objects with the eyes set at an incorrect convergence makes guide dots essential, especially for more complex objects.

Conclusions

Autostereograms are very satisfying to view and are becoming increasingly popular, particularly as recreational posters and in scientific illustrations. A public-domain X-windows demonstration program for displaying and printing autostereograms is available from `ftp.cs.waikato.ac.nz`. It incorporates the basic algorithm of Figure 3, along with an interface which gives control over all of the features discussed in this paper. As a final example, and to illustrate an object that requires continuous change in depth, Figure 7 shows a hemisphere bulging out of a plane.

Previous attempts at drawing autostereograms have relied on an asymmetric algorithm that produces satisfactory but not ideal results. The present paper shows how a symmetric algorithm may be derived and implemented; it is efficient and in practice as fast as the incorrect algorithm. The correct implementation of geometrical constraints, together with the hidden surface feature, enables higher levels of detail in the solid object to be faithfully recorded in the stereogram. The "shimmering" effect that is possible on a dynamic display renders stereograms much easier to view. These features give artists greater freedom and scope for creative use of this new medium and provide the basis for many interesting experiments with optical illusions.

Acknowledgments

We would like to acknowledge the contribution of Kerry Guise, who assisted us with some of our experiments and wrote the X-windows demonstration program.

References

Gulick, W. L. and Lawson, R. B. (1976) *Human Stereopsis. A Psychological Analysis*. Oxford University Press.

Julesz, B. and Miller, J.E. (1962) Automatic stereoscopic presentation of functions of two variables. *Bell System Technical Journal* **41**: 663–676; March.

Land, R.I. and Sutherland, I.E. (1969) Realtime, color, stereo, computer displays. *Applied Optics* **8**(3): 721–723; March.

Marr, D. and Poggio, T. (1976), Cooperative computation of stereo disparity, *Science* **194**, 283–287; October 15.

Marr, D. and Poggio, T. (1979), A computational theory of human stereo vision, *Proceedings Royal Society of London*, **B204**, 304–328.

Slinker, G.S. and Burton, R.P. (1992) The generation of random dot and line autostereograms. *J Imaging Science and Technology* **36**(3): 260–267; May/June.

Tyler, C.W. and Clarke, M.B. (1990) The autostereogram. *SPIE Stereoscopic Displays and Applications 1258*: 182–196.

Wheatstone, C. (1838) Contributions to the physiology of vision. Part I. On some remarkable, and hitherto unobserved, phenomena of binocular vision, *Royal Society of London Philosophical Transactions*, **128**, 371–394.

Wheatstone, C. (1852) Contributions to the physiology of vision. Part II. On some remarkable, and hitherto unobserved, phenomena of binocular vision (continued), *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, series 4, **3**, 504–523.

Box 1: Viewing a single image random dot stereogram

Figure B1a is a random dot stereogram of a large annulus, exactly the same size and texture as that of Figure B1b, floating above the middle of the picture. The challenge of successful viewing is that the viewer must decouple the eyes' convergence and focusing. Since this is never done in normal circumstances (except perhaps when tired) it may be difficult to achieve a successful 3D effect initially. The trick is to converge the eyes at a distance well beyond the paper on which the dots lie, yet focus them on the dots themselves. Working in bright light can help, as contraction of the iris causes the depth of field of the eyes to increase, and they rely less on focusing.

Altering the eyes' convergence is not easy, and the two circular marks we have drawn near the bottom of the picture can be used for assistance. Blinking (or covering) the eyes alternately, the marks can be made to produce four images, two with the right eye and two with the left. Now deconverge your vision so that all four images can be seen with both eyes open. The aim is to move the two pairs of images closer together until the inner two coincide and only three distinct marks are seen. The center mark is binocular while the outer two are seen monocularly, with only one eye each. Fixate on the center one to stabilize the image, and then carefully allow your gaze to wander around the whole picture. Placing the convergence marks in the center of the picture instead of near the bottom can make for easier viewing but usually spoils the picture.

When viewing the autostereogram you should strive to converge your eyes on the far plane. In Figure B1a this is the same distance behind the paper as the eyes are in front. Copying the stereogram onto a transparency greatly aids viewing. Mark an X on a plain sheet of paper and hold the transparency about halfway between your eyes and the paper. View the paper through the transparency and converge your eyes on the X. Carefully allow your gaze to wander across the whole picture. Once you have some experience, you can dispense with the transparency and work directly from the printed image.

Do not be content with an inferior 3D effect. If you become dimly aware of some circularity in the dot pattern, along with some tenuous 3D effects, this is certainly not the full illusion! Viewed properly, the annulus should leap out of the plane at you, and the effect should be stable and persist through sustained and detailed examination of each part of the image, both foreground and background. Stereopsis has hysteresis: once you have begun to fuse an image, it becomes clearer as you look. The further your eyes are away from the paper, the greater the depth effect. If you see two interlocking rings instead, your eyes have converged incorrectly, twice as wide as they should be. (This and other artifacts are described in the article.) Re-examine the convergence dots and strive to make the four images into three. If you see the annulus behind the plane rather than in front of it, you are viewing the stereogram with your eyes crossed, converging on a point in front of the paper, instead of having them “diverged”, converging on a point behind the paper (some people call the latter “wall-eyed or “boss-eyed”). Some people are able to cross their eyes or diverge them at will; others can only diverge them.

Although you may at first capture the 3D picture without using the convergence marks, it is worth learning to use them properly. If you wear spectacles to correct for short-sightedness, it is advantageous to remove them: your natural focus will be closer than the patterns appear to be when the eyes are correctly converged. If your spectacles also correct for other sight defects, such as astigmatism, removing them may not help.

Give yourself plenty of time: it is often necessary to work for some minutes to achieve the full 3D effect the first time around. Viewing becomes much easier with practice.

If you have real difficulty with viewing, photocopy two identical transparencies from Figure B1a, line them up carefully, and slide one horizontally over the other until the annulus appears. Although the picture is not in stereo, this will at least serve to convince your eyes that there *is* an annulus hidden in the dots! Two identical transparencies are recommended, rather than one and the original, since the heating and mangling in the photocopier will tend to stretch the transparencies, and the annulus may only be visible if both images are distorted similarly.

Box 2: Previous method for generating single image random dot stereograms

Previous methods of generating autostereograms work by creating a random vertical strip of dots and repeating the pattern along each horizontal scan line independently, modulating it by the depth of the object at that point (Tyler & Clark, 1990; Slinker & Burton, 1992). We will describe this process more precisely using a quantitative example, but first we need some terminology.

Suppose the object comprises a foreground called the near plane and a background called the far plane; both lie behind the image screen. In Figure B1a, for example, the annulus is the near plane and the surface it floats over is the far plane, while the screen corresponds to the piece of paper on which the Figure is drawn. Consider the two rays from a point on the far plane to the eyes. These intersect the screen at two pixels, the distance between which is called the stereo separation corresponding to the far plane. If the background is the same distance behind the screen as the screen is from the eyes, then this stereo separation is half the distance between the eyes, say 1.25 in (half of 2.5 in, a typical eye separation).

The width of the initial strip should be the stereo separation corresponding to the far plane. Suppose this is 90 pixels, corresponding to approximately half the distance between the eyes, measured in pixels on a typical display screen with 72 pixels/in. To give a depth impression of the far plane, the first dot would be repeated at position 91, the second at position 92, and so on. At the point where the foreground object begins, the dots are repeated at a smaller lag, say 72 pixels instead of 90, corresponding to the stereo separation of the near plane. To portray an object that lies between the near and far planes, an appropriate interpolation between 72 and 90 pixels would be used for the lag.

This is a simple but technically incorrect algorithm for generating an autostereogram, which, nevertheless generally produces reasonable results. Copying the initial randomly-selected strip of dots gives a directional bias that belies the symmetry of the actual physical situation, and some people have reported difficulty viewing such images. Viewing the generation process as a constraint-satisfaction problem, as described in the article, yields a procedure that is correct and not much more complicated, and has no left/right bias.

Box 3: The advantage of symmetric eye convergence

For stereoscopic depth to be perceived across a range of depths without seeing double, the angular separation of corresponding parts of the retinas' images must not be too great. Specifically, the angle subtended on a retina—the stereo disparity—must be less than about 0.5°. Under certain circumstances, larger or smaller disparities can be maintained. For example, a rod held on the nose and pointing away from the face subtends too high a disparity to be seen single near the face, but you "know" it is a single rod, and this impression suffices to sense, if not patently see, depth uniformly along its length.

The range of angles subtended on a retina depends in which direction the eye itself is pointing. Suppose one eye is converged inwards at a fixed angle θ to a perpendicular to the image plane. Setting θ higher for one eye reduces it for the other, given that both eyes converge on the same point in the image plane. Hence, equal, or symmetric, eye convergence permits the largest range of depth change within any given limitation of disparity. Take the eye separation to be 2.5 in, μ=1/3, the viewing distance to be 15 in from the eye to the image plane, and consider a symmetric convergence angle $\theta = \arctan(E / 4D)$. Then all depth changes ($0 \le z \le 1$) have a disparity less than 0.5°.

The previous algorithm (Box 2) assumes that one eye looks straight at the image (θ=0). For the full range of z and the same viewing conditions, this results in a huge disparity of 5.6°. Put another way, for a disparity of at most 0.5° the effective range of z is halved. Although this assumption makes constructing the image easy (because each pixel only images *one* point on the object for the θ=0 eye; see Box 2), it distorts the image asymmetrically and reduces the complexity of the images that some people can see clearly. Note that a slight rotation of the head has no affect on these arguments, since the eyes continue to converge at the same point (except for the minor change and unequal eye distance to the image).

In contrast, the advantages of the symmetric method we describe are twofold: the range of stereo disparity required for viewing is reduced, and parallax distortion is reduced and also symmetric. Further advantages are hidden surface removal and correct depth interpolation, though these refinements could also be applied to the previous algorithm. Although the symmetric convergence approach places multiple constraints for both eyes on each image pixel, we have shown that these constraints are readily solved.

List of figures

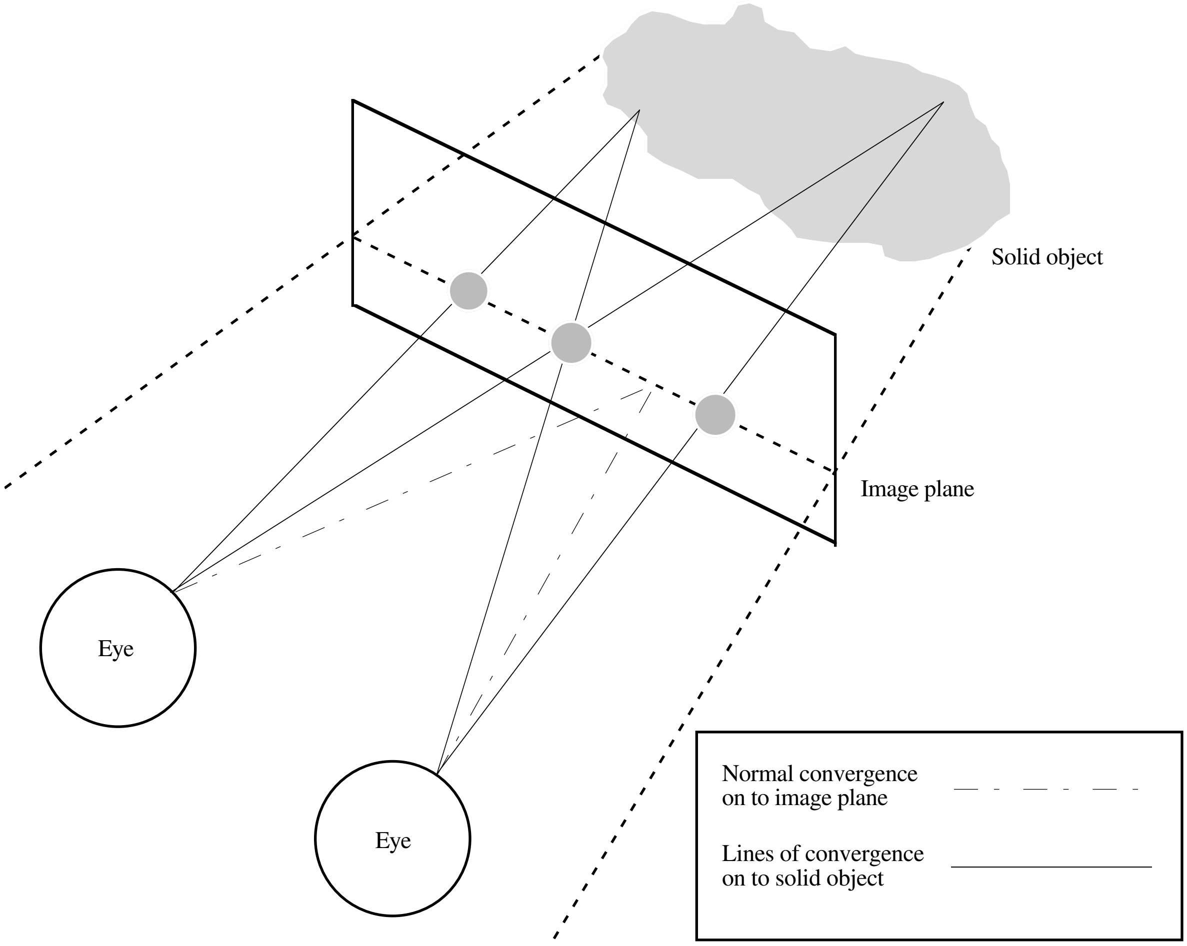


Figure 1 Simulating a solid object on a 2D image plane

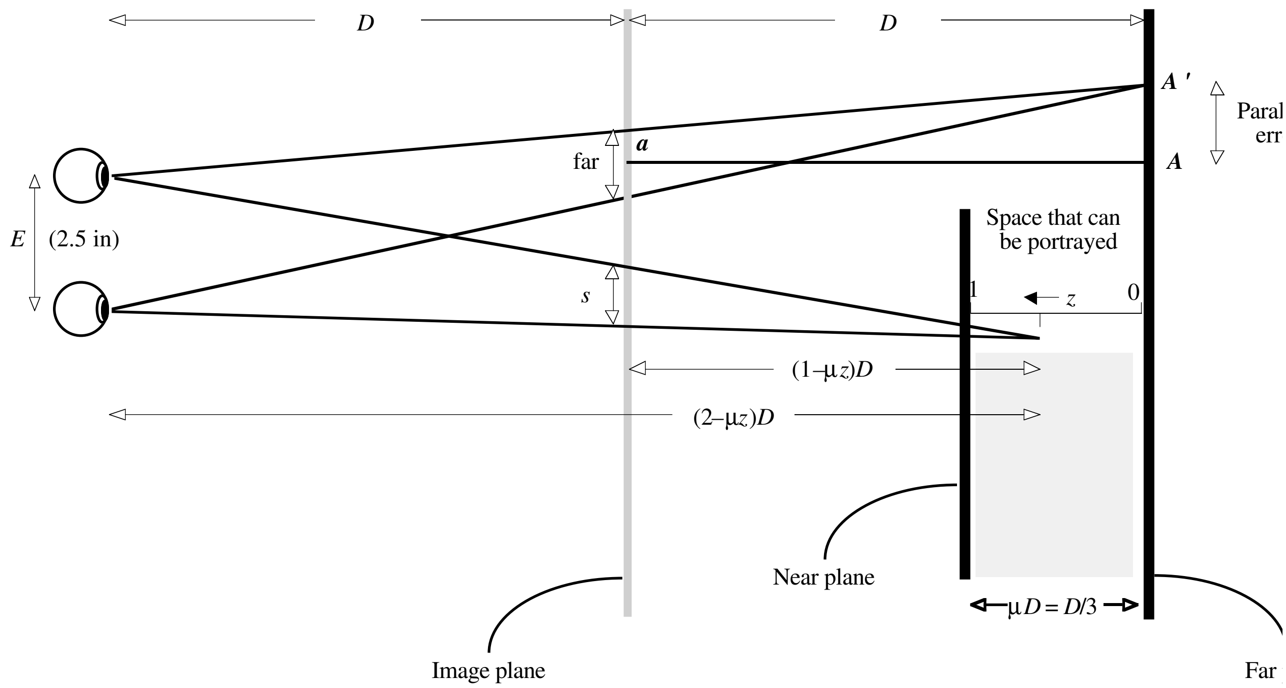


Figure 2 Geometry on which the program is based

```
/*  Algorithm for drawing an autostereogram                                         */

#define round(X)  (int)((X)+0.5) /*     Often need to round rather than truncate */
#define DPI 72                   /*         Output device has 72 pixels per inch */
#define E round(2.5*DPI)         /*       Eye separation is assumed to be 2.5 in */
#define mu (1/3.0)               /* Depth of field (fraction of viewing distance) */
#define separation(Z) round((1-mu*Z)*E/(2-mu*Z))
                                 /* Stereo separation corresponding to position Z */
#define far separation(0)        /*      ... and corresponding to far plane, Z=0 */
#define maxX 256                 /* Image and object are both maxX by maxY pixels */
#define maxY 256

void DrawAutoStereogram(float Z[][]) {
                                 /*  Object's depth is Z[x][y] (between 0 and 1) */
    int x, y;                    /*              Coordinates of the current point */
    for (y=0; y < maxY; y++) {   /*         Convert each scan line independently */
        int pix[maxX];           /*                           Color of this pixel */
        int same[maxX];          /*           Points to a pixel to the right ... */
                                 /*      ... that is constrained to be this color */
        int s;                   /*        Stereo separation at this (x,y) point */
        int left, right;         /* X-values corresponding to left and right eyes */

        for (x=0; x < maxX; x++)
            same[x] = x;         /*   Each pixel is initially linked with itself */

        for (x=0; x < maxX ; x++) {
            s = separation(Z[x][y]);
            left = x - s/2;      /*                  Pixels at left and right ... */
            right = left + s;    /*                      ... must be the same ... */
            if (0 <= left && right < maxX) { /*               ... or must they? */
                int visible;     /*        First, perform hidden-surface removal */
                int t=1;         /*  We will check the points (x-t,y) and (x+t,y) */
                float zt;        /*           Z-coord of ray at these two points */

                do {
                    zt = Z[x][y] + 2*(2 - mu*Z[x][y])*t/(mu*E);
                    visible = Z[x-t][y]<zt && Z[x+t][y]<zt; /*  False if obscured */
                    t++;
                } while (visible && zt < 1); /*  Done hidden-surface removal  ... */
                if (visible) {   /*        ... so record the fact that pixels at */
                    int l = same[left]; /*        ... left and right are the same */
                    while (l != left && l != right)
                        if (l < right) { /*    But first, juggle the pointers ... */
                            left = l; /*         ... until either same[left]=left */
                            l = same[left]; /*            ... or same[left]=right */
                        }
                        else {
                            same[left] = right;
                            left = right;
                            l = same[left];
                            right = l;
                        }
                    same[left] = right; /*    This is where we actually record it */
                }
            }
        }
        for (x=maxX-1 ; x>= 0 ; x--) {/*    Now set the pixels on this scan line */
            if (same[x] == x) pix[x] = random()&1;/* Free choice; do it randomly */
            else pix[x] = pix[same[x]]; /*   Constrained choice; obey constraint */
            Set_Pixel(x, y, pix[x]);
        }
    }
    DrawCircle(maxX/2-far/2, maxY*19/20); /* Draw convergence dots at far plane, */
    DrawCircle(maxX/2+far/2, maxY*19/20); /*      near the bottom of the screen */
}
```

Figure 3 Algorithm for drawing an autostereogram

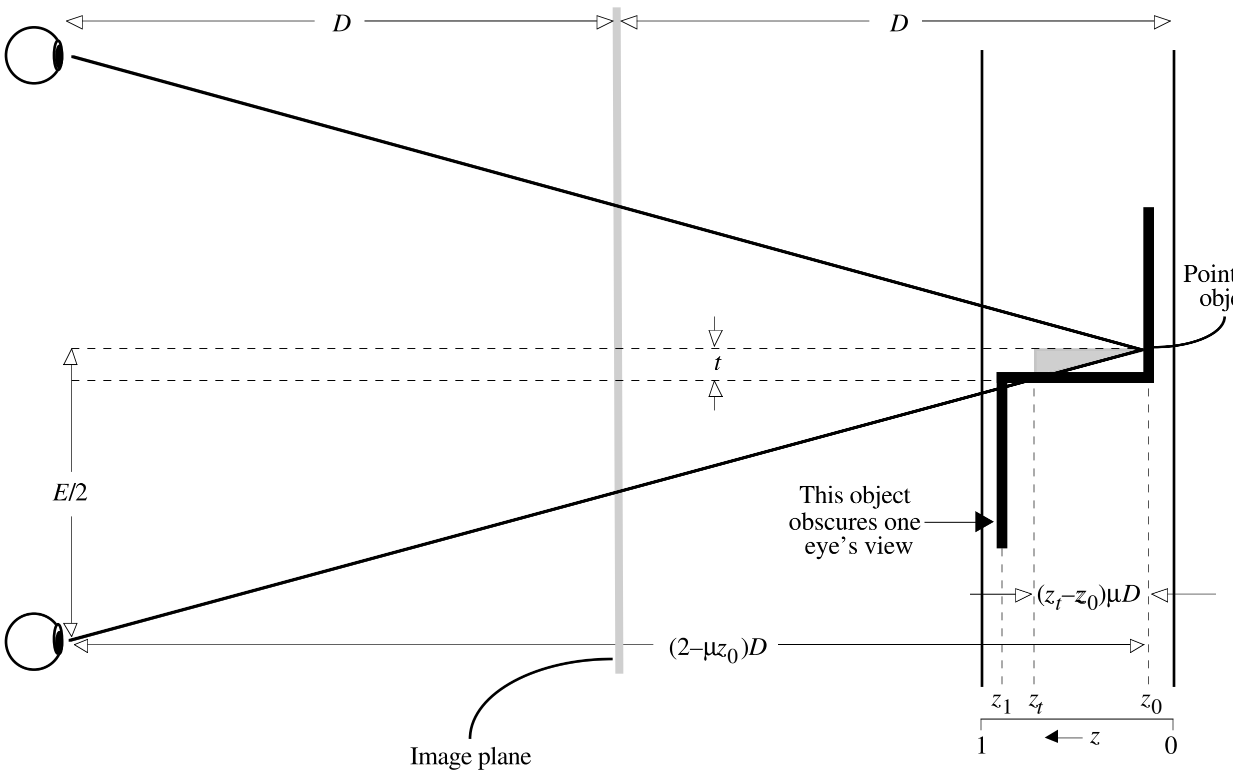
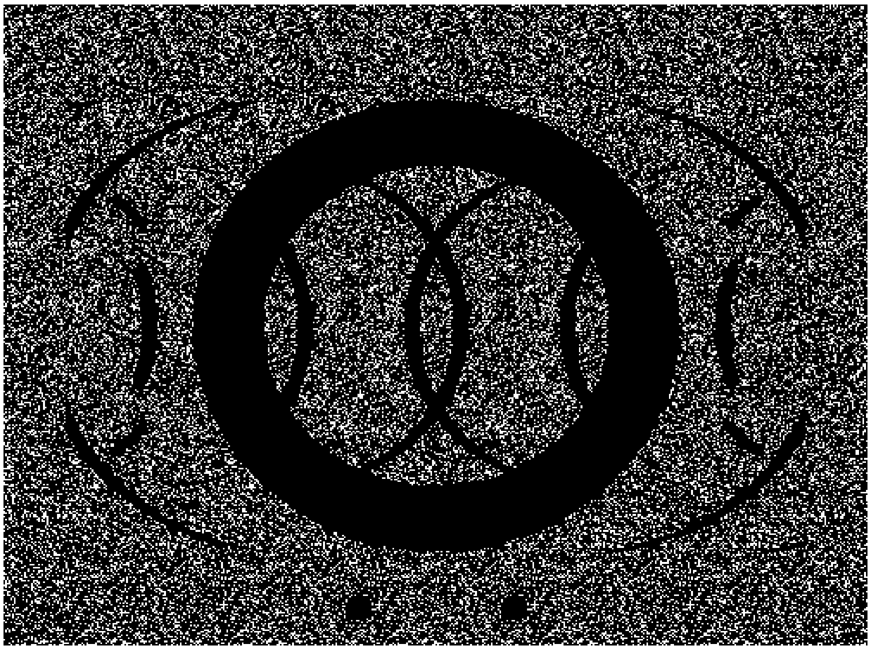
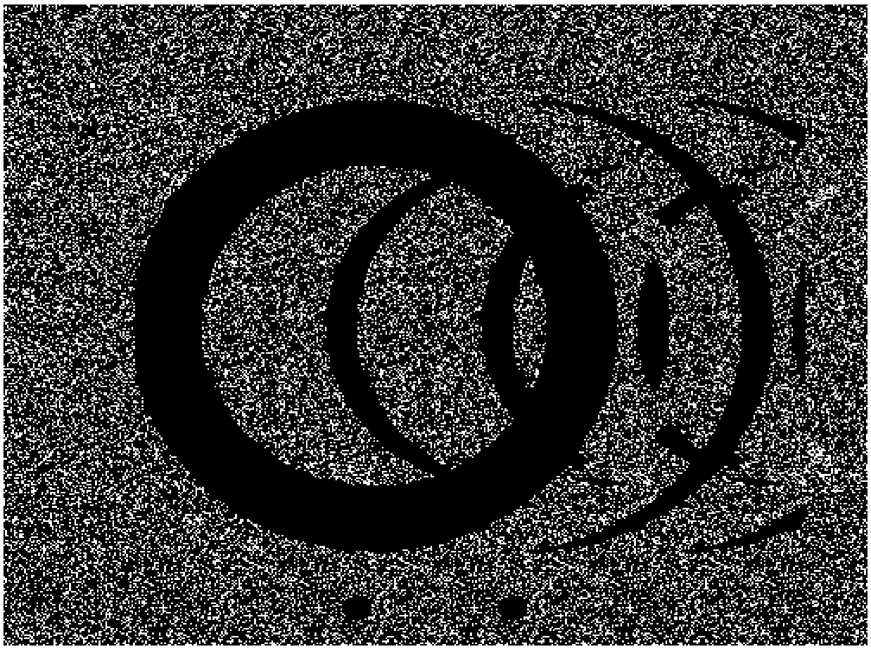


Figure 4 Geometry for hidden surface removal



(a)



(b)

Figure 5 Echos that can be seen in the same plane as the annulus
(a) For the algorithm of Figure 3 without hidden-surface removal
(b) For the previous algorithm, sketched in Box 2

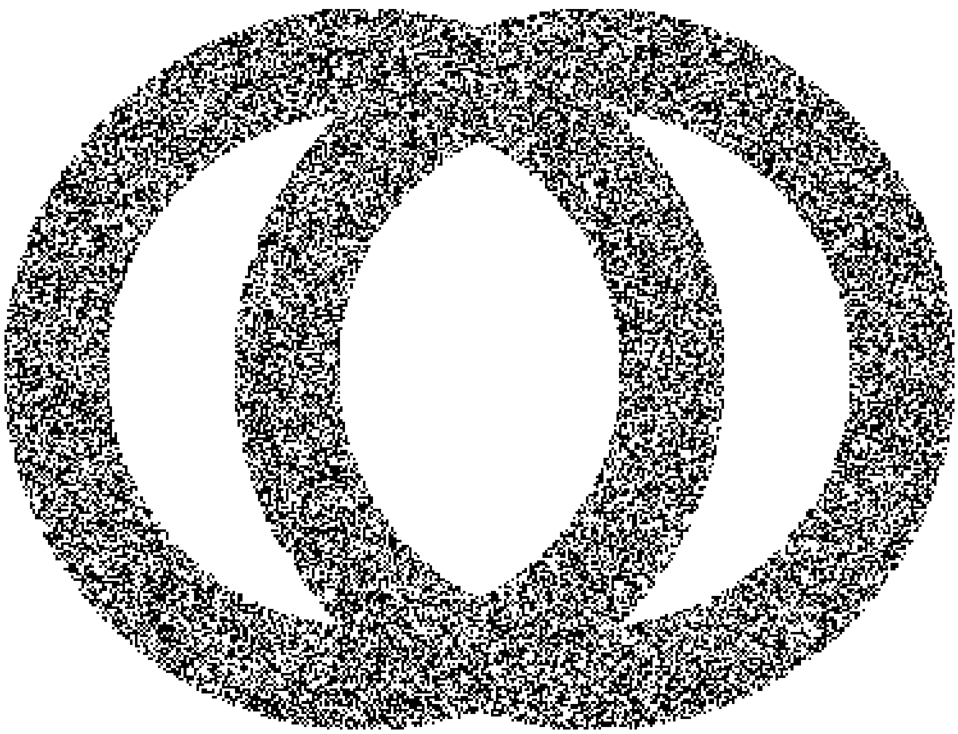


Figure 6. False fusion caused by incorrect convergence of the eyes

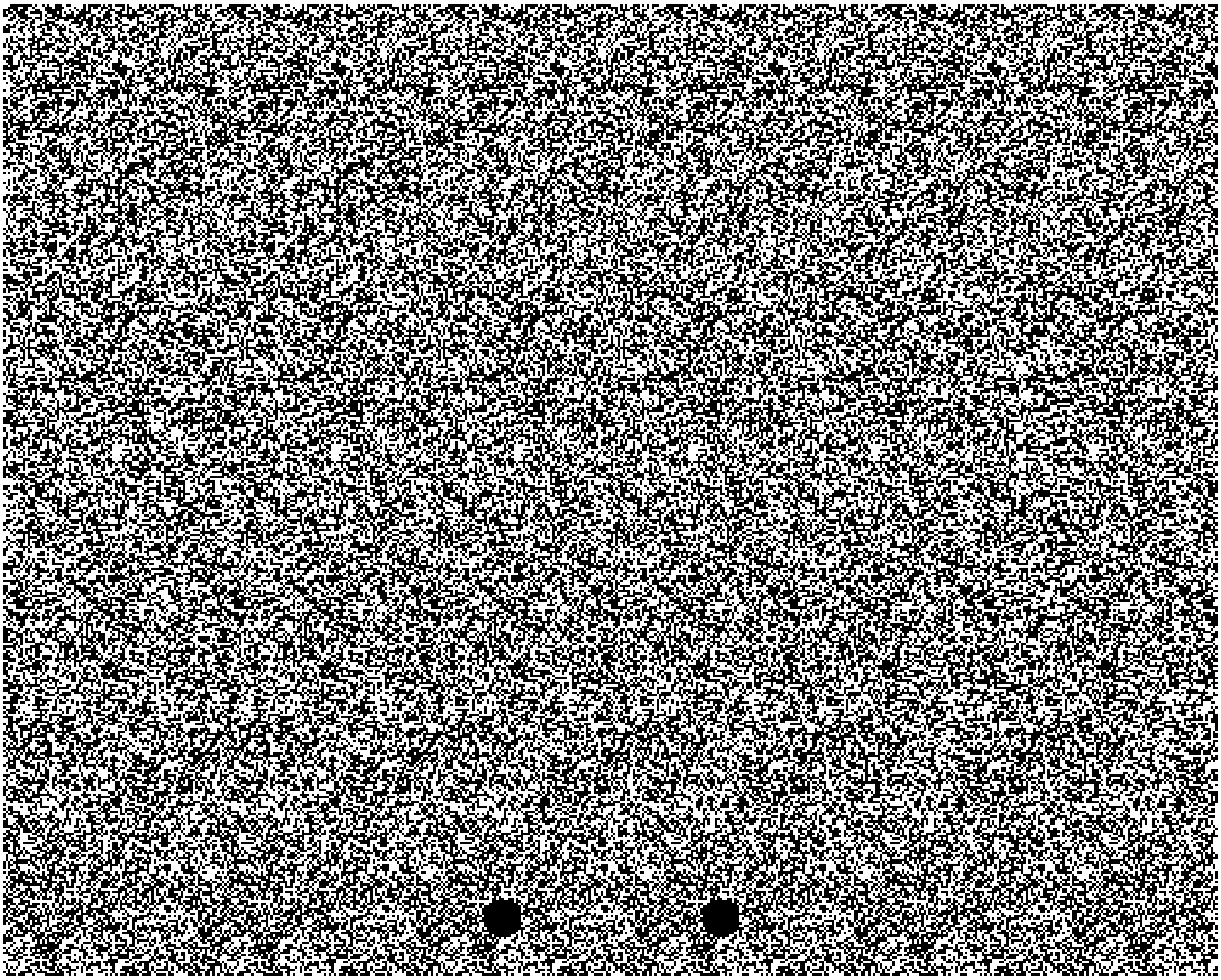


Figure 7. Stereogram showing a hemisphere

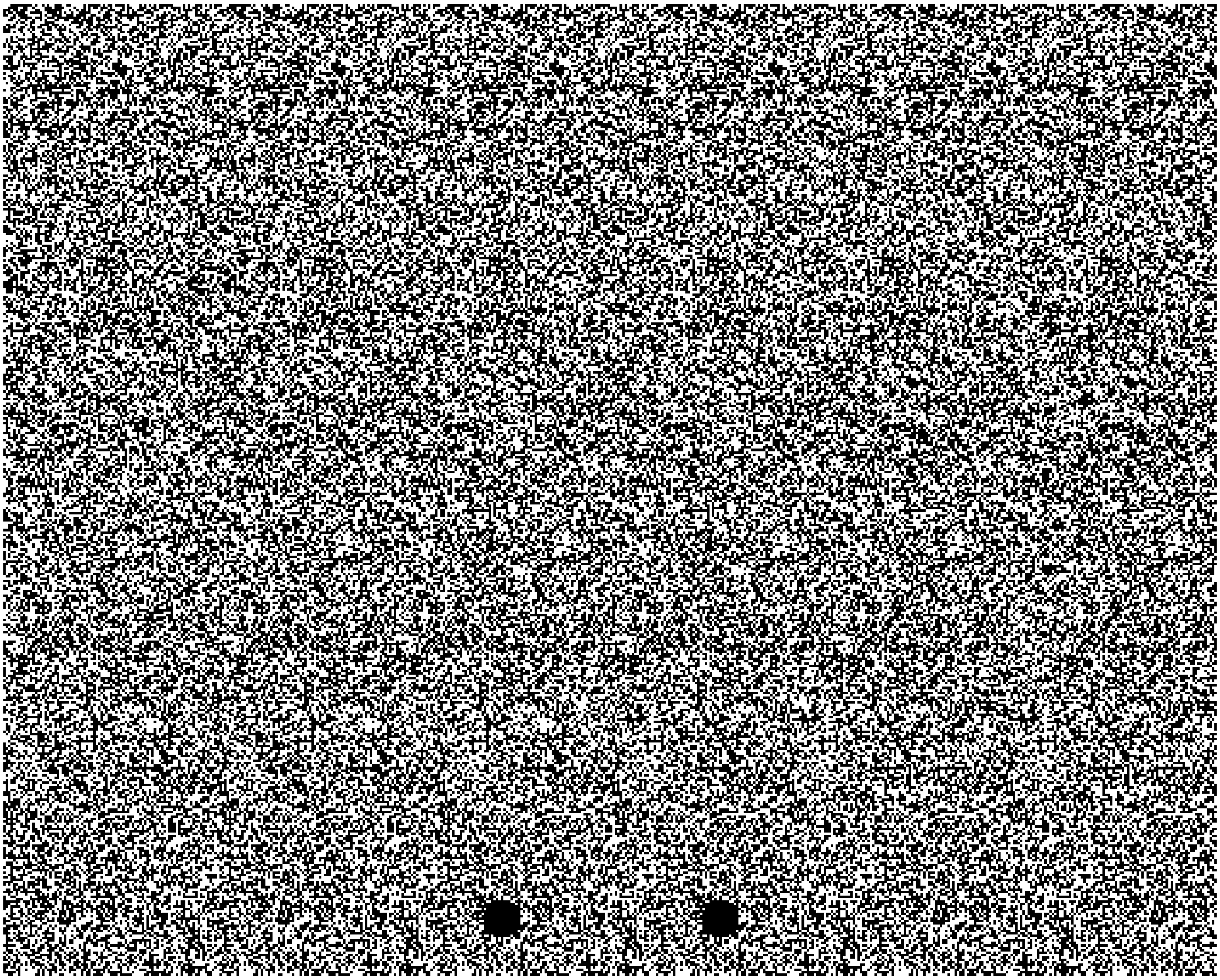


Figure B1 (a) Stereogram showing an annulus floating above a surface

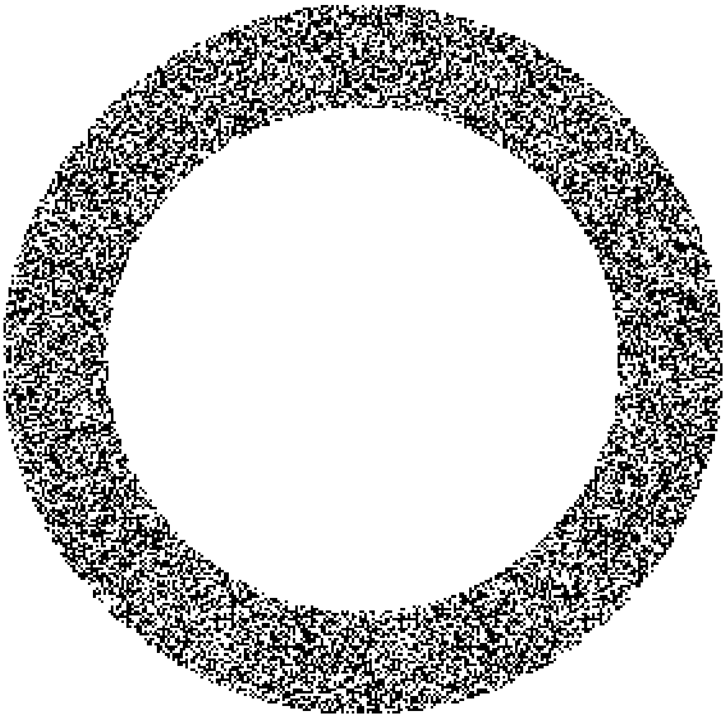


Figure B1 (b) the annulus itself