SEEING BETWEEN THE PIXELS

Brian Hayes

A reprint from

American Scientist

the magazine of Sigma Xi, the Scientific Research Society

Volume 87, Number 3
May–June 1999
pages 202–207

SEEING BETWEEN THE PIXELS

Brian Hayes

The French painter Georges Seurat had a lively interest in the sciences. His technique of pointillism—painting with tiny flecks of pure colors—was inspired by theories of light and perception that trace back to James Clerk Maxwell and Hermann von Helmholtz, among others. Unfortunately for Seurat, the scientific underpinnings of pointillism did nothing to soften criticism of his paintings when he exhibited them in the 1880s, but his ideas have certainly been vindicated since then. A century later we are all pointillists. We tend to see every image as a collection of little dots.

Decomposing a picture into dots seems natural today because that's the way images are displayed on a computer monitor or a television screen—by lighting up spots of phosphor that glow red, green and blue. Printed images are just as dotty. Under a magnifying lens, the pictures in this magazine dissolve into dots of cyan, magenta, yellow and black. Ink-jet printers spray similar arrays of microscopic colored droplets.

From printing and displaying pictures as dots, it's an easy step to storing them that way inside the computer—as rows and columns of "pixels," or picture elements. Photographs made with a digital camera are born as pixel arrays; satellite imagery and other kinds of scientific data also arrive in this format; conventional photographs can be converted into pixels by a scanner or even by a fax machine. In the case of a monochrome image, each pixel is represented in computer memory by a number that gives the pixel's brightness on a scale from black to white. For color images, a pixel typically consists of three or four numbers, encoding the intensities of the component colors. A snapshot-size color image might have a million pixels and fill up a few megabytes of computer memory.

With the prevalence of digital imaging, the rectangular array of pixels begins to seem like the one and only way of representing graphic information; you can't look at a picture without seeing spots before your eyes. But there are many alternatives to pixels, with interesting and useful properties. Some of the pixelless representations are more compact; some preserve information about the structure and meaning of an image; some are easier to revise or edit; some can be displayed at various resolutions without loss of quality. Perhaps most intriguing, some of the alternatives could give hints about the way the brain stores and interprets images.

Brian Hayes is a former editor of American Scientist. *From January to June 1999 he is Journalist in Residence at the Mathematical Sciences Research Institute in Berkeley. Internet: bhayes@amsci.org.*

Straightedge and Compass

Pixels were not always the universal building blocks of computer graphics. In the 1960s, many computer display screens worked on another principle altogether, called vector graphics. Instead of scanning a beam over the entire surface of the screen in closely spaced parallel lines (the *raster* pattern of modern displays), the beam was steered from point to point as needed to create a line drawing. For example, directing the beam from the x,y coordinates 0,0 to the point 6,0, then to 0,6 and finally back to 0,0 would draw an isosceles right triangle. The corresponding hardcopy device was the pen plotter, a kind of automated Etch-A-Sketch in which a motor-driven pen zipped from point to point over a sheet of paper.

Specialized hardware for vector graphics is now a rarity, and yet the underlying idea has not gone away. Even though modern displays and printers are raster devices, many computer programs adopt the vector format as an internal data structure for graphic objects. Computer-aided-design software works this way; an architect's rendering of a building is made up of lines, not pixels. Often, vector-based programs are referred to as "drawing" software, whereas pixel-based programs are for "painting." (To those who work in the graphic arts, the difference between vectors and pixels is the difference between Adobe Illustrator and Adobe Photoshop.)

Although the term "vector graphics" suggests drawing with straight lines, modern programs offer a compass as well as a straightedge, and even a French curve. Indeed, vector systems can depict the entire universe of planar Cartesian geometry—any form described by equations in x and y. Linear equations yield lines or line segments; quadratic equations describe circles and other conic sections; cubic equations define a particularly im-

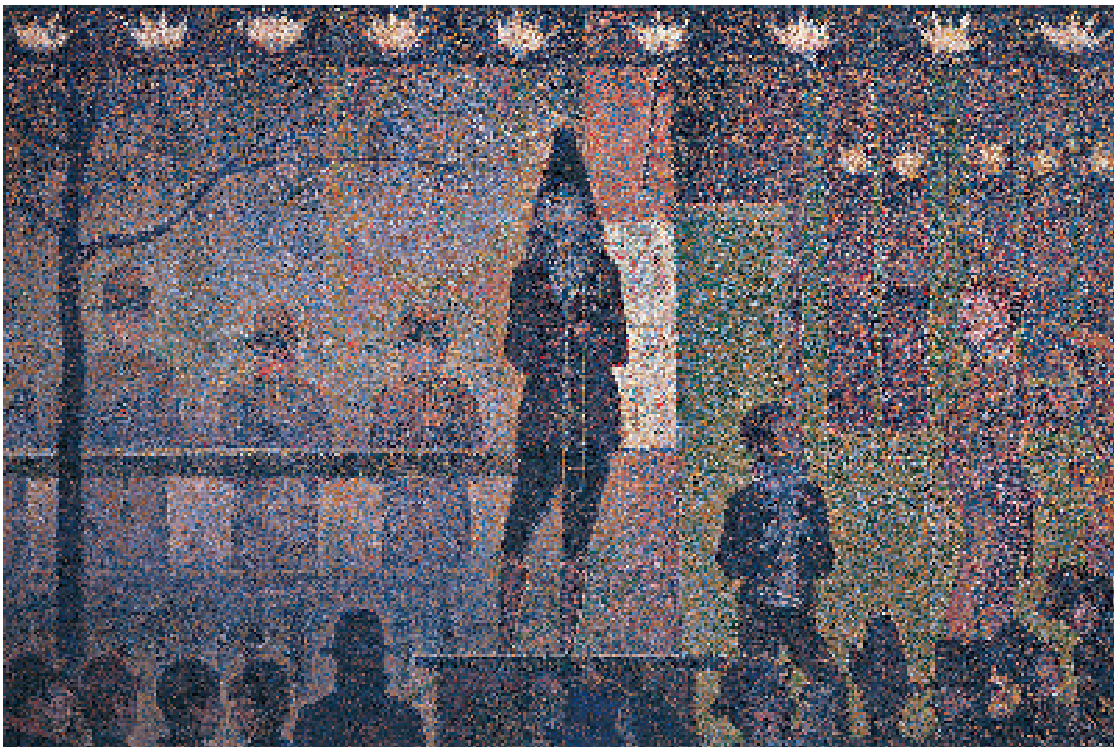

Figure 1. *La Parade de Cirque*, **which Georges Seurat painted in 1887–88, is composed entirely of colored dots, much like the pixels of modern digital images. The painting is now in the Metropolitan Museum of Art in New York.**

portant class of curves are called splines, after the flexible strips of metal or wood that were once used to outline smooth curves in shipbuilding.

At least one common vector-graphics system evolved from a language for driving pen plotters; it is HP-GL, the Hewlett-Packard Graphics Language. But the most widespread notation for vector graphics is Postscript, developed by Adobe Systems. Every letter on this page was created as a sequence of line and arc and spline commands in Postscript.

Postscript is a versatile language that can accommodate pixel-based images as well as vectors. It is also a fully equipped programming language, with facilities for iteration, recursion and defining procedures. One consequence is that a finite expression in Postscript can describe a geometric object of unbounded complexity, such as the Hilbert curve, a one-dimensional line so contorted that it fills a two-dimensional area. (But is the Postscript encoding a representation of the image or merely a program that when executed generates a representation?)

An important advantage of vector formats is that geometric objects retain their identity and structure. In a vector-based drawing program you can select a line and alter its thickness or color. These operations are much harder with an image made up of nothing but pixels, where the concept of "line" does not exist. In many cases vector drawings are also more compact than pixel-based images. And a final virtue of vector formats is that the same drawing can be displayed or printed at any resolution. For pixel images, the only way to show finer detail is to squeeze in more pixels, but the equations that define vector forms are mathematical objects of potentially unlimited precision. The same Postscript letterforms can be rendered on a monitor screen at less than 100 dots per inch or on film at more than 1,000 dots per inch.

Brushstrokes

Vector formats are ideal for mechanical or geometric drawings, but it's hard to imagine that Seurat would have been very happy working in vector mode. On the other hand, he might have been delighted to try a form of computer painting de-

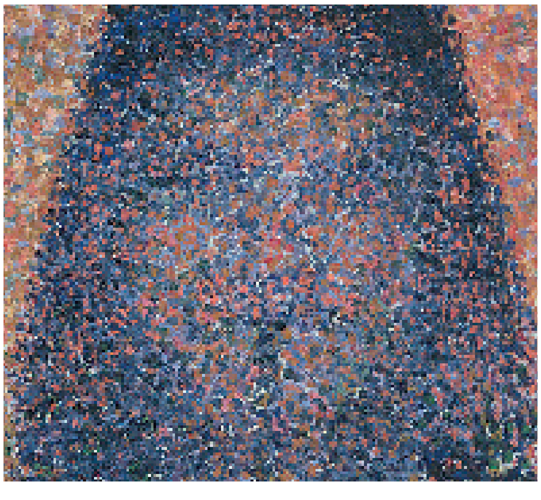

Figure 2. Detail of *La Parade* **shows Seurat's pointillist technique. Inspired by optical science, he believed colors should mix in the eye, not on the artist's palette.**

vised by Paul Haeberli of Silicon Graphics. Haeberli's program captures something of the kinesthetic experience of painting, and in skilled hands the program can produce distinctly painterly results. It achieves this through a strategy that seems perfectly obvious once someone else has thought of it: The program represents a painting not as an array of pixels or as a collection of lines or splines but as a sequence of brushstrokes.

To paint with Haeberli's program you begin with an existing template image in pixel format, and next to it a blank canvas. As you run the mouse over the canvas, the program chooses a color from the corresponding region of the template, and paints a brushstroke in that color on the new canvas. But the brushstrokes do not merely reproduce the pixel structure of the template; brushes have numerous adjustable properties, including size, shape, orientation and texture. Indeed, the computerized brushes can do things that no brush of pig bristle or camel's hair could achieve, such as spraying squares and circles of color or tiling the surface of the canvas with the polygons known as Dirichlet domains. The result is a highly stylized and often impressionistic version of the template image.

Weird and wonderful brushes are not unique to Haeberli's software; several other painting programs have a wide selection of them. But other painting software does not represent the finished artwork as a list of brushstrokes; instead the brushes merely spray pixels onto the image surface, so that any subsequent operations have to be done at the pixel level. In effect, you are left with a flat photograph of a painting, whereas the brushstroke format is more like a movie of the artist at work—a movie so precise and detailed that it reproduces the exact motions needed to apply all the layers of paint to the canvas.

Storing brushstrokes opens up a number of possibilities for image manipulation that simply cannot be done either with pixels or with conventional pigments on canvas. At the simplest level, you can undo a brushstroke if you make a mistake. Beyond that is the possibility of "post-processing" the brushstrokes. For example, you might experiment with changing the diameter or length or direction of the brushstrokes after a painting is completed, or you might use one template image to determine the color of each brushstroke and another image to control brush orientation. You could even change the order in which the brushstrokes are layered onto the canvas, perhaps sorting them by color.

The most elaborate versions of Haeberli's painting program run only on Silicon Graphics workstations, but a Java applet called the Impressionist has some of the basic functions. It is available at http://www.sgi.com/grafica/index.html and should run on any Java-equipped computer.

Squeezing the Pixels

Both the strengths and the weaknesses of pixel-based graphic formats lie in the absence of hierarchical structure in the image. On the one hand, having only a flat array of pixels to worry about facilitates many useful kinds of image processing, such as filtering to sharpen edges, or adjusting contrast and color balance. On the other hand, pixels are maddeningly oblivious to what they are portraying. You might look at an image and see parasols, rowboats, picnic baskets and top hats, but none of those objects exist in the computer file. There are only pixels there, ignorant of the patterns they form.

Extracting meaningful structures from image data is a notoriously difficult problem; it's called vision. But there are simpler ways of imposing structure on an array of pixels. These methods cannot pick out top hats or parasols, but they do identify certain patterns and regularities. Much of the work on such pixel patterns derives from efforts to compress images into a smaller space.

Image compression is possible only because scenes of the natural world make up a small and rather special subset of all possible arrays of pixels; they are not random data but have inherent redundancies and correlations. The simplest compression scheme is run-length encoding. If you list pixel values in raster-scan order, you often discover several pixels in a row with the same color. You can then save space by writing down the color only once, along with the number of times it should be repeated. An image compressed in this way is really no longer an array of pixels; it is a collection of lines of various lengths, which are fitted together like planks in a hardwood floor to reconstruct the complete image.

Run-length encoding exploits correlations only along a single dimension. Another way of breaking down and reassembling a picture, called a quadtree, takes advantage of both vertical and horizontal regularities. To build a quadtree, take a square image and average all the pixel values,

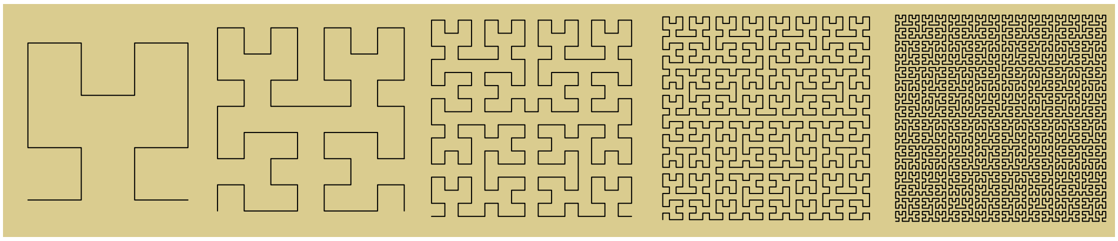

Figure 3. Vector drawings, made up of line segments instead of pixels, show five stages in the construction of the space-filling Hilbert curve. Postscript, a programming language as well as a graphics notation, represents such objects concisely.

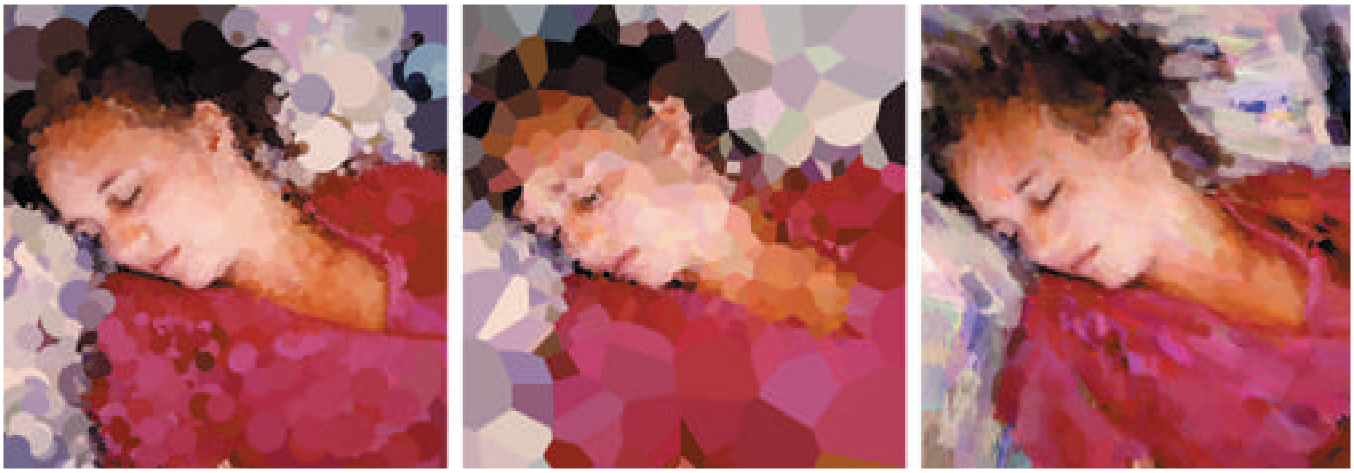

Figure 4. Three renderings of a single image were created by Paul Haeberli. Each version is a sequence of brushstrokes; one brush scatters colored disks, another draws Dirichlet domains (which look like the interior of a foam) and the third gives a more conventional painterly effect.

so that the entire area is a flat expanse of color expressed by a single number; this is the first and crudest level of the quadtree. Now divide the original image into four square quadrants, and average the pixels within each of these regions. The four averages become the four branches at the next level of the quadtree. Continue in the same way, dividing each quadrant into subquadrants, until eventually you reach the level of single pixels. (The construction is easiest when the image is a square whose side is a power of 2, but other shapes and sizes can be accommodated with a little extra effort.)

Quadtrees offer a multiresolution view of pixel data. Looking at just the first few levels of the tree gives a rough impression of the image content; continuing to further levels adds progressively more detail. Some images posted on the Web, called "progressive JPEGs," work in just this way, starting out as blocky approximations and growing sharper through gradual refinement.

A fundamental property of quadtrees is that pixels stored in the same node of the tree are also close together in the image. As a result, sibling nodes tend to be highly correlated, creating an opportunity for image compression. After averaging the color in a quadrant, you can record not the average itself but the difference between the average and the color of the quadrant's parent node. These parent-child differences tend to be small, and so they can be given a very compact encoding.

Surely the most unusual image-compression technique is one based on the idea of fractals, or self-similar patterns. Certain features of the natural landscape are famously fractal—coastlines, mountain ranges and ferns being the favorite examples. An idealized black spleenwort fern has become the mascot of the fractal industry. Given an image of the entire fern, you can make a number of photocopies at reduced size, then rotate and slide the smaller images until they exactly cover the original. By applying the same transformation to any one of these frond images, you create even smaller frondlet images that exactly cover the frond. In nature this process comes to an end after just a few iterations, but mathematically it can be continued indefinitely. It suggests a curious but compact way of representing the fern image—just store a list of instructions for scaling, rotating and translating photocopies. There's no need to keep any form of the original image. Starting with an arbitrary image and repeating the list of transformations enough times, you can generate an image that approximates the fern as closely as you wish.

This is wonderful news for fern-fanciers, but what about images of other subjects? As it turns out, patterns amenable to this kind of treatment are more common than you might guess. Any image can be represented as a "collage" of fractal parts. The mathematical ideas behind this process were first explored by John E. Hutchinson of the Australian National University and were further refined and applied to image compression by Michael F. Barnsley of the Georgia Institute of Technology. The basic idea is that certain geometric transformations—combinations of scaling, rotating and translating—eventually reach a fixed point, where further operations cause no further change. If a section of an image is similar to such a fixed-point pattern, the corresponding list of transformations can serve as an approximate encoding of the section. The trick is dividing the image into the right sections and finding the right transformations.

The fractal description of an image certainly satisfies the desire for a hierarchical structure; indeed, there is no limit to the levels in the hierarchy. You can even zoom in on details—microfronds and nanofronds—that were not present in the original.

Lost in Frequency Space

Another alternative to pixels replaces an image with its spectrum.

The idea of a spectrum is most familiar in the context of one-dimensional signals, such as the sound of a flute or the light of a star. In these instances the signal is a graph of amplitude as a function of time, and the spectrum of the signal is a graph of power or intensity as a function of frequency. The conversion from the time domain to the frequency domain relies on Joseph Fourier's remarkable discovery that any periodic waveform, no matter how complex, can be constructed as the sum of simple sine and cosine waves.

Fourier analysis can be applied to a two-dimensional image by measuring "spatial frequencies" along both the *x* and *y* axes. Think of a photograph of a ladder leaning against a picket fence. If you scan across the image horizontally, recording the brightness of each point as you go, you get a spectrum with an energy peak at the frequency corresponding to the spacing of the fence pickets; when you scan vertically, the strongest peak is at a lower spatial frequency, that of the ladder rungs. The complete spectrum of an image records the contributions of all spatial frequencies, from zero up to the maximum resolution of the image data.

A form of Fourier analysis commonly used in image processing is called the discrete cosine transform. The procedure begins by dividing the image into square patches of, say, eight by eight pixels. For each such patch there are 64 discrete Fourier components, representing all possible spatial frequencies and phases along the *x* and *y* axes. Any eight-by-eight patch of the original image can be reconstructed by adding up various weighted combinations of these basic frequency elements. The zero-frequency component gives the average brightness of the patch. If the patch has gradual variations in brightness over its entire width or height, then some of the lower frequency components will be strongly represented in the spectrum. Sharp edges and fine lines in the image emphasize the high-frequency components.

Fourier analysis yields yet another method of image compression. It might seem at first that nothing is gained by converting to frequency space, since an eight-by-eight patch with 64 pixels also has 64 Fourier components. But in most images not all the Fourier components are equally important. For example, in a patch of nearly uniform color only the zero-frequency element has a significant weight, and the rest of the coefficients can be thrown away. This situation is common enough to make Fourier compression worthwhile. Indeed, the discrete cosine transform is the main compression method in JPEG images.

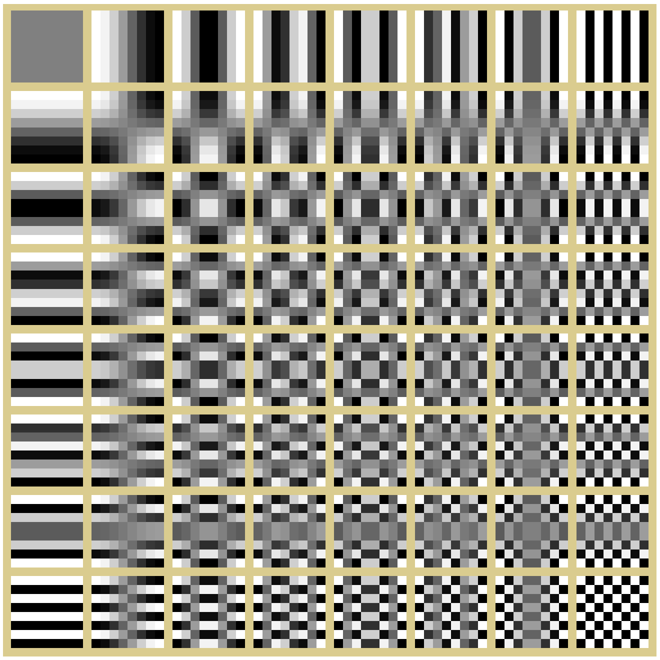
Figure 5. Basis functions for the discrete cosine transform can be combined to reconstruct any eight-by-eight-pixel grayscale image.

The Mind's Eye

When you ponder how best to represent a picture, a promising approach is to ask how people or animals see images. What kinds of data structures does the brain employ for visual information?

The input end of the visual system is clearly a pixel-based device: The retina of the eye has much in common with the array of photosensors in a digital camera. Nevertheless, we do not *see* in pixels. No one experiences the visual surround as an array of colored dots—not even Seurat. At the level of conscious awareness, what we see are faces, trees, trombones, streetlights, paintings—and all of these objects seem to have a continuous and unpixelated surface, no matter how closely we look at them. Evidently visual information is re-encoded somewhere along the neural pathways of the eye and the brain.

A part of the brain where image representation has been studied extensively is a region known variously as the primary visual cortex, the striate cortex or simply V1. In the 1960s David Hubel of Harvard University and Torsten Wiesel of the Rockefeller University recorded the response of individual V1 neurons when animals were shown various simple patterns, such as spots and stripes. These experiments and later work revealed that the "receptive field" of a typical V1 neuron has a distinctive geometry, with three main characteristics. First, the receptive field is localized: The cell responds most strongly to stimuli in a particular area of visual space. The field is also oriented: Each neuron has a favored axis for stripes or elongated features. And finally the field is most sensitive to variations in luminance over a specific size range; in other words, it has a preferred band of spatial frequencies. Thus the V1 cortex seems to classify features according to their position, orientation and angular size.

Why should the mammalian visual system favor this particular way of representing images? That's a good question, which neurobiological experiments have not yet answered. Bruno A. Olshausen of the University of California at Davis and David J. Field of Cornell University have therefore approached the problem from the opposite direction. Instead of looking into the brain for clues to how it encodes images, they look at images of natural scenes and ask what encoding would give the simplest or most efficient representation. (In this context a "natural" scene is not necessarily the forest primeval; it could be a city street, or even a page of text. What the term excludes are artificial patterns such as random visual noise.)

The technique adopted by Olshausen and Field is a distant cousin of Fourier analysis. The aim is to find a set of "basis functions" that can be combined in various proportions to generate an image. The sine and cosine functions of Fourier analysis are one such basis set, but it is not the best set for natural images. Following earlier work by John G. Daugman of the University of Cambridge, Olshausen and Field argue that the optimal basis functions are those that yield a sparse encoding for images. What this means is that any single image is likely to excite only a few of the V1 neurons. The basis set should also be complete, in the sense that it can account for the features of any natural image.

A set of functions that satisfy these criteria is not something that can be cooked up analytically. Olshausen and Field search for a sparse basis set through an iterative learning procedure, which might even resemble the mechanism by which a developing organism (or perhaps an evolving species) learns to make sense of visual input. Several images are selected as a training set, from which many small square patches are extracted at random. The functions chosen to describe these patches are initially arbitrary; they are refined by repeatedly making small changes and accepting a change if the resulting functions yield a sparser representation. With 16-by-16 patches, the procedure takes a few hours to converge on a basis set.

What kinds of basis functions emerge from this process? They look nothing like the stripes and checkerboards of the discrete cosine transform. Instead most of the functions are elongated ellipses, which seem at first glance to be scattered randomly over the square patches. On examining the entire set of functions, it turns out that nearly all combinations of position, orientation and spatial frequency are present in the set, which means that each function can respond to a specific combination of these three properties. Of course position, orientation and spatial frequency are just the features detected by V1 neurons. Finding a resemblance between the basis functions and the V1 receptive fields is not a proof that the brain employs functions of this particular form, but the result is encouraging and suggests strategies for further experimental work.

Even if we knew the brain's own graphics file format, we would not necessarily want to adopt it for computer graphics files. A biological precedent is not binding on technology. Furthermore—and here I depart on a perilous flight of speculation—the brain's encoding may be well adapted only to image analysis and understanding, not to image generation. Because of a curious asymmetry in mammalian sensory architecture, the brain has no need ever to recreate a pixel array. Although audible channels of communication go both ways—we have ears to hear with and a mouth to speak with—in the electromagnetic spectrum we have receptors but no projectors. Thus the images we receive and interpret never have to be reconstructed for display or transmission (unless you are Seurat, painting with pixels). It is easy enough to imagine a planet where creatures have organs for both input and output of images; their visual cortex would doubtless be different from ours. Maybe we should check out the V1 neurons of Teletubbies.

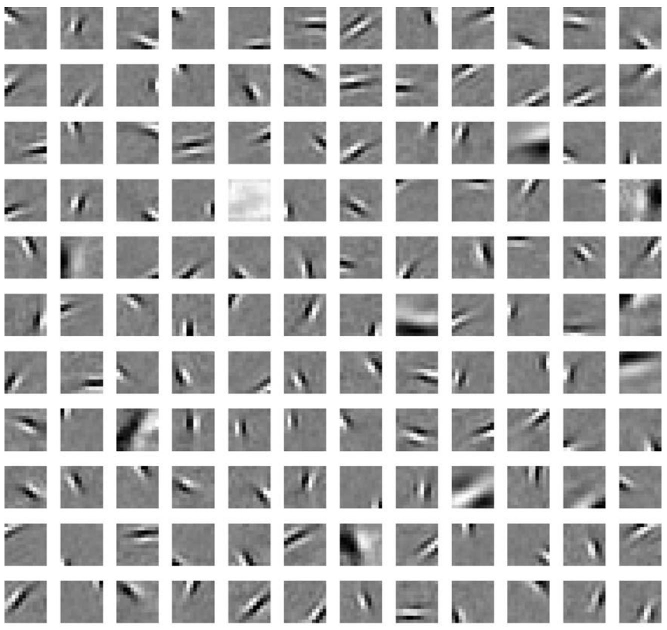
Figure 6. Sparse basis functions for 12-by-12-pixel patches were extracted from training images by an iterative learning procedure. Most of the functions are oriented, localized and cover a limited range of spatial frequencies, like the receptive fields of neurons in the visual cortex. (Image courtesy of Bruno A. Olshausen.)

Bibliography

Barnsley, Michael. F., and Lyman P. Hurd. 1993. *Fractal Image Compression*. Illustrations by Louisa F. Anson. Wellesley, Mass.: AK Peters.

Daugman, John G. 1989. Entropy reduction and decorrelation in visual coding by oriented neural receptive fields. *IEEE Transactions on Biomedical Engineering* 36:107–114.

Haeberli, Paul. 1990. Paint by numbers: abstract image representations. *Computer Graphics* Vol 24(4):207–214.

Haeberli, Paul. GRAFICA Obscura. http://www.sgi.com/grafica/index.html

Homer, William Innes. 1964. *Seurat and the Science of Painting*. Cambridge, Mass.: The MIT Press.

Hutchinson, John E. 1981. Fractals and self similarity. *Indiana University Mathematics Journal* 30(5):713-747.

Olshausen B. A., and D. J. Field. 1996. Natural image statistics and efficient coding. *Network* 7:333–339. ftp://redwood.ucdavis.edu/pub/papers/stirling.ps.Z

Olshausen, Bruno A., and David J. Field. 1996. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* 381:607–609.

Olshausen, Bruno A., and David J. Field. 1998. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research* 37:3311–3325. ftp://redwood.ucdavis.edu/pub/papers/VR.ps.Z