Behavior Research Methods, Instruments, & Computers
1989, 21 (5), 491-501

METHODS & DESIGNS

All about EVE: The Early Vision Emulation software

MICHAEL S. LANDY, LEV Z. MANOVICH, and GEORGE D. STETTEN
New York University, New York, New York

EVE, the *E*arly *V*ision *E*mulation software, is a set of computer programs designed to compute models of early visual processing. EVE may be used with a wide variety of models concerning spatial detection and discrimination, motion analysis, and issues of spatial sampling. EVE is modular and flexible. It runs under the UNIX operating system, and is device-independent. We describe the implementation of the EVE software and discuss how it may be applied to several visual models.

Computer simulation has gained in popularity recently as a means of exploring models of early visual processing. Researchers have proposed computational models for a variety of tasks, including spatial pattern detection and discrimination (e.g., Watson, 1983; Watt & Morgan, 1983; Wilson & Gelb, 1984), motion processing (Adelson & Bergen, 1985; Barlow & Levick, 1965; Marr & Ullman, 1981; van Santen & Sperling, 1985; Watson & Ahumada, 1985), and so forth. These models comprise one or more layers of processors, which analyze a visual stimulus using both linear (receptive field) and nonlinear operations. The processor outputs in the final layer are then combined in a nonlinear fashion in order to produce a number that may be related to human performance in a similar task. These models have grown increasingly complex, and a number of scientists have therefore begun to use computer simulation in order to test their models.

The model proposed by Watson (1983) for pattern detection and discrimination is a case in point. In this model, a large number of processors have linear receptive fields that respond to the visual input. These receptive fields vary in spatial position, spatial scale, phase, and orientation. They are scaled in size with retinal eccentricity. Outputs are scaled so that the entire array (of thousands of processors) has a pooled response consistent with a measured contrast-sensitivity function. The number that is related to human performance is the Minkowski vector magnitude of the set of processor outputs, which is considered to be an estimate of d' in a detection task. The model is also applicable in spatial discrimination tasks. One computes the vector difference between the outputs from each of the two patterns to be discriminated. The vector magnitude of this difference vector is the estimate of d'. Computer simulation is imperative for such a model if it is to be used to predict performance using complex stimuli.

In this paper, we describe a software system that we have devised for testing visual psychophysical models. The software is called EVE, which stands for the *E*arly *V*ision *E*mulation software. We discuss the design and some implementation details of the EVE software, its capabilities, and how it is used to compute several visual models. For a complete account of the software, we refer the interested reader to the reference manual supplied with it (Landy, 1988). There is also a technical note on the results of several model simulations carried out with EVE (Landy, Manovich, & Stetten, in press). We intend that EVE be used by researchers so that models may be shared among laboratories, and their performance be easily compared.

THE SOFTWARE SYSTEM

EVE may be used to test any visual model consisting of a number of layers of processors that are wired together in a feed-forward manner. For example, consider the model outlined in Figure 1. This model is not based on any current visual theory, but is intended simply to illustrate how models are expressed using EVE. The network consists of an input, two layers of processors, and a computation (or "linking hypothesis") that relates the output of the second layer to performance in a detection task.

This work was supported primarily by Information Science and Technology Grant IST-8418867 from the National Science Foundation, and in part by Grant ONR-N00014-K-0077 from the Office of Naval Research. Special thanks are due, as always, to Robert Picardi for technical assistance, and to J. Anthony Movshon, Sofia Würger, and Charles Chubb for careful editing. Portions of this work were presented at the annual meeting of the Association for Research on Vision and Ophthalmology, Sarasota, Florida, May 6, 1988, and published in abstract form (Manovich & Landy, 1988). Correspondence should be addressed to Michael S. Landy, Department of Psychology and Center for Neural Science, New York University, 6 Washington Place, #961, New York, NY 10003.

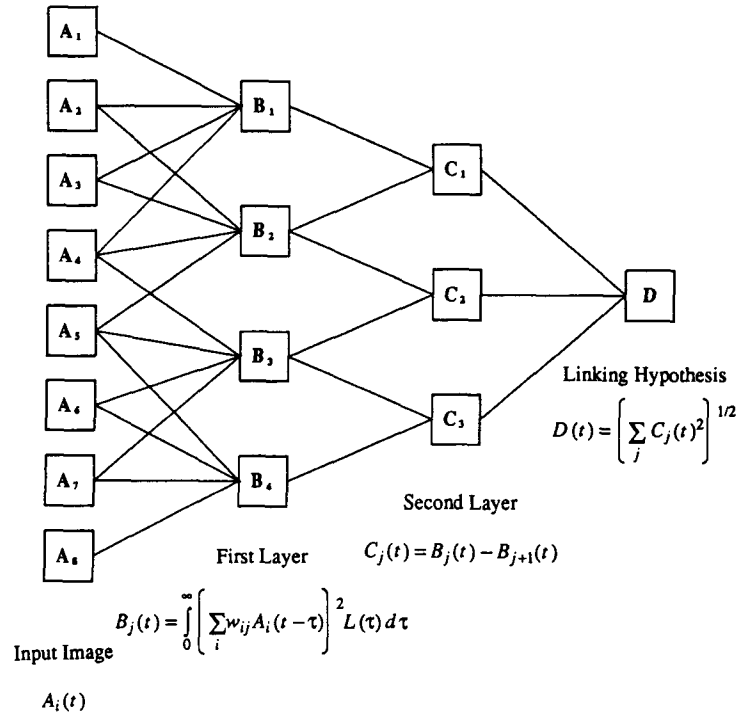

Figure 1. An example of a model that may be calculated with the EVE software. (A) The input consists of a sequence of images with luminance values $A_1(t), \ldots, A_8(t)$ at time t. (B) The first layer of processors computes the output of a set of linear receptive fields, squares the result, and then applies a temporal filter. (C) The second layer of processors computes differences between pairs of processors in the first layer. (D) A summary statistic is computed based on the outputs of the processors in the second layer. This computation represents a "linking hypothesis," which becomes the model's prediction of human performance in a detection or discrimination task using the same input stimulus.

The input is a temporal sequence of visual images. The first layer has many processors, each of which computes the output of a linear spatial receptive field, squares the result, and then applies a temporal filter. The second layer consists of processors that compute the differences between particular pairs of processors in the first layer. The final stage computes a summary statistic of the activation of the second-layer processors.

In EVE, it is quite simple to test models similar to the one we have just described. The unifying principle of the design is that the input and output of all programs in the system are treated as an *image sequence*, which can be a sequence of outputs of processors from a previous layer, a sequence of input images to be presented to the first layer in the model, or simply a description of a processor layer that has not yet been applied to an input sequence. Any given layer consists of a set of processors, each characterized by its spatial position and the way it combines processor outputs from previous layers. In Figure 2, we outline the way the first layer of the model of Figure 1 is represented in EVE. The modeling of this layer is decomposed into three parts: (1) the definition of the processor array; (2) the calculation that involves the combination of several processor output values from a previous stage (in this case, the receptive field calculation); and (3) a series of spatially pointwise operations. We define spatially pointwise calculations to be any operations that do not change the sampling positions and treat each processor independently. Examples include point nonlinearities, temporal filtering (both of which are represented in Figure 2), and the addition of independent noise. This is in contradistinction to the operations in part 2 above, which can involve more than one input processor value, and which effectively resample the input. This series of three operations is common to all processing in EVE.

The first operation for a given layer is to define the processor array by specifying the processor positions (i.e., the sampling characteristics of that layer) and the parameters that are particular to each processor (Figure 2—"create processors"). EVE allows images to be sampled in a variety of ways including square or hexagonal sampling grids, or arbitrary specification of sample positions. Each processor may be associated with a set of processor parameters, which is called a *processor structure*. For example, a processor might compute a linear receptive field as applied to the layer's input, where the receptive field has a Gabor profile. In that case, each processor will be

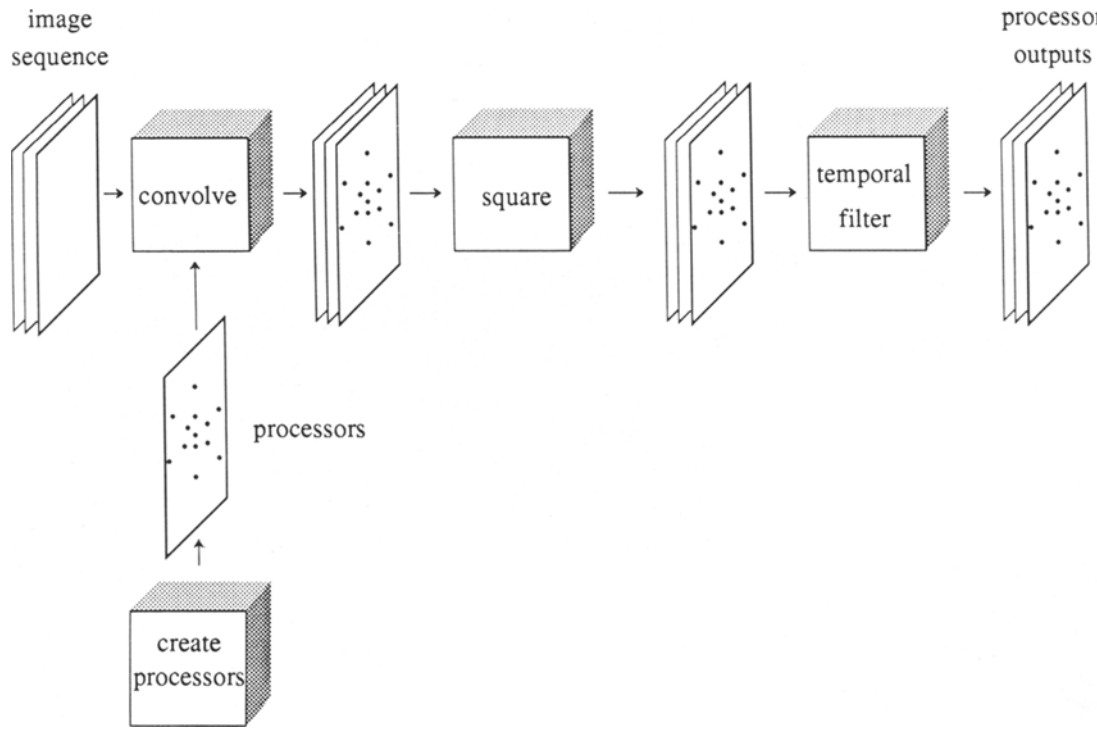

Figure 2. An outline of the way the first layer from Figure 1 is implemented with EVE. There are three operations for each layer: (1) Define the set of processors in that layer including spatial locations and processor parameters ("create processors"); (2) combine information from the previous layer or input (here the combination is effected by a linear receptive field, "convolve"); and (3) apply a series of spatially pointwise transformations (here: "square" followed by "temporal filter").

associated with a processor structure that specifies the parameters of that processor's receptive field: spatial frequency, orientation, phase, and Gaussian width.

The next computation applies the processor array to the input to this layer. For example, if the array consists of Gabor processors, then a correlation of the Gabor receptive fields with the input is computed for each processor (Figure 2—"convolve"). For each frame of the input to this layer, this results in a new frame consisting of the output of each Gabor receptive field. The sampling characteristics of the current layer were defined in the previous step; the sampling characteristics of the input to this layer can be quite different. EVE handles this change in sampling from layer to layer automatically.

Finally, having combined various values from the input to a layer, we may apply a series of spatially pointwise operations (in the sense defined above). In the current example, the correlation result for each processor is squared by one EVE program, and then temporally filtered by another (Figure 2—"square" and "temporal filter").

Other processor layers are computed in a similar manner: define a sampling array and the parameters of the processors in that array, use this array to combine outputs of a previous layer or input image (this need not necessarily be a linear combination as in the example), and then apply a series of pointwise operations to each processor (instantaneous nonlinearity, temporal filtering, addition of noise, etc.). The form of the model need not be a simple sequence of layers as in Figure 1, but can involve any feed-forward arrangement of processor layers (e.g., involving parallel channels; see Figure 3).

Implementation

EVE is flexible in its ability to test visual models, because it is modular. Each simple operation is computed by a single EVE program. For example, there are dozens of programs in EVE that compute spatially pointwise operations, and one can use them in any desired combination. The data passed between all EVE programs are in a standard image format. Each image consists of an *image header* followed by data concerning processor positions, processor parameters (called *processor structures*), and processor output values. The design was based on the HIPS image processing system (Landy, Cohen, & Sperling, 1984a, 1984b), but it extends that software in several ways critical for use in vision modeling. The software is written entirely in the C programming language (Kernighan and Ritchie, 1978), and it relies to a certain extent on the facilities of the UNIX operating system (Kernighan & Mashey, 1981; Ritchie & Thompson, 1978). EVE can be used on any machine that runs UNIX, and it is there-

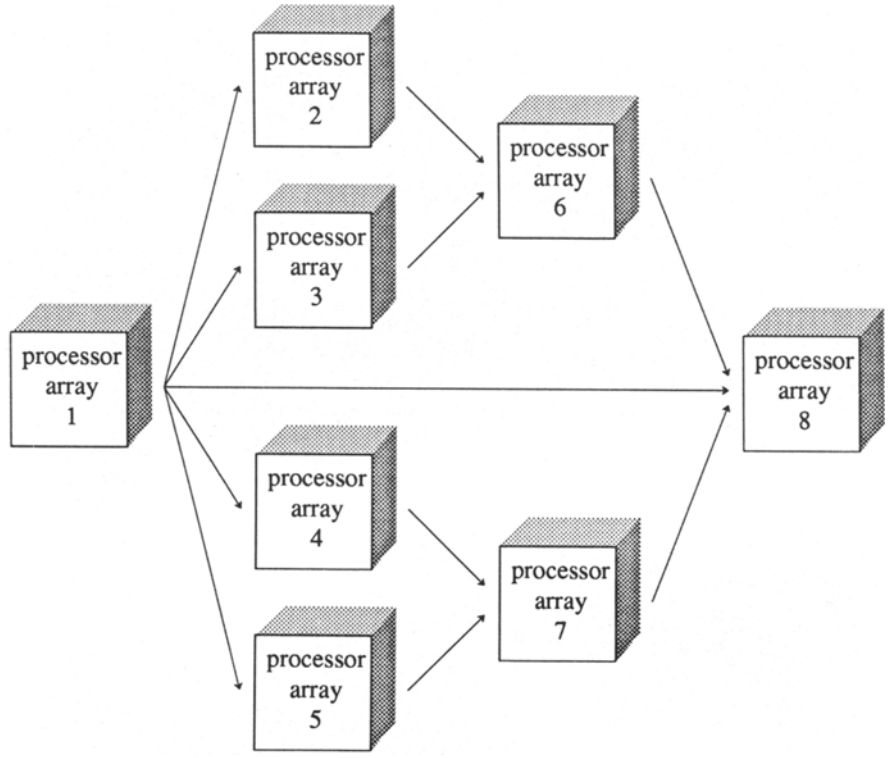

Figure 3. The EVE software can be used with models with any number of processor arrays arranged in any feed-forward manner. This can be a simple pipeline of layers as in Figure 1, but can also involve arbitrary arrangements of layers involving parallel channels.

```
Header format:                  1
Originator name:                Michael Landy
Sequence name:                  Gabor Model
Number of frames:               3
Original date:                  6/6/88
Image format:                   positions table
Number of processors:           4
Number of rows:                 0
Number of columns:              0
Processor format:               Reals
Size of outputs (bytes):        4

Sequence history:
read_posns "-D Wed Jul 27 11:36:49 1988" | \
add_struct -g Gabor-structures "-D Wed Jul 27 11:45:34 1988"

Sequence description:
This sequence is based on the processor positions needed for the blurred
hyperacuity task.

Scale:                          1.000
XO:                             0.000
YO:                             0.000

Processor structure:            Gabor
Size of processor structure:    28
Size of POSNS(x,y):             8
```

Figure 4. A sample EVE image header as output by the program eve_seeheader.

fore device-independent. However, special facilities exist to use EVE with the HIPS software, and to view EVE processor images on Sun workstations.

In EVE, all image sequences begin with an image header, which describes the basic parameters of the sequence. The image header gives EVE its flexibility. Each EVE program can operate on images in a variety of formats by checking the contents of the image header. An example image header is shown in Figure 4, as printed by the program eve_seehead. This sequence consists of three frames of data (three discrete time steps have been simulated), and four processors. Note that we use the term *processor* in this paper, rather than the term *pixel* or *picture element* from image processing, in order to emphasize the generality of processing available in EVE. For this sequence, the images (or processor arrays, if you prefer) are in *positions table* format. Directly after the image header, there is a processor positions table that gives the spatial position of each of the four processors. After the processor positions table, there follow the processor structures. For this sequence, the processor structures are in Gabor format. Thus, for each processor, there is an associated set of parameters for a receptive field with a Gabor profile. The parameters include orientation, spatial frequency, phase, and the width of the Gaussian window. Finally, the processor output values are floating point numbers. Thus, after the processor structures, there are 12 floating point numbers corresponding to the output values of the four processor outputs for each of the three time steps. The image header also includes documentary information, which is stored along with the image and remains with the image sequence as it is processed by further EVE programs. Most important is the *sequence history* (a notion borrowed from HIPS; see Landy et al., 1984a, 1984b). The sequence history automatically summarizes all of the EVE commands that have been executed to compute a given image sequence. Each EVE program automatically keeps the history up to date by adding an additional line corresponding to its own command line.

EVE images may be spatially sampled in a variety of ways. The previous example presented an image in *positions table* format. The other formats currently implemented are *square* and *hexagonal* sampling. For these two formats, the header indicates the numbers of rows and columns in the sampling array. For square sampling, the processors are arranged in rows and columns that are aligned and equidistant. For hexagonal sampling, a hexagonally packed rectangular array is achieved by offsetting odd-numbered rows ½ of an intercolumn distance to the right, and spacing rows more closely together than columns by a factor of $\sqrt{3}/2$ (see Emerson, 1986, for more discussion of simulation of hexagonal sampling arrays for visual modeling). Since the sample position information is stored with each image, successive EVE programs can handle all three sampling formats without user intervention, enhancing the flexibility of the system. Finally, each EVE image includes a coordinate transformation from the natural coordinate system of the image (e.g., row and column numbers, etc.) to retinal coordinates (degrees of visual angle), so that modelers can refer to processor and stimulus parameters in retinal terms.

EVE allows the user to store an arbitrary set of parameters in image sequences to describe each processor. These parameters are held in the *processor structures*, which are stored in the image sequence after the header and positions table. There is one processor structure for each processor. For example, if the model consists of a set of Gabor filters, each processor structure can describe the parameters of the Gabor (spatial frequency, orientation, phase, etc.). There are currently three formats for processor structures implemented in EVE: Gabor, DoG, and General. The Gabor and DoG formats are tailor-made for describing receptive fields with a Gabor or Difference of Gaussians profile, respectively. The General processor structure is just that: it allows the user to store an arbitrary set of up to 10 parameters with each processor to govern its performance. General processor structures can be used to control several kinds of model behavior: shape of linear receptive fields (by storing parameters of a functional form of the receptive field profile), wiring of linear or nonlinear processor computations (by specifying the processors that provide input to a given processor), and parameters of spatially pointwise transformations. EVE can easily work with arbitrary nonisotropic models. Each processor is associated with its own parameters in the processor structure. Thus, the processors in a layer can have different receptive field shapes, different wiring, different forms of nonlinearity, and so forth.

There are currently about 60 different programs in EVE. These include programs involved with image headers, binary format conversion, conversion to and from ASCII, operations on sequences, operations on single processors, frame arithmetic, operations using processor structures, and image statistics (see the Appendix for a list of currently available programs). Images in HIPS format can be converted to EVE format for use as input stimuli, and processor arrays in EVE format can be converted to HIPS format for display purposes (to view a "neural image"; see Robson, 1980). There is a program for viewing EVE images directly on the console of a Sun workstation under SunTools.

There are a variety of programs that compute what we have termed spatially pointwise operations. There are programs that apply a log, an exponential, a power function, a quadratic, and a threshold. There is a program that adds independent noise to the processor output values. There is a general digital temporal filter program (see Hamming, 1983, and Oppenheim & Schafer, 1975, for further discussion of digital filtering). Finally, there is eve_calcoutput, which takes a line of C program code from the user as the definition of a spatially pointwise transformation, and generates a new EVE program which applies that transformation. Eve_calcoutput is one of several EVE programs that can be used to easily extend EVE by generating new EVE programs given user-specified

program code (the others perform convolution, generate new processor arrays, or perform arbitrary processor computations).

There are several programs that can be used as the first EVE computation in a layer—the one that combines outputs of processors in a previous layer. All of these programs can use processor structures. For linear receptive field calculations, there are programs for Gabor and DoG processors. In addition, there is eve_genconv, which allows the user to specify the functional form of a receptive field profile as a line of C program code (e.g., the Laplacian of a Gaussian, the difference of three Gaussians, etc.). The program then generates a new EVE program, which calculates the result of the application of this receptive field form to the input to the layer. The other kind of processor computation in EVE that combines previous layer processor outputs involves a simple combination of a small number of processor outputs from a previous layer. EVE programs are provided that can perform such a combination using addition, subtraction (e.g., an opponency stage), multiplication (e.g., the cross-correlation within a motion detector such as that described by Reichardt, 1957, 1961), or division. A general processor computation program is provided, eve_calcproc, which takes a user-specified line of program code describing the functional form of the processor computation.

Typical EVE Usage

The use of EVE for simulation typically consists of three steps: (1) define the processors for each layer; (2) create the input stimulus; and (3) apply the sequence of calculations for each layer in turn, terminating with a computation of a statistic of the final output layer.

The first step involves creating a sampling array (giving each processor a spatial location), and then specifying the parameters of each processor (the processor structures). The processor locations and parameters may be specified by hand (or by programs of your own devising) by using EVE programs read_eve, read_posns, read_sqr, read_hex, and add_struct. Alternatively, one may specify the program code used to compute processor locations (with gen_posns), or simply write an EVE program specially tailored to the purpose (as we did for the Watson spatial model described below).

There is a variety of ways to generate an EVE input stimulus. The stimuli may be imported from other software (using read_eve, read_posns, read_sqr, or read_hex). Users of the HIPS image processing software (Landy et al., 1984a, 1984b) may convert their HIPS images to EVE format for use as an input (with hips_to_eve). Finally, there are three EVE programs that allow the user to program simple input images easily by specifying a single line of C program code, which computes the processor output values: gen_posns, gen_sqr, and gen_hex.

Having created the processor arrays and the input image, the various layers are applied in turn to the input, or to the outputs of previous layers. If the model consists of a single series of layer computations, then this may all be accomplished with a single UNIX command string using the UNIX *pipe* facility. This facility (which is denoted with a |) allows the output of each program to become the input of the succeeding one. Consider the example pictured in Figure 1. The main computation for this model can be carried out with the following command string (command parameters have been omitted for clarity):

```
eve_gabor layer1-processors < input-stimulus| \
    eve_power|eve_tempfilt| \
    eve_diffsens layer2-processors|eve_mink
```

The model consists of two layers. The first layer calculates a set of Gabor receptive fields (using eve_gabor), squares their outputs (using eve_power), and temporally filters the resulting processor outputs (using eve_tempfilt). The second layer subtracts pairs of layer 1 outputs (using eve_diffsens). The final linking hypothesis consists of a computation of the vector magnitude of the layer 2 processors (using eve_mink). Of course, the user need not apply all stages of the model at one time, and can cut off the computation after any intermediate stage. These intermediate results, which are EVE images, can be converted to readable ASCII (using eve_to_ascii), or viewed as an image on a display system (using eve_sunview or eve_to_hips).

EXAMPLE APPLICATIONS OF EVE

Here, we discuss the way in which the EVE software may be applied to a variety of visual models. We concentrate on the range of models for which EVE is useful, and on some of the details concerning how the EVE programs are combined to perform each simulation. Some results from these simulations are discussed in Landy et al. (in press). Note that this does not constitute an exhaustive list of the potential applications of EVE. Other possibilities include applications to binocular brightness summation (by using two input layers, one for each eye), more complicated motion processing such as the computation of speed, and so forth.

The Watson Spatial Model

Watson (1983) proposed a model for the detection and discrimination of spatial patterns that is perfectly suited to simulation using EVE. The first layer of the model consists of a large array of linear processors with a variety of receptive fields with Gabor profiles (they are called "sensors" by Watson). The processors vary in spatial frequency, phase, orientation, and spatial position. There are eight families of processors, which are distinguished by their preferred spatial frequency—the central processor in each family has a preferred spatial frequency of .25, .5, 1, 2, 4, 8, 16, and 32 cpd, respectively. The other processors in each family are arranged in a series of circular rings around the central one (which is placed at the center of the fovea). The spacing between processors is always sufficient to meet the Nyquist sampling criterion, and the processors scale in size and preferred spatial fre-

quency with eccentricity. At any sampling position in this array of rings for a given family, there are 10 processors, which include each of 5 preferred orientations at each of 2 phase angles (sine and cosine phase). The processor sensitivities are the principal free parameters, and they are adjusted so that the output of the model for sine wave gratings conforms to an empirically measured contrast-sensitivity function.

The Watson model consists of only two layers. The first layer is the large array of processors with Gabor receptive fields that we have just described. The only other computation in the model relates the output of this collection of thousands of linear processors in response to a visual stimulus to performance by a human in a detection task using the same stimulus. This computation, or "linking hypothesis," involves treating the collection of receptive field outputs as one large vector. The magnitude of that output vector is treated as an estimate of d' in the detection task. To model performance in a discrimination task, one computes the output vectors resulting from each of the two patterns to be discriminated. The magnitude of the vector difference between those two output vectors is the model's estimate of the discrimination d'.

It is easy to compute the results of this model using EVE. Given that the determination of processor positions and preferred spatial frequencies is given by Watson as an algorithm, we wrote a simple EVE program to generate the processor arrays, called *wgrid*. This program has arguments that specify the spatial frequency family (preferred spatial frequency at the fovea), the orientation, the phase, the sensitivity, the eccentricity scaling constant, and the radius of the largest ring of processors (in degrees). The output of wgrid is in *positions table* format, and it has Gabor processor structures, so that each processor now has an assigned position and receptive field parameters. The EVE command, gen_sqr, can be used to generate an input stimulus (e.g., a sine wave grating). Each processor grid is applied to an input pattern using eve_gabor. Note that eve_gabor determines from the input stimulus the positions of the input samples, and from the processor array it determines the processor sample positions, so the sampling properties of the input and new processor array are effectively independent.

The pair of commands wgrid and eve_gabor are repeated for each subfamily of processors in the model (8 frequencies, 2 phases, and 5 orientations, resulting in 80 subfamilies). This allows the user to compute the separate contribution of each subfamily to the total model response if so desired. Finally, a merged vector of processors is created from the separate subfamilies using eve_merge, and the vector magnitude is computed with a Minkowski metric with a power of 3.5, using the program eve_mink. Pelli (1981, 1985) suggests that effects of stimulus uncertainty may be modeled reasonably by using an exponent greater than 2.0.

Figure 5A shows a natural image that was converted to EVE format using hips_to_eve. We ran one subfamily of processors from the Watson model on this image, resulting in the set of processor outputs shown in Figure 5B (using eve_sunview). The individual processors are visible as dark or bright squares corresponding to a receptive field result that is positive or negative. The processors are located on a series of rings centered on the central one. Processors with very weak outputs are represented using a medium gray tone.

The Wilson Spatial Model

We have also simulated the line element model for pattern discrimination described by Wilson (1986). Like the Watson model, this model begins with a layer of processors with linear receptive fields and ends with a linking hypothesis, which is a computation of output vector magnitude. However, a different receptive field profile is used, and a nonlinearity is applied to the outputs of each linear receptive field prior to pooling across processors. The distance between two output vectors is again the model prediction for pattern discriminability. The model consists of six families of processors (indexed by preferred spatial frequency). For each family and any given task, Wilson describes the spatial positions and orientations of processors used to model performance in the task. The spatial positions may be read using read_posns, and the preferred orientations added as processor structure information using add_struct. The program gen_sqr can be used again to generate the stimulus. Next, the processor receptive fields are calculated using eve_genconv, which is a generalized convolution program. In the Wilson model, receptive fields are the difference of three Gaussians in one direction multiplied by a Gaussian in the orthogonal direction, which requires the use of eve_genconv instead of eve_dog. Next, eve_calcoutput is used to apply a nonlinearity (first accelerating, then saturating) to each processor response. This pair of computations (receptive field followed by nonlinearity) is repeated for each of Wilson's six spatial frequency channels. Finally, as with the Watson model, the processors from each channel are merged into one vector for a given input stimulus, the whole thing is repeated for a second stimulus, and the magnitude of the vector difference is a measure of discriminability.

Retinal Sampling

One of the novel aspects of the EVE software is its ability to handle different spatial sampling arrays easily. As such, it may be used to explore the consequences of spatial sampling in the visual system, both retinal and at later processing stages. For example, a hexagonal sampling grid using Gaussian sampling functions may be applied to an image by using read_hex to set up the grid and eve_gabor to compute the results of the Gaussian sampling functions. Figure 5C shows the result of hexagonally sampling the central portion of the image shown in Figure 5A, as viewed using eve_sunview. Of course, since later stages in a model also use EVE routines that are general with respect to the sampling of the input image, one can use the results of the commands above as the image input to

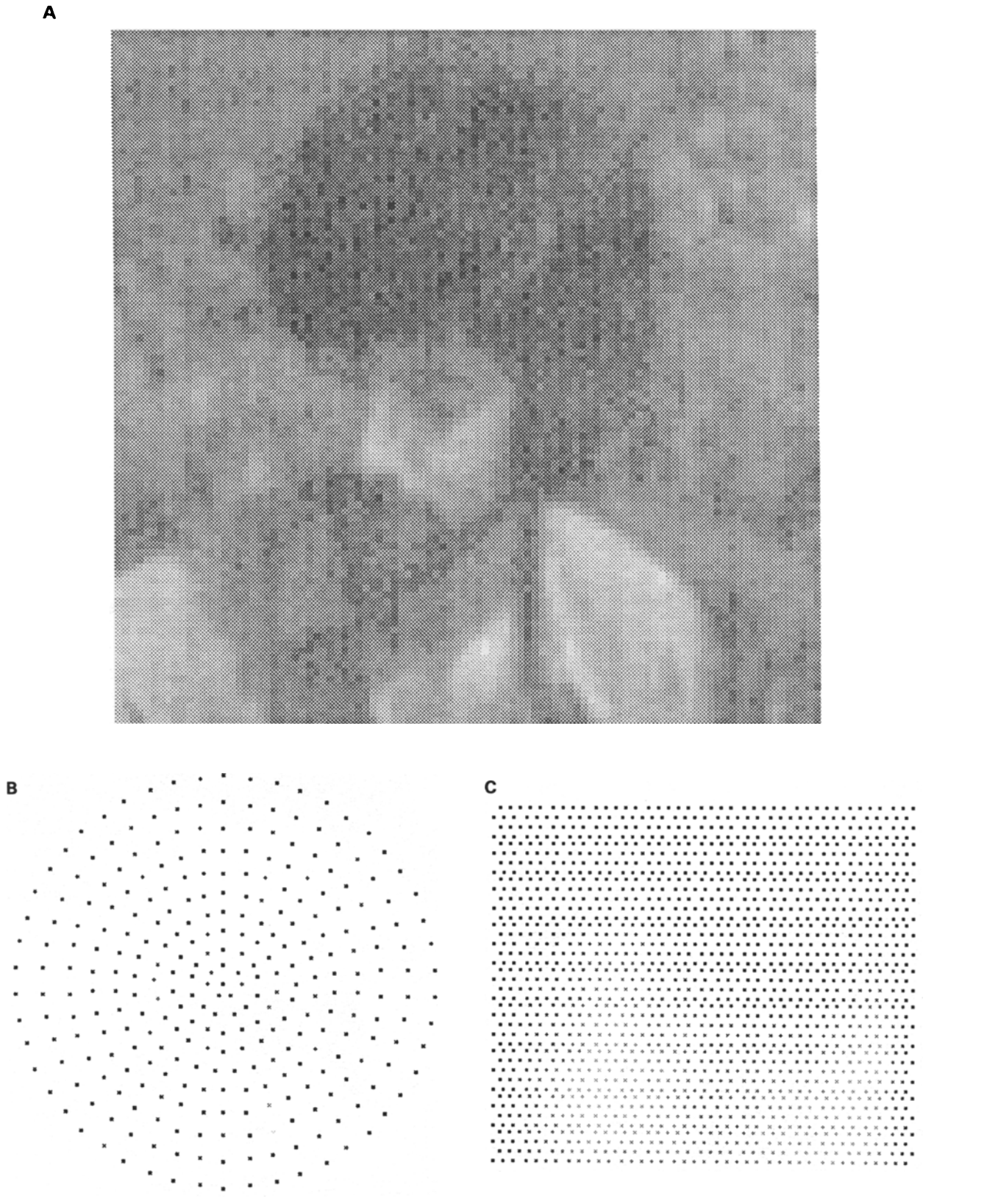

Figure 5. An example of using EVE to simulate the Watson (1983) model of spatial pattern detection and discrimination. (A) A natural image. (B) The outputs of the 4-cpd family of processors as applied to the image in panel A. The family consists of a series of concentric rings centered on the fovea. The central processor is tuned to 4 cpd. More eccentric processors are tuned to successively lower frequencies. In this image, we indicate the output of units with Gabor receptive field profiles in sine phase and with vertical orientation. Positive outputs result in a dark spot, and negative outputs in a light spot; weakly activated processors are represented using a midgray tone. (C) A hexagonal grid of linear processors with Gaussian receptive fields used to sample the image in panel A.

a model. For example, we have used the preceding technique to create simulated foveal retinal images, and used them as input to the Watson (1983) model simulation described above. This was easy to do, because the EVE routines used in the Watson simulation are general with respect to the input image-sampling properties.

Motion Detection

EVE may also be used with models of motion detection. For example, consider the motion energy model of Adelson and Bergen (1985). This model constructs linear filters that are tuned to a particular direction of motion (and hence are not space-time separable) by suitable sums and differences of space-time separable filters. The total power in two such leftward-tuned filters (which are approximately in space-time quadrature) is added, as is the power in two such rightward-tuned filters. The opponent response "right minus left" is the output of the model.

Again, models such as this are easy to compute using EVE. In our simulation, we use gen_sqr to generate the input image sequence (e.g., a moving sine wave grating). Next, read_sqr is used to define the spatial sampling of the motion processors. At each processor location, a sine-phase and cosine-phase Gabor receptive field is cross-correlated with the input (using eve_gabor). Next, the output of each of the sine and cosine phase Gabor operators is processed by each of two temporal filters using eve_tempfilt. Both temporal filters are bandpass, but one has a more delayed impulse response than the other (see Adelson and Bergen, 1985, for details of the filters). The resulting outputs have thus been filtered by the composition of a spatial and a temporal filter. In other words, in each case, a space_time separable filter has been applied. Linear filters are constructed from these by sums and differences (using eve_add and eve_diff) that are no longer space-time separable functions of the input sequence, and which have receptive fields that are oriented in space-time (i.e., are tuned for motion direction). The power in these processors (the square) is then computed using eve_power, followed by a sum of the two phases in each direction. Finally, eve_diff computes the opponent motion signal. As a result, it is simple to explore this motion model on a variety of stimuli using the EVE software.

The elaborated Reichardt detector (or ERD; see van Santen & Sperling, 1985) is an alternative model for early motion detection. Particular realizations have been shown to be equivalent at the model's final output (Adelson & Bergen, 1985; van Santen & Sperling, 1985). We have demonstrated this with the EVE software, by simulating a simple ERD. We use the same space-time separable filter outputs as those in the simulation of the Adelson and Bergen model. Multiplying outputs of pairs of these processors (using eve_mul) results in processors that are tuned for leftward or rightward motion, but are nonlinear. As before, rightward and leftward subunits are set in opponency. This final output is proportional to the final output of the Adelson and Bergen simulation (as can be shown by a trivial mathematical argument).

CONCLUSIONS

The EVE software is a general package for the simulation of models of early visual processing. We have outlined the software, and indicated the way in which it is used to compute a variety of models. It is a flexible, modular system for simulation. It is easy to use, and easy to extend. It is fully documented (with manual pages, and a complete reference manual [Landy, 1988], as well as example scripts that apply EVE to all of the model simulations discussed in this paper). We have described applications to spatial pattern detection and discrimination and to motion detection. EVE is also suitable for models in other areas of vision, including binocular brightness computation (using two input images, one for each eye), further motion analysis (such as the computation of speed), and so forth. As it stands, EVE is restricted to the computation of models consisting entirely of feed-forward connections between processors. We are interested in using EVE to model the transformation of receptive fields in the visual cortex, and we will be adding to EVE the capability of computing models that include feedback within a layer and to earlier layers using a similar, iterative computational scheme.

DISTRIBUTION

If one is interested in using the EVE software, it is available. There are two ways to obtain EVE. One can send a tape to Michael Landy, New York University, Department of Psychology, 6 Washington Place #961, New York, NY 10003. It can be either a half-inch 9-track tape or a quarter-inch Sun cartridge. Alternatively, if one has ftp access to Arpanet hosts, one may obtain it via anonymous ftp to my Sun file server 'vml.psych.nyu.edu' (internet number 128.122.132.4; the file is "eve.tar.Z" in directory "pub"). The distribution includes source, documentation, and all of the examples discussed in this paper. The source directory takes about 1.3 MB. On my Sun, the binaries require about 3.2 MB. EVE should run on any UNIX machine with no, or only minor, changes. We have run it on Suns and DEC Vax computers so far. It should be a simple matter to port the software to MS-DOS (e.g., using Turbo C). Anyone who is using EVE should notify us, so that we may provide information about new versions and bug fixes. Also, since the EVE package is intended to be shared among laboratories, we would appreciate copies of useful extensions made to EVE, so that we may include them in subsequent versions.

REFERENCES

ADELSON, A. H., & BERGEN, J. R. (1985). Spatiotemporal energy models for the perception of motion. *Journal of the Optical Society of America A*, **2**, 284-299.

BARLOW, H. B., & LEVICK, W. R. (1965). The mechanism of directionally selective units in rabbit's retina. *Journal of Physiology*, **178**, 477-504.

EMERSON, P. L. (1986). A honeycomb data array for simulating visual processes using the C programming language. *Behavior Research Methods, Instruments, & Computers*, **18**, 312-320.

HAMMING, R. W. (1983). *Digital filters*. Englewood Cliffs, NJ: Prentice-Hall.

KERNIGHAN, B. W., & MASHEY, J. R. (1981, April). The UNIX programming environment. *Computer*, pp. 12-22.

KERNIGHAN, B. W., & RITCHIE, D. M. (1978). *The C programming language*. Englewood Cliffs, NJ: Prentice-Hall.

LANDY, M. S. (1988). *The EVE Early Vision Emulation software: Reference manual* (Mathematical Studies in Perception and Cognition 88-11). New York University, Department of Psychology.

LANDY, M. S., COHEN, Y., & SPERLING, G. (1984a). HIPS: A Unix-based image processing system. *Computer Vision, Graphics, & Image Processing*, **25**, 331-347.

LANDY, M. S., COHEN, Y., & SPERLING, G. (1984b). HIPS: Image processing under UNIX. Software and applications. *Behavior Research Methods, Instruments, & Computers*, **16**, 199-216.

LANDY, M. S., MANOVICH, L. Z., & STETTEN, G. D. (in press). Applications of the EVE software for visual modeling. *Vision Research*.

MANOVICH, L. Z., & LANDY, M. S. (1988). EVE: Software for psychophysical modeling. *Investigative Ophthalmology & Visual Science*, **29**(Suppl.), 447.

MARR, D., & ULLMAN, S. (1981). Directional selectivity and its use in early visual processing. *Proceedings of the Royal Society of London B*, **211**, 151-180.

OPPENHEIM, A. V., & SCHAFER, R. W. (1975). *Digital signal processing*. Englewood Cliffs, NJ: Prentice-Hall.

PELLI, D. G. (1981). *Effects of visual noise*. Unpublished doctoral dissertation, University of Cambridge.

PELLI, D. G. (1985). Uncertainty explains many aspects of visual contrast detection and discrimination. *Journal of the Optical Society of America A*, **2**, 1508-1532.

REICHARDT, W. (1957). Autokorrelationsauswertung als Funktionsprinzip des Zentralnervensystems. *Zeitschrift für Naturforschung*, **12B**, 447-457.

REICHARDT, W. (1961). Autocorrelation, a principle for the evaluation of sensory information by the central nervous system. In W. A. Rosenblith (Ed.), *Sensory communication* (pp. 303-317). Cambridge, MA: MIT Press.

RITCHIE, D. M., & THOMPSON, K. (1978). The UNIX time-sharing system. *Bell System Technical Journal*, **57**, 1905-1929.

ROBSON, J. G. (1980). Neural images: The physiological basis of spatial vision. In C. Harris (Ed.), *Visual coding and adaptability* (pp. 177-214). Hillsdale, NJ: Erlbaum.

VAN SANTEN, J. P. H., & SPERLING, G. (1985). Elaborated Reichardt detectors. *Journal of the Optical Society of America A*, **2**, 300-321.

WATSON, A. B. (1983). Detection and recognition of simple spatial forms. In O. J. Braddick & A. C. Sleigh (Eds.), *Physical and biological processing of images* (pp. 100-114). New York: Springer-Verlag.

WATSON, A. B., & AHUMADA, A. J., JR. (1985). Model of human visual-motion sensing. *Journal of the Optical Society of America A*, **2**, 322-342.

WATT, R. J., & MORGAN, M. J. (1983). The recognition and representation of edge blur: Evidence for spatial primitives in human vision. *Vision Research*, **23**, 1465-1477.

WILSON, H. R. (1986). Responses of spatial mechanisms can explain hyperacuity. *Vision Research*, **26**, 453-469.

WILSON, H. R., & GELB, D. J. (1984). Modified line-element theory for spatial-frequency and width discrimination. *Journal of the Optical Society of America A*, **1**, 124-131.

APPENDIX
Currently Available Eve Programs

HEADERS

eve_adddesc – Add descriptive information to a header
eve_grabhead – Throw away an EVE sequence, preserving the header
eve_seehead – Print the header of an EVE sequence
eve_striphead – Remove the header from a sequence

BINARY FORMAT CONVERSIONS

eve_btof – Convert byte format to floating point
eve_btoi – Convert byte format to integer
eve_ftob – Convert floating point to byte
eve_ftoc – Convert floating point to complex
eve_ftoi – Convert floating point to integer
eve_itob – Convert integer to byte format
eve_itof – Convert integer to floating point
eve_sunview – Display byte EVE sequences on a Sun console
eve_to_hips – Convert an EVE sequence to HIPS format
hips_to_eve – Convert a HIPS sequence to EVE format

CONVERSIONS TO AND FROM ASCII

ascii_to_eve – Convert an ASCII file to EVE format
eve_to_ascii – Convert an EVE sequence to ASCII format
posns_to_ascii – Convert processor positions to ASCII format
read_eve – Create an arbitrary EVE sequence from ASCII data
read_hex – Create a hexagonal image format sequence from ASCII data
read_posns – Create a positions table image format sequence from ASCII data
read_sqr – Create a square grid image format sequence from ASCII data
struct_to_ascii – Convert processor structures to ASCII format
value_to_ascii – Convert processor output values to ASCII format

APPENDIX (Continued)

OPERATIONS ON SEQUENCES

eve_cat – Concatenate several EVE sequences
eve_merge – Merge the processors from several sequences
eve_subseq – Extract a subsequence of frames

OPERATIONS ON PIXELS

eve_absval – Take the absolute value of each processor output value
eve_calcoutput – Generalized point transformation
eve_exp – Take the exponential of the sequence
eve_log – Take the logarithm of the sequence
eve_mag – Convert complex format to floating point magnitude
eve_neg – Take the negative of each processor output value
eve_noise – Add noise to a sequence
eve_phase – Convert complex format to floating point phase
eve_power – Raise processor output values to a power
eve_scale – Scale processor output values by a linear or quadratic function
eve_tempfilt – Temporal filtering
eve_thresh – Apply a fixed or variable threshold to an image

FRAME ARITHMETIC

eve_absdiff – Absolute value of the difference of two sequences
eve_add – Compute the sum of two sequences
eve_combine – Combine phase and magnitude images to form a complex image
eve_diff – Compute the difference of two sequences
eve_div – Compute the quotient of two sequences
eve_mul – Compute the product of two sequences

OPERATIONS INVOLVING PIXEL STRUCTURES

add_struct – Add processor structures to the input sequence
drop_struct – Drop processor structures from the input sequence
eve_activate – Set/clear processor structure 'active' flags
eve_addproc – Compute the value of 'summing processors'
eve_calcproc – Generalized processor computation
eve_diffproc – Compute the value of 'difference processors'
eve_divproc – Compute the value of 'division processors'
eve_dog – Perform Difference-of-Gaussians convolution
eve_gabor – Perform Gabor convolution
eve_genconv – Perform generalized convolution
eve_mulproc – Compute the value of 'product processors'
eve_trim – Trim away inactive processors

STATISTICS

eve_framevar – Compute basic descriptive statistics
eve_histo – Compute a histogram of processor output values
eve_mink – Compute the Minkowski metric magnitude of processor outputs

MISCELLANEOUS

eve_procedit – Edit an EVE sequence interactively
gen_hex – Generate a hexagonal sequence
gen_posns – Generate a sequence with a processor positions table
gen_sqr – Generate a square grid sequence

(Manuscript received May 5, 1989;
revision accepted for publication August 29, 1989.)