COMPUTER SIMULATION

A honeycomb data array for simulating visual processes using the C programming language

PHILLIP L. EMERSON

Cleveland State University, Cleveland, Ohio

A method of simulating visual processes is formulated as a C language program. The two-dimensional image array has a honeycomb structure, in which each cell has six immediate neighbors. A general iterative simulation method that represents changes in time is used. The method is illustrated with tests of Horn's (1974) model of the retinex lightness process of Land and McCann (1971) and of a modified form of that model.

In the primate fovea, the receptor cells are arrayed approximately in honeycomb fashion, in which each cell usually has six immediate neighbors. The regularity is mainly local, with irregular changes of the orientations of the hexagons over distance. The regularity, however, is absent outside of the fovea (Williams, 1985).

This regularity has been of much interest in connection with a two-dimensional form of the Nyquist sampling theorem. The one-dimensional form of this theorem predicts well-known aliasing phenomena in spectral estimates from equally spaced time-series data. Aliasing is undesirable from an information-theoretic point of view, because estimates of low-frequency spectral components are contaminated by contributions from frequencies above the Nyquist limit, which is half the reciprocal of the sample spacing. Blackman and Tukey (1958) suggested that there is not much hope of eliminating aliasing effects by some special scheme of unequal spacing of the sample points. On the other hand, Yellot (1982) showed that a sufficient degree of randomness in the spacing of elements in a two-dimensional sampling array (the fovea) could eliminate aliasing. Yellot also proposed that the anatomical irregularity of the foveal receptor spacing is sufficient to eliminate the aliasing.

Two-dimensional aliasing is illustrated by Moiré patterns, in which a scene is viewed through a regular array of apertures, such as an ordinary wire screen door. If the viewed scene contains regularly spaced elements, then the transmitted pattern often contains interesting lower frequency light-and-dark stripes that are due to the combination of screen and scene, but are not a feature of either alone. A question addressed by several researchers (e.g., Miller & Bernard, 1983; Williams, 1985; Yellot, 1982)

concerns possible aliasing due to the finite regular spacing of foveal cones: Is there aliasing in the image available for postreceptor processing? The answer may be the same as the prescriptive design for avoiding aliasing in time-series experiments (Blackman & Tukey, 1958). Aliasing can be eliminated by a prior stage of low-pass filtering. Williams (1985) reasoned that the known low-pass optical characteristics of the ocular media, under normal viewing conditions, are sufficient to eliminate the aliasing that would arise from the known characteristics of foveal cone spacing.

These sampling-theorem issues might motivate the formulation of a honeycomb data array for simulation purposes. However, they are not addressed by the example simulations reported here. The processes at issue in these simulations were originally thought to be retinal, but now are thought to be cortical (Land, 1983), and the hexagonal geometry is actually a convenient heuristic idealization rather than an established anatomical fact. Horn introduced this honeycomb heuristic in his reformulation of the retinex lightness model, for the convenience of formulating lateral interactions in a simple way. The hexagonal structure makes each immediate neighbor of a cell equally distant, thus avoiding the question of what to do about diagonal neighbors in a rectangular array. If that is all the difference it makes, perhaps the choice is rather arbitrary. On the other hand, findings on the performance of Horn's (1974) model might be mistrusted unless they were based on the same hexagonal structure that Horn used. It was in the interest of testing the model that the present simulation method was developed. Horn's model seemed to fail when implemented with one- and two-dimensional rectangular arrays, and when the hexagonal structure was simulated; however, the present method of simulation is more general than Horn's (1974) model and can be used to test other models.

Figure 1 shows the arrangement of cells in the array used in the simulations. This is a nomographic represen-

The author is indebted to anonymous reviewers and the editor for suggestions on the preparation of this revision. The author's address is Department of Psychology, Cleveland State University, Cleveland, OH 44115.

Figure 1. Nomographic display of an image registered in a honeycomb array of 1,163 cells. The array itself is bounded by a circle, and the image has circular and radial symmetry about the center of the array. Relative intensities are represented on a 10-point scale using a single digit for each cell, from 0 to 9. The figure represented is of the Cornsweet (1970) type that is produced by suitable white-black patterns on a spinning disk. An inner circular region is separated from an outer annular one by a significant discontinuity of intensity.

tation of intensities on a scale ranging from 0 for the minimum to 9 for the maximum. The radius of the circular array is 18 cell widths, and the array contains 1,163 cells. This size approximates that of Horn's (1974) array. A simple programming approach to a honeycomb array is to treat a rectangular array as if alternate rows were offset by half a step horizontally. This gives each cell 6 immediate neighbors, but a scale factor of $\sqrt{3}/2$ must be applied to equalize row and column spacing of adjacent elements. That approach is combined with the C language option of defining a special data structure, which permits a single rectangular matrix of pointers to be used for access to several quantities associated with each cell.

The source code is presented in three sections, although these are compiled together as a unit. The first two sections are relatively fixed and need not be changed under minor variations of the model under test. In the third section, the theoretical model under test is defined in detail, and the general iterative calculations are programmed.

THE HONEYCOMB ARRAY

The program section shown in Figure 2 contains external declarations of variables used by various routines. The *struct cell* declaration specifies three quantities attributed to each cell, *input*, *state*, and *change*. A rectangular table of cell pointers is declared with dimensions large enough for a honeycomb inside a circle of radius 18.

The three variables, *radius*, *sizeh*, and *sizev*, are used to specify the size of the array and its horizontal and vertical limits.

The external declarations are followed by two functions, *x(i,j)* and *y(i,j)*, that evaluate the rectangular coordinates of a given cell. Alternate rows in the rectangular table are treated as shifted by half a step, and the factor .866025 adjusts the scale for equating distances of neighbors in adjacent rows and columns. The origin of the x,y plane is the center cell of the array.

The construction of the array is performed by the *setup* routine. The array has a circular boundary, with radius *radius*. For any x,y pair that falls inside the circle, *setup* allocates enough memory for one cell and stores a pointer to it in *table[i][j]*. For x,y pairs outside the boundary, a NULL is stored in *table[i][j]*, which indicates to the processing routines that no processing is necessary.

Following *setup* is the routine *nabor(i,j,k)*, which provides access to the six neighbors ($k = 0, 1, 2, 3, 4, 5$) of cell i,j. The numbering of neighbors is in counterclockwise order, starting at the right-hand neighbor in the same row. The use of *nabor* is illustrated in the functions *ginput* and *gstate*, which return the values of *input* and *state*, respectively, for the *k*th neighbor of cell i,j. Boundary conditions are implemented by the return of floating zero for any neighbor outside the array boundary.

INPUT AND OUTPUT

Figure 3 shows the second section of the program, consisting of functions for getting data into and out of the array. It is the two-dimensional nature of the data, as much as the honeycomb pattern, that poses some minor problems. Actual input is avoided in the version of *get-data*, defined in Figure 3. The input image is computed by a simple formula that generates a Cornsweet (1970) figure consisting of a disk and an annular surround, with the intensity in each region following a curved function of radial position. Segments of a fourth-degree parabola are used.

The first type of output, provided by the function *putout*, is a one-dimensional list of the intensities along the horizontal diameter of the circular array. This is almost sufficient in cases in which certain kinds of symmetry are present in the array. For other cases, the routine *plotem* provides a nomographic two-dimensional display, as explained in the caption of Figure 1. Both *putout* and *plotem* can be used to access either the *input* or the *state* variables in the array. The argument *n* should be set to 0 for *input*, or 1 for *state*.

RETINEX LIGHTNESS THEORY

The third section of the program depends much more on the particular details of the theoretical model to be simulated. Horn's (1974) model of the retinex lightness process is simulated because it is simple and it is formulated with a hexagonal array of elements. It is derived

```

#define void int
#include <math.h>
#include <stdio.h>
struct cell
{float input,state,change; } ;
struct cell *table[49][43] ;
float radius; char *alloc();
int sizev,sizeh,pflag ;

float x(i,j) int i,j ;
{float r ;
 r = (float) (j-sizeh/2) ;
 if( i % 2 ) r += .5 ;
 return(r) ; }

float y(i,j) int i,j ;
{return( .866025*( (float) (sizev/2 - i) ) ) ; }

float sq(x) float x; {return( x*x ) ; }

void setup()
{int i,j ;
 for(i=0; i < sizev ; i++)
 for(j=0; j < sizeh ; j++)
 { if( sq( x(i,j) ) + sq( y(i,j) ) >= sq(radius) )
  table[i][j] = NULL ;
  else
  if( !( table[i][j] = (struct cell *)
    alloc( sizeof(struct cell) ) )
  ) {puts("memory overflow\n") ; exit(1) ; }
 }
}

struct cell *nabor(i,j,k) int i,j,k;
{static int di[]={0,-1,-1,0,1,1} ;
 static int djev[]={1,0,-1,-1,-1,0} ;
 static int djod[]={1,1,0,-1,0,1} ;
 int iprime,jprime;
 iprime = i + di[k] ;
 if( i % 2 ) jprime = j + djod[k] ;
 else jprime = j + djev[k] ;
 return( table[iprime][jprime] ) ;
}

float ginput(i,j,k) int i,j,k ;
{if( !nabor(i,j,k) ) return(0.) ;
 return( nabor(i,j,k)->input ) ;
}

float gstate(i,j,k) int i,j,k ;
{if( !nabor(i,j,k) ) return(0.) ;
 return( nabor(i,j,k)->state ) ;
}

```

Figure 2. The first of three program segments. This segment declares some variables used generally in all parts of the program and defines functions that are associated with representing the honeycomb structure in which each cell has six immediate neighbors. The *#include* and *#define* compiler directives at the top are adequate with both Software-Toolworks C, and Ecosoft Eco-C88, running under MS-DOS. Other implementations may require different sets of directives.

from surprising phenomena reported by Land and McCann (1971) concerning the perception of the lightnesses and colors of regions in certain two-dimensional scenes. These scenes were patchworks of irregularly overlapping rectangles of different colors, shades, and sizes. These rectangles were illuminated in various ways, producing significant gradients of illumination across the complete scenes and also across each of the uniformly pigmented patches. Land and McCann (1971) discovered a remarkable degree of invariance of the apparent light-

nesses of equally pigmented regions in different parts of the scenes, under large luminance differences induced by the nonuniform illumination. Moreover, the interiors of the uniformly pigmented patches appeared uniform even when the illumination gradients were fairly steep.

Land and McCann (1971) concluded that there can be only one explanation of these phenomena. The visual system must determine the lightnesses and colors of the interiors of the regions solely from the luminance ratios at the boundaries. This principle became a central tenet of

```

void getdata()
{struct cell *p; int i,j,k; float r;
 for(i=0; i < sizev; i++)
  for(j=0; j < sizeh; j++)
   {if(p=table[i][j])
    {r=sqrt( sq(x(i,j)) + sq(y(i,j)) ) ;
     if( r < 9. )
      p->input = 10.*sq(sq(r/10.)) -1. ;
     else
      p->input = -10.*sq(sq( (radius-r)/10.)) ;
    }
  }
}

void putout(n) int n;
{int i,j; float v; struct cell *p ;
 puts("\f\n      x      y      intensity\n") ;
 i=sizev/2 ;
 for(j=0; j <= sizeh ;j++)
  if(p=table[i][j])
   {if(n) v=p->state ; else v=p->input ;
    printf("%10.4g %10.4g %10.4g\n",x(i,j),y(i,j),v) ;
   }
}

void plotem(n) int n ;
{float max,min,v; struct cell *p; int i,j;
 puts("\f\n") ;
 for(i=0,max= -1000.,min=1000.; i < sizev ; i++)
  for(j=0; j < sizeh ; j++)
   if( p=table[i][j] )
    {if (n) v=p->state; else v=p->input ;
     if (v > max) max = v ;
     if (v < min) min = v ;
    }
  puts("\f\n") ;
  for(i=0; i < sizev ; i++)
   {putchar('\n') ;
    if( i % 2 ) putchar(' ') ;
    for(j=0; j < sizeh ; j++)
     {printon(i,j,n,max,min); putchar(' ') ; }
   }
}

void printon(i,j,n,max,min) int i,j,n; float max,min;
{struct cell *p; float v; int byte;
 if( p=table[i][j] )
  {if( !n ) v=p->input; else v=p->state ;
   byte = 10.*(v-min)/((max-min)*1.01) ;
   putchar( byte + '0' ) ;
  }
 else putchar(' ') ;
}

```

Figure 3. The second of three program segments. This segment defines functions that are used for getting data into and out of the honeycomb array in useful ways.

the retinex theory of color vision (Land, 1983). Land and McCann (1971) went on to simulate this hypothetical process to show that it accounts well for the phenomena. Although the general principle seems clear enough and has been amply demonstrated, questions remain about how the visual system can yield perceptions of uniform surfaces, using only information near their boundaries.

A main mechanism in most of Land and McCann's (1971) simulations is a gradient threshold, such that luminance ratios at nearby points are registered in the visual system as unity unless they deviate significantly from unity. Another main mechanism is that a prior logarithmic transformation, early in the transduction process, con-

verts the luminance ratios to differences. Thereby, the threshold operation is reduced to a simple matter of setting small differences to zero, rather than ratios to unity.

HORN'S VERSION

The formulations that Land and McCann (1971) used in their simulations seemed somewhat inelegant and physiologically implausible to Horn (1974; see also Marr, 1974, 1982), so he introduced a different model of the process, with two main processing stages and a thresholding operation. The input is assumed to be the logarithm of the retinal illuminance on each cell in the honeycomb

array. The first stage performs a two-dimensional second-differencing operation, which is to say that all first-stage cells are pure antagonistic center-surround cells. In terms of the variables used in the present program, the first-stage computations are of the pattern,

$$state = input - \sum input_k/6, \quad (1)$$

where the summation is over the six neighbors of the cell in question. This computation is carried out for every cell in the array, considered as the focal cell. The threshold-

ing operation is then applied, before the second-stage computations.

Horn's (1974) second stage is merely the mathematical inverse of the first-stage operation, but the result is not trivially the same as the original input because of the intervening thresholding operation. The first-stage operation is simple (Equation 1) because the *state* variables are computed from the *input* variables. However, the opposite problem, where the *states* are given and the *inputs* are to be found, presents a problem of solving a full non-

```
void stage1()
{int i,j,k; struct cell *p ;
 for(i=0; i < sizev ; i++ )
  for(j=0; j < sizeh ; j++ )
   {if(p=table[i][j])
    {for(k=0,p->state=0.; k < 6 ; k++)
     p->state += ginput(i,j,k) ;
    p->state = p->input - p->state/6. ;
   }
  }
}

int stage2(h,nmax,tol,thresh) float h,tol,thresh; int nmax ;
{float tem,dostep(); int n,i,j; struct cell *p ;
 for(i=0; i < sizev ; i++)
  for(j=0; j < sizeh ; j++)
   {if(p=table[i][j])
    {tem = p->state ;
     if( fabs(tem) < thresh ) tem = 0.;
     p->state = p->input ;
     p->input = tem ;
    }
   }
 for(n=0; n <= nmax; n++)
  if( dostep(h) < tol ) return(n) ;
 return(n) ;
}

float dostep(h) float h ;
{int i,j,k; struct cell *p; float maxch;
 for(i=0, maxch=0.; i < sizev ; i++)
  for(j=0; j < sizeh ; j++ )
   {if( p=table[i][j] )
    {for(k=0,p->change=0.; k < 6 ; k++ )
     p->change += gstate(i,j,k) ;
     p->change = h*(p->input - p->state + p->change/6.) ;
     if(fabs(p->change) > maxch) maxch = fabs(p->change) ;
    }
   }
 for(i=0; i < sizev ; i++)
  for(j=0; j < sizeh ; j++)
   if(p=table[i][j]) p->state += p->change ;
 if( pflag ) putout(1) ;
 return(maxch) ;
}

main()
{float thresh,h,tol; int nmax,n ;
 radius=18.; thresh=.7; h=1.; nmax=1000; tol=.00001;
 pflag=1;
 sizev = (2.*radius+3.)/.866025 ;
 sizeh = 2.*radius+3.;
 setup(); getdata() ;
 puts("input data:\n") ; putout(0) ; plotem(0) ;
 stage1() ; putout(1) ; plotem(1) ;
 n = stage2(h,nmax,tol,thresh) ;
 puts("results:\n") ; putout(1) ; plotem(1) ; exit(0) ;
}
```

Figure 4. The third program segment. This segment contains the routines that perform the computations in the iterative simulation method for testing Horn's model, and a main program that calls routines in all three segments to conduct one complete simulation run.

trivial system of 1,163 equations in 1,163 unknowns. There is a simple iterative method that is similar to feedback methods used by the nervous system itself (Ratliff, 1965). Each iteration is based on a difference-equation computation of the form

$$\text{change} = h(\text{state} - \text{input} + \Sigma \text{input}_k/6),$$

followed by an update step,

$$\text{input} = \text{input} + \text{change}.$$

The quantity h is a discrete-time scaling constant whose value must be taken small enough to simulate the continuity of real time¹. This method is formulated in the *stage2* and *dostep* routines of Figure 4.

First, *stage2* applies the threshold to the output of *stage1* (the *state* variables at the end of *stage1*). In the same process, it interchanges the *input* and *state* variables to avoid replicating the memory requirements. The original *input* variables to *stage1* are no longer strictly needed, but they are used as initial estimates of the final results of *stage2*.²

The *stage2* routine iterates until the maximum of the *change* magnitudes falls below a given error tolerance, *tol*, or until a given maximal number of iterations, *nmax*, has been reached. The main computations of each iteration are performed by the routine *dostep*, which returns

Figure 5. Results of the first test of Horn's model, where the input figure was of the Cornsweet type represented in diametrical cross section by the top trace. The second and third traces are the cross-sectional representations of the results after the first and second stages, respectively. Distance and intensity scale units are arbitrary, and intensities for the different traces are offset vertically. Vertical lines indicate points of discontinuity in the intensity patterns.

Figure 6. Results of the second test of Horn's model, with the slanted flat-top figure whose cross section is shown in the top trace. The second trace shows the output from the first stage, and the third shows the output from the second stage.

the maximum of the *change* values that it has computed. In *dostep*, the complete array of values of *change* is calculated before they are added to the corresponding *states*. This makes the results of an iteration independent of the order in which the serial computations are performed, for simulation of the parallel processing that presumably occurs in the nervous system.

A minimal version of the main program is shown in Figure 4. A recompilation must follow any changes of parameter values, but compilation time is a small component of a normal simulation. After the parameter values are assigned, *main* computes *sizev* and *sizeh* from the given value of *radius*. It calls *setup* to construct the honeycomb array of cells and *getdata* to fill in the *input* values. It then proceeds through the two stages of the simulation, calling the output routines at appropriate times. The parameter *pflag* can be set to 1 to select the option of having output after each iteration, or to 0 otherwise.

RESULTS

The input figure defined by *getdata* provides one kind of test for retinex lightness models. Its intensity cross section is shown in the top of Figure 5. Such figures have been of interest to vision researchers for some time (Cornsweet, 1966, 1970; Craik, 1966; O'Brien, 1958) because they evoke perceptual effects that have seemed paradoxical. Although the central circular region and the outer an-

nular region are graded considerably in intensity, the visual appearance is that both are uniform and that the central region is lighter than the outer annulus. The general principle of retinex lightness theory predicts exactly that effect, because of the separating discontinuous boundary between the two regions.

The cross-sectional representations of the outputs of *stage1* and *stage2* are shown in the last two traces in Figure 5. The *stage2* output was obtained after 1,000 iterations, with the parameter values shown in Figure 4. Finding appropriate values of *thresh* requires a bit of trial and error, but $h = 1$ was found adequate in all cases. This simulation result is fairly close in representing the visual phenomenon. The original intensity gradients have been flattened considerably, and the intensity in the center circular region is generally greater than that in the annulus. However, there are many other simple tests for models of the retinex lightness process, using different stimulus figures. Just one is considered here. Note the two statements that sandwich the *else* directive in *getdata*. These two statements generated the two different curved sur-

faces for the inner circle and outer annulus. Let them now be replaced by

$$p \rightarrow \text{input} = 5. + x(i,j)/9.$$

and

$$p \rightarrow \text{input} = 0. ,$$

respectively. This modification generates an input figure that is even simpler than the original. The intensity is uniform (zero) in the outer annulus but is graded linearly along the horizontal dimension inside the circle. There still is a boundary of discontinuity separating the two regions, of size 4 at the left end and size 6 at the right end of the horizontal diameter of the inner circular region. The results of the simulation with this figure are shown in Figure 6. From top to bottom, the cross-sectional representations are shown for the input figure, the output of *stage1*, and the output of *stage2* after 1,000 iterations. The parameter values were the same as those shown in Figure 4. The final result, the output of *stage2*, is not at all similar to the visual perception of such a figure, nor

```
int stage2(h,nmax,tol,thresh) float h,tol,thresh; int nmax ;
{float tem,dostep(); int n,i,j; struct cell *p ;
 for(i=0; i < sizev ;i++)
  for(j=0; j < sizeh ; j++)
   if(p=table[i][j]) p->state = p->input ;
 for(n=0; n <= nmax; n++)
  if( dostep(h,thresh) < tol ) return(n) ;
 return(n) ;
}

float dostep(h,thresh) float h,thresh;
{int i,j,k; struct cell *p; float maxch,q,r;
 for(i=0, maxch=0.; i < sizev ; i++)
  for(j=0; j < sizeh ; j++)
   {if( p=table[i][j] )
    {for(k=0,p->change=0.; k < 6 ; k++ )
     {q = gstate(i,j,k) - p->state ;
      r = ginput(i,j,k) - p->input ;
      if( fabs(r) < thresh ) p->change += q ;
      else if( fabs(q) < fabs(r) )
       p->change += (q - r) ;
     }
    p->change *= h/6. ;
    if(fabs(p->change) > maxch) maxch = fabs(p->change) ;
   }
  }
 for(i=0; i < sizev ; i++)
  for(j=0; j < sizeh ; j++)
   if(p=table[i][j]) p->state += p->change ;
 if( pflag ) putout(1) ;
 return(maxch) ;
}

main()
{float thresh,h,tol; int nmax,n ;
 radius=18.; thresh=3.; h=1.; nmax=1000; tol=.00001;
 pflag=1;
 sizev = (2.*radius+3.)/.866025 ;
 sizeh = 2.*radius+3.;
 setup(); getdata() ;
 puts("input data:\n") ; putout(0) ; plotem(0) ;
 n = stage2(h,nmax,tol,thresh) ;
 puts("results:\n") ; putout(1) ; plotem(1) ; exit(0) ;
}
```

Figure 7. A revised version of the third segment of the program, incorporating the modifications to Horn's model.

to the general prediction of retinex lightness theory. What one sees when viewing such a slanted flat-top figure is qualitatively the same as when viewing the Cornsweet figure. The lightness of each region appears uniform, the lightness within the inner circle appears markedly greater than that in the surrounding annulus, and the separating boundary appears sharp and discontinuous.

Because of this anomalous result, the same simulation was reconducted with several different threshold values. The results were similar, except when the threshold was set at .65 or lower. Then the output of *stage2* was identical to the input of *stage1*. The model does not represent the retinex lightness process adequately: it flattens gradients in the proper way only in special figures in which the gradient is closely correlated with the curvature. Otherwise, the reduction of small second differences has no necessary effect on the gradient. The gradient is much more closely related to oriented first differences. A modification of the model, formulated below, shows how easy it is to change the fundamental theoretical assumptions with this general method of simulation.

ANOTHER RETINEX LIGHTNESS MODEL

There are two fundamental changes. The first is that the threshold is now applied to oriented first differences among neighbors in the honeycomb array, rather than to nonoriented second differences. The second is that "slack" is provided to allow the final intensity differences at boundaries to exceed the corresponding differences in the input figure.³ The need for the slack can be illustrated with a simple one-dimensional array of eight elements. Let [2,2,4,5,6,7,2,2] be the input array. It is a simplified one-dimensional mock-up of the slanted flat-top figure defined above. Taking oriented first differences, left to right with zeros at the outer boundaries, yields the result [2,0,2,1,1,-5,0]. If we now apply a threshold of 1.5, the result is [2,0,2,0,0,-5,0]. Inversion of the differencing operation (cumulation, left to right) then gives [2,2,4,4,4,4,-1,-1]. The central slanted segment has been flattened, as it should be, but the result does not satisfy the condition that the last two elements be equal to the first two. In one dimension, this condition corresponds to flatness of the gradient of the surrounding annulus in the two-dimensional slanted flat-top figure. Somehow, this condition must be enforced without defeating the fundamental flattening process. The simplest way seems to be to allow the magnitudes of boundary contrasts in the result to exceed those in the input array. Then, a solution is [2,2,7,7,7,7,2,2]. The largest contrast, at the right-hand boundary, imposes the stress that is relieved by the slack at the left-hand boundary. This is the kind of slack that is implemented in the two-dimensional model.

The detailed modifications are shown in Figure 7, which contains a new version of the complete third section of the program. The *stage1* routine has been eliminated completely, and some of the analogous operations

have been moved to the *stage2* routine to be performed during the iterations. The iterations proceed with repeated calls to the new version of the *dostep* routine.

The thresholding on oriented first differences with slack at the boundaries is implemented in those few statements in *dostep* in which the value of *change* is calculated for each cell. In this process, floating variables *r* and *q* are determined as oriented differences for the *inputs* and *states*, respectively. Six components of *change* are computed, one for each orientation. If the magnitude of *r* is below threshold, then the component of *change* is such that neighboring *states* move toward each other. If the magnitude of *r* exceeds threshold, the component of *change* is determined so that the *state* difference approaches the corresponding *input* difference, but only if the *state* difference is less in magnitude than the *input* difference. Thus, a given *state* difference grows to the corresponding suprathreshold *input* difference, but then may grow further due to the homogenizing pull of neighbors that are only slightly different.

Results from the modified model are shown in Figure 8. These were obtained for the two figures that were defined previously, with the parameter values shown in Figure 7. The modified model does much better on the slanted flat-top figure, and somewhat better on the Cornsweet figure, than does the original model. It will be interesting to see how this model fares under further testing.

In the meantime, this general method of simulation is available for further tests of this model and others that might be of interest. It does not require much primary computer memory. Some of these simulations were performed on a small microprocessor system with 56k bytes

Figure 8. Results of two tests of the revised version of Horn's model. The top trace shows the cross section of the result with the input Cornsweet figure. The second shows the cross section of the result with the slanted flat-top figure. These are the results after 1,000 iterations, and they appear to be near the final asymptotic solutions.

of memory. Simulation times ranged from about 18 h on a 2-mHz 8080 to about 3 h on an 8-mHz 80186 system.

The honeycomb lattice structure may not be crucial for a retinex model. On the other hand, this geometry provides a very simple heuristic way of representing two-dimensional lateral interactions for simulation purposes.

The C language seems much more convenient for this sort of simulation than does Fortran, which I often used in the past. The pointers and structures in C seem to have potential for simulating other kinds of cognitive processes; the present application demanded only some of their more elementary uses.

REFERENCES

- BLACKMAN, R. B., & TUKEY, J. W. (1958). *The measurement of power spectra*. New York: Dover.
- CORNSWEET, T. N. (1966). Stabilized image techniques. In M. A. Whitcomb (Ed.), *Recent developments in vision research* (National Research Council Publication No. 1272). Washington, DC: National Academy of Sciences.
- CORNSWEET, T. N. (1970). *Visual perception*. New York: Academic Press.
- CRAIK, K. J. W. (1966). *The nature of psychology: A selection of papers, essays, and writings* (S. L. Sherwood, Ed.). Cambridge, England: Cambridge University Press.
- HORN, B. K. P. (1974). Determining lightness from an image. *Computer Graphics & Image Processing*, **3**, 277-299.
- LAND, E. H. (1983). Recent advances in retinex theory and some implications for cortical computations: Color vision and the natural image. *Proceedings of the National Academy of Sciences*, **80**, 5163-5169.
- LAND, E. H., & McCANN, J. J. (1971). Lightness and retinex theory. *Journal of the Optical Society of America*, **61**, 1-11.
- MARR, D. (1974). The computation of lightness by the primate retina. *Vision Research*, **14**, 1377-1388.
- MARR, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information*. San Francisco: Freeman.
- MILLER, W. H., & BERNARD, G. D. (1983). Averaging over the foveal receptor aperture curtails aliasing. *Vision Research*, **23**, 1365-1369.
- O'BRIEN, V. (1958). Contour perception, illusion and reality. *Journal of the Optical Society of America*, **48**, 112-119.
- RATLIFF, F. (1965). *Mach bands: Quantitative studies on neural networks in the retina*. San Francisco: Holden-Day.
- WILLIAMS, D. R. (1985). Aliasing in human foveal vision. *Vision Research*, **25**, 195-205.
- YELLOTT, J. I., JR. (1982). Spectral analysis of spatial sampling by photoreceptors: Topological disorder prevents aliasing. *Vision Research*, **22**, 1205-1210.

NOTES

1. A solution can be recognized when found because all values of *change* will approach zero. The important question is whether satisfactory approximations to the solution can be found in reasonable amounts of time. In the present application, each simulation was run for several hours. Horn (1974) described a faster (convolutional) method of obtaining the second-stage solution. The iterative method is used here because it can be applied to a broader class of models.

2. These initial estimates can be set to zero instead, with no effect on the final solution. However, there is a useful side effect of this choice of the initial estimates of the *stage2* solution. If the final solution from the second stage happens to be identical to the original input values to the first stage, that fact will be discovered on the first step of the iterations. The condition can be forced, for debugging purposes, by setting the threshold equal to zero.

3. The methods of Land and McCann (1971) recognized the necessity for thresholding on first differences (or derivatives), but perhaps not for the slack at the boundaries. Under certain conditions, the necessity for the slack would not be demonstrated, and these conditions were approximated in most of the tests performed by Land and McCann (1971). One condition is that if one bounded region completely surrounds another, the luminance ratio must not vary along the boundary between them. This is satisfied by the Cornsweet figure but is violated by the slanted flat-top figure.

(Manuscript received August 30, 1985;
revision accepted for publication March 27, 1986.)