9 Depth Perception

A. Weitzenfeld and M. Arbib

9.1 Introduction

Depth Perception enables us to see the world in terms of objects located at various distances from us. From a single eye at a single time we can determine the direction in space of various features of the world. Different techniques are available to locate where the feature is in depth along the given direction (see Arbib 1989 for further details):

- *Stereopsis* uses cues provided by correlating visual input to two spatially separated eyes.
- *Optic flow* uses the information provided to the eye at different times.
- *Accommodation* works by determining what focal length will best bring an object into focus.
- *Convergence* is based on how the eyes must turn to fixate the object in question.

A three dimensional scene presented to the left eye differs from that presented to the right eye. A single point projection on each retina corresponds to a whole ray of points in space, but points on two retinae determine a single point in space, the intersection of the corresponding rays (figure 9.1).

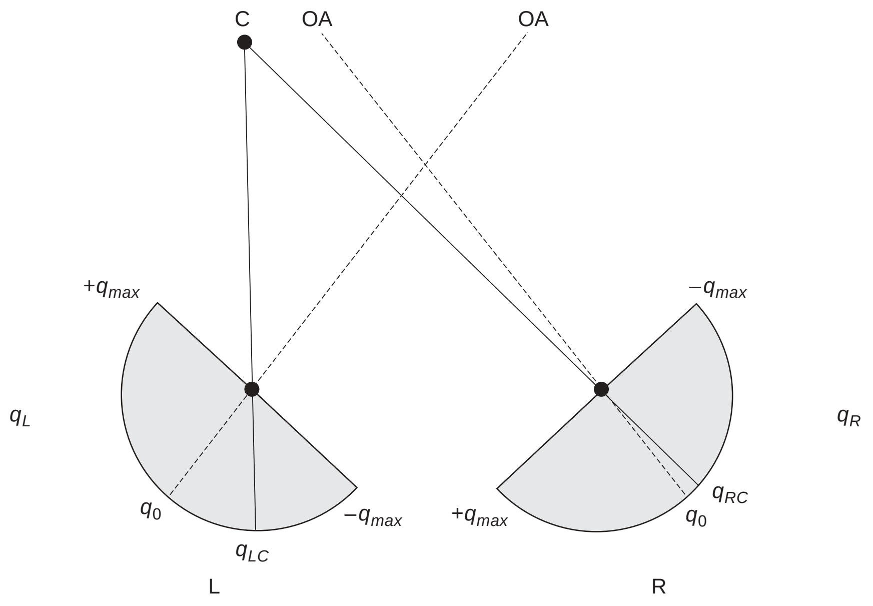

Figure 9.1
Points on a single ray (projector) match to a single point on each retina. The inter-section of two rays, a ray for each retina, determines a single point in space. In this example, *C* projects to q_{RC} on the right retina and q_{LC} on the left retina, respectively. *OA* defines the *optic axis.*

In *stereopsis*, depth computation is based on the disparity or displacement between the projection of corresponding points on the two retinae. For example, from figure 9.1, the disparity generated by the projection of point *C* in the two retinae is defined as the spatial displacement between q_{RC} and q_{LC}. This is calculated by $(q_{RC}–q_0) – (q_{LC}–q_0) = q_{RC} –q_{LC}$.

In order to visualize the relationship between disparity *d* and retinal angle *q*, the mapping of the right eye coordinate system onto a Cartesian grid is shown in figure 9.2. Radial lines are at equal angular increments *q* and arcs are lines of constant disparity *d* spaced at equal increments of disparity. Disparity increases as the arcs get closer to the retina, where depth resolution becomes finer for closer objects.

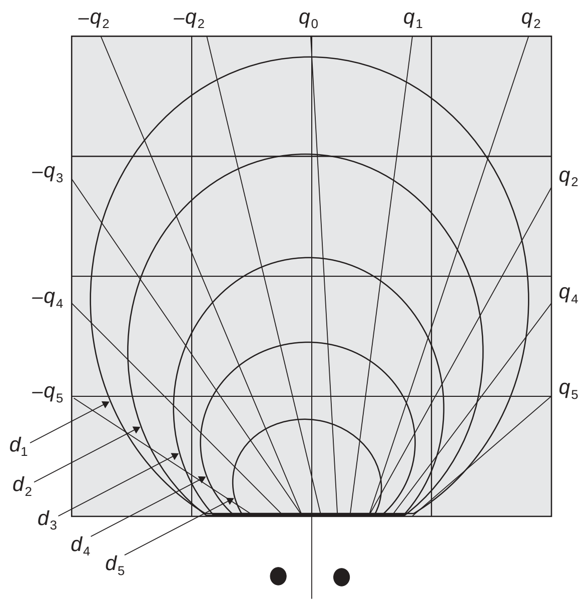

Figure 9.2
Mapping of the right eyed coordinate system onto a Cartesian grid. The two disks under the grid represent the two eyes. Arcs correspond to disparity *d*, increasing as objects get closer to the retinae. Disparity resolution increases as well, as the arcs get closer to the eyes. Angular position *q* corresponds to the rays emanating from the retina at constant angular increments.

An problem arising from *stereopsis* is the ambiguity created by pairs of points generating similar retinal projections (figure 9.3),

$$q_{RA} = q_{RD}, q_{LA} = q_{LC}, q_{RB} = q_{RC}, q_{LB} = q_{LD}$$

The generated disparities are:

$$d_A = q_{RA} - q_{LA}, d_B = q_{RB} - q_{LB}, d_C = q_{RC} - q_{LA}, d_D = q_{RD} - q_{LD}$$

giving rise to the following ambiguity:

$$d_A + d_B = d_C + d_D$$

where one pair of points would be the be correct one, while the second pair corresponds to "ghost" points emerging from the disparity maps. In this situation there is no way of knowing which pair is the "true" pair of points.

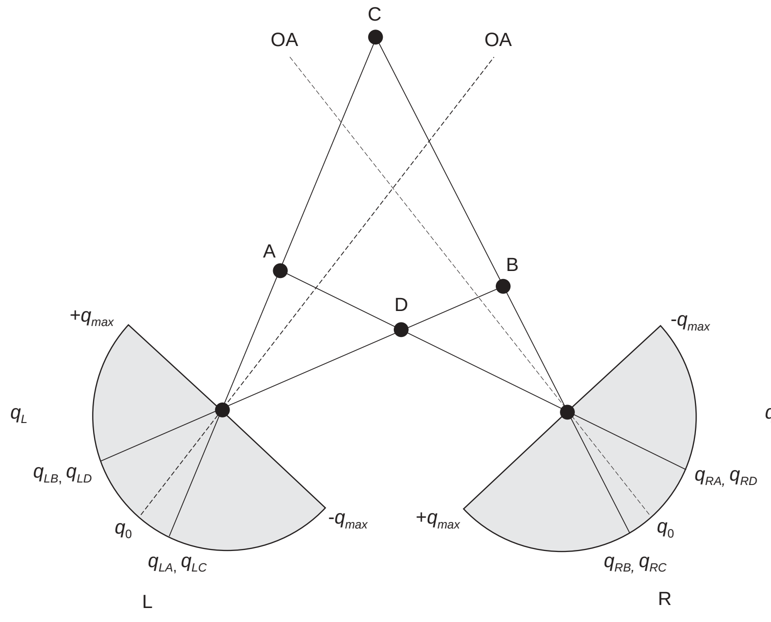

Figure 9.3
There is an ambiguity generated by projections of pairs of points. Stimuli *B* and *C* project to the same point on the right retina, similarly *A* and *D*. The same happens on the left retina, where *A* and *C* match to the same point, similarly to *B* and *D*. While one of the pairs may be the true pair, there is no way to distinguish it from the second ghost pair since their projection is exactly the same.

Two depth perception models resolving this ambiguity are described here: (1) The disparity model by Dev (1975), based exclusively on disparity cues, and (2) the disparity and accommodation model by House (1985), using accommodation cues cooperatively with disparity cues to improve depth estimates.

9.2 Model Description: Disparity

To remove the ambiguity problem Dev (1975) developed a cooperative computational model for building the depth map "guided by the plausible hypothesis that our visual world is made up of relatively few connected regions." This model used neurons whose firing level represented a degree of confidence that a point was located at a corresponding position in three-dimensional space. The neurons were so connected via inhibitory interneurons that cells that coded for nearby directions in space and similar depths should excite each other, whereas cells that corresponded to nearby directions in space and dissimilar depths should inhibit each other. The model is shown in figure 9.4: (1) an excitatory manifold *M* indexed by retina position *q* and disparity *d*, where nearby cells excite each other, and cells for a given *q* and differing *d* inhibit each other via (2) an inhibitory interneuron *U* indexed only by retinal position *q*. Competition along the *d* dimension ensures that for each *q*, a cell (*q*,*d*) will be active for at most one *d*; cooperation along the *q* dimension encourages groups of active cells for a nearby *q* to have similar *d*, thus yielding a segmentation of the image. (See Amari and Arbib 1977 for more detail).

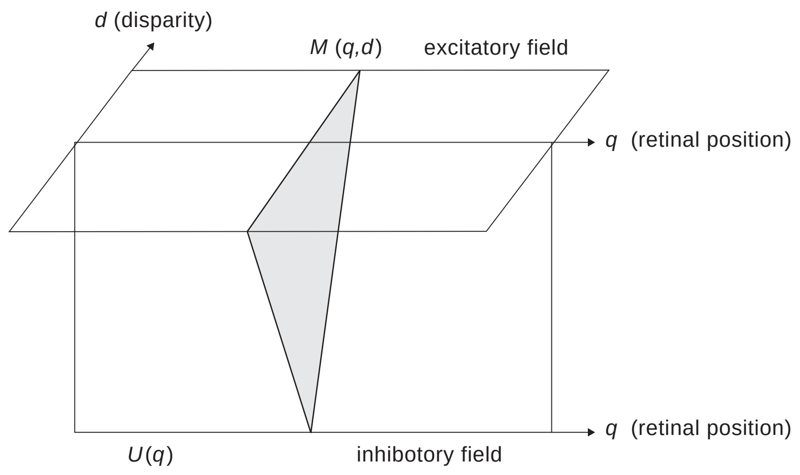

Figure 9.4
The Dev disparity model incorporates an excitatory manifold *M* indexed by retina position *q* and disparity *d*, and an inhibitory interneuron *U* indexed only by retinal position *q*.

The model contains a **Dev** disparity module (figure 9.5):

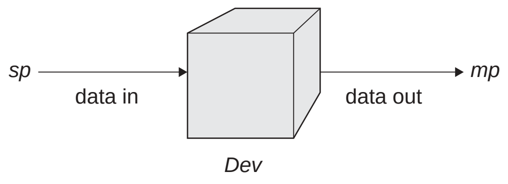

Figure 9.5
The **Dev** disparity module contains an input port **sp** that receives external data and an output port **mp** that generates output data.

The **Dev** module is implemented by a neural network described by the following equations, where *m* corresponds to the excitatory field, *u* corresponds to the inhibitory field and *s* corresponds to the input from the retina (figure 9.6):

$$\tau_m \frac{\partial m_{ij}}{\partial t} = -m_{ij} + w_m * f(m_{ij}) - w_u * g(u_j) - h_m + s_{ij} \tag{9.1}$$

$$\tau_u \frac{\partial u_j}{\partial t} = -u_j + \sum_i f(m_{ij}) - h_u \quad (9.2)$$

$$f(m_{ij}) = step(m_{ij}, k) \quad (9.3)$$

$$g(u_j) = ramp(u_j) \quad (9.4)$$

Input s is computed by calculating disparity between left r_L and right r_R retina mappings. $r_L(q)$ is set to 1 if some object projects to point q on the left retina, $r_L(q)$ is set to 0 otherwise; and similarly for $r_R(q)$. Stereo input is then defined as

$$s_d(q) = R_L(q)\, R_R(q+d) \quad (9.5)$$

which is 1 only if there is an object at position q on the left retina as well as at $q+d$ on the right retina, and is otherwise 0. In the present section, we simply present the computed s-array to the Dev module; in the second half of the chapter we will present a Retina module that explicitly computes the s-array (and an accommodation array) from the activity on the 2 retinas. (Note that a more subtle version of the model would require similar local features, rather than mere presence of an object, at q_L and $(q+d)_R$. However, the modular design of NSL would come to the rescue here, since the input s, rather than being computed by the above formula, would then be supplied by a module computing feature-based disparities instead.)

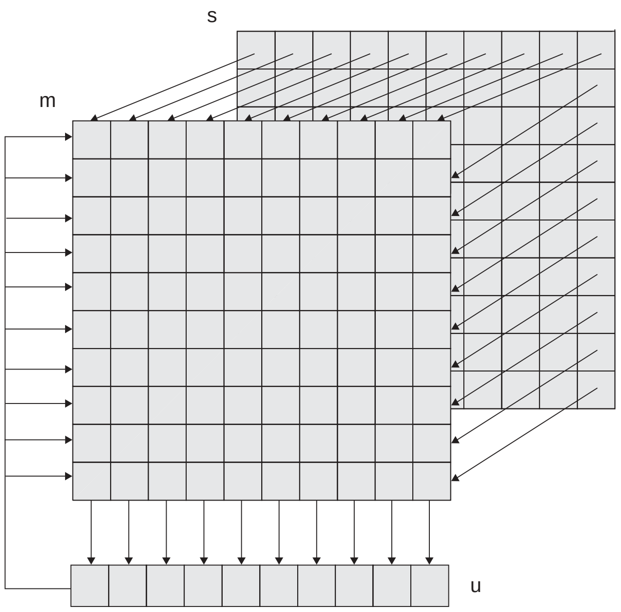

Figure 9.6
The **Dev** module consists of layers *m* corresponding to the excitatory field and *u* corresponding to the inhibitory field and *s* to the input from the retina.

9.3 Model Implementation: Disparity

There exist a number of possible different designs when building the model in NSLM. Different neural layers may be assigned to different modules, or a number of them may be part of a single module. In the design presented here, the second approach is taken, where the **Dev** and **DevModel** implementation in NSLM are as shown below:

Dev

The **Dev** module has the following definition:

```
nslModule Dev (int sizeX, int sizeY) {
```

Input layer *s*(*d*,*q*), corresponds to the *s* layer. The excitatory layer *m*(*d*,*q*) and inhibitory layer *u*(*q*) are each defined by a membrane potential array (*mp* and *up*) and a firing array (*mf* and *uf*).

```
private NslDinFloat2 s(sizeX, sizeY);
private NslDoutFloat2 mf (sizeX, sizeY);
private NslFloat2 mp(sizeX, sizeY);
private NslFloat1 up (sizeY);
private NslFloat1 uf(sizeY);
```

Following, constants are declared. *tm* and *tu* are time constants, *hm* and *hu* are threshold constants. *ksm*, *kmu*, and *kum* are connectivity constants; *k* is used as a step function parameter; and *wm* is used as an excitatory convolution mask.

The **initRun** procedure reinitializes all layers for each simulation run:

```
public void initRun () {
   mp = 0;
   mf = 0;
   up = 0;
   uf = 0;
}
```

The **simRun** procedure defines the dynamic equations:

```
public void simRun () {
   mp  = nslDiff(mp, tm, -mp + ks*s –
          kum*nslExpandRows(uf,mp.getRows())+ wm@mf + hm);
   mf = nslStep(mp, k);
   up  = nslDiff(up,tu, -up + kmu*nslReduceRows(mf) + hu);
   uf = nslRamp(up);
}
```

DepthModel

The **DepthModel** module is used to instantiate the **Dev** module. No connections are required between the two modules since no information is passed. The **Dev** module produces output but does not receive input from other modules. (Input port *sp* is actually not used, but is defined for future extensibility.)

```
nslModel DepthModel () {
```

Constant sizes for arrays are:

```
private int sizeX = 10;
private int sizeY = 8;
```

The assemblage consists of the following module:

```
private Dev dev(sizeX, sizeY);
```

9.4 Simulation and Results: Disparity

The NSLS script for the Dev model contains system simulation parameter assignments. Three of these parameters are time step, simulation end time, and the approximation method:

```
nsl set system.simDelta 0.1
nsl set system.simEndTime 10.0
nsl set system.diff.approximation euler
```

The **Dev** module parameters are then assigned. Connectivity constants are assigned to 1.0. *tm* and *tu* need a common value, as well as *hm* and *hu*. *wm* is assigned five elements between 0 and 1.0. (These constants could have been assigned values directly in NSLM, but can be overridden by the script language.):

The parameters within the Dev module are set as follows:

```
nsl set devModel.kum 1.0
nsl set devModel.kmu 1.0
nsl set devModel.ks 2.0
nsl set devModel.wm 0.4 0.6 1.0 0.6 0.4
nsl set devModel.tm 1.0
nsl set devModel.tu 1.0
nsl set devModel.hm -1.2
nsl set devModel.hu -0.7
nsl set devModel.k 0.75
```

Input data is directly generated into *s*, mapping real points as well as “ghosts” (points with value 1). In the present example, we have followed the scenario of figure 9.3 where 2 “real” points generate a set of disparities that is also consistent with 2 “ghost” points, yielding a total of 4 “initial candidates” in the array below.

```
nsl set devModel.s {
{ 0 0 0 0 0 0 0 0 }
{ 0 0 0 0 0 0 0 0 }
{ 0 0 0 0 1 0 0 0 }
{ 0 0 0 0 0 0 0 0 }
{ 0 0 1 0 1 0 0 0 }
{ 0 0 0 0 0 0 0 0 }
{ 0 0 1 0 0 0 0 0 }
{ 0 0 0 0 0 0 0 0 }
{ 0 0 0 0 0 0 0 0 }
{ 0 0 0 0 0 0 0 0 }}
```

Graphics is specified by first creating a frame to contain the desired display windows:

```
nsl create DisplayFrame .depth
```

In each frame we reproduce three windows containing a layer activity each: on top the input *s* staying the same throughout the simulation, in the middle the main layer activity *mp* and in the bottom the main layer activation *mf*. The three display windows are created using the default layout where each new window is added beneath the previous one:

```
      nsl create DisplayWindow  s -width 500 -height 200
         -graph areaLevel -wymin -1.0 -wymax 2.0
      nsl create DisplayWindow mp -width 500 -height 200
         -graph areaLevel -wymin -3.0 -wymax 3.0
      nsl create DisplayWindow mf -width 500 -height 200
         -graph areaLevel -wymin 0.0 -wymax 1.0
```

Simulation results during time steps 0 and 2, corresponding to the network building up internal values, are shown in figure 9.7.

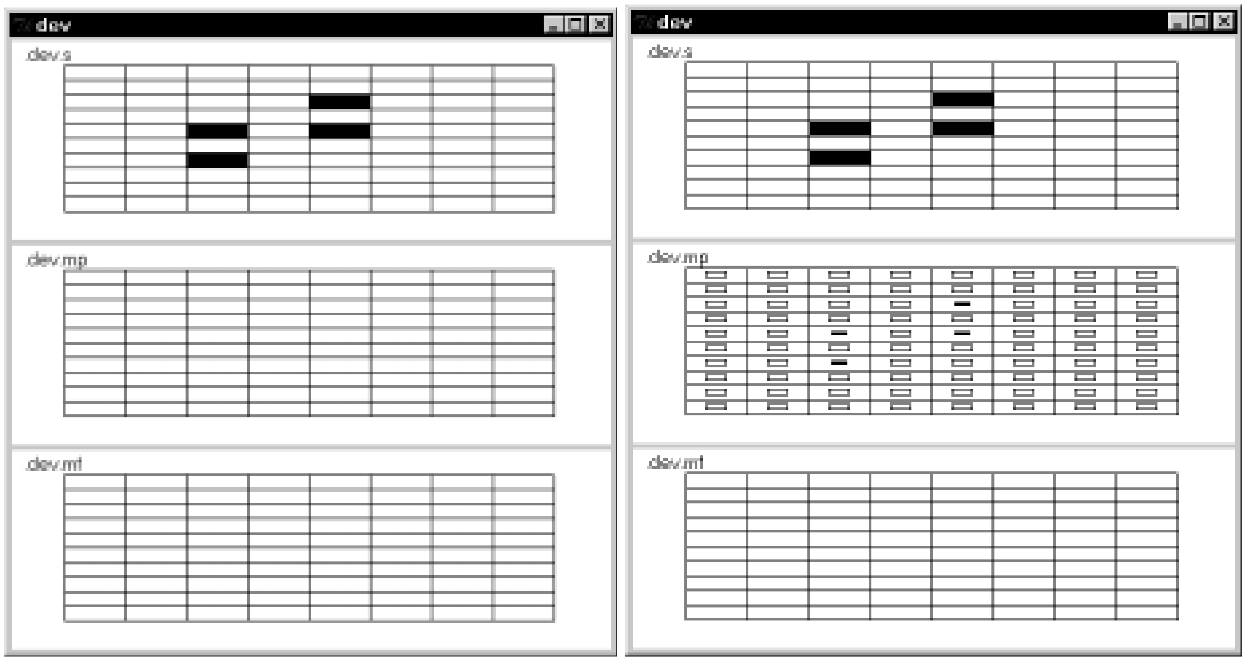

Figure 9.7
The Dev model simulation steps 0 and 2.

Simulation results during time steps 4 and 6, corresponding to the network completing building up internal values and solving the ambiguity, are shown in figure 9.8.

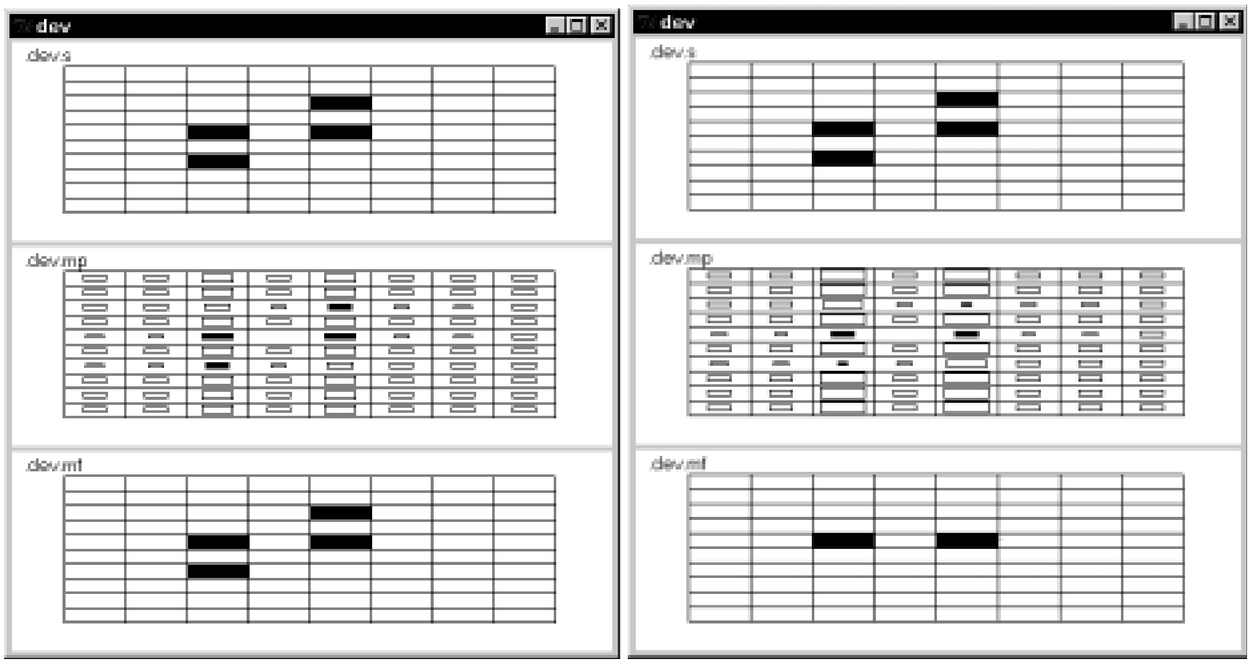

Figure 9.8
The Dev model simulation steps 4 and 6.

Returning to the situation shown in figure 9.3, the reader will note that the Dev model favors targets *A* and *B* as the "real" targets, and exorcises *C* and *D* as "ghost targets", even though we had noted that the retinal data were neutral as to the choice of (*A*,*B*) versus (*C*, *D*). This is because the design of the Dev model meets the constraint that the world is made up of surfaces, and thus favors a choice consistent with nearby points of similar disparity over other choices. We now turn to an architecture which exploits accommodation as well as disparity cues. Although we do not show this explicitly, the reader can check that if *C* and *D* are the "real" inputs, then the new model will verify this, whereas the Dev model will not.

9.5 Model Description: Disparity and Accommodation

In many cases, depth perception models depending entirely on disparity cues will converge to an adequate depth segmentation of the image. However, such a system may need extra cues. The ambiguity resulting from matching a number of points in space to the same retina coordinate can be reduced by the use of vergence information to give the system an initial depth estimate. Another method is to use accommodation information to provide the initial bias for a depth perception system. It is the latter approach that we adopt here.

The cue interaction model (House 1985) uses two systems, each based on Dev's stereopsis model, to build a depth map. One is driven by disparity cues, the other by accommodation cues, while corresponding points in the two maps have excitatory cross-coupling. The model is sketched in figure 9.9. *M* is an accommodation driven field; it receives information about accommodation and—left to its own devices—sharpens up that information to yield depth estimates. *S* is the disparity driven-field, corresponding to Dev's original system: it receives disparity information and suppresses (what may be) ghost targets. Moreover, the systems are intercoupled so that a point in the accommodation field *M* excites the corresponding point in the disparity field *S*, and viceversa. Thus, a high confidence in a particular (direction, depth) coordinate in one layer will bias activity in the other layer accordingly. The model is so tuned that binocular depth cues predominate where available, but monocular accommodative cues remain sufficient to determine depth in the absence of binocular cues.

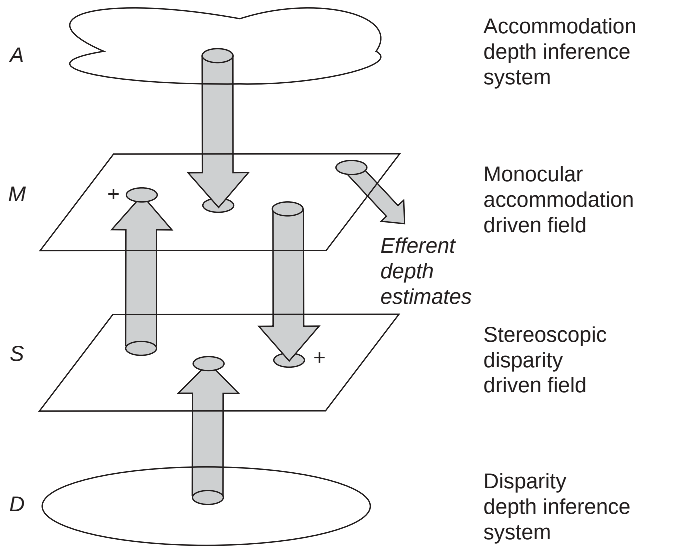
A
Accommodation depth inference system
M
+
Monocular accommodation driven field
Efferent depth estimates
S
+
Stereoscopic disparity driven field
D
Disparity depth inference system

Figure 9.9
The cue interaction model for depth mapping uses cross-coupling between an accommodation-driven system and a disparity-driven system.

The model is composed of the following modules (figure 9.10):

- **Dev2**: There are two instances of an extended Dev module, now called Dev2, one processing disparity information and the other accommodation information. Each consists of an input port *a*, receiving data from the **Retina**, a second input port *s*, receiving input from the other Dev2 module, and an output port *mf*, delegating its output back to Stereo.
- **Retina**: The Retina module processes retina information. It contains an input port *in*, delegated from Stereo, and two output ports, *d* and *a*, for disparity and accommodation, respectively.
- **Stereo**: The Stereo assemblage provides composition and encapsulation for the entire model. It delegates its processing to the Retina and two Dev2 modules. The Stereo module consists of two external ports, input port *in* and output port *out*.
- **Visin**: The Visin module generates the external stimuli. It contains an *out* output port connected to the Stereo module.

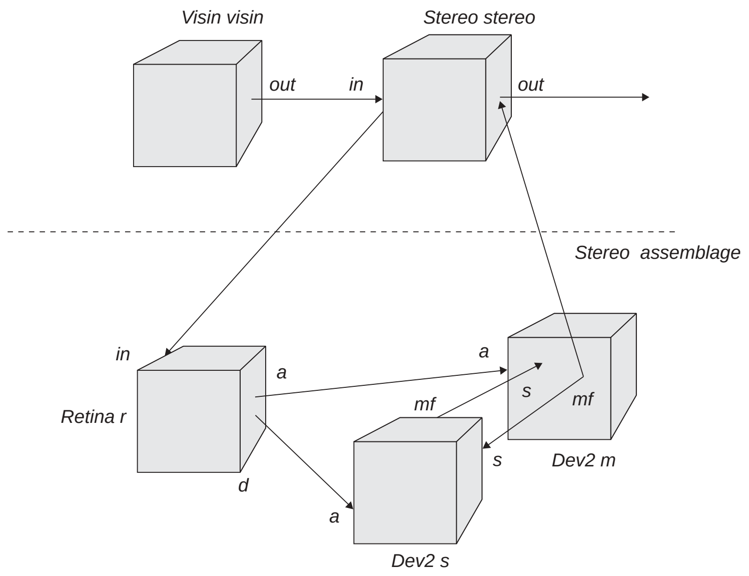

Figure 9.10
Disparity and Accommodation model modules: A Visin input module, a Stereo assemblage, a Retina module, and two Dev2 modules.

Note, the two Dev2 modules are implemented by neural networks similar to those defined in the original Dev module. These equations are extended to enable cross-coupling between Dev modules, where *m* and *s* correspond to the excitatory fields, *u* and *v* to the inhibitory fields and *a* and *d* are the input from the retina (figure 9.11). Consistent with the definition of Stereo below, you must make the Dev2 modules identical—we have 2 instances of the same module, and we will show how we connect them to yield their differential function. The variables internal to each instance must be identical, only connections and relabelings distinguish them:

- Disparity (**Dev2** *s*):

$$\tau_s \frac{\partial s_{ij}}{\partial t} = -s_{ij} + w_s * f(s_{ij}) + w_t * f(t_{ij}) - w_v * g(v_j) - h_s + d_{ij} \tag{9.6}$$

$$\tau_v \frac{\partial v_j}{\partial t} = -v_j + \sum_i f(s_{ij}) - h_v \tag{9.7}$$

$$f(s_{ij}) = sigma(s_{ij}) \tag{9.8}$$

$g(v_j) = ramp(v_j)$ (9.9)

- Accommodation (**Dev2** *m*):

$$\tau_m \frac{\partial m_{ij}}{\partial t} = -m_{ij} + w_m * f(m_{ij}) + w_t * f(t_{ij}) - w_u * g(u_j) - h_m + a_{ij} \tag{9.10}$$

$$\tau_u \frac{\partial u_j}{\partial t} = -u_j + \sum_i f(m_{ij}) - h_u \tag{9.11}$$

$$f(m_{ij}) = sigma(m_{ij}) \tag{9.12}$$

$$g(u_j) = ramp(u_j) \tag{9.13}$$

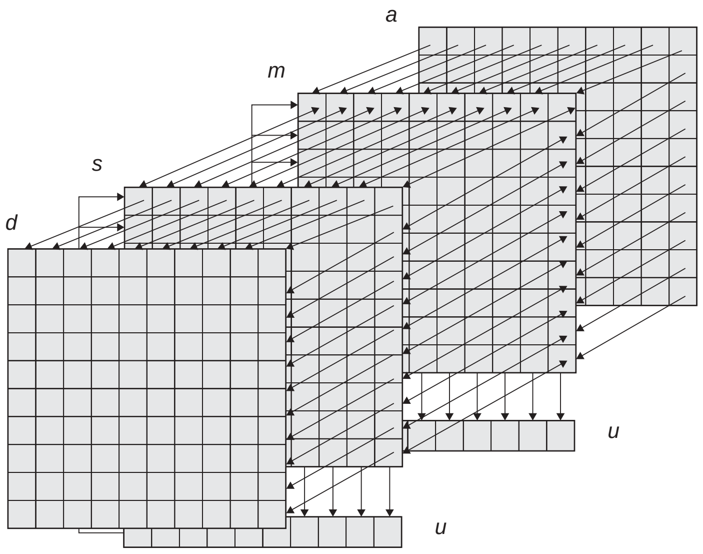

Figure 9.11
Two **Dev** neural networks consisting of *m* and *s* corresponding to the excitatory fields, *u* and *v* to the inhibitory fields, and *a* and *d* are the input from the retina.

9.6 Model Implementation: Disparity and Accommodation[1]

The model is composed of the following modules: **Dev2**, **Retina**, **Stereo**, and **Visin**:

Dev2

The **Dev2** module is defined as before:

```
nslModule Dev2 (int sizeX, int sizeY) {
```

Input layer *s*(*d*,*q*), corresponds to the *s* layer. An additional layer *t* is used for cross-coupling between **Dev2** modules.

```
public NslDinFloat2 a(sizeX,sizeY);
public NslDinFloat2 s(sizeX,sizeY);
public NslDoutFloat2 mf(sizeX,sizeY);
private NslFloat2 mp(sizeX,sizeY);
private NslFloat1 up(sizeY);
private NslFloat1 uf(sizeY);
```

Following, constants are declared similar to the original Dev module. *wm* is the convolution mask, with size 3, and instead of a step function, a saturation function is used; thus instead of *k*, *x1* and *x2* are used as parameters.

```
private NslFloat0 ksm();
private NslFloat0 kmu();
private NslFloat0 kum();
private NslFloat0 ktm();
private NslFloat1 wm(3);
private NslFloat0 tm();
private NslFloat0 tu();
private NslFloat0 hm();
private NslFloat0 hu();
private NslFloat0 x1();
private NslFloat0 x2();
```

The **initRun** procedure reinitializes all layers to 0 for each simulation run:

```
public void initRun () {
   mp = 0;
   mf = 0;
   up = 0;
   uf = 0;
}
```

The **simRun** procedure defines the dynamic equations. They are similar to the Dev equations except for the addition of *t* in the expression:

```
public void simRun () {
   mp = nslDiff(mp, tm, -mp + ksm*s –
      kum*nslExpandRows(uf,mp.getRows())+ ktm*t + wm@mf + hm);
   mf = nslSigmoid(mp,x1,x2);
   up = nslDiff(up,tu, -up + kmu*nslReduceRows(mf) + hu);
   uf = nslRamp(up);
}
```

Retina

The **Retina** module is defined as follows (detailed processing for this assemblage is described in House (1985)):

```
   nslModule Retina (int sizeX, int sizeY, int sizeR) {
```

The external data arrays are: world input *in* and output retina vector *r*:

```
   private NslDinFloat2 in(sizeX, sizeY);  // world input
   private NslDoutFloat2 a(sizeX, sizeY); // accommodation layer
   private NslDoutFloat2 d(sizeX, sizeY); // disparity layer
   private NslFloat1 rr(sizeR);     // right retina
   private NslFloat1 rl(sizeR);     // left retina
```

Following, constants are declared:

```
   private NslFloat0 w(); // 1/2 of interpupillary distance (cm)
   private NslFloat0 yf();// intersection of optical axes
      (0,yf) (cm)
   private NslFloat0 l(); // interpupillary line distance from
      origin (cm)
   private NslFloat0 dmax();  // maximum disparity
   private NslFloat0 sigma(); // spread parameter
```

The **initRun** procedure produces the retina mapping (since images are static, there is no need for a **simRun** procedure):

```
   public void initRun()
   {
      view_to_right_retina(rr,in,w,yf,l);
      view_to_left_retina(rl,in,w,yf,l);
      retina_to_accommodation(a,in,rr,w,yf,l,dmax,sigma);
      retina_to_disparity(d,rr,rl);
   }
```

Stereo

The **Stereo** assemblage is defined as follows:

```
   nslModule Stereo (int sizeX, int sizeY, int sizeR) {
```

The assemblage consists of the following modules:

```
   private Retina r(sizeX, sizeY, sizeR);
   private Dev2 m(sizeX, sizeY), s(sizeX, sizeY);
```

Input and output ports are defined as follows:

```
   public NslDinFloat2 in(sizeX, sizeY);
   public NslDoutFloat2 out(sizeX, sizeY);
```

Connections and relabels are as follows:

```
public void makeConn () {
   nslConnect (r.a,m.a);
   nslConnect (r.d,s.a);
   nslConnect (m.mf,s.s);
   nslConnect (s.mf,m.s);
   nslRelabel (in,retina.in);
   nslRelabel (s.mf,mf);
}
```

Visin

The **Visin** module is defined as follows:

```
nslModule Visin (int sizeX, int sizeY) {
```

The external data array is: world input *in*:

```
private NslDoutFloat2 out(sizeX, sizeY);
```

DepthModel

The **DepthModel** model instantiates the **Visin** and **Stereo** modules. A connection is made between ports in these two modules.

```
nslModel DepthModel () {
```

Constant sizes for arrays are:

```
private int sizeX = 11; // 81;
private int sizeY = 11; // 81;
private int sizeR = 21; // 161;
```

The assemblage consists of the following modules:

```
private Visin visin(sizeX, sizeY);
private Stereo stereo(sizeX, sizeY,sizeR);
```

Connections and relabels are as follows:

```
public void makeConn () {
   nslConnect (visin.out, stereo.in);
}
```

9.7 Simulation and Results: Disparity and Accommodation[2]

The NSLS code for the House model involves system simulation parameter assignments, including time steps, simulation end time, and the integration method to be used by all differential equations:

```
nsl set system.simDelta 0.05
nsl set system.simEndTime 2.0
nsl set system.diff.approximation euler
nsl set system.diff.delta 0.05
```

Retina parameters,

```
nsl set DepthModel.stereo.r.w 3.0
nsl set DepthModel.stereo.r.yf -10.0
nsl set DepthModel.stereo.r.l 22.0
nsl set DepthModel.stereo.r.dmax 0.25
nsl set DepthModel.stereo.r.sigma 0.25
```

Dev2 disparity parameters,

```
nsl set DepthModel.stereo.s.tu 0.1
nsl set DepthModel.stereo.s.tm 0.3
nsl set DepthModel.stereo.s.hu 0.0
nsl set DepthModel.stereo.s.hm 0.0
nsl set DepthModel.stereo.s.x1 0.1
nsl set DepthModel.stereo.s.x2 1.1
nsl set DepthModel.stereo.s.wm 0.25 0.68 0.25
nsl set DepthModel.stereo.s.kmu 1.0
nsl set DepthModel.stereo.s.kam 0.5
nsl set DepthModel.stereo.s.kum 0.6
nsl set DepthModel.stereo.s.ks 0.8
```

Dev2 accommodation parameters,

```
nsl set DepthModel.stereo.m.tu 0.1
nsl set DepthModel.stereo.m.tm 0.3
nsl set DepthModel.stereo.s.hu 0.0
nsl set DepthModel.stereo.s.hm 0.0
nsl set DepthModel.stereo.m.x1 0.1
nsl set DepthModel.stereo.m.x2 1.1
nsl set DepthModel.stereo.m.wm 0.25 0.68 0.25
nsl set DepthModel.stereo.m.kmu 1.0
nsl set DepthModel.stereo.m.kam 0.5
nsl set DepthModel.stereo.m.kum 0.6
nsl set DepthModel.stereo.m.ks 0.8
```

Input to the model, **Visin** module, is

```
nsl set DepthModel.visin.out{35,41} 1.0
nsl set DepthModel.visin.out{45,55} 1.0
```

Graphics is specified as follows:

```
nsl create DisplayFrame .fw0
nsl create DisplayWindow vis.out -width 450 -height 600 -graph
    areaLevel \-wymin 0.0 -wymax 1.0
nsl create DisplayFrame .fw1
nsl create DisplayWindow st.m.mf -width 450 -height 300 -graph
    spatial3 \   -wymin -1.0 -wymax 1.0 -x0 20 -x1 80 -y0 35
    –y1 50 -sz 100 nsl create DisplayWindow st.s.mf -width 450
    -height 300 -graph spatial3 \
   -wymin -1.0 -wymax 1.0 -x0 20 -x1 80 -y0 35 -y1 50 -sz 100
```

Simulation input (time step 0) is shown in figure 9.12. Input array *in* and output arrays *a* and *d*, all read from the **Retina**.

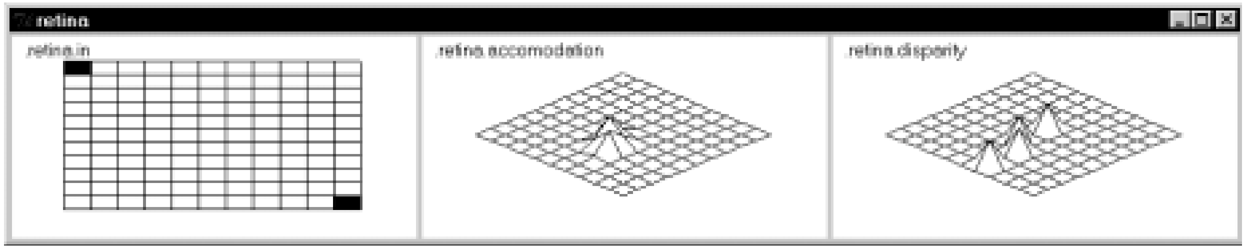

Figure 9.12
World input for processing: Input array *in* and output arrays *a* and *d*, from the **Retina**.

Simulation for disparity *s* and accommodation *m* corresponding to the two **Dev2** modules, is shown in figure 9.13, during time 0.25.

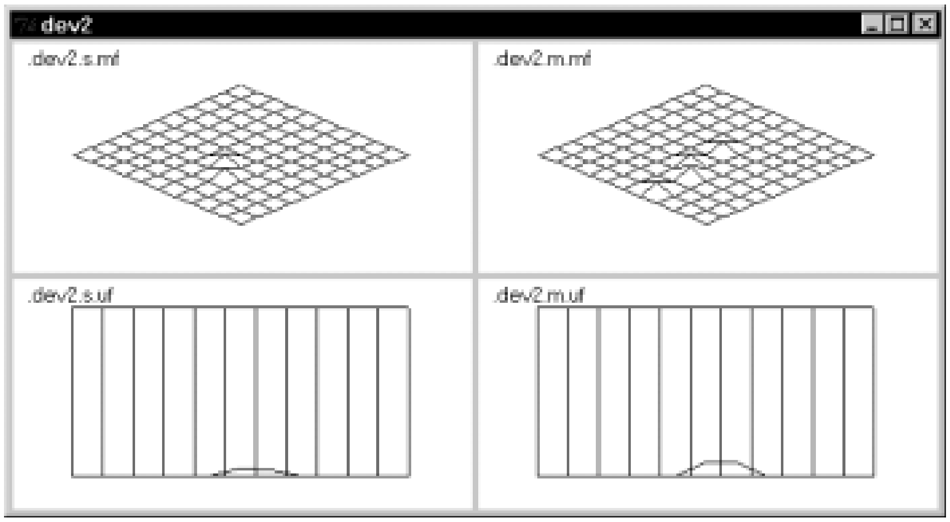

Figure 9.13
Disparity *s* and accommodation *m* corresponding to the two **Dev2** modules, during time 0.25.

Simulation for disparity *s* and accommodation *m* corresponding to the two **Dev2** modules, is shown in figure 9.14, during time 0.50.

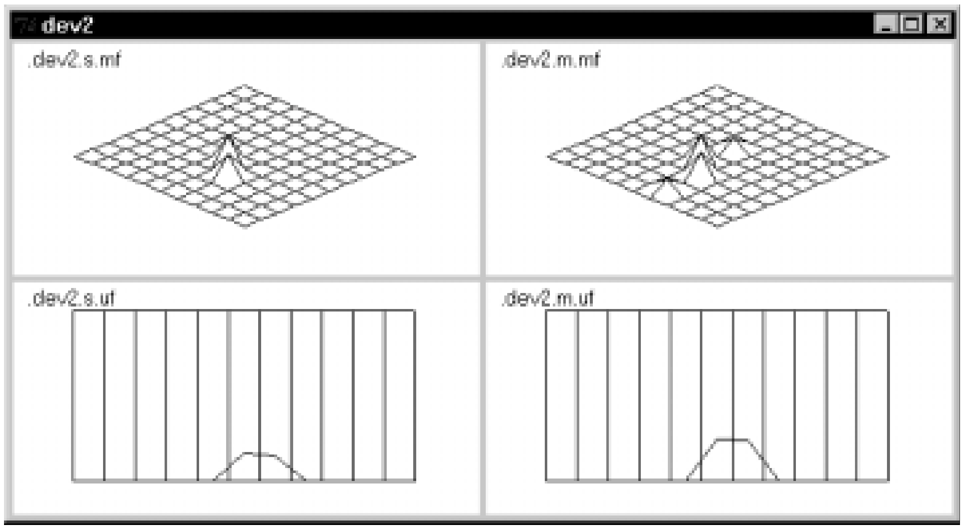

Figure 9.14
Disparity *s* and accommodation *m* corresponding to the two **Dev2** modules, during time 0.50.

Simulation for disparity *s* and accommodation *m* corresponding to the two **Dev2** modules, is shown in figure 9.15, during time 0.75.

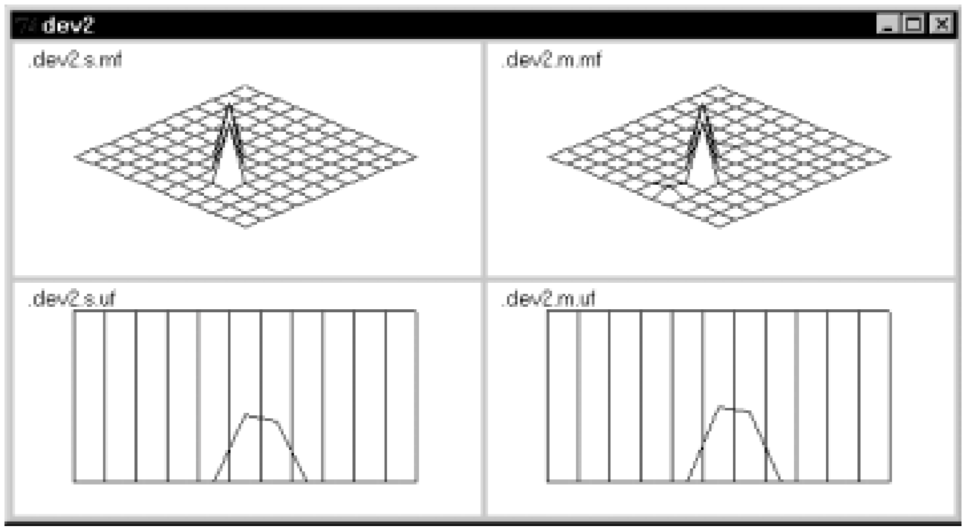

Figure 9.15
Disparity *s* and accommodation *m* corresponding to the two **Dev** modules, during time 0.75.

9.8 Summary

We have shown how to take advantage of the features in NSLM to modularly extend the original Dev disparity model into the House depth perception model. The ability to extend models makes NSL a very powerful simulation language. There are other modular decompositions alternative to the one presented in this model. When to choose one decomposition versus another one, depends on the complexity of the model and how much extensibility is desired. For example, we could further decomposing the retina module into an assemblage made of a left and right retina and disparity and accommodation components as shown in figure 9.16. This would require further refinement of the model equations and would be useful as far as we can actually assign separate code to each box. We leave this to the user as an exercise.

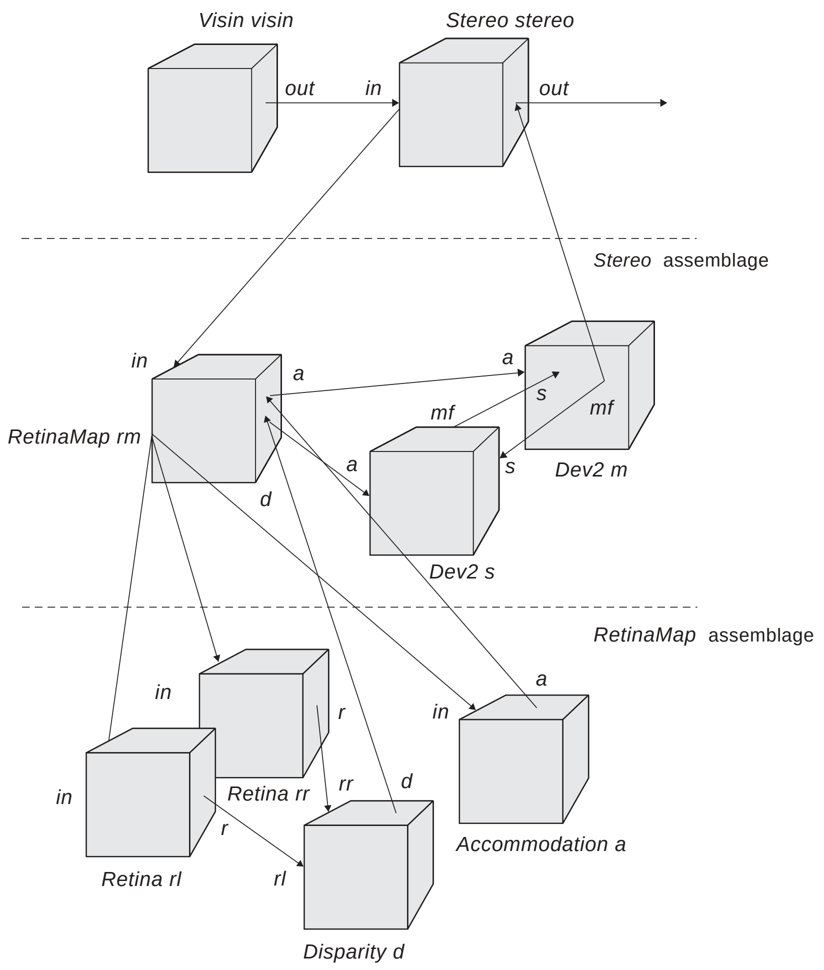

Figure 9.16
Additional depth model decomposition by making the Retina module a RetinaMap assemblage composed of two retina submodule (left and right) and separate accommodation and disparity submodules.

Notes

1. The Depth Perception model was implemented and tested under NSLC.
2. The Depth Perception model was implemented and tested under NSLC.