Implement Neural Mesh on Cellular Neural Network

Tai-Hei Wu, Yng-Kae Tzeng, Cheng-Yuan Liou[1]
Department of Computer Science and Information Engineering, National Taiwan University
Taipei, Taiwan, R.O.C.
This work was supported by NSC 93-2213-E-002-008. Correspondent[1].

***Abstract*— A mesh model is implemented on cellular neural networks. We devise a new energy function for the mesh model in the form of cellular neural network. By operating the parameters of the cellular neural networks, the mesh guided by this energy function will behave as in.**

***Index Terms*— Hopfield model, cellular neural network, meshed snakes**

I. Introduction

Mesh generation has been studied for object modeling, nonuniform sampling, visual data reconstructing, etc. A mesh is designed to represent an image with a small number of nodes conveying most of the information about the image. There are several works which focus on these topics, e.g., Weiss[15], Terzopoulos[6][11], Wang and Lee[14]. In [8] and [10], mesh is used to emulate soap film and solve the Steiner Tree Problem[3]. [9] proposes a method that uses mesh to solve the minimal surface problems. They use Potts neural network to minimize a three-dimensional mesh. In this paper, we implement the neural mesh discussed in [7] on cellular neural networks[2]. We use a similar approach in [7] but ours is more suitable for cellular neural networks to implement mesh. We define an energy function for mesh and then minimize it using cellular neural networks.

II. Mesh on the Hopfield Network

A. The Snake

Kass et al.[6] proposed an energy-minimizing active contour model (snake) which could drive a set of points to lie on features of interest, e.g. edges, in a image. Representing the position of a snake parametrically by $v(s) = (x(s), y(s))$, the energy function can be written as:

$$\begin{aligned} E^*_{snake} &= \int_0^1 E_{snake}(v(s))\,ds \\ &= \int_0^1 [E_{int}(v(s)) + E_{image}(v(s)) + E_{con}(v(s))]\,ds. \end{aligned} \quad (1)$$

Where, E_{int} represents the internal energy of the spline due to bending or discontinuities, E_{image} represents the image forces, and E_{con} is the external constrained forces. The internal spline energy can be written as:

$$E_{int} = \left(\mu(s)\,|v_s(s)|^2 + \nu(s)\,|v_{ss}(s)|^2\right)/2.$$

In the above equation, the first-order term causes the snake to behave like a string(i.e. resists stretching) and the second-order term causes the snake to behave like a rod (i.e. resists bending). Adjusting the weights, $\mu(s)$ and $\nu(s)$, controls the relative importance of the string and rod terms. The image force in equation (1) comes from various events such as lines or edges.

Many methods have been proposed to minimize this energy function[1,6,7,12,16]. Liou and Chang[7] proposed an approach that is more suitable for evolving mesh. Discretizing equation (1) and ignoring E_{con} for simplicity, the energy function can be written as:

$$E_{snake} = \sum_{i=1}^{n} \mu_i |v_i - v_{i-1}|^2 + \nu_i |v_{i-1} - 2v_i + v_{i+1}|^2 + E_{image}(v_i). \quad (2)$$

Where, $E_{image}(v_i) = -\gamma_i |\nabla I(v_i)|^2$ denotes the negated intensity gradient at point v_i. They use a two-dimensional Hopfield neural network to minimize equation (2). The network consists of $n * m$ mutually interconnected neurons where n is the number of points of the snake and m is the number of neighbors around each point of the snake. The energy function is computed at v_i and each of its neighbors, as shown in Fig. 1. The position which minimizes the energy function is chosen as the new position of v_i. Let $v_{i,p}$ denote the binary state of the (i,p)th neuron (1 for firing and 0 for resting). Let $x_{i,p}$ and $y_{i,p}$ be the x and y coordinates, respectively, of the neighboring point (i,p), and let $g_{i,p}$ be the negated intensity gradient at the neighboring point (i,p) defined as $g_{i,p} = -\gamma_i |\nabla I(x_{i,p}, y_{i,p})|^2$. The Hopfield type energy function for equation (2) is

$$\begin{aligned} E_{snake} = \\ \sum_{i=1}^{n} \{ & w_1 \alpha_{i,i-1} [(\sum_{p=1}^{m} x_{i,p} v_{i,p} - \sum_{p=1}^{m} x_{i-1,p} v_{i-1,p})^2 + \\ & (\sum_{p=1}^{m} y_{i,p} v_{i,p} - \sum_{p=1}^{m} y_{i-1,p} v_{i-1,p})^2] + w_2 \beta_{i-1,i+1} [\end{aligned}$$

$$
\begin{aligned}
&(\sum_{p=1}^{m} x_{i-1,p}v_{i-1,p} - 2\sum_{p=1}^{m} x_{i,p}v_{i,p} + \sum_{p=1}^{m} x_{i+1,p}v_{i+1,p})^2 \\
&+(\sum_{p=1}^{m} y_{i-1,p}v_{i-1,p} - 2\sum_{p=1}^{m} y_{i,p}v_{i,p} + \sum_{p=1}^{m} y_{i+1,p}v_{i+1,p})^2] \\
&+w_3\gamma_i(\sum_{p=1}^{m} g_{i,p}v_{i,p}) + w_4\varepsilon_i[(1 - \sum_{p=1}^{m} v_{i,p})^2] \\
&+w_5\tau_i \sum_{p=1}^{m} v_{i,p}(1 - v_{i,p})\}. \qquad (3)
\end{aligned}
$$

This energy function includes the fourth term to imply that only one point can be selected from the neighboring points (*i*,1), (*i*,2), ..., (*i*,m) for each *i*. The last term is called self-connection eliminating term, which will change the diagonal value of the interconnection matrix of the Hopfield network. The μ_i and ν_i in equation (2) are substituted by $\alpha_{i,i-1}$ and $\beta_{i-1,i+1}$ respectively. $\alpha_{i,j}$ and $\beta_{i,j}$ are defined as:

$$
\left\{ \begin{array}{l} \alpha_{i,j} = 1 - (g_{i,5} + g_{j,5})/10 \\ \beta_{i,j} = (15 + g_{i,5} + g_{j,5})/40 \end{array} \right\}.
$$

The choices of $\alpha_{i,j}$ and $\beta_{i,j}$ are not limited. Different choices produce different results. The value of $g_{i,j}$ are normalized to range from 0 to -255. The constants w_1 ,w_2 ,w_3 ,w_4 and w_5 are used to weight the five terms.

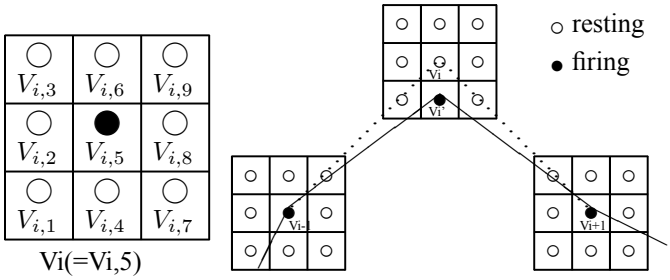

Fig. 1. Position of a Neuron

Each node in the snake has eight neighbors. Thus, m=9 in this case(including itself). The energy function is computed at v_i and each of its eight neighbors. The location $v_i^{'}$,which minimizes the energy function, is selected as the new position of v_i.

B. The Mesh

The neural mesh in [7] is composed of many snakes(Fig. 2(a)). The energy function of the mesh is:

$$
E_{mesh} = \sum_{s=1}^{S_n} E_{snake(s)}. \qquad (4)
$$

And, the Lyapunov function of a two-deimensional Hopfield network is written as:

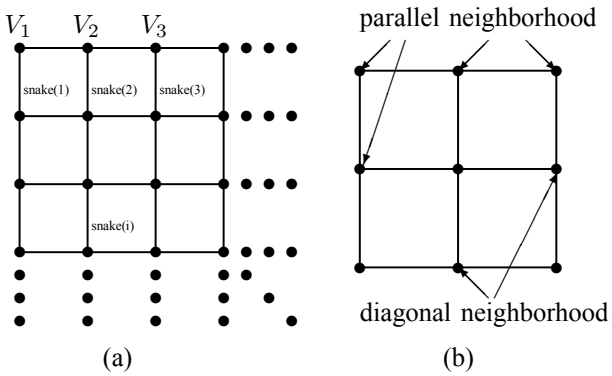

Fig. 2. Composition of Neural Mesh and their relation

(a): The mesh is composed of many snakes. Each snake has four nodes. (b): The snake has the tendency to shrink if there are no constraints imposed upon it. The internal nodes in the mesh are influenced by eight forces.

$$
\begin{aligned}
E_{Hopfield} &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{p=1}^{m}\sum_{j=1}^{N}\sum_{q=1}^{m} W_{i,p;j,q}v_{i,p}v_{j,q} \\
&\quad -\sum_{i=1}^{N}\sum_{p=1}^{m} I_{i,p}v_{i,p}. \qquad (5)
\end{aligned}
$$

Where, N is the number of nodes of the mesh. Rearranging equation (5) and comparing it with equation (3) we obtain:

$$
\begin{aligned}
W_{i,p;j,q} &= (T_1(i,j) + T_2(i,j) + T_3(i,j)) \\
&\quad *(x_{i,p}x_{j,q} + y_{i,p}y_{j,q}) + T_4(i,j). \qquad (6)
\end{aligned}
$$

Where,

$$
\begin{aligned}
T_1(i,j) &= (2 - \delta_{p,q})\delta_{i,j}[\sum_{k=1}^{N} -4\phi_2(i,k)(w_1\alpha_{i,k} + w_2\beta_{i,k}) \\
&\quad -8\sum_{l=1}^{N} w_2\beta_{k,l}\phi_1(i,k)\phi_1(i,l)\phi_2(k,l)] \\
T_2(i,j) &= 8w_1\alpha_{i,j}\phi_2(i,j) \\
T_3(i,j) &= 8w_2\{\sum_{k=1}^{N}[\beta_{i,k}\phi_1(j,k)\phi_2(i,k) + \\
&\quad \beta_{j,k}\phi_1(i,k)\phi_2(j,k)]\phi_1(i,j) - \beta_{i,j}\phi_2(i,j)\} \\
T_4(i,j) &= [\sum_{k=1}^{N}(-2w_4\varepsilon_i(2 - \delta_{p,q}) + 2w_5\tau_i\delta_{p,q})\phi_2(i,k)]\delta_{i,j} \\
I_{i,p} &= \sum_{k=1}^{N}[(-w_3\gamma_i g_{i,p} + 2w_4\varepsilon_i - w_5\tau_i\delta_{p,q})\phi_2(i,k)
\end{aligned}
$$

respectively. $\delta_{i,j}$ is the Kronecker delta function. $\phi_1(i,j)$ and $\phi_2(i,j)$ are the neighborhood functions defined as (Fig. 2(b))

$$\phi_1(i,j) = \left\{ \begin{array}{ll} 1, & \text{if i and j are parallel neighborhood} \\ 0, & \text{otherwise} \end{array} \right\}$$
$$\phi_2(i,j) = \left\{ \begin{array}{ll} 1, & \text{if i and j are diagonal neighborhood} \\ 0, & \text{otherwise} \end{array} \right\}.$$

The (i,p)th neuron in the network receives input weighted by $W_{i,p;j,q}$ from the (j,q)th neuron for each neighboring neuron, and a bias input $I_{i,p}$. The total input, $net_{i,p}$, of the (i,p)th neuron is computed as:

$$net_{i,p} = \sum_{j=1}^{N}\sum_{q=1}^{m} W_{i,p;j,q} v_{i,p} + I_{i,p}. \tag{7}$$

Then, the (i,p)th neuron is updated as follows:

$$v_{i,p} = \left\{ \begin{array}{ll} 1, & \text{if } net_{i,p} \geq 0 \\ 0, & \text{if } net_{i,p} < 0 \end{array} \right\}.$$

The rule described above for updating the neuron is applied in an asynchronous fashion. This means that for a given time, only a single neuron (which is selected randomly) is allowed to update its output.

III. Mesh on Cellular Neural Network

L. O. Chua and L. Yang [2] proposed a new circuit architecture, called cellular neural networks, which can perform parallel signal processing in *real* time. It shares the best features of neural network and cellular automata, and its local dynamic seems useful to realize mesh which also have constraint for each neuron over their neighbors. Rather than directly apply above works, we slightly modify the energy function of mesh (equation (3)), and make it more suitable for cellular neural network.

We implement $m * n$ points mesh using two parallel, size-equaled cellular neural networks, denoted as C_x and C_y, with $m*n$ neurons arranged in m rows and n columns. Let $c(i,j)$ denote the $(i,j)\,th$ neuron in a cellular neural network, and $c_{Nr}(i,j)$ denote the neighbors of $c(i,j)$. We use the four parallel neighbors of $c(i,j)$ as $c_{Nr}(i,j)$. By the dynamic characteristic of cellular neural network (ouputs are either -1 or 1), we redefine the energy function of the mesh as

$$\begin{aligned} E_{int_x} = & \sum_{c(i,j)\in C_x} \sum_{c(k,l)\in c_{Nr}(i,j)} \{ \\ & \alpha_{i,j}\left[\left(x_{i,j} + v_{x,i,j} - x_{k,l}\right)^2\right] + \\ & \sum_{\substack{c(p,q)\in c_{Nr}(i,j) \\ c(p,q)\notin c_{Nr}(k,l)}} \beta_{i,j}\left[\left(x_{k,l} - 2\left(x_{i,j} + v_{x,i,j}\right) + x_{p,q}\right)^2\right]\} \end{aligned}$$

$$E_{image_x} = \sum_{c(i,j)\in C_x} \gamma_{i,j} \left[\begin{array}{l} g_{-1,x,i,j}\left(1 - \left(v_{x,i,j}+1\right)^2\right) \\ +g_{+1,x,i,j}\left(1 - \left(v_{x,i,j}-1\right)^2\right) \end{array} \right]$$

$$E_{mesh_x} = E_{int_x} + E_{image}. \tag{8}$$

Where, $v_{x,i,j}$ is the output of $c(i,j) \in C_x$, $x_{i,j}$, y_{ij}(mentioned later) and $g_{a,x,i,j}$ are the x, y coordinates and negative intensity gradient, respectively, of the (i,j)th neuron in mesh. $g_{a,x,i,j}$ is defined as $g_{a,x,i,j} = -\gamma_{i,j}|\nabla I(x_{i,j}+a, y_{i,j})|^2$ and $g_{a,y,i,j}$(mentioned later) is defined as $g_{a,y,i,j} = -\gamma_{i,j}|\nabla I(x_{i,j}, y_{i,j}+a)|^2$. The energy function for C_y, E_{int_y} and E_{image_y} are defined similarly by substituting all x in E_{int_x} and E_{image_x} into y; in the following we only consider the case for C_x, because C_y is similar. The values of $v_{x,i,j}$ and $v_{y,i,j}$output of $c(i,j) \in C_y$) are used to update the new position of (i,j)th neuron in mesh as follows:

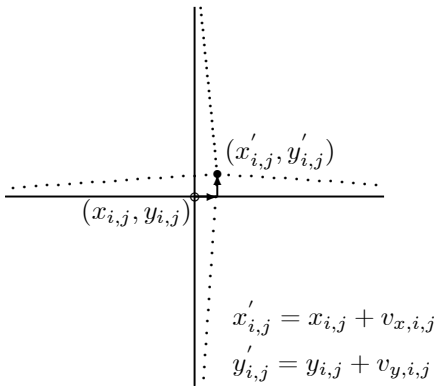

Fig. 3. The Movement of a Mesh Node.

The outputs of C_x and C_y serve as the movement of mesh nodes. Because energy functions of C_x and C_y are minimized separately, x_{ij} and y_{ij} are updated in turns, not synchronously.

$$\left\{ \begin{array}{l} x'_{i,j} = x_{i,j} + v_{x,i,j} \\ y'_{i,j} = y_{i,j} + v_{y,i,j} \end{array} \right\}.$$

One can think of $v_{x,i,j}$ and $v_{y,i,j}$ as the movement for (i,j)th neuron in the mesh. The Lyapunov function of a cellular neural network with $m*n$ neurons is:

$$\begin{aligned} E_{cnn} = & -\frac{1}{2}\sum_{(i,j)}\sum_{(k,l)} A(i,j;k,l)\, v_{i,j} v_{k,l} \\ & -\sum_{(i,j)}\sum_{(k,l)} B(i,j;k,l)\, v_{i,j} v_{u,k,l} \\ & +\frac{1}{2R_x}\sum_{(i,j)} v_{i,j}^2 - \sum_{(i,j)} I v_{i,j}. \end{aligned} \tag{9}$$

Where, R_x is an adjustable parameter, $v_{i,j}$ and $v_{u,i,j}$ is the output and input, respectively, of the (i,j)th neuron in the cellular neural network. Let I, $A(i,j,k,l)$ and $B(i,j,k,l)$ be zero for all $i,k < m$, $j,l < n$ where $i \neq k, j \neq l$ and let $v_{u,i,j}$ be 1 for all $i < m, j < n$, we can write the Lyapunov function in a more simple form:

$$\begin{aligned} E'_{cnn} = & -\frac{1}{2}\sum_{(i,j)} A(i,j;i,j)\, v_{i,j}^2 + \frac{1}{2R_x}\sum_{(i,j)} v_{i,j}^2 \\ & -\sum_{(i,j)} B(i,j;i,j)\, v_{i,j}. \end{aligned} \tag{10}$$

By rewriting equation (10) in this way, the Lyapunov function of C_x now becomes:

$$E_{C_x} = -\frac{1}{2}\sum_{(i,j)} A_x\left(i,j;i,j\right) v_{x,i,j}^2 + \frac{1}{2R_{xx}}\sum_{(i,j)} v_{x,i,j}^2 - \sum_{(i,j)} B_x\left(i,j;i,j\right) v_{x,i,j}. \quad (11)$$

Rearranging and comparing E_{mesh_x} to E_{C_x}, we have:

$$A_x(i,j;i,j) = \frac{1}{R_{xx}} - 8\alpha_{i,j} - 32\beta_{i,j} + 2\gamma_{i,j}\left(g_{-1,x,i,j} + g_{+1,x,i,j}\right) \quad (12)$$

$$B_x\left(i,j;i,j\right) = \sum_{c(k,l)\in c_{Nr}(i,j)} \alpha_{ij} 2\left(x_{k,l} - x_{i,j}\right) + \sum_{c(k,l)\in C_x}\sum_{c(p,q)\in C_x} 4\beta_{i,j}\left(x_{k,l} + x_{p,q} - 2x_{i,j}\right) * \phi_1\left(i,j;k,l\right)\phi_1\left(i,j;p,q\right)\phi_2\left(k,l;p,q\right) + 2\gamma_{i,j}\left(g_{-1,x,i,j} - g_{+1,x,i,j}\right). \quad (13)$$

Where, $\phi_1\left(i,j;k,l\right)$ and $\phi_2\left(i,j;k,l\right)$ are the neighborhood functions redefined as:

$$\phi_1\left(i,j;k,l\right) = \left\{ \begin{array}{ll} 1, & \text{if } c\left(i,j\right) \text{ and } c\left(k,l\right) \text{ are parallel} \\ & \text{neighborhood} \\ 0, & \text{otherwise} \end{array} \right\}$$

$$\phi_2\left(i,j;k,l\right) = \left\{ \begin{array}{ll} 1, & \text{if } c\left(i,j\right) \text{ and } c\left(k,l\right) \text{ are diagonal} \\ & \text{neighborhood} \\ 0, & \text{otherwise} \end{array} \right\}.$$

$A_y(i,j;i,j)$ and $B_y\left(i,j;i,j\right)$ for C_y are similar.

Finally, we can use C_x and C_y to obtain $v_{x,i,j}$ and $v_{y,i,j}$ for the (i,j)th neuron in mesh, respectively, by setting $A_x(i,j;i,j)$, $B_x\left(i,j;i,j\right)$, $A_y(i,j;i,j)$ and $B_y\left(i,j;i,j\right)$ properly[2].

The x and y coordinates can be updated in an asynchronous fashion, so one cellular neural network is sufficient for implementing the mesh. This means that for a given time, only the x or y coordinate of the neural mesh is updated to a new value but not both. The experimental results are showed below(Fig. 4 and 5).

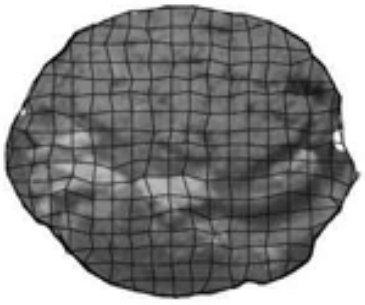

Figure 4. The Final Mesh.

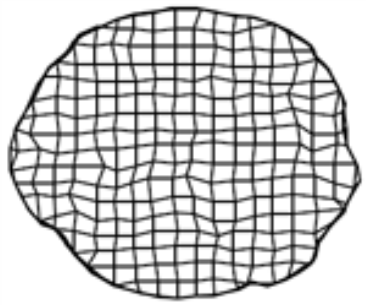

Figure 5. The coverged mesh only

IV. Conclusion

In this paper we implemented mesh on cellular neural networks. The method is to employ the cellular neural network to solve the energy-minimizing problem of the mesh. There are several ways to write the energy function of cellular neural network for the mesh, e.g., [7] uses 9 neurons for each mesh point. We choose an elegant one which is easy to implement on cellular neural networks which uses only one cell in one cellular neural network for each mesh point.

References

[1] A. A. Amini, S. Tehrani, and T. E. Weymouth, "Using Dynamic Programming for Minimizing the Energy of Active Contours in the Presence of Hard Constraints," Proc. IEEE Conf. Computer Vision, pp. 95-99, 1988

[2] L. O. Chua and L. Yang, "Cellular Neural Network: Theory," IEEE Trans. Circuit Systs., vol. 35, no. 10, pp. 1257-1272, Oct. 1988

[3] E. N. Gilbert and H. O. Pollak, "Steiner minimal trees," SIAM J. Allp. Math., no. 16, pp. 1-29, 1968

[4] J. J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," Biolog. Cybern., 52, pp. 141-152, 1985

[5] F. K. Hwang and D. S. Richards, "Steiner tree problems," Network, no. 22, pp. 55-89, 1992

[6] M. Kass, A. Witkin and D. Terzopoulos, "Snakes: Active Contour Models," Porc. of Int. Conf. on Computer Vision, pp. 259-268, 1987

[7] C. Y. Liou and Q. M. Chang, "Meshed Snakes," Proc. Int. Conf. Neural Network, vol.3, pp. 1516-1521, 1996

[8] C. Y. Liou and Q. M. Chang, "Numerical Soap Film for the Steiner Tree Problem," Proc. Int. Conf. Neural Information Processing, vol. 1, pp.642-647, Sep., 1996

[9] C. Y. Liou and Q. M. Chang, "Active Mesh for Minimal Surface Problems," Proc. Int. Conf. Neural Information Processing and Intelligent Information Systems, vol. 1, pp. 486-489, 1997

[10] C. Y. Liou and Q. M. Chang, "Neural Mesh for the Steiner Tree Problem," J. Information Science and Engineering, vol. 13, no, 2, pp. 335-354, 1997

[11] D. Terzopoulos and M. Vasilescu, "Sampling and Reconstruction with Adaptive Meshes," Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR'91), pp. 70-75, 1991

[12] C. T. Tsai, "Minimizing The Energy of Active Contour Model Using a Hopfield Network," IEEE Proceedings-E, vol. 140, no. 6, pp. 297-303, 1993

[13] M. Vasilescu and D. Terzopoulos, "Adaptive Meshes and Shells," Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR'92), pp.829-831, 1992

[14] Y. Wang and O. Lee, "Active Mesh - A Feature Seeking and Tracking Image Sequence Representation Scheme," IEEE Trans. on Image Procedding, vol. 3, no. 5, pp. 610-624, 1994

[15] I. Weiss, "Shape Reconstruction on a Varying Mesh," IEEE Trans. on Pattern Analysis and Machine Intelligence., vol. 12, no. 4, pp. 345-362, 1990

[16] D. J. Williams and M. Shah, "A Fast Algorithm for Active Contours and Curvature Estimation," CVGIV: Image Undersanding, vol. 55, no. 1, pp. 14-26, 1992