

Neural network algorithm for solving ray-tracing problem

Pavel V. Skribtsov

Science Center of Neurocomputers,
Novaya Basmannaya-20, Moscow, 107066, Russia
scn@mail.cnt.ru

1.1 Abstract

This work is dedicated to the study of neural network method for solving of ray-tracing task, which appears in 3D visualization algorithms. Physical representation of the task is the problem of finding the nearest point of the "vision" ray crossing with the surfaces of the scene. Application: Real time 3D visualization, rendering of the complex scenes, containing semi-transparent, reflecting, diffusive objects, soft shadows and volume light sources.

1.2 Introduction

To receive image of the 3D scene on the computer display, one needs to receive its 2D projection, provided that only visible parts of the objects are shown.



Fig 1

Figure 1 shows the physical definition of the ray-tracing task. In the virtual 3D space the point of vision is chosen (position and direction of the camera). Some distance apart the vision point the virtual projection screen is located. Since the image on the screen is defined by the pixels, the task of finding the projection of the 3D scene on the projection plane is the task of finding the colors for each pixel. For the sake of that, from the vision point a so called "vision ray" is emitted which is passing through the given pixel. This ray crosses some number of surfaces of the 3D objects. The task is to

find the nearest body, which the ray has "touched" and the coordinate of the intersection.

2. Mathematical definition of the task

2.1 Vision ray

– is the geometrical ray in the three-dimensional Euclidean space (x,y,z) , specified by the start point \mathbf{a} , and direction vector \mathbf{b} . Represents the set of points given by the formula

$$\vec{r}(l) = \vec{a} + \vec{b}l, \quad (1)$$

$$l \in [0, +\infty)$$

where l - is a parameter.

2.2 Scene

– Set of data describing the objects subjected for visualisation. Let the scene consists of K objects, which has their surfaces defined by the equations:

$$f_k(x,y,z) = 0, \quad (2)$$

$$k = 1..K$$

which is equivalent to:

$$f_k(\vec{r}) = 0, \quad \vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3)$$

$$k = 1..K$$

Let's note that normal vector to the surface of the object at the point $\vec{r}_0 : f_k(\vec{r}_0) = 0$ equals to the gradient of the surface function in the given point (provided that this function is differentiable)

$$\vec{n}(\vec{r}) = \text{grad}(f_k(\vec{r})), \quad (3.a)$$

$$\vec{r} \in \{f_k(\vec{r}) = 0\}$$

Next equation gives the production of all the surface functions.

$$\prod_{k=1}^K f_k(\vec{r}) = 0 \quad (3.1)$$

One can notice that

$$\prod_{k=1}^K f_k(\vec{r}_0) = 0 \Rightarrow \exists k_0, f_{k_0}(\vec{r}_0) = 0 \quad (3.2)$$

$$f_{k_0}(\vec{r}_0) = 0 \Rightarrow \prod_{k=1}^K f_k(\vec{r}_0) = 0 \quad (3.3)$$

$$f_k(\vec{r}_0) \neq 0, k = 1..K \Leftrightarrow \prod_{k=1}^K f_k(\vec{r}_0) \neq 0 \quad (3.4)$$

So we can define the **union of all surfaces**:

$$S(\vec{r}) = \prod_{k=1}^K f_k(\vec{r}) = 0 \quad (4)$$

2.4 Points of crossings of the vision ray with the union of all surfaces

To find them, let's solve the equation:

$$S(\vec{a} + \vec{b}l) = 0 \quad (8)$$

$$l \in [0, +\infty)$$

let ? solutions of this equation (if there are any) be defined as a set:

$$l = \{l_1..l_p\} \quad (9)$$

if there are no solutions, then ?=0 and the set (9) is empty.

2.5 Ray-tracing function

Let's define ray-tracing function:

$$\vec{F}(\vec{a}, \vec{b}) = \begin{pmatrix} \vec{r}_x \\ k \\ \vec{n} \end{pmatrix} \left\{ \begin{array}{l} S(\vec{a} + \vec{b}l) = 0, l = \min\{l_1..l_p\} | l > 0, \vec{r}_x = \vec{a} + \vec{b}l \\ f_k(\vec{r}_x) = 0 \\ \vec{n} = \text{grad}(f_k(\vec{r}_x)) \end{array} \right. \quad (10)$$

it is obviously defined in the points where equation (8) has solutions.

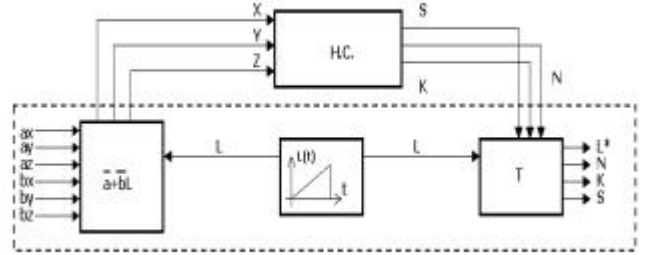
So, the ray-tracing task is the task of finding the values of the F function for all possible values of it's argument.

2.7 Functional scheme of neural network as a black-box for the ray-tracing algorithm

This scheme shows the work (usage) of the trained neural network.

The generator of the sawtooth signal (in the middle) transmits it's signals to the geometrical converter (from the left) and to the trigger (on the right), also the geometrical converter receives the parameters of the ray (vectors ? and b). Geometrical converter

outputs the points of the vision ray which passes to the neural network which gives the values of the S function, number of the object, nearest to (x,y,z) and normal function. (To keep this article short the problem of learning of such neural network is omitted, although author did some investigations showing that it is possible to train the multilayer neural network to respond in this way)



Denote	Meaning
L	parameter of the vision ray $\mathbf{r}=\mathbf{a}+\mathbf{b}l$
S	$S(x,y,z)$ - union of all surfaces
K	$k(x,y,z)$ - nearest object function, that gives the number of the object nearest to the point (x,y,z)
N	gradient of S

The outputs of the neural networks are transmitted to the trigger. In the beginning of the cycle trigger is reset. Then, unless trigger receives no zero signal on the S line, it gives all zeros at all output lines, which corresponds to "not finding" of the point of the vision ray crossing with any object. As soon as a trigger receives zero at the S input (suppose it happened when L was equal to L*) it transmits the values of the functions on its inputs in the moment $L=L^*$. The value of the trigger does not change until the end of cycle.






One can notice that the area indicated by the dash contour could be implemented in hardware as well as in software.

It's very easy to use the proposed scheme. External algorithm gives on the geometrical converter the parameters of the current ray. (arguments ?,b of the ray-tracing function). Then, we need to wait one cycle of the generator, if it outputs zeros, it means that this ray did not cross any object, otherwise, the \mathbf{r}_x value of the ray-tracing function is received by substituting the parameter L of the vision ray equation in to the ray equation $\mathbf{r}_x=\mathbf{a}+\mathbf{b}L$, number of the body we receive on the output K and the normal is received on the output N.

3. Intermediate results

This table shows some pictures rendered using the proposed approach.

Left column is the wireframe model of the scene subjected to rendering (with some learning point set displayed) and the right column is the result.

Wireframe model	Rendered image
	
	
	
	

4. Conclusion

- this work proves principal possibility of neural network ray tracing algorithm
- neural algorithm, being implemented in hardware will give significant improvement of the rendering time, which will enable using ray-tracing algorithm for real time 3D visualization environments.
- neural raytracing algorithm is simple enough, so it's hardware implementation will cost several times less compared with regular multi-processor ray-tracing solution.
- neural algorithm of ray-tracing is very sensible for the learning error. Even small errors, like of 10% level will cause negative visual effects visible at the examples above, which will make impossible to use this algorithm on practice. So, more investigation is needed to be done, mostly on how to improve the quality of learning of the neural network, and of course the speed of its learning.

5. References

- [1] An Introduction to Ray Tracing , Andrew Glassner ISBN -0-12-286160-4, 1997
- [2] Optics, G.S. Landsberg 1976
- [3] PC Magazine Online 2000
- [4] NN Theory, Galushkin A. I. Radiotekhnika, . Moscow 2000
- [5] Avedian E.D. Automatics and telemechanics 1999 N3 "Cascading neural networks"

Internet - resources (3D products manufacturers)

- [6] PowerAnimator, Maya <http://www.aw.sgi.com>
- [7] 3D Studio, AutoCAD <http://www.autodesk.com>
- [8] Pioneer, Truespace <http://www.caligari.com>
- [9] Painter, Ray Dream Studio, Poser, Bryce 3D <http://www.metacreations.com>
- [10] 3D Studio MAX <http://www.discreet.com>
- [11] Lightwave <http://www.newtek.com>
- [12] RenderMan <http://www.pixar.com>
- [13] NPGL, OpenGL <http://www.portable.com>
- [14] Rhino <http://www.mcneel.com>
- [15] Softimage 3D <http://www.softimage.com>
- [16] Advanced Render Technologies <http://www.art-render.com>

