# GPU-based MULTI-LAYER PERCEPTRON AS EFFICIENT METHOD FOR APPROXIMATION COMPLEX LIGHT MODELS IN PER-VERTEX LIGHTING

Konrad Pietras[1], Marek Rudnicki[2]

This paper describes a display method of the sky color on GeForce FX hardware. Lighting model used here is taken from “Display of the Earth taking into account atmospheric scattering” by Tomoyuki Nishita et.al., however this model is not the only suitable one in the proposed method.

A model of lighting used here is described by the following expression:

$$\mathbf{I} = \mathbf{I_0} \int_V \mathbf{b}\rho_v F(\beta)\exp(-\mathbf{b}(t(\rho_v, \alpha_v) + t(\rho_v, \theta_v)))dv$$

which describes a dependence of observed light color of position and orientation of observer.

$t(\rho,\alpha)$ - Optical length, amount of air along viewing direction.

$$t(\rho,\alpha) = \int_5^\infty \rho_5 ds$$

$F(\alpha)$ - Phase function, amount of light scattered in

$\alpha$ - angle between viewing direction $V$ and vertical direction $U$.

$\beta$ - angle between $V$ and sun direction $L$.

$\theta$ - angle between $U$ and $L$.

$\rho$ - relative density (surface = 1). Real density is contained in coefficient $\mathbf{b}$.

$$\rho = \exp\left(\frac{-h}{H_0}\right)$$

[1] None

[2] Institute of Computer Science, Technical University of Lodz, ul. Wolczanska 215, 93-005 Lodz, Poland

*h* – altitude of observer.

$$h = r - R_0$$

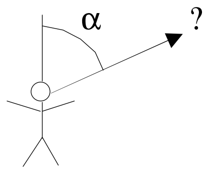
*r* – distance between observer and planet center.

$H_0$ - scale factor

$R_0$ - planet radius

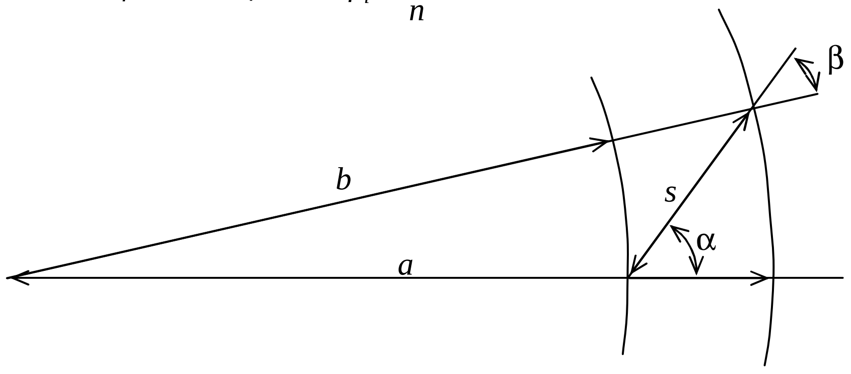
## I. Optical length

Attenuation of a light beam depends exponentially on optical length. In this implementation optical length is calculated only for acute angles between viewing direction and zenith.



$$t(\rho, \alpha) = \int_{5}^{\infty} \rho_5 ds \approx \sum_{i}^{n} \rho_i s_i$$

In order to improve speed of calculations of optical length, values of $\rho_i s_i$ are calculated, and used as lookup table $S[\rho,c]$. Because ρdepends exponentially on $h$, light-air interactions primarily depends on density, all calculations are made on ρ. This allows a constant interval in the above integral. Therefore function $S[\rho,c]$ is sampled on $\rho_i = \frac{i}{n}$.



A single sample $s_i$ is evaluated as follows:

$$s = \sqrt{a^2 - b^2 \sin(\alpha)^2} - b\cos(\alpha)$$

where:

$$a = R_0 - H_0 \ln(\rho_i) \text{ and } b = R_0 - H_0 \ln(\rho_{i+1})$$

Function $S[\rho,c]$ is nearly constant at small angles, but it grows rapidly, when $\alpha$ is close to right angle. In order to improve precision, and speed, the second dimension of $S[\rho,c]$ is sampled at constant intervals of $c = \cos(\alpha)$.

After lookup table $S[\rho,c]$ is calculated, optical length is sampled using the same intervals as with function $S$:

$$L[\rho,c] = \sum_{\rho_1=\rho}^{0} S[\rho_i, c_i]$$

While $\rho_i$ decreases during summation in interval-length steps, $c_i$ must be calculated as follows:

$$c_i = \cos(\beta) = \frac{\sqrt{a^2 - b^2(1-\cos(\alpha)^2)}}{a} = \frac{\sqrt{a^2 - b^2(1-c_{i-1}^2}}{a}$$

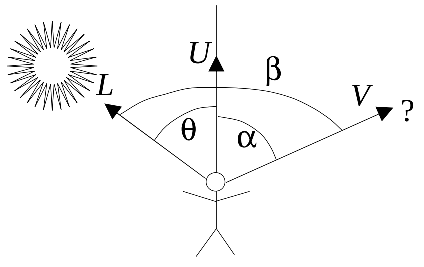
Lookup table $L[\rho,c]$ is used directly for angles, where $c \le 0$. For angles, where $c \in (0, -1]$, value of optical length may be obtained by the following property:

$$L[\rho,c] = 2L[\rho_x,0] - L[\rho,-c]$$

where:

$$\rho_x = \min\left\{\exp\left(\frac{R_0 - r\sin(\alpha)}{H_0}\right), 1\right\}$$

## II. Color of atmosphere



The color of atmosphere is calculated alike as linear mass of atmosphere. Increased complexity is caused by the fact that treated directions lie in three-dimensional space, summed samples depend on four parameters $(\alpha,\beta,\theta,\rho)$, which depend in different manner on integration step $v$.

Step $v$ moves along direction $V$. If four coordinates of point of observation are marked as $(\alpha_0, \beta_0, \theta_0,\rho_0)$, then a point moved a distance $v$ in direction $V$ has coordinates calculated as follows:

Let us mark a distance from planet center to point of observation as $r_0$

$$r_0 = R_0 - H_0 \ln(\rho_0)$$

Distance from planet center to point $v$ is:

$$r_v = \sqrt{r_0^2 + v^2 + 2 r_0 v \cos(\alpha_0)}$$

Density in point $v$ equals:

$$\rho_v = \exp\left(\frac{R_0 - r_v}{H_0}\right)$$

As point $v$ moves, direction $U$ changes due to planet curvature. Angle $\alpha$ changes accordingly:

$$\sin(\alpha_v) = \frac{r_0 \sin(\alpha_0)}{r_v}$$

$$\cos(\alpha_v) = \frac{\cos(\alpha_0)\sqrt{r_v^2 - v^2 \sin(\alpha)} + v \sin(\alpha)^2}{r_v}$$

Angle $\beta$ is constant because light beams are considered parallel.

Angle $\theta$ changes as follows:

$$\cos(\theta_v) = \cos(\alpha_v)\cos(\beta_0) + \sin(\alpha_v)\sin(\beta_0) + \cos(\Lambda)$$

$$\sin(\theta_v) = \sqrt{1 - \cos(\theta_v)^2}$$

Angle $\Lambda$ is measured between surface spanned by $U$-$V$ vertices and $L$-$V$ surface. During translation of point $v$ along direction $V$ only direction $U$ is changing, and only in surface $U$-$V$, so angle between surfaces $U$-$V$ and $L$-$V$ is constant.

$$\cos(\Lambda) = \frac{\cos(\theta_0) - \cos(\alpha_0)\cos(\beta_0)}{\sin(\alpha_0)\sin(\beta_0)}$$

Finally, color of atmosphere is equal:

$$\begin{aligned}\mathrm{I}(\alpha_0,\beta_0,\theta_0,\rho_0) = \sum_{\mathrm{v}} \mathbf{b}\mathrm{F}(\beta_0)\rho_v \exp(-\mathbf{b}(L[\rho_0,\cos(\alpha_0)] - L[\rho_v,\cos(\alpha_v)] + \\ + L[\rho_v,\cos(\theta_v)]))\Delta v\end{aligned}$$

Additionally, when $\cos(\theta_v) < 0$ and $\sin(\theta_v) r_v - R_0 < 0$, then point $v$ is in planet's shadow, and the sample is not summed.

$\mathbf{I}$ and $\mathbf{b}$ consist of three coordinates: red, green and blue. This gives three expressions used in application for obtaining the color of atmosphere:

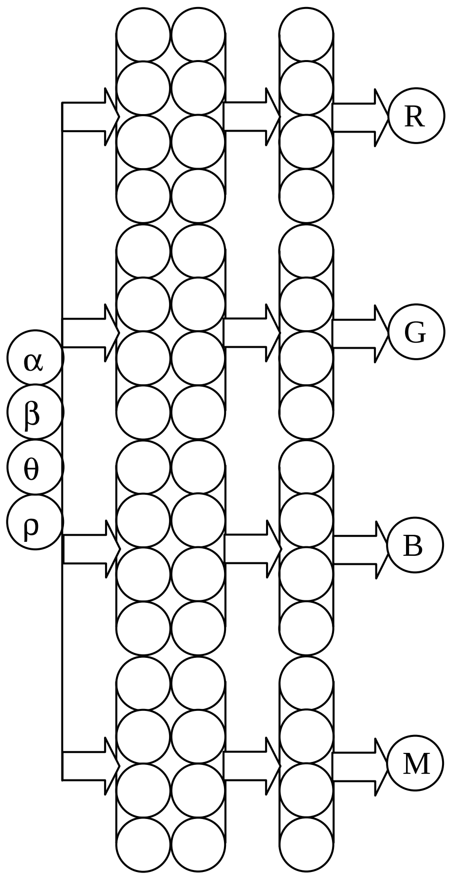
$$\begin{aligned}I_R(\alpha_0,\beta_0,\theta_0,\rho_0) &= \sum_v b_R F(\beta_0)\rho_v \exp(-b_R(L[\rho_0,\cos(\alpha_0)] - L[\rho_v,\cos(\alpha_v)] + \\ &\quad + L[\rho_v,\cos(\theta_v)]))\Delta v \\ I_G(\alpha_0,\beta_0,\theta_0,\rho_0) &= \sum_v b_G F(\beta_0)\rho_v \exp(-b_G(L[\rho_0,\cos(\alpha_0)] - L[\rho_v,\cos(\alpha_v)] + \\ &\quad + L[\rho_v,\cos(\theta_v)]))\Delta v \\ I_B(\alpha_0,\beta_0,\theta_0,\rho_0) &= \sum_v b_B F(\beta_0)\rho_v \exp(-b_B(L[\rho_0,\cos(\alpha_0)] - L[\rho_v,\cos(\alpha_v)] + \\ &\quad + L[\rho_v,\cos(\theta_v)]))\Delta v\end{aligned}$$

## III. Approximation

In previous points implementation of lighting model was described.

The color of atmosphere $\mathrm{I} = [I_R, I_G, I_B]$ is a function dependent of four parameters. Its complexity causes impossibility of using it directly in real-time working applications. Dobashi, Yamamoto and Nishita in “Interactive Rendering of Atmospheric Scattering Effects Using Graphic Hardware” proposed using many sampling layers, and exploiting blending for integration.

In order to avoid rendering multiple layers, the light model was approximated. Additionally approximation is performed directly on GPU’s vertex processor.



Precisely, function of color of atmosphere **I** is approximated by function N($\alpha_0,\beta_0,\theta_0,\rho_0$), which is a multi-layer percepron.

Neural network used here has a simple topology. It consists of four independent and identical perceptrons, each one for calculating a single output.

Each sub-network shares the same input, which is four-dimensional vector ($\alpha_0,\beta_0,\theta_0,\rho_0$).

Single sub-network consists of two hidden layers, with eight neurons in first layer, and four in second. Whole network uses 324 weights. Each neuron is biased, and has a following activation function:

$$f(x) = \frac{1}{1+2^{2x}}$$

The form of this approximator is deeply influenced by graphic processor architecture, and exploits its speed in calculating dot products.

Network is learned of the light model by separate program, and after learning process, it is used in GPU for coloring the sky and surface.

Three first outputs are learned of scaled color values:

$$R = \frac{I_r}{N} \qquad G = \frac{I_g}{N} \qquad B = \frac{I_b}{N},$$

where $N = \max\{I_r, I_g, I_b\}$. Fourth output is learned of value:

$$M = \frac{0.9 \times N}{\max\{N\}} + 0.05,$$

where $\max\{N\}$ is calculated from 100 thousands of random samples ($\alpha_0,\beta_0,\theta_0,\rho_0$).

This "normalization" of color values, and separate calculation by independent sub-networks was necessary to achieve low-error results. As inputs, values of $(\cos(\alpha_0),\cos(\beta_0),\cos(\theta_0),\rho_0)$ from ranges $([-1,1],[-1,1],[-1,1],[0,1])$ are used. During learning process, these values are taken in a random fashion with uniform distribution.

In case of night, when $N = 0$, network is learned of value:

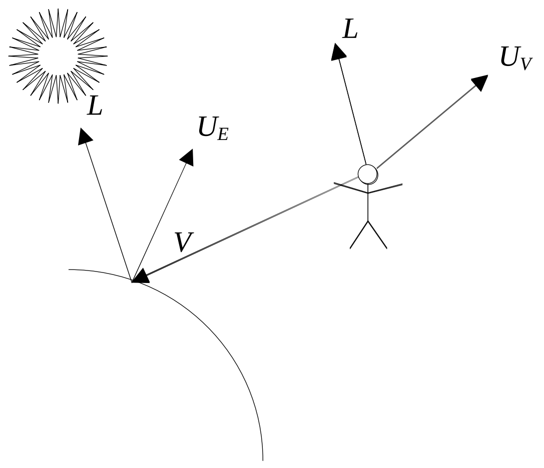
$$M = 0$$

This case caused multiplication by 0.9, and addition of 0.05 for coefficient $\mathrm{M}$, for better recognition between night and day.

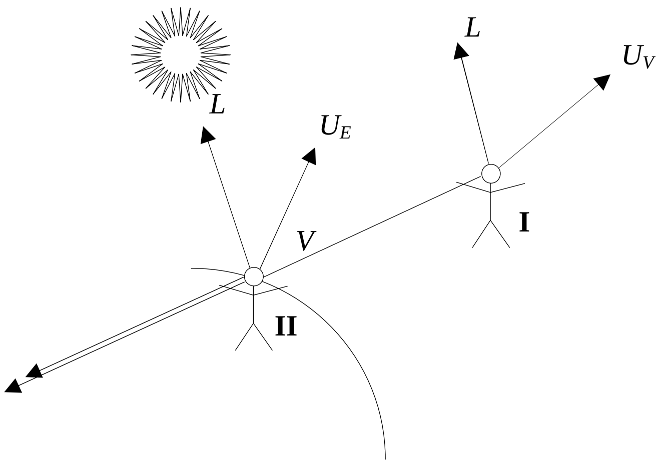
In order to reduce error, and to exploit extrapolation property of perceptron, in case of night R,G and B outputs are not learned.

## IV. Color of surface

Method presented above, gives only the color of atmosphere, when observed from inside into space. It is necessary to modify this method, in order to obtain surface color, or color of atmosphere between observer and surface.



This situation is presented on picture below. Direction *L* is constant, due to infinite light source. An incoming light beam to observer looking into direction *V* may be divided into two parts: Light reflected from surface, and light scattered in atmosphere along direction *V*.



Scattered light part is obtained from approximator by using it twice.

Observer **I** sees the same atmosphere as observer **II**, plus light scattered between both observers. Thus color of the light scattered between observer and surface can be obtained as a difference between neural network's output for observer, and output received for probe positioned on watched point on surface.

Reflected light part is obtained by using the model of atmosphere directly:

$$\begin{aligned}\mathbf{I} = \mathbf{b}CF(\alpha_{II},\beta_{II},\theta_{II})\rho_{II} \exp(-\mathbf{b}(L[\rho_I,\cos(\alpha_I)] - L[\rho_{II},\cos(\alpha_{II})] + \\ + L[\rho_{II},\cos(\theta_{II})]))\end{aligned},$$

where angles and densities are taken for observers **I** and **II**, *C* means color of surface with no atmosphere in point **II**, function *F* is in most general case a BDRF for surface in point **II.** This calculation is performed on GPU's fragment processor.

## V. Multiple atmospheres

Atmosphere of earth is a multi-component atmosphere. It consists of gaseous part, and aerosols. Each of components scatters incident light differently, and thus must be treated separately. All above mechanisms are described for single atmosphere. In order to obtain proper color of atmosphere for multi-component atmospheres, a small addition is needed.

This addition is correct, if a single component differs only by phase function *F*, coefficient **b**, and scaling factor $H_0$.

Because all the above calculations are based on densities, rather than on heights, it is necessary to scale all densities to density of the highest atmospheric component:

$$\rho_2 = \rho_1^{\frac{H_2}{H_1}}$$

The light model becomes:

$$\mathbf{I} = \sum_{\nu} (\mathbf{b}_1 F_1 \rho_1 + \mathbf{b}_2 F_2 \rho_2) \exp(-\mathbf{b}_1(...) - \mathbf{b}_2(...)) \Delta\nu$$

By this method, model can be expanded to any number of components.

**VI. Neural network on GPU**

This implementation of perceptron depends on NV_vertex_program2_option extension. It uses both address registers for weights access, and branching and subroutines to make program fit into 256 instruction limit. It uses 9 temporaries and 86 local parameters, and only first vertex attribute. Weights used for matrix multiplication are separated from weights used as bias. One address register is used for access to matrix multiplication weights and it moves by 4 during address addition, while the other is used for access to bias weights and it moves by 1.

There are two versions of vertex program used, one for observer, and one for probe positioned on watched point on surface. They differ by method of $U$ vector calculation.

```
!!ARBvp1.0
OPTION NV_vertex_program2;

ATTRIB iPos = vertex.attrib[0];

PARAM mvinv[4] = {
state.matrix.modelview.invtrans };
PARAM mv[4] = {
state.matrix.modelview };
PARAM mvp[4] = { state.matrix.mvp
};
PARAM lightDir =
state.light[0].position;
PARAM const1 = { 1.0, -2.0, 10.0,
0.0 };
PARAM a1const = { 0.0, 0.0, 0.0,
4.0 };
PARAM a2const = { 0.0, 0.0, 0.0,
1.0 };
PARAM xxx = { 1.1111, 0.2, 0.05, -
0.05 };
PARAM nn[83] = {
program.local[4..86] };
PARAM nndat = program.local[0];
PARAM pdat = program.local[1];
PARAM pdot = program.local[2];
TEMP t1, t2, t3, t4, t5, t6, t7, t8, t9,
t10;

ADDRESS addr1, addr2;

OUTPUT oPos = result.position;

ALIAS up = t3;
ALIAS lPos = t4;
ALIAS eye = t5;

#nndat.y is the number of weights
used
#in matrix multiplication.
MOV t1, a2const;
ADD t1.y, t1, nndat.y;
ARL addr1, a1const;
ARL addr2, t1;

#vector U:
MOV t1, mvinv[3];
DP3 up.x, mvinv[0], t1;
DP3 up.y, mvinv[1], t1;
DP3 up.z, mvinv[2], t1;
DP3 t1.w, up, up;
RSQ t1.w, t1.w;
MUL up.xyz, t1.w, up;

#vector L
DP3 lPos.x, mvinv[0], lightDir;
DP3 lPos.y, mvinv[1], lightDir;
DP3 lPos.z, mvinv[2], lightDir;
DP3 t1.w, lPos, lPos;
RSQ t1.w, t1.w;
MUL lPos.xyz, t1.w, lPos;

#this is used for drawing
atmosphere layer
#when planet sphere is scaled by
small factor
MOV t2, iPos;
MUL t2.xyz, t2, pdat.x;
MOV t2.w, const1.x;

#vector V
DPH eye.x, t2, mv[0];
DPH eye.y, t2, mv[1];
DPH eye.z, t2, mv[2];
DP3 t1.w, eye, eye;
RSQ t1.w, t1.w;
MUL eye.xyz, t1.w, eye;

DP4 oPos.x, t2, mvp[0];
DP4 oPos.y, t2, mvp[1];
DP4 oPos.z, t2, mvp[2];
DP4 oPos.w, t2, mvp[3];

ALIAS nnIn = t7;
#this is calculation of input of neural
network
DP3 nnIn.x, eye, up;
DP3 nnIn.y, eye, lPos;
DP3 nnIn.z, lPos, up;
MOV nnIn.w, pdat.y;
MUL nnIn, nnIn, pdot;

ALIAS arg = t5;
ALIAS res = t6;
ALIAS arg2 = t8;
ALIAS res2 = t9;

#Neural network:
CAL nnSubNet;
MOV t1, res;
CAL nnSubNet;
MOV t2, res;
CAL nnSubNet;
MOV t3, res;
CAL nnSubNet;
MOV t4, res;
DP4 res.x, nn[addr1.y + 0], t1;
DP4 res.y, nn[addr1.y + 1], t2;
DP4 res.z, nn[addr1.y + 2], t3;
DP4 res.w, nn[addr1.y + 3], t4;
#ARA addr1.xy, addr1;
CAL nnBiasFunc (TR);

#final transformation
SGE arg, res, xxx.z;
MUL arg, arg, xxx.x;
ADD res, res, xxx.w;
MUL res, res, arg;
MUL res.w, res.w, nndat.x;
MUL arg.xyz, res, res.w;
MUL arg.xyz, arg, const1.z;
MOV arg.w, const1.x;
MOV result.color.front, arg;

BRA endProg (TR);
```

```
#a single subnetwork
nnSubNet:
MOV arg, nnIn;
CAL nnLayer4_8 (TR);
MOV arg, res;
MOV arg2, res2;
CAL nnLayer8_4 (TR);
RET;

#transformation from 4-dim nn-layer
to 4-dim nn-layer (unused)
nnLayer4_4:
DP4 res.x, nn[addr1.y + 0], arg;
DP4 res.y, nn[addr1.y + 1], arg;
DP4 res.z, nn[addr1.y + 2], arg;
DP4 res.w, nn[addr1.y + 3], arg;
ARA addr1.xy, addr1;
CAL nnBiasFunc (TR);
RET;

#transformation from 4-dim nn-layer
to 8-dim nn-layer
# arg => res, res2
nnLayer4_8:
DP4 res.x, nn[addr1.y + 0], arg;
DP4 res.y, nn[addr1.y + 1], arg;
DP4 res.z, nn[addr1.y + 2], arg;
DP4 res.w, nn[addr1.y + 3], arg;
ARA addr1.xy, addr1;
DP4 res2.x, nn[addr1.y + 0], arg;
DP4 res2.y, nn[addr1.y + 1], arg;
DP4 res2.z, nn[addr1.y + 2], arg;
DP4 res2.w, nn[addr1.y + 3], arg;
ARA addr1.xy, addr1;
CAL nnBiasFunc (TR);
MOV arg2, res;
MOV res, res2;
CAL nnBiasFunc (TR);
MOV res2, res;
MOV res, arg2;
RET;

#transformation from 8-dim nn-layer
to 4-dim nn-layer
# arg, arg2 => res
nnLayer8_4:
DP4 res.x, nn[addr1.y + 0], arg;
DP4 res2.x, nn[addr1.y + 1], arg2;
DP4 res.y, nn[addr1.y + 2], arg;
DP4 res2.y, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
DP4 res.z, nn[addr1.y + 0], arg;
DP4 res2.z, nn[addr1.y + 1], arg2;
DP4 res.w, nn[addr1.y + 2], arg;
DP4 res2.w, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
ADD res, res, res2;
CAL nnBiasFunc (TR);
RET;

#transformation from 8-dim nn-layer
to
# 8-dim nn-layer (10th temporary is
needed)
# arg, arg2 => res, res2
nnLayer8_8:
DP4 res.x, nn[addr1.y + 0], arg;
DP4 res2.x, nn[addr1.y + 1], arg2;
DP4 res.y, nn[addr1.y + 2], arg;
DP4 res2.y, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
DP4 res.z, nn[addr1.y + 0], arg;
DP4 res2.z, nn[addr1.y + 1], arg2;
DP4 res.w, nn[addr1.y + 2], arg;
DP4 res2.w, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
ADD res, res, res2;
MOV t10, res;
DP4 res.x, nn[addr1.y + 0], arg;
DP4 res2.x, nn[addr1.y + 1], arg2;
DP4 res.y, nn[addr1.y + 2], arg;
DP4 res2.y, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
DP4 res.z, nn[addr1.y + 0], arg;
DP4 res2.z, nn[addr1.y + 1], arg2;
DP4 res.w, nn[addr1.y + 2], arg;
DP4 res2.w, nn[addr1.y + 3], arg2;
ARA addr1.xy, addr1;
ADD res, res, res2;
MOV arg, res;
MOV res, t10;
CAL nnBiasFunc (TR);
```

```
MOV t10, res;
MOV res, arg;
CAL nnBiasFunc (TR);
MOV res2, res;
MOV res, t10;
RET;

#Bias addition and perceptron
activation function.
nnBiasFunc:
ADD res, nn[addr2.y + 0], res;
MUL res, res, const1.y;
EX2 res.x, res.x;
EX2 res.y, res.y;
EX2 res.z, res.z;
EX2 res.w, res.w;
ADD res, res, const1.xxxx;
RCP res.x, res.x;
RCP res.y, res.y;
RCP res.z, res.z;
RCP res.w, res.w;
ARA addr2.xy, addr2;
RET;

endProg:
END
```

**VII. Results**

Images below depict atmosphere and planet form different heights. This atmosphere consists of two components, one with following parameters:

$$F = \frac{3}{16\pi}(1+\cos^2(\alpha))$$
$$H_0 = 8.3$$

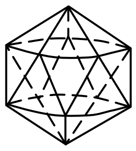
$$\mathbf{b} = \begin{bmatrix} \left(\frac{475}{700}\right)^4 \\ \left(\frac{546}{700}\right)^4 \\ \left(\frac{700}{700}\right)^4 \end{bmatrix} / r$$

and second:

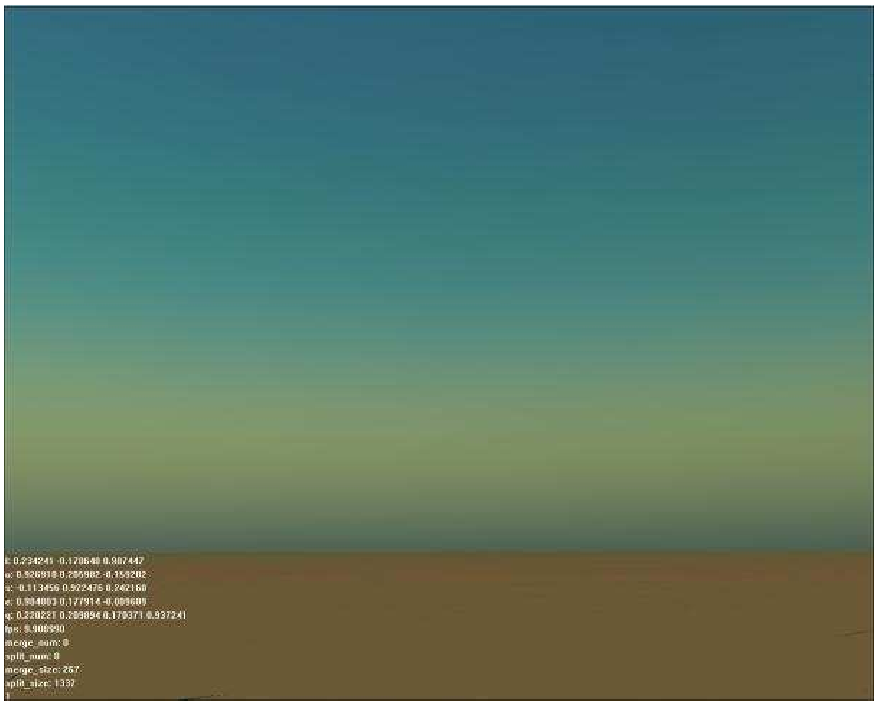
$$F = \frac{1-g^2}{16\pi(1+g^2-2g\cos(\alpha))^{3/2}}$$
$$H_0 = 1.6$$

$$\mathbf{b} = \begin{bmatrix} \left(\frac{475}{700}\right)^4 \\ \left(\frac{475}{700}\right)^4 \\ \left(\frac{475}{700}\right)^4 \end{bmatrix} / m$$

Color of surface is constant ($RGB = [0.5, 0.5, 0.5]$), and any variation from this value is due to atmosphere. Surface and atmosphere is a triangular mesh, and aplication uses a slightly modified CLOD algoritm described by Henri Hakl in “Diamond Algorithm” Mesh is based on sides of icosahedron (on left).

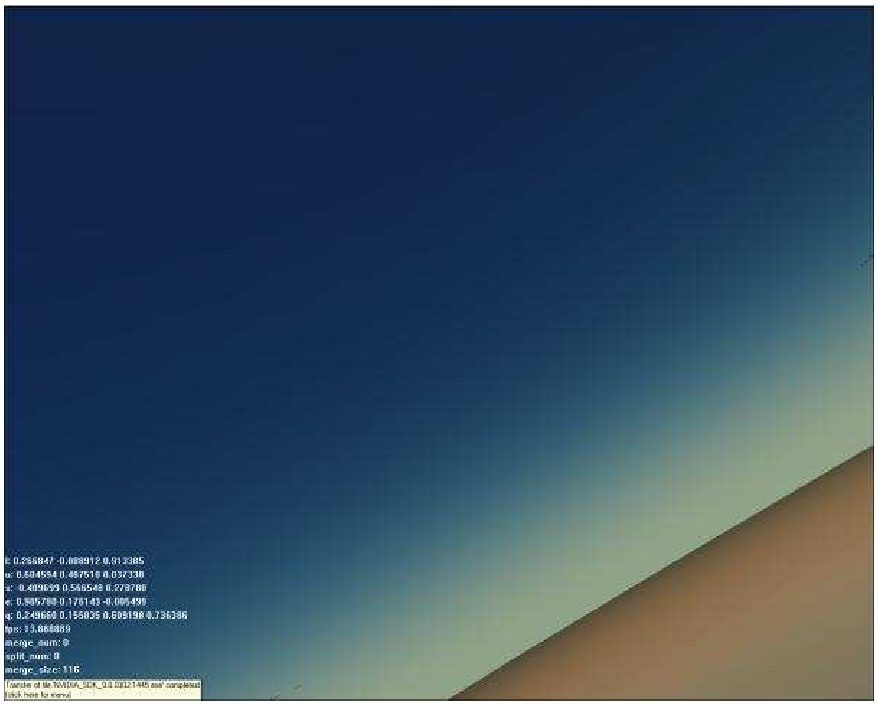


On Athlon 1700+, GeForce FX 5500, in 1280x1024x32 resolution, when surface mesh consists of 2000 triangles, the program runs with speed of 9 frames per second.



```
t: 0.234241 -0.170640 0.987447
o: 0.926910 0.209982 -0.159202
s: -0.113456 0.922476 0.242160
e: 0.984003 0.177914 -0.009609
q: 0.220221 0.209894 0.170371 0.937241
fps: 9.908990
merge_num: 0
split_num: 0
merge_size: 267
split_size: 1337
```

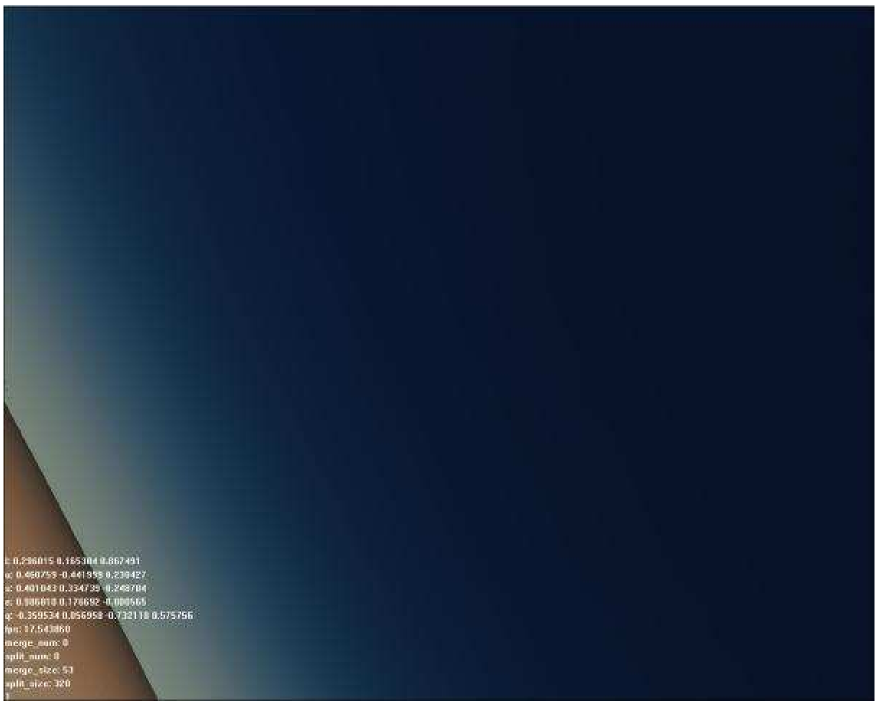
**Sea level (r=10000, m=10000)**



```
l: 0.266847 -0.088912 0.913385
u: 0.604594 0.487510 0.037338
s: -0.409699 0.566548 0.278780
e: 0.985780 0.176143 -0.005499
q: 0.249660 0.155835 0.609198 0.736386
fps: 13.888889
merge_num: 0
split_num: 0
merge_size: 116
```

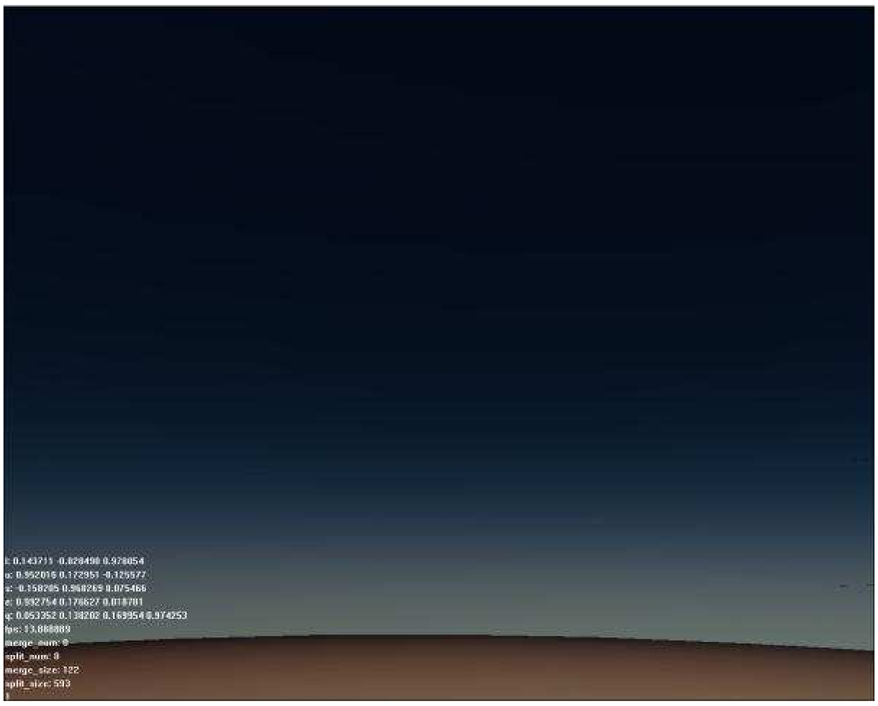
Transfer of file 'NVIDIA_SDK_9.0.0002.1445.exe' completed
(click here for menu)

**A bit higher (r=10000, m=10000)**



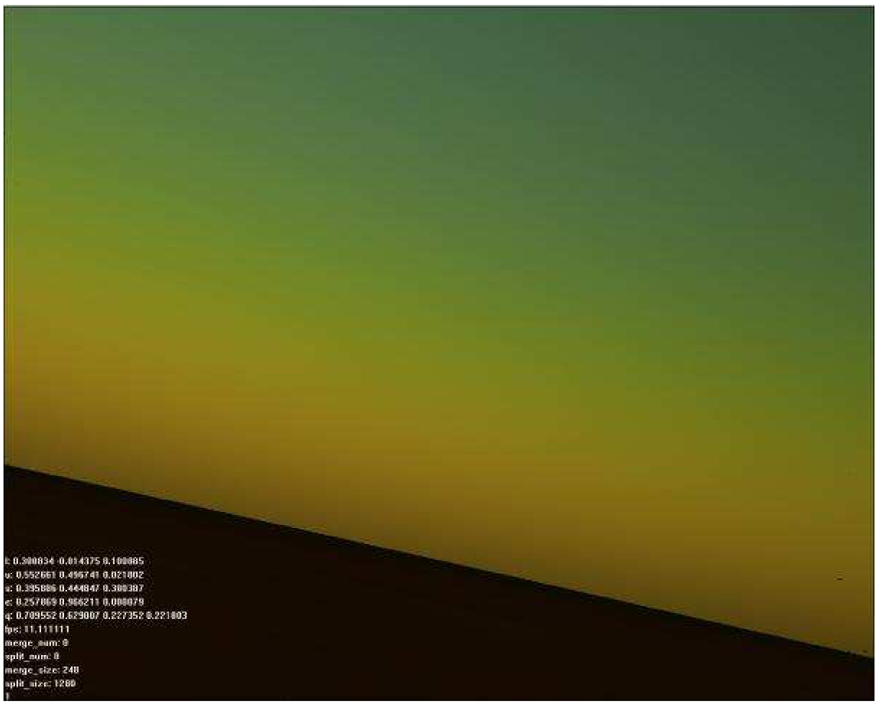
```
t: 0.296015 0.165304 0.867491
u: 0.460759 -0.441959 0.230427
s: 0.401043 0.334735 -0.248704
e: 0.986018 0.176692 -0.000565
q: -0.359534 0.056958 -0.732118 0.575756
fps: 17.543860
merge_num: 0
split_num: 0
merge_size: 53
split_size: 320
```

**...higher (r=10000, m=10000)**



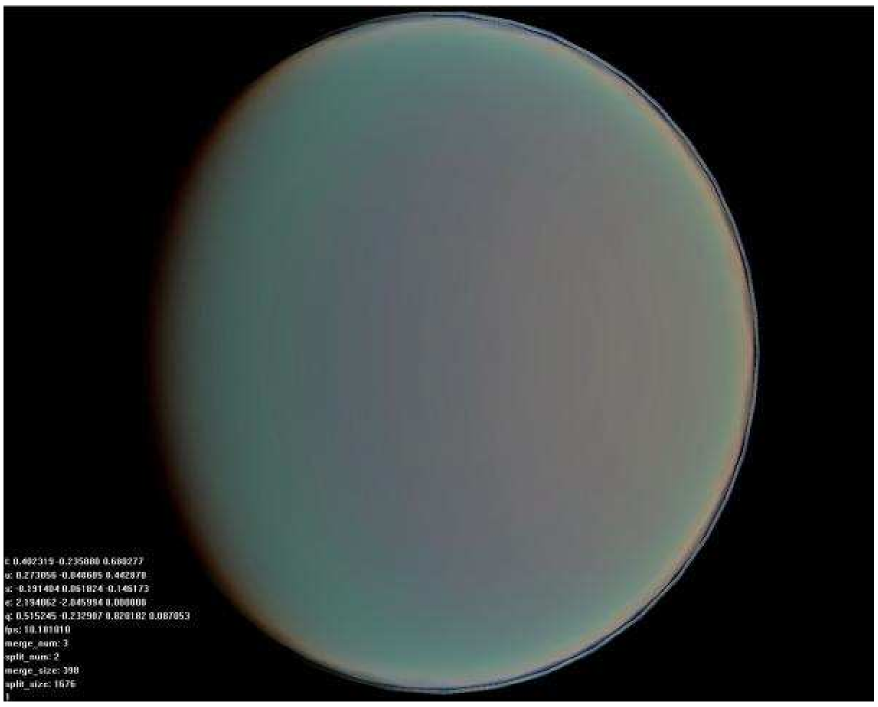
```
l: 0.143711 -0.028490 0.978054
u: 0.952016 0.172951 -0.125577
s: -0.158205 0.968269 0.075466
e: 0.992754 0.176627 0.018701
q: 0.053352 0.138202 0.169954 0.974253
fps: 13.888889
merge_num: 0
split_num: 0
merge_size: 122
split_size: 593
```

**..and higher (r=10000, m=10000)**



```
l: 0.300834 -0.014375 0.100085
u: 0.552661 0.496741 0.021802
s: 0.395886 0.444847 0.300387
e: 0.257069 0.966211 0.000079
q: 0.709552 0.629007 0.227352 0.221803
fps: 11.111111
merge_num: 0
split_num: 0
merge_size: 240
split_size: 1280
```

**Sun is low. Observer at sea level. (r=10000, m=10000)**

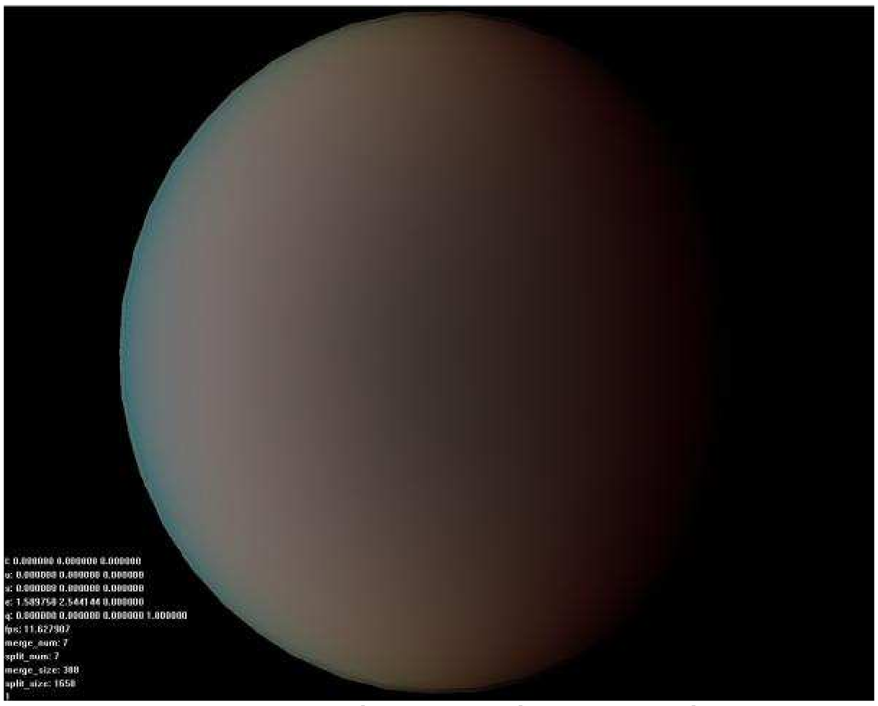


```
c: 0.402319 -0.235880 0.680277
u: 0.273056 -0.840605 0.442870
s: -0.191404 0.061824 -0.146173
e: 2.194062 -2.045994 0.000000
q: 0.515245 -0.232907 0.820182 0.087053
fps: 10.101010
merge_num: 3
split_num: 2
merge_size: 398
split_size: 1676
```

**Very high. (r=10000, m=10000)**

```
l: 0.000000 0.000000 0.000000
u: 0.000000 0.000000 0.000000
s: 0.000000 0.000000 0.000000
e: 1.000500 0.000000 0.000000
q: 0.000000 0.000000 0.000000 1.000000
fps: 5.882353
merge_num: 431
split_num: 418
merge_size: 292
split_size: 1367
```

**Dense atmosphere, sea level (r=100, m=100)**

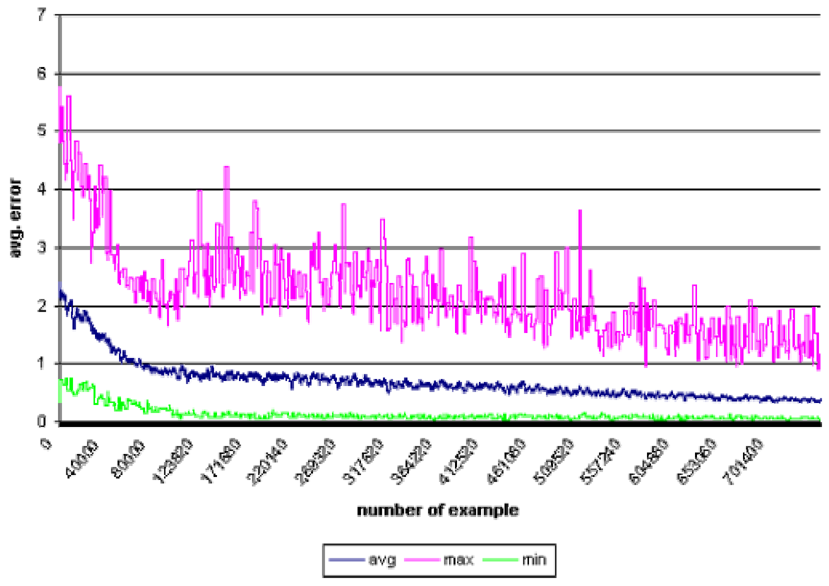


```
c: 0.000000 0.000000 0.000000
u: 0.000000 0.000000 0.000000
s: 0.000000 0.000000 0.000000
e: 1.589758 2.544144 0.000000
q: 0.000000 0.000000 0.000000 1.000000
fps: 11.627907
merge_num: 7
split_num: 7
merge_size: 308
split_size: 1658
```

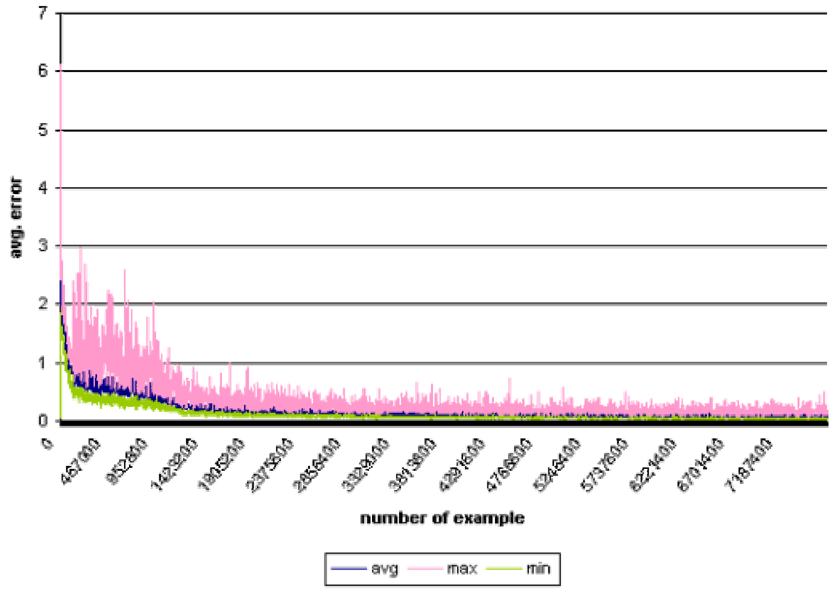
**Dense atmosphere, space (r=100, m=100)**

Neural network used here during learning uses 800.000 random samples of function I.

It is presented below how error was changing during learning in case of 800.000 samples, and in case of 8.000.000 samples. As this charts show, in first case, the network is not fully learned, but this case is much faster, and it yields good results.



**Fewer samples**



**Less samples**

## VIII. References

[1] Nishita T., Shirai T., Tadamura K., Nakamae E.: *Display of The Earth Taking into account Atmospheric Scattering.* Proc. SIGGRAPH'93*,* 175-182, 1993.

[2] Dobashi Y., Yamamoto T., Nishita T.: *Interactive rendering of atmospheric scattering effects using graphics hardware*. Proceedings of the conference on Graphics hardware*,* 2002.

[3] Hakl H.: *http://www.cs.sun.ac.za/~henri/terrain.html*

[4] Nishita T., Dobashi Y., Kaneda K., Yamashita H.: *Display Method of the Sky Color Taking into Account Multiple Scattering*, Proc. Pacific Graphics'96, 117-132, 1996.