Ministry of Education and Science of Ukraine

ODESA NATIONAL UNIVERSITY OF TECHNOLOGY

International Competition of
Student Scientific Works

BLACK SEA SCIENCE 2023

PROCEEDINGS

ODESA, ONUT 2023

Ministry of Education and Science of Ukraine

Odesa National University of Technology

International Competition of Student Scientific Works

BLACK SEA SCIENCE 2023

Proceedings

Odesa, ONUT
2023

DEVELOPMENT OF MEANS FOR HIGH PRODUCTIVE RENDERING

Author: Yevgen Stanislavenko
Advisor: Oksana Romaniuk
Vinnytsia National Technical University (Ukraine)

Abstract. *The analysis of the most common bidirectional functions of surface reflectance and methods of shading in realistic computer graphics has been conducted. The property of constancy of the increase of color intensity along the horizontal (vertical) rasterization lines of the triangle has been proved. Methods of parallelization of the computational process, new models of surface reflectivity and modification of the Gouraud method have been proposed. Shader programs based on the proposed methods have been developed and integrated into a professional graphics pipeline. A new device structure for determining the intensity of color is proposed.*

Keywords: *Gouraud shading, Phong shading, Blinn model, rendering, bidirectional reflectance distribution function, normal vector, surface reflectance, shader.*

I. INTRODUCTION

Relevance of the research topic. The role of computer graphics [1-10], as one of the main providing subsystems of computer technology is constantly growing, since it allows in the conditions of the modern level of development of computer technology to implement the most acceptable and familiar to the user technology of presenting information [1-3].

When synthesizing graphic scenes, it is necessary to solve a double problem - ensuring high realism of graphic objects and achieving an acceptable time for a specific task to form graphic scenes.

Three-dimensional images are realistic if they reproduce the most significant aspects of a process or phenomenon, convey the constructive, visual and visual features of objects. For the realistic formation of graphic images, it is necessary to adequately reproduce the shape and color of surfaces.

Today, the performance of graphic tools does not fully meet the needs of various branches of three-dimensional graphics.

In this regard, the urgent task is to increase the realism and productivity of the formation of images of three-dimensional scenes by developing new methods and means that would simplify the visualization procedures both at the software and hardware levels.

Scientific novelty of the results

1. The property of the constancy of the increase in color intensity along the horizontal (vertical) rows of rasterization of the triangle has been proved, which made it possible to increase the productivity of shading due to the fact that the increase in color intensity is calculated for the triangle, and not for each of its rasterization lines. This made it possible on average to reduce the shading time of the average triangle by an average of 1.7 times and simplify the hardware implementation.

2. Based on the proven property of the constancy of color intensity increase, methods of parallelization of the computational process are proposed, which provide an increase in shading performance with a balanced load of renders.

3. New models of surface reflectivity are proposed, which differ from the existing ones by the absence of an elevation operation to the power, and in relation to the well-known Schlick BRDF – the absence of division operation, the presence of only basic addition and multiplication operations, which makes it possible to develop devices with a relatively simple hardware implementation. The proposed BRDF have a second and third degree.

3. A modification of the Gouraud method is proposed, which consists in determining the maximum values of color intensities on the edges of a triangle, followed by the division of the triangle into components, which makes it possible to reproduce the specular highlights that cross the edges of the triangle and, as a result, increase the realism of the formation of the graphic scene.

The practical value of the work:

1. Shader applications have been developed and integrated into a professional graphic pipeline. Experimental evaluations for the developed methods were obtained.

2. A new device structure for determining the intensity of color is proposed.

Implementation. The work was implemented in the company 3D GENERATION UA, 2021. Result of implementation: methods and software.

Publication. 7 scientific works were published on the subject of research. Including: 6 – in the materials of the International Conferences of the Conference, 1 – certificate of copyright registration for a computer program.

II. LITERATURE ANALYSIS

The general formula for calculating the reflected energy brightness from point *x* of the surface of the object in the $\vec{\omega}_o$ direction using bidirectional reflectance distribution function (BRDF) is as follows [1-3, 8]:

$$L_o(x,\vec{\omega}_o) = \int_{\Omega 2\pi} f_r(\vec{\omega}_o,\vec{\omega}_i)\cdot L(x,\vec{\omega}_i)\cdot(\vec{\omega}_i \bullet \vec{n})\, d\vec{\omega}_i, \quad (1)$$

where f_r – BRDF, $\vec{\omega}_i$ – the vector of the beam of light fall direction, $L(x,\vec{\omega}_i)$ – the energy brightness, coming in the $\vec{\omega}_i$ direction, $(\vec{\omega}_i \bullet \vec{n})$ – the scalar product of the vector $\vec{\omega}_i$ and the normal of the surface $\vec{n}$. Input data for the calculation of BRDF are shown in Fig. 1.1.

Phong BRDF [1-4, 6] became the first empirically model for calculating the specular highlight. To calculate the intensity of specularly reflected light according to the Phong model, three parameters are used: the vector of the reflected beam direction $\vec{R}$, the vector to the observer direction $\vec{V}$ and the specular coefficient *n*. In the presence of such data, the calculation of the specular component of the surface color is carried out as follows:

$$f_r(\omega_i,\omega_o) = (\vec{R} \bullet \vec{V}). \quad (2)$$

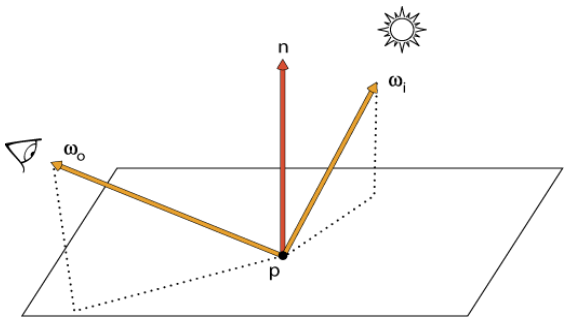


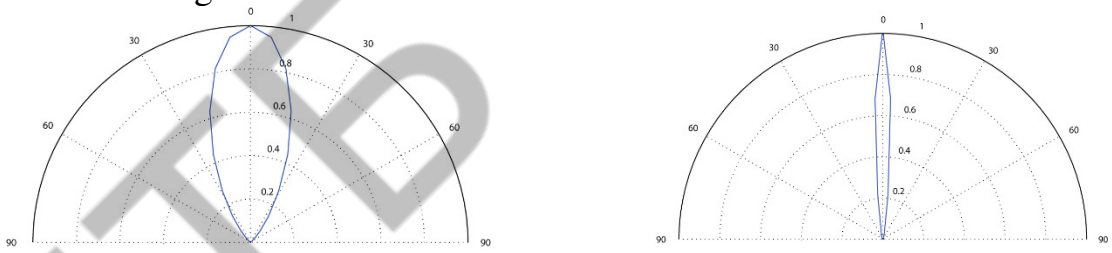
Fig. 1. Input data for the calculation of BRDF

Fig. 1 shows images of two spheres for visualization of which Phong (specular reflection) model was used.

a) b)

Fig.1. Visualization using the Phong model at: a) n=15, b) *n*=75

Fig. 3 shows the distribution of the values of the Phong BRDF depending on the magnitude of the angle between the vectors $\vec{R}$ and $\vec{V}$.



a) b)

Fig.2. Phong's BRDF value depending on the angle between vectors $\hat{R}$ and $\hat{V}$ at: a) n=10, b) *n*=75

The main disadvantage of Phong BRDF [1-4] and other empirical BRDF is that the resulting distribution of specular highlight intensity (Fig. 3) differs significantly from the experimental data obtained as a result of the study of the reflectivity of real materials using a gonioreflectometer. This leads to the fact that the visualization of the surface, which in its properties of scattering incident light resembles real-life materials (metals, dielectrics), using the Phong model is impossible. When using Phong BRDF, the value of the specular coefficient $n \in [1;1000]$. This causes its high computational complexity, which negatively affects the rendering performance of an object that uses such BRDF.

D. Blinn [1-4, 7, 8] proposed to use in the Phong BRDF the value of the cosine of the angle not between vectors $\vec{R}$ and $\vec{V}$, but between the normal $\vec{N}$ and the half-way vector $\vec{H}$, which is equal to $\left(\vec{L}+\vec{V}\right)/\left|\vec{L}+\vec{V}\right|$, where $\vec{L}$ is the beam of light incidence direction [3]. The resulting BRDF was called the Blinn model and has the following form:

$$f_{r_s}(\omega_i,\omega_o)=(\vec{H}\bullet\vec{N})^n. \quad (3)$$

In this form, Blinn BRDF has an additional definition as a function of the distribution of the normals of microfacets of the surface of a three-dimensional object, and the parameter *n* in its context acquires a physical value – the standard deviation of the normals of microfacet from the $\vec{H}$ direction.

If the Blinn function [1-4] is used as a function of the distribution of normal microfacets, the normalization coefficient $\frac{n+2}{2\pi}$ must be entered into the formula (2), the presence of which ensures the equality of the visible area of microfacets to the surface area on which they are located [1].

The use of an $\vec{N}\bullet\vec{H}$ operand also causes a change in the shape of the surface reflection at the limiting angles of light incidence. For convex surfaces, the shape of the specular highlight does not actually differ in the calculation according to the Phong and Blinn models, but the specular highlight on flat surfaces in the case of limiting angles of light incidence differ significantly. The difference in the reflection of specular highlight by Phong and Blinn models described above is shown in Fig. 4.

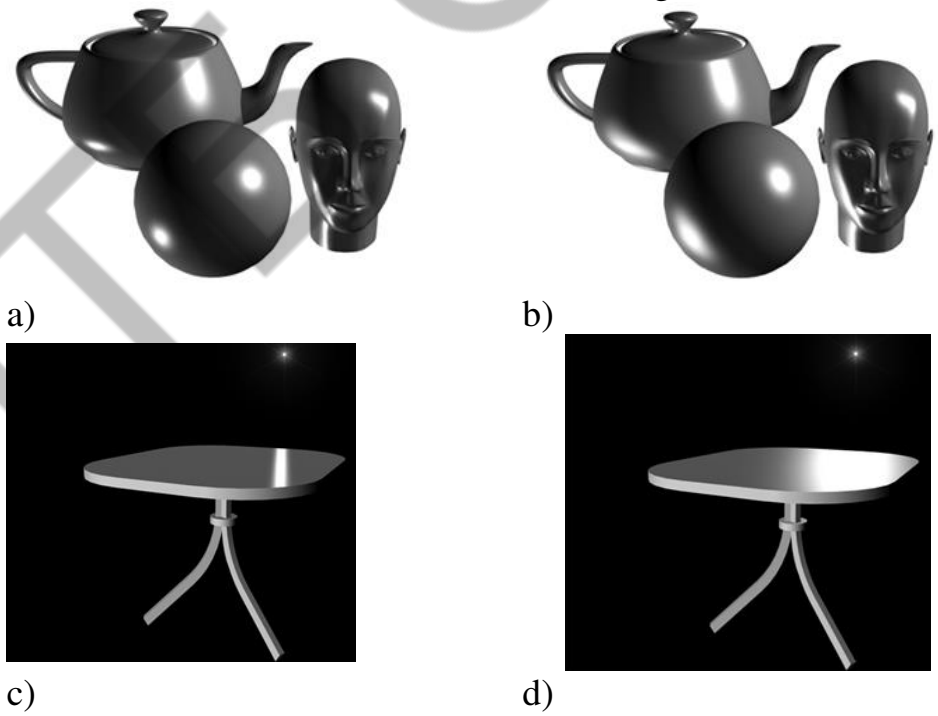


Fig.4. Visualization of specular highlight in the case of: a) convex surfaces and Blinn model, b) convex surfaces and Phong models, c) flat surfaces and Blinn model, d) flat surfaces and Phong model

This feature of specular highlight visualization when light falls at a limiting angle allows to reproduce the specular scattering of light on flat surfaces more realistically. Fig.5 shows images with specular reflection of light from relatively flat surfaces.

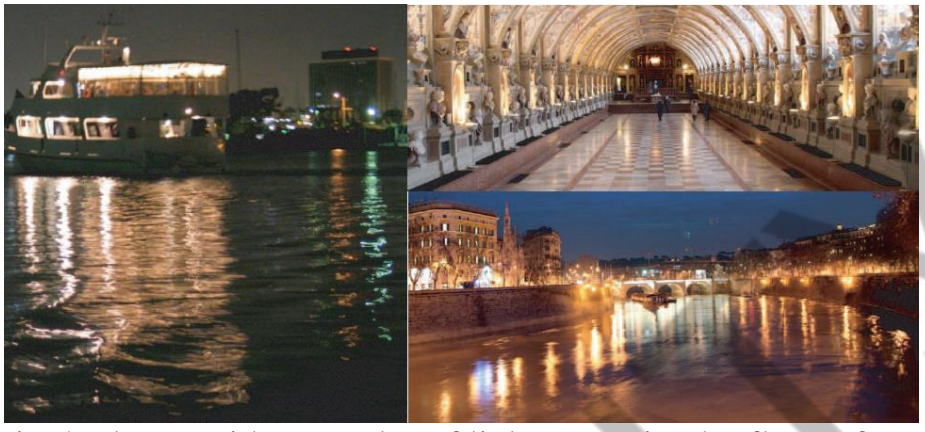


Fig.5. Photos with examples of light scattering by flat surfaces

From fig.5, it can be seen that in a real environment, specular highlight from flat surfaces has a oblong shape, and therefore Blinn BRDF, through the use of a half-way vector, provides a more realistic reproduction of such specular highlight than the Phong model.

At the same time, the use of other vectors did not solve the problem of high computational complexity due to the presence of an exponentiation operation.

To reduce the computational complexity of the Phong and Blinn BRDF by eliminating the operation of elevation to the power, it is possible to use Schlick BRDF [1, 4]:

$$f_{r_s}(\omega_i, \omega_o) = \frac{(\vec{H} \bullet \vec{N})}{n - n(\vec{H} \bullet \vec{N}) + (\vec{H} \bullet \vec{N})}. \quad (4)$$

A significant disadvantage of Schlick BRDF is that the distributive function in the bluming zone drops to zero extremely slowly, as demonstrated in Fig.6, which causes unnatural lighting of the graphic object and additional calculations by increasing the interval of changing the argument.

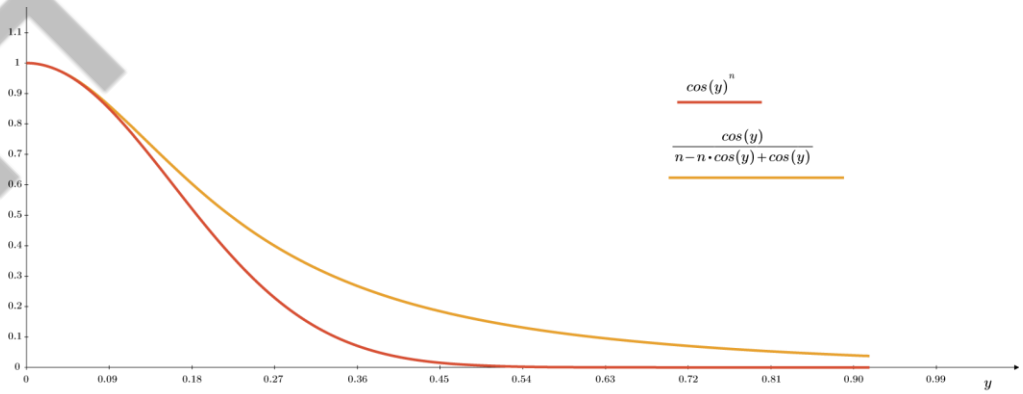


Fig.6. Graphs of the Blinn and Schlick functions for n = 40

To visualize materials like bronze, copper, nickel, it is necessary to take into account the Fresnelian reflection of light and the change in the color of the specular highlight depending on the angle of light incidence. Such phenomena are taken into account when calculating the intensity of the specular color of the surface using the Cook-Torrance BRDF. According to the Cook-Torrance model, the value of the specular component of the color is calculated as follows [1]:

$$f_{r_s}(\omega_i, \omega_o) = \frac{FDG}{\pi(\vec{N} \bullet \vec{L})(\vec{N} \bullet \vec{V})}, \quad (5)$$

where F is the coefficient that determines the Fresnelian reflection of light, D is the distribution function of the normals of the microfacets of the surface, G is the coefficient of geometric overlap of microfacets. Calculating the coefficients F, G and D requires significant computational resources.

Despite the presence of models like the Cook-Torrance BRDF, which take into account the characteristic features of light scattering by metals, modern computer graphics still actively use Phong and Blinn models, especially if there is only an integrated type of GPU. Graphics adapters used in mobile phones, tablets and laptops belong to the class of video cards designed for embedded systems [7] and have significantly less computing power than graphics adapters.

The use of Phong and Blinn models is also relevant in the editors of three-dimensional scenes for shading objects in a scene at the stage of its layout.

Given the relevance of using the Blinn and Phong BRDF, the author chose Blinn BRDF as a prototype to create a more efficient BRDF with a similar form of specular highlight.

The most common methods of shading include the Gouraud method [1-4, 9], which provides an acceptable compromise between the speed of formation of three-dimensional images and their quality. The shading process has the following stages: a) calculate the vectors of the normals to each face; b) by averaging the normals of all faces to which the vertex belongs, the normals at the vertices of the triangle (polygon) are calculated; c) determine the intensity of color at the vertices of the polygon using the values of the normals; e) paint over an area bounded by a polygon by linear interpolation of color intensities along the edges, and then between the edges along each rasterization line. Recently, shading according to Gouraud has often been used as an intermediate stage in the interactive formation of a 3D image, putting the construction of a full-fledged scene at the stage of final rendering.

The disadvantages of the Gouraud method include: a) the method uses linear interpolation to determine the intensity of color, while the diffuse and specular components of color have a nonlinear nature of change; b) the local curvature of the surface is not taken into account, since the vectors of the normals are determined only for the vertices of the triangle; c) specular highlight is reproduced only if the vertices of the triangles are in their zone (a specular highlight that does not have common points with

the vertices of the triangles, or is located inside the triangle, is not formed); d) on the boundaries of two triangles, Mach bands appear [3], which are associated with literal inhibition on the retina; e) there is a change in the intensity of the image color from frame to frame, which is expressed in flashing, especially specular highlight, since during the formation of dynamic images the structure and position of the nodes of the triangulation mesh changes; g) the presence of an artifact of the "star" type, which consists in the fact that the specular highlight, which should have the shape of an ellipse, has the shape of a star. This is explained by the fact that areas of specular highlight are formed in different triangles and the effect of Mach bands appeared; g) the method does not take into account the perspective of the object.

Although the Gouraud method continues to be widely used to form three-dimensional images in real time, the Phong method [1-4, 5, 10] is considered more promising, in which normal vectors are interpolated instead of color intensity values, which are then used in the toning function to calculate the color intensity of each image element. The method is characterized, compared to the Gouraud method, by much higher computational costs, but at the same time a better local approximation of surface curvature is achieved and, as a result, more realistic images are obtained. In recent years the works on the use of Phong shading intensified especially.

In most cases, when shading according to Phong, Blinn and Phong lighting models are used [3]. This is because the indicated BRDF are derived from the cosine of the angle between vectors, which are easily found due to their scalar product. When the light source and observer are located at a sufficiently large distance from the object (the most common case in computer graphics), the $\vec{H}$ vector for the Blinn model is calculated once for each triangle.

III. OBJECT, SUBJECT, AND METHODS OF RESEARCH

The purpose and objectives of the research. The aim of the work is to increase the productivity and realism of the formation of three-dimensional graphic images through the development of new methods and means.

The object of the research is the process of shading three-dimensional graphic objects.

The subject of the research is methods and means of shading three-dimensional graphic objects.

Research methods. In the work there were used: the theory of differential calculus, analytical geometry, number theory for the development of shading methods; theory of algorithms for the development of programs for shading and normalization of vectors; computer modeling to test theoretical statements.

IV. ORGANIZATION OF THE COMPUTATIONAL PROCESS OF HIGH-PRODUCTIVE RENDERING

4.1. Property of constancy of the increase in color intensity along the horizontal (vertical) rasterization lines of the triangle

When developing methods and means of shading, it is important to establish such functional relationships between the output parameters that will significantly simplify the implementation of rendering both at the software and hardware levels.

Definition. The rasterization line of a triangle (RLT) is the space between its edges in one of the orthogonal directions.

Statement. With linear interpolation of color intensities along edges and triangle rasterization lines, the increase in color intensity along horizontal (vertical) rasterization lines is constant.

Proof. Let the triangle ABC be given by the coordinates of its vertices (X_A, Y_A), (X_B, Y_B), (X_C, Y_C) and their corresponding color intensities: I_A, I_B, I_C. Let`s pass through the vertex B a segment BK (Fig.7), which is parallel to the abscissa axis. We get two triangles – ABK and BKC.

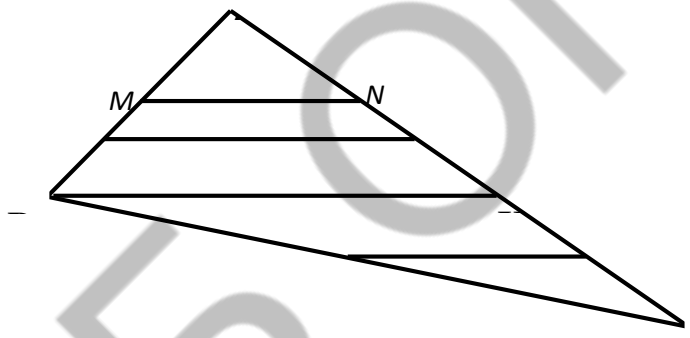


Fig.7. Rasterization lines of the triangle

Let's find increments in color intensities for arbitrarily selected rasterization strings FG and MN:

$$\Delta I_{MN} = \frac{I_N - I_M}{MN}, \quad \Delta I_{FG} = \frac{I_G - I_F}{FG} \quad . \tag{6}$$

Express the intensity of the color of the points G and F through the intensity of the color of the vertices $A, B\ i\ C$: $I_G = \frac{I_C - I_A}{AC} \cdot AG + I_A$, $I_F = \frac{I_B - I_A}{AB} \cdot AF + I_A$.

Substitute the value of I_G, I_F into the formula (6). Then we obtain

$$\Delta I_{FG} = \frac{I_C - I_A}{AC} \cdot \frac{AG}{FG} - \frac{I_B - I_A}{AB} \cdot \frac{AF}{FG}. \tag{7}$$

Similarly, for points *M* and *N,* write

$$I_N = \frac{I_C - I_A}{AC} \cdot AN + I_A, \; I_M = \frac{I_B - I_A}{AB} \cdot AM + I_A.$$

Find the increase in color intensity along the rasterization line MN

$$\Delta I_{MN} = \frac{I_C - I_A}{AC} \cdot \frac{AN}{MN} - \frac{I_B - I_A}{AB} \cdot \frac{AM}{MN}. \quad (8)$$

Triangles AFG and AMN similar, since their corresponding angles are equal. From the similarity of triangles it follows that

$$\frac{AG}{FG} = \frac{AN}{MN}, \quad \frac{AF}{FG} = \frac{AM}{MN}. \quad (9)$$

Given the ratio (9), it can be stated that the right parts of the expressions (7) and (8) coincide, i.e. $\Delta I_{FG} = \Delta I_{MN} = \Delta I_g$.

Since the triangle ABK is similar to the triangle AFG, then $\Delta I_{BK} = \Delta I_{FG}$.

Similarly, it can be shown that for an arbitrary RLT WV of triangle BKC $\Delta I_{WV} = \Delta I_{BK}$. Since the triangles ABK and BKC have a common line BK, it can be argued that the resulting property takes place for the entire triangle.

Thus, the increase in intensity is a constant value for all horizontal rasterization lines. Obviously, increments in color intensities have constant values for vertical rasterization lines as well.

The proven property can significantly reduce the volume of calculations during the implementation of shading, since the increase in color intensity is determined for the entire triangle, and not for each line of rasterization. This allows to perform these calculations in the preparation cycle and eliminate "long" operations directly from the shading cycle.

The analysis showed that compared to the classical implementation, a reduction in shading time by 1.7 times is achieved.

Increments in color intensities for adjacent triangles have different meanings, therefore, due to the discrete nature of the formation of step trajectories of the edges of the polygon, an artifact violation of the monotony of the change in color intensity within neighboring polygons is possible.

Find expressions for calculating intensity increments along horizontal and vertical rasterization lines. To do this, make a system of equations

$$\begin{cases} I_B = I_A + \Delta I_v \cdot \Delta Y_{BA} - \Delta I_g \cdot \Delta X_{AB}, \\ I_C = I_A + \Delta I_v \cdot \Delta Y_{CA} - \Delta I_g \cdot \Delta X_{CA}. \end{cases}$$

Solving a system of equations with respect to I_g, I_v, we obtain

$$\Delta I_g = \frac{\Delta X_{AB} \cdot (I_A - I_C) + \Delta X_{CA} \cdot (I_A - I_B)}{\Delta X_{AB} \cdot \Delta Y_{CA} - \Delta X_{BA} \cdot \Delta X_{CA}}, \quad \Delta I_v = \frac{\Delta Y_{CA} \cdot (I_B - I_A) + \Delta Y_{BA} \cdot (I_A - I_C)}{\Delta X_{AB} \cdot \Delta Y_{CA} - \Delta X_{BA} \cdot \Delta X_{CA}}.$$

The diagonal step can be considered as the simultaneous execution of vertical and horizontal step movement, therefore, for edges with positive coordinate increments

$$\Delta I_{dg} = \frac{(\Delta X_{AB} + \Delta Y_{BA}) \cdot (I_A - I_C) + (\Delta X_{CA} - \Delta Y_{CA}) \cdot (I_A - I_B)}{\Delta X_{AB} \cdot \Delta Y_{CA} - \Delta X_{BA} \cdot \Delta X_{CA}}.$$

In other cases, it is necessary to take into account the signs of increments of coordinates.

The proven property of constancy of color intensity increase can also be used for vectors, replacing the intensity of color with a coordinate component. It can be argued that the increments of orthogonal coordinate constituent vectors of normals [10] along horizontal (vertical) RLT have constant values. That is, for normal vectors along horizontal (vertical) lines $\Delta x = const,\ \Delta y = const,\ \Delta z = const$, where $\Delta x,\ \Delta y,\ \Delta z$ are the differences between the corresponding coordinate components of the two adjacent vectors in the rasterization line.

To normalize the normal vector at the starting point of the rasterization line of the triangle, use the formula

$$\vec{N} = \frac{x}{\sqrt{x^2 + y^2 + z^2}}\vec{i} + \frac{y}{\sqrt{x^2 + y^2 + z^2}}\vec{j} + \frac{z}{\sqrt{x^2 + y^2 + z^2}}\vec{k}.$$

For the normal vector following it, the sub-root expression will look like this

$$F_i = (x + \Delta x)^2 + (y + \Delta y)^2 + (z + \Delta z)^2 =$$

$$= (x^2 + y^2 + z^2) + 2 \cdot (x \cdot \Delta x + y \cdot \Delta y + z \cdot \Delta z) + ((\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2).$$

Introduce the notation

$$A = x^2 + y^2 + z^2\ ,\ \ B = (x \cdot \Delta x + y \cdot \Delta y + z \cdot \Delta z),\ C = (\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2.$$

For i-vector relative to the first vector in the rasterization line $F_i = A + 2 \cdot i \cdot B + i^2 \cdot D$. Use the finite difference method for the last member [1], for which find

$$\Delta_1 = (i + 1)^2 \cdot D - i^2 \cdot D = 2 \cdot i \cdot D + D,$$

$$\Delta_2 = 2 \cdot (i + 1) \cdot D + D - 2 \cdot i \cdot D - D = 2 \cdot D.$$

Taking into account the resulting expressions, write

$F_{i+1} = F_i + 2 \cdot B + C_{i+1}$, where $C_{i+1} = C_i + 2 \cdot D$, $C_1 = D,\ F_o = A,\ C_0 = 1$.

The obtained formulas for calculating the sub-root expression do not have multiplication operations.

The proven property of constancy of the increase in color intensity along the RLT can be used to parallelize the shading.

4.2. Methods of parallelization of the shading procedure

The first method is based on the distribution of computational actions of the shading procedure at the lower level, when the geometric processor is given the parameters of comparable triangles, as well as the intensity of color at their vertices. The shading of the area bounded by a triangle is carried out using Sierpinski triangulation. According to the method, the triangle is divided into four comparable by connecting the middle of its sides. The result is four triangles that are equal in area. Comparable triangles are shaded over in parallel. In this case, only one shading device is used, which tones one of the triangles, and the results of its work are transferred to all other triangles with a

certain displacement in coordinates and corresponding transformations of the color intensity of the points. Due to the parallelization of the computational process, the original triangle will be shaded 4 times faster.

When shading, rasterization of the upper triangle is performed and all elementary step increments for other triangles are repeated, taking into account the fact that triangles 1, 2, 3 have the same shading directions, and triangle 4 is shaded in the opposite direction (Fig. 8). Before rasterization, the coordinates of the vertices of all comparable triangles are found. Similarly, when determining the intensity of color of the inner points of the upper (left) comparable triangle, all actions of code interpolation are duplicated for other comparable triangles, but with respect to their vertices.

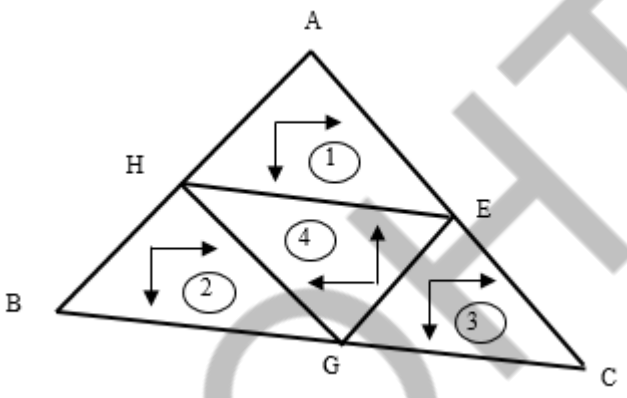


Fig.8. Shading of the triangles

Since the triangulation of the original triangle is carried out by dividing its edges in half, the problem arises of such a division of the triangle into comparable ones, at which there will be no "cut" points, since the increments of the sides of the original triangle are not always even.

One of the methods of parallelization is the independent determination of color intensities at even and odd points of the rasterization line (Fig. 9, a), which provides an increase in the productivity of shading up to two times. In the preparation cycle, the color intensities I_1, I_2 are found, respectively for the first and subsequent points in the rasterization line. The color intensities of the following points in the RLT are found by the formulas:

$$I_4 = I_2 + 2\Delta I_g,\ I_6 = I_4 + 2\Delta I_g,\ \ ...,\ ...;\ I_{2w} = I_{2w-2} + 2\Delta I_g,$$
$$I_3 = I_1 + 2\Delta I_g,\ I_5 = I_3 + 2\Delta I_g,\ \ ...,\ ....\ I_{2w+1} = I_{2w-1} + 2\Delta I_g,$$

Since the increase in color intensity along the rasterization line is constant, the intensity of the color at its i point can be determined by the formula $I_i = I_1 + i \cdot \Delta I_\Gamma$, which allows to independently determine the intensity of the color of the whole segment of points (Fig. 9, b) in the rasterization line. The simplest hardware implementation of the method takes place with segment sizes that are multiples of the power of two.

With segmental implementation, color intensity increases are determined for all

points in the segment and the color intensity of the first point of the next segment. When forming the last segment due to masking, the recording in the video memory of those points that go beyond the rasterization line is blocked.

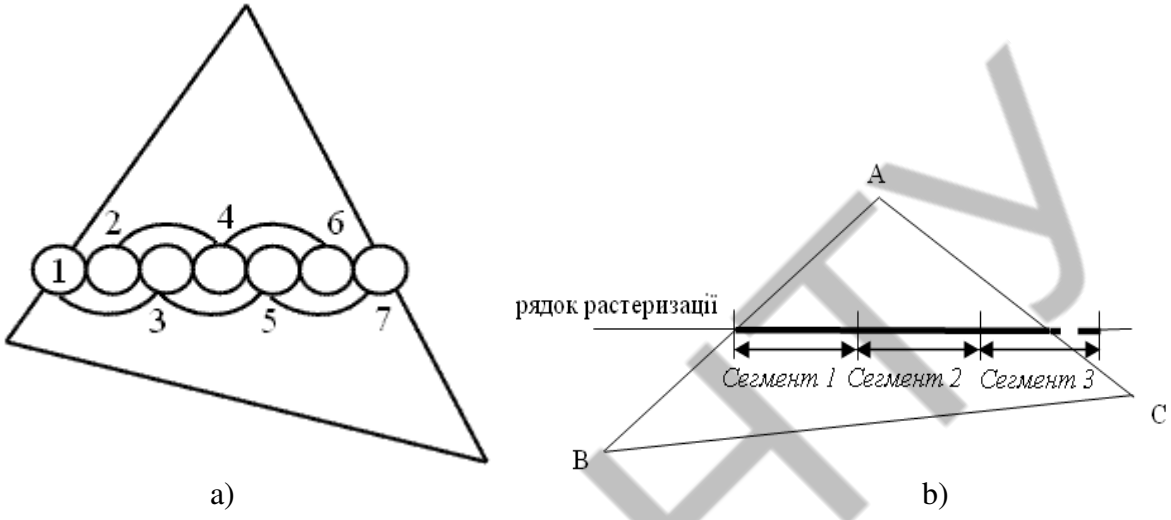


Fig.9. Methods of parallelization: a) shading over even and odd points in a rasterization line of a triangle; b) separating the rasterization line into segments

4.3. Polynomial approximation of BRDF

Consider the approximation of the BRDF by quadratic function with respect to $cos\gamma$, which has the following form

$$cos^n\gamma = a\cdot cos^2\gamma + b\cdot cos\gamma + c. \qquad (10)$$

To find unknown coefficients, use points $a,\ b,\ c$ whose BRDF values can be easily determined.

If $\gamma = 0 \quad cos^n\gamma = 1$, then

$$a + b + c = 1. \qquad (11)$$

If $cos\gamma = 0$, then $cos^n\gamma = 0$. Therefore

$$a\cdot 0 + b\cdot 0 + c = 0. \qquad (12)$$

From the formulas (11) and (12) find that

$$c = 0; \quad b = 1 - a. \qquad (13)$$

To find the coefficient a, equate the value of the BRDF and the quadratic function at the inflection point of the function $cos^n\gamma$. The abscissa of this point is $\vartheta = arctg(\frac{1}{\sqrt{n-1}})$ [3] . Taking into account (10) and (13), write

$$a\cdot cos^2\vartheta + (1-a)\cdot cos\vartheta = cos^n\vartheta.$$

Introduce the following notation $t = cos\vartheta,\ \ t^n = cos^n\vartheta$. Then the previous formula

will have the form: $a \cdot t^2 + (1-a) \cdot t = t^n$. From the last equation find that $a = (t^{n-1} - 1)/(t-1)$. Taking into account the values t and ϑ write the formula for determining the coefficient a

$$a = \frac{\cos^{n-1}(\operatorname{arctg}(\frac{1}{\sqrt{n-1}})) - 1}{\cos(\operatorname{arctg}(\frac{1}{\sqrt{n-1}})) - 1}.$$

It is advisable to pre-calculate the values of a and store them in memory.

The maximum value of a is 786.4. Up to the inflection point of the BRDF, the maximum relative approximation error does not exceed 3.14 %, and the maximum absolute error does not exceed 0.025. When the accuracy of approximation is not subject to strict requirements, the value of the coefficient a can be determined by the approximation formula $a = 0{,}786 \cdot n$ with a relative error less than 0.3% for all $21 \le n \le 1000$. For the remaining range the relative error does not exceed 4%.

In determining the coefficient a, other levels of the distribution function can be used. For example, when using an ordinate level equal to *0.5*, the maximum approximation error of the distributive function $\cos^n \gamma$ will increase to 6%, but the specular highlight attenuation zone will be better reproduced.

In the case when the level q is selected to determine the coefficient a, then

$$a = \frac{\cos^{n-1}(\exp(\frac{\ln(q)}{n})) - 1}{\cos(\operatorname{arctg}(\exp(\frac{\ln(q)}{n}))) - 1}.$$

Characteristic features of the approximation of BRDF with a polynomial of the second power:

a) it is sufficient to determine only two coefficients – a and b;

b) due to a sharp decrease of the approximating function when it approaches the zero level, the specular highlight has a visually noticeable boundary, which is typical for materials with a high specular coefficient;

c) by selecting the level of BRDF to determine the coefficient a, it is possible to control the error of reproduction of the epicenter of the specular highlight and its attenuation zone;

d) since the new BRDF takes a negative value, it is necessary to provide for the cutting off of the function at the boundary of the transition through the zero level.

Consider the approximation of the BRDF $\cos(\gamma)^n$ by function $f_1(x)$, which has the form

$$f_1(\gamma) = a_1 \cdot \cos(\gamma)^3 + b_1 \cdot \cos(\gamma)^2 + c_1 \cdot \cos(\gamma) + d_1.$$

Let's find the unknown a_1, b_1, c_1, d_1. To do this, it is needed to make a system of four equations. Taking into account that $\cos(0)=1$, get $a_1 + b_1 + c_1 + d_1 = 1$, $\cos(\gamma)=0$

when $d_1 = 0$. Equating the value of the BRDF and the cubic polynomial at the point t, where $cos(t)^n = Q$, find that

$$a_1 \cdot cos^3(t) + b_1 \cdot cos^2(t) + c_1 \cdot cos(t) + d_1 = Q.$$

The last equation of the system to find the unknowns a_1, b_1, c_1, d_1 can be obtained by equating the derivatives of the BRDF and the proposed cubic polynomial at the point t where $cos(t)^n = Q$.

$$\frac{\partial(cos^n \gamma)}{\partial \gamma} = -n \cdot cos(\gamma)^{n-1} \cdot sin(\gamma).$$

$$f(\gamma)' = -3 \cdot a_1 \cdot cos(\gamma)^2 \cdot sin(\lambda) - 2 \cdot b_1 \cdot cos(\gamma) \cdot sin(\gamma) - c_1 \cdot sin(\gamma).$$

Let's equate the right parts of the last two expressions and divide them into $-sin(t)$ Get

$$n \cdot cos(t)^{n-1} = 3 \cdot a_1 \cdot cos(t)^2 + 2 \cdot b_1 \cdot cos(t) + c_1.$$

To find the coefficients of the cubic multimember $f_1(\gamma)$, reduce the obtained equations into a system of equations

$$\begin{cases} a_1 + b_1 + c_1 + d_1 = 1, \\ d_1 = 0, \\ a_1 \cdot cos^3(t) + b_1 \cdot cos^2(t) + c_1 \cdot cos(t) + d_1 = Q, \\ n \cdot cos^{n-1}(t) = 3 \cdot a_1 \cdot cos^2(t) + 2 \cdot b_1 \cdot cos(t) + c_1. \end{cases}$$

After equivalent transformations, obtain a system of two equations:

$$\begin{cases} a_1 \cdot cos^3(t) + b_1 \cdot cos^2(t) + (1 - a_1 - b_1) \cdot cos(t) = Q, \\ 3 \cdot a_1 \cdot cos^2(t) + 2 \cdot b_1 \cdot cos(t) + (1 - a_1 - b_1) = n \cdot cos^{n-1}(t). \end{cases}$$

The solution of the resulting system of equations is as follows:

$$a_1 = \frac{n \cdot cos^n(t)(cos(t) - 1) - Q \cdot (2 \cdot cos(t) - 1) + cos^2(t)}{cos^2(t) \cdot (cos(t) - 1)^2},$$

$$b_1 = \frac{n \cdot cos^n(t)(1 - cos^2(t)) + Q \cdot (3 \cdot cos^2(t) - 1) - 2 \cdot cos^3(t)}{cos^2(t) \cdot (cos(t) - 1)^2}.$$

If choose $Q = 0,5$, then

$$a_1 = \frac{(q-1)(2q+n)+1}{2 \cdot (q-1)^2 \cdot q^2}, b_1 = -\frac{q^2(4q-3) + n(q^2-1) + 1}{2 \cdot (q-1)^2 \cdot q^2},$$

where $q = e^{\frac{-0.6931471}{n}}$.

Using a cubic polynomial to reproduce the epicenter of the specular highlight provides sufficiently high accuracy (Fig.10). The simulation showed that for the most common range, the maximum relative error in the approximation of the epicenter of the

specular highlight does not exceed 0.67%. Compared to the Schlick distribution function [3] at this interval, the maximum relative approximation error is reduced by more than 17 times. $\left(\forall \boldsymbol{n} \in [1;256]\right)$.

Fig.10. Three-dimensional figures visualized using BRDF of the third power

4.3. Hardware implementation of the shading device using the developed BRDF

In this paper, a new BRDF is proposed using a second-power polynomial, the calculation of which involves the use of operations that are simple from a computational point of view, which allows it to be implemented by hardware. Fig. 11 shows the block diagram of the device. The peculiarity of the structure is the usage only one coefficient, the value of which determines all the others.

The device (Fig.11) includes registers $RG1-RG5$, block of permanent memory $PROM$, multiplication blocks $MUL1-MUL5$, block of inverters BI, combination adders $Sm1$-$Sm2$. The value n of the specular coefficient is written in the register $RG1$ and the cosine value of the angle between the normal vector and the vector $\vec{H}$ is written in the register $RG2$. The registers $RG3$- $RG5$ store the intensity values of R, G, B components of the color of the light source respectively, multiplied by k_s. For each value n the corresponding values of coefficient a are stored in the $PROM$ block. The value $\cos\gamma$ from the output of register $RG2$ is fed to both inputs of the multiplication block $MUL1$, which ensures the formation at its output $\cos^2\gamma$. At the output of the block BI an inverse value of $\cos\gamma$ is formed, which enters the information input of the adder $Sm1$, at the output of which the operand $\cos^2\gamma-\cos\gamma$ is obtained. In the adder $Sm1$ subtraction operation is conducted in supplementary code, therefore the level of the logical unit is given to its transfer input. At the output of $MUL2$ the product $a\cdot(\cos^2\gamma-\cos\gamma)$ is formed that goes to the information input of the adder $Sm2$. At the output of the adder $Sm2$, the value of the BRDF is formed, which is equal to $a\cdot(\cos^2\gamma-\cos\gamma)+\cos\gamma$. From the output of the adder $Sm2$, the value of the BRDF is fed to the first inputs of the multiplication blocks $MUL3-MUL5$, at the outputs of which the intensities of the specular component of the color for each channel are formed.

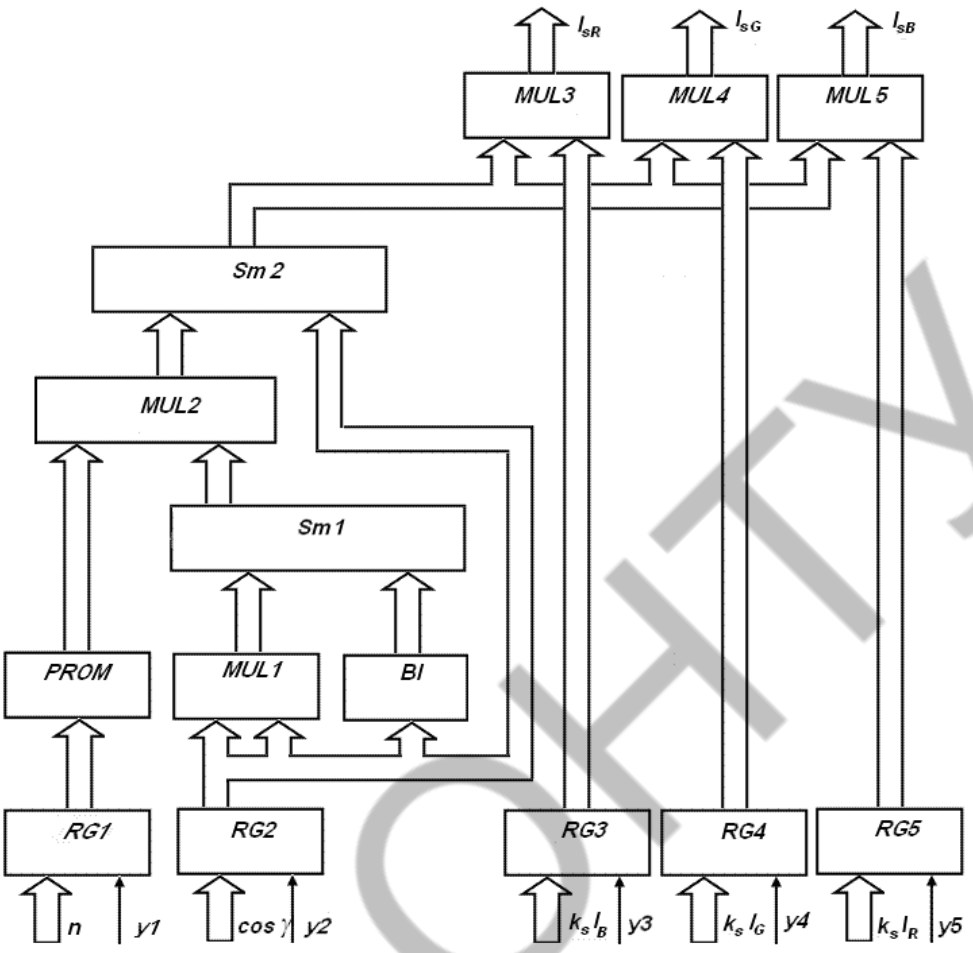


Fig.11. Block diagram of the device for determination of the specular component of color using BRDF of type $a \cdot cos^2 \gamma + b \cdot cos \gamma + c$

4.4. Modification of the Gouraud method

The most common methods of shading include the Gouraud method [3, 9], which provides an acceptable compromise between the speed of formation of three-dimensional images and their quality. The shading process has the following stages [3,8]: a) calculate the vectors of the normals to each face; b) by averaging the normals of all faces to which the vertex belongs, the normals at the vertices of the triangle (polygon) are calculated; c) determine the intensity of color at the vertices of the polygon using the values of the normals; e) shading over an area bounded by a polygon by linear interpolation of color intensities along the edges, and then between the edges along each rasterization line.

The most significant drawback of the Gouraud method is that the specular highlights that cross the edges of the triangle will not be reproduced.

The author proposes a method that eliminates this drawback.

The normal vectors to the points of the edge *AB* can be found from the parametric equation $\vec{N}_i^{AB} = \vec{N}_A + t_1 \cdot (\vec{N}_B - \vec{N}_A)$. Let's normalize the vector $\vec{N}_i^{AB}$ (Fig. 12).

$$\frac{\vec{N}_i^{AB}}{\left|\vec{N}_i^{AB}\right|} = \frac{\vec{N}_A + t_1 \cdot (\vec{N}_B - \vec{N}_A)}{\sqrt{(\vec{N}_A)^2 + 2 \cdot t_1 \cdot \vec{N}_A \cdot (\vec{N}_B - \vec{N}_A) + t_1^2 \cdot (\vec{N}_B - \vec{N}_A)^2}}.$$

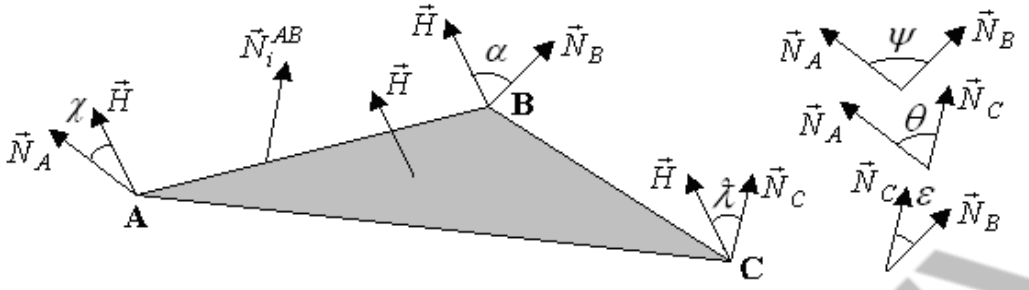


Fig.12. Normal vectors of the triangle ABC

Since $\vec{N}_A$, $\vec{N}_B$ are unit, then $\vec{N}_A^2 = \vec{N}_B^2 = 1$. Taking into account the latter, as well as the fact that $\vec{N}_A \cdot \vec{N}_B = cos\psi$, find

$$\frac{\vec{N}_i^{AB}}{\left|\vec{N}_i^{AB}\right|} = \frac{\vec{N}_A + t_1 \cdot (\vec{N}_B - \vec{N}_A)}{\sqrt{2 \cdot t_1^2 \cdot (1 - cos\psi) - 2 \cdot t_1 \cdot (1 - cos\psi) + 1}}.$$

Find positions on the edges of the triangle *ABC*, where the specular highlight is the most intense. The above can be judged by the value of the cosine of the angle between the vector $\vec{H}$ and the unit vectors of the normals to the points of the edges.

So, for example, for the edge *AB*

$$\frac{\vec{N}_i^{AB} \cdot \vec{H}}{\left|\vec{N}_i^{AB}\right|} = \frac{(\vec{N}_A + t_1 \cdot (\vec{N}_B - \vec{N}_A)) \cdot \vec{H}}{\sqrt{2t_1^2 (1 - cos\psi) - 2t_1 (1 - cos\psi) + 1}} = \frac{cos\chi + t_1 (cos\alpha - cos\chi)}{\sqrt{2t_1^2 (1 - cos\psi) - 2t_1 (1 - cos\psi) + 1}},$$

where χ, α are the angles between the vector $\vec{H}$ and the vectors $\vec{N}_A, \vec{N}_B$ respectively.

In the above expression, all quantities are scalar.

Let's find t_1, in which the scalar component of the color on the edge *AB* has the maximum value. To do this, find the derivative of the previous expression and equate it to zero.

$$\left(\frac{\vec{N}_i^{AB} \cdot \vec{H}}{\left|\vec{N}_i^{AB}\right|}\right)' = \frac{t_1 (1 - cos\psi) \cdot (cos\alpha - cos\chi) - cos\chi \cdot cos\psi + cos\alpha}{(2t_1^2 (1 - cos\psi) - 2t_1 (1 - cos\psi) + 1)^{3/2}} = 0.$$

The last equation has a root

$$t_1 = \frac{cos\chi \cdot cos\psi - cos\alpha}{(cos\psi - 1)(cos\alpha + cos\chi)}. \qquad (14)$$

Similarly, for the edges *AC* and *BC*, the values of the parameters t_2, t_3, at which the maximum value of the specular component of the color is achieved, respectively are equal to

$$t_2 = \frac{\cos\chi \cdot \cos\theta - \cos\hat{\lambda}}{(\cos\theta - 1)(\cos\hat{\lambda} + \cos\chi)}, \quad t_3 = \frac{\cos\alpha \cdot \cos\varepsilon - \cos\hat{\lambda}}{(\cos\varepsilon - 1)(\cos\hat{\lambda} + \cos\alpha)}. \qquad (15)$$

By value t, it is easy to find coordinates x, y on the edges of a triangle, where the specular component of the color is maximum. So, for example, for the edge AB

$$x = \lceil x_A + t_1 \cdot (x_B - x_A) \rceil, \quad y = \lceil y_A + t_1 \cdot (y_B - y_A) \rceil.$$

When determining the diffuse component of a color, a vector $\vec{L}$ is used instead of a vector $\vec{H}$.

The shading process according to the developed approach contains: calculation of the points on the edges that intersect with the specular highlight zone, where the maximum value of the scalar product of the normalized normal vector and vector $\vec{H}$ takes place; calculation of the coordinates of points on the edge, where there is the greatest value of the specular component of the color; comparison of the obtained value of color intensity with the limit and making a decision on taking into account the specular component of the color; calculation of color intensity at predetermined points; dividing the original triangle into components; shading of the obtained triangles according to the Gouraud method.

The original triangle can be divided into several depending on the maximum values of color intensities at the points t_1, t_2, t_3. If all three values of color intensities are greater than the threshold, then the triangle is divided into four components (Fig. 13,a). In the case when only one of the points t_1, t_2, t_3 reaches the value of color intensities greater than the threshold, the triangle is divided into two components (Fig. 13,b). If the maximum values of the specular component of the color on the edges of the triangle are greater than the threshold value at only two points, then the triangle can be divided into three components as shown in Fig.13, c, d. In the latter case, at the point D, which divides the segment AC in half, it is easy to find the value of color intensity as the average value of color intensities at points A and C.

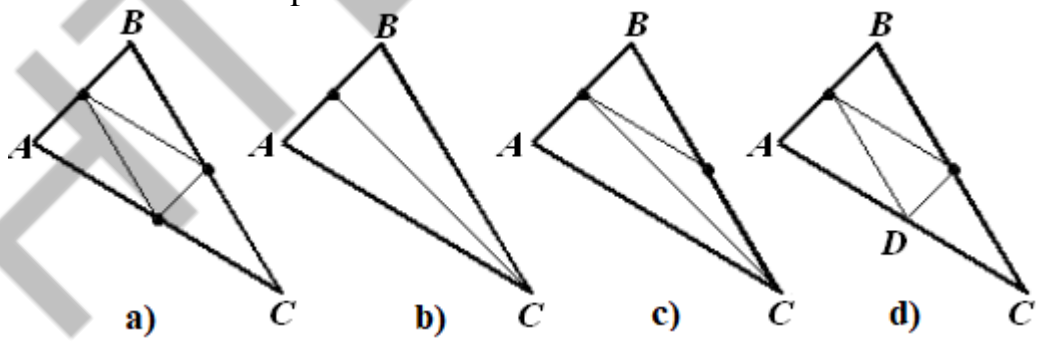


Fig. 13. Dividing a triangle into components

4.5. Software implementation

There has been developed a computer program (see listing, Appendix B) (Fig. 14) based on an open professional pipeline for modeling and testing proposed methods.

For the development of the software product, the Java programming language was chosen, the implementation of which is adapted to different platforms (the most famous of them are Win32, Unix/x86, Unix/Alpha, Solaris/Sparc, MacOS). A large number of libraries have been developed for the Java language, which greatly simplifies the development of software products. The advantage of the Java language is that the software module with an interface based on Swing or AWT can be easily adapted for use as a Java applet, which makes it possible to use the software module directly from the Internet browser.

The software module is built on the basis of the idx3d library, which has basic functions for working with 3D models and three-dimensional geometry. The object-oriented model of the software module is shown in Fig. 15.

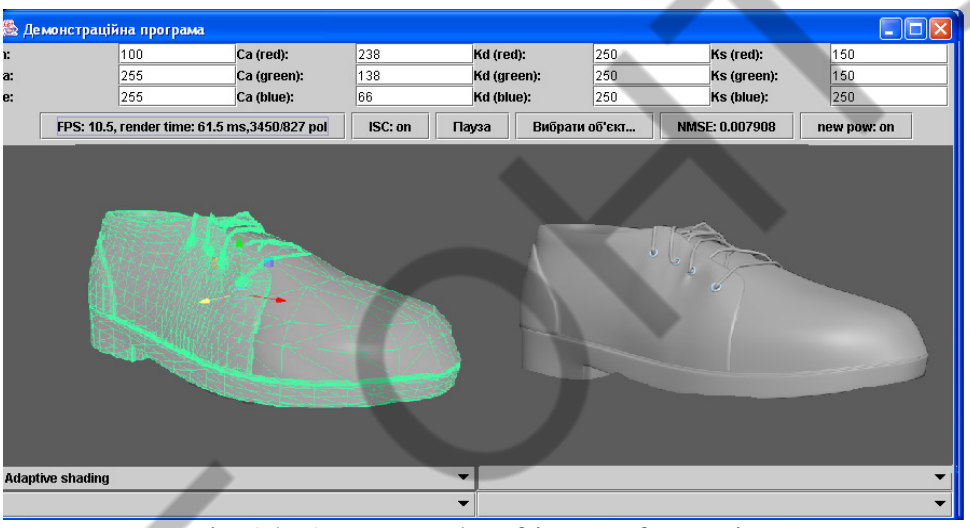


Fig.14. An example of image formation

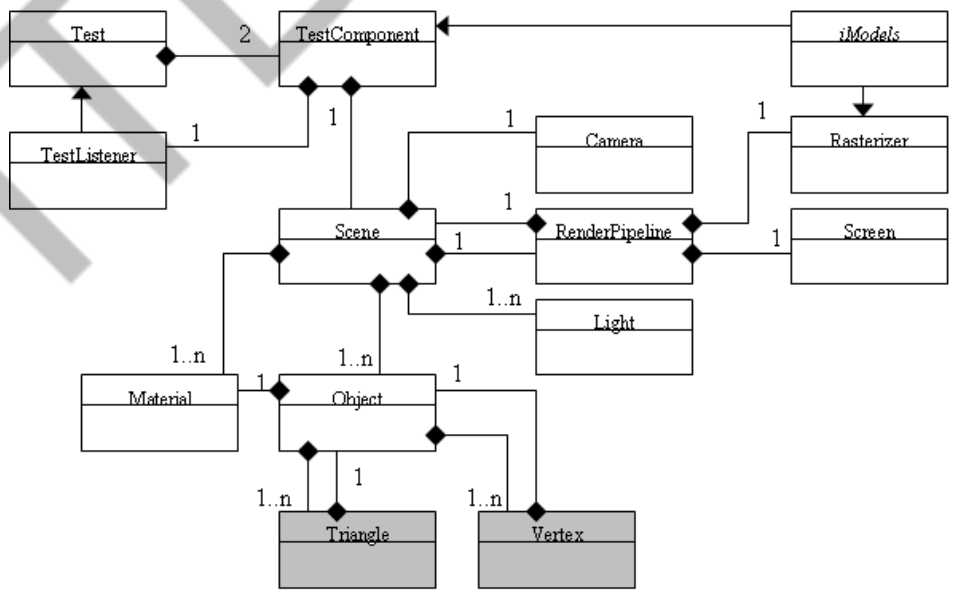


Fig. 15. Object-oriented model of the software module

Fig. 16 shows one of the examples of images, formed by using proposed methods.

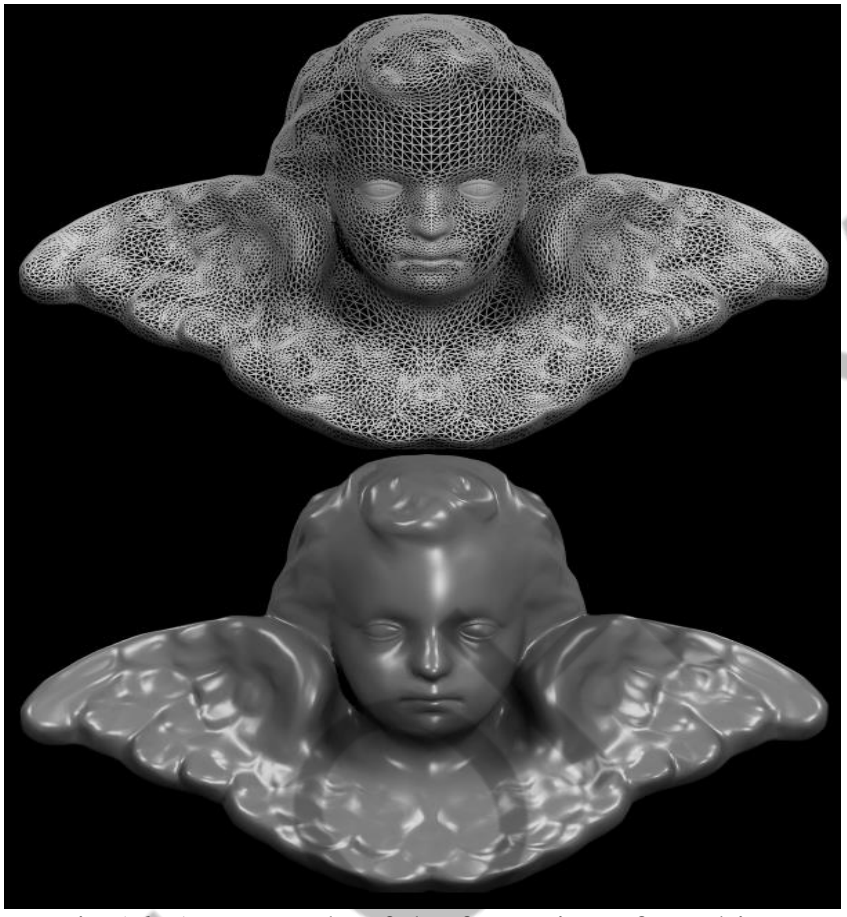


Fig.16. An example of the formation of an object

The simulation using the developed program confirmed the reliability of the obtained theoretical statements.

V. CONCLUSIONS

In the course of the work, the following results were obtained:

1. The analysis of the most common bidirectional functions of surface reflectance in realistic computer graphics has been conducted. The need to develop new models of surface reflectivity, which, unlike the existing ones, provided less computational complexity, simplicity of hardware implementation and reproduction of color intensities on the surface of the object within the limits that determine the visual identity with the reference image, is substantiated. The methods of shading are analyzed.

2. The property of constancy of the increase of color intensity along the horizontal (vertical) rasterization lines of the triangle has been proved, which made it possible to increase the productivity of shading due to the fact that the increase in color intensity is calculated for the triangle, and not for each of its rasterization lines. This made it possible on average to reduce the shading time of the average triangle by an average of 1.7 times and simplify the hardware implementation.

3. Based on the proven property of constancy of the increase of color intensity, methods of parallelization of the computational process are proposed, which provide an increase in shading performance with a balanced load of renders.

4. New models of surface reflectivity are proposed, which differ from the existing ones by the absence of an elevation operation to the power, and in relation to the well-known Schlick BRDF – the absence of a division operation, the presence of only basic addition and multiplication operations, which makes it possible to develop devices with a relatively simple hardware implementation. The proposed BRDFs have a second and third degree.

5. A modification of the Gouraud method is proposed, which consists in determining the maximum values of color intensities on the edges of a triangle, followed by the division of the triangle into components, which makes it possible to reproduce the specular highlights that intersect the edges of the triangle.

Practical value of the work: Shader programs have been developed and integrated into a professional graphics pipeline. A new device structure for determining the intensity of color is proposed.

VI. REFERENCES

1. Hearn D. Computer graphics and OpenGL standard / D. Hearn, M. Baker. – M.: Publishing House “Vilyams", 2005. — 1168 p.

2. Tsysarzh V.V. Mathematical methods of computer graphics / V. Tsisarzh., R.I. Marusin. - K. : Fact, 2004. — 464 p.

3. Romanyuk O.N. High-performance methods and means of shading three-dimensional graphic objects. Monograph / O.N. Romanyuk, A.V. Chornyi. - Vinnytsia : UNIVESUM-Vinnytsia, 2006. — 190 p.

4. Romanyuk O.N., Ozerchuk D.A., Stanislavenko E.G., Kotlyk S.V. Adaptive use of Gouraud and Phong methods for rendering problems. Information Technologies and Automation – 2021 / Proceedings of the XIV International Scientific and Practical Conference. Odesa, October 21-22, 2021 -Odesa, ONAFT Publishing House, 2021 –P. 330-332.

5. Romanyuk O.N., Stanislavenko E.G., Romanyuk O.V. Innovations in "Blender" software // Abstracts of the XII International Scientific and Technical Conference "Information and Computer Technologies – 2021 (ICT-2021)", Zhytomyr, April 01 - 03, 2021 - Zhytomyr: Zhytomyr Polytechnic, 2021. – 205 p. – pp. 15-16.

6. Romanyuk O.N., Khomyuk I.V., Vintonyuk V.V., Stanislavenko E.G. Shader implementation of shading. Electronic information resources: creation, use, access. Proceedings of the International Scientific and Practical Internet Conference, November 9-10, 2021. – Sumy/Vinnytsia: NIKO/VNTU, 2021. – pp. 184-185.

7. Romanyuk O.N., Savytska L.A., Romanyuk O.V., Yakovenko O.O. Stanislavenko E.G. Computer program for calculating the bidirectional reflectance distribution function for rendering tasks: Certificate of copyright registration for the work No91849. 28.08. 2019.

8. Romanyuk O.N. , Stanislavenko E.G. , Romanyuk S.O. Analysis of surface reflectance models. Modern science: problems and prospects (part II): materials of the VI International Scientific and Practical Conference in Kyiv, January 12-13, 2022. – Kyiv: MCNiD, 2022. – pp. 66-68.

9. Romanyuk O.N., Stanislavenko E.G. Chekhmestruk Romanyuk O.V. Method of simplification of Gouraud rendering. Actual problems of modern science and education (part I): materials of the IV International Scientific and Practical Conference in Lviv on January 20-21, 2022. – Lviv: Lviv Scientific Forum, 2022. – pp. 68-70.

10. Romanyuk O.N., Chekhmestruk R.Yu., Stanislavenko E.G., Vintonyuk V.V., Zakharchuk M.D. Relationship between normal vectors to the surface, observer vector and light source vector for rendering problems. Proceedings of the VIII International Scientific and Practical Conference "TOPICAL ISSUES OF MODERN SCIENCE, SOCIETY AND EDUCATION", January 29-31, 2022 Kharkiv.

APPENDIX A

Директор ТОВ «3Д ДЖЕНЕРАЙШН ЮЕЙ»

Швець О.В

«20» грудня 2021 р.

АКТ

впровадження результатів студентської наукової роботи

Комісія у складі: директора Швеця О.В., технічного директора Чехместрука Р.Ю. та інженера-розробника Перуна І.В. склали цей акт про те, що у ТОВ «3д Дженерайшн Юей» впроваджено матеріали студентської наукової роботи на тему « Розробка моделі відбивної здатності поверхні, методів і апаратних засобів для високопродуктивного зафарбовування тривимірних графічних сцен ».

У ТОВ «3д Дженерайшн Юей» впроваджено нові моделі відбивної здатності поверхні, модифікацію метода Гуро.

Використання результатів студентської наувої роботи дозволило підвищити реалістичність та продуктивність формування тривимірних зображень.

[redacted] передано в ТОВ «3д Дженерайшн Юей» програмне забезпечення для формування тривимірних зображень.

Члени комісії Швець О.В.

Чехместук Р.Ю.

Перун І.В.

APPENDIX B

LISTING OF SOFTWARE MODULES

LISTING OF THE AdaptivGourauAndPhong module.java

```
public class AdaptivGourauAndPhong extends idx3d_Rasterizer
{
   private boolean bTypeRasterizer=false; true - Guoro, false - Phong.
   private double m12, m13, m23;
   private double q_1, q_2, q_3;
   private double p_1, p_2, p_3;
   private double opt_m=0.9;
   private double opt_q=0.004;
   private double opt_p=0.008;
   private idx3d_Vector h;
   int i1, i2, i3;
   float ii12r, ii12g, ii12b;
   float ii13r, ii13g, ii13b;
   float ii23r, ii23g, ii23b;
   float liir, liig, liib;
   float riir, riig, riib;
   float lir, lig, lib;
   float rir, rig, rib;
   float i4r, i4g, i4b;
   End for Gouraund
   for Phong
   private idx3d_Vector v4;
   private idx3d_Vector vi13, vi12, vi23;
   private idx3d_Vector lvi, rvi;
   private idx3d_Vector lv, rv;
   End for Phong
   public boolean selectMetod()
   {
      m12=p1.n.x*p2.n.x+p1.n.y*p2.n.y+p1.n.z*p2.n.z;
      m13=p1.n.x*p3.n.x+p1.n.y*p3.n.y+p1.n.z*p3.n.z;
      m23=p2.n.x*p3.n.x+p2.n.y*p3.n.y+p2.n.z*p3.n.z;
      if(Math.abs(m12)>opt_m ||
           Math.abs(m13)>opt_m ||
           Math.abs(m23)>opt_m) return false;
      else
      {
        h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
         q_1=p1.n.x*h.x+p1.n.y*h.y+p1.n.z*h.z;
         q_2=p2.n.x*h.x+p2.n.y*h.y+p2.n.z*h.z;
         q_3=p3.n.x*h.x+p3.n.y*h.y+p3.n.z*h.z;
         if(Math.abs(q_1-q_2)>opt_q ||
              Math.abs(q_1-q_3)>opt_q ||
              Math.abs(q_2-q_3)>opt_q) return false;
         else
         {

p_1=p1.n.x*scene.light[0].v.x+p1.n.y*scene.light[0].v.y+p1.n.z*scene.light[0].v.z;

p_2=p2.n.x*scene.light[0].v.x+p2.n.y*scene.light[0].v.y+p2.n.z*scene.light[0].v.z;

p_3=p3.n.x*scene.light[0].v.x+p3.n.y*scene.light[0].v.y+p3.n.z*scene.light[0].v.z;
            if(Math.abs(p_1-p_2)>opt_p ||
               Math.abs(p_1-p_3)>opt_p ||
               Math.abs(p_2-p_3)>opt_p) return false;
            else return true;
         }
      }
      return true;
   }
   public AdaptivGourauAndPhong()
   {
   }
   protected void initRender()
   {
      bTypeRasterizer=selectMetod();
    if(bTypeRasterizer)
    {
      for Gouraund
      h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
      int dy12 = Math.abs(p1.y-p2.y);
      int dy13 = Math.abs(p3.y-p1.y);
      int dy23 = Math.abs(p3.y-p2.y);
      i1 = calcColor(p1.n2);
      i2 = calcColor(p2.n2);
      i3 = calcColor(p3.n2);
      if (dy12 != 0)
      {
         ii12r = ((float)(idx3d_Color.getRed(i2) - idx3d_Color.getRed(i1))) / (float)dy12;
         ii12g = ((float)(idx3d_Color.getGreen(i2) - idx3d_Color.getGreen(i1))) / (float)dy12;
         ii12b = ((float)(idx3d_Color.getBlue(i2) - idx3d_Color.getBlue(i1))) / (float)dy12;
      }
      if (dy13 != 0)
      {
         ii13r = ((float)(idx3d_Color.getRed(i3) - idx3d_Color.getRed(i1))) / (float)dy13;
         ii13g = ((float)(idx3d_Color.getGreen(i3) - idx3d_Color.getGreen(i1))) / (float)dy13;
         ii13b = ((float)(idx3d_Color.getBlue(i3) - idx3d_Color.getBlue(i1))) / (float)dy13;
      }
      if (dy23 != 0)
      {
         ii23r = ((float)(idx3d_Color.getRed(i3) - idx3d_Color.getRed(i2))) / (float)dy23;
```

```
        ii23g = ((float)(idx3d_Color.getGreen(i3) -
idx3d_Color.getGreen(i2))) / (float)dy23;
        ii23b = ((float)(idx3d_Color.getBlue(i3) -
idx3d_Color.getBlue(i2))) / (float)dy23;
      }
      i4r = (dy12*ii13r) + idx3d_Color.getRed(i1);
      i4g = (dy12*ii13g) + idx3d_Color.getGreen(i1);
      i4b = (dy12*ii13b) + idx3d_Color.getBlue(i1);
      End for Gouraund
    }
    else
    {
      for Phong
       h = idx3d_Vector.add(scene.light[0].v,
scene.defaultCamera.lookat);
      int dx12 = Math.abs(p1.x-p2.x);
      int dx13 = Math.abs(p3.x-p1.x);
      int dx23 = Math.abs(p3.x-p2.x);
      int dy12 = Math.abs(p1.y - p2.y);
      int dy13 = Math.abs(p3.y - p1.y);
      int dy23 = Math.abs(p3.y - p2.y);
      vi13 = new idx3d_Vector((p3.n2.x - p1.n2.x) / dy13,
                        (p3.n2.y - p1.n2.y) / dy13,
                        (p3.n2.z - p1.n2.z) / dy13);
      vi12 = new idx3d_Vector((p2.n2.x - p1.n2.x) / dy12,
                        (p2.n2.y - p1.n2.y) / dy12,
                        (p2.n2.z - p1.n2.z) / dy12);
      vi23 = new idx3d_Vector((p3.n2.x - p2.n2.x) / dy23,
                        (p3.n2.y - p2.n2.y) / dy23,
                        (p3.n2.z - p2.n2.z) / dy23);
      v4 = idx3d_Vector.add(idx3d_Vector.scale(dy12,
vi13), p1.n2);
      End for Phong
    }
   }
   protected void prerenderFirstPart()
   {
    if(bTypeRasterizer)
    {
      for Gouraund
      lir = idx3d_Color.getRed(i1);
      lig = idx3d_Color.getGreen(i1);
      lib = idx3d_Color.getBlue(i1);
      rir = idx3d_Color.getRed(i1);
      rig = idx3d_Color.getGreen(i1);
      rib = idx3d_Color.getBlue(i1);
      if (dx > 0)
      {
        liir = ii12r;
        liig = ii12g;
        liib = ii12b;
        riir = ii13r;
        riig = ii13g;
        riib = ii13b;
      }
      else
      {
        riir = ii12r;
        riig = ii12g;
        riib = ii12b;
        liir = ii13r;
        liig = ii13g;
        liib = ii13b;
      }
      End for Gouraund
    }
      else
    {
      for Phong
      lv = p1.n2;
      rv = p1.n2;
      if (dx > 0)
      {
        lvi = vi12;
        rvi = vi13;
      }
      else
      {
        lvi = vi13;
        rvi = vi12;
      }
      End for Phong
    }
    }
    protected void prerenderSecondPart()
    {
     if(bTypeRasterizer)
     {
      for Gouraund
      if (dx > 0)
      {
        lir = idx3d_Color.getRed(i2);
        lig = idx3d_Color.getGreen(i2);
        lib = idx3d_Color.getBlue(i2);
        rir = i4r;
        rig = i4g;
        rib = i4b;
        liir = ii23r;
        liig = ii23g;
        liib = ii23b;
        riir = ii13r;
        riig = ii13g;
        riib = ii13b;
      }
      else
      {
        rir = idx3d_Color.getRed(i2);
        rig = idx3d_Color.getGreen(i2);
        rib = idx3d_Color.getBlue(i2);
        lir = i4r;
        lig = i4g;
        lib = i4b;
        riir = ii23r;
        riig = ii23g;
        riib = ii23b;
        liir = ii13r;
        liig = ii13g;
        liib = ii13b;
```

```
    }
    End for Gouraund
  }
  else
  {
    for Phong
    if (dx > 0)
    {
       lv = p2.n2;
       rv = v4;
       lvi = vi23;
       rvi = vi13;
    }
    else
    {
       lv = v4;
       rv = p2.n2;
       lvi = vi13;
       rvi = vi23;
    }
    End for Phong
  }
}
protected void prerenderLine()
{
  if(bTypeRasterizer)
  {
    for Gouraund
    renderLine();
    lir += liir;
    lig += liig;
    lib += liib;
    rir += riir;
    rig += riig;
    rib += riib;
    End for Gouraund
  }
  else
  {
    for Phong
    renderLine();
    lv = idx3d_Vector.add(lv, lvi);
    rv = idx3d_Vector.add(rv, rvi);
    End for Phong
  }
}
public String getRasterizerName()
{
    return "Adaptiv Gouraud And Phong";
}
protected void renderLine()
{
  if(bTypeRasterizer)
  {
    for Gouraund
    int cdx = xR - xL;
    if (cdx == 0) return;
    float iir = 0;
    float iig = 0;
    float iib = 0;
    float cir = 0;
    float cig = 0;
    float cib = 0;
    float lastr = lir;
    float lastg = lig;
    float lastb = lib;
    boolean bReset = true;
    for (x=xL;x<xR;x++)
    {
       pos=x+offset;
       if (z<zBuffer[pos])
       {
          if (rir==255 && rig==255 && rib==255 )
          {
             return;
          }
          if (bReset)
          {
             cdx = xR - x;
             if ((xR - x)>1)
             {
                iir = (rir - lir) / cdx;
                iig = (rig - lig) / cdx;
                iib = (rib - lib) / cdx;
                bReset = false;
             }
             cir = lir;
             cig = lig;
             cib = lib;
          }
          screen.p[pos]=0xFF000000                    |
idx3d_Color.getColor(Math.round(cir > 255 ? 255 : cir),
Math.round(cig > 255 ? 255 : cig), Math.round(cib > 255 ?
255: cib));
          screen.p[pos]=0xFF000000                    |
idx3d_Color.getColor(255, 255, 255);
          zBuffer[pos]=z;
          if (useIdBuffer) idBuffer[pos]=currentId;
          cir += iir;
          cig += iig;
          cib += iib;
       }
       else
          bReset = true;
    }
    z += dz;
    End for Gouraund
  }
  else
  {
    for Phong
    int cdx = xR - xL;
    if (cdx==0)
       return;
    for (int x=xL; x<xR; x++)
    {
       pos=x+offset;
       if (z<zBuffer[pos])
```

```
        {
          idx3d_Vector cv = idx3d_Vector.GetVectorOnPos(lv, rv, xL, xR, x);
          float angle = idx3d_Vector.angle(scene.light[0].v, cv);
          if (angle < 0) angle = 0;
          c = idx3d_Color.scale(color, (int)(angle*255));
          angle = idx3d_Vector.angle(h, cv);
          if (angle < 0) angle = 0;
          angle = (float)Math.pow(angle, n);
          int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
          c = idx3d_Color.add(c, c2);
          screen.p[pos]=0xFF000000|c;
          zBuffer[pos]=z;
          if (useIdBuffer) idBuffer[pos]=currentId;
        }
        z+=dz;
        nx+=dnx;
        ny+=dny;
      }
      End for Phong
    }
  }
  for Gouraund
  private int calcColor(idx3d_Vector vector)
  {
    float angle = idx3d_Vector.angle(scene.light[0].v, vector);
    if (angle < 0) angle = 0;
    int c = idx3d_Color.scale(color, (int)(angle*255));
    angle = idx3d_Vector.angle(h, vector);
    if (angle < 0) angle = 0;
    angle = (float)Math.pow(angle, n);

    int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
    c = idx3d_Color.add(c, c2);
    return c;
  }
  End for Gouraund
  protected void beforeRenderTriangle()
  {
  }
}
```

LISTING OF the NormalVector module.java

```
public class NormalVector extends idx3d_Rasterizer
{
  private idx3d_Vector h, v4;
  private idx3d_Vector vi13, vi12, vi23;
  private idx3d_Vector lvi, rvi;
  private idx3d_Vector lv, rv;
  int n_2;
  private idx3d_Vector N_2 ;
  int Nl,Nr, count_n;
  private idx3d_Vector dN_i,Na,Nb;
  private RET Ret[];
  public NormalVector()
  {
  }
  protected void initRender()
  {
    h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
    int dx12 = Math.abs(p1.x-p2.x);
    int dx13 = Math.abs(p3.x-p1.x);
    int dx23 = Math.abs(p3.x-p2.x);
    int dy12 = Math.abs(p1.y-p2.y);
    int dy13 = Math.abs(p3.y-p1.y);
    int dy23 = Math.abs(p3.y-p2.y);
    vi13 = new idx3d_Vector((p3.n2.x - p1.n2.x) / dy13,
                (p3.n2.y - p1.n2.y) / dy13,
                (p3.n2.z - p1.n2.z) / dy13);
    vi12 = new idx3d_Vector((p2.n2.x - p1.n2.x) / dy12,
                (p2.n2.y - p1.n2.y) / dy12,
                (p2.n2.z - p1.n2.z) / dy12);
    vi23 = new idx3d_Vector((p3.n2.x - p2.n2.x) / dy23,
                (p3.n2.y - p2.n2.y) / dy23,
                (p3.n2.z - p2.n2.z) / dy23);
    v4 = idx3d_Vector.add(idx3d_Vector.scale(dy12, vi13), p1.n2);
  }
  protected void prerenderFirstPart()
  {
    lv = p1.n2;
    rv = p1.n2;
    if (dx > 0)
    {
      lvi = vi12;
      rvi = vi13;
    }
    else
    {
      lvi = vi13;
      rvi = vi12;
    }
  }
  protected void prerenderSecondPart()
  {
    if (dx > 0)
    {
      lv = p2.n2;
      rv = v4;
      lvi = vi23;
      rvi = vi13;
    }
    else
    {
      lv = v4;
      rv = p2.n2;
      lvi = vi13;
      rvi = vi23;
    }
```

```
    }
    protected void prerenderLine()
    {
       norm();
       renderLine();
       lv = idx3d_Vector.add(lv, lvi); increment of vectors
horizontally in levo
       rv = idx3d_Vector.add(rv, rvi); increment of the
vector horizontally to the right
    }
     protected idx3d_Vector dN(int n, int k,idx3d_Vector
N_p,idx3d_Vector N_g )//for boundary irregular vectors
     {
      int n2=1, k2=1;
      n2<<=n;
      k2<<=k;
      idx3d_Vector Ngp= idx3d_Vector.sub(N_g,N_p);
      return new
idx3d_Vector((n2*Ngp.x/k2),(n2*Ngp.y/k2),(n2*Ngp.z/
k2));
     }
      protected void norm()
    {
       idx3d_Vector Nbuf12,N12;
       Na=lv; Nb=rv;
       int st=Math.abs(xR - xL);
       for(n_2=1;n_2<st;) // search for the number 2 in the
power of k
       {
        n_2<<=1;
       }
       if(st!=n_2)
       { //finding the finite vector
        idx3d_Vector buf= idx3d_Vector.sub(Nb,Na);
        N_2=new
idx3d_Vector((buf.x*n_2/st),(buf.y*n_2/st),(buf.z*n_2/st
));
         N_2.normalize();
         Nb=N_2;
       }
       double cos_W = idx3d_Vector.angle(Na,Nb);
       double z_12=Math.pow((2*(1+cos_W)),0.5);
       Nbuf12 = idx3d_Vector.add(Na,Nb);
       N12= new
idx3d_Vector((float)(Nbuf12.x/z_12),(float)(Nbuf12.y/z
_12),(float)(Nbuf12.z/z_12) );
N12.normalize();//middle
 double di=0;
       double zi=z_12;
       idx3d_Vector Ni=N12;
       Nl=xL;
       count_n=0;
       Ret=new RET[n_2];
       RET Wet= GetSegment( Na, Ni,Nb
,n_2/2,zi,n_2/2); search for intervals
       Ret[count_n]=Wet;
      for(int k=0;k<count_n;k++)
      {
       int min=n_2,point=0;
       for(int i=k;i<=count_n;i++)
       {
        int retx=Ret[i]. GetX();
        if(min>retx)
        {
          min=retx;
          point=i;
        }
       }
       RET buf= Ret[k];
       Ret[k]=Ret[point];
       Ret[point]=buf;
      }
      if(count_n>2)
       Ret[count_n+1]= new RET(Nb,n_2);
    }
    public RET GetSegment(idx3d_Vector
NA,idx3d_Vector Ni,idx3d_Vector NB , int n_2,double
zi,int R)
    {
        double di=1-(Math.pow((2+zi),0.5)/2); maximum
oshibka
        if(di<0.02 )
        {
         return new RET(Ni,R);
        }
        double zi1 = Math.pow((2+zi),0.5);
        idx3d_Vector  NLi= idx3d_Vector.add(NA,Ni);
        idx3d_Vector  Li=new
idx3d_Vector((float)(NLi.x/zi1),(float)(NLi.y/zi1),(float)
(NLi.z/zi1));
        idx3d_Vector  NRi= idx3d_Vector.add(NB,Ni);
        idx3d_Vector  Ri=new
idx3d_Vector((float)(NRi.x/zi1),(float)(NRi.y/zi1),(float)
(NRi.z/zi1));
        int dN=n_2/2;
       if(dN>=2)
        {
         Ret[count_n++]= GetSegment(NA, Li,Ni , dN,
zi1,R-dN);//vlevo
         Ret[count_n++]=GetSegment(Ni, Ri,NB , dN,
zi1,R+dN);//right
        }
     return new RET(Ni,R);
    }
    public String getRasterizerName()
    {
       return "NormalVector";
    }
    protected void renderLine()
    {
       int cdx = Math.abs( xR - xL);
      if (cdx==0)
          return;
       int i=0,k=0,xi=0;
       idx3d_Vector Np=Na,Ng;
       idx3d_Vector dN,Ni;
       Ng=Ret[k]. GetVector();
       dN= idx3d_Vector.sub(Ng,Np);
```

```
dN.x*=n_2/cdx;
dN.y*=n_2/cdx;
dN.z*=n_2/cdx;
for (int x=xL; x<xR; x++)
{
 if(count_n>2)
  if(Ret[k]. GetX()<xi )
  {
   Np=Ret[k]. GetVector();
   Ng=Ret[k+1]. GetVector();
   i=0;
   k++;
   dN= idx3d_Vector.sub(Ng,Np);
   dN.x*=n_2/cdx;
   dN.y*=n_2/cdx;
   dN.z*=n_2/cdx;
  }
   dN.x*=i;
   dN.y*=i;
   dN.x*=i;
  Ni=idx3d_Vector.add(Np,dN);
  i++;
  xi++;
  pos=x+offset;
  if (z<zBuffer[pos])
          {
           idx3d_Vector cv=Ni;
          float angle =
idx3d_Vector.angle(scene.light[0].v, cv);
          if (angle < 0) angle = 0;
          c = idx3d_Color.scale(color, (int)(angle*255));
          angle = idx3d_Vector.angle(h, cv);
          if (angle < 0) angle = 0;
          angle = (float)Math.pow(angle, n);
          int c2 = idx3d_Color.scale(0xFFFFFF,
(int)(angle*255));
          c = idx3d_Color.add(c, c2);
          screen.p[pos]=0xFF000000|c;
          zBuffer[pos]=z;
          if (useIdBuffer) idBuffer[pos]=currentId;
         }
         z+=dz;
         nx+=dnx;
         ny+=dny;
       }
   }
  protected void beforeRenderTriangle()
  {
  }
}
```

LISTING OF THE GouraudRasterizer4 module.java

```
public class GouraudRasterizer4 extends idx3d_Rasterizer
{
   private idx3d_Vector h;
   int i1, i2, i3;
   float ii12r, ii12g, ii12b;
   float ii13r, ii13g, ii13b;
   float ii23r, ii23g, ii23b;
   float liir, liig, liib;
   float riir, riig, riib;
   float lir, lig, lib;
   float rir, rig, rib;
   float i4r, i4g, i4b;
   double Pr[];
   double Pg[];
   double Pb[];
   private idx3d_Vector vA, vB, vC,nA, nB, nC ;
   double X1,X2,X3;
   double Y1,Y2,Y3;
   double X12,X23,X13;
   double Y12,Y23,Y13;
   public GouraudRasterizer4()
   {
   }
   protected void initRender()
   {
      h = idx3d_Vector.add(scene.light[0].v, scene.defaultCamera.lookat);
      vA = new idx3d_Vector(p1.n2);
      vB = new idx3d_Vector(p2.n2);
      vC = new idx3d_Vector(p3.n2);
      double cos_1 = idx3d_Vector.angle(vA, h);//  cos(Na H)
      double cos_2 = idx3d_Vector.angle(vB, h);//  cos(Nb H)
      double cos_3 = idx3d_Vector.angle(vC, h);//  cos(Nc H)
      double cos_4 = idx3d_Vector.angle(vA, vB);//  cos(Na Nb)
      double cos_5 = idx3d_Vector.angle(vA, vC);//  cos(Na Nc)
      double cos_6 = idx3d_Vector.angle(vC, vB);//  cos(Nc Nb)
      double t1=(cos_1*cos_4 - cos_2)/((cos_4 - 1)*(cos_2 + cos_1)) ;
      double t2=(cos_1*cos_5 - cos_3)/((cos_5 - 1)*(cos_3 + cos_1)) ;
      double t3=(cos_2*cos_6 - cos_3)/((cos_6 - 1)*(cos_3 + cos_2)) ;
      X1=Math.abs(p1.x+t1*(p2.x-p1.x));
      X2=Math.abs(p1.x+t2*(p3.x-p1.x));
      X3=Math.abs(p2.x+t3*(p3.x-p2.x));
      Y1=Math.abs(p1.y+t1*(p2.y-p1.y));
      Y2=Math.abs(p1.y+t2*(p3.y-p1.y));
      Y3=Math.abs(p2.y+t3*(p3.y-p2.y));
       ////////////////////////////
       int   v1 = calcColor(p1.n2);
       int   v2 = calcColor(p2.n2);
       int   v3 = calcColor(p3.n2);
       ////////////////////////////
      double Fr[][]={{0,0,0,0,0,1,idx3d_Color.getRed(v1)},
                {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getRed(v2)},
        {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getRed(v1)+idx3d_Color.getRed(v2))/2},{X3*X3,X3*Y3,
        Y3*Y3,X3,Y3,1,idx3d_Color.getRed(v3)},X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,
        (idx3d_Color.getRed(v1)+idx3d_Color.getRed(v3))/2},{(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+
        Y3)/4,(X2+X3)/2,(Y2+Y3)/2,1,(idx3d_Color.getRed(v2)+idx3d_Color.getRed(v3))/2}
      };
      Kramer sist= new Kramer();
      Pr=new double[6];
      Pr= sist. GetKramer(Fr,6,7);
```

```
    double Fg[][]={{0,0,0,0,0,1,idx3d_Color.getGreen(v1)},
            {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getGreen(v2)},
      {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getGreen(v1)+idx3d_Color.getGreen(v2))/2},{X3*X3,X3*
      Y3,Y3*Y3,X3,Y3,1,idx3d_Color.getGreen(v3)},          {X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,
      (idx3d_Color.getGreen(v1)+idx3d_Color.getGreen(v3))/2},
      {(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+Y3)/4,(X2+X3)/2,(Y2+Y3)/2,1,(idx3d_Color.getGreen(
      v2)+idx3d_Color.getGreen(v3))/2}
    };
    Pg=new double[6];
    Pg= sist. GetKramer(Fg,6,7);
    double Fb[][]={{0,0,0,0,0,1,idx3d_Color.getBlue(v1)},
            {X2*X2,X2*Y2,Y2*Y2,X2,Y2,1,idx3d_Color.getBlue(v2)},
      {X2*X2/4,X2*Y2/4,Y2*Y2/4,X2/2,Y2/2,1,(idx3d_Color.getBlue(v1)+idx3d_Color.getBlue(v2))/2},
            {X3*X3,X3*Y3,Y3*Y3,X3,Y3,1,idx3d_Color.getBlue(v3)},
{X3*X3/4,X3*Y3/4,Y3*Y3/4,X3/2,Y3/2,1,(idx3d_Color.getBlue(v1)+idx3d_Color.getBlue(v3))/2},
      {(X2+X3)*(X2+X3)/4,(X2+X3)*(Y2+Y3)/4,(Y2+Y3)*(Y2+Y3)/4,(X2+X3)/2,(Y2+Y3)/2,1,(idx3d_Color.getBlue(v
      2)+idx3d_Color.getBlue(v3))/2}
    };
    Pb=new double[6];
    Pb= sist. GetKramer(Fb,6,7);
  protected void prerenderFirstPart()
  {
  }
  protected void prerenderSecondPart()
  {
  }
  protected void prerenderLine()
  {
    renderLine();
  }
  public String getRasterizerName()
  {
    return " --------- ";
  }
  protected void renderLine()
  {
    int cdx = xR - xL;
    if (cdx == 0) return;
    for (x=xL;x<xR;x++)
    {

      double Ixyr=Pr[5]*x*x+Pr[4]*x*y+Pr[3]*y*y+Pr[2]*x+Pr[1]*y+Pr[0];
      double Ixyg=Pg[5]*x*x+Pg[4]*x*y+Pg[3]*y*y+Pg[2]*x+Pg[1]*x+Pg[0];
      double Ixyb=Pb[5]*x*x+Pb[4]*x*y+Pb[3]*y*y+Pb[2]*x+Pb[1]*y+Pb[0];

     float   cir = (float)Ixyr;
     float   cig = (float)Ixyg;
     float   cib = (float)Ixyb;

      if(cir<0){ cir=0;}
      if(cig<0){ cig=0;}
      if(cib<0){ cib=0;}
      pos=x+offset;
      if (z<zBuffer[pos])
      {
        screen.p[pos]=0xFF000000 | idx3d_Color.getColor(Math.round(cir > 255 ? 255 : cir), Math.round(cig > 255 ?
      255 : cig), Math.round(cib > 255 ? 255: cib));
        zBuffer[pos]=z;
        if (useIdBuffer) idBuffer[pos]=currentId;
```

```
            }
        }
    }
    private int calcColor(idx3d_Vector vector)
    {
        float angle = idx3d_Vector.angle(scene.light[0].v, vector);
        if (angle < 0) angle = 0;
        int c = idx3d_Color.scale(color, (int)(angle*255));
        angle = idx3d_Vector.angle(h, vector);
        if (angle < 0) angle = 0;
        angle = (float)Math.pow(angle, n);
        int c2 = idx3d_Color.scale(0xFFFFFF, (int)(angle*255));
        c = idx3d_Color.add(c, c2);
        return c;
    }
    protected void beforeRenderTriangle()
    {
    }
}
```