4. PERSPECTIVE-CORRECT FORMATION OF GRAPHIC IMAGES

Oleksandr Romanyuk, Oksana Romaniuk

DOI: 10.31274/isudp.2024.151.04

The article presents advancements in color reproduction techniques, enhancing the realism of graphic scenes. It introduces novel methods for accurate shading of three-dimensional objects, improves the Barenbrug perspective-correct texturing method, and proposes quadratic approximation for perspective-correct texturing, reducing computation time and error.

4.1. FEATURES OF FORMING REALISTIC THREE-DIMENSIONAL GRAPHIC IMAGES

The scope of computer graphics, which is the main means of communication between humans and computers, is constantly expanding, as graphic images are the most visual and adequately reproduce real objects and processes. The main task of modern computer graphics is to synthesize realistic three-dimensional images that reproduce real-world objects to the maximum extent possible. At the same time, it is important to achieve a graphic scene generation performance acceptable for a given application. Image synthesis involves the execution of a certain sequence of specialized stages, which together constitute a 3D graphics pipeline [1, 2].

The process of creating a three-dimensional image can be divided into the following main stages: the stage of describing a three-dimensional scene, the stage of geometric transformations, the stage of rendering, or visualization, and the stage of displaying the image on a monitor or printer. Each stage, in turn, contains several stages.

At the stage of describing a three-dimensional scene, the constituent objects of the scene, their main characteristics, state and relative position, and the strategy for further actions are determined. In the object space, three-dimensional objects, materials, light sources, virtual cameras, and additional tools for modeling special effects, including atmospheric phenomena, are operated with.

Synthesis of graphic images is a complex task, so in most cases, a graphic scene is decomposed into its components. Surfaces, including curved ones, are approximated by a polygonal mesh, which in most cases includes triangles. At this stage of computer graphics, the representation of surfaces by triangles is one of the ways to create real-time dynamic images and processes in an interactive mode.

At the stage of geometric transformations, in addition to tessellation, affine transformations are also performed, such as rotation, scaling, and displacement. After that, the triangles are merged into a solid wireframe model. To take into account the curvature of the surfaces, vectors of normals to the vertices of the constituent triangles are determined, for which geometric parameters are predefined.

When generating graphic images, it is necessary to take into account a number of cameras, which implies the inclusion of the corresponding stage in the graphic pipeline. The position of the cameras makes it possible to exclude invisible objects and surfaces, as well as objects that are located outside the camera's field of view, and, as a result, reduce computational costs.

At the final stage of the geometric transformation stage, a projection of the three-dimensional scene onto the visualization plane is formed, i.e. the output is a set of polygons in screen coordinates.

Projection is the mapping of points specified in a coordinate system with dimension E to points in a system with a smaller dimension. In computer graphics, we mainly consider projections of three-dimensional images onto a two-dimensional picture plane. Flat geometric projections are divided into two main classes: central and parallel. The difference between them is determined by the ratio between the center of the projection and the projection plane. If the distance between them is finite, then the projection will be central, but if it is infinite, then the projection will be parallel. In real space, the reflection of rays from objects is perceived at the point of the observer's location, i.e., according to the principle of central projection [1]. Correct color reproduction takes place provided that the components of the color intensities of the corresponding surface points in the world (object) and screen coordinate systems coincide.

Existing shading methods do not provide correct color reproduction when shading three-dimensional graphic objects, since they do not take into account the z-coordinate in perspective projection. In this chapter, we develop the theoretical foundations for correct color reproduction in perspective projection.

At the final visualization (rendering) stage, pixels of the image of a three-dimensional scene are formed on the screen, which involves calculating not only their addresses but also color intensities. The rendering stage is the most time-consuming, since all operations are performed at the pixel and subpixel levels, unlike the geometric transformation stage, where only the vertices of polygons were processed.

At the rasterization stage, for a selected polygon that falls within the camera space, all its internal points are searched, which involves determining their addresses in the on-screen coordinate system. For the selected point, normalized vectors to the object surface, light source, and observer are calculated, as well as auxiliary vectors depending on the lighting model selected. These vectors are then used to determine the color intensities of the image points. The intensity value of the original light source is used or the color intensity value is sampled from the texture. If necessary, texture filtering or anti-aliasing is performed to eliminate artifacts.

When forming images of relief surfaces, the normal vector to the surface is perturbed by one of the known methods.

Additional procedures can be applied to the final image to simulate various natural phenomena.

At the post-processing stage, pixel visibility is checked once again and the finished image is displayed on a screen or printer.

In the course of their evolution, GPUs have gone from graphics pipelines with hard logic to programmable computing environments. Over the course of several generations, individual stages of pipelines were gradually replaced by programmable devices called shaders [1] A shader is a program that performs a certain stage of the graphics pipeline to determine the final parameters of an object or image. There are three types of shaders: vertex, pixel, and geometric, which process polygon vertices, pixels, and polygons, respectively. Vertex shaders determine the order of transformation of polygon vertices. Pixel shaders are used to dynamically change the properties of individual pixels. They calculate the color of a single pixel based on the input data received from the vertex raider and the specified lighting parameters. Geometric shaders were first used in NVidia 8 series video cards. They process primitives and can create new vertices and generate new primitives without using CPU resources.

A special place among the numerous stages of the graphics pipeline is occupied by the lighting stage, the texture mapping stage, and the painting stage. These three stages are directly responsible for the realism of objects.

Creating realistic images in computer graphics involves accurately conveying the external characteristics of natural objects in their computer model. The features of a realistic image are depth rendering, which allows us to determine the distance between objects, lighting of objects, as well as to evaluate the shape, color, and material of objects, shadow casting, and other optical effects.

One of the ways to increase the realism of an image is to display fine details, irregularities, and relief on the surfaces of objects that make up a graphic scene. This can be achieved by changing the geometry of the object [3]. The geometric model of the object is divided into a large number of low-level polygons, the vertices of which are shifted to the appropriate distance and in the appropriate direction. The larger the number of polygons and the smaller their size, the more realistic the object looks. This approach allows to display the relief and irregularities on the surface quite realistically, but is characterized by high computational complexity and low performance. It is difficult to use it to display such fine details as sand, skin pores, wood structure, etc.

An alternative to changing the geometry of an object is texture overlay [1, 2], which is widely used in graphics systems. A texture is a raster image that is superimposed on the surface of a polygon, which make up 3D models, to give it color or the illusion of relief [4]. Textures allow to simulate a variety of materials, complex surface structures (porous, with cracks, etc.) that are difficult to implement with a set of polygons. The quality of the texture surface is determined by the texture pixels — the number of pixels per minimum texture unit. Since a texture is an image, its resolution and format play a crucial role, which ultimately affects the quality of the synthesized graphic image. Therefore, in this chapter, we propose methods to improve the efficiency of perspective-correct texturing.

4.2. PERSPECTIVE-CORRECT COLOR REPRODUCTION WHEN INTERPOLATING COLOR INTENSITIES

In the classical implementation of Gouraud shading [1–4], the z-coordinate is taken into account only at the vertices of the triangle when determining their color intensities. Subsequently, the z-coordinate is used only to remove invisible surfaces. Thus, the perspective of the object is not taken into account when Gouraud coloring.

Fig. 4.1 shows a line segment $A'B'$, projected onto an screen plane. When using linear interpolation to calculate the color intensities along the edge $A'B'$, which is located in the world coordinate system, the color intensity at point Q' $(A'Q' = B'Q')$ is equal to $I_{Q'} = (I_{A'} + I_{B'})/2$. Since $BQ \neq QA$, this regularity at the point Q will be broken (Fig. 4.2).

The example above shows that there is a violation of the regularity of color change between a real object and its image on the screen.

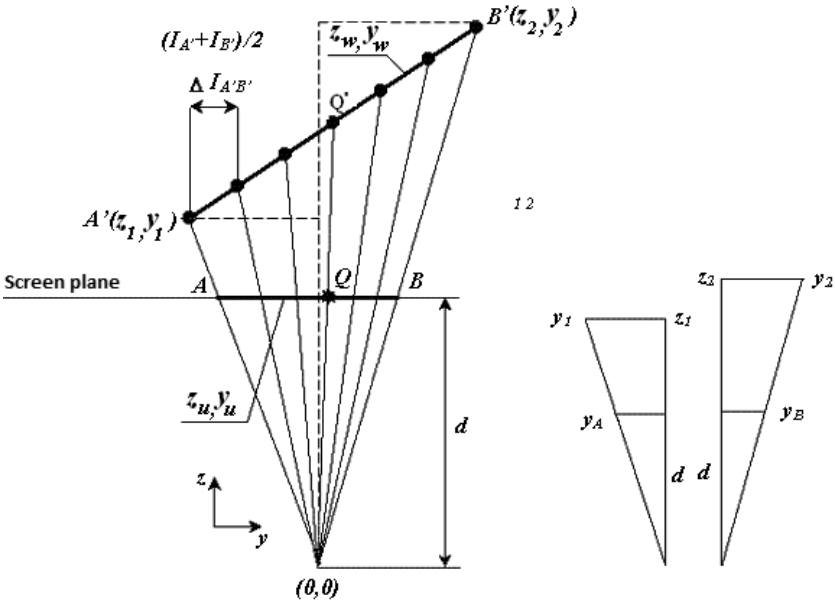

Fig. 4.1. Perspective vector projection on the screen plane

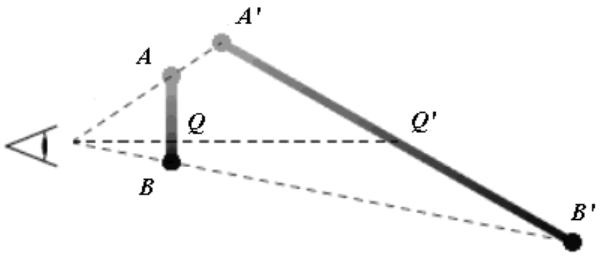

Fig. 4.2. Color mismatch for segments in the screen and world coordinate systems

This subsection discusses the issues of correct coloring of three-dimensional objects using the Gouraud method. According to the Gouraud method [1–4], the intensity of a color along a segment AB in the world coordinate system is determined by the formula:

$$I_w = I_A + w \cdot (I_B - I_A).$$

In the screen coordinate system:

$$I_v = I_A + v \cdot (I_B - I_A).$$

Since $v \neq w$, proportionality is violated when reproducing colors in all cases when AB and $A'B'$ are not parallel (Fig. 4.2). Let's find the relationship between the variables *v* and *w*.

Let the observation point be located at a distance d from the screen plane at the origin of the *yoz* coordinate system (Fig. 4.1).

For the segment $A'B'$, write the parametric equations

$$z_w = z_1 + w \cdot (z_2 - z_1),\ y_w = y_1 + w \cdot (y_2 - y_1).$$

For the segment AB $y_u = y_A + u \cdot (y_A - y_B)$.

Given the similarities of the corresponding triangles (Fig. 4.1), we can write

$$\frac{d}{z_1} = \frac{y_A}{y_1}\ ;\quad \frac{d}{z_2} = \frac{y_B}{y_2}\ ;\quad \frac{d}{z_w} = \frac{y_u}{y_w}.$$

From the last expressions we find

$$y_1 = \frac{z_1 \cdot y_A}{d}\ ;\qquad y_2 = \frac{z_2 \cdot y_B}{d};\qquad z_w = \frac{y_w}{y_u} d.$$

Taking into account that $y_w = y_1 + w \cdot (y_2 - y_1)$ and the values for y_1, y_2

$$z_w = \frac{(y_1 + w \cdot (y_2 - y_{1)}) \cdot d}{y_u} = \frac{(\frac{z_1 \cdot y_A}{d} + w \cdot (\frac{z_2 \cdot y_B}{d} - \frac{z_1 \cdot y_A}{d})) \cdot d}{y_A + u \cdot (y_B - y_A)} =$$

$$= \frac{(z_1 \cdot y_A + (z_2 \cdot y_B - z_1 \cdot y_A)) \cdot w}{y_A + u \cdot (y_B - y_A)}.$$

The right-hand side of the last expression is equivalent to the equivalent value, which is equal to $z_1 + w \cdot (z_2 - z_1)$. We get the expression

$$\frac{(z_1 \cdot y_A + (z_2 \cdot y_B - z_1 \cdot y_A)) \cdot w}{y_A + u \cdot (y_B - y_A)} = z_1 + w \cdot (z_2 - z_1).$$

After the transformations, we find that

$$w = \frac{u \cdot z_1}{z_2 - u \cdot (z_2 - z_1)}.$$

The last expression relates the parametric variables *w*, *u* using the *z*-coordinates of the edge $A'B'$ in the world system.

According to the Gouraud method, the color intensity along an edge is determined by the formula: $I_w = I_A + w \cdot (I_B - I_A)$. Substituting the *w* values into the above expression, we obtain

$$I_w = I_A + \frac{u \cdot z_1}{z_2 - u \cdot (z_2 - z_1)} (I_B - I_A) .$$

Let's estimate the error that occurs when the perspective correction of color intensity is not taken into account.

$$\Delta I = I_A + (I_B - I_A) \cdot \frac{u \cdot z_1}{z_2 - u \cdot (z_2 - z_1)} - I_A - (I_B - I_A) \cdot u =$$

$$= (I_B - I_A) \cdot (u - \frac{u \cdot z_1}{z_2 - u(z_2 - z_1)}) = (I_B - I_A) \cdot u \cdot (1 - \frac{1}{\frac{z_2}{z_1} + u(1 - \frac{z_2}{z_1})}). \qquad (4.1)$$

Fig. 4.3 shows a graph of the change $\Delta = u \cdot (1 - \frac{1}{\frac{z_2}{z_1} + u \cdot (1 - \frac{z_2}{z_1})})$ in the multiplier for different ratios z_2 / z_1.

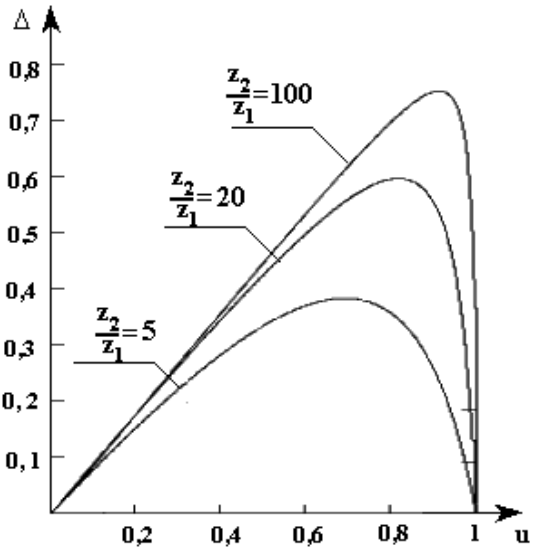

Fig. 4.3. Dependence Δ on u

When $z_1 > z_2$ the graph of dependence is similar to the one above with the difference that $\Delta < 0$, and the maximum value occurs at $1 - u$. Let determine the maximum values Δ for different ratios z_2 / z_1. To do this, find the derivative of Δ and set it to zero. We get two roots

$$\frac{\sqrt{\frac{z_2}{z_1}-\frac{z_2}{z_1}}}{1-\frac{z_2}{z_1}},\quad \frac{-\sqrt{\frac{z_2}{z_1}-\frac{z_2}{z_1}}}{1-\frac{z_2}{z_1}}.$$

Researchers have shown that the first root determines the argument at which the function has a maximum value for $z_2 \geq z_1$, and the second root determines the maximum value for $z_1 > z_2$, and the maximum values of Δ coincide, so we will give only one expression for the first root.

$$\max\Delta = \frac{\sqrt{\frac{z_2}{z_1}-\frac{z_2}{z_1}}}{1-\frac{z_2}{z_1}} \cdot \left(1-\frac{1}{\frac{z_2}{z_1}+\left(1-\frac{z_2}{z_1}\right)\cdot\frac{\sqrt{\frac{z_2}{z_1}-\frac{z_2}{z_1}}}{1-\frac{z_2}{z_1}}}\right) = \frac{\left(\frac{z_2}{z_1}-\sqrt{\frac{z_2}{z_1}}\right)\left(\sqrt{\frac{z_2}{z_1}}-1\right)}{\sqrt{\frac{z_2}{z_1}-1}}.$$

The graph of the maximum value Δ versus the ratio z_2 / z_1 is shown in Fig. 4.4 for different ratios z_2 / z_1.

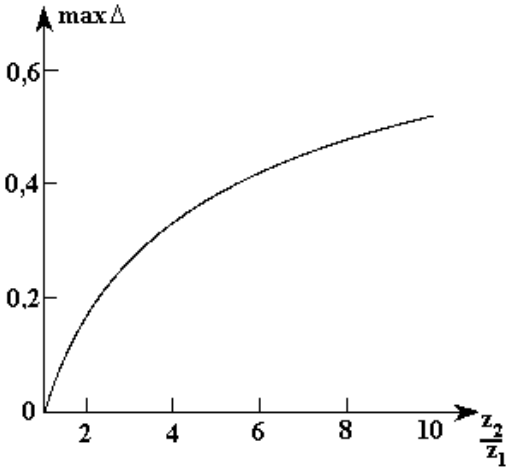

Fig. 4.4. Graph of dependence of the maximum value Δ on the ratio z_2 / z_1

The graph shows that not taking into account the perspective of a graphic object leads to visual differences between images. It should be noted that

in expression (4.1), the maximum value of the difference $I_B - I_A$ is equal to I_B and occurs when the intensity at a point B is equal to the maximum possible.

4.3. DETERMINATION OF VECTORS TO TAKE INTO ACCOUNT THE PERSPECTIVE OF A THREE-DIMENSIONAL SCENE

In the classical implementation of Phong shading [1–4], there are no geometric transformations that are inherent in perspective projections, which certainly affects the realism of reproducing graphic scenes. If Gouraud shading requires perspective correction of the color intensity of points, then when shading by Phong, it is necessary to correct the normal vectors and subsequently determine the color of the pixels. When correcting the normal vectors, it is necessary to take into account all the stages of geometric transformations established by the OpenGL and DirectX 10 standards.

We will consider the right-handed coordinate system [1], which is used in most computer graphics programs. Fig. 4.5 shows the features of perspective projection in three-dimensional space.

The volume of observation is set between the near and far cutoff planes, which are perpendicular to the axis Z_w. The display of the scene projection includes only those objects that are inside the cut-off pyramid. According to the OpenGL standard [1], the observation volume is displayed in a symmetric normalized cube (Fig. 4.6 — left image).

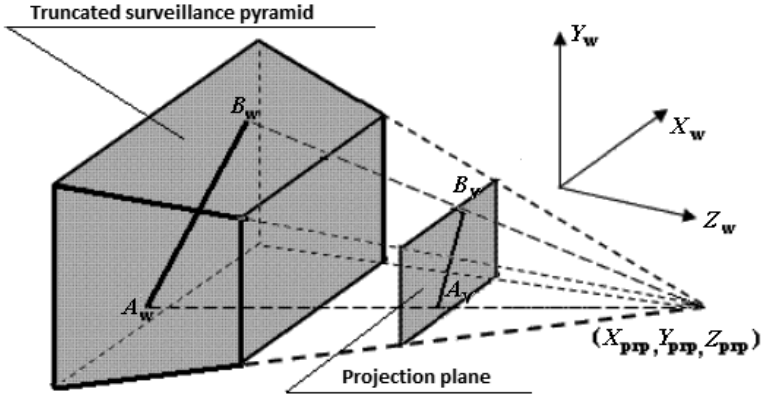

Fig. 4.5. Pyramidal observation volume of the perspective projection

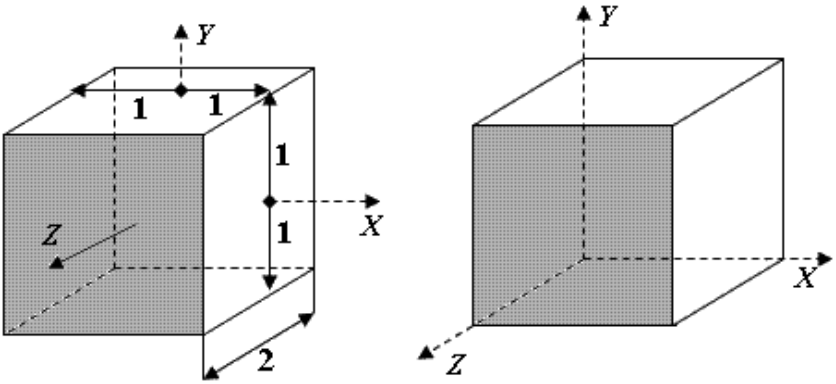

Fig. 4.6. Offset of the origin of the coordinate system of the normalized observation volume

Let a line segment $A_w B_w$ be given in the world coordinate space (Fig. 4.5). In the plane of observation, it corresponds to the segment $A_v B_v$.

The normalized coordinates $X_{norm}, Y_{norm}, Z_{norm}$ of the point X_w, Y_w, Z_w in the world coordinate system are found by the homogeneous coordinates using the formula [2]

$$X_{norm} = \frac{-Z_n \cdot S_x \cdot X_w + S_x (X_m - X_n)/2}{-Z_w},$$

$$Y_{norm} = \frac{-Z_n \cdot S_y \cdot Y_w + S_y (Y_m - Y_n)/2}{-Z_w},$$

$$Z_{norm} = \frac{S_z \cdot Z_w + t_z}{-Z_w},$$

where $S_x = \frac{2}{X_m - X_n}$, $S_x = \frac{2}{Y_m - Y_n}$, $S_z = \frac{Z_m - Z_n}{Z_m - Z_n}$, $t_z = \frac{2Z_m \cdot Z_n}{Z_m - Z_n}$.

The contents of the normalized observation volume must be converted to screen coordinates. For simplicity, let's move the origin of the coordinate system of the normalized cube from its center to the leftmost corner, that is, shift all coordinates by one (Fig.4.6). Then the transformation matrix for the coordinates $(X_n + 1)$, $(Y_n + 1)$, $(Z_n + 1)$ will have the following form [3]

$$\begin{bmatrix} \frac{X_{vm}-X_{vn}}{2} & 0 & 0 & SM_{vx} \\ 0 & \frac{Y_{vm}-Y_{vn}}{2} & 0 & SM_{vy} \\ 0 & 0 & \frac{Z_{vm}-Z_{vn}}{2} & SM_{vz} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In most cases, the zero depth value is correlated to the screen area, for which $SM_{vz}=0$. The above transformations can be written in matrix form

$$\begin{bmatrix} X_v \\ Y_v \\ Z_v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{X_{vm}-X_{vn}}{2} & 0 & 0 & SM_{vx} \\ 0 & \frac{Y_{vm}-Y_{vn}}{2} & 0 & SM_{vy} \\ 0 & 0 & \frac{Y_{vm}-Y_{vn}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times$$

$$\times \left[\frac{-1}{Z_w} \times \begin{bmatrix} \frac{2Z_n}{X_m-X_n} & 0 & \frac{X_m+X_n}{X_m-X_n} & 0 \\ 0 & \frac{2Z_n}{Y_m-Y_m} & \frac{Y_m+Y_n}{Y_m-Y_n} & 0 \\ 0 & 0 & \frac{Z_n+Z_m}{Z_n-Z_m} & \frac{2Z_nZ_m}{Z_n-Z_m} \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right]$$

From the latter system, we find that

$$\begin{bmatrix} X_v \\ Y_v \\ Z_v \end{bmatrix} = \begin{bmatrix} \frac{(X_{vm}-X_{vn})\cdot(X_n\cdot Z_w+Z_n\cdot X_w)-SM_{vx}\cdot(X_m-X_n)\cdot Z_w}{Z_w(X_n-X_m)} \\ \frac{(Y_{vm}-Y_{vn})\cdot(Y_n\cdot Z_w+Z_n\cdot Y_w)-SM_{vy}\cdot(Y_m-Y_n)\cdot Z_w}{Z_w(Y_n-Y_m)} \\ Z_n\cdot(Z_{vm}-Z_{vn})\frac{(Z_w-Z_m)}{Z_w(Z_n-Z_m)} \end{bmatrix}.$$

The v-index determines whether the point belongs to the on-screen coordinate system, and the w-index — to the world coordinate system. From

the resulting system of equations, it is easy to find the X_w, Y_w, Z_w point's coordinates in the world coordinate system.

$$\begin{bmatrix} X_{Aw} \\ Y_{Aw} \\ Z_{Aw} \end{bmatrix} = \begin{bmatrix} \dfrac{(SM_{vx} - X_{Av}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn})}{(X_{vm} - X_{vn}) Z_n} \\ \dfrac{(SM_{vy} - Y_{Av}) \cdot (Y_m - Y_n) - Y_n \cdot (Y_{vm} - Y_{vn})}{(Y_{vm} - Y_{vn}) Z_n} \\ \dfrac{(SM_{vz} - Z_{Av}) \cdot (Z_m - Z_n) - Z_n \cdot (Z_{vm} - Z_{vn})}{(Z_{vm} - Z_{vn}) Z_n} \end{bmatrix}. \quad (4.3)$$

The system of equations (4.3) is similar for the points B and C, and its denominators do not depend on the location of the points in the world system and are not equal to zero, provided that there is indeed a display window in the screen coordinate system. In the world coordinate system, for the current point C_w belonging to the segment $A_w B_w$, we can write

$$\begin{bmatrix} X_{Cw} \\ Y_{Cw} \\ Z_{Cw} \end{bmatrix} = \begin{bmatrix} X_{Aw} \\ Y_{Aw} \\ Z_{Aw} \end{bmatrix} + t_w \cdot \begin{bmatrix} X_{Bw} - X_{Aw} \\ Y_{Bw} - Y_{Aw} \\ Z_{Bw} - Z_{Aw} \end{bmatrix}. \quad (4.4)$$

Similarly, for the screen coordinate system for the point C_v of the segment $A_v B_v$

$$\begin{bmatrix} X_{Cv} \\ Y_{Cv} \end{bmatrix} = \begin{bmatrix} X_{Av} \\ Y_{Bv} \end{bmatrix} + t_v \cdot \begin{bmatrix} X_{Bv} - X_{Av} \\ Y_{Bv} - Y_{Av} \end{bmatrix}. \quad (4.5)$$

Let's write the first equation of the system (4.4) taking into account the first equation of the system (4.3).

$$\begin{aligned} &(SM_{vx} - X_{Cv}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn}) \cdot Z_{Cw} = \\ &= (SM_{vx} - X_{Av}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn}) \cdot Z_{Aw} + \\ &+ ((SM_{vx} - X_{Bv}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn}) \cdot Z_{Bw} - \\ &- (SM_{vx} - X_{Av}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn}) \cdot Z_{Aw}) \cdot t_w. \end{aligned}$$

In the last equation, instead of X_{cv} and Z_{cw} in accordance with the first formula of system (4.5), substitute the value of $X_{Av} + (X_{Bv} - X_{Av}) \cdot t_v$. We obtain

$$\left[(SM_{vx} - X_{Av} - (X_{Bv} - X_{Av}) \cdot t_v) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn})\right] \times$$
$$\times (Z_{Aw} + (Z_{Bw} - Z_{Aw}) \cdot t_w) =$$
$$\left[(SM_{vx} - X_{Av}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn})\right] Z_{Aw} +$$
$$\left[\left[(SM_{vx} - X_{Bv}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn})\right] Z_{Bw} - \right.$$
$$\left. \left[(SM_{vx} - X_{Av}) \cdot (X_m - X_n) - X_n \cdot (X_{vm} - X_{vn})\right] Z_w \right] \cdot t_w$$

Opening the brackets and performing the equivalent transformations, we find that

$$t_w = \frac{Z_{Aw} \cdot t_v}{Z_{Bw} - t_v(\cdot Z_{Bw} - Z_{Aw})}. \quad (4.6)$$

Dividing the numerator and denominator of the fraction by Z_{Bw}, we obtain

$$t_w = \frac{t_v}{\frac{Z_{Bw}}{Z_{Aw}} - t_v(1 - \frac{Z_{Bw}}{Z_{Aw}})}. \quad (4.7)$$

If we know the vectors of the normals $\vec{N}_l$ and $\vec{N}_p$, respectively, at the left and right points of the triangle rasterization line, then the intermediate vector $\vec{N}_s$ can be found by the formula

$$\vec{N}_s = \vec{N}_l + t_w \cdot (\vec{N}_p - \vec{N}_l).$$

Failure to take into account the depth of the object when calculating the vectors leads to an error in determining its orthogonal components, which can be determined by formula (4.2) by replacing the color intensity value with the value of the orthogonal component.

For perspective-correct color reproduction in Phong shading, it is necessary to use nonlinear interpolation of normal vectors using the variable t_w. Unfortunately, the calculation of t_w by formula (4.6) involves performing a division operation for each current value of t_v. Let us consider the possibility of approximation t_w to simplify the hardware implementation. Since the dependence of t_w is nonlinear, the use of linear interpolation over the entire variable interval of t_v is excluded. Let's approximate t_w with a second-degree polynomial $a \cdot t_v^2 + b \cdot t_v + c$. Let's find the unknowns a, b, c. To do this, let's create a system of equations using three points $t_v = 0$, $t_v = 1$, $t_v = 1/2$.

$$\begin{cases} c = 0, \\ a + b + c = 1, \\ \frac{1}{4} \cdot a + \frac{1}{2} \cdot b + c = \frac{Z_{Aw}}{Z_{Bw} + Z_{Aw}}. \end{cases}$$

The system has the following solution

$$a = \frac{2 \cdot (Z_{Bw} - Z_{Aw})}{(Z_{Bw} + Z_{Aw})},\ b = \frac{(3 \cdot Z_{Aw} - Z_{Bw})}{(Z_{Bw} + Z_{Aw})},\ c = 0.$$

If $\hbar = \frac{Z_{Bw}}{Z_{Aw}}$, then $a = \frac{2 \cdot (\hbar - 1)}{(\hbar + 1)},\ b = \frac{(3 - \hbar)}{(\hbar + 1)}$.

The quadratic approximation gives satisfactory results only for $\hbar \le 3$. Fig. 4.7 shows a graph of the change in the absolute approximation error from t_v and $\hbar$.

A higher approximation accuracy can be achieved by using piecewise quadratic interpolation on two intervals of change t_v. For $0 \le t_v \le 0{,}5$

$$a = \frac{8 \cdot Z_{Aw} \cdot (Z_{Aw} - Z_{Aw})}{(Z_{Bw} + Z_{Aw})(3 \cdot Z_{Bw} + Z_{Aw})},\ b = \frac{(3 \cdot Z_{Aw} + Z_{Bw})}{(Z_{Bw} + Z_{Aw})(3 \cdot Z_{Bw} + Z_{Aw})},\ c = 0.$$

For $0{,}5 < t_v \le 1$

$$a = \frac{-8 \cdot Z_{Bw} \cdot (Z_{Aw} - Z_{Bw})}{(Z_{Bw} + Z_{Aw})(3 \cdot Z_{Bw} + Z_{Aw})},\ b = \frac{2 \cdot (9 \cdot Z_{Aw} - 5 \cdot Z_{Bw}) \cdot Z_{Bw}}{(Z_{Bw} + Z_{Aw})(3 \cdot Z_{Bw} + Z_{Aw})},$$

$$c = \frac{3 \cdot (Z_{Aw} - Z_{Bw})^2}{(Z_{Bw} + Z_{Aw})(3 \cdot Z_{Bw} + Z_{Aw})}.$$

The analysis showed that in this case, with $\hbar = 2, 3, 4, 5$ the maximum modulus of the relative error does not exceed 1 %, 4 %, 8 %, 13 %, respectively. As for three-dimensional objects, $\hbar$, usually does not exceed 3.

Let's consider for approximation using of a third-degree polynomial of the form $a \cdot t_v^{\,3} + b \cdot t_v^{\,2} + ct + d$. To find the unknowns, we will create a system of four equations. To do this, we equate the values of the polynomial and t_w (see Form 4.6) at points $t_v = 0,\ 1/3,\ 2/3,\ 1$. We find that

$$a=\frac{9\cdot\left(Z_{Bw}-Z_{Aw}\right)^2}{\left(2\cdot Z_{Bw}+Z_{Aw}\right)\cdot\left(Z_{Bw}+2\cdot Z_{Aw}\right)},\quad b=\frac{-9\cdot\left(Z_{Bw}-Z_{Aw}\right)\left(Z_{Bw}-2\cdot Z_{Aw}\right)}{\left(2\cdot Z_{Bw}+Z_{Aw}\right)\cdot\left(Z_{Bw}+2\cdot Z_{Aw}\right)},$$

$$c=\frac{\left(2\cdot Z^2{}_{Bw}-4\cdot Z_{Aw}\cdot Z_{Bw}+11\cdot Z_{Aw}\right)}{\left(2\cdot Z_{Bw}+Z_{Aw}\right)\cdot\left(Z_{Bw}+2\cdot Z_{Aw}\right)}.$$

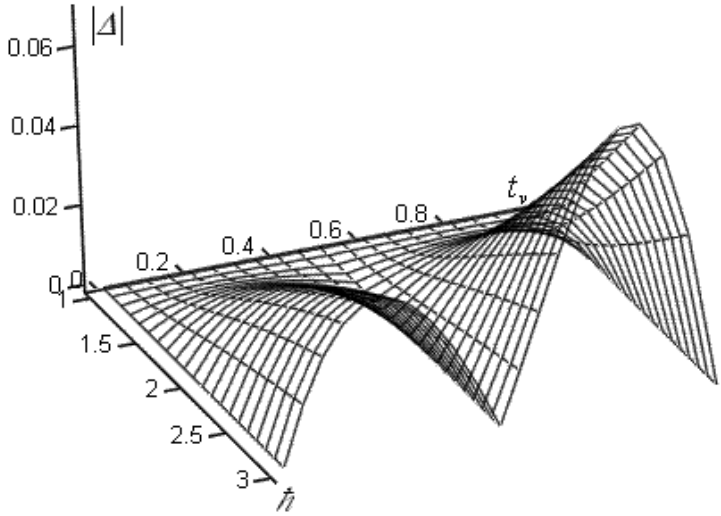

Fig. 4.7. Dependence of the absolute approximation error modulus on t_v and $\hbar$

The analysis showed that the use of cubic interpolation results in higher accuracy compared to piecewise quadratic interpolation. For example, at $\hbar = 2, 3, 4, 5$ the maximum modulus of the relative error does not exceed 0.64 %, 2.9 %, 6.3 %, and 10.6 %, respectively.

The angular interpolation (Fig.4.8) of unit vectors of normals between the initial $\vec{N}_a$ and final $\vec{N}_b$ vectors is performed according to the expression [2]

$$\vec{N}(w)=\vec{N}_a\frac{\sin((1-w)\psi)}{\sin\psi}+\vec{N}_b\frac{\sin(w\psi)}{\sin\psi},$$

where $w\in[0,1]$ is a parametric variable that determines the position of the streaming vector $\vec{N}(w)$ with respect to the vectors $\vec{N}_a$ and $\vec{N}_b$, and Ψ is the angle between the normal vectors $\vec{N}_a$ and $\vec{N}_b$.

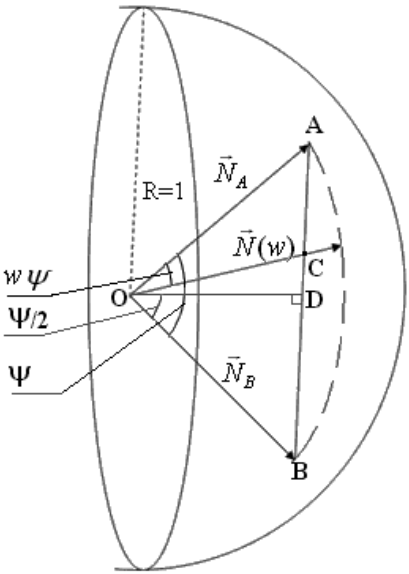

Fig. 4.8. Determination of normal vectors in spherical-angular interpolation

Since the vectors $\vec{N}_A$, $\vec{N}_B$ are unit (Fig. 4.8), the triangle OAB is isosceles. From the triangle OBD ($OD \perp DB$) we find that $DB = \left|\vec{N}_B\right| \cdot \sin\frac{\psi}{2}$. Given that $\left|\vec{N}_B\right| = 1$, then

$$AB = 2 \cdot DB = 2 \cdot \sin\frac{\psi}{2}.$$

Consider the triangle OAC.

$$\angle OAC = \frac{180 - \psi}{2}. \quad \angle OCA = 180^0 - \angle OAC - w \cdot \psi = \frac{180^0 + \psi}{2} - w \cdot \psi.$$

From the triangle AOC, by the sine theorem [3], we find that

$$AC = \left|\vec{N}_A\right| \frac{\sin(w \cdot \psi)}{\sin(\frac{180^0 + \psi}{2} - w \cdot \psi)} = \frac{\sin(w \cdot \psi)}{\sin(\frac{180^0 + \psi}{2} - w \cdot \psi)} = \frac{\sin(w \cdot \psi)}{\cos(w \cdot \psi - \frac{\psi}{2})}.$$

For the world coordinate system, the current value of the parametric variable d, which varies from 0 to 1, can be easily found through the ratio of the segment AC to AB.

$$d = \frac{\sin(w \cdot \psi)}{2\sin\frac{\psi}{2} \cdot \cos(w \cdot \psi - \frac{\psi}{2})} = \frac{\sin(w \cdot \psi)}{\sin(\frac{\psi}{2} + w \cdot \psi - \frac{\psi}{2}) + \sin(\frac{\psi}{2} - w \cdot \psi + \frac{\psi}{2})} =$$

$$= \frac{\sin(w \cdot \psi)}{\sin(w \cdot \psi) + \sin(\psi - w \cdot \psi)} = \frac{1}{1 + \sin\psi \frac{\cos(w \cdot \psi)}{\sin(w \cdot \psi)} - \cos\psi}.$$

It was proved [3] that the following relationship exists between a parametric variable v in the on-screen coordinate system and a variable d in the world coordinate system

$$d = \frac{1}{1 + (1 - \frac{1}{v}) \cdot \frac{z_2}{z_1}},$$

where z_2, z_1 — Z-coordinates, respectively, of the start and end points of a line segment in the world coordinate system. For the screen coordinate system, the following relation is true

$$v = \frac{1}{1 + \sin\psi \cdot \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi},$$

where η varies from zero to one.

Substituting the value v into the formula for d, we obtain

$$d = \frac{1}{1 + \frac{z_2}{z_1}\left(\sin\psi \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right)}.$$

Thus, to find the vectors of normals in the world coordinate system, using the value of the parametric variable η in the on-screen coordinate system, use the formula

$$\vec{N}(w) = \vec{N}_a \cos(\frac{1}{1 + \frac{z_2}{z_1}\left(\sin\psi \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right)}\psi) +$$

$$+ \vec{N}_k \sin(\frac{1}{1 + \frac{z_2}{z_1}\left(\sin\psi \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right)}\psi).$$

For the world coordinate system

$$d = \frac{1}{1 + \sin\psi \dfrac{\cos(w \cdot \psi)}{\sin(w \cdot \psi)} - \cos\psi}.$$

When perspective-correct shading

$$d = \frac{1}{1 + \dfrac{z_2}{z_1}\left(\sin\psi \dfrac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right)}.$$

Equating the right-hand sides of the above expressions, we find

$$\frac{1}{1 + \sin\psi \dfrac{\cos(w \cdot \psi)}{\sin(w \cdot \psi)} - \cos\psi} = \frac{1}{1 + \dfrac{z_2}{z_1}\left(\sin\psi \dfrac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right)}.$$

From the last equation we find

$$1 + \sin\psi \cdot \frac{\cos(w \cdot \psi)}{\sin(w \cdot \psi)} - \cos\psi = 1 + \frac{z_2}{z_1}\left(\sin\psi \cdot \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right).$$

After simplification, we obtain

$$\sin\psi \cdot \frac{\cos(w \cdot \psi)}{\sin(w \cdot \psi)} - \cos\psi = \frac{z_2}{z_1} \cdot \left(\sin\psi \cdot \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right).$$

We introduce the notation $a = \sin(w \cdot \psi)$. It is clear that $\sqrt{1 - a^2} = \cos(w \cdot \psi)$

$$\sin\psi \frac{\sqrt{1 - a^2}}{a} - \cos\psi = \frac{z_2}{z_1}\left(\sin\psi \frac{\cos(\eta \cdot \psi)}{\sin(\eta \cdot \psi)} - \cos\psi\right).$$

From the last equation we find

$$a = \frac{z1 \cdot \sin(\eta \cdot \psi) \cdot \sin(\psi)}{\sqrt{(z1 \cdot \sin(\eta \cdot \psi) \cdot \sin(\psi))^2 + (z1 \cdot \sin(\eta \cdot \psi) \cdot \cos(\psi) - z2 \cdot \sin(\psi \cdot (i-1))^2}}.$$

The above ratio involves the calculation of two functions at once, sine and cosine, which complicates the calculation and requires the storage of tabular data for two functions. Dividing the denominator and numerator of the fraction by $z1 \cdot \sin(\eta\psi) \cdot \sin(\psi)$ and performing the equivalent transformations, we find

$$\sin(w \cdot \psi) = \frac{1}{\sqrt{1 + \left[\operatorname{ctg}\psi + \dfrac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}(\psi))\right]^2}}.$$

The above expression is valid for all cases except when $z1 \cdot \sin(\eta \cdot \psi) \cdot \sin(\psi) = 0$. This is the case at the starting point of the triangle rasterization line, where the normal vector is given and does not need to be calculated, so the proposed division can be performed. When the vectors $\vec{N}_A$, $\vec{N}_B$ coincide, $\psi = 0$, there is no need to perform interpolation.

Similarly, we find

$$\cos(w \cdot \psi) = \frac{\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}(\psi))}{\sqrt{1 + \left[\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}(\psi))\right]^2}}.$$

Finally, we can write down that

$$\vec{N}(w) = \vec{N}_a \cdot \cos(w \cdot \psi) + \vec{N}_k \cdot \sin(w \cdot \psi) =$$

$$= \frac{\vec{N}_a \cdot (\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}\psi)) + \vec{N}_k}{\sqrt{1 + \left[\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}\psi)\right]^2}}$$

Let us introduce the notation $b = \operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}\psi)$, then

$$\vec{N}(w) = \frac{\vec{N}_a \cdot b + \vec{N}_k}{\sqrt{1 + (b)^2}}.$$

When applying spherical-angular interpolation, the time-consuming procedure of normalizing the normal vectors is excluded from the computational process. An important issue in this aspect is the need to normalize the vectors of normals obtained during the perspective-correct formation of images. Let us find the vector modulus according to the expression

$$\left|\vec{N}(w)\right|^2 = \left(\frac{\vec{N}_a(\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}\psi)) + \vec{N}_k}{\sqrt{1 + \left[\operatorname{ctg}\psi + \frac{z2}{z1}(\operatorname{ctg}(\eta \cdot \psi) - \operatorname{ctg}\psi)\right]^2}} \right)^2 =$$

$$= \left|\vec{N}_a\right|^2 \frac{\left(\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right)^2}{\left(\sqrt{1+\left[\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right]^2}\right)^2} +$$

$$+2\cdot\vec{N}_a\cdot\vec{N}_k \frac{\left(\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right)}{\left(\sqrt{1+\left[\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right]^2}\right)} +$$

$$+\left|\vec{N}_k\right|^2 \frac{1}{\left(\sqrt{1+\left[\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right]^2}\right)^2}.$$

The second term of the last expression is zero, since the vectors $\vec{N}_a, \vec{N}_k$ are orthogonal, and, as a result, their scalar product is zero. The vectors $\vec{N}_a, \vec{N}_k$ are unit, so $\left|\vec{N}_k\right|^2 = \left|\vec{N}_a\right|^2 = 1$. Taking into account the above

$$\left|\vec{N}(w)\right|^2 = \frac{1+\left(\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right)}{\left(1+\left[\operatorname{ctg}\psi + \frac{z2}{z1}\left(\operatorname{ctg}(\eta\cdot\psi) - \operatorname{ctg}\psi\right)\right]\right)} = 1.$$

Thus, it can be stated that normalization is not required for perspective-correct object rendering.

The results obtained allow us to increase the realism of the generated images due to the fact that the color intensities of the corresponding surface points in the screen and object coordinate systems will coincide, that is, the adequacy of the generated image to the real object increases.

4.4. ANALYSIS OF METHODS FOR APPROXIMATING PERSPECTIVE-CORRECT TEXTURING

One of the approaches to forming highly realistic images is to use textures [1, 5–9], which are mapped on graphic objects to give the image relief. The use of textures in many cases allows to successfully solve problems that are extremely time-consuming to solve by direct methods and significantly reduce computational costs. In texturing tasks, a correlation is established between screen and texture coordinates.

Perspective-correct texturing (PCT) uses nonlinear functions, the calculation of which involves time-consuming pixel-by-pixel operations. In the vast majority of cases, perspective-correct texturing is implemented using the following formulas [7, 9]

$$u = \frac{ax + by + c}{gx + hy + i}, \quad v = \frac{dx + ey + f}{gx + hy + i}, \tag{4.8}$$

where u and v — texture coordinates (TC), x and y — screen coordinates of an object, $a...i$ — coefficients of the polygon to be textured.

As can be seen from formula (4.8), finding the TC is a time-consuming procedure, since for each pixel of the image, six multiplication and two division operations are required, which significantly affects the speed of generating graphic scenes. Therefore, the task of simplifying the computational process of perspective-correct texturing is quite relevant.

In order to simplify the computational process, scientists have proposed various approaches to approximating perspective-correct texturing.

The simplest approach is linear interpolation, but it provides low accuracy of texture coordinates calculation and introduces significant artifacts and distortions in perspective rendering, which leads to the fact that the perspective of the object is not accurately reproduced.

The quadratic approximation uses the equation

$$u(x) = A_1 \cdot x^2 + A_2 \cdot x + A_3,$$
$$v(x) = B_1 \cdot x^2 + B_2 \cdot x + B_3,$$

where A_1-A_3, B_1-B_3 are approximation coefficients that are constant for each rasterization line, and x-coordinate values are normalized.

The known formulas for calculating the coefficients are as follows [8]

$$A_1 = 2u_0 - 4u_1 + 2u_2,\ A_2 = -3u_0 + 4u_1 - 2u_2,\ A_3 = u_0, \tag{4.9}$$

where u_0, u_1 i u_2, and are the values of the texture coordinate u at the start, middle, and end points of the rasterization line, respectively. The formulas for calculating the coefficients B_1 - B_3, however, the corresponding coordinate values are used instead of the v -coordinate values.

The use of quadratic approximation gives a fairly realistic reproduction of the perspective with relatively simple calculations. The disadvantage of formulas (4.9) is that they can be used only in the case of normalized values of screen coordinates. The normalization procedure requires a division operation, which significantly affects the computational complexity.

The cubic approximation [7] uses the relationship

$$u(x) = C_1 x^3 + C_2 \cdot x^2 + C_3 \cdot x + C_4,$$
$$v(x) = D_1 x^3 + D_2 \cdot x^2 + D_3 \cdot x + D_4,$$

where C_1-C_4, D_1-D_4 are approximation coefficients that are calculated for each rasterization line, the x-coordinate values are also normalized. To calculate the approximation coefficients, you need to find the exact values of the texture coordinates at four anchor points: the start, end, and two interior points that divide the rasterization line into three equal segments. The cubic approximation provides a more realistic perspective than the quadratic approximation, but the computational complexity increases significantly, which limits the scope of this type of approximation in real-time computer graphics systems.

The bi-quadratic [8] approximation uses the equation of the form

$$u(x) = A_1 x^2 + A_2 y^2 + A_3 xy + A_4 x + A_5 y + A_6,$$
$$v(x) = B_1 x^2 + B_2 y^2 + B_3 xy + B_4 x + B_5 y + B_6,$$

where A_1-A_6, B_1-B_6 are the approximation coefficients.

To calculate the 12 bi-quadratic approximation coefficients, you need to know the exact values of a pair of texture coordinates at six points: at the vertices of a triangular polygon and at the midpoints of its edges.

The bi-cubic approximation [8] uses the equation

$$u(x) = A_1 x^3 + A_2 y^3 + A_3 x^2 y + A_4 xy^2 + A_5 x^2 + A_6 y^2 + A_7 xy + A_8 x + A_9 y + A_{10},$$
$$v(x) = B_1 x^2 + B_2 y^2 + B_3 x^2 y + B_4 xy^2 + B_5 x^2 + B_6 y^2 + B_7 xy + B_8 x + B_9 y + B_{10},$$

where A_1-A_{10}, B_1-B_{10} are the approximation coefficients.

The bi-cubic approximation requires accurate values of a pair of texture coordinates at 10 control points and the calculation of 20 approximation

coefficients. The result looks quite realistic, but the computational cost is very high, so this type of approximation is of limited use.

Another approach to approximation is to split the rasterization line into several segments and use linear, quadratic, or cubic interpolation to calculate texture coordinates within the resulting segments. In this case, you need to calculate the exact values of the texture coordinates at the points between which the interpolation is performed.

4.5. RECURRENT DETERMINATION OF TEXTURE COORDINATES

In paper [9], it is proposed to use the midpoint method to approximate the hyperbolic curve, which allows you to find not approximate, but exact values of texture coordinates. In this case, the value of each texture coordinate is checked for compliance with the condition [9] for the values of the object's screen coordinates. If the value of the texture coordinate meets the condition, it is considered that its valid value has been found, otherwise, the next value of the texture coordinate is checked for compliance. At the same time, instead of division operations, addition and comparison operations are used, which greatly simplifies calculations. The disadvantage of the method is the assumption that the texture coordinates are set to integers, which is only a special case of texturing.

According to the OpenGL and DirextX specifications, texture coordinates are specified in the range [0;1], i.e., they are fractional numbers, which necessitates the development of a new PCT method that would take into account the noninteger representation of texture coordinates.

Suppose it is needed to find the texture coordinates u and v. Since the coordinates u and v are calculated similarly, then in the following we will consider only the coordinate u with generalization of the results and on the coordinate v.

When perspective-correct texturing using the midpoint method, texture coordinates are determined that meet the condition [9]

$$\frac{ax+by+c}{gx+hy+i}-0{,}5 \le u < \frac{ax+by+c}{gx+hy+i}+0{,}5\,. \qquad (4.10)$$

Condition (4.10) is acceptable for the case of integer texture coordinates. For texture coordinates from the range $[0;1]$, this condition will take the form

$$\frac{ax+by+c}{gx+hy+i}-\frac{\Delta u}{2}\le u<\frac{ax+by+c}{gx+hy+i}+\frac{\Delta u}{2}, \tag{4.11}$$

where Δu — the difference between neighboring values of a texture coordinate u, which is calculated by the formula $\Delta u = {u_{\max}}/{p}$, $u_{\max}$ — the maximum value of the texture coordinate u, p — number of texture points along the axis u.

Let $E(x,y)=2(gx+hy+i)$. Multiplying inequality (4.11) by $E(x,y)$, we obtain

$$\begin{gathered}2(ax+by+c)-\Delta u(gx+hy+i)\le uE(x,y)<\\<2(ax+by+c)+\Delta u(gx+hy+i)\end{gathered}.$$

Write the resulting inequality in the form of a system of inequalities

$$\begin{cases}uE(x,y)-2(ax+by+c)+\Delta u(gx+hy+i)\ge 0;\\ uE(x,y)-2(ax+by+c)-\Delta u(gx+hy+i)<0.\end{cases}$$

After simplification, we get

$$\begin{cases}uE(x,y)-x(2a-\Delta ug)-y(2b-\Delta uh)-(2c-\Delta ui)\ge 0;\\ uE(x,y)-x(2a+\Delta ug)-y(2b+\Delta uh)-(2c+\Delta ui)<0.\end{cases} \tag{4.12}$$

Let's introduce the following notation

$$W(x,y,u)=uE(x,y)-x(2a-\Delta ug)-y(2b-\Delta uh)-(2c-\Delta ui),$$

$$Q(x,y,u)=uE(x,y)-x(2a+\Delta ug)-y(2b+\Delta uh)-(2c+\Delta ui).$$

Then the condition (3.4) will take the form

$$\begin{cases}W(x,y,u)\ge 0;\\ Q(x,y,u)<0.\end{cases} \tag{4.13}$$

The system of inequalities (4.12) defines the conditions that the texture coordinate u must meet for the given values of the screen coordinates x and y. If the conditions (4.13) are not met for the current coordinate value u, then an increment Δu is added (subtracted) to it. The increment Δu is added (subtracted) until the conditions (4.13) are satisfied. When a texture coordinate value u is found that satisfies the conditions (4.13), the next point in the rasterization line is moved to.

Consider how the values $E(x,y)$, $W(x,y,u)$ and $Q(x,y,u)$ will change when changing coordinates x and y by 1.

When x is increased by 1, $E(x+1,y)=E(x,y)+2g$. Hence

$$W(x+1,y,u)=W(x,y,u)+2gu-(2a-\Delta ug),$$

$$Q(x+1,y,u)=Q(x,y,z)+2gu-(2a+\Delta ug).$$

If we reduce x by 1, then $E(x-1,y)=E(x,y)-2g$. Hence

$$W(x-1,y,u)=W(x,y,u)-2gu+(2a-\Delta ug),$$

$$Q(x-1,y,u)=Q(x,y,z)-2gu+(2a+\Delta ug).$$

With an increase of y by 1 $E(x,y+1)=E(x,y)+2h$. Hence

$$W(x,y+1,u)=W(x,y,u)+2hu-(2b-\Delta uh),$$

$$Q(x,y+1,u)=Q(x,y,u)+2hu-(2b+\Delta uh).$$

Expressions $(2a-\Delta ug)$, $(2a+\Delta ug)$, $(2b-\Delta uh)$ and $(2b+\Delta uh)$ is enough to calculate once for the entire polygon. Then finding the conditions (4.13) for the next point in the rasterization line involves performing only two multiplication operations and four addition operations.

If the current value of the texture coordinate does not satisfy the conditions (4.13), then you need to add or subtract an increment to it Δu. Let's define how the values $W(x,y,u)$ and $Q(x,y,u)$ change when changing a texture coordinate u by Δu.

Increasing u by Δu

$$\begin{cases} W(x,y,u+\Delta u)=W(x,y,u)+\Delta u\cdot E(x,y); \\ Q(x,y,u+\Delta u)=Q(x,y,u)+\Delta u\cdot E(x,y). \end{cases} \tag{4.14}$$

Reducing u by Δu

$$\begin{cases} W(x,y,u-\Delta u)=W(x,y,u)-\Delta u\cdot E(x,y); \\ Q(x,y,u-\Delta u)=Q(x,y,u)-\Delta u\cdot E(x,y). \end{cases} \tag{4.15}$$

The resulting formulas allow you to find the values of the conditions $W(x,y,u)$ and $Q(x,y,u)$ for the new value of the coordinate u, by performing only two multiplication and two addition operations.

From formulas (4.14) and (4.15) it is clear that the expressions $W(x,y,u)$ and $Q(x,y,u)$ increase with increasing u, and decrease with decreasing u,

because $\Delta u > 0$ and $E(x,y) > 0$. This allows to select the type of change of coordinate u. If $W(x,y,u) < 0$, then it is needed to add the increment Δu to the current value of the coordinate u. If $Q(x,y,u) > 0$, then the increment Δu is subtracted from the current value of the coordinate u.

The texture coordinate v is calculated in a similar way.

The principle of the proposed method is shown in Fig.4.9.

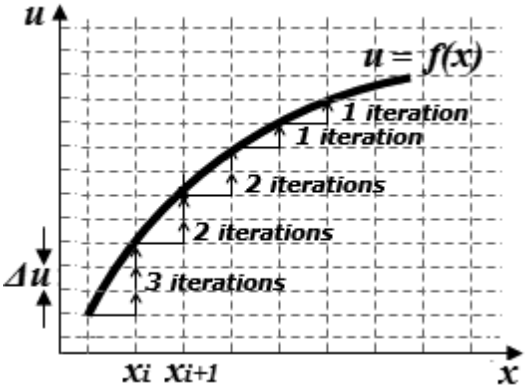

Fig. 4.9. Additive determination of a texture coordinate

From the example shown in Fig.4.9, it can be seen that to find the coordinate u at a point x_i, it is needed to perform 3 iterations of Δu incremental addition, and at a point x_{i+1} — 2 iterations, etc.

In the proposed method of perspective-correct texturing, division operations are replaced by addition operations, which significantly reduces the amount of computation and speeds up the texturing process.

The method allows finding the exact values of texture coordinates. Unlike the proposed method, the classical PCT method requires rounding or discarding the fractional part of the texture coordinate, which leads to texturing errors.

Compared to the method in [9], the proposed method is universal, since it can be applied to both integer and fractional representations of texture coordinates.

4.6. USING QUADRATIC APPROXIMATION FOR PERSPECTIVE-CORRECT TEXTURING

The peculiarity of the known methods of approximating perspective-correct texturing is that the internal reference points of the segments are placed evenly between the start and end points of the rasterization line (RL). In particular, with the traditional quadratic approximation, the internal reference point is the midpoint of the RL, which divides the line into two equal parts. In this case, there is a symmetrical distribution of the absolute approximation error relative to the midpoint of the PL. At the same time, the relative error takes on maximum values when the texture coordinates are close to zero. The relative error can be reduced if the internal reference point is not the middle point, but a point offset from the middle of the RL, depending on the nature of the change in the texture coordinate values. If the values of the texture coordinate increase along the RL, then the internal anchor point should be shifted towards smaller coordinate values, and if they decrease, then towards larger values.

Let's derive general formulas for calculating the approximation coefficients. To do this, solve the system of equations:

$$\begin{cases} Ax_0^2 + Bx_0 + C = u_0; \\ Ax_{вн}^2 + Bx_{вн} + C = u_{вн}; \\ Ax_1^2 + Bx_1 + C = u_1, \end{cases} \tag{4.16}$$

where A, B and C — quadratic approximation coefficients, x_0, $x_{вн}$, x_1 — screen x-coordinates at the first, inner, and end points of the PL, respectively, u_0, $u_{вн}$, u_1 — texture u-coordinates at the first, inner, and end points of the PL, respectively.

Since the traditional quadratic approximation uses normalized values of the screen coordinate, $x_0 = 0$ and $x_1 = 1$. Then the system (4.16) will take the following form

$$\begin{cases} C = u_0; \\ Ax_{вн}^2 + Bx_{вн} + C = u_{вн}; \\ A + B + C = u_1. \end{cases} \tag{4.17}$$

Let's find the coefficient A from the third equation of system (4.17) and substitute it into the second equation.

$$A = u_1 - C - B = u_1 - u_0 - B\ ,$$

$$\left(u_1 - u_0 - B\right)x_{вн}^2 + Bx_{вн} + u_0 = u_{вн}\ .$$

From the last equation, we find the coefficient B

$$B = \frac{x_{вн}^2\left(u_1 - u_0\right) + u_0 - u_{вн}}{x_{вн}^2 - x_{вн}}\ .$$

Hence

$$A = u_1 - u_0 - \frac{x_{вн}^2\left(u_1 - u_0\right) + u_0 - u_{вн}}{x_{вн}^2 - x_{вн}} = \frac{-\left(x_{вн}\left(u_1 - u_0\right) + u_0 - u_{вн}\right)}{x_{вн}^2 - x_{вн}}\ .$$

The formulas for calculating the coefficients A, B, C are as follows

$$\begin{cases} A = -\left(x_{вн}\left(u_1 - u_0\right) + u_0 - u_{вн}\right)\Big/\left(x_{вн}^2 - x_{вн}\right); \\ B = \left(x_{вн}^2\left(u_1 - u_0\right) + u_0 - u_{вн}\right)\Big/\left(x_{вн}^2 - x_{вн}\right); \\ C = u_0. \end{cases} \quad (4.18)$$

Calculating coefficients A and B requires 2 division operations, 5 multiplication operations, and 8 addition operations. Let's introduce the following notation

$$r = x_{вн}\left(u_1 - u_0\right),\ s = u_0 - u_{вн},\ q = \frac{1}{x_{вн}^2 - x_{вн}}\ . \quad (4.19)$$

Let us substitute the notation (4.19) into the system (4.18). Then we get

$$\begin{cases} A = -q\left(r + s\right); \\ B = q\left(r \cdot x_{вн} + s\right); \\ C = u_0. \end{cases} \quad (4.20)$$

The calculation of the approximation coefficients using formulas (4.20) requires 1 division operation, 5 multiplication operations, and 5 addition operations.

If the value of the internal reference point for the next rasterization line remains unchanged, it is obvious that the value of the coefficient q will not change either, which allows to eliminate 1 division, 1 multiplication, and 1 addition operation.

Table 4.1 shows the formulas for calculating the coefficients A and B for different values of the internal point of the rasterization line

Table 4.1

Formulas for calculating the coefficients of quadratic approximation for different values of the internal point

Value $x_{вн}$	Formula for calculating the coefficient A	Formula for calculating the coefficient B
0,75	$4u_1 + 1,333u_0 - 5,333u_{0,75}$	$-3u_1 - 2,333u_0 + 5,333u_{0,75}$
0,7	$3,333u_1 + 1,429u_0 - 4,762u_{0,7}$	$-2,333u_1 - 2,429u_0 + 4,762u_{0,7}$
0,6	$2,5u_1 + 1,667u_0 - 4,167u_{0,6}$	$-1,5u_1 - 2,667u_0 + 4,167u_{0,6}$
0,5	$2u_1 + 2u_0 - 4u_{0,5}$	$-u_1 - 3u_0 + 4u_{0,5}$
0,4	$1,667u_1 + 2,5u_0 - 4,167u_{0,4}$	$-0,667u_1 - 3,5u_0 + 4,167u_{0,4}$
0,3	$1,4297u_1 + 3,333u_0 - 4,762u_{0,3}$	$-0,429u_1 - 4,333u_0 + 4,762u_{0,3}$
0,25	$1,333u_1 + 4u_0 - 5,333u_{0,25}$	$-0,333u_1 - 5u_0 + 5,333u_{0,25}$

Similar formulas are used to calculate the coordinate v.

Fig. 4.10 shows graphs of the relative error of u texture coordinate approximation for a specific rasterization line (polygon: (1,1), (32,96), (96,128), (128,32), $y = 32$) at different locations of the internal anchor points, provided that the texture coordinate values vary from 0 to 1. The maximum relative error occurs when the internal anchor point splits the rasterization line in half, i.e. $x_{вн} = 0,5$. When the internal reference point is shifted to the left, the value of the maximum relative error decreases, and when the value $x_{вн} = 0,25$ is reduced by almost half.

Shifting the internal anchor point to the left allows to find more accurate texture coordinate values in the left part of the RL due to a proportional increase in the error in the right part. This way, the relative error values are centered along the rasterization line.

Shift the internal anchor point to lower values of the screen coordinate x is performed when the value of the texture coordinate along the rasterization line increases, i.e. $u_1 > u_0$, and towards higher values of the screen coordinate x — when $u_1 < u_0$. Table 4.2 shows the recommended values of the internal reference point for different values of texture coordinates at the start and end points of the rasterization line.

The proposed approach reduces the relative error in determining texture coordinates by up to 2 times. The advantage of the approach is that the increase in accuracy is achieved not by using higher-order curves or by introducing additional reference points, but only by shifting the internal ref-

erence point. The formulas for calculating the approximation coefficients for different values of the internal reference point are calculated once for the entire PL.

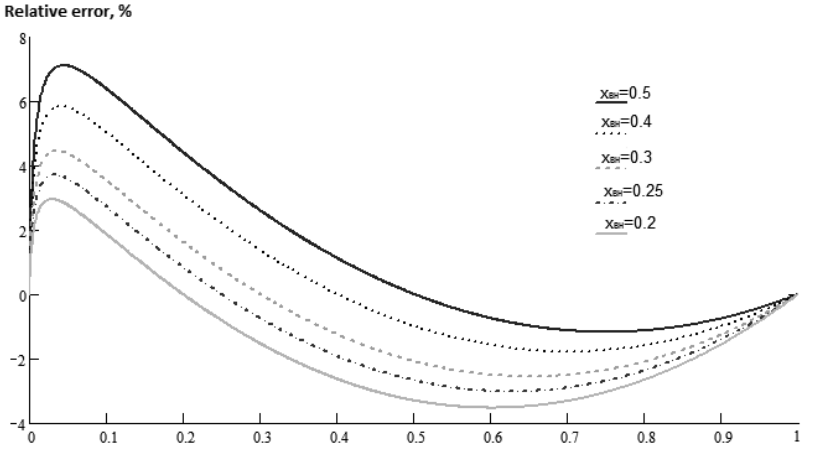

Fig. 4.10. Graphs of relative approximation errors at different locations of the internal reference point

Let's consider another possible approach to using approximation for texturing.

Traditional quadratic approximation [1, 7] uses normalized values of the object's screen coordinates for perspective-correct texturing. The normalization procedure involves performing a time-consuming division operation, which reduces the speed of texture overlay.

Let's find the formulas for calculating the quadratic approximation coefficients, provided that the object's screen coordinates are not normalized.

To determine the texture coordinate u according to quadratic approximation, it is necessary to find the unknown coefficients A_1—A_3. To find them, we write the system of three equations in matrix form:

$$\begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} \times \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix},$$

where (x_0, x_1, x_2) and (u_0, u_1, u_2) — values of coordinates x and u at the start, inner, and end points of the rectangular polygon rasterization line, respectively, A_1, A_2 and A_3 — quadratic approximation coefficients.

Table 4.2

Recommended values for the internal reference point

The value of texture coordinates at the start and end points	Range of the difference between the texture coordinate values at the ends of the rasterization line	Value $x_{вн}$
$u_0 \to 0$, $u_1 \in [0;1]$	–	*0,25*
$u_1 \to 0$, $u_0 \in [0;1]$	–	*0,75*
$u_0 \to 1$, $u_1 \in [0,05;1]$	$(u_0 - u_1) \in [0;0,35]$	*0,5*
	$(u_0 - u_1) \in [0,35;0,75]$	*0,6*
	$(u_0 - u_1) \in [0,75;0,95]$	*0,7*
$u_1 \to 1$, $u_0 \in [0,05;1]$	$(u_1 - u_0) \in [0;0,35]$	*0,5*
	$(u_1 - u_0) \in [0,35;0,75]$	*0,4*
	$(u_1 - u_0) \in [0,75;0,95]$	*0,3*
$u_0 < 0,5$, $u_1 < 0,5$: $u_0 < u_1$ $u_0 > u_1$	– –	*0,4* *0,6*
All other cases	–	*0,5*

Values u_0, u_1 and u_2 can be calculated using formula (4.9). The coefficients A_1, A_2 and A_3 will be found by Kramer's method [3]:

$$A_1 = \frac{\Delta_1}{\Delta}, \quad A_2 = \frac{\Delta_2}{\Delta}, \quad A_3 = \frac{\Delta_3}{\Delta},$$

where Δ, Δ_1, Δ_2 and Δ_3 — determinants of matrices, which are calculated as follows:

$$\Delta = \begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix} = x_0^2 (x_1 - x_2) + x_1^2 (x_2 - x_0) + x_2^2 (x_0 - x_1) ,$$

$$\Delta_1 = \begin{bmatrix} u_0 & x_0 & 1 \\ u_1 & x_1 & 1 \\ u_2 & x_2 & 1 \end{bmatrix} = u_0 (x_1 - x_2) + u_1 (x_2 - x_0) + u_2 (x_0 - x_1) ,$$

$$\Delta_2 = \begin{bmatrix} x_0^2 & u_0 & 1 \\ x_1^2 & u_1 & 1 \\ x_2^2 & u_2 & 1 \end{bmatrix} = x_0^2(u_1 - u_2) + x_1^2(u_2 - u_0) + x_2^2(u_0 - u_1),$$

$$\Delta_3 = \begin{bmatrix} x_0^2 & x_0 & u_0 \\ x_1^2 & x_1 & u_1 \\ x_2^2 & x_2 & u_2 \end{bmatrix} = x_0^2(x_1u_2 - x_2u_1) + x_1^2(x_2u_0 - x_0u_2) + x_2^2(x_0u_1 - x_1u_0).$$

Then the formulas for calculating the coefficients A_1, A_2, and A_3 will look like this

$$A_1 = d\left(u_0(x_1 - x_2) + u_1(x_2 - x_0) + u_2(x_0 - x_1)\right),$$

$$A_2 = d\left(x_0^2(u_1 - u_2) + x_1^2(u_2 - u_0) + x_2^2(u_0 - u_1)\right),$$

$$A_3 = d\left(x_0^2(x_1u_2 - x_2u_1) + x_1^2(x_2u_0 - x_0u_2) + x_2^2(x_0u_1 - x_1u_0)\right),$$

$$d = 1\Big/\left(x_0^2(x_1 - x_2) + x_1^2(x_2 - x_0) + x_2^2(x_0 - x_1)\right).$$

The coefficients $B_1 - B_3$ are calculated in a similar way, but the values v_0, v_1 and v_2 are used instead of the values u_0, u_1 and u_2. The approximation coefficients $A_1 - A_3$ and $B_1 - B_3$ are calculated once for each rasterization line.

Thus, instead of normalizing the screen coordinate value at each point of the rasterization line, only one division operation is performed per rasterization line to find the coefficient d.

4.7. USAGE OF SECOND-ORDER BЙZIER CURVES TO SIMPLIFY THE CALCULATION OF TEXTURE COORDINATES

In computer graphics, the method of constructing Bézier curves [1] is widely used, which allows to form curves of any shape. For perspective-correct texturing, a function is used that can be approximated by a quadratic Bézier curve, the parametric representation of which is as follows

$$r(t) = (1-t)^2 p_0 + 2t(1-t)p_1 + t^2 p_2, \qquad (4.21)$$

where $p_0(x_0, u_0)$, $p_1(x_1, u_1)$, $p_2(x_2, u_2)$ — the anchor points, and $t \in [0,1]$.

In this case, the Bezier curve will be inscribed in a triangle $p_0 p_1 p_2$ (Fig. 4.11).

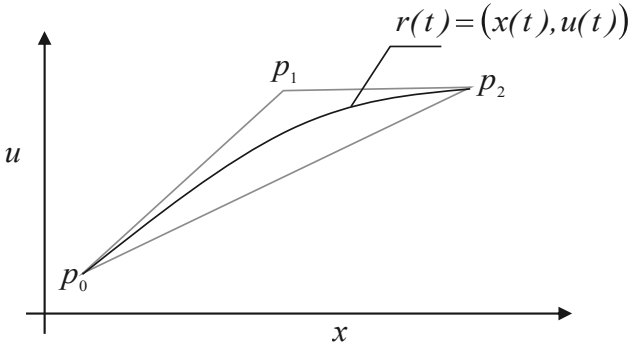

Fig. 4.11. Construction of a Bezier curve using three reference points

Since $r(t)=(x(t),u(t))$, then to build a Bézier curve, it is needed to solve two equations — relative to the screen x and texture u coordinates.

$$x(t)=(1-t)^2 x_0 + 2t(1-t)x_1 + t^2 x_2, \tag{4.22}$$

$$u(t)=(1-t)^2 u_0 + 2t(1-t)u_1 + t^2 u_1. \tag{4.23}$$

Equation (4.21) has a number of properties [1]:

1) $r(0)=p_0$, $r(1)=p_2$, that is, the first and last anchor points of the Bezier curve coincide with the corresponding points of the triangle in which it is inscribed. In order to find them, it is needed to find the exact values of the u_0 and u_2 texture coordinates for the first x_0 and last x_2 points of the rasterization line, respectively.

2) The tangent vectors at the ends of the Bezier curve coincide with the first and second sides of the triangle in which it is inscribed. That is, the point of intersection of these tangents will be the anchor point p_1.

Let's find the formulas for calculating the anchor point p_1.

The equation of the tangent for our case is

$$u = f(x_0) + f'(x_0)(x - x_0), \tag{4.24}$$

where $f'(x_0)$ — the derivative of a function $f(x)$ at a point x_0.

To find the intersection point p_1, equate the equations of the tangents at points x_0 and x_2 and find the expression for the coordinate x_1

$$f(x_0)+f'(x_0)(x_1-x_0)=f(x_2)+f'(x_2)(x_1-x_2).$$

$$x_1=\frac{f(x_0)-f(x_2)-f'(x_0)x_0+f'(x_2)x_2}{f'(x_2)-f'(x_0)}. \tag{4.25}$$

The value of the texture coordinate u_1 can be found by substituting the value x_1 into equation (4.24)

$$u_1=f(x_0)+f'(x_0)(x_1-x_0). \tag{4.26}$$

In the case of perspective-correct texturing

$$f(x)=\frac{ax+by+c}{gx+hy+i}, \text{ тоді } f(x_0)=u_0 \text{ і } f(x_2)=u_2. \tag{4.27}$$

The expression for finding the derivative $f'(x)$ is

$$f'(x)=\frac{a(hy+i)-g(by+c)}{(gx+hy+i)^2}=\frac{A}{(gx+B)^2}, \tag{4.28}$$

where coefficients $A=aB-g(by+c)$, $B=(hy+i)$ are constants for the rasterization line, since the screen coordinate y is a constant value.

Given expressions (4.27), (4.28), formulas (4.25) and (4.26) can be written as follows

$$x_1=\frac{u_0-u_2-\frac{A}{(gx_0+B)^2}x_0+\frac{A}{(gx_2+B)^2}x_2}{\frac{A}{(gx_2+B)^2}-\frac{A}{(gx_0+B)^2}}.$$

After simplification, we get the following

$$x_1=\frac{(u_0-u_2)(gx_0+B)^2(gx_2+B)^2-A\left(x_0(gx_2+B)^2-x_2(gx_0+B)^2\right)}{A\left((gx_0+B)^2-(gx_2+B)^2\right)}.$$

Let's introduce the notation $C_0=(gx_0+B)^2$ і $C_2=(gx_2+B)^2$. Hence

$$x_1=\frac{(u_0-u_2)C_0C_2-A(x_0C_2-x_2C_0)}{A(C_0-C_2)}, \tag{4.29}$$

$$u_1=u_0+\frac{A}{C_0}(x_1-x_0). \tag{4.30}$$

When changing the coordinate y by 1:

$$A(y+1) = a(h(y+1)+i) - g(b((y+1)+c) =$$
$$= a(hy+i) + ah - g(by+c) - gb = A(y) + D,$$

where $D = ah - gb$ — a constant for the entire polygon being textured.

$$B(y+1) = h(y+1) + i = hy + i + h = B(y) + h.$$

Thus, the calculation of the coefficients A and B for the next rasterization line requires only two addition operations.

Formula (4.22) contains 6 multiplication operations, 3 addition operations, and one shift operation. Let's derive recurrent formulas for calculating the screen coordinate x, which do not contain multiplication operations. To do this, we will use the method of finite differences [1, 3].

$$\Delta_x = x(t+dt) - x(t) = (1-t-dt)^2 x_0 + 2(t+dt)(1-t-dt)x_1 + (t+dt)^2 x_2 -$$
$$-(1-t)^2 x_0 + 2t(1-t)x_1 + t^2 x_2,$$

where dt — increment of parameter t along the rasterization line.

Opening the brackets, we get

$$\Delta_x = 2tdt(x_0 - 2x_1 + x_2) + 2dt(x_1 - x_0) + dt^2(x_0 - 2x_1 + x_2). \quad (4.31)$$

Let's introduce the notation

$$S_x = x_0 - 2x_1 + x_2 \text{ i } R_x = 2dt(x_1 - x_0) + dt^2 S_x = dt(2(x_1 - x_0) + dtS_x).$$

The coefficients S_x and R_x are calculated once for the RL. The recurrent formula for calculating the screen coordinate x along the RL will be

$$x(t+dt) = x(t) + 2tdtS_x + R_x. \quad (4.32)$$

Formula (4.32) requires two addition operations, two multiplication operations, and one shift operation. To remove the multiplication operations, we apply the finite difference method for Δ_x one more time so that the equality $\Delta_x(t+dt) = \Delta_x(t) + d_x$.

Then we get

$$d_x = \Delta_x(t+dt) - \Delta_x(t) = 2dt^2(x_0 - 2x_1 + x_2) = 2dt^2 S_x. \quad (4.33)$$

The expression $2dt^2 S_x$ does not depend on the value of the parameter t, so it is a constant for the entire rasterization string.

Taking into account formula (4.33), formula (4.30) can be written as follows

$$x(t+dt) = x(t) + \Delta_x(t+dt), \text{ де } \Delta_x(t+dt) = \Delta_x(t) + d_x. \qquad (4.34)$$

Similar formulas apply to the coordinate u.

The proposed recurrent formulas (4.34) allow to find the coordinate by performing only two addition operations.

In formulas (4.21) and (4.22), x and u depend on the parameter t, and the texturing task involves calculating the texture coordinate u for a specific integer value of the screen coordinate x. Therefore, it is necessary to express the parameter t in terms of the screen coordinate x.

Let's consider two approaches to solving this problem.

The first approach involves gradually increasing the value of the parameter t until it provides an integer value of the screen coordinate x, which is larger than the previous value of the found screen coordinate by 1, that is $x_{i+1} = x_i + 1$, where $i = \overline{0,\ m-1}$ is the ordinal number of the current point in the rasterization line, and m is the number of points in the rasterization line.

Using a sufficiently small step of parameter t does not always guarantee an integer value of the screen coordinate x. Therefore, it is advisable to make the assumption that

$$x_i(t_i) + 1 - \varepsilon \le x_{i+1}(t_{i+1}) \le x_i(t_i) + 1 + \varepsilon, \qquad (4.35)$$

where ε — permissible deviation from the integer value.

If $\Delta t_{i+1} = t_{i+1} - t_i$, then

$$t_{i+1} = t_i + \Delta t_{i+1}. \qquad (4.36)$$

Since for the point x_0 the value of the parameter $t_0 = 0$, then $t_1 = \Delta t_1$.

Let's write down the formula for the calculation Δt_1

$$\Delta t_1 = \sum_1^k dt - t_{kor},$$

where k — the number of iterations of adding a step, t_{kor} – the corrective value.

The increment operation of step dt is performed until the following condition is satisfied

$$x_1(\Delta t_1) \ge x_0 + 1 - \varepsilon.$$

In the case, when $x_1(\Delta t_1) > x_0 + 1 + \varepsilon$, it is necessary to reduce the value Δt_1 by a certain amount t_{kor}, that is, adjust the value Δt_1.

The value t_{kor} is calculated as follows. First $t_{kor} = \frac{dt}{2}$. If, after adjusting the value Δt_1 $x_1(\Delta t_1) > x_0 + 1 + \varepsilon$, then $t_{kor} = \frac{dt}{2} + \frac{dt}{4}$. The value t_{kor} should be increased until $x_1(\Delta t_1) \le x_0 + 1 + \varepsilon$. In this case, the value $\frac{dt}{2^n}$ is added, when n — sequence number of the adjustment iteration.

If during the adjustment the value $x_1(\Delta t_1)$ becomes less than the lower bound of condition (4.35), i.e. $x_1(\Delta t_1) < x_0 + 1 - \varepsilon$, then the sign for the current adjustment term changes to minus. The value t_{kor} calculation procedure is completed when condition (4.35) is met for the current value Δt_1.

Thus, the formula for the calculation t_{kor} is written as follows

$$t_{kor} = \pm\frac{dt}{2} \pm \frac{dt}{4} \pm \ldots \pm \frac{dt}{2^n}. \tag{4.37}$$

To calculate the incremental value Δt_2, we use the incremental value Δt_1 and find only the corresponding value t_{kor}.

The general formula for calculating the increase Δt_{i+1} can be written as follows

$$\Delta t_{i+1} = \Delta t_i - t_{kor}. \tag{4.38}$$

The calculation according to formulas (4.36)-(4.38) is simple from a hardware point of view, since only addition operations are necessary, and division operations can be realized by addition and mounting shift.

Computer simulations have shown that when applying the proposed approach to calculating the parameter t values, the maximum relative error does not exceed 0.7 %, which is 7 times less than when using the traditional quadratic approximation. Unlike traditional quadratic approximation, the proposed method does not require normalization of screen coordinates, which reduces computational complexity and, as a result, speeds up the texturing process.

The second approach to determining a parameter t through a screen coordinate x involves establishing a quadratic relationship

$$t = A_1 x^2 + A_2 x + A_3,$$

where A_1, A_2, A_3 — quadratic approximation coefficients.

The calculation of the approximation coefficients A_1, A_2, A_3 can be performed using formulas presented in the previous section. Given that $t_0 = 0$, and $t_2 = 1$, that formulas can be simplified as follows

$$A_1 = d\left(t_1\left(x_2 - x_0\right) + x_0 - x_1\right),\ A_2 = d\left(t_1\left(x_0^2 - x_2^2\right) - x_0^2 + x_1^2\right),$$
$$A_3 = d\left(x_0^2\left(x_1 - x_2 t_1\right) - x_0\left(x_1^2 + x_2^2 t_1\right)\right),$$
$$d = 1\Big/\left(x_0^2\left(x_1 - x_2\right) + x_1^2\left(x_2 - x_0\right) + x_2^2\left(x_0 - x_1\right)\right).$$

The results of computer modeling have shown that the maximum relative error when applying the quadratic dependence of the parameter t on the screen coordinate x does not exceed 1.7 %, which is almost 3 times less than the traditional quadratic approximation.

The proposed methods for calculating texture coordinates using the second-order Bezier curve, unlike the traditional quadratic approximation, do not require normalization of screen coordinates. This increases the accuracy of the approximation with a slight complication of the computational process.

4.8. NON-ORTHOGONAL RASTERIZATION METHODS

It is possible to reduce the number of division operations when texturing by rasterizing the object in the world coordinate system, provided that the rasterization lines are placed at a fixed distance from the observer. Let's find the slope coefficient of a line in the screen coordinate system, which corresponds to a line segment in the world coordinate system with a constant value of the coordinate z for the line (Fig.4.12). Let a triangle be given in the world coordinate system. It uniquely defines a plane whose equation is as follows

$$AX_w + BY_w + CZ_w = D\,, \tag{4.39}$$

where A, B, C are the coefficients determined by the coordinates of the vertices of the triangle.

From the last equation we find that

$$Z_w = \frac{A \cdot X_w + B \cdot Y_w + D}{C}\,. \tag{4.40}$$

The following relations exist between the screen and world coordinates $X_v = \dfrac{X_w}{Z_w},\ Y_v = \dfrac{Y_w}{Z_w}$. We write the last equations in the form

$$X_w = X_v \cdot Z_w,\quad Y_w = Y_v \cdot Z_w.$$

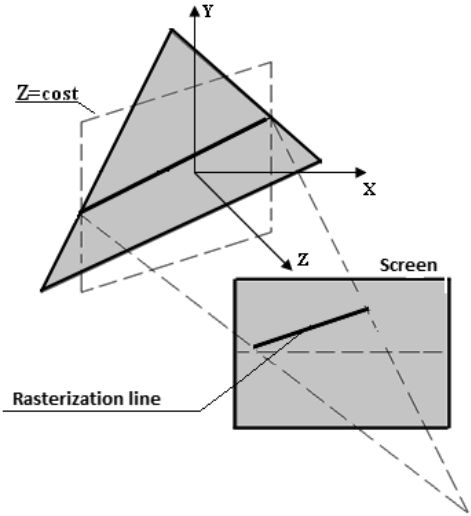

Fig. 4.12. Rasterization lines in the world and screen coordinate systems

Let's look for the angle of the line whose equation is $Y_v = k \cdot X_v + h$.
With this in mind, we write that $Y_w = Y_v \cdot Z_w = (k \cdot X_v + h) \cdot Z_w$.
Substituting into equation (4.40) and the value of X_w, Y_w, we obtain

$$A \cdot X_v \cdot Z_w + B \cdot (k \cdot X_v + b) \cdot Z_w + C \cdot Z_w = D.$$

From the last equation we find that

$$Z_w = \frac{D}{X_v \cdot (A + B \cdot k) + B \cdot h + C}. \qquad (4.41)$$

Provided that for a triangle rasterization line in the on-screen coordinate system $Z_w = const$, then for any j holds

$$\frac{D}{X_v \cdot (A + B \cdot k) + B \cdot h + C} = \frac{D}{(X_v + j) \cdot (A + B \cdot k) + B \cdot h + C}.$$

The last equation has a unique solution $k = -A / B$. Since j and X_v were chosen arbitrarily, it can be stated that the slope of the scanning rasterization line does not change for the entire triangle considered in the original coordinate system. The value of the coordinate Z_w for a given rasterization line is easy to find by substituting the obtained value k into equation (4.41).

$$Z_w = D / (B \cdot h + C).$$

The following relationship exists between the coordinates of the texture and screen spaces [1, 3]:

$$u = \frac{a \cdot X_v + b \cdot Y_v + c}{A \cdot X_v + B \cdot Y_v + C}, \quad v = \frac{d \cdot X_v + e \cdot Y_v + f}{A \cdot X_v + B \cdot Y_v + C}.$$

Denote the denominator of the above expressions by T, and $1/T = \Re$, then

$$u = (a \cdot X_v + b \cdot Y_v + c) \cdot \Re, \quad v = (d \cdot X_v + e \cdot Y_v + f) \cdot \Re.$$

The denominator for determining the texture coordinates u, v is constant for the rasterization row, while in the conventional approach it is calculated for each point of the row. In the following, we will consider only one of the coordinates, for example, u since the expressions for their calculation are similar. Let's express Y_v through k and substitute it into the previous expression. We get

$$u = (a \cdot X_v + b \cdot (k \cdot X_v + h) + c) \cdot \Re = [X_v \cdot (a + b \cdot k) + \ (b \cdot h + c)] \cdot \Re.$$

For the starting point of the rasterization line $X_v = 0$. With this in mind $u_0 = \Re \cdot (b \cdot B + c)$. Consider how u changes when coordinate X_v changes by one

$$u_{i+1} = [(X_v + 1) \cdot (a + b \cdot k) + \ (b \cdot h + c)] \cdot \Re = u_i + \Re \cdot (a + b \cdot k).$$

The resulting ratio can be easily calculated in hardware, provided that $\Re$ is known.

The non-orthogonal direction of rasterization of the area bounded by the polygon allows to reduce the computational complexity of the process of applying textures to the surface of a three-dimensional graphic object. For a triangle containing T internal points, *(T-q)* division operations are performed, where q is the number of horizontal and vertical rasterization lines of the triangle.

The issue of triangle rasterization is important, since the direction of rasterization, which is not orthogonal to the coordinate axes, will inevitably lead to artifacts — the presence of "gaps" and duplicate points due to the offset of the starting points of the rasterization lines. This can be avoided if the leading edge of the triangle is parallel to the ordinate axis, but this requires a special surface triangulation and does not meet the requirements of graphic standards. Artifacts can be eliminated by adaptive phasing of the sequence of step increments of the scan line, which involves setting different initial values of the evaluation function during interpolation, which significantly complicates the linear interpolator.

The easiest way to solve the problem is to rasterize not the triangle, but the rectangle into which it is virtually inscribed (Fig. 4.13, a). You can determine the parameters of such a rectangle by comparing the co-ordinates of the vertices of the triangle (the left and rightmost vertices of the triangle represent the abscissa of the left and right sides of the rectangle, and the bottom one represents the ordinates of the bottom side of the rectangle). The parameter r can be easily found by substituting the value of the abscissa of the top vertex of the triangle into the equation of the line used for rasterization. It is clear that the ordinate of the upper left vertex of the rectangle is equal to the sum of r and the ordinate of the upper vertex of the triangle.

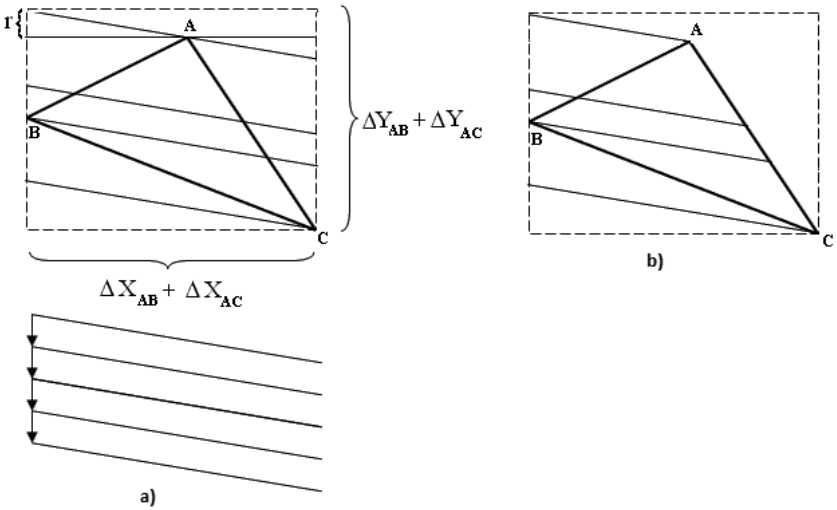

Fig. 4.13. Rasterizing an area bounded by a triangle

When rasterizing a rectangle, provided that the rasterization direction is not orthogonal, there will be no "cutting through" or "sticking" artifacts, since the step increments of neighboring lines will be identical along the ordinate directions.

Rasterization of a rectangle is performed to determine the coordinates of the left and right points of the triangle edges that intersect the rasterization line without calculations that require "long" operations. When the right edge of the triangle is reached, the transition to a new rasterization line of the rectangle is made, which is located one ordinal level lower (Fig.4.13,

a), that is, the area of the triangle behind its right edge is not rasterized (Fig. 4.13, b). Determining the coordinates of the left and right edges of a triangle is achieved by comparing the foreground color of the triangle with a reference. In this case, one can, for example, use the principle of the parity criterion [4], according to which the number of intersections of a polygon is an even number. To improve performance, it is possible to rasterize the areas that have the background color with a pulse sequence of increased frequency. Direct texturing is performed at the clock frequency at which the video memory operates.

With anisotropic filtering [5–7], which significantly increases the realism of reproducing graphic scenes, the color of a pixel is determined by several textures. The shape of the light spot changes with the position of the polygon relative to the observer's point. The higher the level of anisotropic filtering, the more the performance of the graphic scene generation decreases. The high computational complexity of texturing using anisotropic filtering limits its widespread use in image formation, although modern graphics cards include such functions as basic ones.

With anisotropic filtering, the projection of a pixel onto the model surface is not seen as a circle, but as an elongated ellipse (Fig. 4.14). In order to correctly calculate the color of a pixel, it is necessary to take into account the colors of all texture samples that fall into the ellipse. This is a rather complicated procedure for generating images in real time, so a simplification is used — replacing the ellipse with a parallelogram (Fig. 4.15) or a rectangle that bounds it.

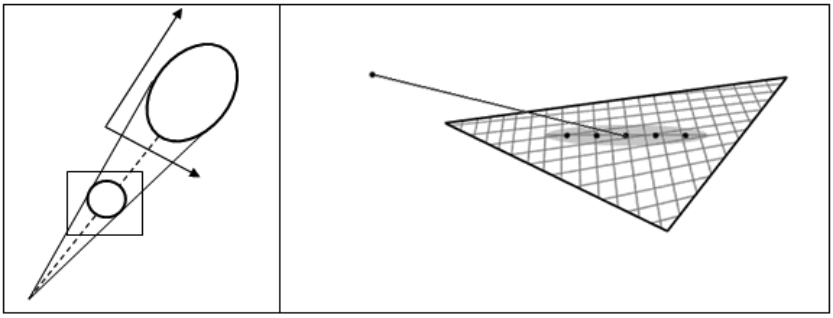

Fig. 4.14. Determination of the trace in anisotropic filtration

To calculate the principal axes of the ellipse, the functional determinant, the Jacobian, is calculated [3]:

$$j = \begin{vmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{vmatrix} = \Re^2 \begin{vmatrix} aBY + aC - AbY - Ac & bAX + bC - BaX - Bc \\ dBY + dC - AeY - Af & eAX + eC - BdX - Bf \end{vmatrix} =$$

$$= \Re^2 \begin{vmatrix} Y(aB - Ab) + aC - Ac & X(bA - Ba) + bC - Bc \\ Y(dB - Ae) + dC - Af & X(eA - Bd) + eC - Bf \end{vmatrix}.$$

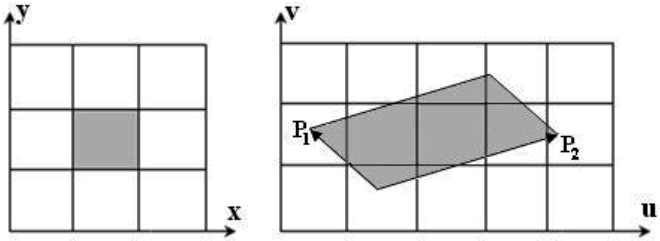

Fig. 4.15. Pixel trace in the form of a parallelogram with anisotropic filtering

The parameter $\Re$ is constant for the rasterization line. Unknown parameters P_1, P_2 can be determined by the formula

$$P_2 = u((x+1),\ y) - u(x,y),\ v((x+1),y) - v(x,y),$$
$$P_1 = u(x,\ (y+1)) - u(x,y),\ u(x(y+1)) - v(x,y).$$

P_1, P_2 can be calculated through the Jacobian, which defines the transformation from one coordinate system to another (in this case, from the coordinate system OXY to the coordinate system OVU).

P_1, P_2 can be found through the derivatives in the directions *(1, 0)* and *(0, 1)*.

$$P_1 = J\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix} Y(aB - Ab) + aC - Ac \\ Y(dB - Ae) + dC - Af \end{bmatrix},\ P_2 = J\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} X(bA - Ba) + bC - Bc \\ X(eA - Bd) + eC - Bf \end{bmatrix}.$$

In the case when $z = const$, the sides P_{1z}, P_{2z} of the parallelogram are found by derivatives in the directions *(0, 1), (1, k)*, respectively.

$$P_{1z} = J(0,\ 1) = P_1 \quad P_{2z} = J(1,\ k) = \Re^2 \begin{bmatrix} (aB - Ab) \\ (dB - Ae) \end{bmatrix}$$

The last expression shows that P_{2z} is constant for the rasterization line, since

$$\Re^2 = \frac{1}{(AX_v + BY_v + C)^2} = \frac{1}{(AX_v + B(-\frac{A}{B}X_v + h) + C)^2} = \frac{1}{(Bh + C)^2}.$$

In most cases, anisotropic filtering uses a rectangular image window to rasterize a texture surface. In this case, the issue of fast rasterization of such a window, which is generally rotated relative to the coordinate axes, is important.

The authors have developed a method for forming a window at an arbitrary angle to the coordinate axes.

The vectors AB and AC (Fig. 4.16) have the same inclination angles with respect to the abscissa and ordinate axes. Thus, their component step increments relative to the given coordinate axes coincide and differ only in signs. Thus, when forming the output vector *AB* by memorizing its step increments, it is possible to implement the *AC* path. To do this, when forming the *i*-th line in the direction of *AB*, the type of the *i*-th step increment is memorized and formed after the completion of the development of the given vector in another orthogonal direction.

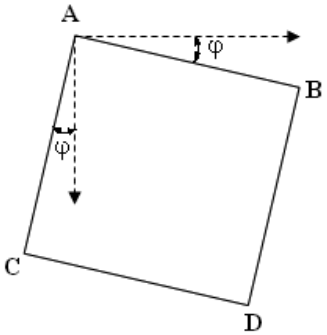

Fig. 4.16. Location of the window relative to the axes coordinates

To rasterize a rectangular image window, a loop traversal of the rectangular image window points is performed. During the loop traversal, the image points located in the first line *AB* of the image window are read (Fig. 4.17), after which a function of transition from point *B* to point *C* is formed for one discrete. After performing these actions, the direction of the traversal is reversed, that is, the traversal is performed along the *SD* vector, the signs of which are opposite to the *AB* vector. A similar traversal procedure takes place for all subsequent lines of the window.

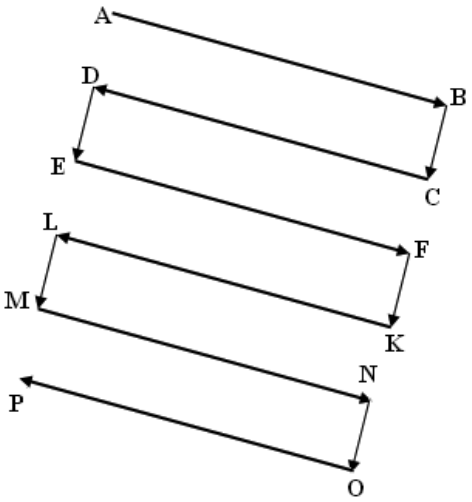

Fig. 4.17. Daisy-chain bypass of window rows

Suppose you need to rasterize a rectangular window *ABQP* (Fig. 4.18), the internal points of which define a certain texture image. To do this, perform the following steps.

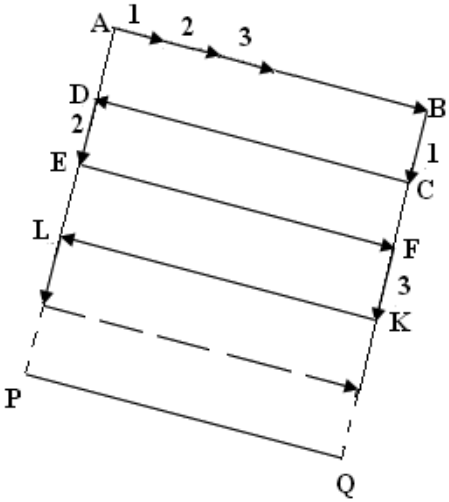

Fig. 4.18. Determining the coordinal displacements for the sides of the window

Determine the increments ΔX, ΔY of the base vector *AB*. Using the linear interpolation function, determine the coordinates of the image points in the direction of the base vector *AB*. When you reach point *B*, which corresponds to the end point of the vector *AB*, form a signal of transition from point *B* to point *C*. To do this, when forming a step trajectory of the base vector *AB*, the value of the step increment corresponding to point 1 of the *AB* vector is memorized, and when point B is reached, a step increment is formed with the opposite sign along the abscissa axis and the orthogonal components are swapped. This will ensure the transition to point *C*, which is located one discrete step away from point *B*. Then change the direction of the base vector to the opposite and repeat the above steps. Thus, when forming the *CD* vector, the step increment is memorized, which is formed in the second interpolation step. The recorded step increment is used at point *D* to move to point *E*. Similarly, when forming the third vector, the value of the step increment in the third cycle is memorized, which is used at point *F* to move to point *K*.

In general, when forming the *i*-th line, the type of the *i*-th step increment is memorized and formed relative to the ordinate axis after the specified vector is finished. The described actions are repeated until the image window is rotated.

During anisotropic filtering, the textures obtained during the rasterization of the image window are averaged according to the selected filtering method.

The proposed method of rasterizing a rectangular window formed at a certain angle to the coordinate axes has a simple hardware implementation and involves the use of not two, but only one linear interpolator, which allows reducing hardware costs by up to two times.

4.9. DETERMINING THE DIRECTION OF RASTERIZATION TO SPEED UP PERSPECTIVE-CORRECT TEXTURING

In texture mapping tasks, polygon points are traditionally processed along horizontal rasterization lines. Let's prove that for polygons whose two opposite sides are parallel to one of the screen coordinate axes, one of the texture coordinates remains constant along the rasterization line. Consider two cases of this dependence.

Statement 4.1. For polygons with two opposite sides parallel to the X-axis, the texture coordinate v remains constant for the horizontal rasterization line when perspective-correct texturing is performed.

Proof. Let's assume a polygon with two opposite sides parallel to the X-axis and a sample texture to be applied to this polygon (Fig. 4.19). The coordinates of the vertices of the polygon are expressed in terms of x and y; the coordinates of the vertices of the texture are expressed in terms u of and v.

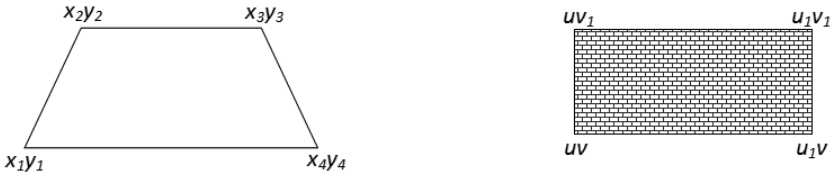

Fig. 4.19. A polygon with opposite sides parallel to the x-axis and a sample texture

According to formulas (3.1), the texture coordinate v will remain constant for the horizontal rasterization line in the case when the coefficients d and g take zero values.

The coefficients d and g are calculated using the formulas [7]:

$$d = E \cdot I - F \cdot H\ ,\ g = D \cdot H - E \cdot G\ , \tag{4.42}$$

$$H = \begin{vmatrix} \Delta x_1 & \sum x \\ \Delta y_1 & \sum y \end{vmatrix} \Bigg/ \begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix},\ E = y_4 - y_1 + Hy_3\ , \tag{4.43}$$

$$\sum y = y_1 - y_2 + y_3 - y_4\ ,\ \Delta y_1 = y_2 - y_3\ ,\ \Delta y_2 = y_4 - y_3\ . \tag{4.44}$$

From Fig. 4.19 it is obvious that $y_1 = y_4$ and $y_2 = y_3$. Then, according to the formulas (4.44)

$$\sum y = 0,\ \Delta y_1 = 0\ .$$

The obtained values allow to find the solution to the formulas (4.43):

$$\begin{cases} \sum y = 0, \\ \Delta y_1 = 0. \end{cases} \Rightarrow H = 0 \text{ and } \begin{cases} H = 0, \\ y_4 - y_1 = 0. \end{cases} \Rightarrow E = 0\ . \tag{4.45}$$

Substituting the values of (4.45) into formulas (4.42), we obtain

$$d = 0,\ g = 0. \tag{4.46}$$

Considering equation (4.46), the coordinates v and v_1 are calculated as follows

$$v = \frac{ey + f}{hy + i}\ ,\ v_1 = \frac{ey_1 + f}{hy_1 + i}\ . \tag{4.47}$$

The system of equations (4.47) shows that for a polygon with two opposite sides parallel to the X-axis, the texture coordinate v does not depend on the value of the polygon's screen coordinate x, but only on the value of the screen coordinate y. That is, the texture coordinate v for each horizontal rasterization line is calculated once and used for the entire RL.

Then the formula for determining the coordinate u will be as follows

$$u = \frac{ax + by + c}{hy + i} = (ax + A_u) B_u,$$

where $A_u = by + c$, $B_u = \frac{1}{hy + i}$ — are the constants for the horizontal RL.

Statement 4.2. For polygons whose two opposite sides are parallel to the Y-axis, the texture coordinate u remains constant for a vertical rasterization line when perspective-correct texturing is performed.

Proof. Let's take a polygon with two opposite sides parallel to the Y-axis and a sample texture that will be applied to this polygon (Fig. 4.20).

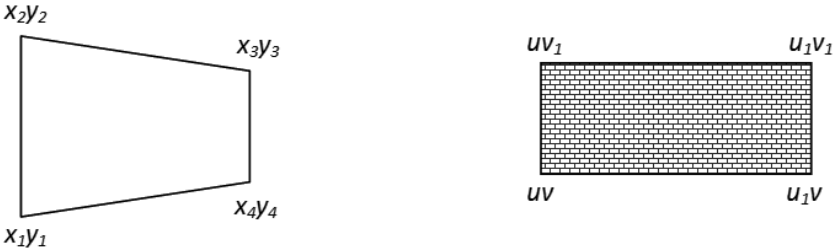

Fig. 4.20. A polygon with opposite sides parallel to the Y axis and a sample texture

Similarly to statement 4.1, we can prove that the coefficients b and h take on zero values, and the texture coordinate u remains constant for each vertical rasterization line.

The formula for determining the coordinate v is as follows

$$v = \frac{dx + ey + f}{gx + i} = (ey + A_v) B_v,$$

where $A_v = dx + f$, $B_v = \frac{1}{gx + i}$ — are constants for the vertical RL.

Polygons that have two opposite sides parallel to the X or Y axis are often found in software applications for visualizing room interiors or building exteriors, as well as in computer games where the main action takes place indoors. In particular, such elements of decor and furnishings as carpets,

paintings, mirrors, doors, windows, and furniture are also described by such polygons.

A special case of perspective-correct texturing is affine texturing [1, 4], which occurs when the object to be textured is parallel to the projection plane, i.e., the screen. For affine texturing $g = h = 0$, $i = 1$. Then the texture coordinates u and v are defined as follows

$$u = ax + by + c \text{ and } v = dx + ey + f. \quad (4.48)$$

Formulas (4.48) do not contain division operations, which simplifies the texturing process. The constancy property of texture coordinates can also be applied to affine texturing, which will eliminate redundant operations when finding texture coordinates for cases where two opposite sides of a polygon are parallel to one of the coordinate axes.

If an object subject to affine texturing has two opposite sides parallel to the X-axis, the coefficient $d = 0$ and texture coordinate v do not depend on the value of the screen coordinate x, but are defined as $v = ey + f$, that is, for each horizontal rasterization line, the value v is calculated once and used along the entire line.

Similarly, the texture coordinate u is calculated using the formula $u = ax + c$ if the object has two opposite sides parallel to the Y axis. In other words, the value u is calculated once for each vertical RL.

The application of the proven properties to perspective-correct and affine texturing can significantly reduce the computational complexity of calculating texture coordinates without compromising the quality of texturing.

You can choose a particular direction of rasterization by checking the values of the coefficients in the Heckbert formula. If none of the proven properties can be applied, it is suggested to perform texturing along some non-orthogonal rasterization line (NRL). The denominator $gx + hy + i$ in formulas (4.8) is determined by the z-coordinate. Provided that in the object coordinate system we draw a plane parallel to the z-axis through the selected surface point, then, in general, we will get a RL, which in the screen system will correspond to a line with a non-orthogonal direction relative to the screen coordinate system. It is clear that in this case, the denominator in formulas (4.8) will have a constant value for the entire RL. Therefore, the task is to find a RL for which the expression $gx + hy + i$ will have a constant value.

Suppose that the coordinate x is increased by 1, then the coordinate y for the corresponding NRL will change by a certain value Δy. Let's find the value of the increment Δy, given that for each point of the NRL the value

of the expression $gx+hy+i$ is a constant. Then, for two adjacent points of the NRL, we can write

$$gx+hy+i=g\left(x+1\right)+h\left(y+\Delta y\right)+i=gx+hy+i+g+h\Delta y\ .$$

From the last equation we find Δy

$$\Delta y=-g/h\ . \tag{4.49}$$

As can be seen from formula (4.49), the value of the increment Δy does not depend on the values of the coordinates x and y, but only on the coefficients g and h of the polygon being textured. This indicates that the increment Δy is constant for the entire polygon and it is enough to calculate it once.

Let's write down the general formula for calculating the y-coordinate for the NRL

$$y_i=y_p+\Delta y\cdot\left(x_i-x_p\right), \tag{4.50}$$

where $\left(x_p,y_p\right)$ — coordinates of the starting point of the NRL that belongs to the polygon.

In the recurrent form, the coordinate y for the NRL can be found by the formula

$$y_{i+1}=y_i+\Delta y\ . \tag{4.51}$$

Formula (4.51) requires only one addition operation, which leads to its simple hardware implementation.

Taking into account the discrete nature of the RL formation (Fig. 4.21), the values of the coordinate y must be integer, so we rewrite formula (4.50) as follows

$$y_i=y_p+\left\lceil\Delta y\cdot\left(x_i-x_p\right)\right\rceil\ , \tag{4.52}$$

If you start counting NRL from the first point belonging to the polygon, you may encounter cutoff points between neighboring RLs, which will lead to artifacts. According to the property that cutoff points will not occur if the neighboring RLs are shifted relative to each other by one horizontal or one vertical step. This can be easily achieved by assuming that the initial points of all NRLs belong to the Y-axis. Let denote them by $\left(x_0,y_0\right)$, and for all NRLs $x_0=0$, and the values y_0 differ by 1. Then the formula (4.52) will take the following form

$$y_i=y_o+\left\lceil\Delta y\cdot x_i\right\rceil\ .$$

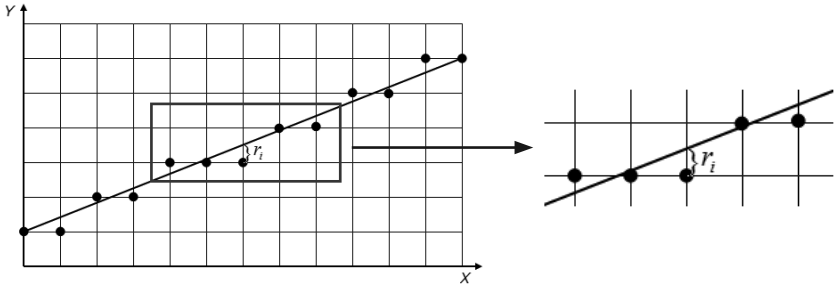

Fig. 4.21. Discretization of the NRL

The value y_0 can be found through the coordinate values of the point (x_p, y_p)

$$y_o = y_p - \lceil \Delta y \cdot x_p \rceil \ .$$

Consider the calculation of texture coordinates along the NRL.

Introduce the notation $k = \dfrac{1}{gx_p + hy_p + i}$.

Then the texture coordinates u and v along the NRL will be calculated as follows

$$u_i(x,y) = (ax_i + by_i + c)k, \ \ v_i(x,y) = (dx_i + ey_i + f)k.$$

Taking into account the discrete nature of the formation of rasterization lines, the coefficient k must be adjusted.

To adjust the value k, we find the first derivative $k'(y)$ of the coordinate y, then the formula for adjustment k will be as follows

$$kor_i = k - r_i \cdot k'(y), \tag{4.53}$$

where $r_i = \Delta y \cdot x_i - \lceil \Delta y \cdot x_i \rceil$ — is the vertical distance (Fig. 4.21) between the discretized and undiscretized NRL for the current point, and $k'(y)$ — is a derivative that characterizes the change in the value k depending on the change in the value of y.

$$k'(y) = \frac{dk}{dy} = \frac{-h}{(gx + hy + i)^2} = -k^2 h \ . \tag{4.54}$$

Taking into account expression (4.54), formula (4.53) can be rewritten as follows

$$kor_i = k + r_i \cdot k^2 h \, .$$

The expression $k^2 h$ is a constant value for the NRL, so it is enough to calculate it once.

Thus, the proposed method of perspective-correct texturing along the NRL allows to remove division operations from the procedure for calculating texture coordinates. The number of division operations depends on the number of NRLs, since for each rasterization line it is necessary to find the coefficient k. To find the increment Δy, it is needed to perform only one division operation by polygon. The total number of polygon division operations is $(q+1)$, where q is the number of NRLs. Calculating a pair of texture coordinates directly requires 8 multiplication operations and 7 addition operations.

4.10. CONCLUSIONS

The theoretical foundations of correct color reproduction in image formation have been developed, which has increased the realism of graphic scenes.

1. For the first time, methods of correct color reproduction when shading three-dimensional graphic objects based on linear interpolation of color component intensities, linear and spherical-angular interpolation of vectors and taking into account perspective projection are proposed, which allows to increase the realism of the formation of graphic scenes by establishing the color matching of surface points in the object and screen coordinate systems.

2. The method of Barenbrug perspective-correct texturing has been improved, in which, unlike the existing one, new formulas for calculating texture coordinates have been proposed. This made it possible to expand the scope of the method by both integer and fractional representation of texture coordinates and, as a result, to meet the requirements of the OpenGL and DirectX graphics standards.

3. It is proposed to use quadratic approximation for perspective-correct texturing without normalizing the values of screen coordinates and non-binary division of the rasterization line into segments. This made it possible to reduce the time for calculating the texture coordinates and reduce the relative error of their determination by up to two times.

4. A method for improving the performance of perspective-correct texturing is proposed, in which new recurrence relations are used for the first

time to determine the texture coordinates of a streaming point in a rasterization line, which made it possible to exclude division operations from the texturing cycle and increase performance by simplifying the computational process. By using the second-order Bezier curve for approximation instead of quadratic interpolation, the accuracy of texture coordinates determination was improved.

5. An adaptive approach to perspective-correct texturing based on the analysis of coefficients in the Heckbert formula is proposed, which allows to choose the optimal orientation of the rasterization line, which will increase the performance of calculating texture coordinates by removing redundant division operations without compromising the accuracy of determining texture coordinates.

6. A method for improving the performance of perspective-correct texture application by using a non-orthogonal rasterization direction of a polygon-bounded area is proposed. Rasterization of an object in the world coordinate system is carried out under the condition that the rasterization lines are displaced at a fixed distance from the observer. For a triangle containing T internal points, $(T\text{-}q)$ division operations are performed, where q is the number of horizontal rasterization lines of the triangle.

7. A method for rasterizing a rectangular window placed at an arbitrary angle to the coordinate axes is developed, the peculiarity of which is that during the formation of the current rasterization line of the window, the step movement is remembered, which is used to move to the next rasterization line. This allows you to simplify hardware implementation by using one linear interpolator instead of two.

8. A method for increasing the performance of perspective-correct texturing for software components of graphic systems by means of non-orthogonal rasterization is proposed, which allows to reduce the number of division operations. The number of division operations per polygon is only $(q+1)$, where q is the number of non-orthogonal rasterization lines.

4.11. REFERENCES

1. Matt Pharr, Wenzel Jakob, and Greg Humphreys. Physically Based Rendering, fourth edition: From Theory to Implementation. USA: MIT Press, 2023.
2. O. N. Romanyuk, and A. V. Chornyi. High-performance methods and means of painting three-dimensional graphic objects. Monograph. Vinnytsia: UNIVESUM-Vinnytsia 2006.

3. T. Möller, B. Hamann, and R. Russell Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration. Springer; 2009. 360 p.
4. O. N. Romanyuk, O. V. Romanyuk, and R. Yu. Chekhmestruk. Computer graphics.VNTU, 2023.
5. O. N. Romanyuk, I. V. Abramchuk, and O. O. Dudnyk, "Anisotropic filtering using a weight function based on the Gaussian pixel model", Measuring and computing equipment in technological processes, No. 2, pp. 117–121, 2016.
6. O. N. Romanyuk, and O. O. Dudnyk, "Anisotropic filtering using texture maps of weighting coefficients", Registration, storage and processing of data, Vol. 19, No. 3, pp. 34–41, 2017.
7. O. N. Romanyuk, A. A. Dudnyk, O. V. Romanyuk, and D. A. Ozerchuk. Method of perspective-correct texturing, Information technologies and computer engineering", No. 1, pp. 55–63, 2021.
8. O. V. Romaniuk. Productive methods of quadratic approximation of perspective-correct texturing, Bulletin of the East Ukrainian National University Volodymyr Dahl East Ukrainian National University, No. 10 (152), pp. 194–198, 2010.
9. O. V. Romaniuk. High-productive method of perspective-correct texturing, Bulletin of Vinnytsia Polytechnic Institute, № 2, pp. 74–77, 2010.