



**International
Center for Science
and Social
Transformation**



THEORETICAL AND PRACTICAL APPLICATION OF THE RESULTS OF MODERN SCIENCE

COLLECTIVE MONOGRAPH

www.isst.co.ua

SCIENTIFIC MULTIDISCIPLINARY MONOGRAPH

**«THEORETICAL AND PRACTICAL APPLICATION OF THE RESULTS OF
MODERN SCIENCE»**

The collective monograph is a scientific and practical publication containing scientific articles by Doctors and Candidates of Sciences, Doctors of Philosophy and Arts, postgraduate students, applicants for higher (professional pre-higher) education, student researchers and practical worker from European and other countries. The articles contain research that reflects current processes and trends in the development of world science.



Recommended for printing and publication on the Internet
(Decision № 14-2025/1)
www.isst.co.ua

Published by Primedia eLaunch
<https://primediaelaunch.com/>

©Text Copyright 2025 by the International Center for Science and Social Transformation (isst.co.ua) and authors.

©Cover art: International Center for Science and Social Transformation (isst.co.ua).

©All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, distributed, or transmitted, in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher. When using and borrowing materials, reference to the publication is required.

Reviewers: Olena ARSHA, Maksym KAISTRO

The texts of scientific materials are published in the author's version with minor corrections (without changing the meaning). The authors and their supervisors are responsible for the accuracy of the information provided.

UDC 001.1

ISBN – 979-8-89814-999-4

The materials are approved by the
scientific and metrological base:



TABLE OF CONTENT

ВИКОРИСТАННЯ ІННОВАЦІЙНИХ МЕТОДІВ У НАВЧАЛЬНОМУ ПРОЦЕСІ ЯК ІНСТРУМЕНТ ПІДВИЩЕННЯ ЯКОСТІ ОСВІТНІХ ПОСЛУГ.....	6
МЕТОДОЛОГІЯ ПЕДАГОГІЧНОГО ФОРМУВАННЯ НАУКОВИХ КОМПЕТЕНТНОСТЕЙ ЗДОБУВАЧІВ ОСВІТИ ЗАСОБАМИ ОСВІТНІХ ПРОГРАМ І ТРЕНІНГОВИХ ПРАКТИК	18
CROSS-CURRICULAR INTEGRATION AND PROFESSIONAL ORIENTATION OF THE CONTENT OF FOREIGN LANGUAGE TEACHING FOR VETERINARY STUDENTS	33
PERSONALITY-BASED APPROACH TO PROFESSIONAL TRAINING OF FUTURE LAWYERS IN CANADA	51
САМОРЕГУЛЯЦІЯ ЗДОБУВАЧІВ ОСВІТИ ЯК ЧИННИК ЕФЕКТИВНОГО НАВЧАННЯ У ЗАКЛАДАХ ВИЩОЇ ОСВІТИ.....	56
САМОСТІЙНА РОБОТА ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ ПІД ЧАС ВИВЧЕННЯ ЛАТИНСЬКОЇ МОВИ	65
FEATURES OF IMPLEMENTING PERSPECTIVE-CORRECT TEXTURING	71
INNOVATIVE APPROACHES TO SURFACE HARDENING OF MATERIALS USING SHS TECHNOLOGIES	84
THE ROLE OF COMPUTER GRAPHICS AND VISUALIZATION IN THE TRAINING OF STUDENTS AND PUPILS.....	96
АДСОРБЦІЙНЕ ОЧИЩЕННЯ СТИЧНИХ ВОД ФАРБУВАЛЬНОГО ВИРОБНИЦТВА ТЕКСТИЛЬНИХ МАТЕРІАЛІВ З ПОВТОРНИМ ЇХ ВИКОРИСТАННЯМ В ТЕХНОЛОГІЇ ФАРБУВАННЯ БАВОВНЯНИХ ТКАНИН	123
ENVIRONMENTAL SAFETY OF ELECTROCHEMICAL FORMATION OF PROTECTIVE COATINGS ON ENERGY EQUIPMENT	147
МЕТОД АДАПТИВНОГО ХАОТИЧНОГО КОДУВАННЯ ГРАФІЧНИХ ЗОБРАЖЕНЬ.....	173
СТРАТЕГІЧНИЙ АНАЛІЗ КАДРОВОГО ПОТЕНЦІАЛУ ПІДПРИЄМСТВА	188
THEORETICAL AND APPLIED APPROACHES TO IMPROVING THE METHODOLOGY FOR ASSESSING AND MANAGING INVESTMENT RISK UNDER CONDITIONS OF UNCERTAINTY	194

DIGITAL TRANSFORMATION AS A TOOL FOR RISK MITIGATION IN SUSTAINABLE ENTREPRENEURSHIP DURING ECONOMIC RECOVERY	202
WYZWANIA I PERSPEKTYWY WDRAŻANIA NOWOCZESNYCH TECHNOLOGII W BIZNESIE: OD SZTUCZNEJ INTELIGENCJI PO BLOCKCHAIN	254
ЕТИКА ДІЛОВОГО СПІЛКУВАННЯ ЯК СТРАТЕГІЧНИЙ РЕСУРС КОМУНІКАТИВНОГО МЕНЕДЖМЕНТУ НА ПІДПРИЄМСТВАХ СФЕРИ ГОСТИННОСТІ ТА ХАРЧОВОГО БІЗНЕСУ	263
АРХЕТИП МАТЕРІ ЯК РЕСУРС ПСИХОЛОГІЧНОЇ ПІДТРИМКИ У ПРОФЕСІЙНІЙ ПІДГОТОВЦІ МАЙБУТНІХ ФАХІВЦІВ ДОПОМІЖНИХ ПРОФЕСІЙ.....	270
СПРИЙНЯТТЯ І СЛОВО ЯК ЧИННИКИ ПСИХОЛОГІЧНОЇ КОМУНІКАЦІЇ ХУДОЖНИКА.....	275
ПРАКТИЧНИЙ ПІДХІД ДО ФОРМУВАННЯ ОСОБИСТОСТІ СУЧАСНОГО МИТЦЯ ТА ЙОГО ТВОРЧОГО І ФІНАНСОВОГО УСПІХУ	282
СТРАТЕГІЯ ВИПРОБУВАНЬ ДЛЯ ОЦІНКИ ЕКОТОКСИКОЛОГІЧНИХ ВЛАСТИВОСТЕЙ ВІДХОДІВ.....	288
POSTCOLONIAL NARRATIVES AND COLLECTIVE TRAUMA IN UKRAINIAN, EUROPEAN, AND AMERICAN DRAMA: A COMPARATIVE ANALYSIS.....	293
ENERGY EFFICIENCY IN UKRAINE: CURRENT STATUS AND FUTURE PROSPECTS.....	301
THE ROLE OF TRICHODERMA IN BIOLOGICAL CONTROL: BIOTECHNOLOGICAL BASES AND PROSPECTS	320

ТЕХНІЧНІ НАУКИ**Romaniuk Oleksandr**Doctor of Technical Sciences, Professor, Vinnytsia National
Technical University**Stakhov Oleksiy**Candidate of Technical Sciences, Senior Lecturer, Vinnytsia
National Technical University**FEATURES OF IMPLEMENTING PERSPECTIVE-CORRECT TEXTURING**

Texturing for rendering is the process of overlaying images or patterns on the surface of three-dimensional objects to give them visual properties such as color, detail, relief, transparency, or reflectivity. It is one of the most important stages of graphics rendering, allowing you to get a realistic or stylized image without having to create overly complex geometry.

One of the important tasks in computer graphics when implementing realistic images of three-dimensional scenes is the correct application of textures to geometric surfaces. Among the most accurate methods of displaying visual information is perspective-correct texturing, which ensures the correct display of textures taking into account perspective distortions in the process of rasterizing polygons.

The basic problem that perspective-correct texturing solves is that simple linear interpolation of texture coordinates using barycentric coordinates or screen coordinates leads to visual artifacts. This is especially noticeable when rendering large triangles that occupy a significant area on the screen, or when polygons are strongly tilted relative to the camera. In such cases, without correction, textures can appear blurry or distorted¹.

¹ Khomyak, I. V. (2022). Three-dimensional rendering algorithms . Lviv: Ivan Franko National University of Lviv.

Perspective-correct coordinate interpolation is performed by transitioning to a homogeneous space and performing operations on coordinates that take into account the w component, the fourth coordinate resulting from perspective transformations. In the calculation process, the texture coordinates (u, v) and depth (z) are divided by w at the vertex stage, after which the values u/w , v/w , z/w are interpolated. After the interpolation at the pixel level is completed, these values are multiplied by w to restore the true texture coordinates².

This approach ensures accurate texture matching to the geometry of objects in three-dimensional space, especially with strong perspective distortions. Perspective correction algorithms are widely implemented in modern graphics APIs such as OpenGL and DirectX, as well as in GLSL and HLSL shader languages, allowing graphics programmers to manually adjust the interpolation process if necessary³.

In modern graphics engines, such as Unreal Engine or Unity, perspective-correct texturing is a standard feature in the rendering pipeline. However, in cases where non-standard shaders or new methods of lighting, depth, or object transformation are implemented, the interpolation process needs to be manually controlled. This is particularly true for procedural rendering, where computational efficiency and texturing accuracy can be critical to the performance of the scene⁴.

It is worth noting that perspective-correct texturing is not limited to planar textures. In modern methods such as parallax displacement, displacement mapping, and relief mapping, it is also important to adhere to the principles of perspective correction to ensure the visual integrity of the result. This is especially important in VR/AR applications, where inaccuracies in textures can create discomfort for the user or reduce the realism of the scene⁵.

Research in the field of perspective-correct texturing is aimed both at optimizing computations for mobile platforms and at expanding the application for complex

² Melnyk, Y. V. (2020). *Fundamentals of Graphical Programming in GLSL*. Kharkiv: Kharkiv National University of Radio Electronics.

³ Soroka, M. O. (2021). *Interactive visualizations based on OpenGL*. Odessa: Odessa National Polytechnic University.

⁴ Kulyk, P. M. (2021). *Fundamentals of Computer Graphics: A Textbook*. Kyiv: Aristei.

⁵ Romanyuk, O. N., & Romanyuk, O. V. (2023). *Computer Graphics [Electronic Textbook]*. Vinnytsia: Vinnytsia National Technical University.

shader models. In particular, approaches to hardware acceleration through the use of the latest GPU architectures and intelligent methods for reducing the load on the rasterization pipeline are considered⁶.

The most basic approach to texture overlay is affine texturing, which is implemented by linearly interpolating the texture coordinates (u, v) within the plane of a triangle. This method does not take into account perspective reduction and is effective only when the surfaces are relatively flat and parallel to the view plane. Affine texturing is fast to compute and is suitable for older game engines or mobile devices with limited resources. However, affine interpolation produces noticeable artifacts when the view angle changes, especially in areas of depth⁷.

To ensure correct visual display of the texture, perspective-correct interpolation is used. This approach assumes that the texture coordinates (u, v), as well as the depth (z) and other parameters, must be normalized using the w coordinate - the fourth coordinate that appears after transformation into homogeneous space. At the vertex processing stage, the coordinates are divided by w : i.e. $u/w, v/w, z/w$. These values are interpolated linearly at the rasterization stage. Then, after interpolation, the inverse operation of multiplication by w is performed, i.e. $(u/w) \times w = u$. This method provides a correct image even on strongly inclined or distant surfaces⁸.

One of the options for implementing perspective correction is the barycentric interpolation method taking into account w . It is based on the fact that the texture coordinates are calculated using the formula:

$$u = (\alpha \times (u_1 / w_1) + \beta \times (u_2 / w_2) + \gamma \times (u_3 / w_3)) / (\alpha / w_1 + \beta / w_2 + \gamma / w_3),$$

where α, β, γ are the barycentric coordinates for each pixel of the triangle, and u_1, u_2, u_3 are the texture coordinates at the vertices, corresponding to w_1, w_2, w_3 - the values of the w coordinate for each vertex. The v coordinate is calculated similarly.

⁶ Khomyak, I. V. (2022). Three-dimensional rendering algorithms. Lviv: Ivan Franko National University of Lviv.

⁷ Khomyak, I. V. (2022). Three-dimensional rendering algorithms. Lviv: Ivan Franko National University of Lviv.

⁸ Romanyuk, O. N., & Romanyuk, O. V. (2023). Computer Graphics [Electronic Textbook]. Vinnytsia: Vinnytsia National Technical University.

This method allows to achieve high accuracy and is used in software rasterizers and custom graphics engines⁹.

In graphics programming languages such as GLSL or HLSL, implementing perspective-correct texturing in shaders looks like this. At the vertex shader stage, the texture coordinates are divided by `gl_Position.w`, and the value $1.0 / w$ is also stored. Then, in the fragment shader, the interpolated values are multiplied by z to restore the true coordinates. This approach allows for flexible control of accuracy and allows for unique effects such as parallax shifting, reflections, or depth-aware shading¹⁰.

It is worth highlighting another method - perspective-correct calculation of the texture level of detail (LOD) when using mipmaps. The optimal texture level is determined based on the partial derivatives of the coordinates u/z and v/z in x and y . In GLSL, the functions `dFdx` and `dFdy` are used for this, which take into account distortion at an angle. Without this correction, artifacts are possible: choosing a texture that is too detailed (memory overruns) or not detailed enough (blurring)¹¹.

To improve performance, hybrid texturing methods are used in some cases, where perspective is only partially taken into account. For example, in the background of a scene, where distortions are not noticeable, simplified affine interpolation can be used, and in the foreground or for large objects, full perspective correction. This allows saving GPU resources without significant loss of quality¹².

In some high-precision systems, another method is used - perspective correction through depth (z -correction). It is especially useful in scientific visualization or in augmented reality systems, where it is important that the texture exactly matches the geometry of the scene regardless of the viewing angle. The change in the z value is taken into account when calculating not only the depth of the fragment, but also the texture coordinates, which eliminates microscopic errors¹³.

⁹ Khomyak, I. V. (2022). Three-dimensional rendering algorithms. Lviv: Ivan Franko National University of Lviv.

¹⁰ Melnyk, Y. V. (2020). Fundamentals of Graphical Programming in GLSL. Kharkiv: Kharkiv National University of Radio Electronics.

¹¹ Romanyuk, O. N., & Romanyuk, O. V. (2023). Computer Graphics [Electronic Textbook]. Vinnytsia: Vinnytsia National Technical University.

¹² Kulyk, P. M. (2021). Fundamentals of Computer Graphics: A Textbook. Kyiv: Aristei.

¹³ Soroka, M. O. (2021). Interactive visualizations based on OpenGL. Odessa: Odessa National Polytechnic University.

Thus, perspective-correct texturing methods cover a wide range of techniques - from basic division of coordinates by w to complex barycentric interpolations with consideration of derivatives. Each of them has its place in practice - depending on the tasks, system performance, rendering accuracy and scene type. Perspective-correct texturing is an integral part of modern graphics APIs, and its correct implementation determines the quality of visualization in video games, VR/AR applications, technical graphics and scientific data visualization.

In modern GPUs, perspective-correct texturing is implemented at the hardware level using specialized function blocks that are part of the graphics pipeline. This process ensures an exact correspondence between texture coordinates and the 3D geometry of the scene, taking into account perspective distortions. Without perspective correction, texture overlay leads to distortions, especially on polygons that are tilted relative to the plane of view or in scenes with great depth.

The first stage is the transformation of geometric coordinates into a homogeneous space in the vertex shader. After model transformation, view and projection transformation, each vertex acquires four-dimensional coordinates (x, y, z, w) , where w is the scale component that reflects the perspective effect. The vertex attributes, in particular the texture coordinates u and v , are also divided by w . Thus, the values of u/w , v/w and $1/w$ are calculated in the vertex shader for each vertex. These values are then passed to the next processing stage - rasterization.

In the rasterizer block, which is responsible for constructing the triangle pixels, the texture coordinates are interpolated taking into account the barycentric coordinates of the pixel. The video card interpolates not the u and v coordinates themselves, but the u/w , v/w and $1/w$ values. This allows you to achieve linear interpolation in a homogeneous space, which corresponds to the real change in the texture space during perspective reduction. For each pixel, the summed u/w value and the corresponding $1/w$ value are calculated. Then, in the fragment shader, the texture coordinates are inversely restored by dividing:

$$u = (u/w) / (1/w) = u,$$

$$v = (v/w) / (1/w) = v.$$

Thus, the texture coordinates correspond exactly to the position of the pixel on the surface of the 3D object, taking into account the perspective projection. This avoids the distortions observed in simple affine texturing, where the coordinates are interpolated without taking into account the w -component. At the hardware level, these operations are implemented in specialized GPU modules, called attribute interpolators. They support interpolation of an arbitrary number of parameters with perspective correction and work in parallel with the rasterization blocks.

After determining the correct texture coordinates, the video card calls the texture sampling unit. At this stage, the texture color is determined by the u and v coordinates. If filtering is enabled, neighboring texels are sampled taking into account the smoothing method - bilinear, trilinear or anisotropic. In the case of using mipmaps, the system determines the level of texture detail based on the partial derivatives of u and v with respect to x and y . To ensure accuracy even in perspective space, the derivatives are also calculated from the u/z , v/z values. This allows you to correctly choose the mipmap level even on very inclined surfaces.

In graphics APIs such as OpenGL, DirectX, or Vulkan, perspective-correct texturing is enabled by default. The programmer simply declares the attributes in the vertex shader, and the GPU automatically performs the necessary transformations and interpolation. For example, in OpenGL, attributes of type `in vec2 texCoord;` are interpolated taking perspective into account. In HLSL (DirectX), the linear modifiers are used for this, which are responsible for perspective-correct interpolation by default.

At the hardware level, all of these calculations are optimized for maximum performance. The GPU uses parallel data processing with vectorization, SIMD (single instruction, multiple data) architecture, and caching of interpolated values. This allows for the interpolation of dozens of attributes for millions of pixels every second with minimal hardware overhead.

In most modern graphics cards, such as NVIDIA GeForce, AMD Radeon or Intel Arc, perspective-correct interpolation blocks are part of the fixed-function rasterizer and do not require manual implementation. However, when developing specialized shaders or non-standard rendering scenarios, the programmer can explicitly control the calculation of texture coordinates, independently performing division by w in the vertex shader and restoration in the fragment shader. This approach is used, in particular, in cases of implementing parallax displacement effects, displacement mapping or procedural surface deformation.

In general, perspective-correct texturing in video cards is implemented as a standard and mandatory part of the graphics pipeline. Its implementation is based on precise mathematical principles that are effectively implemented in specialized GPU blocks. Due to this, computer graphics provide a high level of visual realism, reliable depth and an exact correspondence between the geometry of objects and the textures applied to them.

Texturing in computer graphics is the process of superimposing a two-dimensional image (texture) onto the surface of a three-dimensional object in order to create a more detailed and realistic image. There are two main approaches to texturing: conventional, or affine, texturing, which is based on linear interpolation of texture coordinates, and perspective-corrected texturing, which takes into account the distortions that occur when a scene is projected in perspective. The key difference between these methods is the computational complexity and the quality of the visual result.

Affine texturing is the simplest method. It interpolates texture coordinates along the plane of the triangle, regardless of its orientation in space. The calculations are limited to adding and multiplying coordinates within the triangle without taking into account depth components or perspective deformation. During rendering, the u and v coordinates are calculated for each pixel belonging to the polygon using a simple linear formula. This approach has a low load on the GPU, as it does not require division, inverse scaling, or other complex arithmetic operations. On older or resource-

constrained systems, affine texturing was dominant due to its speed. However, this method does not provide correct texture curvature that corresponds to the perspective of the scene. As a result, textures on distant or tilted objects appear distorted - too stretched or compressed, especially at the edges of the image.

Perspective-correct texturing, on the other hand, involves much more complex calculations. The essence of this method is to use homogeneous coordinates. In the vertex shader, each texture coordinate u and v is divided by the coordinate w , which is the result of the perspective transformation. Thus, the GPU passes the values u/w , v/w , and $1/w$ to the rasterizer. In the fragment shader, these values are interpolated taking into account the barycentric weights of the pixel, and then restored by dividing $u = (u/w) / (1/w)$ and $v = (v/w) / (1/w)$. This approach ensures an exact correspondence of textures to objects, taking into account changes in perspective and depth, which allows you to avoid the typical artifacts of the affine method. But in return, it requires more operations per pixel - not only addition and multiplication, but also at least two divisions and inverse scaling.

In terms of computational complexity, affine texturing is significantly faster: the GPU only needs to process linear interpolations without additional transformations. Perspective correction is about 2–3 times more expensive in terms of arithmetic operations per fragment. This increase is due not only to the more complex interpolation, but also to the need to store and process more intermediate values, including $1/w$, u/w , and v/w . On older or mobile graphics systems, this could lead to a 10–20% performance penalty with a large number of textured pixels.

However, in modern graphics cards, the implementation of perspective-correct texturing is carried out at the hardware level. Special functional blocks in the GPU - interpolation modules, rasterizers and texture processors - perform the necessary operations in parallel, using SIMD architecture (single-instruction, multiple-data), vectorized calculations and caching of repeated values. Due to this, even complex calculations associated with division by w do not significantly affect the overall performance of the graphics pipeline. In graphics APIs such as OpenGL or DirectX,

perspective-correct texturing is the default mode, and the programmer does not need to manually implement these operations unless he uses non-standard interpolation.

It should be noted that in terms of labor intensity, affine texturing is a much simpler and less expensive method, but it is inferior in image quality. Perspective-correct texturing, although it requires more computing resources, provides high reliability of texture display in three-dimensional space. In the conditions of modern hardware, the costs of perspective correction are almost completely compensated by hardware optimization, so its use is justified even in real time, in particular in games, VR applications and engineering visualization.

In modern graphics processing units (GPUs), perspective-correct texturing is implemented automatically at the level of specialized hardware modules that are part of the graphics pipeline. These blocks work together to provide accurate interpolation of texture coordinates taking into account perspective distortion. The main hardware components involved in this process are discussed below:

Perspective-correct texturing in modern graphics processing units (GPUs) is implemented by integrating specialized hardware blocks that provide accurate interpolation of texture coordinates according to the perspective distortion of the scene. All major processing stages - from vertex transformation to texture sampling - support the calculations necessary for perspective correction, making this process highly accurate and at the same time optimized for parallel processing.

The first stage of processing is the vertex shader unit. At this stage, the coordinates of each vertex (x, y, z, w) are transformed from local space to clip space using a projection matrix. Along with the spatial coordinates, the vertex attributes are also passed, in particular the texture coordinates u and v . They are either directly divided by w in the vertex shader, or passed as a pair (u, w) for further interpolation. Thus, already at the first stage, the GPU prepares the data for perspective-correct interpolation.

The next stage is the rasterization and interpolation block, or rasterizer and attribute interpolator. This is where the key principle of perspective correction is

implemented. Each triangle is divided into pixels, and all the transferred attributes, including u and v , are interpolated taking into account the barycentric coordinates (α , β , γ). In the case of perspective interpolation, the GPU does not work with u and v themselves, but with the values u/w , v/w and $1/w$. The interpolation for a pixel is performed according to the formula:

$$u = (\alpha \times (u_1 / w_1) + \beta \times (u_2 / w_2) + \gamma \times (u_3 / w_3)) / (\alpha / w_1 + \beta / w_2 + \gamma / w_3)$$

and similarly for v . This formula ensures the correct warping of the texture taking into account perspective projection. It is the hardware interpolation module that performs these calculations, automatically applying perspective correction to each pixel included in the triangle.

After interpolation, the computed texture coordinates are fed into the fragment shader core, where final processing occurs before the texture is accessed. If u/w and $1/w$ coordinates were passed separately, the shader performs the inverse scaling using the formula:

$$u = (u / w) / (1 / w) = u$$

The same applies to the v coordinate. The resulting values are used to select the appropriate texture element. In the case of more complex shaders or procedural texturing, the programmer can manually implement these operations for more flexible control, but in most cases everything is done automatically in hardware mode.

The fragment shaders then pass the coordinates to the texture unit. This unit accesses the texture memory and retrieves the color value based on the u and v coordinates. The texture unit also performs filtering (e.g., bilinear or anisotropic) and calculates the level of detail (LOD), which takes into account the du/dx , dv/dy derivatives, but with perspective deformation taken into account, since the coordinates were correctly adjusted in the previous stages.

All of these blocks operate within the GPU's parallel architecture, which allows it to process thousands of fragments simultaneously. The parallel SIMD (single instruction, multiple data) architecture and the use of caches allow for reduced overhead even with a large number of divisions and backscaling. Additionally, thread

schedulers and instruction dispatching blocks are used to distribute interpolation calculations between GPU cores as efficiently as possible.

Thus, perspective-correct texturing in the GPU is implemented in a complex and hardware way. The main performers of this function are the vertex processing units (which prepare u/w), the rasterizer with an interpolation module (which calculates barycentric weights and performs perspective interpolation), fragment shaders (which perform inverse scaling), and the texture module (which applies the obtained coordinates and filters the texture). Due to the hardware implementation of this complex mathematics, the GPU allows you to achieve high visual accuracy of textures without a significant impact on performance even in real-time rendering mode.

Perspective-correct texturing has long been an integral part of 3D computer graphics, providing accurate texture overlays that take into account perspective distortions. However, despite its widespread hardware implementation and standardization in graphics APIs, this method continues to evolve in response to increasing demands for visual quality, performance, and integration with new rendering technologies. The prospects for the development of this approach are associated with several key areas.

First, with the proliferation of high-quality real-time rendering, especially in the context of virtual and augmented reality, there is a need for even more precise control over the interpolation of texture coordinates. Modern VR/AR displays have high pixel densities and small subpixel sizes, so even minor texturing errors, particularly at geometry boundaries or during fast camera movement, can be noticeable to the user. In this context, adaptive perspective-correct texturing methods are being developed that use hybrid strategies: for example, automatic switching between accurate and optimized texturing depending on the scene area, camera distance, or viewing angle.

¹⁴ Romanyuk, O. N., Dudnyk, O. O., Romanyuk, O. V., & Ozerchuk, D. A. (2021). Perspective-correct texturing method. *Information Technologies and Computer Engineering*, (1), 55–63.

¹⁵ Romanyuk, O. N., Dudnik, O. O., Snigur, A. V., Reida, O. M., & Romanyuk, O. V. (2021). Peculiarities of vector normalization in perspective-correct color reproduction. *Scientific works of DonNTU. Series: Informatics, cybernetics and computing technology*, (1[32]), 11–17.

Second, perspective-correct texturing is integrated with new lighting and shading algorithms. In modern physically based rendering (PBR), it is important that all surface attributes, including normals, roughness, specular, and albedo, are spatially consistent. This requires interpolating not only the base texture coordinates but also their derivatives, as well as ensuring that levels of detail (LOD) in mipmaps are correctly calculated. New GPU architectures allow derivatives to be calculated with greater precision, which opens the way to dynamically adjust the level of perspective correction.

Another promising direction is the introduction of machine learning into the texturing process. Models are already being studied that, based on a preliminary analysis of the scene, predict in which areas of the image an exact correction is necessary, and where it can be replaced by an approximate version. For example, neural networks can detect areas of high visual attention (foveated rendering) and increase the accuracy of perspective-correct texturing in them, reducing resources for peripheral areas.

Of particular note is the development of perspective correction for irregular meshes and curved surfaces. Traditional methods mainly use flat polygons (triangles), but the growing popularity of surface subdivision, Bezier curves, NURBS, and other complex geometric primitives requires new approaches to interpolation. In this context, perspective correction options for parametric and implicit surfaces are being developed that take into account not only the position and w-component, but also the geometric derivative of the surface.

Also important is the development of methods for accurate texturing when objects are deformed - for example, in the case of skeletal animation, morphing or physically based simulations. Here, the creation of a correct w-component transfer during deformation is relevant, which guarantees the preservation of correspondence between geometry and texture even with significant changes in shape.

¹⁶ Romanyuk, O. N., & Dudnik, O. O. (2016). Improving the performance of perspective-correct texturing using anisotropic filtering. *Measuring and computing equipment in technological processes*, (3), 192–195.

¹⁷ Romanyuk, O. N., & Dudnik, O. O. (2016). Method for increasing the productivity of perspective-correct texturing. *Scientific works of DonNTU. Series: Informatics, cybernetics and computing*

No less promising is the direction of optimizing perspective-correct texturing in mobile and power-constrained devices. Since mobile GPUs have limited memory bandwidth and power budget, research is underway to simplify the calculations without losing visual quality, for example, through the use of lookup tables, approximations, or pre-encoding perspective deformation in texture space.

In conclusion, perspective-correct texturing is not a finished and frozen technology. It continues to actively develop in the direction of increasing accuracy, adaptability, integration with new graphics technologies, the use of artificial intelligence, support for complex surfaces and reducing power consumption. All this indicates its continued importance for future 3D visualization systems, in particular in the field of interactive graphics, game engines, engineering modeling and virtual environments.