UDC 004.925.8 + 519.6

MODIFY SHADERS AND RENDER TEXTURES ON CURVED SURFACES

Suhoniak I. – Associate Professor, PhD in Technical Sciences, Associate Professor of the Department of Computer Science, State University "Zhytomyr Polytechnic", Zhytomyr, Ukraine.

Marchuk G. – Senior Lecturer of the Department of Computer Science, State University "Zhytomyr Polytechnic", Zhytomyr, Ukraine.

Oleksiuk O. – Applicant of the Department of Computer Science, State University "Zhytomyr Polytechnic", Zhytomyr, Ukraine.

ABSTRACT

Context. The display of curvilinear surfaces on flat screens is a complex task. The development of an interface for such surfaces is a relevant task that requires the solution of numerous issues. This paper presents an approach to UI development for curvilinear surfaces and shader modifications for the creation of realistic landscape elements. The object of the research is the development of an interface system based on a custom raycaster to ensure interactivity and create an immersive effect within the game environment..

Objective. The purpose of the paper. The primary objective of this research is to create, optimise and adapt shaders on curved surfaces to achieve more efficient rendering with high-quality visualisation.

Method. The development of user interfaces (UI) for curvilinear surfaces requires consideration of geometric parameters. To resolve this issue, a custom component based on BaseRaycaster was developed, enabling the computation of the intersection between the camera ray and the physical surface. To provide correct and efficient interaction with the canvas a custom component based on BaseRaycaster was created. The developed component solves the problem by identifying the ray intersection point from the camera with the canvas surface. The implementation of this component involves an algorithm for detecting the camera's ray intersections with colliders, using a mathematical model to process the detected elements, according for their depth to ensure proper interaction.

Results. This approach facilitates the creation of interfaces on arbitrary static curved surfaces that are applicable in various gaming and interactive scenarios.

Conclusions. The use of splines and modified shaders ensures the placement of text on curvilinear surfaces and the natural arrangement of roads and other landscape elements according to the terrain contours. This approach is important for developing open-world games or games with complex geometry, where the UI on curvilinear surfaces appears natural and integrated into the environment.

KEYWORDS: game development, shaders, spline, rendering, custom raycaster, curvilinear surfaces, user interface, visualization.

ABBREVIATIONS

UI – User Interface.

NOMENCLATURE

C_p is a position of the camera in 3D space;

C_d is a direction of the beam from the camera;

C_s is a parameters of the collider;

P_c is a collider parameters;

P_n is a normal parameters;

S is a set of surfaces in 3D space, where each surface $s \in S$ has a collider C_s;

T_w is a width of the canvas texture;

T_h is a height of the canvas texture;

UI is a set of UI elements on the texture, where each element $ui \in$ UI;

$T_{c,\min}$ is a minimum coordinates on the texture;

$T_{c,\max}$ is a maximum coordinates on the texture;

T_{px} is a coordinates of the intersection point on the texture of each UI element along the x-axis;

T_{py} is a coordinates of the intersection point on the texture of each UI element along the y-axis;

z-depth is a depth of the element;

P_i is a control points for creating splines;

I_p is a point of intersection of the ray with the surface;

T_c is a coordinates of the intersection point on the texture;

ui is a active UI element;

R_c is a ray from the camera;

I_{px} is a coordinates of the intersection point on the surface along the x-axis;

I_{py} is a coordinates of the intersection point on the surface along the y-axis;

Lerp is a interpolation of textures based on the alpha channel to create smooth transitions between pre-horn segments.

INTRODUCTION

In the modern world of 3D games, user interaction with the virtual environment plays a vital role in ensuring the quality of the user experience. Interfaces are often displayed on the main screen's surfaces, which does not allow maximum immersion in the game. Still, in cases where the interface is located on the curved surface of an object, additional implementation difficulties arise. Texture rendering allows you to create such interfaces, but ensuring accurate and intuitive interaction on curved surfaces is more complex, as standard interaction processing methods are designed for flat surfaces.

The object of research. The study focuses on creating an interface system using a specially developed ray tracing method (raycasting) to increase the player's interactivity and immersion in the game world.

The placement of the interface on a curved surface is an essential aspect of modern game design, especially in terms of maximum immersion in the game world, which allows for a more organic and realistic interaction between the player and the game world. The use of rendered textures and modified shaders opens up new opportunities for integrating UI elements into complex three-dimensional environments, increasing the level of immersion and aesthetic integrity of the game.

DOI 10.15588/1607-3274-2025-4-19

The subject of study. Modern computer graphics methods allow you to create virtual worlds with a high degree of realism. However, modeling curved surfaces often found in nature and technology remains challenging. Traditional methods based on polygonal approximation can lead to loss of detail and shape distortion, so there is a need to develop effective methods for modifying shaders and rendering textures that would allow accurate and realistic display of curved surfaces. The subject of study is adapting existing tools and developing new ones to reproduce realistic 3D images on varying complex surfaces.

The study aims to develop and adapt computer graphics methods for creating realistic 3D images on various curved surfaces.

1 PROBLEM STATEMENT

One key approach is to map the interface canvas to a texture, which allows it to be displayed on any surface, including curves or uneven planes. It is achieved by using a separate camera that displays only the necessary layer with the canvas. It renders it not on the screen but into a texture, which is subsequently used to overlay the object material. A shader then processes this rendered texture to realize the retro style using *ShaderGraph*.

For example, let's consider creating text on the screen of an old tube TV. Since the kinescope in such TVs has a convex shape, several visual effects must be applied to make the image realistic, such as additional geometry curvature (Fig. 1(1)); pixelation effect (Fig. 1(2)); noise and interference (Fig. 1(3)); stripes (Fig. 1(4)), which can be vertical or horizontal; displacement and freeze effect (Fig. 1(5)). To create the desired result, you must develop each effect separately and combine them into a single whole (Fig. 1(6)).

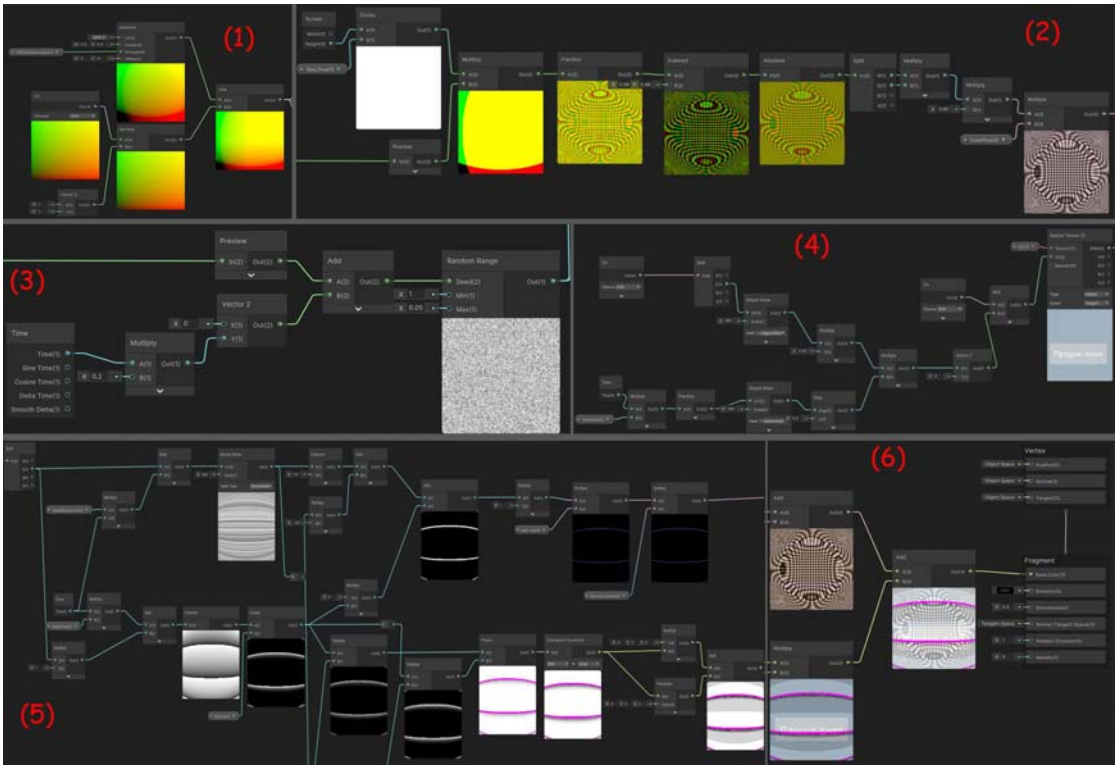

Figure 1 – Step-by-step parameters of the raider: screen curvature (1); pixelation (2); noise (3); stripes on the screen (4); image displacement and freezing (5); holistic picture (6)

Shaders are essential in creating a visual atmosphere and immersing the player in the game. However, the quality of shaders can vary, and they do not always meet high standards. Poor-quality shaders can ruin the atmosphere of the game and reduce the level of immersion. Poorly optimized shaders can lead to a decrease in game performance and freezes. Poor-quality shaders can make game graphics less attractive and detailed. It is also important to note that the textures generated by the raiders also mediate the player's interaction with the game world. Due to the use of a low-quality shader, the text on the screen of the old monitor is displayed with defects, as seen in Figure 2.

Figure 2 – The result of a low-quality shader

DOI 10.15588/1607-3274-2025-4-19

Shader development affects visual perception, as it determines how the player will perceive the game world's elements and how realistically they will be able to immerse themselves in it. However, it is essential to provide visual effects and ensure that the texture on which the shader is applied remains interactive. This means that the player should be able to interact with the objects covered by the texture with the active shader, for example, by clicking on interface elements or performing other actions.

Thus, the shader not only adds an aesthetic component but also considers functionality and interactivity, ensuring the usability of game elements. Developing shaders that combine effects to improve graphics and the possibility of interactive interaction is a complex but essential task to achieve complete immersion in the virtual world.

The algorithm for implementing a custom *Raycaster* component using splines can be represented as a sequence of stages and key elements that ensure the exact intersection of the beam with the surface, projection onto the texture, and construction of splines for modeling curves (Fig. 3).

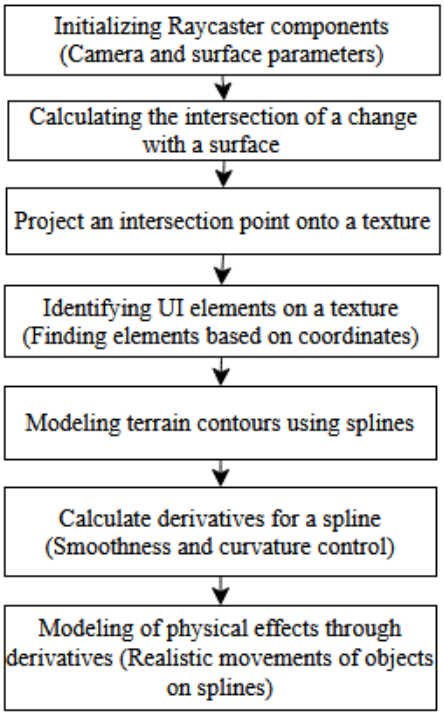

Figure 3 – Flowchart with the definition of the main stages and mathematical components

To initialize the Raycaster components, let the input data be given:

1. Camera parameters (position, beam direction).
2. Physical surface parameters (collider parameters, normal parameters).

The task of creating a Raycaster component for interacting with 3D objects is as follows:

– calculating the point of intersection of the ray with the surface: I_p=*raycast*(C_p,C_d,P_c);

– identifying UI elements on the texture: UI*list*=*findUI*(*Tc*);

– processing interaction with UI elements: *I*=*handleInteraction*(UI*active*);

– determining the intersection of the ray with the surface: T_c=*project*(I_p);

– applying depth to interact with UI elements: *I'*=*apply z-depth*(*i*) – where *i*=0, 1, ..., *n*.

2 REVIEW OF THE LITERATURE

In today's video games, shaders are critical to creating visual effects that make games more realistic and immersive. The book Building Quality Shaders for Unity® [1] will help beginners understand what shaders are and how they work. Game engines use different methods to process shaders and achieve the desired result.

Creating game worlds is a complex process requiring developers' considerable effort and resources. Shaders are tiny programs that run on the GPU and are responsible for image processing. The quality of shaders can vary and lead to rendering failures, which can cause objects to disappear, blink, or display incorrectly. They can significantly reduce performance, leading to low frame rates and freezes. The study's authors [2] analyzed 12 shader compilers from Intel, AMD, Nvidia, ARM, and Apple and found more than 16 thousand input data that cause errors affecting the rendering quality. These errors were identified and described, and the developers confirmed their typicality.

Article [3] explores in detail the technical aspects of using textures and shaders in various game engines to achieve various goals. The authors also provide recommendations on how to use these tools effectively.

Article [4] represents a research about combining mesh and shader levels of detail (LOD) using two special algorithms for leads to smoother visual transitions. The mesh LOD effectively reduces the quantity of polygons for distant objects, which hard load the GPU. Shader LOD, on the other hand, allows to reproduce or save small image details (such as texture nuances or complex normals) even on models with a reduced polygon count, thus improving visual quality without significantly increasing the complexity of the geometry

In article [5], the authors propose several improvements (Improved shader and texture detail level using ray cones) to the ray cone algorithm that improve image quality, increase performance, and simplify integration with game engines. The results show that a GPU-based tracer can reduce frame processing time by about 10%. The authors also provide an open-source implementation of this algorithm.

The paper [6] focuses on low-power optimization, which targets texture memory, geometry engine, pixel, and rasterization, as these components contribute significantly to a typical GPU's power consumption. The proposed methodology integrates the dynamic voltage frequency scaling (DVFS) technique, adjusting voltage and frequency based on analyzing the frame rate workload for 3D game scenes.

DOI 10.15588/1607-3274-2025-4-19

The authors of [7] present a tool for researching shader design using an interactive evolutionary algorithm integrated with the Unity editor. The framework uses an essential graph-based representation of the latest shader editors and interactive evolution to allow designers to explore multiple visual options starting from an existing shader. The framework encodes a graphical representation of the current shader as a chromosome used to spawn the evolution of a population of shaders. It applies graph-based recombination and mutation with a set of heuristics to generate possible raiders.

In [8], the authors presented a new particle-based fluid surface reconstruction method that includes mesh shaders for the first time. This approach eliminates storing triangular meshes in GPU global memory, significantly reducing the required memory. In addition, our method uses a bidirectional two-level uniform mesh, which speeds up the computationally expensive stage of surface cell detection and effectively solves the problem of vertex overflow among mesh shaders.

The authors of [9] propose using tetrahedra as primitives for volumetric rendering and a pipeline for their rasterization. The work relies on recently introduced mesh shaders to encode each tetrahedron so that the rasterizer calculates the depth of the front and back faces simultaneously when interpolating vertex attributes. However, the fragment shader receives two depths and can calculate its shading in the interval. The presented method is easy to implement and efficient, opening up new possibilities for the pasteurization pipeline.

Paper [10] presents a deep learning-based framework for predicting the shader simplification space, where shader variants can be immediately embedded into a metric space for efficient quality assessment. The new structure allows for a single embedding of the space rather than a single instance. In addition, simplification errors can be interpreted by the mutual attention between shader fragments, presenting an informative focus-aware simplification structure that can help experts optimize codes. The results show that the new framework achieves significant speedups over existing search approaches. The focus-aware simplification framework opens up new possibilities for interpreting shaders for different programs.

The article [11] discusses various ways to use a geometric shader for particle visualization. Geometric shaders are practical and flexible ways to visualize particles. They allow you to create a variety of visual effects with significant performance. This article is helpful for developers interested in visualizing particles using the GPU.

A study [12] used it to fit flash images to a model that can be efficiently evaluated in real-time using a custom shader in the Unity game engine. The custom shader accurately represents the object's mirror properties more accurately than the standard Unity default shaders, with little to no performance impact.

Due to the many options for using modern GPUs, it is usually difficult for engineers to manually check the many shader codes arising from these programs. To this end, the authors of [13] developed a framework that converts shader codes into graphs and uses sophisticated graph analysis and machine learning techniques in several applications to simplify the analysis of shader graphs efficiently and understandably, aimed at speeding up the entire debugging process and improving the overall performance of the hardware. The authors studied the evolution of shader codes by analyzing time graphs and structure analysis with frequent subgraphs, using them as the primary tools, and performing scene detection and selecting representative frames. The scene (program) was grouped to identify typical scenes and predict inefficient shaders of a new program.

3 MATERIALS AND METHODS

Developing a UI for curved surfaces is a complex task that requires considering the peculiarities of geometry and user interaction.

To provide interaction with the canvas, you should add a custom component based on *BaseRaycaster* (since the standard *GraphicRaucaster* cannot interact with objects in this way for several reasons: it is tightly tied to the main camera and not to the interactive object and does not take information from it), which determines the intersection of the beam from the camera with the physical surface. Given the intersection coordinates, it is necessary to determine the position on the canvas and a list of controls for further processing. The implementation includes searching for beam intersections with colliders using a mathematical formula and further processing the found elements, considering their depth for proper interaction. This approach allows you to create an interface on any curved static surface, which can be used for various gaming and interactive scenarios.

Implementing the custom Raycaster allows you to determine the camera beam's intersection with the object's physical surface. The Raycaster displays the intersection coordinates and is used to further search for UI elements (Fig. 4).

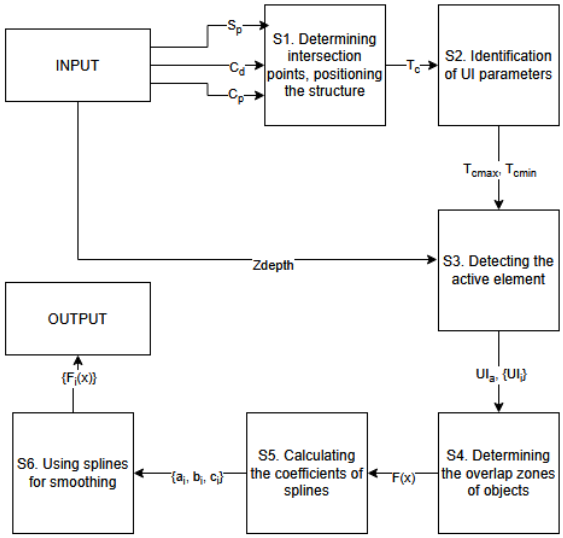

Figure 4 – A formalized spline interpolation model for smoothing planes

DOI 10.15588/1607-3274-2025-4-19

List of input variables:
– Camera position in 3D space;
– Camera ray direction;
– Set of surfaces in 3D space, where each surface $s \in S$ has a collider;
– Width and height of the canvas texture.

The UI is a set of UI elements on the texture, where each element $ui \in UI$ has a collider:
– Minimum and maximum coordinates on the texture;
– The depth of the element;
– Set of control points for the construction of splines.

List of output variables:
– Point of intersection of the ray with the surface;
– Coordinates of the intersection point on the texture;
– Active UI element.

Criteria:
– Maximum accuracy of determining the intersection point;
– Accurate projection of T_C onto the texture;
– Correct identification of the active UI element *ui*, considering *z-depth*;
– The smoothness and continuity splines;
– The realism of physical effects modeled using derived splines.

Limitations:
– Restrictions on computing resources;
– Restrictions on the time of operations;
– Restrictions on the accuracy of calculations;
– Restrictions on the complexity of splines.

The area where the beam intersects the physical surface is described by equation (1):

$$I_p = R_c \bullet C_s . \quad (1)$$

The point of intersection is projected onto the fabric texture with the following coordinates:

$$T_c = \left(\frac{I_{px}}{T_w}, \frac{I_{py}}{T_h} \right). \quad (2)$$

The resulting coordinates T_c define the position of the texture

After determining the coordinates on the texture, you can find the controls in this area and process them, considering the depth (for proper interaction with objects at different levels). Suppose rectangles or other geometric shapes represent the elements in the UI system. In that case, we can compare the resulting point T_c with the coordinates and dimensions of these elements on the texture.

Let's formalize the parameters of UI elements: $T_{c,\min}$ and $T_{c,\max}$ are the minimum and maximum coordinates of the UI element on the texture, defining its rectangular area.

If the point $T_c=(T_{cx}, T_{cy})$ is within the UI element, then this element is active:

$$T_{c,\min} \le T_c \le T_{c,\max} . \quad (3)$$

If several UI elements may overlap in the projection, you should add additional criteria for selecting an element, such as depth or layer priority. One of the most common ways to solve this problem is to use *z-depth*. *Z-depth* is a value that determines which element is higher than others on the screen. An element with a higher *z-depth* value appears on top of elements with a lower value:

$$z - depth = element\ depth\ UI . \quad (4)$$

For correct interaction with UI elements in a 3D environment, it is necessary not only to find the elements but also to determine exactly which element the user interacts with, especially when they intersect, using the depth of the element. Taking into account the accuracy of coordinates and perspective ensures correct interaction, and a formalized interface allows you to standardize the processing of input events such as clicks, hovering, and pressing. The formalized definition of the interface for interacting with the UI is determined by formula (2). Identification of a UI element by formula (3), where

$$\begin{aligned} T_{c,\min} &= \left(T_{cx,\min}, T_{cy,\min}\right), \\ T_{c,\max} &= \left(T_{cx,\max}, T_{cy,\max}\right). \end{aligned} \quad (5)$$

When a user interacts with the interface, it is essential to know how the elements are relative to each other, that is, their depth. The depth of a UI element on a texture determines its distance from the user or other elements on the screen. This distance is denoted by the value z, where z=0 means that the element is closest, and larger z values mean that the element is farther away.

A custom Raycaster determines which UI elements should respond to user actions. To accurately determine which UI elements are closest to the user and should be activated, Raycaster calculates the depth (*z-depth*) of the point where the ray intersects the surface. This allows you to prioritize UI elements correctly, providing an intuitive interaction.

When the ray crosses the surface, you need to determine which UI is at this point. Since UI elements can overlap, depth (z-depth) determines priority, allowing you to select the element closest to the user.

Identifying UI elements using *z-depth* consists of three steps.

Step 1. Collecting data on the depth of UI elements. Each UI element on the texture has a *z-depth*, *i*, determining its distance from the camera in 3D space. The smaller the depth, the closer the element is.

Let's say for *n* UI elements on a texture, we have sets of coordinates and depths:

$$\left\{\left(T_{cx}, T_{cy}, z - depth\right)\right\}, i = 1, 2, ..., n . \quad (6)$$

I is an element of the texture, which varies from 1 to *n*.

DOI 10.15588/1607-3274-2025-4-19

Step 2: Determining the active UI element. The task is to determine which UI element on the texture will be active for interaction, considering the *z-depth* of the elements. For each intersection point on the texture T_c , the active UI element is determined by the following formula:

$$active\ UI\ element = \arg\min\left(z-depth, i\right). \quad (7)$$

arg min(*z-depth, i*) is the index of the element for which the depth is *z-depth* and is the minimum among all elements that overlap the intersection point.

Step 3. Check for overlapping UI elements. Since several UI elements on the texture can overlap the same T_c point, selecting the element closer to the user is important. If several elements have the same T_c coordinate, the element is selected using depth minimization.

Each element, *i*, is checked to see whether its area intersects the point T_c, and then its *depth*, *i*, is determined for each element.

Of particular interest is the use of modified shaders to create roads that smoothly follow the contours of the terrain. This approach uses splines for road placement and ensures its smooth overlay on the landscape, creating more realistic and detailed scenes. The cubic polynomial is the basis for building splines, which are widely used in computer graphics and modeling to create smooth curves and contours. Splines defined by cubic polynomials provide a smooth transition between points, making them ideal for modeling surfaces and paths in 3D environments. The basic form of a cubic polynomial used to create splines is as follows:

$$F(x) = ax^3 + bx^2cx + d\ . \quad (8)$$

The coefficients a, b, c, and d determine the shape of the curve.

A cubic spline consists of a set of polynomials (8), where each polynomial corresponds to a separate segment between adjacent points. It is essential to ensure smoothness at the junction points, i.e., that the first and second derivatives coincide. This ensures the smoothness of the curve, which is critical for implementing tasks such as camera trajectories, building roads, or object contours in a game. For example, for a cubic spline defined by a four-point formula, it looks like this:

$$P(x)=(1-x)^3P_0+3(1-x)^2xP_1+3(1-x)x^2P_2+x^3P_3. \quad (9)$$

To ensure smoothness and control the curvature of the spline, you need to calculate the first and second derivatives that describe the slope and curvature of the curve. We use the product and chain rules to calculate the derivative. The first derivative, $P'(t)$, is calculated by the formula (5).

$$P'(x)=-3(1-x)^2xP_0+[3(1-x)^2-6(1-x)x]P1+ \\ +[6(1-x)x-3(1-x)x^2]P_2+3x^2P_3. \quad (10)$$

The second derivative, $P''(t)$, is calculated using the following formula:

$$P''(x)=6(1-x)P_0+[-6(1-x)+6x]P_1+ \\ +[6(1-x)-12x]P_2+6xP_3. \quad (11)$$

Both derivatives play a key role in the formation of splines. The smoothness of a spline is determined by the first derivative, $P'(x)$, which is responsible for the slope (or rate of change) of the spline. For a spline to look smooth, the first derivatives on neighboring segments must be identical. This means that the spline shouldn't have sudden slope changes, ensuring a smooth transition between points.

The first derivatives also reflect the rate of change of the spline's position over time, which can be useful in animation or motion modeling.

The second derivative, $P''(x)$, is responsible for the curvature of the spline. It describes how the slope of the spline changes over time. To ensure a smooth transition, the second derivatives must also coincide at the boundaries of neighboring segments. This ensures that the curvature of the spline does not change suddenly.

The second derivative also helps control the shape of the spline, allowing you to create smooth curves and avoid sharp turns. This is especially important in graphics and modeling, where aesthetics are critical.

In physical simulations, the second derivative can calculate the acceleration of objects moving along the spline, allowing you to model realistic movements.

Thus, the first and second derivatives in splines provide smoothness, curvature control, and aesthetic appeal of curves, making them essential tools in computer graphics, animation, and modeling, especially in landscape construction.

4 EXPERIMENTS

To conduct the experiments, a prototype game system was developed for the post-apocalyptic game Broken World Wandere Last Horizont, where players can explore, interact, and develop in a post-apocalyptic environment.

However, during the development of the menu, a problem arose with dropdown lists. The standard *Dropdown List* used in Unity uses *GrapiRaycaster* by default, which does not allow interactivity using a custom *Raycaster* system. This happens because *GraphicRaycaster* is tied to the main camera and does not consider the interaction through the new *Raycaster*. To solve this problem, we created a custom component called *CustomDropdown* based on *TMP_Dropdown*. It rewrote the interaction in such a way as to disable *GraphicRaycaster* on the *Dropdown List* element and replace it with a custom *MashColliderRaycaster* with the transfer of parameters to the new system. This approach ensures correct interaction with the drop-down list within the created system, allowing you to conveniently use the menu in the game, considering all the effects of shaders and other interactivity on any surface (Fig. 5).

DOI 10.15588/1607-3274-2025-4-19

Figure 5 – The drop-down list

This approach is critical for creating more interactive and realistic interfaces, especially in games with open worlds or complex geometry. For example, in post-apocalyptic or sci-fi game worlds where the player can interact with large three-dimensional maps or objects, UI on a curved surface does not look artificial or disconnected from the environment. Instead, it seamlessly integrates into the world, following the shape of the surface it is placed on, making interaction more natural and attractive. With transparent textures and alpha channels, shaders allow you to layer different textures that can adapt to the surface while maintaining transparency and image structure, allowing you to create more sophisticated gaming interfaces.

It is worth noting the use of modified shaders for the realization of roads and other elements that should naturally follow the shape of the landscape. Using splines to create roads allows them to be placed smoothly and organically in accordance with the contour of the cityscape, ensuring realism and smooth transitions between different segments. To do this, the redefined shader and the code component responsible for width and smoothing dynamically consider and display the spline zones. The modified shader based on the spline ensures the correct repetition of the shape of the road surface by passing the *height* parameter, allowing it to naturally "wrap" the terrain (Fig. 6) while maintaining high-quality textures and details. This is especially important for games where landscape details play a significant role in the overall game experience, such as racing, shooters, or adventure games, or it makes it possible to use a spline to change the shape of the landscape dynamically during the game.

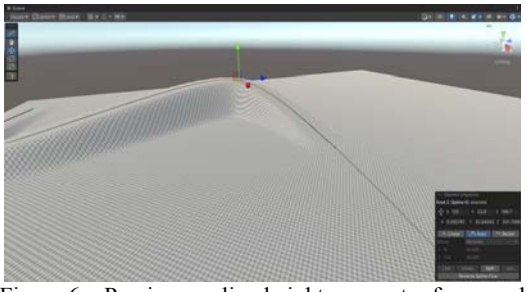

Figure 6 – Passing a spline height parameter for a road

The modified shader overlays the height of the road, taking into account the surface normals and its contours:

$$Shader_{road}=Lerp(Texture_{base}, Texture_{detail}, Alpha\ Chan\text{-}nel). \quad (12)$$

Lerp is an alpha-channel-based texture interpolation to create smooth transitions between road segments.

Using this approach allows you to place roads and other landscape elements by the shape of the terrain, ensuring realism and smooth transitions between road segments, i.e., giving smooth bends at their edges for greater realism (Fig. 7).

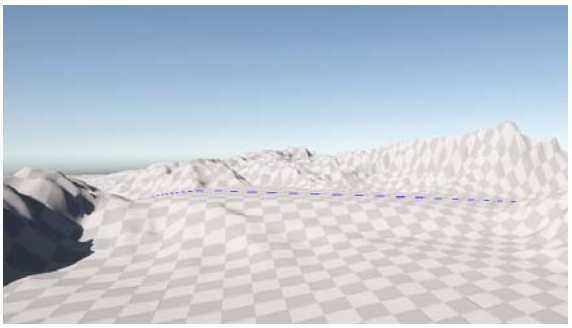

Figure 7 – Road on the landscape

In addition, modified shaders allow you to control texture updates in real-time, which can significantly improve game performance. For example, the interface texture can be updated on demand only when necessary, reducing the load on the system and increasing optimization. This approach allows the implementation of an interface that can adapt to any surface, such as a curved monitor panel in a futuristic vehicle or an interface superimposed on a virtual reality object.

In addition to improving interactivity, modified shaders significantly impact the visual style of the game. They allow you to achieve a new level of depth and detail in the design of game environments, making them more believable and rich. Well-chosen shaders can change the perception of the game, making it unique and inimitable. For example, in post-apocalyptic games like Metro Exodus, the integration of complex shaders can emphasize the abandonment and destruction of the environment by adding detail to broken or warped objects.

The image in Figure 8 shows how cubic splines behave during the experiment.

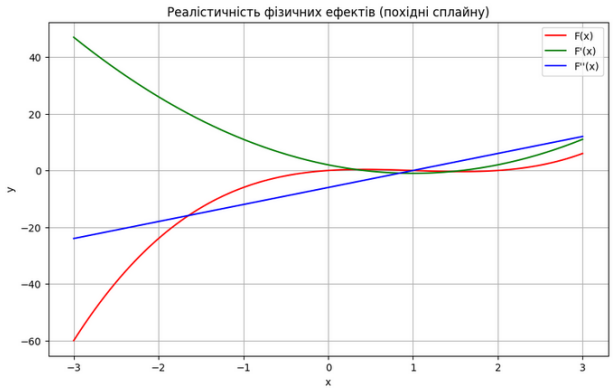

Figure 8 – Realistic physical effects

DOI 10.15588/1607-3274-2025-4-19

The curve shown in red ($F(x)$) shows smoothness, indicating that the spline provides continuous and smooth transitions between specified points. This is critical for modeling physical phenomena where abrupt changes produce unrealistic results.

The green line ($F'(x)$) shows how the slope of the red line changes. When the green line crosses the horizontal axis, the red line becomes flat and changes direction. And all this happens very smoothly, without jerks.

The blue line ($F''(x)$) shows the linear change in curvature, i.e., how the red line's curvature changes. This change occurs evenly in cubic splines, as if in a straight line. This demonstrates their ability to accurately model curvature where precise control of the curve shape is required.

5 RESULTS

The use of standard GraphicRaycaster is unacceptable because it depends on the main camera and cannot receive information directly from the interactive object. Based on the obtained intersection coordinates, the position on the canvas is determined, and a list of controls is generated for further processing.

A custom component based on BaseRaycaster was developed to ensure correct and efficient interaction with the canvas. The proposed component solves this problem by determining the point of intersection of the beam coming out of the camera with the physical surface of the canvas. The implementation includes an algorithm for searching for beam intersections with colliders using a mathematical formula and further processing the found elements, considering their depth to ensure correct interaction.

This approach allows for creating an interface on an arbitrary curved static surface, opening up vast possibilities for use in various games and interactive scenarios.

6 DISCUSSION

The problem of UI interactivity on uneven surfaces is relevant in modern game development, especially in open-world games, where complex geometric shapes are often used.

The presented research focuses on developing and optimizing graphical user interfaces in a 3D gaming environment, particularly on curved surfaces. Custom rails and modified shaders were used to solve the problem of UI interactivity on uneven surfaces. The proposed method, using mathematical formulas to determine the intersection of the ray with a physical surface, provides accurate localization of UI elements. This is consistent with studies [9] emphasizing the importance of accurate geometry and optimized rasterization pipelines. In both the study and article [9], accurate geometry is key to achieving high-quality results. In the study, accurate geometry is used to calculate ray intersections, and in [9], it is used to generate shading intervals.

The study used modified shaders to create roads that smoothly follow the contours of the landscape. This demonstrates the effectiveness of using splines and cubic polynomials. The review notes that shaders are critical in creating realistic visual effects [1, 3].

The study's results confirm the effectiveness of using shaders to create realistic roads that smoothly follow the contours of the landscape. The use of splines and cubic polynomials allows for a high level of detail and naturalness of display.

The use of shaders allows for significant optimization and increased performance, which is important in creating immersive gaming environments.

The proposed approach achieves a high level of detail and smooth transitions, essential for open-world games. This is consistent with research [5], which proposes improvements to ray tracing algorithms to improve image quality and performance.

CONCLUSIONS

In summary, the use of rendered textures and modified shaders allows you to create more realistic game scenes with a high level of immersion and adds visual harmony between the interface and the game world. Spline technology and advanced shaders significantly impact interactivity, performance, and game style, providing smoothness, realism, and visual appeal.

The use of rendered textures and modified shaders significantly improves the integration of the interface into the game environment, providing a more organic interaction between the player and UI elements. This opens new possibilities for creating complex game worlds with interactive UI on non-standard surfaces, such as curved panels, monitors, or books. Another important aspect is the improvement of performance and the ability to adapt textures during the game.

This paper presents a comprehensive approach to designing UI for curved surfaces and using modified shaders to create realistic roads and other landscape elements.

The scientific novelty of the research is developing and successfully applying a ray-cast component using splines to provide efficient and intuitive interaction with the user interface on complex, curved 3D surfaces. This innovative approach solves the problem of the limitations of standard rake tools, which cannot adequately handle interaction with objects on uneven surfaces. Achieve high accuracy in determining the points of intersection of the beam with the surface and their projections by using mathematical spline models for detailed modeling of curves. Ensure reliable identification of active interface elements, considering their depth and possible overlap, which is key to creating an intuitive user interface.

The obtained results have practical significance because the developed and modified shaders can adapt textures during the game, which is an essential aspect for achieving high performance.

Further research is needed to develop methods for automatically adapting UI elements to the geometry of curved surfaces, which will greatly simplify the process of creating UI for complex objects.

The study results can be used to create more realistic and interactive virtual worlds.

DOI 10.15588/1607-3274-2025-4-19

REFERENCES

1. Ilett D. Building Quality Shaders for Unity®. Using Shader Graphs and HLSL Shaders. Berkeley, Apress, 2022, 734 p. ISBN 978-1-4842-8651-7. Mode of access: https://doi.org/10.1007/978-1-4842-8652-4.
2. Xiao D., Liu Z., Wang S. Metamorphic Shader Fusion for Testing Graphics Shader Compilers, *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne, Australia, 14–20 May. 2023. Melbourne, Australia, IEEE, 2023, pp. 2400–2412. Mode of access: https://doi.org/10.1109/ICSE48619.2023.00201.
3. Souza D. A., Mota R. R.Materials for games – An overview on creating materials for games materials for games, *Extended Proceedings of the XX Brazilian Symposium on Games and Digital Entertainment, Porto Alegre, Brazil, 18–21 October. 2021*. Porto Alegre, Brazilian Computer Society, 2021, pp. 136–142. URL: https://doi.org/10.5753/sbgames_estendido.2021.19633.
4. Liang Y. et al. Automatic Mesh and Shader Level of Detail, *EEE Transactions on Visualization and Computer Graphics (TVCG),* 2023, vol. 29, No. 10, pp. 4284–4295. Mode of access: https://doi.org/10.1109/TVCG.2022.3188775.
5. Akenine-Möller T. et al. Improved shader and texture level of detail using ray cones, *Journal of Computer Graphics Techniques (JCGT),* 2021, Vol. 10, No 1, pp. 1–24. ISSN 2331-7418. Mode of access: http://jcgt.org/published/0010/01/01/.
6. Konnurmath G., Chickerur S. GPU Shader Analysis and Power Optimization Model, *Engineering, Technology & Applied Science Research,* 2024, vol. 14, no 1, pp. 12925–12930. Mode of access: https://doi.org/10.48084/etasr.6695.
7. Sasso E., Loiacono D., Lanzi P. L. A Tool for the Procedural Generation of Shaders Using Interactive Evolutionary Algorithms, *2024 IEEE Gaming, Entertainment, and Media Conference (GEM), Turin, Italy, 5–7 June. 2024. Turin*, Italy, 2024, pp. 1–4. Mode of access: https://doi.org/10.1109/GEM61861.2024.10585418.
8. Nishidate Y., Fujishiro I. Efficient Particle-Based Fluid Surface Reconstruction Using Mesh Shaders and Bidirectional Two-Level Grids, *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2024, Vol. 7, No 1, pp. 1–14. Mode of access: https://doi.org/10.1145/3651285.
9. Tricard T. Interval Shading: using Mesh Shaders to generate shading intervals for volume rendering, *Proceedings of the ACM on Computer Graphics and Interactive Techniques,* 2024, Vol. 7, No 3, pp. 1–11. Mode of access: https://doi.org/10.1145/3675380.
10. Huo Y. et al. ed.: Nandigjav M., Mitra N. J., Hertzmann A. ShaderTransformer: Predicting Shader Quality via One-shot Embedding for Fast Simplification, *SIGGRAPH '22: ACM SIGGRAPH 2022 Conference Proceedings, Vancouver BC, Canada, 7–11 July 2022*. New York, NY, United States, 2022, pp. 1–9. ISBN 978-1-4503-9337-9. Mode of access: https://doi.org/10.1145/3528233.3530722.
11. Wiryadi F. I., Kosala R. R. Particle rendering using geometry shader, *1st International Conference on Game, Game Art, and Gamification (ICGGAG), Jakarta, Indonesia, 19–21 December 2016*. Jakarta, Institute of Electrical and Electronics Engineers (IEEE), pp. 1–9. Mode of access: https://doi.org/10.1109/ICGGAG.2016.8052661.
12. Bissell B. et al. A cultural heritage game for entertainment, *Archiving Conference 2021, Granada, Spain, 1–6 July 2021*. Kilworth Lane, Springfield, VA 22151 USA, Society for Imaging Science and Technology, 2021, pp. 1–6. Mode of access: https://doi.org/10.2352/issn.2168-3204.2021.1.0.2.
13. Zhao L. et al. Graph Mining and Machine Learning for Shader Codes Analysis to Accelerate GPU Tuning, *Complex Networks and Their Applications XI, 4 January 2023*. Cham, Springer, 2023, pp. 426–439. Mode of access: https://doi.org/10.1007/978-3-031-21127-0_35.

Received 20.05.2025.
Accepted 18.09.2025.

УДК 004.925.8 + 519.6

МОДИФІКАЦІЯ ШЕЙДЕРІВ І РЕНДЕР-ТЕКСТУР НА КРИВОЛІНІЙНИХ ПОВЕРХНЯХ

Сугоняк І. І. – канд. техн. наук, доцент, доцент кафедри комп'ютерних наук, Державний університет «Житомирська політехніка», м. Житомир, Україна.

Марчук Г. В. – старший викладач кафедри комп'ютерних наук, Державний університет «Житомирська політехніка», м. Житомир, Україна.

Олексюк О. С. – здобувач кафедри комп'ютерних наук, Державний університет «Житомирська політехніка», м. Житомир, Україна.

АНОТАЦІЯ

Актуальність. Криволінійні поверхні є складними для відображення на плоскому екрані. Розробка інтерфейсу для таких поверхонь є актуальною задачею, яка потребує вирішення багатьох проблем. У даній роботі представлено підхід до розробки UI для криволінійних поверхонь, а також модифікація шейдерів для створення реалістичних елементів ландшафту. Об'єктом дослідження є розробка інтерфейсної системи на основі кастомного рейкаста для забезпечення інтерактивності та занурення у ігровий світ.

Мета роботи. Основна мета даної роботи полягає у створенні, вдосконаленні та адаптації шейдерів на криволінійних поверхнях для досягнення більш ефективного рендерингу, зберігаючи при цьому високу якість візуалізації.

Метод. Розробка UI для криволінійних поверхонь потребує врахування особливостей геометрії. Для вирішення цієї проблеми було розроблено кастомний компонент на базі BaseRaycaster, який дозволяє визначати перетин променя від камери з фізичною поверхнею. З метою забезпечення коректної та ефективної взаємодії з полотном, розроблено кастомний компонент на основі BaseRaycaster. Запропонований компонент вирішує проблему шляхом визначення точки перетину променя, що виходить з камери, з фізичною поверхнею полотна. Реалізація включає в себе алгоритм пошуку перетинів променя з колайдерами за допомогою математичної формули та подальшу обробку знайдених елементів з урахуванням їх глибини для забезпечення коректної взаємодії. Завдяки цьому підходу, з'являється можливість створювати інтерфейс на довільній кривій статичній поверхні, що відкриває широкі можливості для використання в різноманітних ігрових та інтерактивних сценаріях.

Результати. Представлений підхід до розробки UI для криволінійних поверхонь та використання модифікованих шейдерів дозволяє створювати більш інтерактивні та реалістичні інтерфейси та ігрові світи.

Висновки. Використання сплайнів та модифікованих шейдерів забезпечує розташування тексту на криволінійних поверхнях та органічне розташування доріг та інших елементів ландшафту відповідно до контурів місцевості. Цей підхід є важливим для створення ігор з відкритим світом або складною геометрією, де UI на криволінійній поверхні виглядає природно та інтегровано в середовище.

КЛЮЧОВІ СЛОВА: розробка ігор, шейдери, сплайн, рендеринг, кастомний райкастер, криволінійні поверхні, користувацький інтерфейс, візуалізація.

DOI 10.15588/1607-3274-2025-4-19

ЛІТЕРАТУРА

1. Ilett D. Building Quality Shaders for Unity®. Using Shader Graphs and HLSL Shaders / D. Ilett. – Berkeley : Apress, 2022. – 734 p. – ISBN 978-1-4842-8651-7. – Mode of access: https://doi.org/10.1007/978-1-4842-8652-4.
2. Xiao D. Metamorphic Shader Fusion for Testing Graphics Shader Compilers / D. Xiao, Z. Liu, S. Wang // IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May. 2023. – Melbourne, Australia : IEEE, 2023. – P. 2400–2412. – Mode of access: https://doi.org/10.1109/ICSE48619.2023.00201.
3. Souza D. A. Materials for games – An overview on creating materials for games materials for games / D. A. Souza, R. R. Mota // Extended Proceedings of the XX Brazilian Symposium on Games and Digital Entertainment, Porto Alegre, Brazil, 18–21 October. 2021. – Porto Alegre : Brazilian Computer Society, 2021. – P. 136–142. – URL: https://doi.org/10.5753/sbgames_estendido.2021.19633.
4. Automatic Mesh and Shader Level of Detail / Y. Liang et al. // EEE Transactions on Visualization and Computer Graphics (TVCG). – 2023. – Vol. 29, No 10. – P. 4284–4295. – Mode of access: https://doi.org/10.1109/TVCG.2022.3188775.
5. Improved shader and texture level of detail using ray cones / T. Akenine-Möller et al. // Journal of Computer Graphics Techniques (JCGT). – 2021. – Vol. 10, No 1. – P. 1–24. – ISSN 2331-7418. – Mode of access: http://jcgt.org/published/0010/01/01/.
6. Konnurmath, G. GPU Shader Analysis and Power Optimization Model / G. Konnurmath, S. Chickerur // Engineering, Technology & Applied Science Research. – 2024. – Vol. 14, No 1. – P. 12925–12930. – Mode of access: https://doi.org/10.48084/etasr.6695.
7. Sasso E. A Tool for the Procedural Generation of Shaders Using Interactive Evolutionary Algorithms / E. Sasso, D. Loiacono, P. L. Lanzi // 2024 IEEE Gaming, Entertainment, and Media Conference (GEM), Turin, Italy, 5–7 June. 2024. – Turin : Italy, 2024. – P. 1–4. – Mode of access: https://doi.org/10.1109/GEM61861.2024.10585418.
8. Nishidate Y. Efficient Particle-Based Fluid Surface Reconstruction Using Mesh Shaders and Bidirectional Two-Level Grids / Y. Nishidate, I. Fujishiro // Proceedings of the ACM on Computer Graphics and Interactive Techniques. – 2024. – Vol. 7, No 1. – P. 1–14. – Mode of access: https://doi.org/10.1145/3651285.
9. Tricard T. Interval Shading: using Mesh Shaders to generate shading intervals for volume rendering / T. Tricard // Proceedings of the ACM on Computer Graphics and Interactive Techniques. – 2024. – Vol. 7, No 3. – P. 1–11. – Mode of access: https://doi.org/10.1145/3675380.
10. ShaderTransformer: Predicting Shader Quality via One-shot Embedding for Fast Simplification / Y. Huo et al. // SIGGRAPH ’22: ACM SIGGRAPH 2022 Conference Proceedings, Vancouver BC, Canada, 7–11 July 2022 / ed.: M. Nandigjav, N. J. Mitra, A. Hertzmann. – New York, NY : United States, 2022. – pp. 1–9. – ISBN 978-1-4503-9337-9. – Mode of access: https://doi.org/10.1145/3528233.3530722.
11. Wiryadi F. I. Particle rendering using geometry shader / F. I. Wiryadi, R. R. Kosala // 1st International Conference on Game, Game Art, and Gamification (ICGGAG), Jakarta, Indonesia, 19–21 December 2016. – Jakarta : Institute of Electrical and Electronics Engineers (IEEE). – P. 1–9. – Mode of access: https://doi.org/10.1109/ICGGAG.2016.8052661.
12. A cultural heritage game for entertainment / B. Bissell et al. // Archiving Conference 2021, Granada, Spain, 1–6 July 2021. – Kilworth Lane, Springfield, VA 22151 USA : Society for Imaging Science and Technology, 2021. – P. 1–6. – Mode of access: https://doi.org/10.2352/issn.2168-3204.2021.1.0.2.
13. Graph Mining and Machine Learning for Shader Codes Analysis to Accelerate GPU Tuning / L. Zhao et al. // Complex Networks and Their Applications XI, 4 January 2023. – Cham : Springer, 2023. – P. 426–439. – Mode of access: https://doi.org/10.1007/978-3-031-21127-0_35.

DOI 10.15588/1607-3274-2025-4-19