UDC: 004.4:004.94

ADDON FOR BLENDER3D. AUTOMATION OF THE OBJECT TEXTURING PROCESS

АДДОН ДЛЯ BLENDER3D. АВТОМАТИЗАЦІЯ ПРОЦЕСУ ТЕКСТУРУВАННЯ ОБ'ЄКТІВ

[1]Liskovetskiy V.V., [2]Boltach S.V., [3]Lanzhenko L.O.
[1]Лісковецький В.В., [2]Болтач С.В., [3]Ланженко Л.О.

[1,2] Odesa National Technological University, Odesa, Ukraine
ORCID:[2] https://orcid.org/0000-0003-1902-2405, [3]https://orcid.org/0009-0004-6210-3520
E-mail: [2]boltach.svetlana@gmail.com, [1]vlad.liskovetskiy@gmail.com

DOI: 10.15673/atbp.v16i3.2916

Abstract. *Today, the use of 3D visualization technologies is one of the key directions in the field of information technologies. The 3D direction has become extremely popular in many fields, such as architecture, interior design, game industry, medicine, automotive and other fields. This method opens up new opportunities for perception and interpretation of information, adding it to a three-dimensional virtual space. The topic of the research work is the creation of a software module (addon) for the Blender3D program, which will provide the user with a library of ready-made materials, divided by categories. This addon has similar functionality to Substance Painter and Quixel Mixer, the main functions of which are to create and adjust materials and textures for imported ready-made 3D models. The addon has its own name "QuickShader" and will provide the user with 8 shaders and 35 PBR materials. Throughout the development of the project, the Waterfall life cycle model was used, stability and a clear structure were put first, which helped to efficiently work through each step of creating an addon for the Blender program. In the conclusion to the scientific work, it can also be noted that the development of a software addon for Blender3D, which includes a library of shader operators and PBR materials, will greatly facilitate the work of users when assigning textures to objects on the scene. A library of shader operators makes it easy to implement and customize shaders for light, glass, and some metals, giving you more control over the visual aspect of the scene. PBR materials provide realism due to the accurate representation of physical properties using Physically Based Rendering technologies. This addon gives users the ability to interact more effectively with procedural materials, providing a quick means to create realistic renders. This module automates the creation of materials by providing a list of prepared shaders and textures that can be applied to scene objects. The user will be provided with a set of ready-made shaders (glass, light material, metals) and PBR materials (wood, fabric, metals, and others), which can be applied with one button on the module panel.*

Анотація. *Сьогодні використання технологій 3D візуалізації є одним із ключових напрямків у сфері інформаційних технологій. Напрям 3D став надзвичайно популярним у багатьох сферах, таких як архітектура, дизайн інтер'єру, ігрова індустрія, медицина, автомобілебудування та інших сферах. Цей метод відкриває нові можливості для сприйняття та інтерпретації інформації, додаючи її до тривимірного віртуального простору. Тема наукової роботи – створення програмного модуля (addon) для програми Blender3D, який надасть користувачеві бібліотеку готових матеріалів, розділених за категоріями. Цей аддон має схожу функціональність із Substance Painter і Quixel Mixer, основними функціями яких є створення та налаштування матеріалів і текстур для імпортованих готових 3D-моделей. Аддон має власну назву «QuickShader» і надасть користувачеві 8 шейдерів і 35 матеріалів PBR. При розробці проекту використовувалася модель життєвого циклу Waterfall, на перше місце ставилися стабільність і чітка структура, що допомогло ефективно пропрацювати кожен крок створення аддона для програми Blender. У висновку до наукової роботи також можна відзначити, що розробка програмного аддона для Blender3D, що включає бібліотеку шейдерних операторів і PBR-матеріали, істотно полегшить роботу користувачів при призначенні текстур об'єктам на сцені. Бібліотека операторів шейдерів спрощує реалізацію та налаштування шейдерів для світла, скла та деяких металів, надаючи вам більше контролю над візуальним аспектом сцени. Матеріали PBR забезпечують реалістичність завдяки точному відображенню фізичних властивостей за допомогою технологій фізичного відтворення. Цей аддон дає користувачам можливість ефективніше взаємодіяти з процедурними матеріалами, надаючи швидкі засоби для створення реалістичних візуалізацій. Цей модуль автоматизує створення*

матеріалів, надаючи список готових шейдерів і текстур, які можна застосувати до об'єктів сцени. Користувачеві буде наданий набір готових шейдерів (скло, легкий матеріал, метали) і матеріалів PBR (дерево, тканина, метал та інші), які можна застосувати однією кнопкою на панелі модуля.
Keywords: Blender3D, addon, scripting, texturing, shaders, PBR materials, visualization.
Ключові слова: Blender3D, аддон, скриптинг, текстурування, шейдери, PBR матеріали, візуалізація.

I. INTRODUCTION

Today, the use of 3D visualization technologies is one of the key directions in the IT field. The 3D direction has become extremely popular in many fields, such as architecture, interior design, game industry, medicine, automotive and other fields. This method opens up new opportunities for perception and interpretation of information, adding it to a three-dimensional virtual space.

Modeling of three-dimensional objects is performed by specialists in the field of visualization using such programs as 3DsMax, Maya, Blender3D, Cinema4D, ZBrush, etc. Each of these programs also provides tools for working with visual object detailing using materials and textures.

The topic of this scientific work will be the creation of a software module (addon) for the Blender3D program, which will provide the user with a library of ready-made materials, divided by categories. This addon will have similar functionality to Substance Painter and Quixel Mixer, the main functions of which are to add and adjust materials and textures for imported ready-made 3D models. The addon will have its own name "QuickShader" and will provide the user with 8 shaders and 35 PBR materials.

II. LITERATURE ANALYSIS

2.1. What is Blender3D?

Blender3D is a free and opensource program for 3D modeling, animation, rendering, and post-processing of finished images and videos. It is available on various platforms, including Windows, macOS, and Linux, and has a huge community of users and extensions (addons) that make most workflows much easier. The program has a simple user interface and an eye-pleasing design (which can be changed according to the styling of other 3D packages if desired).

Blender has a wide range of features that allow users to create visual 3D models, animations, videos and other visual effects. Some of the most important features of Blender3D include:

1) modeling: Blender3D allows you to create complex 3D models with different shapes and details. Users can use various modeling tools such as extrude, scale, move, rotate and others to create complex shapes and structures;

2) animation: There is a necessary toolkit to create complex animations that can include motion, shape changes, rigging, particle effects, and other visual effects. Users can use keyframes to provide a smooth transition between different animation frames;

3) rendering: the program allows users to render their 3D models on two built-in engines in various formats such as video files, images, GIF files, etc.;

4) sculpting: in the Sculpting tab, you can create detailed sculpts of models for their further retopology or 3D printing;

5) scripting: Python programming language is integrated into the 3D editor, which allows users to program their own software extensions (addons) that can expand, facilitate or automate certain processes when working in Blender3D.

Further work will be focused on scripting. Blender3D has a built-in Scripting development environment [1] in which you can program the future addon directly in the 3D editor itself, which will allow you to see the work of the programming addon in real time.

2.2. Addons for Blender3D

Addons [2] (software modules) for Blender3D are extensions that add new features and capabilities to the program. They are made by a community of developers and users, making Blender3D even more flexible and suitable for different tasks. The relevance of addon development is determined by the rapid development of the 3D graphics industry and the ever-growing need for new tools.

Addons make it easier to work in Blender3D by adding convenient tools, automating certain tasks and optimizing the workflow. For example, there are addons for fast modeling, advanced animation management, advanced rendering, and integration with other programs and services. One of the main addon programming languages is Python, which is integrated into the graphic editor and has a separate bpy package that allows interaction with the Blender functionality.

2.3. Functionality of the developed addon called "QuickShader"

The addon, which will be developed in this work, will touch on texturing and shading of objects in Blender3D. The purpose of this module is to automate the process of adding ready-made materials to an object, which will speed up texturing work for experienced Blender users and make it much easier for new users of this 3D editor. The main and quite flexible tool for creating materials for objects is the Shader editor.

Shader editor in Blender is a functionality that allows you to create and edit shaders using a visual interface based on nodes that are connected to each other with the help of separate inputs and outputs and form a procedural generation of graphic information.

Nodes [3] (nodes) are graphical elements or blocks that represent various operations or actions in the graphical editor. They can be used to build large graphs or networks of functions that interact with each other, filling the material with graphic information that is visually applied to objects. In addition to programmed shader operators, PBR material operators will also be created with the help of nodes.

2.4. About shaders and PBR-materials

A shader [4] is a mathematical formula that takes color, position, and various values as input and output to a shader that can interact with light. They are used to create realistic effects of light, shadows, textures and colors. Shaders in Blender3D are also a program code that determines the appearance and properties of the surface of a 3D object , but for ease of use and work with shaders, they are implemented by the system of nodes described above in the Shader Editor . For example, there are notes :

1) Principled BSDF - includes parameters such as basic color, metalness, roughness of the surface , transparency, surface geometry, light emission and others. The working principle of this node is based on physically correct principles, which makes it an indispensable tool for creating realistic materials ;
2) Glass BSDF – allows you to create materials that reproduce the realistic properties of glass;
3) Glossy – used to model materials with a high level of gloss or light reflection, such as lacquered surfaces or metal materials;
4) Emmision – used to give the material the properties of independent light emission.

PBR materials [5] (Physically Based Rendering materials) are an approach to modeling materials in graphic rendering that seeks to reproduce the physical properties of materials in the real world. This means the use of physically based parameters, such as metalness of the material (metallic), surface hardness (roughness), obscuration on the surface of the material (ambient occlusion), surface geometry (normal map) and information about the color of the material (base color). The PBR material stores all this information with the help of textures in the format of png/jpg images. The textures used in this addon were taken from the PolyHaven and Poliigon.com web resources [11–12].

2.5. Analogues with similar functionality

Probably the main similar software that repeats the functionality of the addon and works with visual information on 3D objects is Substance Painter.

Substance Painter is a highly accurate texture mapping tool used to create photo-realistic materials for 3D models. An intuitive interface, integration with other programs and access to a large library of materials make Substance Painter a powerful tool for creative projects in the field of 3D graphics. The difference between Substance Painter and Blender3D is that it is not essentially a 3D editor and does not have the ability to create and edit models. Its main functionality consists in exporting ready-made models for their further texturing. The program has a fairly large library of materials with the ability to create your own textures, but it is not a 3D editor. Unlike the module for Blender3d, Substance Painter will require a comprehensive study of the interface and more powerful PC specs for further work.

Another software similar to the addon being developed is a product called Quixel Mixer. It is an application developed by Quixel that allows users to create photo-realistic textures and materials for 3D models. With a nodal GUI, users can combine, mix and edit different materials using high-quality Quixel Megascans assets. Nevertheless, Quixel Mixer requires learning the interface and more powerful computing characteristics of the computer for comfortable operation.

III. OBJECT, SUBJECT, AND METHODS OF RESEARCH

The development and coding of addons for Blender can be done directly in the program itself using the Python language, in order to see further changes in the interface of the 3D editor itself. Actually, addon will use two main libraries, named bpy (library for interaction with Blender) and os (library for interaction with the operating system).

bpy [6] – the Blender Python API library. It provides access to Blender functions and objects in an external Python script. With bpy, you can manipulate scene objects, perform rendering operations, and interact with various aspects of Blender. This is the main library when creating any addon for Blender.

os – library for working with the operating system. It provides functionality for interacting with the file system, calling commands in the system, and obtaining information about the environment. The functionality of this library will be needed for the addon to build paths to textures in already generated and connected ImageTexture nodes.

Almost every addon for Blender describes 'bl_info' at the beginning of the code, which is required for the normal functioning of the addon. 'bl_info' [7] is a dictionary that defines addon metadata for Blender. This is an important part of the addon that provides information to the 3d editor about the addon and defines its properties. This is important for Blender, as the program uses this information to correctly register and interact with the addon.

The general structure of the addon interface is created using classes and class types, which is also helped by the bpy library. All panels [8] and operators [9] created in the addon are also described using classes. This addon with a set of materials should not take up a lot of screen space (it is desirable to have its own tab) and will be placed in the quick access panel "Toolshelf" [10]. For the addon interface, the organization of panels and tabs in the side toolbar is key . Using panels to group related functions promotes logic and ease of use. The placement of operators in panels simplifies interaction with the addon, and the built-in capabilities of creating an interface design facilitate its creation for user perception.

Usually, in order to create shaders or apply textures, the user must work with a system of nodes and connect them to each other in the correct inputs and outputs, adjust their parameters. In the operators that are programmed in the addon, the same processes of creating, connecting and configuring nodes are performed, only using Python. Shaders and PBR materials will be created using nodes that are initialized and configured using a script. Each shader has its own unique settings and properties, so each of them is programmed separately. As for PBR materials, they are all similar in their creation structure and represent a set of images (Base color, Ambient occlusion, Normal map, Roughness). Therefore, for material PBR operators , a parent class will be created that will deploy the main node structure, and in the child class

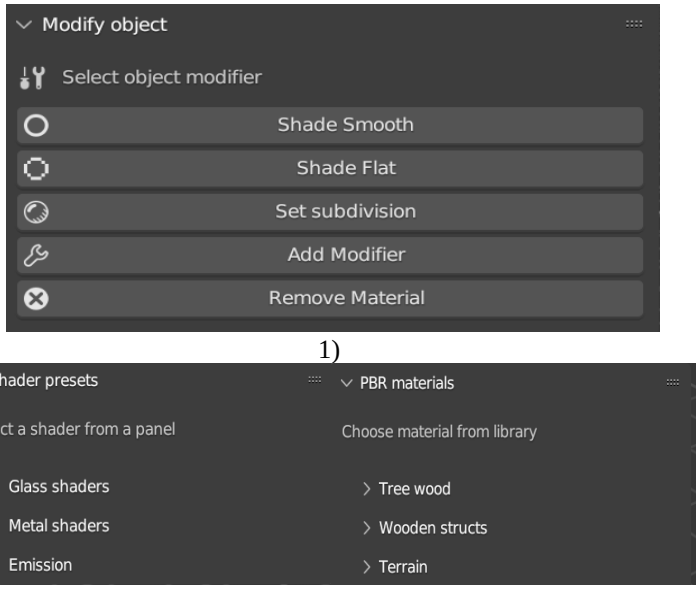
of the material PBR operator , its corresponding textures will be loaded.

IV. RESULTS

4.1 Interface of the finished addon

The result is a complete software extension for Blender3D version 3.4 . After installing this addon, a tab with the same name “QuickShader” appears in the Toolshelf quick access panel . This tab has three main panels, each of which provides the user with a certain functionality :

1) Modify object panel: interaction with objects, provision of certain modifications;
2) Shader preset panel: contains category lists with operators that will apply shaders to the active object;
3) PBR materials panel: contains category lists with operators that will apply PBR textures to the active object;



1)

2) 3)

1 – Modify obect; 2 – Shader presets; 3 – PBR materials

Fig. 1 – Three general panels of an addon

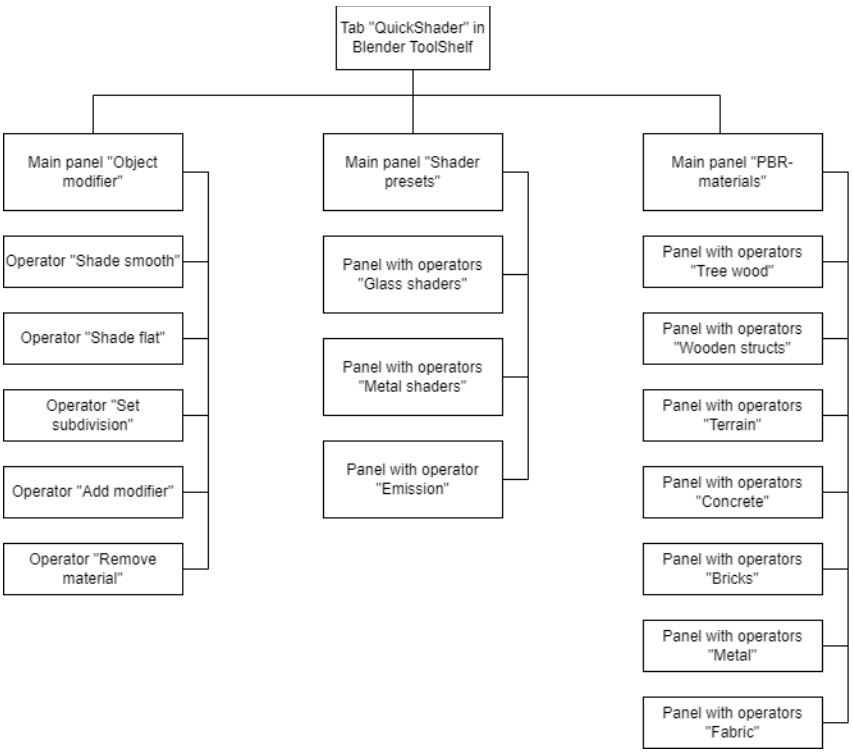


Fig. 2 – Interface structural diagram

4.2 Structural diagram

The flowchart details the structure of the addon interface and the functionality provided by each of the three main panels. The "Modify object" panel will provide the user with operators (buttons) for modifying and configuring the active object . In the “Shader presets” panel , there will be three main lists with operators for glass shaders, metal

shaders, as well as a lighting shader, which can be customized with its own color. The total number of shaders is 3 types of glass, 4 types of metals and a lighting shader. In the panel "PBR materials" there will be 7 categories of materials, where each has 5 operators with different types of textures. The total number of PBR materials is 35 types of textures.

4.3 Code listing that describes the basic addon functionality

Class that creates “Modify object” main panel:

```
class ModifierPanel(bpy.types.Panel):
    bl_label = "Modify object"
    bl_idname = "PT_ ModifierPanel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = 'QuickShader'
    def draw(self, context):
        layout = self.layout
        layout.scale_y = 1.2
        row = layout.row()
        row.label(text = "Select object modifier ", icon = 'TOOL_SETTINGS')
        col = layout.column()
        col.operator("object.shade_smooth", icon = 'ANTIALIASED')
        col.operator("object.shade_flat", icon = 'ALIASED')
        col.operator("object.subdivision_set", icon = 'SHADING_RENDERED', text = "Set subdivision",)
        col.operator("object.modifier_add", icon = 'MODIFIER_DATA')
col.operator('object.remove_material', icon = 'CANCEL')
```

Class that creates “Shader presets” main panel:

```
class ShaderPanel(bpy.types.Panel):
    bl_label = "Shader presets"
    bl_idname = "SHADER_PT_MAINPANEL"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = 'QuickShader'
    bl_options = {'DEFAULT_CLOSED'}
    def draw(self, context):
        layout = self.layout
        layout.scale_y = 1.2
        row = layout.row()
        col = layout.column()
row.label(text = "Select a shader from a panel")
```

Example of class that creates one of the “Shader presets” sub-panels called "Glass shaders":

```
class GlassPanel(bpy.types.Panel):
    bl_label = "Glass shaders"
    bl_idname = "SHADER_PT_GLASSPANEL"
    bl_parent_id = "SHADER_PT_MAINPANEL"
    bl_space_type = 'VIEW_3D'
      bl_region_type = 'UI'
      bl_category = 'QuickShader'
      bl_options = {'DEFAULT_CLOSED'}
      def draw(self, context):
          layout = self.layout
          layout.scale_y = 1.2
          row = layout.row()
          col = layout.column()
          col.operator('shader.glass_operator')
          col.operator('shader.tr_glass_operator')
          col.operator('shader.mc_glass_operator')
```

Example of an operator that creates a glass shader after execution and is located in the “Glass shaders” sub-panel:

```
class SHADER_GLASS(bpy.types.Operator):
    bl_label = "Glass"
    bl_idname = 'shader.glass_operator'
    def execute(self, context):
        material_glass = bpy.data.materials.new(name ="Glass")
        material_glass.use_nodes = True
material_glass.node_tree.nodes.remove(material_glass.node_tree.nodes.get('Principled BSDF'))
```

```
        material_output = material_glass.node_tree.nodes.get('Material Output')
        material_output.location = (561, 213)
        glass1_node = material_glass.node_tree.nodes.new('ShaderNodeBsdfGlass')
        glass1_node.location = (187, 275)
        glass1_node.inputs[0].default_value = (1, 1, 1, 1)
        glass1_node.inputs[2].default_value = 1.470
        glass2_node = material_glass.node_tree.nodes.new('ShaderNodeBsdfGlass')
        glass2_node.location = (187, 140)
        glass2_node.inputs[0].default_value = (0.4, 0.6, 0.6, 1)
        glass2_node.inputs[2].default_value = 1.250
        mix_shader = material_glass.node_tree.nodes.new('ShaderNodeMixShader')
        mix_shader.location = (394, 243)
      material_glass.node_tree.links.new(glass1_node.outputs[0], mix_shader.inputs[1])
      material_glass.node_tree.links.new(glass2_node.outputs[0], mix_shader.inputs[2])
      material_glass.node_tree.links.new(mix_shader.outputs[0], material_output.inputs[0])
      bpy.context.object.active_material = material_glass
        return {'FINISHED'}
```

Class that creates “PBR materials” main panel:

```
class PBRPanel(bpy.types.Panel):
    bl_label = "PBR materials"
    bl_idname = "PT_PBR_MAINPANEL"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = 'QuickShader'
    bl_options = {'DEFAULT_CLOSED'}
    def draw(self, context):
        layout = self.layout
        layout.scale_y = 1.2
        row = layout.row()
        col = layout.column()
        row.label(text = "Choose material from library ")
```

Example of class that creates one of the “PBR materials” sub-panels called " Tree wood ":

```
class pbrWoodPanel(bpy.types.Panel):
            bl_label = "Tree wood"
            bl_idname = "PT_PBR_WOODPANEL"
            bl_parent_id = "PT_PBR_MAINPANEL"
            bl_space_type = 'VIEW_3D'
            bl_region_type = 'UI'
            bl_category = 'QuickShader'
            bl_options = {'DEFAULT_CLOSED'}
            def draw(self, context):
                layout = self.layout
                layout.scale_y = 1.2
                row = layout.row()
                col = layout.column()
                col.operator('pbr.wl_wood_operator')
                col.operator('pbr.oak_wood_operator')
                col.operator('pbr.moss_wood_operator')
                col.operator('pbr.mp_wood_operator')
                col.operator('pbr.sk_wood_operator')
```

Parent class for PBR material operators that creates basic node structure:

```
class PBRMaterialOperator(bpy.types.Operator):
    bl_label = "PBR Material Operator"
    bl_idname = 'pbr.material_operator'
    def execute(self, context):
        self.pbr_material = bpy.data.materials.new(name=self.material_name)
        self.pbr_material.use_nodes = True
        material_output = self.pbr_material.node_tree.nodes.get('Material Output')
        principled_bsdf = self.pbr_material.node_tree.nodes.get('Principled BSDF')
        material_output.location = (600, -60)
        mix_node = self.pbr_material.node_tree.nodes.new('ShaderNodeMixRGB')
        mix_node.blend_type = 'MULTIPLY'
```

```
        mix_node.location = (-300, 300)
        ao_node = self.pbr_material.node_tree.nodes.new('ShaderNodeAmbientOcclusion')
        ao_node.samples = 32
        ao_node.location = (-550, 190)
        normal_map = self.pbr_material.node_tree.nodes.new('ShaderNodeNormalMap')
        normal_map.location = (-300, -200)
        mapping = self.pbr_material.node_tree.nodes.new('ShaderNodeMapping')
        mapping.location = (-1600, -20)
        texture_coord = self.pbr_material.node_tree.nodes.new('ShaderNodeTexCoord')
        texture_coord.location = (-1850, -50)
        self.pbr_material.node_tree.links.new(mix_node.outputs[0], principled_bsdf.inputs[0])
        self.pbr_material.node_tree.links.new(ao_node.outputs[1], mix_node.inputs[0])
        self.pbr_material.node_tree.links.new(ao_node.outputs[0], mix_node.inputs[2])
        self.pbr_material.node_tree.links.new(normal_map.outputs[0], principled_bsdf.inputs[22])
        bpy.context.object.active_material = self.pbr_material
        self.principled_bsdf = principled_bsdf
        self.mix_node = mix_node
        self.ao_node = ao_node
        self.normal_map = normal_map
        self.mapping = mapping
        self.texture_coord = texture_coord
        return {'FINISHED'}
    def add_texture_node(self, texture_path):
        texture_node = self.pbr_material.node_tree.nodes.new('ShaderNodeTexImage')
        texture_node.image = bpy.data.images.load(texture_path)
        texture_node.location = self.texture_node_location
        return texture_node
    def load_texture(self, file_name):
        script_path = os.path.abspath(__file__)
        script_directory = os.path.dirname(script_path)
        texture_path = os.path.join(script_directory, self.texture_folder, file_name)
        texture_node = self.add_texture_node(texture_path)
        return texture_node
```

Example of an operator that inherits the parent class “PBRMaterialOperator” and creates a willow wood PBR material after execution, that loads textures from the corresponding folder:

```
class PBR_WILLOWWOOD(PBRMaterialOperator):
    bl_label = "Willow wood"
    bl_idname = 'pbr.wl_wood_operator'
    material_name = "Willow wood"
    texture_folder = 'PBR/Tree wood/Willow wood'
    texture_node_location = (-900, 380)
    def execute(self, context):
        super().execute(context)
        base_color_tex = self.load_texture('bark_willow_diff_1k.png')
        base_color_tex.location = (-900, 380)
        ambient_oc_tex = self.load_texture('bark_willow_ao_1k.png')
        ambient_oc_tex.location = (-1250, 80)
        roughness_tex = self.load_texture('bark_willow_rough_1k.png')
        roughness_tex.location = (-900, -100)
        normal_tex = self.load_texture('bark_willow_nor_gl_1k.png')
        normal_tex.location = (-600, -250)
        ambient_oc_tex.image.colorspace_settings.name = 'Non-Color'
        roughness_tex.image.colorspace_settings.name = 'Non-Color'
        normal_tex.image.colorspace_settings.name = 'Non-Color'
      self.pbr_material.node_tree.links.new(base_color_tex.outputs[0], self.mix_node.inputs[1])
    self.pbr_material.node_tree.links.new(ambient_oc_tex.outputs[0], self.ao_node.inputs[0])
  self.pbr_material.node_tree.links.new(roughness_tex.outputs[0], self.principled_bsdf.inputs[9])
        self.pbr_material.node_tree.links.new(normal_tex.outputs[0], self.normal_map.inputs[1])
        self.pbr_material.node_tree.links.new(self.mapping.outputs[0], base_color_tex.inputs[0])
        self.pbr_material.node_tree.links.new(self.mapping.outputs[0], ambient_oc_tex.inputs[0])
        self.pbr_material.node_tree.links.new(self.mapping.outputs[0], roughness_tex.inputs[0])
```

```
        self.pbr_material.node_tree.links.new(self.mapping.outputs[0], normal_tex.inputs[0])
self.pbr_material.node_tree.links.new(self.texture_coord.outputs[2], self.mapping.inputs[0])
        return {'FINISHED'}
```

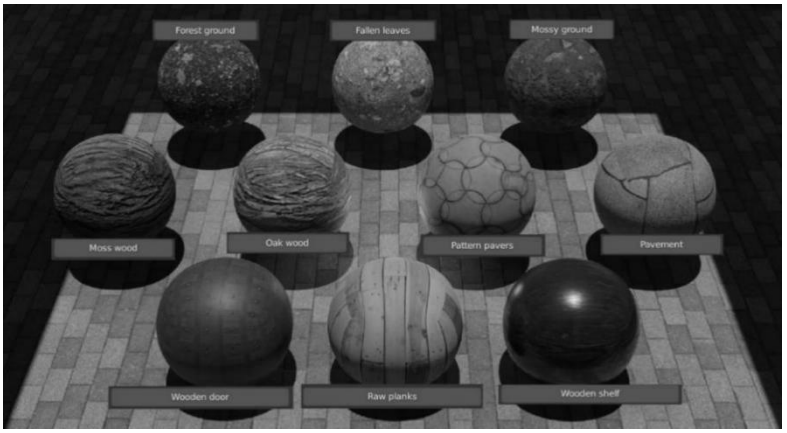
4.3 Addon testing

Software testing is broadly classified as functional testing and non-functional testing. Functional testing will be conducted for this software module, which takes into account the functional requirements of the program. Moving on to software extension tests, you should define the main aspects when testing the application:

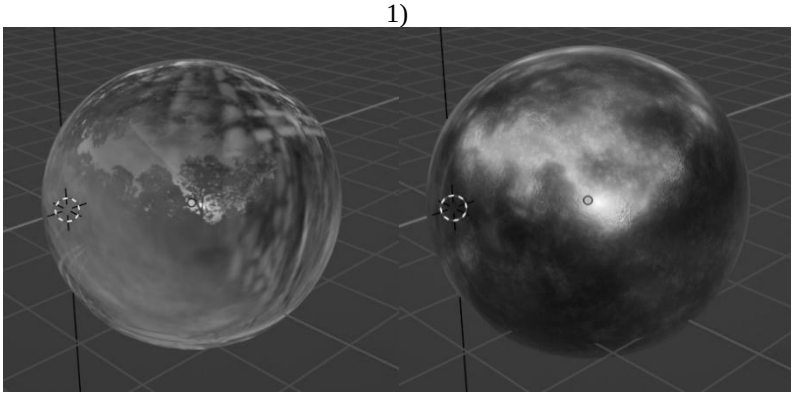
1) whether all panels and lists with material operators are displayed normally;
2) whether all operators work correctly and whether they do not cause critical errors in the Blender program;
3) whether all materials from the addon (shaders and PBR materials) are correctly displayed on viewport and render.

The bpy library for python, which allows the development of addons and interfaces for extension, fully optimizes the creation of additional tabs, panels, operators and lists. Since the addon is fully integrated with the Blender interface, the newly created panels and tabs can be fully stretched or compressed, hidden and expanded. Similarly, they can be reduced to the required size.

During the testing of operators from all main panels and lists, no critical errors were also found. Thanks to the ability to create extensions for Blender directly in the program, you can immediately run the created script and check its functionality. Operators of shaders and PBR-materials are successfully executed when called on the selected object. No errors were detected when rendering the shader or PBR material.



1)



2) 3)

1 – PBR operators; 2 – glass shader; 3 – metal shader

Fig. 3 – The work of shaders and PBR-materials

Table 1- Status of completed tests

No.	Testing	Expected result	Status
1	Functionality of the interface	The interface does not cause errors	Passed
2	Functionality of shader operators	Calling operators does not cause errors	Passed
3	The efficiency of PBR operators	Calling operators does not cause errors	Passed
4	Display of materials	Shaders and PBR materials are rendered correctly	Passed

V. CONCLUSIONS

In conclusion to this work, it can also be noted that the development of a software addon for Blender3D, which includes a library of shader operators and PBR materials, will greatly facilitate the work of users when texturing objects on the scene. In addition, due to its ease of use, the addon can facilitate the work of less experienced users who are just learning to work with visualization and texturing of 3D objects .

The shader operator library makes it easy to implement and customize shaders for light, glass, and some metals, giving you more control over the visual aspect of the scene. PBR materials provide realism due to the accurate representation of physical properties using Physically Based Rendering technologies. This addon gives users the ability to interact more effectively with procedural materials, providing a quick means to create realistic renders.

Список використаних джерел

1. Blender Foundation. Blender API Quickstart. вебсайт. URL: https://docs.blender.org/api/current/info_quickstart.html [дата звернення: 26.10.2023].
2. Blender Foundation. Blender Manual - Add-ons Preferences. вебсайт. URL: https://docs.blender.org/manual/en/latest/editors/preferences/addons.html [дата звернення: 26.10.2023].
3. Blender Foundation. Blender Manual - Introduction to Materials and Shading Nodes. вебсайт. URL: https://docs.blender.org/manual/en/2.79/render/blender_render/materials/nodes/introduction.html [дата звернення: 26.10.2023].
4. Artistic Render. What is the Difference Between Materials, Shaders, and Textures in Blender?) вебсайт. URL: https://artisticrender.com/what-is-the-difference-between-materials-shaders-and-textures-in-blender/ [дата звернення: 31.10.2023].
5. CGHero. PBR Texturing.) вебсайт. URL: https://cghero.com/glossary/pbr-texturing [дата звернення: 31.10.2023].
6. Blender Foundation. Blender API - bpy.ops Module. вебсайт. URL: https://docs.blender.org/api/current/bpy.ops.html [дата звернення: 02.11.2023].
7. Blender Foundation. Blender API - bpy.types.AddonPreferences. вебсайт. URL: https://docs.blender.org/api/current/bpy.types.AddonPreferences.html [дата звернення: 02.11.2023].
8. Blender Foundation. Blender API - bpy.types.Panel. вебсайт. URL: https://docs.blender.org/api/current/bpy.types.Panel.html [дата звернення: 04.11.2023].
9. Blender Foundation. Blender API - bpy.ops Module. вебсайт. URL: https://docs.blender.org/api/current/bpy.ops.html [дата звернення: 04.11.2023].
10. Interplanety. Creating Panels for Placing Blender Add-ons User Interface (UI). вебсайт. URL: https://b3d.interplanety.org/en/creating-panels-for-placing-blender-add-ons-user-interface-ui/ [дата звернення: 24.12.2023].
11. Poly Haven. вебсайт. URL: https://polyhaven.com [дата звернення: 06.11.2023].
12. Poliigon. Free Textures. вебсайт. URL: https://www.poliigon.com/search/free?type=textures [дата звернення: 06.11.2023].

References

[1] Blender Foundation. (2023). Blender API Quickstart. [Online]. Available: https://docs.blender.org/api/current/info_quickstart.html
[2] Blender Foundation. (2023). Blender Manual - Add-ons Preferences. [Online]. Available: https://docs.blender.org/manual/en/latest/editors/preferences/addons.html
[3] Blender Foundation. (2023). Blender Manual - Introduction to Materials and Shading Nodes. [Online]. Available: https://docs.blender.org/manual/en/2.79/render/blender_render/materials/nodes/introduction.html
[4] Artistic Render. (2023). What is the Difference Between Materials, Shaders, and Textures in Blender?) [Online]. Available: https://artisticrender.com/what-is-the-difference-between-materials-shaders-and-textures-in-blender/
[5] CGHero. (2023). PBR Texturing.) [Online]. Available: https://cghero.com/glossary/pbr-texturing
[6] Blender Foundation. (2023). Blender API - bpy.ops Module. [Online]. Available: https://docs.blender.org/api/current/bpy.ops.html
[7] Blender Foundation. (2023). Blender API - bpy.types.AddonPreferences. [Online]. Available: https://docs.blender.org/api/current/bpy.types.AddonPreferences.html
[8] Blender Foundation. (2023). Blender API - bpy.types.Panel. [Online]. Available: https://docs.blender.org/api/current/bpy.types.Panel.html
[9] Blender Foundation. (2023). Blender API - bpy.ops Module. [Online]. Available: https://docs.blender.org/api/current/bpy.ops.html
[10] Interplanety. (2023). Creating Panels for Placing Blender Add-ons User Interface (UI). [Online]. Available: https://b3d.interplanety.org/en/creating-panels-for-placing-blender-add-ons-user-interface-ui/
[11] Poly Haven. (2023). [Online]. Available: https://polyhaven.com
[12] Poliigon. Free Textures. [Online]. Available: https://www.poliigon.com/search/free?type=textures

Отримана в редакції 26.07.2024. Прийнята до друку 17.08.2024. Received 26 July 2023. Approved 17 August 2024. Available in Internet 23 October 2024.