"St. Petersburg State Electrotechnical University

"LETI" them. V.I. Ulyanov (Lenin)"

(St. Petersburg Electrotechnical University "LETI")

Direction of preparation: 09.03.01 – "Informatics and Computational technique"

Profile: "Organization and programming of computing and information systems"

Faculty of Computer Technologies and Informatics

Department of Computer Engineering

Allow for protection:

Department head

Doctor of Technical Sciences, Professor_________________M. S. Kupriyanov

FINAL QUALIFICATION WORK OF THE BACHELOR

Topic: "Software tool for creating 3D animations"

Student		________________ *signature*	O. V. Evdokimov
Supervisor	Doctor of Technical Sciences, Professor (Academic degree, academic title)	________________ *signature*	A. I. Vodyakho
Consultants	k. e. PhD, Associate Professor (Academic degree, academic title)	________________ *signature*	M. A. Kosukhina
	Candidate of Technical Sciences, Associate Professor, p. n. With. (Academic degree, academic title)	________________ *signature*	I. S. Zuev

Saint Petersburg

2023 y.

"St. Petersburg State Electrotechnical University

"LETI" them. V.I. Ulyanov (Lenin)"

(St. Petersburg Electrotechnical University "LETI")

Direction09.03.01 – "InformaticsAnd Vcomputationaltechnique"
Profile"Organization and programming of computing and information systems"
Faculty of Computer Technologies and informatics
Department of Computer Engineering

APPROVE
Head of the Department of VT
Doctor of Technical Sciences, Professor
(M. S. Kupriyanov)
"___"__________ 2023y.

EXERCISE

for graduate work

Student ____Evdokimov Oleg Vitalievich____ Group No. ____**9306**____

1. Theme Software tool for creating 3D animations

(approved by Order No. _____ dated ____________)

Venue of WRC: Saint Petersburg State Electrotechnical University. V.I. Ulyanova (Lenin)

2. Object and subject of research

The process of creating 3D animation videos. A software tool to simplify and speed up the process of creating 3D animations.

3. Target

Development of a software tool for creating 3D animations based on libraries of ready-made animations, such as Mixamo, which allows you to easily and quickly select, customize and combine animations for different characters and scenes.

4. Initial data

The initial data for development are Russian-language and English-language articles and videos on the Internet, documentation for development tools and libraries, developer forums.

5. Technical requirements

5.1. Allow to import 3D character models in different formats (FBX, OBJ, DAE, etc.).

5.2. Allow you to create and edit scenes with characters.

5.3. Allow animations to be imported from a library of pre-made animations for specific characters.

5.4. Allow you to combine different animations in sequence or in parallel using the timeline.

5.5. Availability simple and intuitive graphicinterface.

6. Content

6.1. The study of the theoretical basis for creating 3D animations and existing software for this.

6.2. Development of the architecture and interface of the software tool, taking into account the requirements for functionality and quality.

6.3. Implementation of a software tool using modern technologies and development tools.

7. Additional sections

Development and standardization of software tools

8. results

A working software tool that allows you to load models, add animations for models, set animations on the timeline and play the resulting sequence of animations. Reportingmaterials: explanatory note, abstract, abstract, presentation.

Job issue date	Date of submission of the WRC for defense
"___"_______________2023y.	"___"_______________2023y.

Student ____________________O. V. Evdokimov

Supervisor
Doctor of Technical Sciences,
Professor ____________________A. I. Vodyakho

"St. Petersburg State Electrotechnical University

"LETI" them. V.I. Ulyanov (Lenin)"

(St. Petersburg Electrotechnical University "LETI")

Direction09.03.01 – “Informatics and Computer Engineering"
Profile“Organization and programming of computing and information systems”
Faculty of Computer Technologies and informatics
Department of Computer Engineering

APPROVE
Head of the Department of VT
Doctor of Technical Sciences, Professor
(M. S. Kupriyanov)
“___”__________ 2023y.

CALENDAR PLAN

completion of the final qualifying work

Subject **Software tool for creating 3D animations**

Student Evdokimov Oleg Vitalievich Group No. **9306**

stage num ber	Name of works	Term fulfillment
1	Review of literature on the topic of work	04/05/2023-04/09/2023
2	Architecture design and development	04/12/2023-04/25/2023
3	Implementation of software modules	20.04.2023-05/02/2023
4	Writing an explanatory note	04.05.2023-05/21/2023
5	Preliminary review of the work	05/22/2023-05/26/2023
6	Presentation of work for defense	06/8/2023

Student __________________O. V. Evdokimov
Supervisor
Doctor of Technical Sciences,
Professor __________________A. I. Vodyakho

ABSTRACT

The explanatory note contains:___page,___rice.,___ist.,___adj.

3D animation is the art of creating moving 3D images that requires a lot of data, high performance, and creativity. In order to make 3D animation, you need to be able to work withvarioustools and technologies such as 3D modeling, texturing, skeletal animation, kinematics and more. However, there are ways to simplify and speed up this process with the help of ready-made animation libraries for different characters and scenes. One of these libraries-Mixamo, which offersa bunch offree animations for people, animals, fantasy creatures, etc. Mixamo allows you to quickly and easily select and customize animations for any character through a web interface. But Mixamo is not suitable for creating full-fledged 3D animations, as it does not allowcreatescenes fromseveralcharacters and control the order in which animations play.This problem is intended to be solved by a software tool developed in the course of the final qualification work.

Contents

DEFINITIONS, SYMBOLS AND ABBREVIATIONS

In this explanatory note, the following terms are used with the corresponding definitions:

Animator is a person who creates animations, animated videos.

The graphics API is an application programming interface that allows you to interact with the GPU(GPU).

A shader or shader program is a computer program that runs on a GPU.

An animation keyframe ispoint in time,Vwhichthset the value of some property, for example,orientation or scale positions.

texture - array,bufferallowing you to store any information. Most often usedetsyafor storageimage data.

Mesh(vertex grid)is a mesh of vertices, edges, and faces that defines the shape of a 3D object.

Mesh vertices are points in 3D space that have certainproperties, such as position, color, etc..

A vertex buffer is a data structure that stores information about the vertices of geometric objects in computer graphics.

An index buffer is a data structure that stores vertex indices that determine the order in which they are connected into polygons.

Skeletal mesh(skeletalmesh)is a mesh associated with a skeleton orcarcass, which allows you to animate the mesh by transforming the bones. The skeletal mesh consists of several components:net(mesh), skeleton, skin.

The skeleton is a hierarchical structure of bonesAndjoints, which defines the skeleton of the object.

Skin is the process of attaching mesh vertices(mesh)Tojointsskeleton with certain weights that determine how stronglyevery jointaffects the top.

Skeletal animation is a way of animating 3D models using bone structure, which determines the movement and deformation of the model.

ECS (Entity Component System) is an architectural pattern used in the development of games and graphical applications.

INTRODUCTION

3D animation is a powerful and popular form of visual storytelling that allows you to create realistic and immersive experiences for a variety of audiences and purposes. However, creating 3D animated videos is not an easy task, as it requires a lot of time, skills and resources.

To create a 3D animation video, you need to design and model 3D characters, create and edit scenes with them,set animation keyframes for each object in the scene, up to the joints of the skeletal meshes.There are many software tools for different steps of this process such as Blender[1], Unity[2]etc., but they often have steep learning curves, high cost, or compatibility issues.

The goal is to fill a gap in the literatureand animation toolsby developing a software tool that integrates the various steps involved in creating 3D animation clips in one user-friendly interface.

This tool will allow users to easily and quickly select, customize and combine animations for various characters from librariesready-made animations, such as Mixamo. This will reduce the time, cost, and skill required to create 3D animations.rollersand also increase diversity.

The object of this study is the process of creating animated 3D clips. The subject of this study is a software tool that simplifies and speeds up the process of creating 3D animation clips based on libraries of ready-made animations.

To achieve this goal, it is requireddetermine the functional requirements of a potential user and, in accordance with them, develop the concept of the software tool being developed.Then decide on the technologies and libraries for the implementation of the software tool. Then design and develop software architecture.And finally implement the tool in code.

The first section analyzes and selects existing technologies and libraries that will be used in the development of the software.

In the second sectionthe software architecture is considered in general anddecisions made in the development of various component modules in particular.

The third section deals with the planning of work on the project, standardization andcost estimatesproject.

1 Methods and technologies for developing 3D animation videos

Developing a software tool for creating 3D animations is a task that requires a deep understanding of the methods and technologies used in computer graphics and animation. INgiven sectionwe will look at the maindependencies we need to define before development startsstarting with the choice of programming language and graphics API.

We will consider the principles of representing objects in a scene using the ECS approach. We will also study the issues of importing models,and usedfor this the Assimp library[3].RconsiderquestionscreatingIuser interfaceand relatedlibrariesatDear ImGUI[4].Finallywe will move on to learning the basics of 3D computer animation,discusskeyframes and skeletal animation.

1.1 Choosing a programming language

To develop a software tool for creating 3D animations, it is necessary to choose an appropriate programming language that has the necessary characteristics and capabilities. There are many programming languages that can be used toapplication development for workwith 3D graphics.

Basedsetsfactors, C++ was chosen as the programming language for developing software for creating 3D animations. C++ is one of the most popular and powerful programming languages that is widely used to work with 3D graphics and animation. C++ has the following advantages:

- C++ provides high code performance through compilation to native code and low-level access to computer resources.
- C++ supports object-oriented programming, which allows you to abstract the data and behavior of objects.
- C++ is compatible with most platforms and operating systems. C++ also supports various graphics APIs and libraries that can be used to work with 3D graphics and animation.

Thus, C++ is the optimal choice for developing a software tool for creating 3D animations.

1.2 Selecting a graphicAPI

There are many graphics APIs that differ in functionality, abstraction level, compatibility, and performance. The choice of graphics API depends on the goals and requirements of the software tool.

Some of the most well-known and widely used graphics APIs include the following:[5]:

- Direct3D is part of DirectX, a set of APIs for developing games and multimedia applications on the Windows platform. Direct3D provides low-level access to the GPU. Direct3D has high performance and wide compatibility with different graphics cards and drivers.However, it is only supported on platforms with operating systems frommicrosoft,which does not suit us.
- OpenGL is a cross platform API to work withWith3D graphics. OpenGL is supported by most operating systems. OpenGL also providesrelatively high levelaccessAto the GPU. OpenGL has good performance and flexibility in use.However, it is considered quite outdated today.
- Vulkan is a new cross-platform low-level API forwork with 3D graphics. Vulkan is designed to optimize the use of GPU resources and reduce overhead whendevelopinggraphicapplications. Vulkan has high performance and development potential.

To develop a software tool for creating 3D animations, I chose Vulkan as the most suitable graphics API for the following reasons[6]:

- Vulkanprovides low-level access togpu,Whatallows you to fully control the operation of the GPU and avoid unnecessary costs for synchronization, error checking and data conversion.
- Vulkan supports advanced technologies to create realistic andinteractive3Dcharts, such as ray tracingand so on.
- Vulkan is a cross-platform API and can run onmultitudeoncepersonaldevices, including PCs, mobile phones and

consoles.

As a library for working with mathematics, we will use the well-knownGLM[7],consistent with the shader programming language -GLSL.

1.3 Principles of representing objects in a scene using the ECS approach.

ECS (Entity Component System) is an architectural pattern used in game developmentand graphic applications, which allows you to efficiently organize and process data associated with game objects. ECS is based on three mainobjects:Entity(entity), Component(component)and System(System) [8].

Entity is some entity that exists instage,e.g. character, item. Entity doesn't have any logic,behavioror data, but only represents a unique identifier.

component isan object,kotorystores some information about the Entity, for example,position in space, health, speed or animationAnd. A Component can be added to or removed from an Entity at any time.

system isan object, whichthperformssome actionover Entities and their Component. System usually works with a group of Entities that have a specific set of Components.

The ECS approach has a number of advantages overother approaches, such as:

- ECS allows you to write modular, extensible code, because eachcomponentAndsystemis responsible for his own area of responsibility and does not depend on others.
- ECS improves game performance as data is organized by typecomponents are stored linearlyand are easily cached in memory.

To implement the ECS approach in C++, wewe will useentt library. This is a lightweight and fast library that provides a convenient interface for working withECS [9]. With entt, we can create and delete an Entity, add and remove a Component, access a Component by Entity or by Component type.

1.4 Importing Models

In order to display 3D models in our software, we need to import them from various file formats such as OBJ, FBX, STL and others. To do this, we will use the Assimp library, which allows you to load various 3D formats into a common internal format.

Assimp (Open Asset Import Library) is a library to downloadvarious 3D file formats into a common, internal format[3]. It supports over 40 import formats and a growing number of export formats. Written in C++, it is available underfreeBSD license.

Assimp has a number of advantages that make it suitable for our task:

- It supports many popular and widely used 3D file formats.scenes, such as FBX, OBJ,DAE (COLLADA), glTF, etc. This allowedit to the userupload models and animations from various sources and tools.
- It provides a single andunderstandablea data format for representing objects in a scene.Whatsimplifies the processing andloading model data into our application.
- She offers variousflagspost-processingwhen importing, including frequently needed operations such as calculating normals andtangents. This helps us to optimize and improve the quality of the downloaded data.
- It has an active communityoh Maybehelp with problems or questions.

To import images we will usestb_imagefrom the librarystb[10].

1.5 Creating a graphical user interface

In this section, we'll look at choosing Dear ImGUI as a user interface library in conjunction with GLFW.

Dear ImGUI is a C++ graphical user interface (GUI) library that has minimal dependencies and high performance.[4]. It uses an immediate mode approach, as opposed to the traditional retained mode approach.

In the lazy mode approach, the GUI is pre-built and stored in memory as a tree of widgets.[11]. Every time the GUI state changes, the widget tree needs to be updated and redrawn on the screen. This can lead to redundant calculations, complexity of state managementand problems with state synchronization.

In the immediate mode approach, the GUI is created on the fly every time the draw function is called. There is no need to store the widget tree or update it when the state changes. Instead, the state is stored in user variables and the GUI logic is integrated with the application logic. This simplifies the code and makes it more flexible and modular.

However, the immediate mode approach requires the GUI library to be very fast and efficient so as not to degrade the performance of the application. Dear ImGUI solves this problem by optimizing the process of generating vertex buffers and index buffers for rendering widgets. It also supports various graphics APIs such aspreviously reviewedDirectX, OpenGL, Vulkan and others.

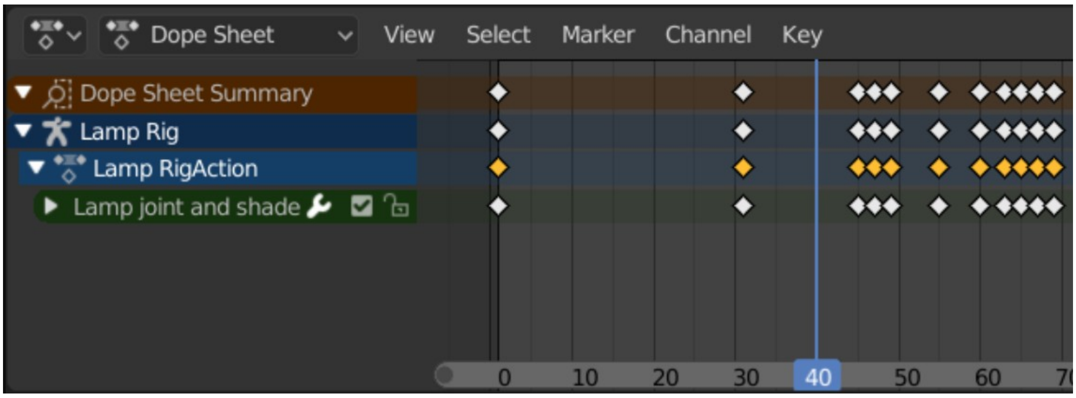
In order to use Dear ImGUI in our tool, we also need a windowing and keyboard/mouse input library. For this, we chose GLFW, which is a lightweight and portable library for creating and managing windows with support for OpenGL and Vulkan.[12].

GLFW integrates well with Dear ImGUI as both libraries have minimal dependencies and a simple API. In order to link them together, we need to use the special module imgui_impl_glfw.cpp. This module contains functions to initialize and update Dear ImGUI with data from GLFW.

1.6 Technological bases and methods of computer animation.

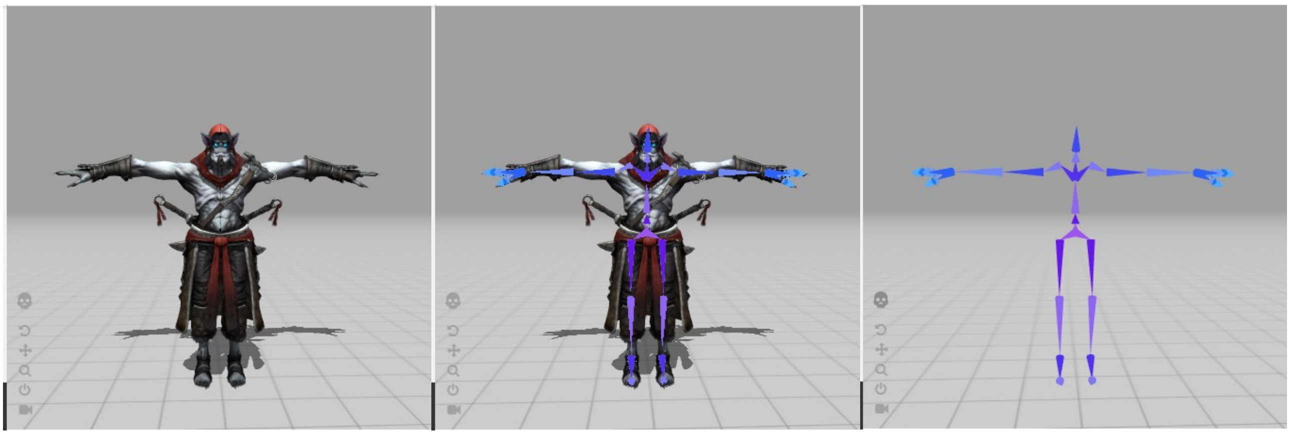
3D computer animation is the process of creatingmovements and shapes of three-dimensional objects using computer programs[13]. To do this, it is necessary to determine the position and orientation of objects at different points in time, as well as the method of interpolation between them. There are several methods of 3D computer animation, but in this paper we will focus on two of them: keyframe animation and skeletal animation.

Keyframe animation is a technique in which the animator sets the position and orientation of an object at specific points in time, called keyframes. The computer program then interpolates the position, orientationand scaleobject at intermediate times,between keyframescreating smooth movement. Various methods can be used for interpolation, such as linear, cubic or Bezier curve,in this paper, only linear and spherical interpolations are used.Keyframe animation allows you to control the movement of an object with high precision, but requires a lot of work from the animator.On the image1.1an example of creating keyframes in the program is givenBlender.



Drawing1.1- An example of keyframes in Blender

Skeletal animation is a method in which the animator sets the movement not of the object itself, but of its skeleton, consisting of bones (bones) and joints (joints). Each bone has its own position and orientation in space, and is also associated with one or morevertices of the object. When changing the position or orientation of a bone, the vertices associated with itaccording to weights, also change their position and orientation. BonesWithOset a hierarchical structure, which, for example, allows you to apply the shoulder transformation also to the fingers. Thus, the movement of the skeleton determines the movement of the object. Skeletal animation allows you to create more realistic and complex movement of objects such as animals or people, but requires you to first create a skeleton and bind vertices to bones.On the image1.2a) a model without a skeleton, b) a skeleton of the model, c) a skeleton superimposed on top of the model.



A) b) V)

Drawing1.2— An example of a model and its skeleton

2 Software development and design

This desktop application was developed using the languageC++using an object-oriented approach, the source code of the program consists of many classes and objects of different levels of abstraction. In this section, we will consider the largest part of objects starting fromlow-levelobjects that arenecessary when usingVulcan API and endinghighest leveluser interface objects.

Specification and source code of the program are given in the appendixA.All user classes are declared inside the namespacedmbrn,which allows them to be distinguished from third parties.

2.1 Wraps around baseextsX objectsVulkanAPI

To get started withVulkanAPIyou need to create and configure many objects that will be used by the rest of the software package, for example, a logical device object for allocating memory forGPU.

In our case, we believe that such objects can exist only in a single copy during the entire program execution time. Accordingly, to define such objects, you can use the singleton template (Singleton). What does it mean that an object constructor is declared privateand only one class (Singletons) has access to it.

However, some objects are dependent on each other, which makes it necessary to regulate the order of initialization and destruction of objects. To do this, all such objects were declared under one structure.Singletons, with the constructor removed, likestatic line.

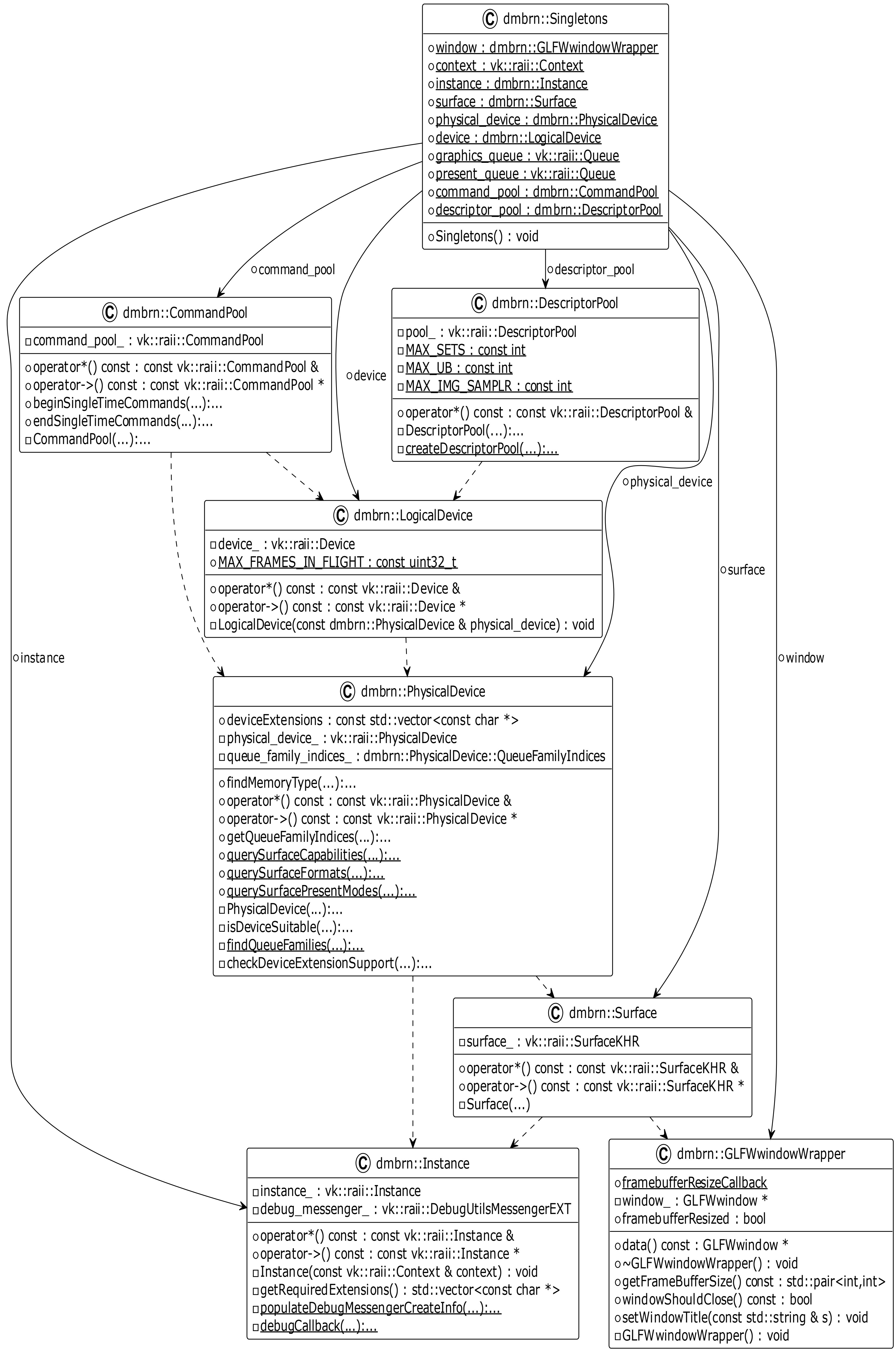
Table2.1given class member variablesSingletonswith a short description of their purpose.Also in the figure2.1, in the form of a class diagram, shows the dependencies between classes.

Table2.1- Member variables of the class Singletons

Name	Type	Purpose
window	dmbrn::GLFWwindow Wrapper	The class provides methods for creating, accessing, manipulating, and destroying a window. It also handles input and window events using callbacks.
context	vk::raii::Context	This is a class that encapsulates the initialization and deinitialization of the Vulkan library
instance	dmbrn::Instance	This is a class that wraps a vk::raii::Instance object that represents an instance of a Vulkan application.
surface	dmbrn::surface	This is a class that wraps a vk::raii::SurfaceKHR object, which is an abstract surface that can present rendered images to the user.
physical_device	dmbrn::PhysicalDevice	This is a class that wraps a vk::raii::PhysicalDevice object that represents a physical device (such as a GPU) that supports Vulkan.
device	dmbrn::LogicalDevice	This is a class that wraps a vk::raii::Device object that represents a logical device (for example, a GPU abstraction) that can be used to interact with Vulkan.

Table continuation2.1

Name	Type	Purpose
graphics_queue	vk::raii::Queue	This is a class that wraps a vk::Queue object, which is an ordered list of commands to be passed to the device for execution. The graphics_queue is obtained from the device using the graphicsFamily index, which indicates that it supports graphics operations.
present_queue	vk::raii::Queue	Obtained from the device using the presentFamily index, which indicates that it supports surface representation of images.
command_pool	dmbrn::command pool	This is a class that wraps a vk::raii::CommandPool object, which is a pool of memory from which command buffers can be allocated.
descriptor_pool	dmbrn::DescriptorPool	This is a class that wraps a vk::raii::DescriptorPool object, which is a pool of memory from whichsetsdescriptors.



Drawing2.1—Loner Class Diagram

2.2 Description of resource classes

In this section, we will consider the structure of classes created to manage resources that are used when rendering 3D models. These are objects such as materials, textures,vertex networksand skeletal networks. These classes encapsulate objectsVulkansuch as buffers and descriptor sets (descriptor set) representing the data forGPU.For easier management, lifetime control and modification of such objects.

2.2.1 diffusematerial and its dependencies

A 3D material is a way of describing the appearance and surface properties of a 3D object. A material determines how an object reflects or transmits light, what color and texture it has, and how it reacts to shadows and highlights.

To set these parameters Also special images are used, called maps (texturesami). A map is a file with information about the color, brightness, or height of each pixel on the surface of an object. Maps can be of different types depending on which material parameter they define.

In this work, we will limit ourselves to only one parameter - color (the main tone of the object's surface) and a diffuse color map (diffuse map) - determines the main color of the object's surface.

To avoid duplication of materials and optimize memory usage. The materials are stored in a hash table, the key to which is an object of the special MaterialRegistryHandle class.

The MaterialRegistryHandle class object contains the raw texture and color data and supports the operators required for the hash table:

- A hash that takes the exclusive or from the hash of the image and a color vector.
- Equality comparison operator.

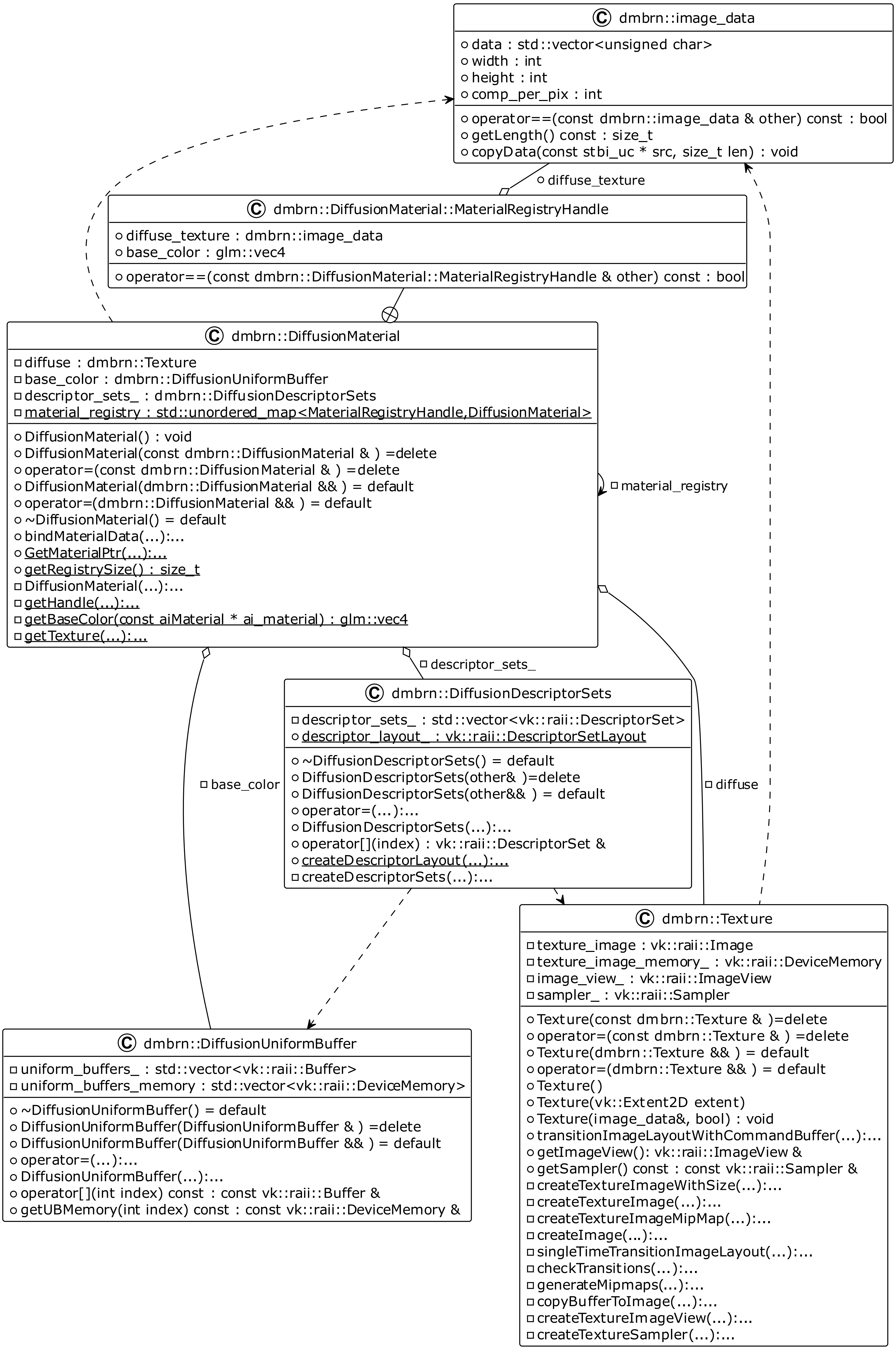
Description of member variables of the DiffusionMaterial classshown in the table2.2.Descriptionmajormember functions of the DiffusionMaterial classshown in the table2.3.On the image2.2a diagram of the considered classes is given.

Table2.2— Member variables of the DiffusionMaterial class

Name	Type	Purpose
diffuse	dmbrn::texture	Object EncapsulationVulkanassociated with the descriptiondiffuselyth texture
base_color	dmbrn::DiffusionUniformBuffer	Object EncapsulationVulkanassociated with the description of the primary color
descriptor_sets_	dmbrn::DiffusionDescriptorSets	Object EncapsulationVulkan related descriptionsets of resources.Resource sets will be discussed later.
material_registry	std::unordered_map<MaterialRegistryHandle, DiffusionMaterial, MaterialRegistryHandle::hash >	Register of materials used in the application.

Table2.3—Member functionsclass DiffusionMaterial

Name	Purpose
bindMaterialData	Pbinds material datafor transmitted frameto the command buffer.
GetMaterialPtr	This function checks if a material with the same MaterialRegistryHandle already exists in the material registry and returns a pointer to that material if found. Otherwise, it creates a new material object and adds it to the registry, then returns a pointer to it.



Drawing2.2— Diagram of classes related to DiffusionMaterial

2.2.2 Static mesh and its dependencies

Vertices can also contain additional information such as color, normal, texture coordinates, and other attributes that affect the appearance and behavior of the mesh. The vertices are connected by edges, which form the faces of the mesh. Faces can be triangular, quadrilateral, or more vertices.

To use mesh data onGPUthey need to be loaded into the memory of the GPU for this, a template class HostLocalBuffer is defined that encapsulates objectsbuffers and memoryVulcan.

To prevent duplication of mesh information. It was decided to split the mesh into two parts: Mesh and MeshRenderData. MeshRenderData contains unique information needed to render the meshand kept in the relevant register. And Mesh contains pointers to MeshRenderData and material, thus it is possible to have many objects with the same mesh, but different materials.

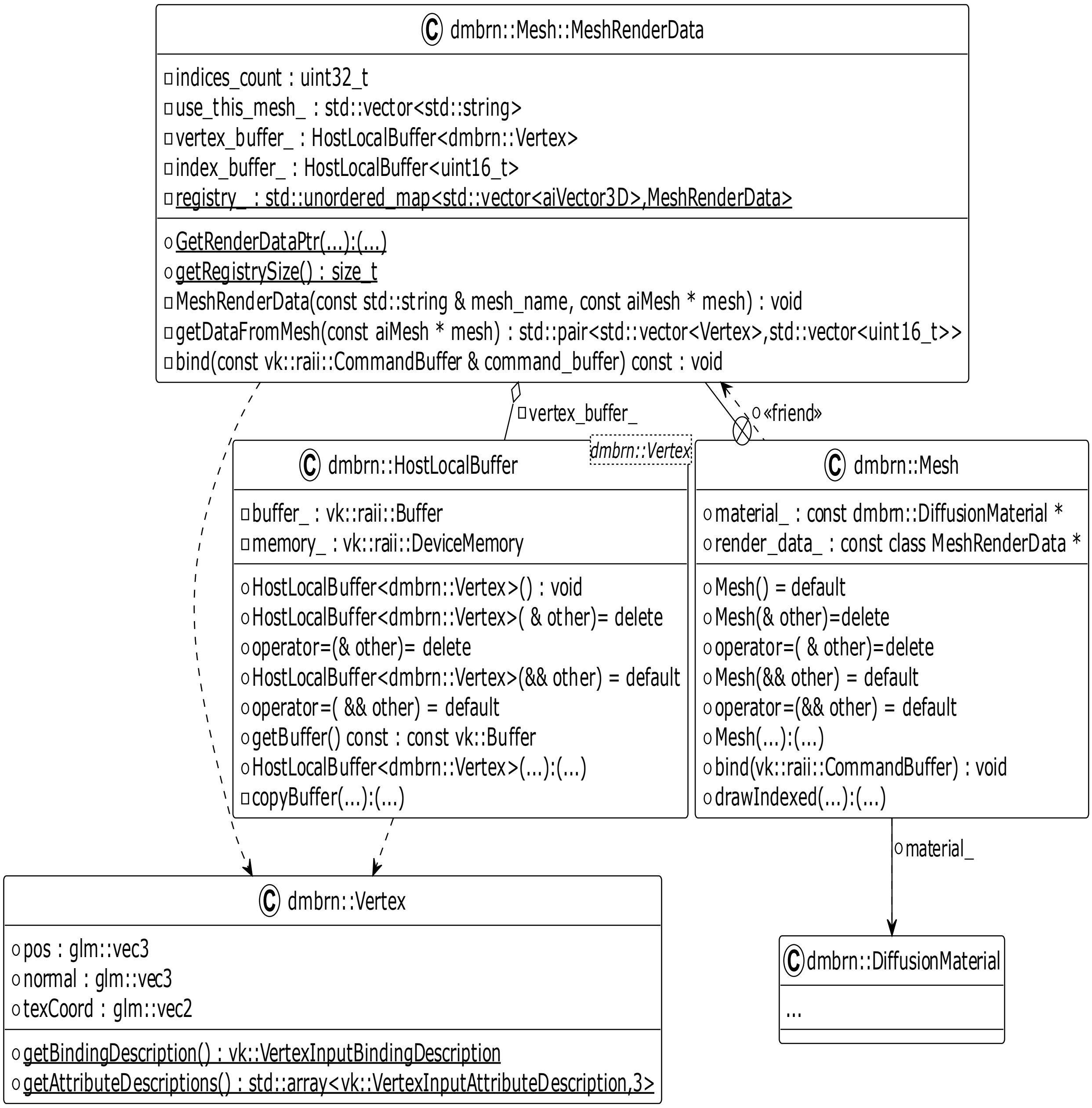
To determine the registry, a hash table is used, the key to which is an array of network nodes. Thus, it can be argued thatmeshes with the same setpositionsvertices (up to order) will refer to the same registry entry.

In our case, the vertices of the mesh contain: position, normal and texture coordinates.

On the image2.3the class diagram is given,where you can see the described dependenciesand class content.

The most important methods are:

- bind-performing mesh data binding to the pipeline;
- drawIndexed-executing the command buffer command to draw the indexed data.



Drawing2.3— Diagram of classes associated with the Mesh class

2.2.3 Skeletal mesh and its dependencies

In this section, we will consider only objects related toskeletalgrid(we interfere)and skin. The skeleton will be discussed in the next sections.

As in the case withOstatic meshes to represent data onGPUHostLocalBuffer is used. With template parameterBoneVertex,which we will consider later.

TOak and sOWith static meshes, a registry was introduced to prevent data duplication and the separation of mesh data into SkeletalMesh and

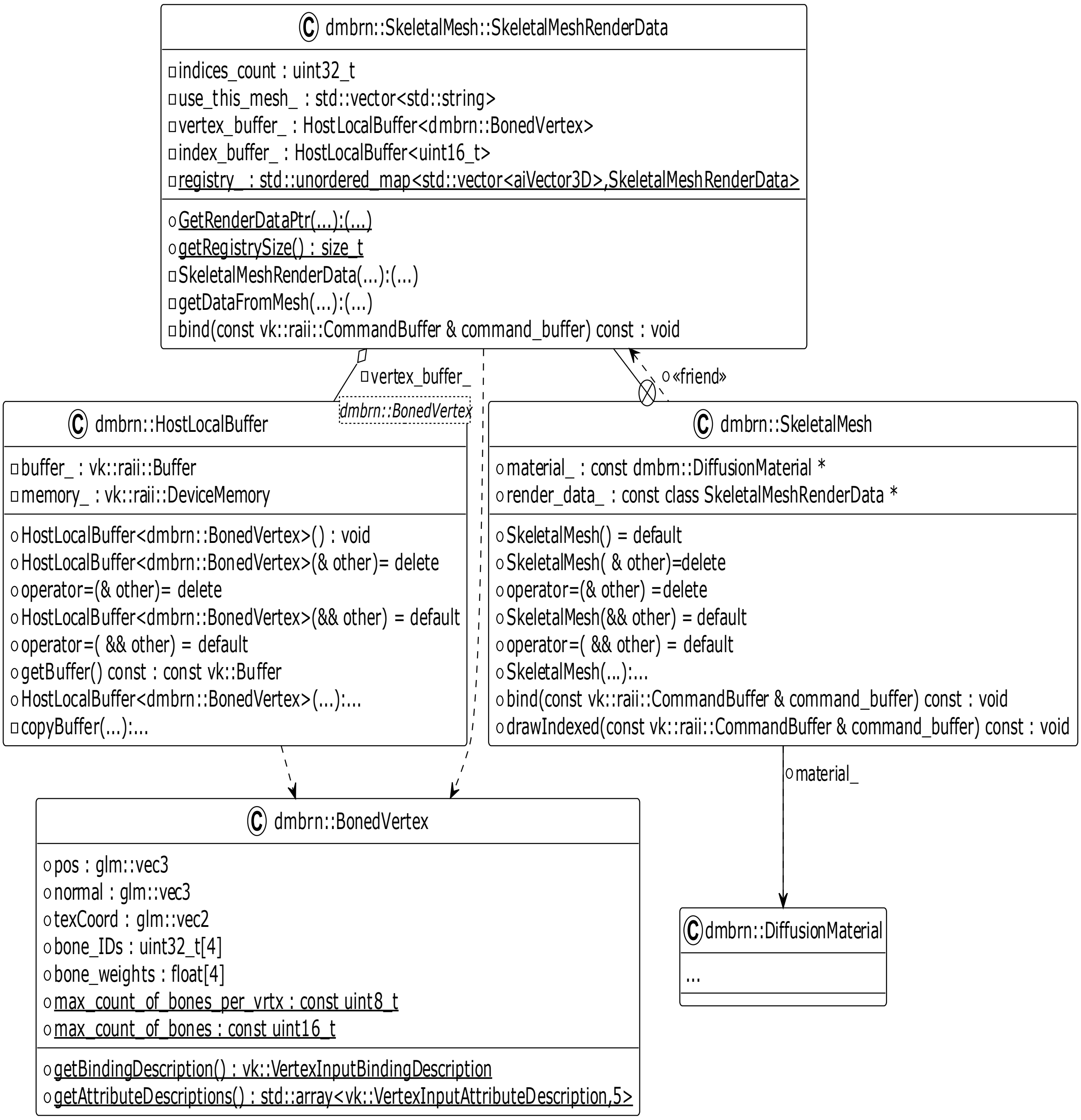
SkeletalMeshRenderData.

The skeletal mesh vertex object contains:

- Position, normal, texture coordinates, as for static.
- Array of indicesVFriendohmarraye,containing the transformations of each bone that has an effect on the given vertex.
- An array of weights definingXthe force with which the bones (their transformations) from the previous array affect the given vertex.

Recentethe two parameters have a constant size given by the constant max_count_of_bones_per_vrtx, currently four.

On the image2.4a class diagram is shown, on which you canAnddethe relationships described.



Drawing2.4— Class diagram associated with the SkeletalMesh class

2.3 Description of rendering subsystem classes

DIn order to use the previously described resources and get them displayed on the screen, it is necessary to describe the way they are displayed, as well as additional data on the position of these objects in space (on the stage) and information about the position and properties of the camera.

The way objects are displayed is set using classes associated with the classshader effect.

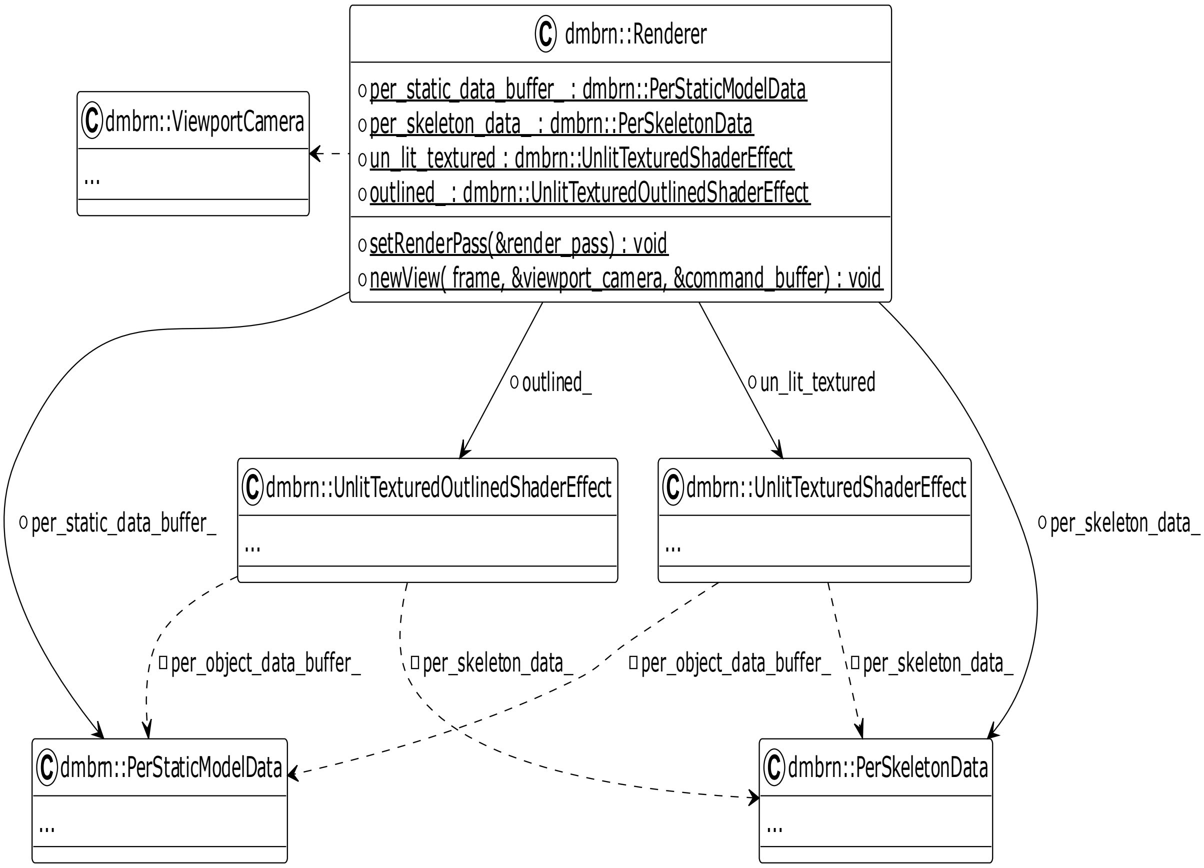
Additional datamiabout the position of objects is:

● For static meshes, the transformation matrix of the mesh model. Such matrices for each mesh should be stored in a separate buffer onGPUcontrolledohmclass PerStaticModelData.

● For skeletal meshes, the transformation matrices for each bone in the skeleton mentioned earlier. Arrays of such matrices are also contained in a separate buffer onGPUcontrolled by the PerSkeletonData class.

The position and properties of the camera are set in the ViewportCamera object created for each viewport of the scene.

The objects described earlier are also singletons, therefore they are combined under one Renderer class,for life time control and convenience.On the image2.5a class diagram describing these relationships is given.



Drawing2.5— Rendering subsystem class diagram

To represent the data of these classes onGPUuniform buffers are used (uniform buffer). To simplify the work, a template class UniformBuffer was created that abstracts the process of creating a buffer and binding memory. The main method of this class is mapMemory, performing buffer memory mapping fromGPUonram.

Excepttransmissionthe data itself onGPUwe needdescribe sets of descriptors (descriptor set) and placement (layout) these resourcesto access them from shaders.

“A handle set is a collection of resources that are bound to a pipeline as a group. Several such sets can be attached to the pipeline at the same time. Each set has a layout that defines the order and types of resources in the set. The location of a set of descriptors is represented by an object, and sets of descriptors are created with the participation of this object.[14]

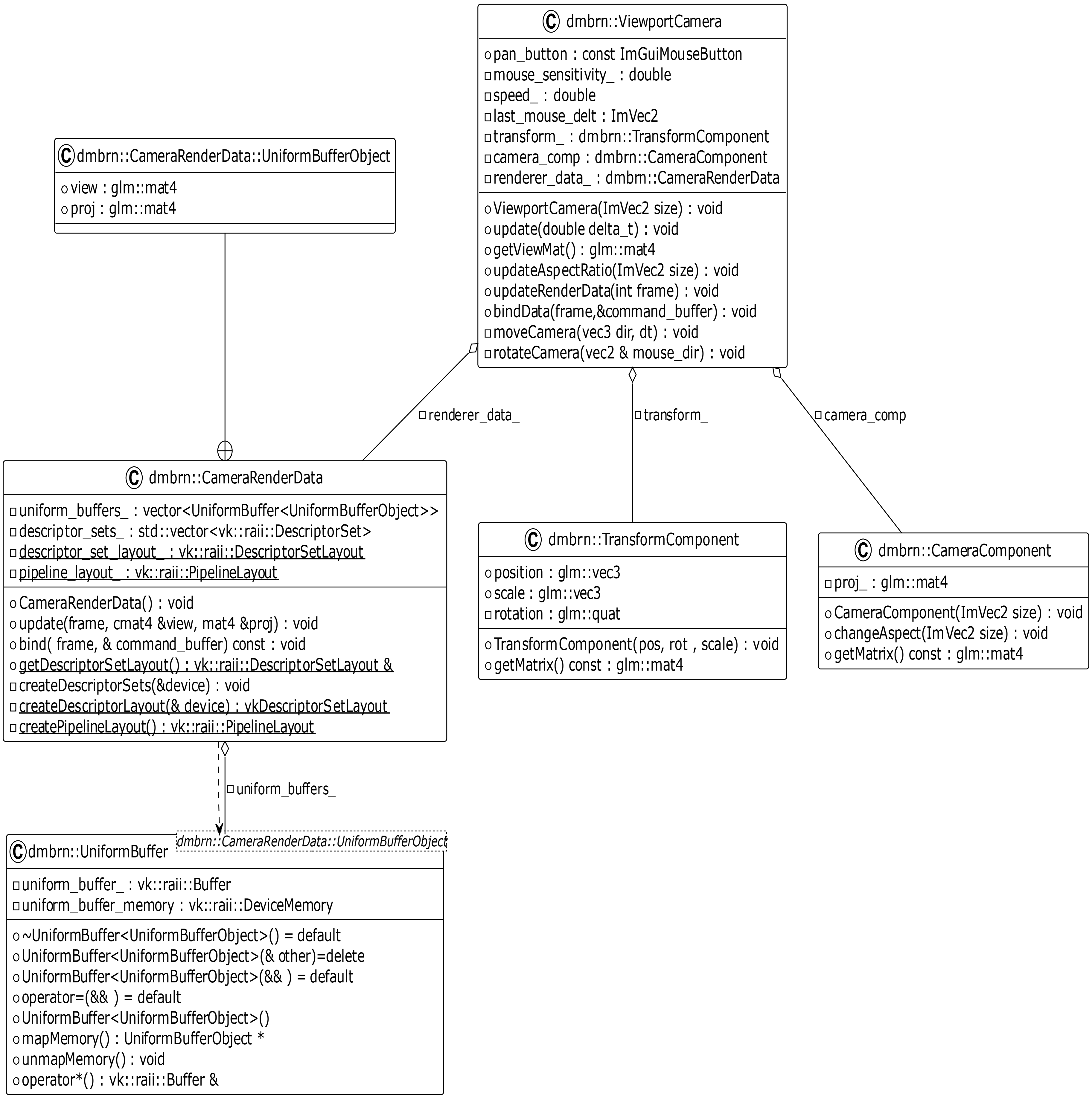
For theseeveryeveryrender related object hasalso objects of classes vkDescriptorSetand vkDescriptorSetLayout.

2.3.1 Classes describing position in space

As already mentioned in our program, there are two classes that describe the direct position of an object on the stage and one that describes the position of the camera on the stage, which affects the position of all objects.

The first two objects have a similar description and purpose, so we will describe them together. These objects are object pools that, when registerObject is requested, allocate space for the corresponding object type in a dynamic buffer onGPU (UniformBufferDynamic)and return the index of the start of the allocated object.In addition, they allow using the map and unMap methods to map the memory of thisGPUmemory buffer to modify values. Also using the functionbindDataForbind valueobject with a given shiftto the graphics pipeline.

The next class we'll look at is the ViewportCamera class. Defines the camera position of the TransformComponent transform_ variable, and the camera properties,at the moment it's only the aspect ratio,defining the camera projection matrix,variable CameraComponent camera_comp.To view camera data management onGPUtemplate class is usedCameraRenderData.On the image2.6givendiagramclasses, on which you can see the described relationships.



Drawing2.6— Diagram of classes describing position in space

2.3.2 Classesdescribingrendering method

Previously mentioned classshader effectis Aabstract base class for dVuh others:

- dmbrn::UnlitTexturedShaderEffect —class responsible for the standard display of the textured model with simple shading.
- dmbrn::UnlitTexturedOutlinedShaderEffect —a class responsible for displaying a textured model with an outline of a given color.

Classshader effectdetermines the presence of twosetsrenders:

- for static meshes - static_render_queue;
- for skeletal meshes -skeletal_render_queue.

Queues themselves are objects of typestd::setthe elements of the set are ordered by a user-defined comparison operator, the task of which is to first sort the objects by the pointer to the mesh, then by the pointer to the material, then by theVyoke into the object's position buffer. Thus, passing linearly through the set, we willmgo first through objects with the same meshes, inside this group with the same materials, and inside this group go through all the positions. Which allows us to save on calling data binding operations to the graphics pipelinefurther.

It also defines a method with two argument overloads for adding objects to the appropriate queue, addToRenderQueue.

Defines a pure virtual draw method that draws static and skinned meshes from the queue.

On the image2.8a class diagram showing the described relationships is given.

Let's take a closer look at the structure of classes inherited from the ShaderEffect class.Both available classes contain one or more objectswith the ending GraphicsPipelineStatics.Such objects containobjectsVulkanrequired to describe rendering such as:

- conveyor placement (pipeline layout) - "... the set of sets that are available to the conveyor is grouped into another object: the location of the conveyor. Conveyors are created with this object in mind."[14]

- graphics pipeline (pipeline)—an objectgraphics pipelinedirectly defining rendering.

Let's look at issues related to the placement of the conveyor. As already mentioned, we have several resources required to draw an object, these are: camera data, material data, shader effect data. Each of these data is grouped into its own set of descriptors with their placements, then the pipeline placement is assembled from them.

On my blog[15] NVIDIArecommends adhering to the principle of arranging resources by the frequency of their binding. So with the increase in the binding index, the frequency also increases. This principle is also justified because the specificationVulkan[16]in paragraph 14.2.1introducesPThe concept of pipeline layout compatibility refers to not bothering previously bound handles when associating a new handle with a compatible pipeline layout.

INin accordance with what was said earlier in this work, it was decided to first place the descriptors in the following order:

1. camera data;
2. shader effect data;
3. material data;
4. static or skeletal model data.

The described relationships are shown in the figure.2.7.Provided descriptors for OutlineGraphicsPipelineStaticsand static model.

TNow let's take a general look at issues related to the pipeline.ConveyorVulkanconsists of many stageswhich can be configured in various ways.The most interesting for us are the stages of vertex andfragmentary(pixel) shaders.To create a pipeline, we need to pass read from a fileSPIR-Vcode, shader source code in languageGLSLattachedB.

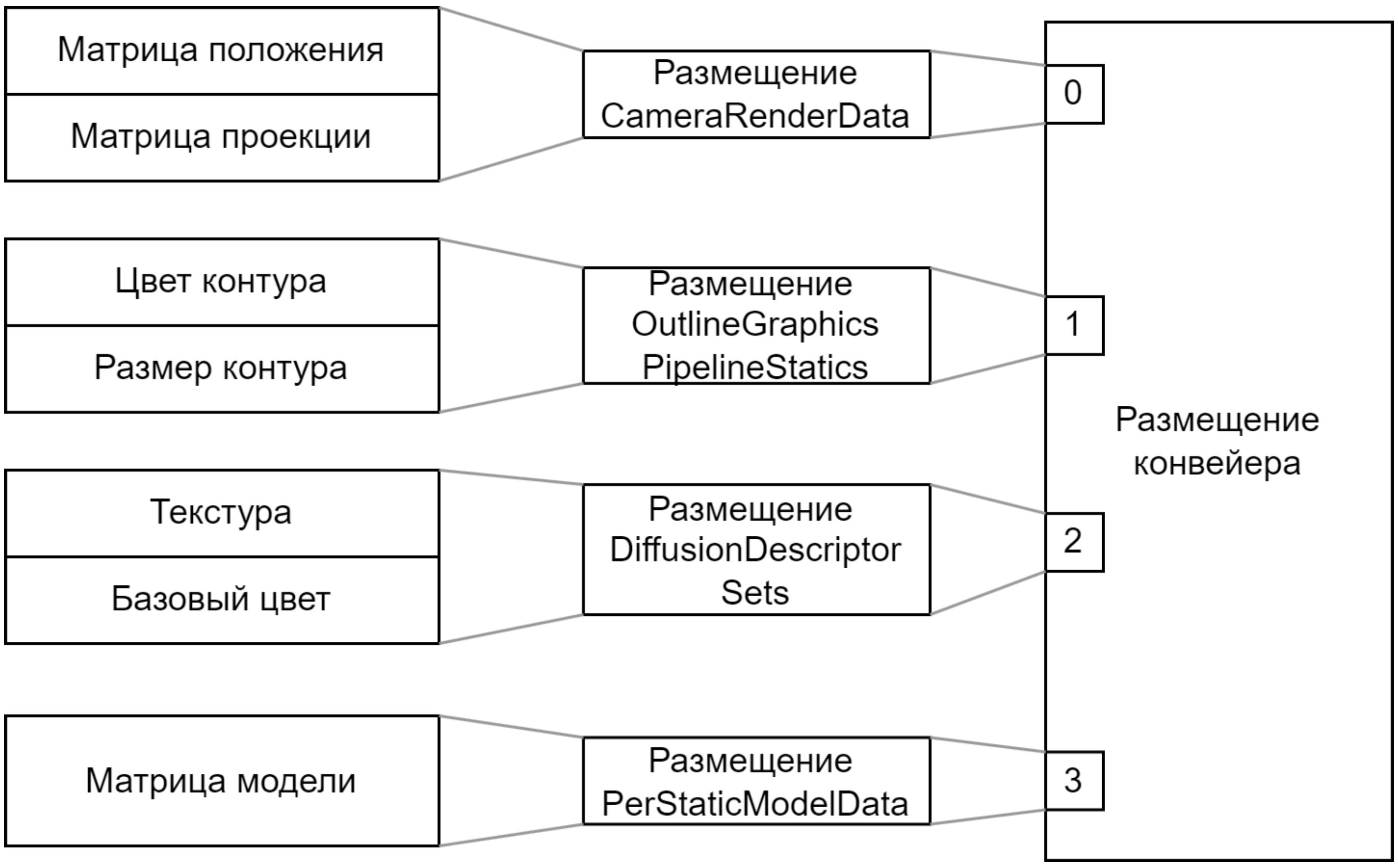
2.4 Subsystem classesECS

As mentioned earlier, scene objects are entities that can have a different composition of components. And the entities themselves are only identifiers. For the convenience of working with the libraryentt,class was writtenentity-wrap aroundtypeentt::entity which, if you look deeper, is an integer std::uint32_t. Accordingly, this object, like the entt::entity object, is not a full owner, i.e., it does not manage the memory and lifetime of the entity.

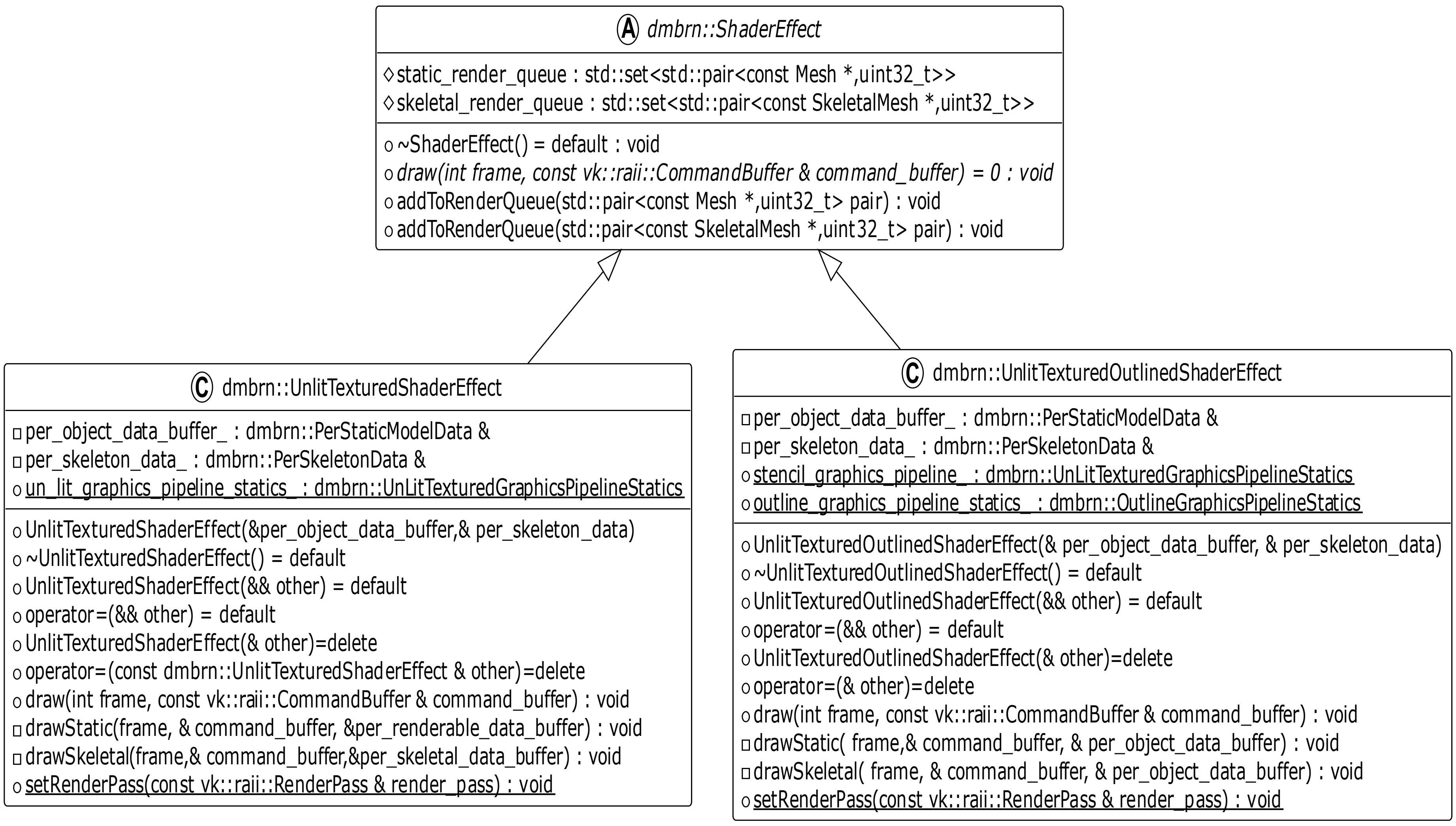
PDuring the development of the program, it was decided to approve the followinguy set of rules:

- an entity (except entt::null) cannot exist without a name (the TagComponent class is responsible for this);
- positions on the stage (the TransformComponent class is responsible for this);
- every entityis in a hierarchical relationship with others (the RelationshipComponent class is responsible for this).
- To comply with this rule, the constructor adds these components when creating the entity.

TagComponentcontains onlystd::stringand all.



Drawing2.7- Conveyor placements



Drawing2.8- Class diagram describing the method of rendering

TransformComponentcontains the position and scale vectors, as well as a quaternion that specifies the orientation of the entity. In addition, two additional attributes-arrays dirty (a sign of pollution) and edited (a sign of change)for each frame. These attributes are needed for the process of hierarchical updating of entity positionsto herhappening every frame.The idea is that when you change the position of some entity in the place of change, you need to call the Enttity::markTransformAsEdited function, which marks the TransformComponent of this entity as changed (edited), then marks all entities located higher in the hierarchy as dirty (dirty) with method Enttity::markTransformAsDirty.

RelationshipComponenta component containing the identifier of the ancestor (parent), its first child (first), the next child of its ancestor (next), the previous child of its ancestor (prev). Thus, it turns out to organize a tree-like relationship between entities.

staticModelComponentthe component containing the Mesh object(talking with which mesh and material to draw this object),pointer to ShaderEffect(talking how to draw a mesh)Anduint32_toffset index inbufferPerStaticModelData discussed earlier.Taalsoa boolean variable need_GPU_state_update indicating whether the position data needs to be updatedthisobject in memorygpu,since updating them every frame can be expensive.

bone componentcomponent, the presence of which says that this entity is a jointsome kind of skeleton. The component contains a boolean variable need_gpu_state_update indicating whether it is required to update data about the position of this joint in GPU memory. Also contains a joint binding matrix - defines the transformation required to transform from mesh space to local space of a given bone. Also containsbone_ind-index of a given joint within one ofPerSkeletonData buffer objects.

SkeletonComponenta component whose presence says that the entity is the root of the skeleton. The component contains the in_GPU_mtxs_offset variable, which is the offset of the objectinsidePerSkeletonData buffer. The bone_enttities array containing the identifiers of the joint entities (having a BoneComponent component) that make up this skeleton.

AnimationComponent— a component whose presence says that the entity and its entire subtree can be animated. Contains a boolean variable sign of recording — is_recording. Also set (set) animation clip elements of the AnimationClip type, which we'll look at later. Also methods for interacting with animation clips updateClipName - for changing the name andinsertto insert new clips.

AnimationClip-a class that describes ready-made animation clips for scene objects, contains the following attributes:

- name -animation name;
- minAndmax—the minimum and maximum value of the keyframe time;
- channels -hash tableWithchannelamianimations of type AnimationChannels for each entity involved in the animation.

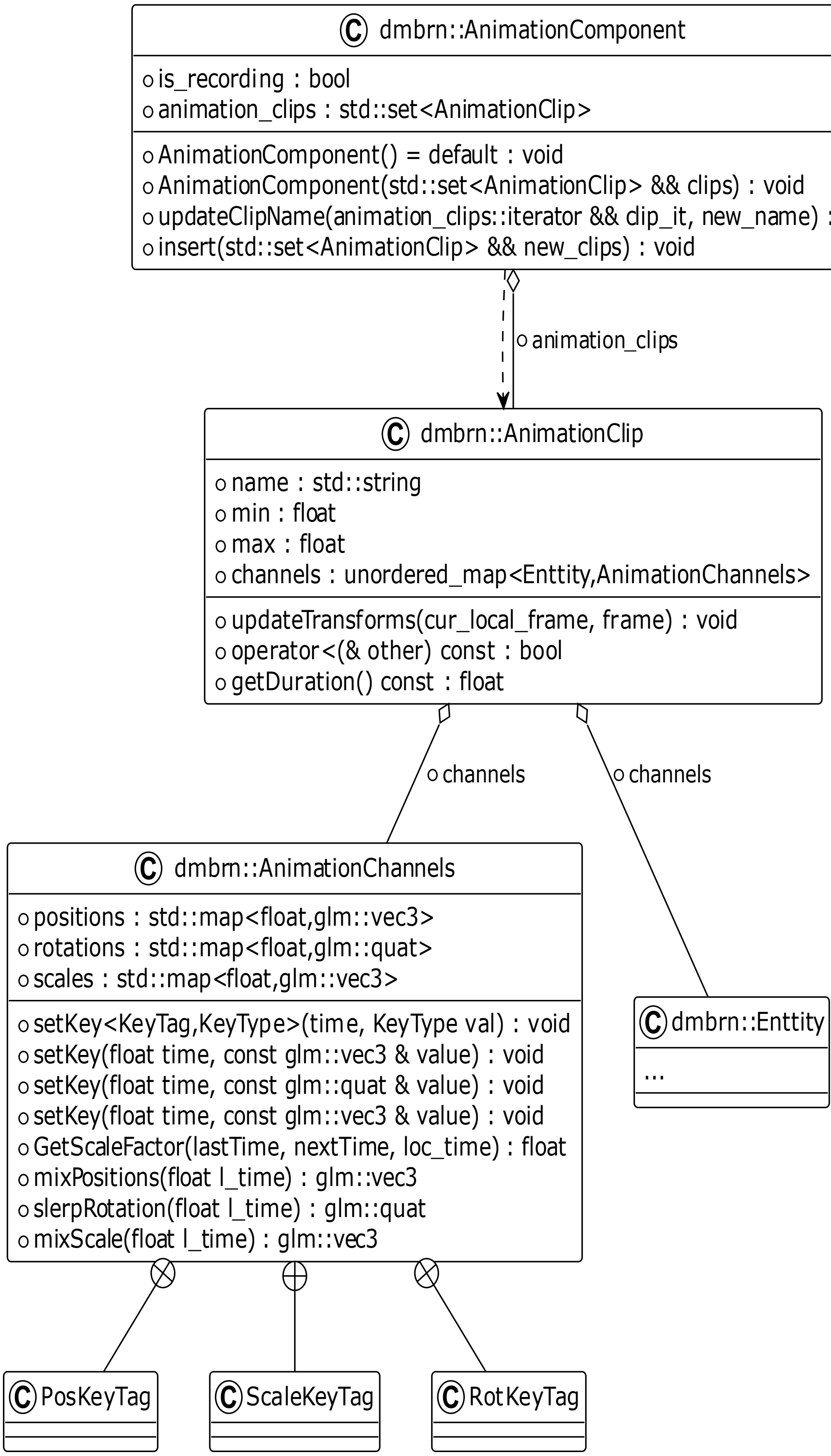
Gthe good thing is thatalso contains AnimationClipThisfollowingth method:updateTransforms-method to update the position of the entities involved in the animation according toclip local time.

AnimationChannelsa class that stores animation keyframes for a specific entity, divided into 3 types (channels): position, orientation, scale. The keyframes themselves are stored in an ordered display (map), keyOwhere m is the time of the key frame and value is the value of the key frame at a given point in time.For interpolationkeyframe values for intermediate position, orientation and scale values, respectively, there are functions mixPositions, slerpRotation and mixScale.

ABOUTthe written relationship between the AnimationComponent, AnimationClip, AnimationChannels and Enttity classes can be seen in the class diagram shown in the figure2.9.

2.5 Scene Subsystem and Model Import Classes

The class is responsible for the scene representations.scene.As mentioned earlier, the scene is a container for entities, it is responsible for thiscontainedan entt::registry object that controls the lifetime of the previously mentioned entt::entities.In addition, the scene contains the scene root identifier scene_root_ and an animation_sequence_ object of the AnimationSequence class, which we'll look at later.



Drawing2.9- AnimationComponent related class diagram

Let's take a quick look at the main methods contained in this class:

- addNewEntityToRoot— adds a new entity as a child of scene_root_.
- addNewEntityAsChild - adds a new entity as a child to the given one.
- updateAnimations-loops through all animated entities for which there is a sequence in animation_sequence_. Finds the clip corresponding to the current global animation time, finds the local time of the clip, and updates the positions of the entities using the AnimationClip::updateTransforms method.
- updateGlobalTransforms -runs from the root of the scene through all the dirty (dirty) to the transformation components until all modified (edited). When the modified one is found, passTto the leaves of the tree, accumulating the transformations of the current entities, if there are entities associated with the GPU memory on this path (such asStaticModelComponent, BoneComponent) markTthem as requiring updating in memoryGPU.
- updatePerStaticModelData— maps buffer memory PerStaticModelData to RAM, iterates through all StaticModelComponents if there is a state update request ingpu,writes the matrix using inGPU_transform_offset from StaticModelComponent.
- updatePerSkeletalData- maps buffer memory PerSkeletonData to RAM, goes through all SkeletonComponentand each of their BoneComponent, if locatedjoint requiringupdate the state in the GPU, writes the matrix usinginGPU_transform_offsetand bone_indfrom BoneComponent.
- getModelsToDraw-get a list of all StaticModelComponent for further rendering.
- getSkeletalModelsToDraw -get a list of all SkeletalModelComponent for further rendering.

Inside a classScenecontains the ModelImporter class, this class contains only static methods and variables,revealsfor classSceneonlytwomethodA:ImportModel and ImportAnimationTo encapsulating helper functions.

ImportModelcreates a child entity of the model in the root of the scene from a file with the specified path. Moreover, you can specify whether to import the bones, in which case the SkeletonComponent component will be added to the model root, and whether to import animations, then the AnimationComponent will be added.

ImportAnimationTo imports animations from a file with the specified path. And adds them to the AnimationComponent::animation_clips of the specified entity, and the AnimationComponent must already exist for the entity.

2.6 GUI classes

Before looking at specific interface elements, let's look at the fundamental objects on which the drawing of the interface to the screen is based.

The first of these objects is an object of the EditorRenderPass class, this object is a wrapper over the render pass objectrenderpass Vulcan,Drawing commands must be written in an instanceProhodarendering. Each render pass instance defines a set of image resources, called attachments, that are used during rendering, theirinitial and final image layout types for these buffers.So the color buffer of this graphics pass has an unknown layout at the beginning (ImageLayout::eUndefined), and at the enddisplay layout (ImageLayout::ePresentSrcKHR).

The next object is an object of the EditorSwapChain class, this object manages the swap chain for the editor window. It contains a vector of EditorFrame objects each representing a frame in the paging chain,we'll look at it in more detail later.. EditorSwapChain objectcontrols the lifetime of its attributes.It also handles the resizing of the swap chain when the window is resized.

Let's take a look at the mentioned classEditorFrame,it represents the frame used in when rendering the application. It contains several Vulkan resources related to rendering and timing.Let's look at its most important attributes:

- command_buffer—bufer commands Vulkan. It is used to write rendering commands.

- image_available_semaphoreAndrender_finished_semaphore: Vulkan semaphores They are used for synchronizationinsideGPU while rendering:
  - image_available_semaphoresignals when an image is available fordrawings on it;
  - render_finished_semaphoresignals the completion of renderingand readiness of the image to display on the screen;
- in_flight_fence —Ofencing Vulkan. HeOused for synchronization between CPU AndGPU, toinformCPUabout the execution of all commands in the buffer of this frame.

The last object of these is an object of the ImGuiRaii class, this object is needed to initialize and control the lifetime of objectsImGUI.

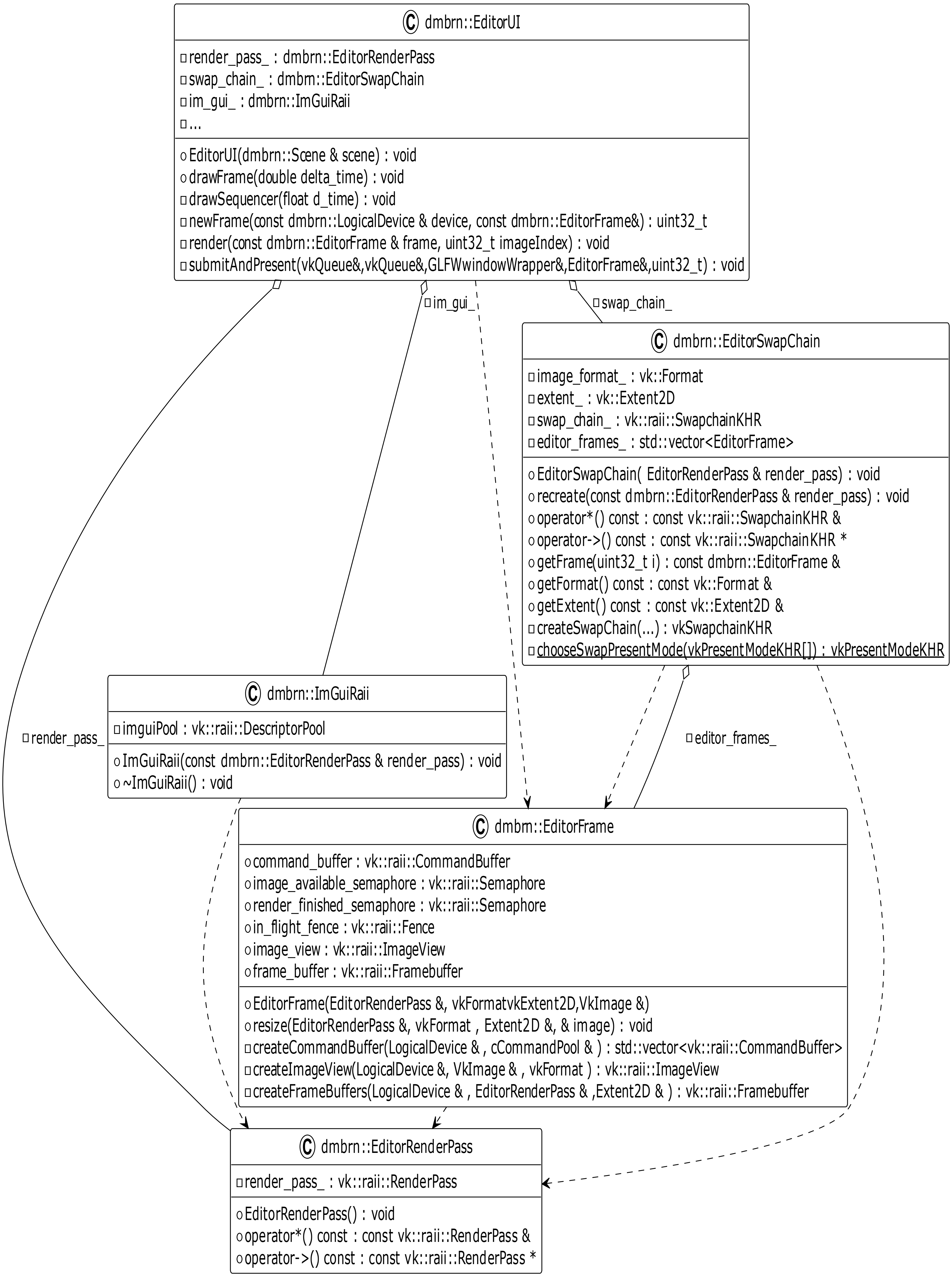
The main orchestrator, container for these and subsequentan objectovis a class objectEditorUI.The objects listed above and following are its attributes and are controlled by it.

Fora visual demonstration of the described relationships in the figure2.10a class diagram containing these classes is presented.

Nowlet's define exactly which GUI elements and their functionality are required to interact with the tool:

- Window dherevAscene, in which the entire subtree is displayed starting from the child elements of the scene root, the functional requirements are:
  - the ability to select an entity by the left mouse button.
- ABOUTscene view window (viewport), in which we can see the scene rendered from the camera position of this window, the functional requirements are:
  - the selected entity must somehow stand out;
  - the ability to change the position of the camera using the buttons on the keyboard;
  - the ability to change the orientation of the camera by dragging the mouse cursor while holding the key.

- Inspector window allowing you to see andinteract withcomponentamiselected entity.
- Animation sequencer window,displaying the location of animation clips on the timeline, for each animated entity, functional requirements:
  - drag and drop (drag'n'drop) new animation clips per panel fromAnimationComponent;
  - the ability to change the start time of the installed clips.

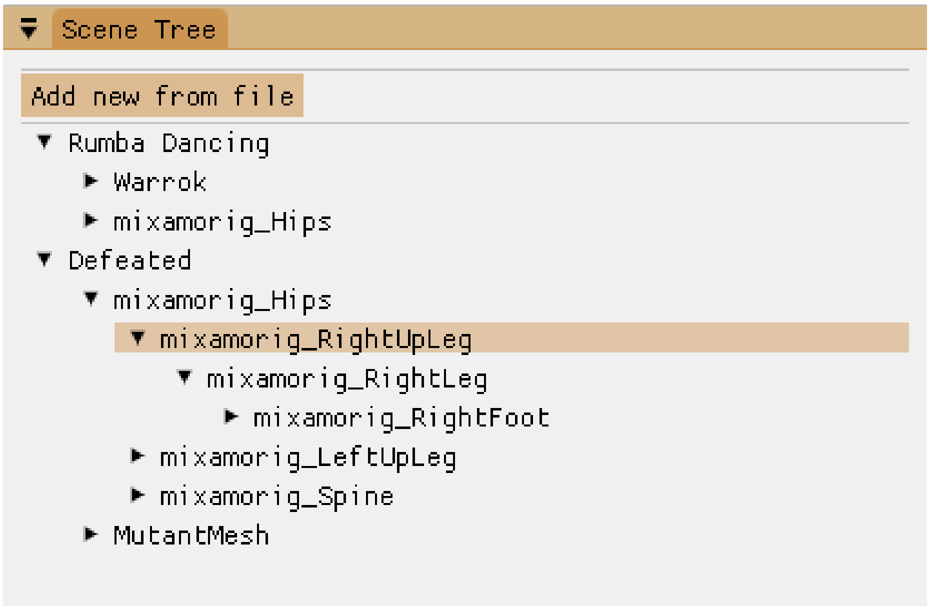


Drawing2.10— GUI Fundamental Class Diagram

First, let's take a look at the scene tree window and its associated SceneTree class. This class contains only two attributes - scene_ reference to the scene and selected_ identifier of the selected entity.

The method responsible for drawing this window is newImGuiFrame. The tree is drawn using the ImGui::TreeNodeEx function, the identifier of the tree vertex is the identifier of the corresponding entity. Using the ImGui::IsItemClicked function, a check is made for clicking on the top of the tree with the selection of an entity.

In addition, this window has a button called "Add new from file", when clicked, a modal window appears in which the user can enter the path to the model and import parameters such as whether to import with bones or with animations. On the image2.11the view of the window is shown with two models previously imported,selected entity with namemixamorig_RightUpLeg, colors are inverted to save ink when printing.



Drawing2.11— Scene tree windows

Next, consider the scene viewport and its associated Viewport class.At its core, the viewportcontains onlyimagedescribed by the Texture class, on which the scene state was drawn.However, since our engine uses double buffering technology, we need to introduce an additional class responsible for changing images ViewportSwapChain.

One "frame", which is controlled by the ViewportSwapChain, consists of:

- color_buffers_-colorbuffer for the description of which the previously encountered class is responsibledmbrn::Texture,however, unlike the previous textures, the one created this time has an additional flag for using it as a target for rendering (vk_ImageUsageFlagBits_ eColorAttachment);
- depth_buffer_ — depth buffer, described by the new class dmbrn::DepthBuffer;
- imgui_images_ds - set of descriptors received fromImguiusing ImGui_ImplVulkan_AddTexture methodWithimage layout VK_IMAGE_LAYOUT_SHADER_READ_ONLY_ OPTIMAL,for each ofcolorbuffers.

DepthBuffermainly different fromtexturebecause instead ofRGBpixels in the depth map arefloat32values, also in case the stencil buffer is enabled may contain8uintmeaning.ANDusage flag as target for depth and stencil buffers (vk::ImageUsageFlagBits::eDepthStencilAttachment).

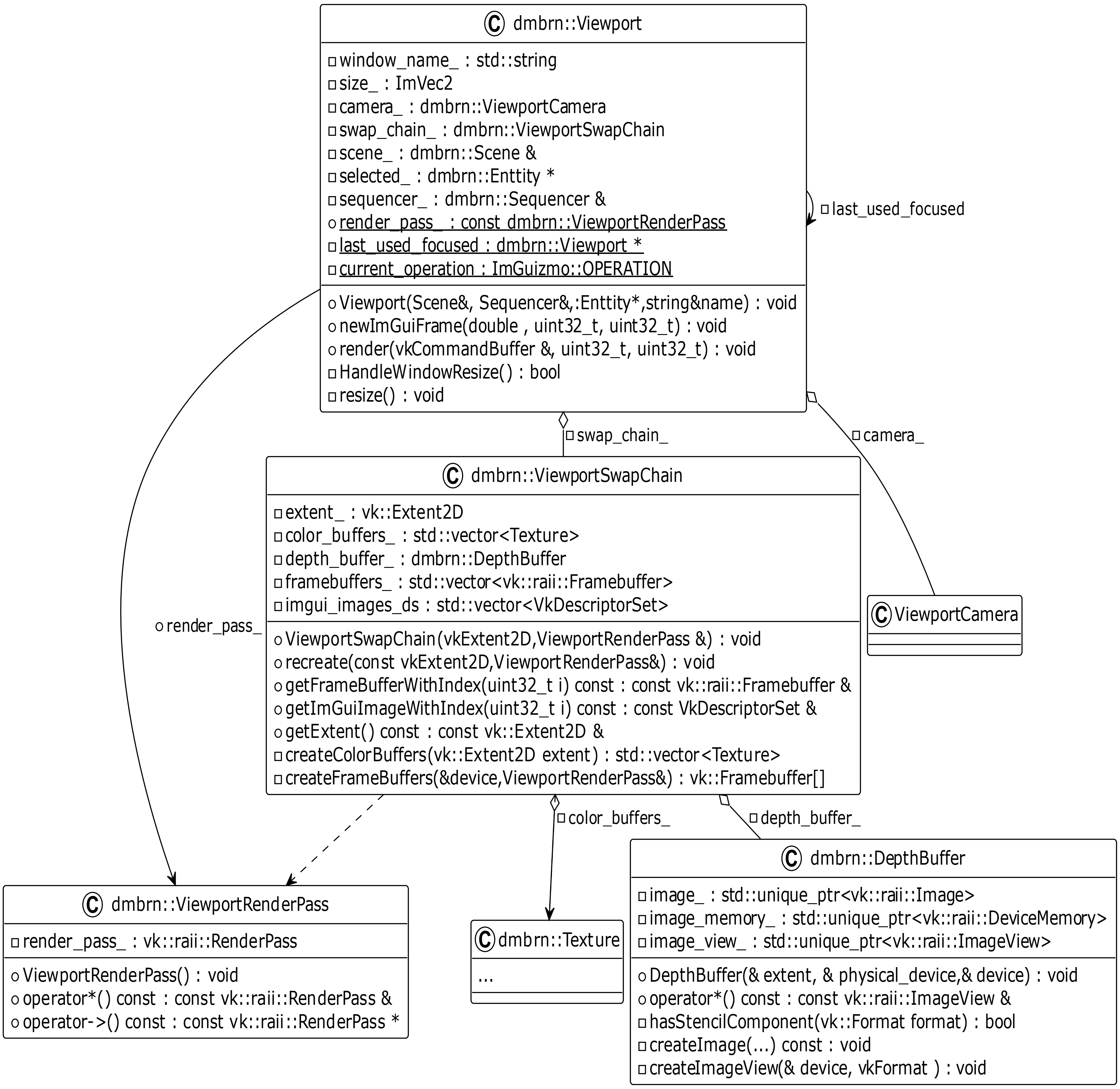
Each viewport renders relative to its camera positionand executes it on its color buffer from ViewportSwapChain with the appropriate sizeviewport size. To organize rendering to this color buffer, we again need to create a graphics pass for each viewport, the ViewportRenderPass class is responsible for this. And this time the initialand finallayoutis ImageLayout::eShaderReadOnlyOptimal toImguicoulduse this image in the fragment shader when renderingui.

For clarity, the described relationships are shown in the figure.2.12in the form of a class diagram.

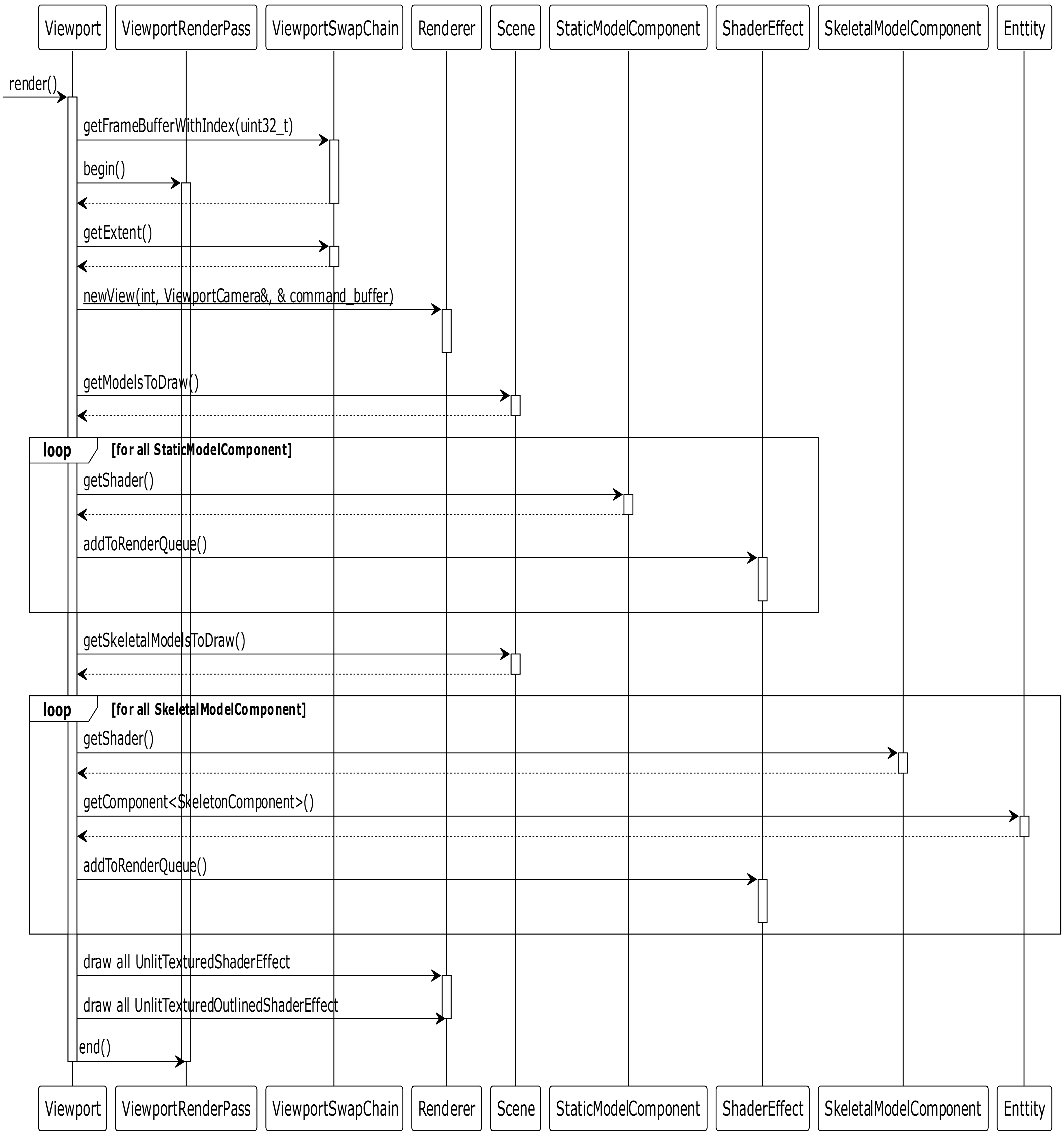
When rendering an interfaceImguithis window, only the image is located ImGui::Image and then the tool is processed and drawngizmo using the ImGuizmo class from the ImGuizmo library[17].

Let's also consider the content of the method responsible for drawing the contents of the window -render.This method first starts the render pass, then goes through all the static models received from the scene and puts them in the queue for rendering for the shader lying in the StaticModelComponent. ThenAtaxablewith skeletal models. It then causes all shaders to be drawn and ends the render pass. For greater clarity, the sequence diagram for this method is shown in the figure.2.13.

Now let's take a look at the inspector window. If the selected entity exists, then we check the existence of each component using the methodentity::tryGetComponent, if the current component exists, we create the top of the drop-down tree-list using the ImGui::TreeNodeEx method and then calling the necessary functionsImguioutput information about the components.



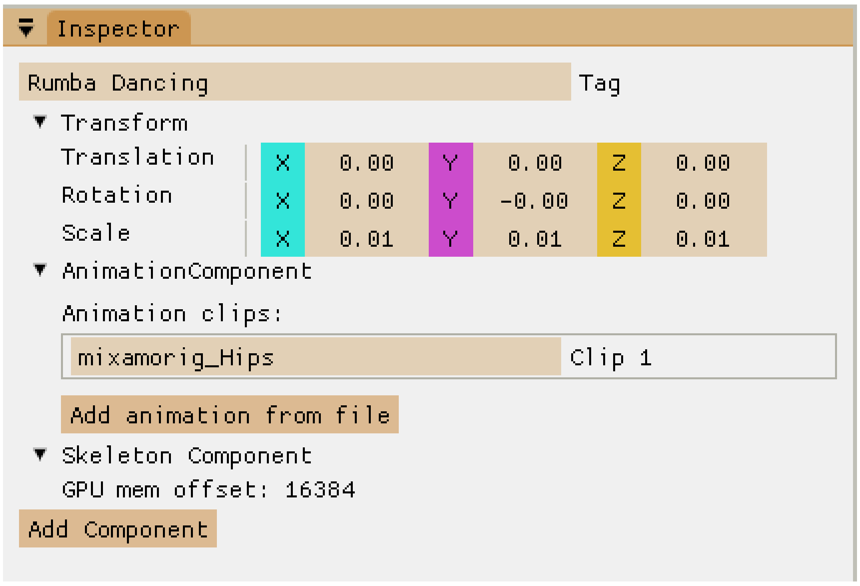
Drawing2.12- Diagram of classes related to rendering the viewportscenes



Drawing2.13- Sequence diagram of the function of rendering the scene view windows

The most interesting to consider is the AnimationComponent because it contains the drag source (drag and drop). INImguisuch a source can be any interface element, in our case it is ImGui::InputText containing the name of the animation. In order to declare a drag element, you need to write ImGui::BeginDragDropSource, if this function returned true, then dragging has begun and we need to form an element for dragging. In our case, this is a pair of values: the identifier of the entity with this component and a pointer to this animation clip, then by calling ImGui::SetDragDropPayload we register it inImgui.Then the cargopayload) can be accepted in DragDroptarget.

On the image2.14showing window viewwith the selected entity havingAnimationComponentAndSkeletonComponent.Colors are inverted to save ink when printing.



Drawing2.14- Inspector window

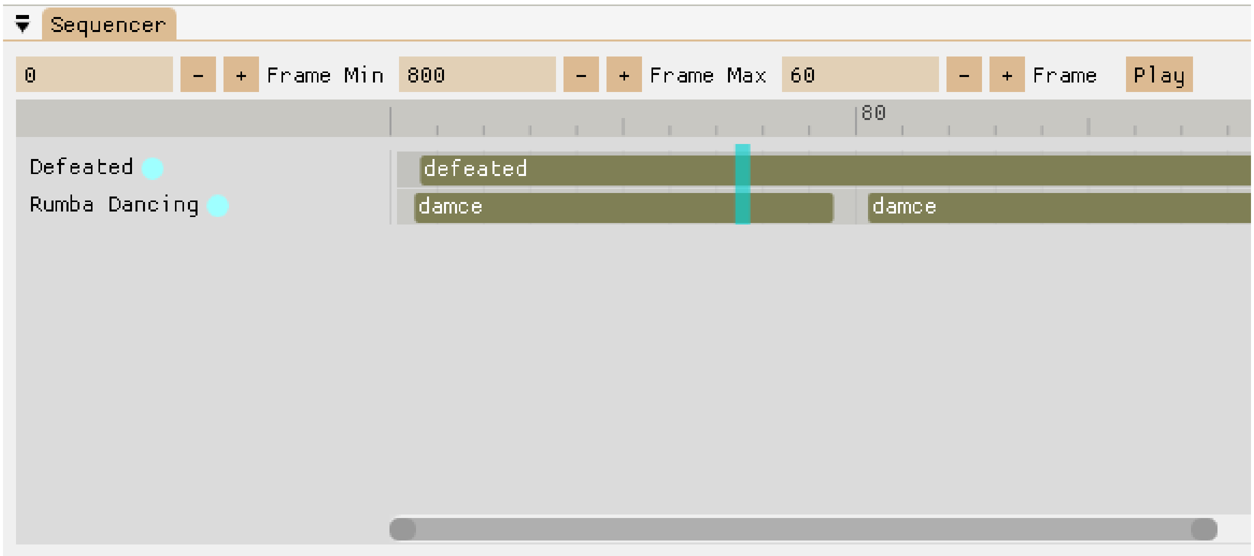
Now let's take a look at the sequencer window and the Sequencer class that defines it.This class is a heavily modified and modified ImSequencerfromImGuizmo[17]. To draw it, the possibilities are usedImguion creating custom widgets using ImDrawList draw lists. The user can populate the lists with variousprimitives, for example, rectangles (AddRectFilled), text (AddText), etc. In this case, absolute positioning is performed using screen coordinatesin pixels. To determine the position and size of the window, the

ImGui::GetCursorScreenPos and ImGui::GetContentRegionAvail functions are used, respectively.In addition to this inconvenient factor, you need to consider that the draw list does not have depth indicators or element layers and elements are displayed in the order they were added to the list.

Thus, briefly rendering this element as a sequence of the following actions:

- Rendering Controls Using Standard FunctionsImgui.The controls are the input field for the minimum, maximum and current frame of the animation clip, start playing or stop.
- Update the positions of the left and right ends of the scrollbar\zoom (frameBarPixelOffsets), by linear interpolation from the current totarget.
- Determining the first visible frame firstFrame. Specifies the width of the frame in pixels.
- Drawing the background.
- Detection of user movement of the current frame when clicking on the top of the panel, updating the current frame.
- Update the current frame if it is playing.
- Rendering of the top panel, division price lines and digital frame values.
- Drawing the names of animated entities on the left.
- Drawing vertical lines in the clip zone.
- Drawing clips of each entity at the appropriate position.
- The start of the drop target ImGui::BeginDragDropTarget.
    - After we've accepted the ImGui::AcceptDragDropPayload payload, we can check if the user has just hovered over the IsPreview target, or if the user has already dropped IsDelivery. In the first case, we can render a clip on the timeline for the user. And after delivery, add it to the corresponding entity
- Draw a vertical rectangle on the current frame.
- Draw a scrollbar and handle the start of its movementmostor its ends.

On the image2.15showing window viewsequencer.The current frame is60, Rumba Dancinghas two clips,Defeatedone. Colors are inverted to save ink when printing.



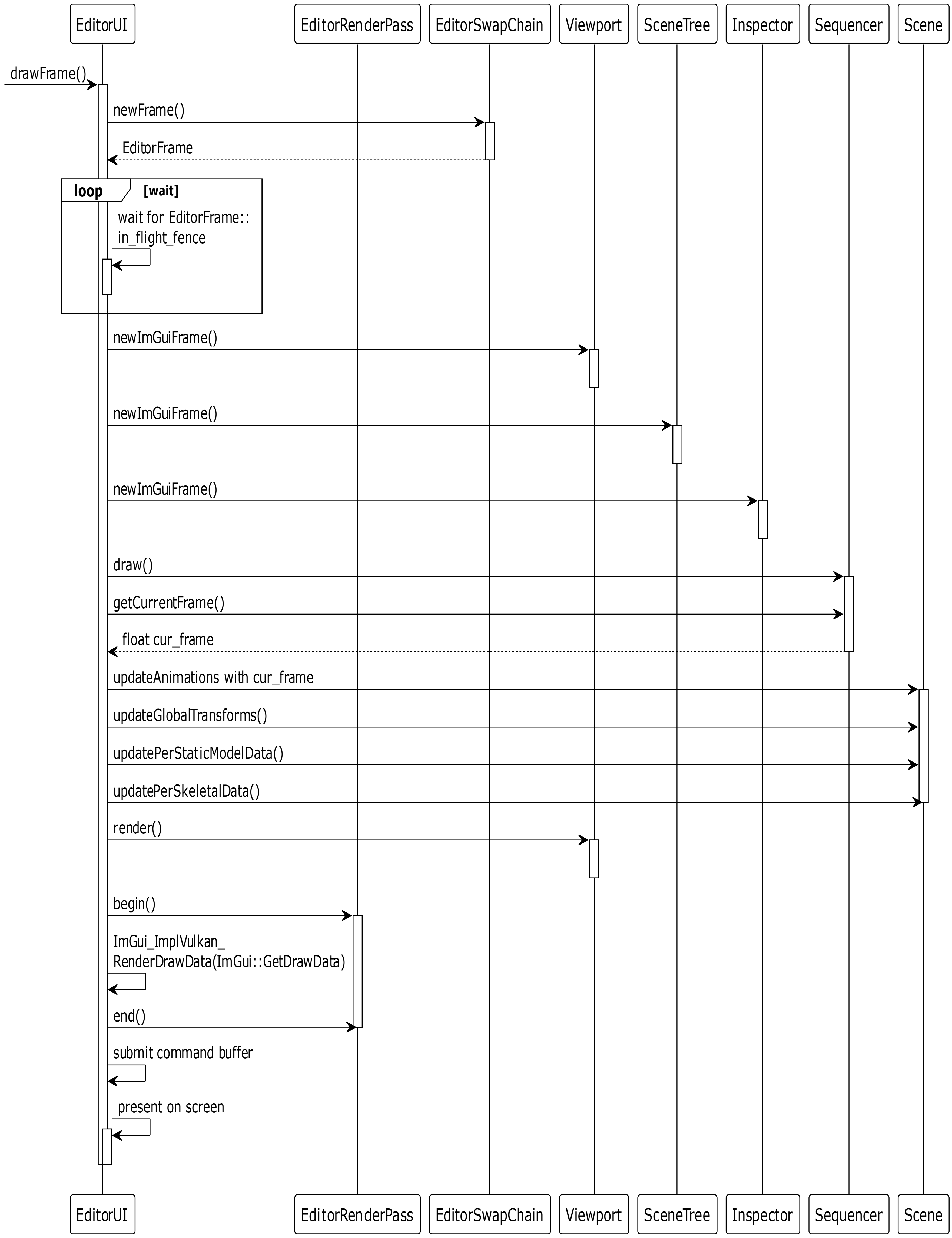
Drawing2.15- Animation sequencer window

Last, let's look at a method that brings everything previously described together. The method for drawing the entire interface and updating all scene states is EditorUI::drawFrame. On the image2.16the sequence diagram of this method is given. Additionally, let's briefly analyze the sequence of actions performed in this method:

- next\current EditorFrame is taken from EditorSwapChain;
- waiting for the in_flight_fence barrier to complete allpastdrawing commandscurrentframe;
- performedcapturing an image signaling the image_available_semaphore semaphore;
- function callImguisignalingXabout a new frame;
- each window of the interface is drawn;
- animations are updatedAnd, then TransformComponent's objects on the stage;
- updated data onGPU
- commands are being writtenrenderingwindowsAscene viewing;
- commands are being writtenrenderinginterfaceImgui;
- sends a command buffer to the queue with a semaphore to wait for

image_available_semaphore and a semaphore to signal render_finished_semaphore;

- the image is displayed on the screen with the render_finished_semaphore wait semaphore.



Drawing2.16— Sequence diagram of interface rendering method

3 DEVELOPMENT AND STANDARDIZATION OF SOFTWARE

As an additional section, the section "Development and standardization of software tools" was chosen, because. the main goal of the final qualifying work is to develop softwarefacilities.

In this section, we will look at organizingprocesssoftware design (PS) using international and domesticmethodsregulating the mainstageslife cycle of the PS and defining the requirements for the final product.

This section includes decisions and results obtained on the following issues:

- project work planning, within the framework of technical and working design using Gantt charts (strip charts);
- determination of the code of the developed software product in accordance with the all-RussianAndclassifierami products;
- calculation of costs for the implementation and implementation of the project, calculation of the project price and the price of the proposed software product.

3.1 Project planning

Work planning is a key management function required to guide the design and implementation of the OS. The main goals of work planning are as follows:

- establishing the overall scope of work and the order in which they will be implemented, taking into account various factors, such as communicationsand dependenciesbetween jobs or the time required to release resources;
- appointment of executors and co-executors for each work;
- determination of the deadline for each work and the time for the implementation of the entire project as a whole.

For a formal description of a set of planned activities, there isseveral methods based on the visualization of processes and allowing to monitor the performance of work to a different extent and adjust their organization. Most commonly used for planning and project management purposes.tapediagrams

(Gantt charts), operograms and network diagrams (PERT - diagrams).

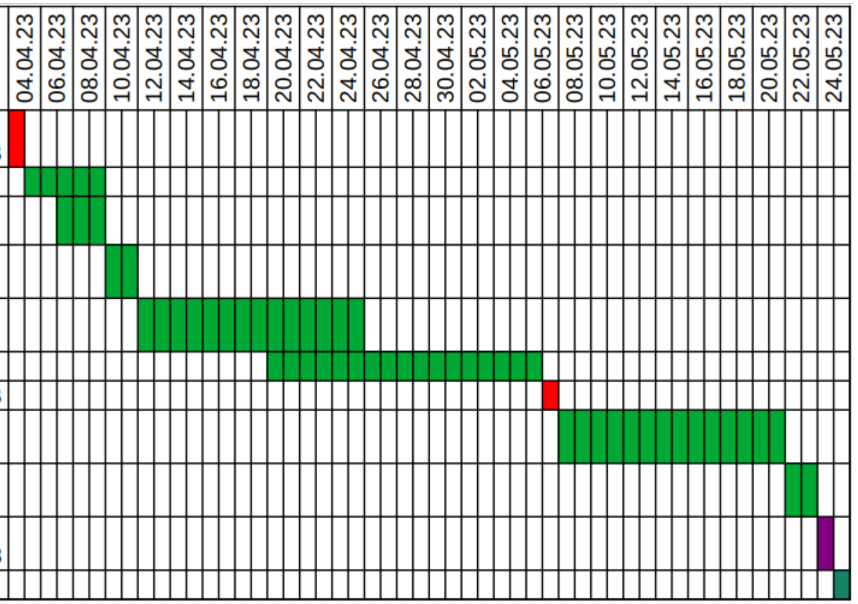
To visualize the set of planned activities, wewe will useGantt chart, which is a bar chart with bars corresponding to the duration of the work. In addition, on the diagram, you can indicate the start and end dates of each work, as well as its performer.

Before setting the deadlines for the work and building a complete diagram, we will form a list of work that needs to be done, forfacilitiesArrange jobs in ordertheir implementation:

1. formulation of terms of reference;
2. search for scientific literature on the topic;
3. study of scientific literature on the topic;
4. choice of technologies and libraries;
5. design and development of software modules architecture;
6. implementation of software modules;
7. testing and debugging;
8. writing an explanatory note and feedback on the work;
9. preparation of illustrative materials;
10. verification and confirmation of compliance with GOSTs;
11. plagiarism check.

Let's make a Gantt chart with the following indicators: the duration of the work in days, the start and end dates of the work, and the list of performers. When compiling the list of performers, the following encoding of names was used:Head of Diploma Design:A. I. Vodyakho - AIV,graduate student:O. V. Evdokimov - OBE,consultant fromDepartment of VT I.S. Zuev—satellite. Thus, we got the diagram shown in the figure3.1.

За-дача	Наименование	Дата на-чала	Дата за-вершения	Длитель-ность (дни)	Исполни-тели
1	формулирование технического зада-ния	04.04.23	04.04.23	1	ОВЕ, АИВ
2	поиск научной литературы по теме	05.04.23	09.04.23	5	ОВЕ
3	изучение научной литературы по теме	07.04.23	09.04.23	3	ОВЕ
4	выбор необходимых технологий и библиотек	10.04.23	11.04.23	2	ОВЕ
5	проектирование и разработка архи-тектуры программных модулей	12.04.23	25.04.23	14	ОВЕ
6	реализация программных модулей	20.04.23	02.05.23	13	ОВЕ
7	тестирование и отладка	03.05.23	03.05.23	1	ОВЕ, АИВ
8	написание пояснительной записки и отзыва по работе	04.05.23	21.05.23	18	ОВЕ
9	подготовка иллюстрационных матери-алов	22.05.23	23.05.23	2	ОВЕ
10	проверка и подтверждение соответ-ствия ГОСТ'у	24.05.23	24.05.23	1	ОВЕ, ИСЗ
11	проверка на плагиат	25.05.23	25.05.23	1	ОВЕ



Drawing3.1— Gantt Chart

3.2 Determining the code of the developed softwarefacilities

Product classifiers are needed so that developers can tell potential buyers about their products, and potential buyers can easily find the product that suits them among the numerous market offers.

Since the formation of our state in the field of software, there have been developed and subsequentlycanceledseveral different classifiers. We will focus on just two of them:

- The first all-Russian classifier of OKP products (OK 005-93) valid until 2017;
- The current OKPD 2 classifier introduced in 2014.

Considering a canceled classifier can be helpful. Because, pprograms developed up to 2017,savecode correspondinggivenclassifier. Therefore, in order to find analogues of the software tool created within the framework of the WRC, it is useful to considerAnd alreadyexcellent classifier. In addition, it gives a more accurate classification of software than the current one.

Codes for both classifiers consist of decimal digits. The classification is hierarchicalmoving from higher to lower ranks refines the classification.

Let's determine the code of our software tool according to the current OKPD 2 classifier,the structure of which is presented in the table3.1.

Table3.1- The structure of the OKPD2 code

Position in code(marked 0)	Level name
00.XX.XX.XXX	Class
XX.0X.XX.XXX	Subclass
XX.X0.XX.XXX	Group
XX.XX.0X.XXX	Subgroup
XX.XX.X0.XXX	View
XX.XX.XX.00X	Category
XX.XX.XX.XX0	Subcategory

According to himsoftware toolbelongs to:

- class 62 with a breakdown: “software products and software development services; consulting and similar services in the field of information technology”;

- subclass 0 with a breakdown: “software products and software development services; consulting and similar services in the field of information technology”;
- group 1 with a breakdown: "software products and services for the development and testing of software";
- subgroup 2 with decoding: "original software";
- type 9 with decoding: “other software originals”;
- categories and subcategories 000 with decryption: "other software originals".

Thus, collecting all the components of the code together, we get: 62.01.29.000.

Now let's carry out the definition of the code according to the OKP classifier, the structure of which is presentedon the image3.2.

X X	**X**	**X**	**X**	**X**	**КЧ**	**Наименование продукции**
Класс продукции	**Пподкласс**	**Ггруппа**	**Подгруппа**	**Вид продукции**	**Контрольное число**	

Drawing3.2— OKP code structure

According to the OKP classifier, the software tool belongs to:

- class 50 with decoding: “software and information products of computer technology”;
- subclass 2 with decryption: "general-purpose software";
- group 6 with decryption: “software tools for multimedia systems”;
- subgroup 1 with decoding: "software tools for designing multimedia elements, explanation: this group includes PS for designing video, sound, computer graphics and other multimedia elements."

Thus, collecting all the components of the code together, we get: 502610.

3.3 Determination of costs for the implementation and implementation of the project

In order to estimate the cost and price of a project in the field of software development at the initial stage of an agreement with a potential customer (investor), we will use the lumped calculation method. This technique allows you to determine the cost of the project based on the cost of one working day of the designer and the labor costs (labor intensity) of the project as a whole. At the same time, we take into account the composition and degree of loading of the project team members.

Conditions that we will accept when calculating project costs andproject prices:

- The main project developer (VKR) is a graduate student who performs design on time: from April 4, 2023 to May 25, 2023, which is 34 working days with a full load ($К_{загр.разр}=1{,}0$).
- The project is attended by the head of the WRC with a load factor $К_{загр.рук}=0{,}05$and consultant for an additional section with a load factor $К_{загр.доп}=0{,}03$. The salary of these participants is accepted in accordance with their positions in the university. The percentage of overhead costs for LETI is conventionally assumed to be 42%.

Next, we need to determine the monthly salary. We will use a career service as a data source.“Habr Career”(https://career.habr.com/salaries).So, according to this service, the average salary for all IT specializations based on 6754 questionnaires for the 1st half of 2023 is 179,307 rubles.

For a novice designer performing WRC,adjust the resulting salary value to 60,000 rubles.

In accordance with the order No. 2206 dated 04.10.2013 "On increasing the level of remuneration of university employees",the funding standard for full-time faculty members is 25,000 (per month).

Before starting the calculations, we determine the necessary constants:

- $T_{ср}=20,58$- average monthly number of working daysfor 2023, with a five-day work week;
- $Ф=0,302$- the percentage (share) of insurance premiums calculated from the payroll fund includes contributions to the Federal Tax Service and the Social Insurance Fund;
- $П=0,15$- the average profit of the contractor in the implementation of projects in the field of IT;
- $Н=0,5$- percentage of overhead costs, for design organizationsin the field of informatics it fluctuates, as a rule, from 40% to 80%;
- $НДС=0,2$- value added tax rate.

Let's calculate the total cost per day per developer, presented in the table3.2, Andstaff member of LETI, are presented in the table3.3.IN tablescalculation formulas are also given.

Table3.2— Developer rate calculation

Article title	unit	Expenses,rub	Note
The value of the average monthly accrued wages specialist fees	rub./month	60,000.00	Employee salary$Z_{зп}$
Rate calculation			
Daily rate($Z_д$)	rub./day	2915.45	$Z_д=Z_{зп}/T_{ср}$
Insurance premiums 30.2% of employees' wages($C_{сд}$)	rub./day	880.47	$C_{сд}=Z_д*Ф$
Payment of essential workers with insurance premiums ($Z_{дс}$)	rub./day	3,795.92	$Z_{дс}=Z_д+C_д$
overhead ($C_{нр}$)	rub./day	1457.73	$C_{нр}=Z_д*Н$
Cost of one person/day ($C_{ч/д}$)	rub./day	5253.64	$C_{ч/д}=Z_{дс}+C_{нр}$
Daily profit ($C_{прд}$)	rub./day	788.05	$C_{прд}=C_{ч/д}*П$
Specialist rate excluding VAT ($C_{дсс}$)	rub./day	6,041.69	$C_{дсс}=C_{ч/д}+C_{прд}$
Daily amount of VAT ($C_{ндс}$)	rub./day	1208.34	$C_{ндс}=C_{дсс}*НДС$

Specialist rate per day from including VAT ($C_{полн.разр}$)	rub./day	7250.03	$C_{полн.разр}=C_{дсс}+C_{ндс}$

Now we can determine the price of the project using the following formula:

$$C_{пр}=\sum_{i=1}^{n} C_{полнi} T_i K_{загрi},$$

Where:

- $C_{полнi}$—full day job costi-first specialist [ruble/day];
- T_i— time of participation of the i-th specialist in the work on the project [days];
- $K_{загрi}$— load factori-th specialist work in the project;
- n— the number of specialists employed in the project.

Applying this formula to our case, we obtain the following calculation formula,where the first term is the cost of the developer, the second term is the cost of the manager, and the third term is the cost of two consultants:

$$C_{пр}=7250{,}03*34*1{,}0\ +\ 2886{,}73*2*0{,}05\ +\ 2886{,}73*2*0{,}03=246962{,}89\ руб.$$

Table3.3— Rate calculationETU "LETI" employee

Article title	unit	Expenses,rub	Note
The value of the average monthly accrued wages specialist fees	rub./month	25,000.00	Employee salary$Z_{зп}$
Rate calculation			
Daily rate($Z_д$)	rub./day	1,214.77	$Z_д=Z_{зп}/T_{ср}$
Insurance premiums 30.2% of employees' wages($C_{сд}$)	rub./day	366.86	$C_{сд}=Z_д*Ф$
Payment of essential workers with insurance premiums ($Z_{дс}$)	rub./day	1,581.63	$Z_{дс}=Z_д+C_д$
overhead ($C_{нр}$)	rub./day	510.20	$C_{нр}=Z_д*Н$
Cost of one person/day ($C_{ч/д}$)	rub./day	2,091.84	$C_{ч/д}=Z_{дс}+C_{нр}$
Daily profit ($C_{прд}$)	rub./day	313.78	$C_{прд}=C_{ч/д}*П$
Specialist rate excluding VAT ($C_{дсс}$)	rub./day	2405.61	$C_{дсс}=C_{ч/д}+C_{прд}$
Daily amount of VAT ($C_{ндс}$)	rub./day	481.12	$C_{ндс}=C_{дсс}*НДС$

Table3.3— Rate calculationETU "LETI" employee

Article title	unit	Expenses,rub	Note
Specialist rate per day from including VAT ($C_{полн.штат}$)	rub./day	2,886.73	$C_{полн.штат}=C_{дсс}+C_{ндс}$

If new (created in a project) software is released to the market as a custom solution and the cost-based pricing method is used, then its price can be found from the following expression:

$$C_{прогр}=C_{пр}+C_{изгот}(1+П)(1+НДС),$$

Where:

- $C_{пр}$- the project price calculated earlier;
- $C_{изгот}$- copying costs (copy making),acceptgivenmeaning$C_{изгот}=200$ руб, as including the cost of hosting the server with the distribution of the installation files of the program andproduction of oneCD-disk with the program;
- $П=0,5$- profit included by the developer in the price.

Then the calculation formula:

$$C_{прогр}=246962,89+200*1,5*1,2=247322,89 \text{ руб}$$

When the project involves the sale of the program to several consumers, then the costs of the project (taking into account the profit of the seller) should be divided among the buyers,according to the following formula:

$$C_{пргр.тир}=\frac{C_{прогр}}{N_{тир}},$$

Where:

- $C_{пргр.тир}$- the price of the program when it is replicated;
- $N_{тир}$- the planned (guaranteed) circulation of the sale.

Accept$N_{тир}=100$, then we get the following calculation formula for the price of the program:

$$C_{пргр.тир}=\frac{247322,89}{100}=2473,22 \text{ руб}$$

CONCLUSION

As a result of the work, a software tool was created that allows the user to easily and quickly combine animation andAndfor various characters from librariespre-made animations such asMixamo. The software tool has an intuitive graphical interface, supports the import of 3D models andtheiranimations in various formats, and also provides the ability to view and editanimation playback sequences for any object on the stage.

Thus, the goal of the work was achieved, and all the tasks were solved. The developed software tool is a contribution to the development of the field of 3D animation.

Possible directions for further development of the topic are:

- Ability to save the scene and created animation sequences to a file on disk.
- Possibility to export received animation clips to various video formats.
- DAdding functionality to create your own animations or edit existing ones.

LIST OF USED SOURCES

1. Blender main page [Electronic resource] - Access mode:https//www.blender.org/. (Date of access: 04/12/2023).

2. Main page Unity [Electronic resource] - Access mode:https://unity.com/ru. (Date of access: 04/12/2023).

3. Assimp GitHub [Electronic resource] - Access mode:https://github.com/assimp/assimp. (Date of access: 04/12/2023).

4. Dear ImGui GitHub [Electronic resource] - Access mode:https://github.com/ocornut/imgui. (Date of access: 04/12/2023).

5. Graphical APIs of high and low level: differences and principle of operation [Electronic resource] - Access mode:https://itigic.com/ru/high-and-low-level-graphical-apis-differences/. (Date of treatment: 05/04/2023).

6. Vulkan Graphics API Presented and Your NVIDIA GPUs Ready [Electronic Resource] - Access Mode:https://www.nvidia.com/ru-ru/drivers/vulkan-graphics-api-blog/. (Accessed: 04.05.2023).

7. GLM github [Electronic resource] - Access mode:https://github.com/g-truc/glm. (Date of access: 04/12/2023).

8. What is ECS and what is it compiled with [Electronic resource] - Access mode:https://dtf.ru/gamedev/954579-chto-takoe-ecs-is-chem-ego-kompilyat. (Accessed: 04.05.2023).

9. GitHub Entt [Electronic resource] - Access mode:https://github.com/skypjack/entt. (Date of access: 04/12/2023).

10. stb github [Electronic resource] - Access mode:https://github.com/nothings/stb. (Date of access: 04/12/2023).

11. IMGUI wiki on GitHub [Electronic resource] - Access mode:https://github.com/ocornut/imgui/wiki/About-the-IMGUI-paradigm. (Accessed: 04.05.2023).

12. GLFW GitHub [Electronic resource] - Access

mode:https://github.com/glfw/glfw. (Accessed: 04.05.2023).

13. Jason Gregory. game engine architecture. 2018.

14. Sellers, G. . Vulcan. Developer Guide. 2017.

15. Vulkan Shader Resource Binding [Electronic resource] - Access mode:https://developer.nvidia.com/vulkan-shader-resource-binding. (Date of access: 04/12/2023).

16. Vulkan® 1.3.250 - A Specification [Electronic resource] - Access mode:https://registry.khronos.org/vulkan/specs/1.3/html/. (Date of access: 04/12/2023).

17. ImGuizmo GitHub [Electronic resource] - Access mode:https://github.com/CedricGuillemet/ImGuizmo. (Date of access: 04/12/2023).

APPENDIX A
Specification andsimilar codeprograms

The software product specification is given in Table A.1.

TableA.1— Software product specification

Identifiermodule	Purpose of the module
Singletons	TOcontainer for singles
Instance	ABOUTwrapper for vkInstance andvalidation debug layers
surface	ABOUTwrap for SurfaceKHR
PhysicalDevice	ABOUTBertka for vkPhysicalDevice
LogicalDevice	ABOUTBertka for vkDevice
command pool	ABOUTBertka for vkCommandPool
DescriptorPool	ABOUTwrapper for vkDescriptorPool
MaterialRegistryHandle	TOkey in the material register
DiffusionMaterial	ABOUTwrites the surface properties of an object
texture	Manages texture image data on the GPU
DiffusionUniformBuffer	Manages simple dataonGPU diffuseWowmaterialA
DiffusionDescriptorSets	ABOUTwrites and keeps sets diffuse material descriptors
image_data	Represents image data onCPU
MeshRenderData	Manages mesh rendering data on the GPU
mesh	Just a combination of material and mesh render data
HostLocalBuffer	Represents a GPU local buffer
Vertex	ABOUTwrites one vertex of the static mesh
SkeletalMeshRenderData	Manages skeletal mesh rendering data on the GPU
SkeletalMesh	Just a combination of material and skeletal mesh render data
BonedVertex	ABOUTwrites one vertex of the skeletal mesh
renderer	Manages and stores objects needed for rendering
shader effect	ABOUTwrites a common interface for further shader effects
UnlitTexturedShaderEffect	ABOUTwrites a shader effect to draw the object as an unlit texture

Table A.1 continued

Identifiermodule	Purpose of the module

UnLitTexturedGraphicsPipelineStatics	Manages the objects needed to draw with UnlitTexturedShaderEffect
UnLitTexturedGraphicsPipeline	HelpsVcreating a graphics pipeline for UnlitTexturedShaderEffect
UnLitTexturedRenderData	Controls the rendering data of the UnlitTexturedShaderEffectonGPU
UnlitTexturedOutlinedShaderEffect	Describes the effect of a shader for drawing an object as an outlined, undamped texture.
OutlineGraphicsPipelineStatics	Controls the objects needed to draw with UnlitTexturedOutlinedShaderEffect
OutlineGraphicsPipeline	Helps V creating a graphics pipeline for drawing the outline of an object
OutlineShaderEffectRenderData	Manages GPU rendering data
PerStaticModelData	Manages single GPU data for each static model in the scene
PerSkeletonData	Manages single GPU data for each skeletal model in the scene
ViewportCamera	Represents the camerascene view windows
CameraRenderData	Manages camera GPU datascene view windows
UniformBuffer	Represents a CPU visible and coherent data buffer on the GPU
entity	Is a wrapper for entt::entity Andidentifier for comfort
TagComponent	Entity name
TransformComponent	Represents a transformationentities
RelationshipComponent	Describes hierarchical relationships between entities
BoneComponent	Describesessence, whichand Iis a bone of some skeleton
SkeletonComponent	An entity that hassuch a component, is the root of the skeleton
StaticModelComponent	Describes an entity with a visualthperformanceasstatic model

Table A.1 continued

Identifiermodule	Purpose of the module
SkeletalModelComponent	Describes an entity with a visualthperformanceas skeletal models
CameraComponent	Describes camera properties

AnimationComponent	Presence means that this entity and its subtree can be animated
AnimationClip	Stores animation channels for everyoneentitiesinvolved in this clip
AnimationChannels	Stores keyframe values forone entity
Scene	Is the container and manager of all entities
ModelImporter	Performs import of model data from a file
AnimationSequence	Stores and manages a sequence of animation clips for each animated object in a scene
EditorUI	Is the main orchestrator and container for all UI elements
EditorFrame	Represents the editor user interface frame with synchronization data
EditorRenderPass	Represents an editor UI rendering pass
EditorSwapChain	Representslistpaging editor UI
ImGuiRaii	RAII wrapper for ImGui objects
scene tree	Represents a user interface windowWithtreesohmscenes
viewport	Represents a user interface windowview scene
ViewportRenderPass	Represents a render passscene view windows
ViewportSwapChain	Representslist scene view paging
DepthBuffer	Represents a depth bufferscene view windows
Inspector	Represents the user interface window of the inspector
sequencer	Represents the animation sequencer user interface window

Due to the limited volume of the WRC, only the main modules are printed below:EditorUI::newFrame, EditorUI::render, EditorUI::submitAndPresent,EditorUI::drawFrame,viewport::render,Scene::update Animations, Scene::updateGlobalTransforms,Scene::dirtyTraverseTree, Scene::editedTraverseTree, UnlitTexturedShaderEffect::draw, UnlitTexturedShaderEffect::drawStatic.

```
/**
* \brief get an image index for frame and be sure that all synchronization
*is done
* \param device vulkan logical device for fence manipulation
* \param frame editor ui frame data
* \return index of acquired image
*/
uint32_tEditorUI::newFrame(const LogicalDevice& device, const EditorFrame&
frame)
```

```
{
// wait while all previous work for this frame wasn't done
device->waitForFences(*frame.in_flight_fence, true, UINT64_MAX);

// acquire image and signal semaphore when we can start render to it
const auto result = swap_chain_->acquireNextImage(UINT64_MAX,
 *frame.image_available_semaphore);

device->resetFences(*frame.in_flight_fence);

frame.command_buffer.reset();

// ImGui new frame
ImGui_ImplVulkan_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
ImGuizmo::BeginFrame();

return result.second;
}
/**
* \brief record render commands to frame command buffer
* \param frame editor frame for command buffer access
* \param imageIndex swap chain image index of frame buffer
*/
voidEditorUI::render(const EditorFrame& frame, uint32_t imageIndex)
{
const vk::raii::CommandBuffer& command_buffer = frame.command_buffer;

command_buffer.begin({vk::CommandBufferUsageFlags()});

// record commands of viewports
viewport_.render( command_buffer, current_frame_, imageIndex);
viewport2_.render( command_buffer, current_frame_, imageIndex);

// begin imgui render pass
vk::ClearValue clearValue;
clearValue.color = vk::ClearColorValue(std::array<float, 4>({0.5f, 0.5f,
0.5f, 1.0f}));
command_buffer.beginRenderPass({
**render_pass_,
*swap_chain_.getFrame(imageIndex).frame_buffer,
{{0, 0}, swap_chain_.getExtent()},
1, &clearValue
}, vk::SubpassContents::eInline);
// record imgui commands
ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), *command_buffer);

command_buffer.endRenderPass();
command_buffer.end();
}

/**
* \brief submit command buffer to queue and call present
* \param present present queue
* \param graphics graphics queue
* \param window window wrapper to handle window resize
* \param frame editor frame data
* \param imageIndex swap chain image index for presenting
*/
voidEditorUI::submitAndPresent(vk::raii::Queue& present,
vk::raii::Queue& graphics, GLFWwindowWrapper& window,
```

```
constEditorFrame& frame, uint32_t imageIndex)
{
const vk::Semaphore waitSemaphores[] = {*frame.image_available_semaphore};
const vk::PipelineStageFlags waitStages[] =
{vk::PipelineStageFlagBits::eColorAttachmentOutput};
const vk::Semaphore signalSemaphores[] =
{*frame.render_finished_semaphore};

const vk::SubmitInfo submitInfo
{
waitSemaphores,
waitStages,
*frame.command_buffer,
signalSemaphores
};

graphics.submit(submitInfo, *frame.in_flight_fence);

try
{
const vk::PresentInfoKHR presentInfo
{
signalSemaphores,
**swap_chain_,
imageIndex
};
present.presentKHR(presentInfo);
}
catch (vk::OutOfDateKHRError e)
{
window.framebufferResized = false;
swap_chain_.recreate(render_pass_);
}
}

/**
* \brief draw UI, update scene both CPU and GPU states, record and submit
*command buffers
* \param delta_time time of previous frame in ms
*/
voidEditorUI::drawFrame(double delta_time)
{
// get current Editor Frame from swap chain
const EditorFrame& frame = swap_chain_.getFrame(current_frame_);

// get an image index of it and be sure that all synchronization is done
const uint32_t imageIndex = newFrame(Singletons::device, frame);

// begin drawing all the UI
beginDockSpace();

showAppMainMenuBar();
ImGui::ShowDemoWindow();

// draw all windows
viewport_.newImGuiFrame(delta_time, current_frame_, imageIndex);
viewport2_.newImGuiFrame(delta_time, current_frame_, imageIndex);
scene_tree_.newImGuiFrame();
inspector_.newImGuiFrame(current_frame_);
drawStatsWindow();
drawSequencer(static_cast<float>(delta_time));
```

```
// end drawing all the UI
endDockSpace();

// update transforms according to current animation states
scene_.updateAnimations(sequencer_.getCurrentFrame(), current_frame_);
// hierarchically update transforms
scene_.updateGlobalTransforms(current_frame_);
// update GPU data of static models
scene_.updatePerStaticModelData(current_frame_);
// update GPU data of skeletal modes
scene_.updatePerSkeletalData(current_frame_);

// record render commands to frame command buffer
render(frame, imageIndex);
// submit command buffer to queue and call present
submitAndPresent(Singletons::present_queue, Singletons::graphics_queue,
Singletons::window, frame, imageIndex);
current_frame_=(current_frame_ + 1)%
Singletons::device.MAX_FRAMES_IN_FLIGHT;
}
/**
* \brief record commands drawing scene to viewport
* \param command_buffer command buffer to record commands
* \param current_frame current frame index to bind proper descriptor sets
* \param imageIndex swap chain image index to access frame buffer
*/
voidviewport::render(const vk::raii::CommandBuffer& command_buffer,
uint32_t current_frame,
uint32_t imageIndex)
{
// set color of background
const std::array<vk::ClearValue, 2> clear_values
{
vk::ClearValue{vk::ClearColorValue{std::array<float, 4>
{0.3f, 0.3f, 0.3f, 1.0f}}},
vk::ClearValue{vk::ClearDepthStencilValue{1.0f, 0}}
};

// begin viewport render pass with proper frame buffer
const vk::RenderPassBeginInfo renderPassInfo
{
**render_pass_,
*swap_chain_.getFrameBufferWithIndex(imageIndex),
vk::Rect2D{vk::Offset2D{0, 0}, swap_chain_.getExtent()},
clear_values
};
command_buffer.beginRenderPass(renderPassInfo,
vk::SubpassContents::eInline);

// set dynamic viewport
const vk::Viewport viewport
{
0.0f, 0.0f
static_cast<float>(swap_chain_.getExtent().width),
static_cast<float>(swap_chain_.getExtent().height),
0.0f, 1.0f
};
command_buffer.setViewport(0, viewport);

// set dynamic scissors
const vk::Rect2D scissor
{
```

```
vk::Offset2D{0, 0},
swap_chain_.getExtent()
};
command_buffer.setScissor(0, scissor);

// update and bind new view
Renderer::newView(current_frame, camera_, command_buffer);

// add static models to corresponding shader queue
auto static_view = scene_.getModelsToDraw();
for (auto entity : static_view)
{
StaticModelComponent& model = static_view.get<StaticModelComponent>
(entity);
model.getShader()→addToRenderQueue(
{&model.mesh, model.inGPU_transform_offset});
}

// add skeletal models to corresponding shader queue
auto skeletal_group = scene_.getSkeletalModelsToDraw();
for (auto entity : skeletal_group)
{
SkeletalModelComponent& skeletal_model = skeletal_group.get<
SkeletalModelComponent>(entity);
skeletal_model.getShader()->addToRenderQueue({
&skeletal_model.mesh,
skeleton_model.skeleton_ent.
getComponent<SkeletonComponent>().in_GPU_mtxs_offset
});
}

// draw all shader effects
Renderer::un_lit_textured.draw(current_frame, command_buffer);
Renderer::outlined_.draw(current_frame, command_buffer);

command_buffer.endRenderPass();
}

/**
* \brief update transforms according to current animation states
* \param anim_frame current global animation frame
* \param frame index of inflight frame
*/
voidScene::updateAnimations(float anim_frame, uint32_t frame)
{
auto view = registry_.view<AnimationComponent>();

// iterate all animated entities
for (auto ent : view)
{
// if it has some animations in sequence
if (!animation_sequence_.entries_[Enttity{registry_, ent}].empty())
{
// get clip which start time is grater or equal to global time
auto clip_it = animation_sequence_.entries_
[Enttity{registry_, ent}].lower_bound(anim_frame);

// if it is past-the-end or
// (is not the first in sequence and not equal)
if (clip_it == animation_sequence_.entries_[
entity{registry_, ent}].end() ||
clip_it != animation_sequence_.entries_[
```

```
Enttity{registry_, ent}].begin()&&clip_it->first!=anim_frame)
// move back to get lessorequal
--clip_it;

// here clip_it have less or equal

// calculate clip local time
const float local_time = glm::clamp(clip_it->second.min +
anim_frame - clip_it->first,
clip_it->second.min,
clip_it->second.max);

// actually updating transform with local time
clip_it->second.updateTransforms(local_time, frame);
}
}
}

/**
* \brief hierarchically update transforms
* \param frame index of current in flight frame
*/
voidScene::updateGlobalTransforms(uint32_t frame)
{
// begin traversing tree starting from scene root
dirtyTraverseTree(scene_root_, frame);
}

/**
* \brief traversing all dirty paths in tree to find edited entities
* \param ent current entity
* \param frame index of current in flight frame
*/
voidScene::dirtyTraverseTree(Entity ent, uint32_t frame)
{
// transform component of this entity
TransformComponent& this_trans = ent.getComponent<TransformComponent>();

if (this_trans.isEditedForFrame(frame))
// if is edited traverse tree up until leaves to
// update global trans mtx's
{
// unedit and clear
this_trans.edited[frame] = false;
this_trans.dirty[frame] = false;

// get relationship component of entity
const RelationshipComponent& ent_rc = ent.
getComponent<RelationshipComponent>();
glm::mat4 parent_trans = glm::mat4(1.0f);

// if this is not a root, parent could be null only for scene root
if(ent_rc.parent)
{
parent_trans = ent_rc.parent.getComponent<TransformComponent>()
.globalTransformMatrix;
//ent_rc.parent.getComponent<TransformComponent>().getMatrix();
}

// traverse tree up until leaves to update global trans mtx's
editedTraverseTree(ent, parent_trans, frame);
}
```

```
else if (this_trans.isDirtyForFrame(frame))
// if is dirty traverse tree while edited not found
{
// clear transform of this
this_trans.dirty[frame] = false;

// get relationship component of this entity
const RelationshipComponent& cur_comp = ent.
getComponent<RelationshipComponent>();
// get first child
entity cur_child = cur_comp.first;

// recursively call dirtyTraverseTree for all children
while (cur_child)
{
dirtyTraverseTree(cur_child, frame);
cur_child = cur_child.getComponent<RelationshipComponent>().next;
}
}
}

/**
* \brief traverse tree accumulating transformation matrix of each entity
* \param ent current entity
* \param parent_trans_mtx parent transformation matrix
* \param frame index of current in flight frame
*/
voidScene::editedTraverseTree(Entity ent, glm::mat4 parent_trans_mtx,
uint32_t frame)
{
// transformation of this node is mul of parent and this
TransformComponent& ent_tc = ent.getComponent<TransformComponent>();
const glm::mat4 this_matrix = parent_trans_mtx * ent_tc.getMatrix();

// unedit and clear
ent_tc.edited[frame] = false;
ent_tc.dirty[frame] = false;

// memorize new global transformation matrix
ent_tc.globalTransformMatrix = this_matrix;

// if this node have model its model matrix GPU state should be updated too
if (StaticModelComponent* static_model_component = ent.
tryGetComponent<StaticModelComponent>())
{
static_model_component->need_GPU_state_update = true;
}

// if this node is bone its transform matrix GPU state
// should be updated too
if (BoneComponent* bone = ent.tryGetComponent<BoneComponent>())
{
bone->need_gpu_state_update = true;
}

// further traverse tree up util the leaves
const RelationshipComponent& ent_rc = ent.
getComponent<RelationshipComponent>();
entity cur_child = ent_rc.first;
while (cur_child)
{
editedTraverseTree(cur_child, this_matrix, frame);
```

```
cur_child = cur_child.getComponent<RelationshipComponent>().next;
}
}

/**
* \brief record commands to draw all objects in render queues
* \param frame current frame index to bind proper descriptor sets
* \param command_buffer command buffer record commands to
*/
void UnlitTexturedShaderEffect::draw(int frame, const
vk::raii::CommandBuffer& command_buffer) override
{
// draw all in static queue
drawStatic(frame, command_buffer, per_object_data_buffer_);
// draw all in skeletal queue
drawSkeletal(frame, command_buffer, per_skeleton_data_);
}

/**
* \brief draw all in static queue
* \param frame current frame index to bind proper descriptor sets
* \param command_buffer command buffer record commands to
* \param per_renderable_data_buffer ref to Per Static Model Data to bind
*/
void UnlitTexturedShaderEffect::drawStatic(int frame, const
vk::raii::CommandBuffer& command_buffer,
constPerStaticModelData& per_renderable_data_buffer)
{
// ptr of previously bound mesh and material
const Mesh::MeshRenderData* prev_mesh=nullptr;
const DiffusionMaterial* prev_mat = nullptr;

// bind shader effect data
un_lit_graphics_pipeline_statics_.bindStaticPipeline(command_buffer);
un_lit_graphics_pipeline_statics_.bindStaticShaderData(frame,
command_buffer);

// linearly iterate all objects in queue
for(auto& [mesh, offset]: static_render_queue){

// if mesh doesn't change no need to rebind it
if(mesh->render_data_!=prev_mesh)
{
mesh->bind(command_buffer);
prev_mesh = mesh->render_data_;
}


// if material doesn't change no need to rebind it
if(mesh->material_!=prev_mat)
{
mesh->material_->bindMaterialData(frame, command_buffer,
*un_lit_graphics_pipeline_statics_.static_pipeline_layout_);
prev_mat=mesh->material_;
}

// bind per object data with given offset
per_renderable_data_buffer.bindDataFor(frame, command_buffer,
*un_lit_graphics_pipeline_statics_.static_pipeline_layout_, offset);

// issue draw command
mesh->drawIndexed(command_buffer);
```

```
}

// clear queue for next frame
static_render_queue.clear();
}
```

APPENDIX B
Source code for shader programs

When creating graphics pipelines for shader effects, two types of shader programs need to be specifiedvertex and fragment.Shader programs are written in the languageGLSLand then interpreted intoSPIR-V.For the two types of graphical objects described in the paper: static models and skeletal models, two different vertex shaders are required, let's look at them.

Static model vertex shader:

```
#version 450
// viewport camera render data
layout(binding = 0) uniform UniformBufferObject {
mat4view;
mat4proj;
}ubo;

// per static model data
layout(set = 3, binding=0) uniform DynamicUBO
{
mat4 model;
}dubo;

// model position of vertex
layout(location = 0) in vec3 inPosition;
// normal of vertex
layout(location = 1) in vec3 inNormal;
// UV texture coordinate
layout(location = 2) in vec2 inTexCoord;

// out texture coord for fragment shader
layout(location = 0) out vec2 fragTexCoord;
// out transformed normal
layout(location = 1) out vec3 outNormal;

void main() {
// apply camera tra transformations and model matrix
gl_Position = ubo.proj * ubo.view * dubo.model * vec4(inPosition, 1.0);
// pass texture coordinates
fragTexCoord = inTexCoord;
// transform normal with inverse transpose matrix
// http://www.lighthouse3d.com/tutorials/
// glsl-12-tutorial/the-normal-matrix/
outNormal = normalize(mat3(transpose(inverse(dubo.model))) *inNormal);
}
```

Skeleton vertex shader:

```
#version 450
// viewport camera render data
layout(binding = 0) uniform UniformBufferObject {
mat4view;
mat4proj;
}ubo;

// skeletal constants MUST BE IN SYNC with same in dmbrn::BonedVertex
const int MAX_BONES = 256;
const int MAX_BONE_INFLUENCE = 4;

// per skeletal model data
layout(set = 3, binding=0) uniform DynamicSkelUBO
{
mat4[256] finalBonesMatrices;
}skel_dubo;


// model position of vertex
layout(location = 0) in vec3 inPosition;
// normal of vertex
layout(location = 1) in vec3 inNormal;
// UV texture coordinate
layout(location = 2) in vec2 inTexCoord;
// bone indexes of bones influence this from finalBonesMatrices
layout(location = 3) in uvec4 inBoneIDs;
// weights of influences bones
layout(location = 4) in vec4 inBoneWeights;

// out texture coord for fragment shader
layout(location = 0) out vec2 fragTexCoord;
// out transformed normal
layout(location = 1) out vec3 outNormal;

void main()
{
// accumalate all influences bone transformaton with weights
mat4 boneTransform = skel_dubo.finalBonesMatrices[inBoneIDs[0]] *
inBoneWeights[0];
boneTransform+= skel_dubo.finalBonesMatrices[inBoneIDs[1]] *
inBoneWeights[1];
boneTransform+= skel_dubo.finalBonesMatrices[inBoneIDs[2]] *
inBoneWeights[2];
boneTransform+= skel_dubo.finalBonesMatrices[inBoneIDs[3]] *
inBoneWeights[3];

// apply camera transformations and model matrix
gl_Position = ubo.proj * ubo.view*boneTransform*
vec4(inPosition, 1.0);
// pass texture coords to fragment shader
fragTexCoord = inTexCoord;
// transform normal with inverse transpose matrix
// http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/
// the-normal-matrix/
outNormal = normalize(mat3(transpose(inverse(boneTransform)))*inNormal);
}
```

During normal rendering of both static and skeletal models, the same fragment shader is used, the code of which is given below:

```
#version 450
// simple white light simulation with light from up to down
const vec3 light_dir = vec3(0,0,-1);
const vec4 lightColor = vec4(1.0,1.0,1.0,1.0);

// shader effect data
layout(set=1,binding=0) uniform UnLitTexturedUBO
{
// gamma correction
float gamma;
}ult;

// model diffuse texture
layout(set=2, binding = 0) uniform sampler2D texSampler;

// material simple properties
layout(set=2, binding = 1)
{
vec4 base_color;
}properties;

// texture coord from vertex shader
layout(location = 0) in vec2 fragTexCoord;
// transformed vertex normal
layout(location = 1) in vec3 inNormal;

// output color for this pixel
layout(location = 0) out vec4 outColor;

void main()
{
// phong shading https://en.wikipedia.org/wiki/Phong_shading
// full ambient strength
const float ambientStrength = 1.0;
vec4 ambient = ambientStrength * lightColor;

// calculate diffuse reflection as dot between
// nominal and direction*towards*light
float diff = 1.0*max(dot(inNormal,-light_dir),0.0);
vec4 diffuse = diff * lightColor;

// collect all with average balance between ambient and diffuse
outColor = (ambient + diffuse)*0.5*properties.base_color *
texture(texSampler, fragTexCoord);

// add gamma correction to color
const float gamma = 2.2;
outColor.rgb = pow(outColor.rgb, vec3(1.0/gamma));
}
```

When drawing the outline of an object, shaders similar to the previously described shaders are used, except that they performutadditional scaling of the object. To demonstrate this approach, let's print only the vertex shader code for the static model:

```
#version 450
// construct sacle matrix with given scale
#define scaleMat(scale)
mat4(vec4(scale,0,0,0),vec4(0,scale,0,0),vec4(0,0,scale,0),vec4(0,0,0,1 ))

// viewport camera render data
layout(binding = 0) uniform UniformBufferObject {
mat4view;
mat4proj;
}ubo;

// outline shader effect data
layout(set = 1, binding=0) uniform OutlineData{
vec3 color;
float scale;
} outline;

// per static object data
layout(set = 3, binding=0) uniform DynamicUBO
{
mat4 model;
}dubo;

// model position of vertex
layout(location = 0) in vec3 inPosition;

void main()
{
// apply camera transform than model than transform for outline
// this order gives better results IMHO, but outline can be putted
//in different places
gl_Position=ubo.proj * ubo.view*dubo.model*
scaleMat(outline. scale) * vec4(inPosition, 1.0);
}
```

The fragment shader for the path simply outputs the specified path color as the pixel color.