A Conversion of 3D Graphics from Blender to Unreal Engine

Pavel Pokorný[1] and Miroslav Zapletal[2]

[1] Department of Computer and Communication Systems
Tomas Bata University in Zlín, Faculty of Applied Informatics
Nad Stráněmi 4511, 760 05 Zlín, Czech Republic
pokorny@utb.cz

[2] Department of Computer and Communication Systems
Tomas Bata University in Zlín, Faculty of Applied Informatics
Nad Stráněmi 4511, 760 05 Zlín, Czech Republic
m_zapletal@utb.cz

Abstract. This paper describes the conversion possibilities between Blender and Unreal Engine. Although it may seem at first sight to be a simple issue, the opposite is true. Unlike 2D graphics, where a simple image is transferred from one graphics program to another, the problem is much more complex in 3D graphics. We want to transfer only the simple geometry of 3D objects very rarely. Blender is a tool for the entire 3D graphics pipeline - offers several types of representations, assigning materials and textures to objects, creating simple and more complex animations, various physical simulations, including collision detection (particle systems, fabric objects, flexible objects, liquids), etc. Unreal Engine is a state-of-the-art real-time engine and editor that features photorealistic rendering, dynamic physics and effects, lifelike animation, robust data translation, and much more. Main task of this article is to summarize the results of research aimed at finding out what can be converted between these programs and in what graphic formats.

Keywords: 3D Graphics, Graphic Formats, Conversion.

1 Introduction

Computer graphics is the branch of computer science that deals with generating images with the aid of computers. Almost most computers include screen displays, where the screen image is composed of discrete units called pixels [1], images can have various representation in the memory. Depending on this representation, it is also possible to save them to disk in various graphic formats. In addition to storing all graphic information, these formats are often required to be able to work with other applications than the one that created it.

According to the representation of the image, there are several types of division [2]. The first group is Two-Dimensional Graphics. This includes Raster Graphics as well as

Vector Graphics. Raster (Bitmap) Graphics is the more common representation form. A bitmap represents an image via a rectangular grid of pixels, where each individual pixel's spatial location and colour is defined [3]. Vector Graphics represents an image mathematically by means of the use of geometrical primitives like points, lines, curves and polygons. In a sense, one may consider a vector image as stored information regarding the shapes in an image rather than the raw image itself.

Another category is Three-Dimensional Graphics. 3D Graphics, unlike 2D Graphics, use three-dimensional representation of geometric data. This geometric data is often referred to as 3D models. A 3D model is a mathematical representation of any three-dimensional object [4]; a model is not technically a graphic until it is displayed. Apart from a model's geometry, a lot of other information - like material properties, surface textures of animations, is often stored in memory. If we want to save all this information correctly into a file and open and process it correctly in another programme, it is then necessary to choose a suitable graphic format.

This paper describes the conversion possibilities between 3D Blender and Unreal Engine programmes. The most commonly-used graphical formats for the conversion process between these programmes are tested with respect to the amount of graphical information and techniques that Blender and Unreal Engine support. The second chapter describes the basic features of both programmes. The content of the third chapter is the characteristics of graphic objects and other features that could be converted between Blender and Unreal Engine. The fourth chapter contains a short description of the 3D Graphics formats that were tested during the conversion process. The results and experiences learned from the conversion study are summarised in the fifth chapter.

2 Software Descriptions

In this chapter, we will present the basic characteristics of the Blender and Unreal Engine programmes. We will focus on the main features and capabilities of both programmes, as well as working with files, since this is important given the content of this paper.

2.1 Blender

Blender [5] is a free and open source 3D creation suite, which is available under GNU GPL License and users can use it on most common platforms – Windows, Linux, Mac OS, etc. It supports the entirety of the 3D pipeline — modelling, animation, rendering, rigging, simulation, compositing and motion tracking, video-editing and a 2D animation pipeline. This software is available under GNU GPL License and users can use it on the most commonly available platforms – Windows, Linux, Mac OS, etc.

Blender opens a world of creativity that has been traditionally exclusive to those able to afford high-end graphics software. Its GUI is an arrangement of windows and panels containing the controls for operating the programme [6]. Blender also uses workspaces that are, essentially, predefined window layouts in order to adapt user interfaces for different tasks like modelling, animating, and scripting.

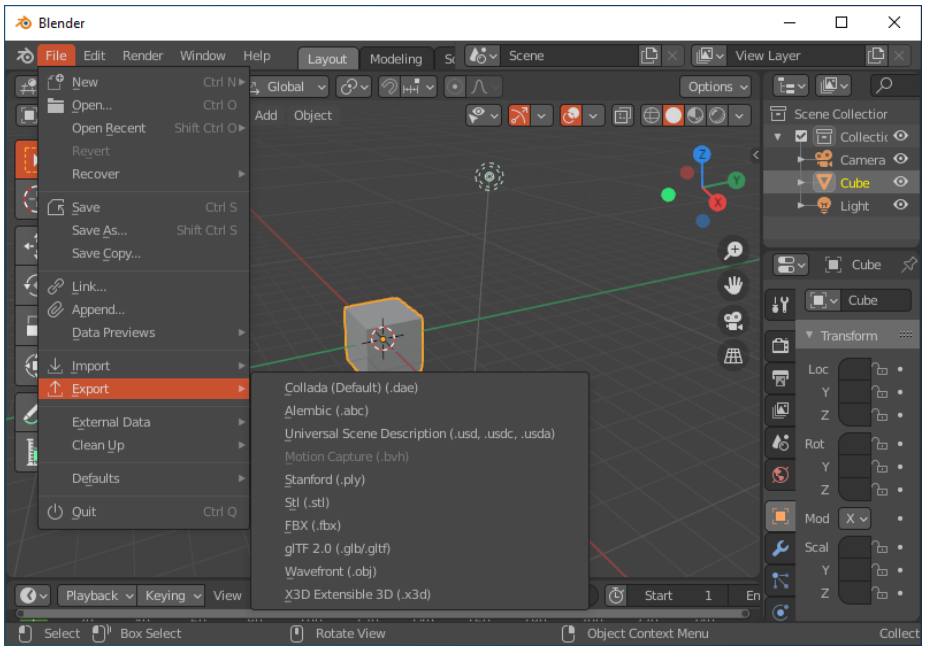

Fig. 1. The Blender User Interface with File-Export Menu

Since Blender is a very complex programme with extensive possibilities, it keeps everything in its database [7]. This database thus contains all scenes, objects, meshes, textures, etc., and when a user saves their current project, the whole database is saved into one .blend file. Although this format is "open" - (its concept is documented), there are not many applications that can work directly with this format. So, if one ever wants to transfer their work from Blender to another programme one will need a different format. The File - Export menu is used for this purpose.

Fig.1 shows the default content of this menu. There are 10 different file formats that can keep 3D Graphics (.dae, .abc, .usd, .bhv, .ply, .stl, .fbx, .glb/.gltf, .obj and .x3d). However, this menu does not include all supported formats. Other types of formats can be added to this menu in the form of extension plugins created in Python. Many such plugins can be found on the Internet - or can be created by the user.

2.2 Unreal Engine

Unreal Engine [8] is the world's most open and advanced real-time 3D creation tool. Continuously evolving to serve not only its original purpose as a state-of-the-art game engine, today it gives creators across industries the freedom and control to deliver cutting-edge content, interactive experiences, and immersive virtual worlds.

Unreal Engine is a complete suite of development tools for anyone working with real-time technology. From design visualisations and cinematic experiences to high-quality games across PC, console, mobile, VR, and AR, Unreal Engine provides one with everything they need to create very high-quality applications.

This engine with its own editor, offer complex tools for photo-realistic rendering, dynamic physics and effects, life-like animation, robust data translation and much more [9]. It performs under two licenses. For creators – this license is free to use and 100% royalty-free; and for publishing – this license is free to use and incurs 5% royalties when a user monetises a game/application developed in Unreal Engine and their life-time gross revenues from that product exceed $1,000,000 USD.

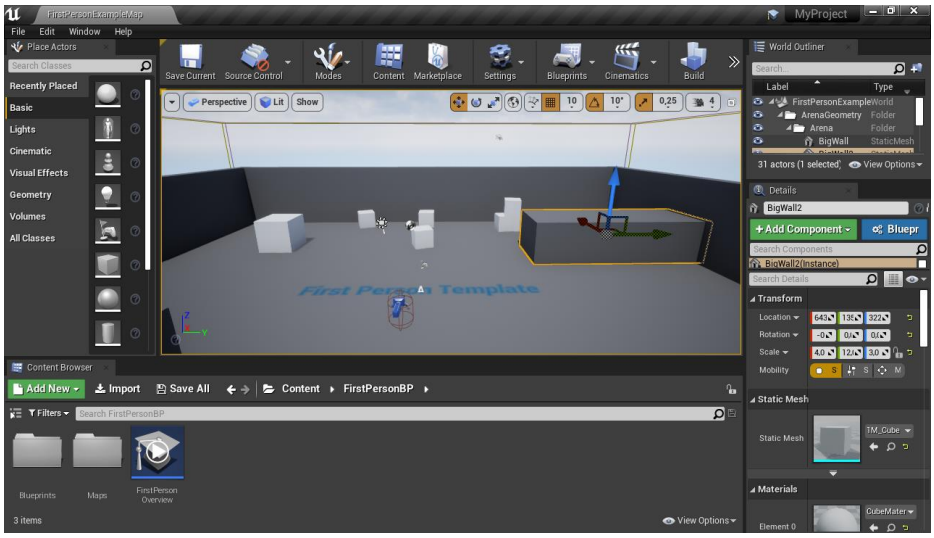

Fig. 2. The Unreal Engine User Interface

3D graphics are added to projects in the asset form in Unreal Engine. An Asset is a piece of content for an Unreal Engine project and can form a variety of resources - 2D images/animations, 3D models, materials, textures, blueprints, sound effects, and visual effects, etc. [10]. In order to include them in a project, Unreal Engine supports a huge number of different file formats. In 3D graphics these formats are mainly supported: .abc, .fbx, and .obj, in basic. Using various plug-ins, Unreal Engine can read other formats like .glb/.gltf or .usd.

3 3D Graphics for Conversion

To determine the correct conversion of various graphical entities between Blender and Unreal Engine, an effort was made to use a wide range of these entities (so-called Data-Blocks), that Blender offers. These are mainly Static Mesh Objects, Particle Systems, Materials and Textures, UV Maps, Character Animations and Physics [7]. These will be described in more detail in this chapter.

- **Static Mesh Objects** – the most common type of objects. Mesh Objects are composed of vertices, edges and polygonal faces which form a geometric shape and can be edited extensively with Blender's mesh editing tools. Mesh

Modelling typically begins with a Mesh Primitive shape (e.g. plane, circle, cube, cylinder, sphere…). From there, users might begin editing and use a large number of modelling tools to create larger, more complex shapes. This type of representation is very suitable for rendering, therefore other types of representations are often converted into mesh objects. [11]

- **Particle Systems** – Particles are lots of items emitted from mesh objects - typically in the thousands. Each particle can be a point of light or a mesh, and be static or dynamic. They may also react to many different influences and forces, and have the notion of a lifespan. Static type particles are joined together and form curves that can represent hair, fur, grass and bristles. Dynamic particles can represent fire, smoke, mist, and other things like dust or magic spells. [7]
- **Materials and Textures** – Materials control the appearance of meshes, curves, volumes and other objects. They define the substance that the object is made of, its colour and texture, and how light interacts with it. Physically-based materials are typically created using the Principled BSDF, Principled Hair or Principled Volume shaders in Blender. By using this wide variety of materials, many types of materials can be created - including plastic, glass, skin, metal, cloth, hair, smoke or fire. Textures can be procedurally generated or standard raster images read from disk.
- **UV Maps** – UV maps define UV coordinates in the image textures in order to attain precise mapping of these textures on the object surfaces. Coordinates can be computer generated or manually created in Blender. It is also necessary to export UV coordinates when one wants to open models created with textures in other applications, otherwise the texture will not be applied correctly.
- **Character Animations** – This is a specialised field of the animation process where models of characters have animation that bring them to life. Blender offers a complete animation toolset - specialised Character Animation Pose Editor, Non-linear Animation Editor, Forward / Inverse Kinematics for fast poses and Sound Synchronisation. Riggins Tools include envelope skeleton and automatic skinning, easy weight painting, mirror functionality, bone layers and coloured groups for organisation and B-spline interpolated bones.
- **Physics** – Blender's Physics System allows one to simulate and animate a number of different real-world physical phenomena. These include Rigid Body Animations, Cloth Systems, Soft Body Systems, Fluid Systems, Forces and Collisions. Since we tried to convert Rigid Body Animations and Fluid Systems, they are described in more detail here.
- **Rigid Body** – These animations can be used to simulate the motion of solid objects. They affect the position and orientation of objects but do not deform them. Rigid bodies can be used like regular objects and be part of parent-child relationships, animation constraints and drivers. There are two types of rigid bodies: active and passive. Active bodies are dynamically simulated, while passive bodies remain static. An example of a Rigid Body Animation frame is shown in Figure 3 - a ball on a chain breaks down stacked cubes.

- **Fluid Systems** – These objects are used to simulate the physical properties of liquids - especially water. While creating a scene in Blender, certain objects can be marked to become a part of the fluid simulation. For a fluid simulation, a user has to have a domain to define the space where the whole simulation proceeds. In the domain settings, users will be able to define the global simulation parameters - like viscosity and gravity. Based on defined objects and set parameters, Blender can calculate the course of the entire simulation.

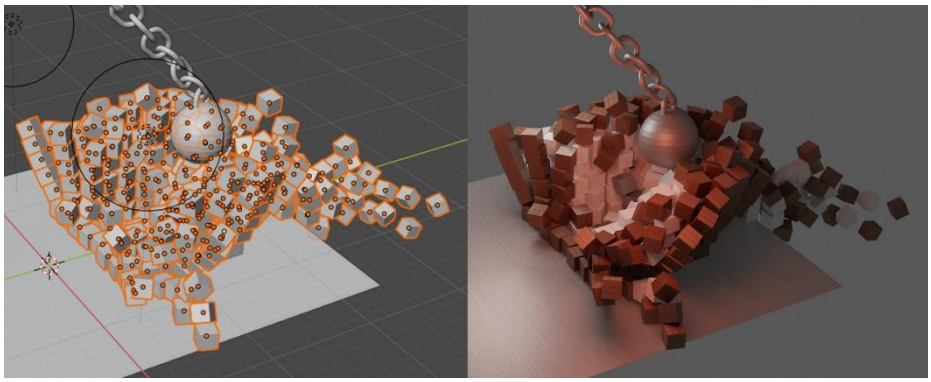

Fig. 3. An Example of Rigid Body Animations in Blender

4 Conversion Formats

Based on an analysis of the export possibilities from Blender and their import to Unreal Engine, the following formats were selected for conversion: .fbx, .abc, .obj, .glb/gltf and .usd. There is also an extension plug-in for Blender called "Blender for Unreal Engine", which specialises directly in this conversion, so this was also included in the testing process. These are described in detail in this chapter.

4.1 The .fbx Format

FBX by Autodesk is a proprietary file format designed to facilitate higher-fidelity data exchange between 3ds Max, Maya, MotionBuilder, Mudbox and other propriety and third-party software. The main features it has are single-step inter-operability between the above-mentioned software, easier data exchange, more efficient workflow as well as customisability. [12]

Since 2006, an Exporter for this format has been implemented in Blender. Since then, it has undergone intensive development - thanks to which exports to this format are among the most sophisticated [13]. If a user chooses to export into this format in Blender, a window with a number of settings will open - (Fig. 4 - Right). Users can select which objects will be exported - (mesh objects can be exported with Modifiers and normal vectors), scale, orientation, animation, object groups - (Batch Export), etc.

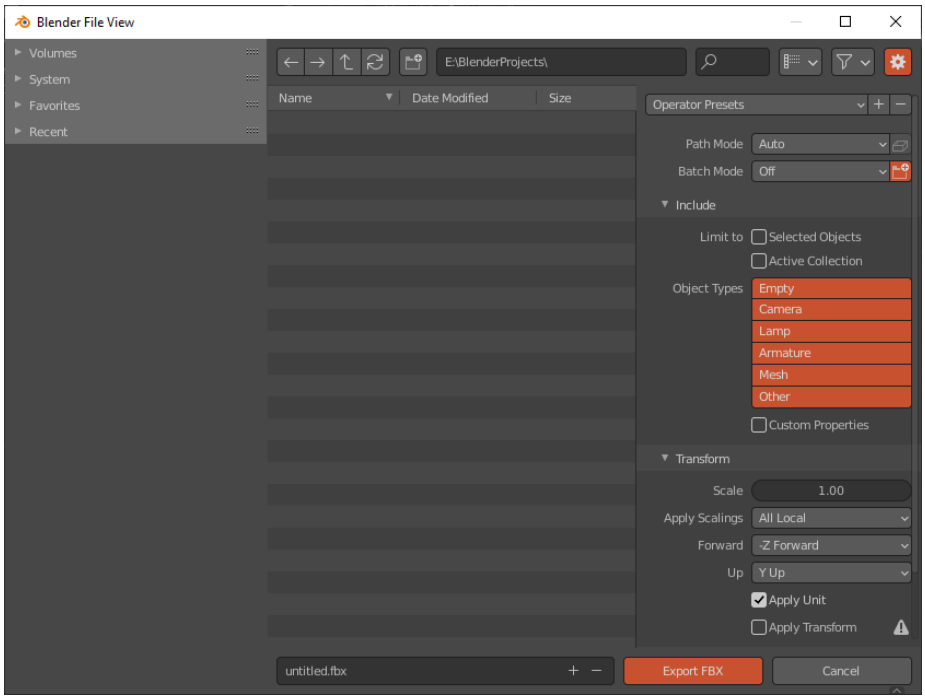

Fig. 4. An Export Window to .fbx file with Adjustable Options in Blender

4.2 The .abc Format

Alembic represents an open computer graphics interchange framework [14]. This is focused on efficiently storing the computed results of complex procedural geometric constructions - that are then saved into .abc files. Alembic is very specifically NOT concerned with storing the complex dependency graphs of procedural tools that are used to create the computed results. For example, Alembic will efficiently store the animated vertex positions and animated transformations that result from an arbitrarily complex animation and simulation process that could involve enveloping, corrective shapes, volume-preserving simulations, cloth and flesh simulations, and so on.

When exporting models from scenes to *.abc file in Blender, users can set different export options like scale, frame start and end - (in animations), use instancing, keeping the relationships between objects - (parent/child), export normal, uv maps, etc. [15]

4.3 The .obj Format

Wavefront OBJ (Object) files are formats that contain geometric definitions. It was designed for Wavefront's Advanced Visualizer application to store geometric objects composed of lines, polygons, and free-form curves and surfaces. However, this file format is open and has been adopted by other 3D graphics application vendors [16].

Wavefront is best known for its high-end computer graphics tools, including modelling, animation, and image composition tools. These programmes run on powerful workstations - such as those made by Silicon Graphics, Inc.

The .obj format is a popular plain-text format; however, it has only basic geometry and material support in Blender. There is also support for object groups and Nurbs curves and surfaces. According to the official Blender documentation [17], there is no support for mesh vertex colours, armatures, animation, lights, cameras, empty objects, parenting, or transformations. An export window contains only a few export options in Blender - like global axis orientations, fixed size, or keeping the order of vertices.

4.4 The .glb/.gltf Formats

The glTF™ (GL Transmission Format) is a royalty-free specification for the efficient transmission and loading of 3D scenes and models by engines and applications that minimise the size of 3D assets - and the runtime processing needed to unpack and use them [18]. This format is commonly used on the web, and has support in various 3D engines like Unity3D, Unreal Engine 4, and Godot. There are two format versions – glTF Binary (.glb) - where all data is stored in a single binary file; and glTF Separate (.gltf) - which is text-based and describes the overall structure; this text file follows a .bin file that contains mesh and vector data - and optionally, a number of .png or .jpg files containing image textures referenced by the .gltf file.

The official information [19] says that Blender's importer / exporter supports the following .glb/.gltf features: mesh objects, simple materials and shadeless textures, cameras, punctual lights - (point, spot, and directional), and simple animations - (keyframe, shape key, and skinning). Export options contain properties like merge vertices, pack images, shading type - (flat, smooth) or heuristic for placing bones in Blender.

4.5 The .usd Format

Universal Scene Description (USD) is a framework for the interchange of 3D computer graphics data, created by Pixar [20]. Each of many cooperating applications in the pipeline - (modelling, shading, animation, lighting, fx, rendering), typically these have their own special form of scene description tailored to the specific needs and workflows of the application, and neither readable nor editable by any other application. Universal Scene Description (USD) is the first publicly available software that addresses the need to robustly and scalable interchange and augment arbitrary 3D scenes that may be composed from many elemental assets.

Blender .usd exporter can save static, deforming or arbitrarily animated meshes, curves, meta-balls, cameras, lights - (except area lights), and static particles - (exported as curves). [21] Exporting options include choices for export selection only - animation, static particles, UV maps, normal and materials.

4.6 Blender for Unreal Engine Plug-ins

This is not an export format, but add-on module that is not implemented in Blender in default. It must be downloaded from [22] and installed in the Blender suite. It simplifies

the method of exporting from Blender to Unreal Engine 4 by allowing one to export all the assets of a scene at the same time. It even automatically distributes them in a proper tree structure in correlation with the Unreal Engine 4 pipeline.

This plug-in does not use its own format to save - but uses third–party 3D formats. Its algorithm recognises which format is suitable for each graphic entity in Blender and automatically offers to save them. It prefers the .fbx and .abc file formats, but there is the possibility to save in multiple formats at once. Its export options are very similar to corresponding file format options.

Fig. 5. A 3D Scene in Blender Created for the Conversion Process

5 Conversion and Results

This chapter describes the examination of the conversion process between these two programmes. It should be noted that the results below were obtained in Blender version 2.90 and Unreal Engine version 4.25.3. These results may not apply to other versions of these programmes.

It's also a good idea to follow certain rules when converting. The first is that Blender and Unreal Engine use different coordinate systems. Although the z-axis points upwards in both programs, the y-axis has opposite directions in Blender and the Unreal Engine. If a user does not set the correct orientation in the export setting, objects can be imported mirrored in Unreal Engine. It is also good to apply transformations in Blender - (a command in the menu Object-Apply-All Transforms), and to set object origins to meaningful positions. In case a user wants to transform their objects into Unreal Engine after the conversion process, these actions prevent the user from seeing weird transformation numbers. The last thing is to control that all faces´ normal vectors point outwards. In Edit mode, it is possible to do that by selecting all faces and then executing the command Mesh-Normals-Recalculate Outside. The reason for doing so

is to do with wrong normal orientations, the textures would be mapped on the opposite side of faces. We did all of these for our objects before the converting process started.

In order to find out the conversion possibilities, a comprehensive model of the Blender programme environment was created - this was a 3D street environment scene (Fig. 5). The basis of this scene are static mesh objects - terrain, roads, houses, fences, benches, sidewalks and a car. Some of these models were modelled by hand in Blender, while others were already inserted into the scene finished from the BlendSwap model collection [24]. All of these models have standard material settings with UV mapped textures that were downloaded from [25].

Particle systems were implemented in the scene in two forms. Trees were randomly generated as particles - (the Emitor particle system), and the grass was generated as narrow polygons representing clumps of them (the Hair Particle system) - corresponding materials have been set for them.

In order to have a character animation in the scene, a human 3D model was used. The core of this model are armature objects - (composed from Bone objects), which simulate bones in the human body. These armatures are linked to a human mesh model. Manipulation with these armatures causes the animation of corresponding parts of the mesh model.

As mentioned above, Rigid Body animations are often used to simulate the motion of solid objects including collision among them. To test this in the conversion process, a simple simulation using Rigid Body was created. This included a demolition ball suspended on a chain model. As the ball moves, it hits the wall composed of simple cubes. Upon this impact, cubes fly into space - (Figure 3).

The last type of object added to the scene is a fluid system. A large container object was created into which fluid flows. There is also one cube-shaped obstacle in this container. The simulation algorithm can calculate the behaviour of this fluid in individual frames of the animation.

Table 1. The Conversion Process Results from Blender to Unreal Engine

Graphical Entities	.fbx	.abc	.obj	.glb/.gltf	.usd	Blender for Unreal
Static Meshes	Yes	Partial	Partial	Partial	Yes	Partial
Materials and textures	Partial	No	Partial	Yes	Partial	No
UV Maps	Yes	Partial	Yes	Yes	Yes	No
Character Animations	Yes	Partial	Yes	Partial	Yes	Partial
Rigid Body	No	No	No	No	No	No
Fluid Simulations	No	Yes	No	No	Partial	No

The scene was created in this way, and then the conversion process began. All types of objects mentioned above - as well as 3D graphic formats described in Chapter 4 were used. The final results are listed in Table 1. Particle Systems are not mentioned in this table, because they were converted to mesh objects in the conversion process.

The "Yes" value means that the graphics entity was displayed (in the case of animations or simulations of how they also behaved) in Unreal Engine - exactly in the same way as in Blender. The "No" value says that the entity failed to load in Unreal Engine. The "Partial" value means that the entity was successfully loaded, but was wrongly displayed - or behaved incorrectly in the Unreal Engine environment. In the case of mesh objects, this was most often caused by incorrect transformations, materials being set inaccurately and textures incorrectly mapped. The character animations and fluid simulations were about their incorrect functionalities. Rigid Body animations could not be successfully converted using any of the listed graphic formats.

The test results show that the .fbx format is the most suitable for converting static objects - including UV Maps and character animations. Its biggest advantage is retaining the positions of many objects in the 3D scene. However, the .fbx format has problems with the correct conversion of material properties – their appearance is slightly different between Blender and Unreal Engine. The .glb/.gltf formats are the most suitable for the correct transfer of material settings. The only format that can correctly convert Fluid Simulations is .abc.

The use of the Blender plug-in the "Blender for Unreal Engine 4" has also proved successful since it facilitates large scenes with a large number of objects correctly. This plug-in allows the user to convert all Blender entities into multiple files at once - the entire conversion process can be then significantly accelerated, especially for big 3D graphics projects.

6 Conclusion

The main goal of this paper was to describe the possibilities and ways of converting 3D graphics from Blender to Unreal Engine. These are both world-famous well-recognised programs with free use options. Blender performs the entirety of the 3D pipeline — modelling, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and a 2D animation pipeline. Unreal Engine is a state-of-the-art real-time engine and editor that features photo-realistic rendering, dynamic physics and effects, life-like animation, robust data translation, and much more.

During testing, it was found that the .fbx format is the most suitable for converting static objects due to its ability to properly maintain the distribution of objects in the scene. The .glb/.gltf formats are the most suitable for converting material properties. The .abc format is the most suitable for fluid simulations. At the same time, the "Blender for Unreal Engine" plug-in can be used for more complex environments - which facilitates the conversion of a large number of objects using the .fbx format. The testing process also found that the .usd format is not suitable for converting any of the created models.

This testing was carried out at the beginning of the second half of 2020. Given that both programmes are intensively developed, it can be expected that conversion process possibilities between these programs will continue to improve. Previous developments suggest that the Blender for Unreal plug-in most likely has the greatest potential to improve conversion options between Blender and the Unreal Engine. And there is a new Blender add-on prepared directly by Epic Games - authors of Unreal Engine [23].

References

1. Miano, J.: Compressed Image File Formats: JPEG, PHG, GIF, XBM, 1st Ed. Addison-Wesley Professional, Ltd., Texas (1999). ISBN 978-0201604436.
2. Murray, J.D.: Encyclopedia of Graphics File Formats, 2nd Ed. O'Reilly Media, Sebastopol (1996). ISBN 978-1565921610.
3. Tan, L. K.: Image File Formats. In: Biomedical Imaging and Intervention Journal 2.1 (2006): e6. DOI: 10.2349/biij.2.1.e6
4. Franklin, C.: How 3-D Graphics Work., https://computer.howstuffworks.com/3dgraphics.htm, last accessed 2021/01/18
5. Blender.org – Home of the Blender project, https://www.blender.org/, last accessed 2021/01/18.
6. Blain, J. M.: The Complete Guide to Blender Graphics, 3rd Edition, CRC Press (2016). ISBN 978-1-4987-4647-2.
7. Blender 2.91 Reference Manual, https://docs.blender.org/manual/en/latest/, last accessed 2021/01/18.
8. The most powerful real-time 3D creation platform - Unreal Engine, https://www.unrealengine.com/en-US/, last accessed 2021/01/18.
9. Plowmanm J.: 3D Game Design with Unreal Engine 4 and Blender, 1st Edition, Packt Publishing (2016). ISBN 978-1-78588-146-6.
10. Unreal Engine 4 Documentation, https://docs.unrealengine.com/en-US/index.html, last accessed 2021/01/18.
11. Hughes, J. F., Van Dam, A., Foley, J. D., McGuire, M., Feiner, S. K., & Sklar, D. F.: Computer graphics: principles and practice, 3rd Ed. Addison-Wesley, USA (2014). ISBN 978-0-321-39952-6
12. FBX – Adaptable File Formats For 3D Animation Software, https://www.autodesk.com/products/fbx/overview, last accessed 2021/01/18.
13. Extensions:Py/Scripts/Manual/Export/FBX, https://web.archive.org/web/20090722160843/http://wiki.blender.org/index.php/Extensions:Py/Scripts/Manual/Export/FBX, last accessed 2021/01/18.
14. Alembic, http://www.alembic.io/, last accessed 2021/01/18.
15. Alembic – Blender Manual, https://docs.blender.org/manual/en/latest/files/import_export/alembic.html, last accessed 2021/01/18.
16. Wavefront OBJ: Summary from the Encyclopedia of Graphics Files, https://www.fileformat.info/format/wavefrontobj/egff.htm, last accessed 2021/01/18.
17. Wavefront .obj file – Blender Manual, https://docs.blender.org/manual/en/2.80/addons/io_scene_obj.html, last accessed 2021/01/18.
18. glTF Overview – The Khronos Group Inc, https://www.khronos.org/gltf/, last accessed 2021/01/18.
19. glTF 2.0 – Blender manual, https://docs.blender.org/manual/en/2.80/addons/io_scene_gltf2.html, last accessed 2021/01/18.

20. USD Documentation: Introduction to USD, https://graphics.pixar.com/usd/docs/Introduction-to-USD.html, last accessed 2021/01/18.
21. Universal Scene Desription – Blender Manual, https://docs.blender.org/manual/en/latest/files/import_export/usd.html, last accessed 2021/01/18.
22. GitHub – xavier150/Blender-For-UnrealEngine-Addons, https://docs.blender.org/manual/en/latest/files/import_export/alembic.html, last accessed 2021/01/18.
23. Download our new Blender addons – Unreal Engine, https://www.unrealengine.com/en-US/blog/download-our-new-blender-addons, last accessed 2021/01/18.
24. Blend Swap – Home, https://blendswap.com/, last accessed 2021/01/18.
25. Textures – Textures for 3D, graphic design and Photoshop!, https://www.textures.com/, last accessed 2021/01/18.