Milica Lazor
MSc Student
University of Novi Sad
Faculty of Technical Sciences
Serbia

Dušan B. Gajić
Assistant Professor
University of Novi Sad
Faculty of Technical Sciences
Serbia

Dinu Dragan
Assistant Professor
University of Novi Sad
Faculty of Technical Sciences
Serbia

Alina Duta
Associate Professor
University of Craiova
Faculty of Mechanics
Romania

Automation of the Avatar Animation Process in FBX File Format

Animated three-dimensional (3D) models are an essential part of interactive content available in products of present day movie and gaming industries. Time required for manual development of animated 3D models is a limiting factor in many of their practical applications. In order to minimize the production time of animated 3D models and, thus, lower costs, there is a need for automation of the animation process. This paper presents an approach to the programmatic addition of animations to automatically rigged avatars stored using the industry-standard FBX file format. The presented solution is developed using the C++ programming language and the Autodesk FBX Software Development Kit (SDK). It allows significantly higher degree of automation and programmability than other standard approaches, based on using standard 3D modelling and animation software, such as Autodesk 3Ds Max or Maya.

Keyword: *computer animation, computer graphics, automation, avatars, FBX, C++ programming.*

1. INTRODUCTION

Present day state-of-the-art 3D computer graphics and animation software [1], such as Autodesk 3Ds Max [2], Autodesk Maya [3], Blender [4], and the Adobe Mixamo web platform [5], can support a wide variety of animations. Using these software environments is rather easy and intuitive. However, these software tools require manual guidance (or completely manual work) and only in some cases they provide support for semi-automatic processing [1]. Further, manual approach to 3D animation is very time consuming. For example, in order to create a simple 10-second animation, which is composed of a minimum of 240 to 300 frames (depending on the used video format), at least 6 to 8 hours of animator's work is required, because it typically takes about 30 minutes to manually define each second of an animation [6].

On the other hand, current trends in animation deve–lopment impose furious rhythm where it is important to get an acceptable result in the shortest possible time [7]. The automated process presented in this paper is not aimed at creating high-quality results, but rather at removing the burden of tedious manual work on creation of supporting and background characters in animated movies, computer games, and mobile phone applications. This medium quality level of animations can be achieved fast and relatively easy using the approach reported in the paper. However, certain prerequisites need to be fulfilled in order for the presented approach to be successful—the 3D model must be already rigged, animation set must also be created previously, and the two skeletons (the one with the animation and the one that drives the 3D model) must have the same number and the same hierarchy of bones. In this way, it is possible to add a great number of different animations to the same model in just a few seconds. Further, once animations are created, they can be assigned to different models, as long as they meet the specified conditions.

Created animations could have a variety of uses in different 3D animations and applications. Created models could have supporting roles in applications that use animated 3D models [8] such as different virtual shops [9], simulations [10, 11], 3D printing software [12, 13, 14], etc.

The remainder of the paper is organized as follows. In Section 2, we describe common software tools which computer animation professionals typically use for creating animations. In Section 3, we discuss main properties and functionalities available in the FBX file format. We analyse the tree data structure, used for storing data in the FBX, as well as stacks and layers, used for keeping the animation data. Further, we examine the programming principles of the Autodesk FBX SDK library. In Section 4 we discuss the proposed approach of automatic addition of animations on already existing skeletons. We also present the details of the architecture and implementation of the software solution. Section 5 closes the paper with main conclusions and plans for future work.

2. SOFTWARE FOR COMPUTER ANIMATION

As previously stated, Autodesk 3Ds Max, Autodesk Maya, Blender, and Mixamo are most commonly used 3D animation software. Each of these tools supports creation of animations in different ways. Also, each of them has its own advantages and drawbacks. In this paper, we describe only methods for creation of animations which are similar to our own approach for applying animations to a rigged 3D model.

Received: June 2018, Accepted: December 2018.
Correspondence to: Dinu Dragan, PhD
University of Novi Sad, Faculty of Technical Sciences
Trg D. Obradovića 6, 21000 Novi Sad, Serbia
E-mail: dinud@uns.ac.rs
doi:10.5937/fmet1902398L
© Faculty of Mechanical Engineering, Belgrade. All rights reserved

There are several options when it comes to the addition of animated files to a model in Autodesk 3Ds Max. In order to add an animation to a rigged model using 3Ds Max, the first step is to load the model to the scene. Next step is to load an animated file. If model's skeleton and animated skeleton have the same hierarchy and number of bones, animation should be added to the model instantly. This approach is fast and accurate and requires minimal user effort. However, it cannot be done for addition of multiple animation files to the same model. Also, it is not possible to automate this process to add animation to great number of models. Further, it may be hard to include this approach in the middle of an automated pipeline for 3D avatar creation.

In Autodesk Maya, to add animation from the file to a rigged model, a user must import the rigged model first, select skeleton and then import an animated file. Each animation can be added to a different layer, that can be exported as one file, but there is no support for adding and exporting multiple stacks. Although this approach is fast and simple, it can hardly be used in automatic pipeline as it cannot be scripted and started easily.

The first step in adding an existing animation to the rigged model using Blender is to import model with skeleton. The next step is to add animated skeleton to the same scene. User has to select an animated skeleton and then choose model's skeleton. When both are selected, user can proceed to the retargeting phase. Complexity of this phase depends on the skeleton adequacy, namespaces, etc. Hierarchy mapping will be automatically approximated, and if successful, animation will be transferred in a few mouse clicks. Unfortunately, the hierarchy mapping approximation is not always accurate, and manual adjustment is needed. In Blender it is possible to store and export multiple animations in the same FBX file. Blender scripting works very well and this process can be automated and included in the automatic processing pipeline.

Adding animation to an already rigged model is quite easy and simple using the Adobe Mixamo web platform. All that user needs to do is to upload the model and choose the desired animation. Mixamo has a huge base of different animations which can be added to almost any 3D model. Significant advantage of this approach is that number of bones does not really matter. The Mixamo algorithm for retargeting works very well and as long as model's skeleton is humanoid, animations look fine. User can select more than one animation, but these will be downloaded as separate files. There is no option to save multiple animations in one file. Another disadvantage of this approach is the need for Internet connection. Integration of Mixamo into an automatic pipeline is not possible, because it is a third-party solution that requires model upload and does not support workflow customization/scripting.

3. FBX FILE FORMAT AND AUTODESK FBX SDK

FBX (FilmBoX) file format [15] is one of the main 3D exchange formats. It is used by almost all 3D graphics and animation software, such as 3Ds Max, Maya, Blender, etc. This format enables the transfer of 3D objects, lights, cameras, and materials, as well as animations. The FBX file format has two different representations: binary and ASCII. Since the main goal of the research presented in this paper is storing and displaying an animated character on the scene, the FBX file format was a natural choice.

The Autodesk FBX SDK (Software Development Kit) [16], used in the development of the presented approach, is based on the C++ programming language. It allows users to create plug-ins, parts, or even whole software solutions based on the FBX technology.

3.1 Scene Organization

The scene within the FBX SDK is organized and presented as hierarchy of nodes (see Fig. 1). Exact position of each element on the scene is defined by translation, rotation, and scaling operations. The FBX Node stores information about the node geometric transformations, while additional information, such as object type, is stored in the *FbxNodeAttribute* subclass. E.g., for a 3D model *FbxMesh* subclass is used.

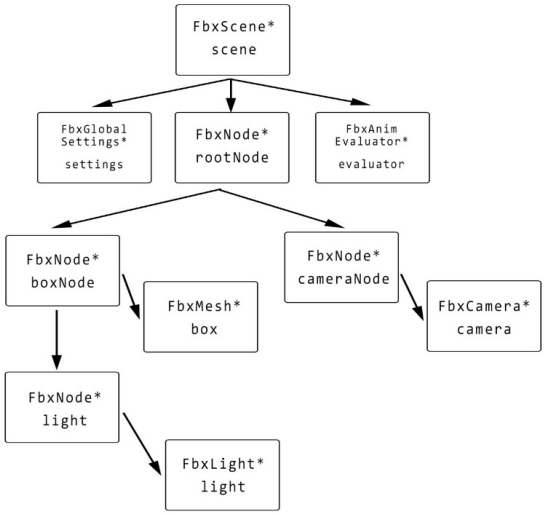

Figure 1. FBX Scene Graph

The nodes within the scene are organized hierar–chically in such a way that position, rotation, and sca–ling of each node is in relation to its parent coordinate system. The order in which transformations are applied is defined by inherited type of the node. Transformation information of each node includes its scaling, trans–lation, and rotation vectors, taking into account its pa–rent transformation information. Each of these vectors contains three variables: X, Y and Z. Vector value is actually the offset in relation to appropriate vector of its parent node. The node default transformation infor–mation is local in relation to its parent node. *FbxTypedProperty* object represents this data, which can be accessed using *LclTranslation*, *LclRotation*, or *LclScaling*, where "lcl" stands for local. The actual value of translation, rotation, and scale vectors in the exact moment depends on the default node values, node limits, constraints, and animation curves of current animation stack. Translation, rotation, and scaling factor of a node, in relation to the global coordinate system of the scene, can be represented in the form of the transformation matrix. It is also possible to evaluate the local transformation matrix.

FbxNodeAttribute subclass is paired with the *FbxNode* in order to define element on the scene with specific position, rotation, and scale. Illustration of rela–tions between a node and its attributes is shown in Fig. 2.

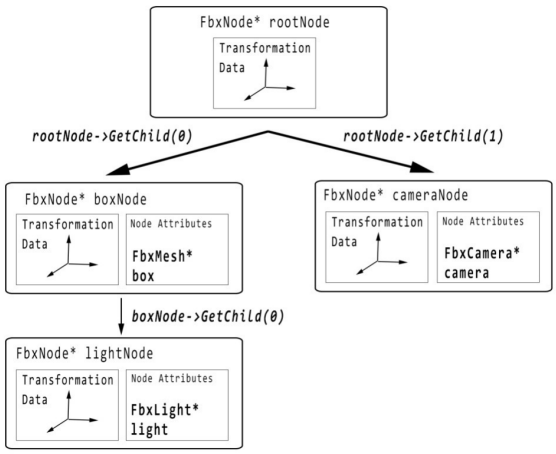

Figure 2. Node information

Since the root node of the scene is not exported, but only the nodes attached to it, it is not recommended to store any of the information in the root node.

3.2 Geometry and Skeleton

3D models used in the presented approach contain certain geometry – a mesh and a skeleton. *FbxGeometry* is the main class for all objects which can be deformed by using control points. Instances of this class can be paired with nodes as node attributes. Classes that inherit from the *FbxGeometry* class are *FbxMesh*, *FbxLine*, *FbxNurb*, etc. Mesh is represented by the *FbxMesh* class. Mesh is defined by the set of control points and layer groups, which define polygon normals, materials, and textures. The *FbxLayerElementNormal* instances is used to define polygon normals. The layer elements can be mapped on the mesh surface by control points, by polygon vertex, by polygon, and by edge.

Another approach is to use the same mapping coordinates across the whole geometry. As assumed, this approach gives the lower quality results when it comes to more complex models. Once we have normal vectors and an array of control points, it is possible to determine how the normal vector is referenced by control points. For example, direct reference mode works by mapping the n-th element from vector normal array to n-th element from control points array. The skeleton inside the 3D model is represented by the *FbxSkeleton* class. Object of this type is a transform node with properties that are useful for visualization of the skeleton. Each skeleton contains chains of *FbxSkeleton* node attributes. For example, an arm can be made from one parent node (root node), followed by children: shoulder, arm, and forearm limb type nodes, respectively, and a hand node as the last node in the chain defined by Effector node attribute type. It is possible to change limb length, orientation, colour, etc.

3.3 Materials and textures

Materials and textures have great significance in making the model as realistic as possible. The *FbxMaterial* class is the main class for applying Lambert and Phong materials. When a material is created, it must be attached to the node which contains the mesh. Once material is paired with the node, it is mapped to the model's surface. *FbxTexture* is the main class repre–senting textures within the FBX SDK. Textures rely on base material, and together with the material, they affect how the geometry looks. Further, materials have dif–ferent channels, such as diffuse, ambient, and emissive. Each of these channels requires a different texture, which can be loaded from file. Another interesting pos–sibility which the FBX SDK provides is to create layered textures. In this way, textures can be loaded one over the other, with different opacity factors, to create mixed texture. It is also possible to load one texture from multiple segments. In order to get a functional and textured FBX file after exporting, it is important to understand how texture is stored inside the FBX file. As output, users will get only the FBX file, but only once it is opened. An FBM folder will be created with texture file in it, and since the FBX file has path to that folder, texture will be loaded correctly.

3.4 Animation Data Structures

The FBX SDK uses the following animation data structures: animation stack, animation layer, animation curve, animation curve node, and animation curve key [16]. Data structures are interconnected by object-object or object-property connection within the FBX scene. Scene can contain one or more animation stacks, or any other animation data structure. If scene or its elements are not animated, there is no need for animation stacks. Animation stack is the high-level container for anima–tion data storage and can contain one or more animation layers. Animation layer contain one or more nodes which are constrained to animation curve. Animation stack must contain at least one layer, but blended animation can be made from two or more layers. Animation curve also known as a function curve defines how are object properties (rotation, translation, scaling) animated in time. Each one of these properties has its own curve for each of the three axes (x, y, z). Animation curves are always attached to animation curve nodes. One animation curve can be attached to more than one animation node, which means that one curve can animate several properties of different models. Animation curves exist only for the objects which are animated. Animation curve node is actually the binding point between animation curves and FBX properties. In order to establish this connection, it is necessary to constrain both subjects to same animation curve node. Animation curve node can be attached to only one property of a single object. Further, the animation curve node must be constrained to the animation layer. If curve node is constrained to multiple layers at the same time, the same value will be used for each of the layers. Animation curve key, also known as keyframe, marks the beginning and the end of an animation curve, or places on the curve where there are some significant changes.

4. THE APPROACH FOR THE AUTOMATIC ADDI–TION OF ANIMATIONS TO AVATARS

The presented programmatic addition of animations to avatars can be divided into six phases (Fig 3.):

1. Pre-animation stage, in which skeleton and animation are created.
2. Creating the main scene and loading model.
3. Accessing the skeleton.
4. Loading animation files.
5. Applying animations.
6. Saving and exporting the scene.

A model which is used is a human mesh with rig and skin. Animations are downloaded from the online service Mixamo [5], but the system is in no way limited to Mixamo animations.

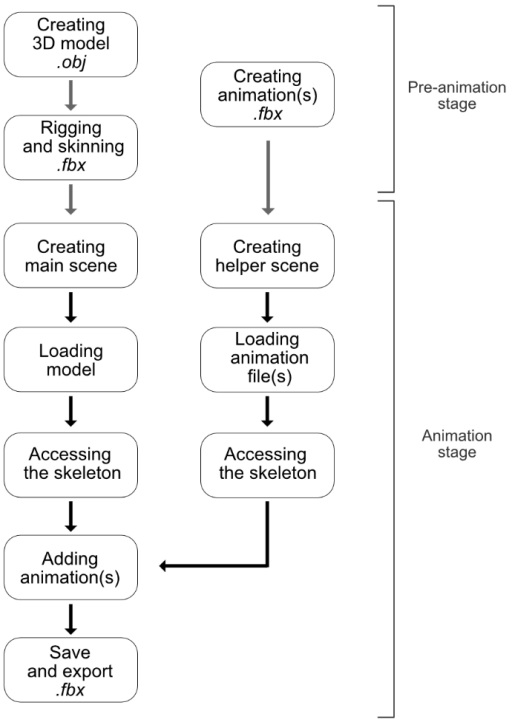

Figure 3. Automated avatar automation workflow

Animations can be taken from any other source, as long as they are in FBX file format. In order to make addition of animations possible, there are several pre–conditions, such as that animated skeleton (reference) and model's skeleton (destination) must have the same number and hierarchy of bones. Orientation and size of the bones must be appropriate. Bones must be repre–sented as bone objects, not helpers or any other. As the final result, users get an animated model situated in a scene. The number of applied animations depends of number of animation stacks, so it is possible to add more than one animation to the model at the same time. Also, it is possible to add cameras and lights to the scene and animate them.

4.1 Animation Data Structures

The scene within the FBX SDK is created by using create method from the *FbxSdk* class. At the very begin–ning, an importer is created. Importer is an object for loading, i.e., importing data. The next step is creating an object which stores main properties and defines them. By default, values of these elements is set to "true", so it is not necessary to explicitly set them, unless some of the properties should not be loaded. After that, an im–porter is initialized with the file-path parameter which contains a path to the desired FBX file. The result of this phase can be seen in Fig. 4.

4.2 Accessing the Skeleton

The first step in the process of applying animations is accessing the skeleton of the model. When the huma–noid skeleton is used, it is necessary to access all the bones from skeleton. This is done using recursion, i.e., calling a method *getBone* from inside the *getBone* method. Moving through the skeleton hierarchy is based on accessing the skeleton root node, and recursively accessing all the children of this node and its children.

Figure 4. OBJ file mesh with the added skeleton

The *getBone* method is described in the C++ code snippet in Fig 5.

4.3 Loading Animation Files

A method for loading of animation files is called for each animation in the FBX file in the folder on the selected path. Since the static model is loaded in the "base" layer/stack on the main scene, animations are added to the main scene as well.

```
node* getBone(node* boneNode, string boneName){
  for(int i=0; i<boneNode->GetChildCount(); i++){
    returnNode = getBone(boneNode->GetChild(i),
                         boneName);
    if (returnNode != NULL) return returnNode;
  }
  return returnNode;
}
```

Figure 5. *getBone* method implementation in C++

Animations cannot be added to the main scene dire–ctly. Therefore, the "helper" scene is created and ani–mations from files are loaded in the layers or stacks in this scene. Afterwards, animation data is moved from the helper scene to the main scene (see Fig 6.).

4.4 Applying Animations

Transferring animations is the most complex step in the whole process of animation addition. Information regarding animations is stored in the animation curves.

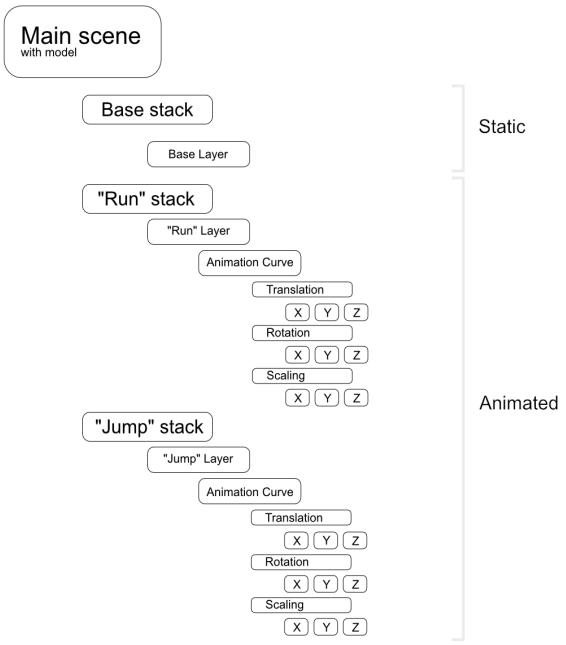

Figure 6. Stack/Layer organization in the Scene

Each animated element must have its own animation curve. First it is necessary to access each bone from the hierarchy of the animation scene, then access the cor–responding animation curve node, and copy the values of rotation, translation and scaling curves for all three axes. Then, these copied values need to be pasted to the corresponding bones of the hierarchy of the model in the static scene. The main idea behind this method is to access all the stacks, then all the layers within the stacks. The next step is to access all nodes of one layer, and eventually access the animation curves which are then copied. The code for creating animation curve for an animated element is described in the C++ source code snippet in Fig. 7.

4.5 Saving and Exporting the Scene

Saving and exporting the scene is equally important as making and transferring animations. In addition to the importer which is created when loading the file, an exporter - an object in charge of storing the scene is also created. Settings are also created and defined, but the values for storing texture and embedded media are expli–citly set to "true". Otherwise, the texture will not be saved along with the scene. Even when these settings are res–pected, the texture can't be saved inside the file. Namely, when an FBX file is opened, a folder with the FBM extension, containing the text, is created auto–matically, while the FBX file stores the name and path to the folder. The output of this last phase is a fully animated model.

5. CONCLUSION

Animated objects and 3D models are a fundamental part of any modern video game or animated movie. Programmatic application of animations is one of the possible ways for optimization and simplification of the whole process. The presented approach enables fast addition of existing animations to the humanoid 3D model using the command line interface (CLI) by simly providing a path to the folder with models and animations. It allows application of multiple animation stacks and layers, which gives users a choice between different currently displayed animations. It is also pos–sible to create a complete scene with lights and cameras and store all of these desired objects in the FBX format. Besides offering significantly lower time requirements and thus costs, the most important advantages of the presented approach are the possibility of storing a multi-animated model with a texture in just a single FBX file, as well as adding the same animation to different 3D models. Another advantage is that it can be used on an as–is basis or as part of a larger automated processing pipeline. Further, it does not require any additional intervention from the user nor application of additional software tools and it can export both binary and ASCII FBX files. A detailed overview of the entire process and code involved in the software solution for animating a human model in the FBX format using the FBX SDK library can be found in [17].

```
void CopyCurves(AnimNode* fromNode, AnimNode* toNode) {
  for (int channel=0; channel<3; channel++){
    Curve newCurve = CreateCurve(toNode→GetName(),
                                 channel);
    if (newCurve != NULL) {
      int keyPointCount =
                    fromNode→getChannelNodes(channel);
      for (int i=0; i<keyPointCount; i++){
        float value =
                  fromNode→GetCurve(channel)→GetKey(i);
        long timePoint =
                 fromNode→GetCurve(channel)→GetTime(i);
        int index = newCurve→InsertKeyPoint(timePoint);
        newCurve→SetKeyValue(index,value);
      }
    }
  }
}
```

Figure 7. *CopyCurves* method implementation in the C++ code

Programmatic addition of animations is an interes–ting area for future research. The solution presented in this paper has several spaces for further improvement. One of the possible improvements could be the integ–ration with motion capture systems. A step further can also be real-time application of animations from motion capture system. As the final goal of the presented re–search, a fully automated pipeline for the support of the whole 3D model creation process – from 2D to 3D conversion to rigging, skinning, texturing, and anima–ting 3D models, can be developed.

ACKNOWLEDGMENT

The authors would like to thank Doob Group AG for the support provided for this research. The reported rese–arch is also partly supported by the Ministry of Educa–tion, Science, and Technological Development of the Republic of Serbia, projects TR32044 (2011-2018), ON174026 (2011-2018), and III44006 (2011-2018).

REFERENCES

[1] Kerlow, I.: *The art of 3D computer animation and effects, 4th edition*, Wiley Publishing, 2009.

[2] Autodesk, 3Ds Max overview. Available at: https: //www.autodesk.com/products/3ds-max/overview (last accessed: 12 November 2018).

[3] Autodesk, Maya overview. Available at: https://www.autodesk.com/products/maya/overview (last accessed: 12 November 2018).

[4] Blender, The official site of the application. Available at: https://www.blender.org (last accessed 12 November 2018).

[5] Adobe, Mixamo. Available at: https://www.mixamo.com (last accessed 12 November 2018).

[6] Riki, J.K.: How fast should you animate? Animator Island, 2013. Available at: https://www.animatorisland.com/how-fast-should-you-animate/?v=8cee5050eeb7 (last accessed 12 November 2018).

[7] Robinson, C.: What is the importance of automation in industries? Quora, 2018. Available at: https://www.quora.com/What-is-the-importance-of-automation-in-industries (last accessed 12 November 2018).

[8] Berdić N., Dragan D., Mihić S., and Anišić Z.: Creation and usage of 3D full body avatars, Annals of Faculty Engineering Hunedoara - International Journal of Engineering, Vol. 15, No 1, pp. 29-34, 2017.

[9] Dragan, D., Gajić, B.D., Petrović, B.V., Lazor, M., and Anišić, Z.: State of the Art in Virtual Reality Shops, in *Proceedings of the 8th International Conference on Mass Customization and Personalization in Central Europe MCP-CE*, 19-21.10. 2018, Novi Sad, Serbia, pp. 74-80.

[10] Veg, E., Regodić, M., Joksimović, A., and Gubeljak, N.: Development of the transmission tower virtual 3D model for structural analysis in ANSYS, FME Transactions, Vol. 45, No. 2, pp. 232-235, 2017.

[11] Jeli, Z., Stojicevic, M., Cvetkovic, I., Duta, A., and Popa, D.L.: A 3d Analysis of Geometrical Factors and Their Influence on Air Flow Around a Satellite Dish, FME Transactions, Vol. 45, No. 2, pp. 262-267, 2017.

[12] Miroljub, A., Dragan, D., Mihić, S., and Anišić, Z.: Review of 3D body scanning systems, Acta Technica Corviniensis – Bulletin of Engineering, Vol. 10, No 1, pp. 17-24, 2017.

[13] Čeliković, D., Batalo, B., Radisavljević, D., Dragan D, and Anišić, Z.: 3D Avatar Platform — A Unique Configurator for 3D Figurine Customization, in *Proceedings of the 8th International Conference on Mass Customization and Personalization in Central Europe MCP-CE*, 19-21.10.2018, Novi Sad, Serbia, pp. 61-65.

[14] Sljivic, M., Pavlovic A., Ilic J., Stanojevic M., and Todorovic S.: Comparing the accuracy of professional and consumer grade 3D printers in complex models production, FME Transactions, Vol. 45, No. 4, pp. 348-353, 2017.

[15] McHenry, K. and Bajcsy, P.: An overview of 3d data content, file formats and viewers, Technical Report, National Center for Supercomputing Applications 1205. Available at: https://www.archives.gov/files/applied-research/ncsa/8-an-overview-of-3d-data-content-file-formats-and-viewers.pdf (last accessed 12 November 2018).

[16] Autodesk, Autodesk FBX SDK documentation. Available at: http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/index.html (last accessed 12 November 2018).

[17] Lazor, M.: An example of a software solution for animating a human model in the FBX format using the FBX SDK library, Diploma thesis, Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia, 2017.

АУТОМАТИЗАЦИЈА ПРОЦЕСА АНИМАЦИЈЕ АВАТАРА У fbx ФОРМАТУ

М. Лазор, Д. Б. Гајић, Д. Драган, А. Дута

Анимирани модели су саставни део сваке видео игре или анимираног филма. Анимирани садржаји и анимирани 3Д модели се све више користе и у другим гранама индустрије. Како би се време израде анимираних 3Д модела свело на минимум, све више се тежи аутоматизацији одређених делова процеса анимирања. Програмско додавање анимација представља један од начина оптимизације и убрзања процеса анимирања. У овом раду описано је аутоматско програмирано додавање предефинисане анимације на већ постојећи скелет 3Д модела човека. Сам модел и скелет су претходно настали аутоматски скенирањем људи. Резултат анимације чува се унутар FBX формата који представља индустријски стандард. Описано јединствено решење је развијено помоћу C++ програмског језика и Аутодескове програмске развојне библиотеке FBX SDK. Предложени приступ омогућује одређену дозу аутоматизације коју традиционални приступи и алати, попут Autodesk Maya и Autodesk 3D Max, не могу пружити.