УДК 004.942(045)
DOI: 10.22213/2410-9304-2022-3-42-54

Modeling and simulation of two ambidextrous anthropomorphic manipulators to perform cooperative tasks

G. Ch. Uboh, студент, Kalashnikov Izhevsk State Technical University
M. A. Al Akkad, кандидат технических наук, Kalashnikov Izhevsk State Technical University

This paper is focusing on the development of a robotic arm using 3D modeling and simulation. The software used is Blender, which is a 3D modeling and simulation software that also supports programming language python as the scripting language. Blender was chosen over Maya because it is a free software suitable for students to develop their projects, and share the same features, more accessible, and the design is more realistic. The robotic arm was designed after studying the human arm and hand. The kinematics of the robotic arm were derived. The simulation shows the movement of rigged objects, e.g., an arm controlling a gun, and bullets projecting from the gun, and it was done using key frame animation and game engine simulation. Modeling and simulation of two robotic arms and hands, shooting with an AK-47 rifle at a bullseye, were done and completed using python in Blender. This work is intended to be integrated into a First-Person Shooter FPS game, which can be used to train biathlon sportsmen and army soldiers for precise shooting. Forward and inverse kinematics are implemented for the rig to move without breaking and deforming, then the bones and mesh together are combined as one unit, then the joined mesh is put to a desired position of a shooting position, then a rifle is added, then the bullets are simulated falling on ground when the bullets hits the target. Then finally the whole file is exported.

Keywords: Blender, Robotic arms, Rendering, Modeling and Simulation, Inverse kinematics, Rigging, Animation.

Introduction

In the realm of modeling and simulation, whatever is imagined can be created, what doesn't get exist can be seen, where time and space, are under people's command. A world where anything can be done and there are no mistakes or wrong answers with the only limits are those of the imagination. Modeling and simulation, are ways of understanding the world around us through creativity and imagination.

There are three main types of models physical, mathematical, and process. A physical model is one whose physical characteristics resemble those of the item being modeled, in other words it looks, or feels like the real thing, it could be a tweet plane, a statue, or the blueprints for a house. A mathematical model is one that uses mathematical symbols and relationships to describe something, evolved on graphs and math class, but it could also be a chemical formula or baseball statistics. The process model describes the steps we need to follow to get something done this could be as simple as to do list, or in computer, a flow chart. With these three types of models basically anything can be modeled, equipment, systems, environments, people even behaviors. In 3D modeling, different spheres, and things can be made, and then just building upon that, using a program like Maya, or Blender. A rig man can be created and lip sync and facial expressions can be practiced. Enthusiasm is all is needed to learn modeling and simulation, it's fun and interactive and it actually opens up a lot of fields for job opportunities. Models can be used in real world applications, like in 3D cartoon production industry, and how to create characters and put everything together. Modeling and simulation, can be used by anybody, no matter what their skills are, there's a place for them.

Just like there are three main types of models, there are three types of simulations, live, virtual, and constructive. A life simulation is one that involves real people operating real systems like practicing golf, or soldiers simulating military maneuvers. A virtual simulation is where real people upgrades simulated systems, like flight simulators, or driving simulators. Constructive simulation is where simulated people operate simulated programs, people provide the input situation it could be traffic, weather, or explosion, and the simulation tells them what might happen. A serious game is the serious application of video game technology in simulation, it is created for real world applications, like a medical training simulation, where doctors are trained in a virtual environment to diagnose patients, and perform surgeries. In a constructive simulation, an incident commander is used for training individuals to put together an emergency situation incident, it is a virtual world with virtual people, where by providing input to simulation, e.g., an explosion, it allows police, firefighters, and other emergency personnel to learn how to work together more effectively.

All simulations, live, virtual, and constructive are great ways for people to learn how to handle situations and become better decision makers. Letting people make mistakes in a virtual simulation allows them to make better decisions when they are going to the real world. Simulations also allow us to test ideas before committing to them, making models and then seeing how they work and simulations allow us to make improvements more quickly more safely and for less money.

Modeling and simulation are used everywhere you look from movies to games to building cars, flying airplanes, the possibilities are endless, people can make their own models and simulate them to do whatever they want, the world of modeling and simulation is wide open, so no matter what the interests of a person are, there is a place for him [1].

Simulation is very essential in modeling of robotics. Engineers can keep working on simulations over and over before creating the hardware of the simulated robot [2]. Focusing on the movement of a robotic arm can be difficult during simulation, e.g., picking up an orange and peeling it [3], or shooting with an AK-47 rifle, the problems stem from the degree of total movement and freedom of the arm needed for the simulation to be perfect and ready for real life modeling and creation [4].

This paper will explain and show the movement of a human-like robotic arm and its kinematics using Blender [5]. To start the modeling process of the hand and the arm, a cube is first created, and several steps will be taken to make hand and the arm look realistic enough [6]. The bones will be created, forward and inverse kinematics will be implemented for the rig to move without breaking and deforming, then the bones and mesh together will be weight painted to make them one unit so they can function together as one, then the joined mesh will be put to a desired position of a shooting position, then a rifle will be added, then the bullets will be simulated coming out from the gun, and a reload simulation, and cans falling on ground when the bullets hits the target [7]. Then everything will be animated together to make them function as one, and finally export the whole file.

Kinematics of the human arm

The human arm has 7 DOF, including the shoulder, where each joint has 1 DOF. The kinematic of the human arm are studied and implemented to predict a realistic motion of the robotic arm [8]. The wrist bone consists of 3 DOF, the shoulder joint consists of 3 DOF, and the elbow consists of just 1 DOF [9]. The human hand works with the combination of rotations, movements, inner rotation, the placement of the shoulder, and movement of the elbow joint [10]. The kinematic movement of an anthropomorphic manipulator is a complicated system, in particular the shoulder [11]. The kinematic frames of the arm are shown in Fig. 1.

There are two spherical joints in the arm, the wrist, and the shoulder, and, a revolute joint, the elbow [12]. By including a rigid coordinate frame {0} placed at the start of the shoulder and a coordinate frame {i}(i = 1.7) placed at every joint, A_i is defined as the 4×4 homogeneous transformation from frame (i - 1) to frame {i} where the joint variable is θ_i.

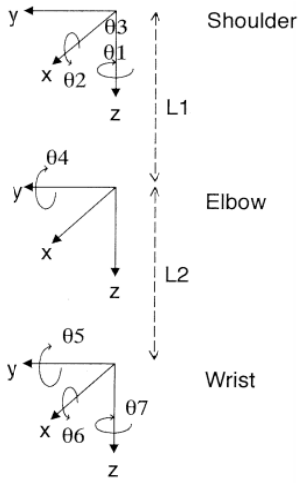

Fig. 1. Simple Diagram of the Human Arm

Рис. 1. Простая схема человеческой руки

A_i is described as follows:

$$A_1 = R_z(\theta_i) = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_2 = R_x(\theta_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & -s_2 & 0 \\ 0 & s_2 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_3 = R_z(\theta_3)T(0,0,L_1) = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_4 = R_y(\theta_4)T(0,0,L_2) = \begin{bmatrix} c_4 & 0 & s_4 & s_4L_2 \\ 0 & 1 & 0 & 0 \\ -s_4 & 0 & c_4 & c_4L_2 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_5 = R_y(\theta_5) = \begin{bmatrix} c_5 & 0 & s_5 & 0 \\ 0 & 1 & 0 & 0 \\ -s_5 & 0 & c_5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_6 = R_x(\theta_6) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_6 & -s_6 & 0 \\ 0 & s_6 & c_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$A_7 = R_z(\theta_7) = \begin{bmatrix} c_7 & -s_7 & 0 & 0 \\ s_7 & c_7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

c_i and s_i indicate cos θ_i and sin θ_i and L_1 and L_2 are calculated as constants portraying the length of the upper arm and forearm. The wrist is placed in a fixed position and the wrist frame is related to the shoulder frame, the chosen inverse kinematics is to determine a set of angles θ_1, …, θ_7 that fulfil the following equation:

$$A_1 A_2 A_3 A_4 A_5 A_6 A_7 = A_{\text{wrist}}. \quad (2)$$

Then A_{wrist} will be as follows:

$$A_{wrist} = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

The given vector $p = [g_{14}, g_{24}, g_{34}]^T$ is placed in the given position of the wrist, being measured by the placed coordinate system. If p is placed, the angle of the elbow θ_4 is determined through the given distance of the placed wrist from the shoulder as:

$$\theta_4 = \pi \pm \arccos\left(\frac{L1^2 + L2^2 - \|P\|^2}{2L1L2}\right) \quad (4)$$

Even though there are two different solutions for θ_4, just one is realistically realizable due to joint limits. Equation 1 is under constrained as A_{wrist} specifies 6 rather than 7 independent quantities, so there are an infinite number of values for $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6$ and θ_7 that satisfy it. To obtain a finite set of solutions, an additional constraint must be provided [13].

Modeling the arm and hand in blender

Inverse kinematics IK helps to make moving limbs and other joints easy to animate. Forward kinematics FK is just the way the parent bones affect the movement of the child bones, i.e., when the bones change, it only transforms the bones that are down the chain, where moving the child will not affect the parent [14]. Inverse kinematics, gives access for the change of a given bone close to the last point of a chain to regulate bones that are high up the chain of bones, basically when the child bone has influence on a parent bone [15]. For this the auto IK option has to be chosen by opening the right hand side menu by clicking N, then tools, which is a simple checkbox that helps to pose bones very quickly. Implementing this, bones can be moved easily at the bottom of the chain to navigate the transformation of the whole chain. This comes to use when modelling hands in real life when people raise their hands, they move their upper arm and extend their forearm quickly allowing their hand to travel upwards, then order their hands to move upwards and their upper arm and forearm act accordingly to accomplish this movement. To apply IK to rigs, the auto IK feature is used for quick posing, but it doesn't give full control over the IK and it doesn't work for animation, so instead the IK bone constraint is used to set it up properly. The first step is choosing pose mode, select the forearm bone, and add the inverse kinematics bone constraint from the bone constraints tab indicated by the wrapped blue bone icon in the properties' editor, from there it is noticed that few input fields have be to filled for the object being targeted, this will be the armature itself. To target a specific bone within the armature, from choosing/selecting armature, the bone is selected. This works in a similar way to the track and stretch to constraint where the target bone is, where the active bone to point to, so in this case the hand bone will be chosen, however when the hand bone is selected to be moved, it doesn't actually seem to do anything, in fact it's not translating at all as it's still joined to the forearm bone, edit mode is selected and hand bone is disconnected from the forearm bone but kept parented. Now the hand bone can be translated and it influences the forearm bone but it also influences the entire rig, which is not what is wanted. The arm is desired to bend so IK bone constraint is selected and the chain line option, the default is zero and this simply goes up the entire bone chain, the active bone is parented to, so it goes all the way down to the bottom of the spine, this is indicated by a yellow dotted line. By changing the chain length value the yellow dotted line will show, how far up the chain, the inverse kinematics will take effect. It is better seen in wireframe mode. The only bones that have to influence by the IK constraint are the forearm and upper arm bones so a chain length of two is required. When

the hand bone is moved, it correctly influences only the forearm and upper arm, but it is not wanted to be freaking out, it doesn't seem like it's very animatable or stable. The forearm is pointing to the hand but the hand is parented to the forearm, i.e., that when the hand moves the forearm will try to point to the hand. If the forearm moves to the selected hand, the hand will have to move because it's being parented to the forearm. Moving the hand further will also move the forearm further to point to the hand which will move the hand further because it's parented which will move the forearm to point to the hand, this is called a cyclic dependency and it is not desirable. So the hand should be unparented entirely so that we can move our hand independently from our forearm. Now the movement of how the hand bends the elbow is very realistic, the only problem is that the hand is no longer connected to the forearm. The hand can be now stretched out away from the forearm, which isn't ideal, so to work this around, edit mode is selected, the hand is reparented to the forearm, then the hand bone is duplicated and the duplicate is unparented. An independent controller bone appears that it can be pointed to, instead, let's call this bone hand_IK.

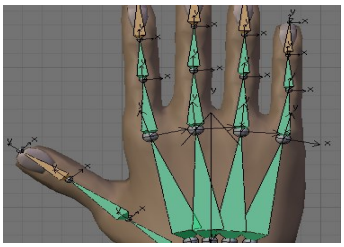

Fig. 2. Wrist Bone Emulating the Human Wrist Movement

Рис. 2. Имитация движения запястья человека

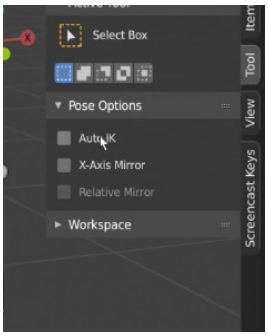

Fig. 3. Auto Rig

Рис. 3. Авто rig

Going back into pose mode, we can select our forearm, go into the bone constraints tab, and change the bone target from hand to hand_IK. Now as we move our hand_IK bone, the forearm and upper arm deform nicely and we can move this hand. The controller is taken out without stretching the hand itself, however, now to rotate the hand bone itself we have to switch between animating the hand_IK and the original hand bone, which is a bit tedious, so a simple copy rotation constraint is added for the original hand deformation bone to follow the hand. The hand eye is going to make the bone a bit bigger in edit mode so it's easier to differentiate from the original hand bone. Now we have a nice basic eye care, except that sometimes when we move the hand bone around, the elbow bends the wrong way, which is not desired. There is still one more input field left in the inverse kinematics constraint. The pool target input field will be helpful for solving the issue of joints bending the wrong way during inverse kinematics. A poll target is simply a reference object or bone that the elbow joint or IK joint will try to point to as the bones bend, for this we can simply create a new bone by duplicating the hand, keep on moving it behind the elbow. It will be rotated so that it points away, but it doesn't matter because only its location is actually referenced, and renamed to elbow_target. Back again into the IK constraint settings and input the armature again for the object field for the bone input. The elbow_target is selected and when the hand is moved, the elbow never bends in the wrong direction anymore, except that's because the elbow doesn't bend at all, this is because it was created too straight and it doesn't know which direction to start bending. So back into edit mode, the elbow joint is to be dragged slightly backwards. The elbow bends but still in the wrong direction, because one more value should be defined in the IK constraint. The elbow is kept bent and then the forearm bone is selected again, and in the IK constraint this pull angle value is clicked and dragged until it points to the pole target correctly. Now the inverse kinematics rig in blender is working.

As shown in Fig. 2 the wrist bone is emulating the human wrist movement shown on axes x and y. Fig. 3 shows a simple checkbox. Blender doesn’t permanently change the armature at all but rather helps when bones posing is desired. Enabling these bones, they can be easily moved at the end of chains to influence the transformation of the entire chain itself. A 3D model consists of faces, edges and vertices. When two vertices are put together the edge is created and when more than two vertices are put together and cover the space between edges then the creation of a face is made. The face is known as a polygon and a vertex is known as a point in space.

Fig. 4 shows that the faces are subdivided into: Quads, Triangles, and Polygons. When it comes to the modelling of a 3D object or mesh it is advisable

to use the quad which is the 4 side faces as it can be smoothed, and if subdivisions are focused on, the model might break or deform. When it comes to utilizing the polygons and triangles, problems may arise, because it can cause pinches in the object and that's not desired, although such method is possible but is not advisable. When trying to start modelling, edit mode has to be selected first or pressing tab on the keyboard, then faces, edges and vertices can be selected, as seen in Fig. 5, where in blender the edges, faces and vertex are the tools needed especially when it comes to extruding. When the face is picked up, it helps extrude the whole face, which means it can drag the main part in a box or a wider part of the box, e.g., it will be like the front part of a box being dragged to make it bigger and also it can be deleted, when the edge is chosen, it helps by selecting just the edges of an object or mesh and the same logic goes to the vertex, a dot to be specified, and it's also possible to select a lot of it by hitting Ctrl B or C. All this can only be done in the edit mode. Fig. 6 shows the process of the whole work from creating the mesh-rigging.

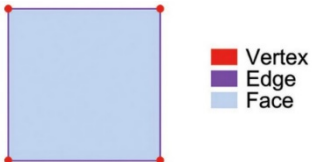

Fig. 4. Vertex, Edges & Faces

Рис. 4. Вершина, ребра и грани

Fig. 5. Accessing the Tools

Рис. 5. Доступ к инструментам

A basic hand shape can be created just from the cube shape:

- Make the cube smaller, and move one of the edges to the center of the palm of the hand. The base of the thumb will be what is left from the diagonal face.
- Add two loop cuts, one close to the wrist and the other one close to the fingers.
- Choose the base of what you want the thumb to be and extrude it to create the thumb.
- Smooth the whole mesh by selecting the whole hand and accessing the smoothing tool to create the geometry.
- Remove the top faces by pressing Ctrl B. and selecting the base and delete it.

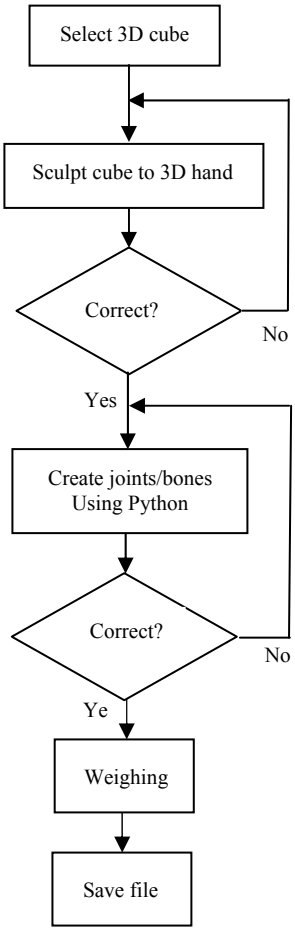

Fig. 6. Activity Diagram

Рис. 6. Диаграмма деятельности

Fig. 8 and Fig. 9 show the steps used in the creation of the hand and fingers.

The bones are essential when it comes to the development of any object because it helps with the movement and motion of the object, technically without the bones the human body is useless because that is what helps us move. Even when smile the bones are needed for it, when walking, talking, swimming, running, jumping, etc. Bones are needed to perform physical tasks. Bone creation, extruding, bones duplication, rigging, contraints, and weighing, using python API will be discussed.

Before starting any project, we must first start by modifying a cube in blender. Fig. 7 shows the 3D cube.

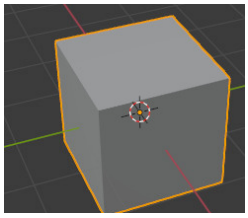

Fig. 7. The 3D Cube

Рис. 7. 3D-куб

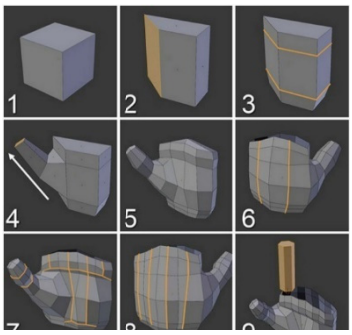

Fig. 8. Steps for Modelling a Hand

Рис. 8. Этапы моделирования руки

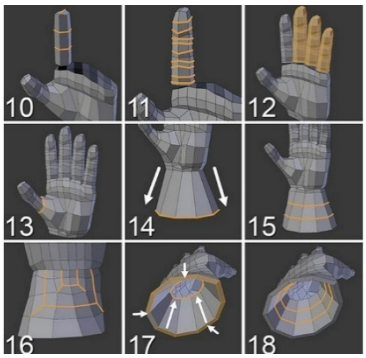

Fig. 9. Finger Modelling

Рис. 9. Моделирование пальцев

Bone creation: The blender armature has nice features but it is almost useless alone and it doesn't render. To create it automatically, edit mode has to be selected, and Shift A is clicked, where a menue shows up like in Fig. 10 for automatic armarture creation.

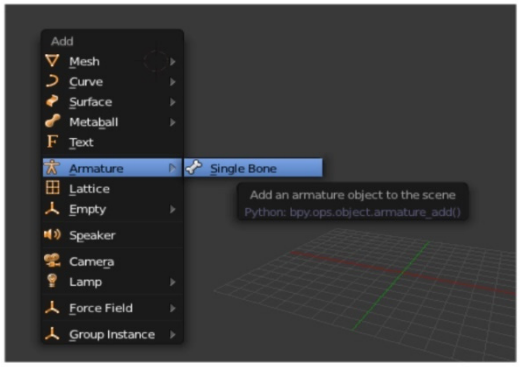

Fig. 10. Shift+A to do it Automatically

Рис. 10. Операция Shift+A

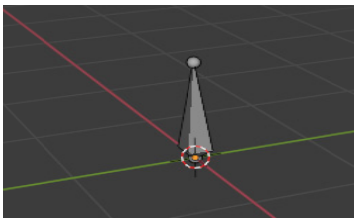

Fig. 11. Armature/Bones using Python

Рис. 11. Арматура/скелет с использованием Python

Python API for armarture creation

```
   bpy.ops.object.mode_set(mode='EDIT', toggle=False)
   obArm = bpy.context.active_object #get the armature object
   ebs = obArm.data.edit_bones
   eb = ebs.new("BoneName")
   eb.head = (0, 1, 1) #if the head and tail are the same, the bone is
deleted
   eb.tail = (0, 1, 2)    #upon returning to object mode
```

Fig. 11 shows the bones in blender which consist of the head and tail, i.e., top and bottom.

Extruding: Blender knows the difference between main and ordinary bones by the extrusion process. When creating a project and starting the process of rigging, the first step has to be adding a bone and extruding the bone until the full set of bones are complete, e.g., when creating the arm, the first step is adding an armature then extruding that armature.

Extruding is just the process of creating or better making a copy from an original bone as shown in Fig. 12, this helps when creating an armature from scratch. It is not advisable to create a bone over and over again because then the rig and its IK can't be implemented.

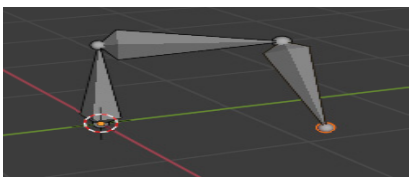

Fig. 12. Extruded Bones

Рис. 12. Вытянутый скелет

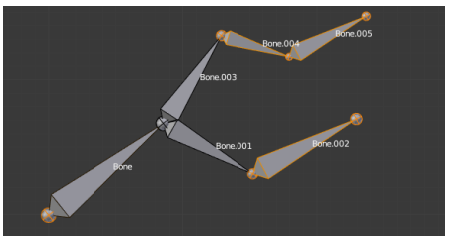

Fig. 13. Duplicated Bones

Рис. 13. Дублированные кости

Python API for Extruding

```
   bpy.ops.armature.extrude_move(TRANSFORM_OT_translat
e={"value":(0,0,.25)})
```

Duplicating bones using Python: Duplication is the process of mirroring a bone without reversing, or making a copy, which is useful when trying to make other bones or add them to a different mesh. Fig. 13 shows the duplication process.

Rigging the arm and the hand: In Blender, rigs are known as armatures. Inside the armature, the bones are there to turn into the rig. Rigs are used to make the process of animation easier. The rig consists of: bone, constraints, custom shapes. The rig of the arm is shown in Fig. 14. Figure 15 shows the creation of the hand rig.

Python API for duplicating bones

```
import bpy
arm = bpy.context.object.data
for b in arm.edit_bones[:]:
    cb = arm.edit_bones.new(b.name)
    cb.head = b.head
    cb.tail = b.tail
    cb.matrix = b.matrix.copy()
    cb.parent = b
```

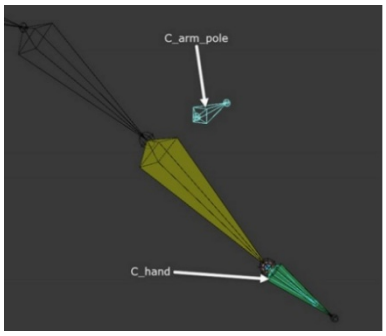

Fig. 14. The IK Rig of the Arm

Рис. 14. IK Rig of Arm

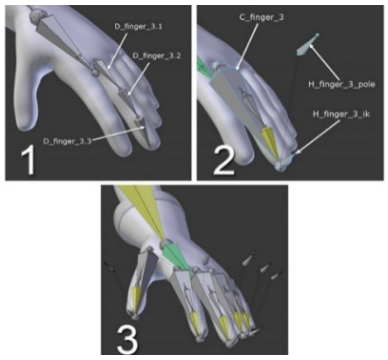

Fig. 15. Creating the Hand Rig

Рис. 15. Создание ручной оснастки

- In Edit mode a new bone has to be extruded from the elbow. Pressing Alt P stops the connection with the arm bone and move it back. This bone is the pole for the arm's IK.

- Duplicate the D_hand bone, disconnect it by pressing Alt P, and scale it down. Using the 3D cursor, Shift S, move the head of the new bone to the wrist joint. This bone serves as the IK target for the arm and also controls the hand's rotation. It is scaled down so that it doesn't completely overlap with D_hand, as the bones are in the same position. Making the bone smaller or bigger allows seeing it in wireframe display mode Z and makes it easier to select. An IK constraint is to be added, using the new bone for the hand C_hand as its target and C_arm_pole as the IK pole.

- Select the D_hand bone. On the Bone tab of the Properties Editor, deactivate the Inherit Rotation option in the Relations panel, as was done for the head bone. Add a Copy Rotation constraint to the bone, using C_hand as its target. When the C_hand is moved, the arm's IK is controlled, and when it is rotated, the hand's bone is rotated as well.

- Create the bones for a single finger and give them their respectful names. D_finger is used to describe the finger position and the articulation number. A tricky thing about fingers is the bone orientation. One of the bones can be selected and oriented correctly by rolling it with Ctrl R, and then select the other bones of that finger, with the one rolled being the active selection. Press Ctrl N, and select the Active Bone option to cause the roll of the selected bones to fit the active bone's roll. When the rotations are set up, select the first bone of the finger chain, and make it a child of the hand bone.

- Create a bone with its head and tail aligned to the beginning and end of the finger chain; this bone controls the entire finger. At its tip, extrude a new bone to act as the IK target. Duplicate that target, and move it up to be used as the pole for the finger's IK. In Edit Mode, make the C_finger_3 bone

a child of the hand, and make the IK target and pole children of C_finger_3. Apply an IK with those targets and poles to the D_finger_3 bone. Remember to set IK Chain Length to 3 so that it goes up through the finger bones only. It is seen that only the C_finger_3 bone has to be used to control it. Rotate the bone to rotate the finger, and scale it up and down to flex the finger.

- Duplicate the finger in Edit Mode, and place copies of it in the rest of the finger spaces; be sure to align and name them properly. For the thumb, the first articulation has to be deleted using only two. The C_finger_1 bone has to be realigned to the deformed bones.

Constraints: a feature in Blender that allows using constraints in a mesh to tweak the range of motion of the skeleton, where without this, the bones will not be able to function properly, it won't have a natural behavior. It is known that the human hand can't bend to a certain degree, if it bends too far it will break and that's not what we want, that's why constraints are needed. To set up constraints we need to specify the max and min of the rotation angles. When all these are done then the bones take their natural position, as shown in Fig. 16.

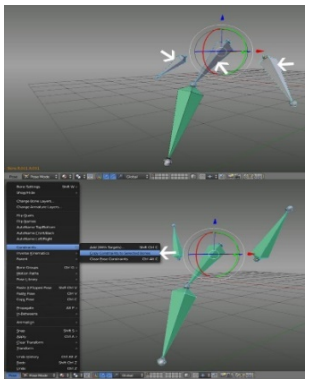

Fig. 16. The Constraints of the Bones Before and After

Рис. 16. Ограничения костей до и после

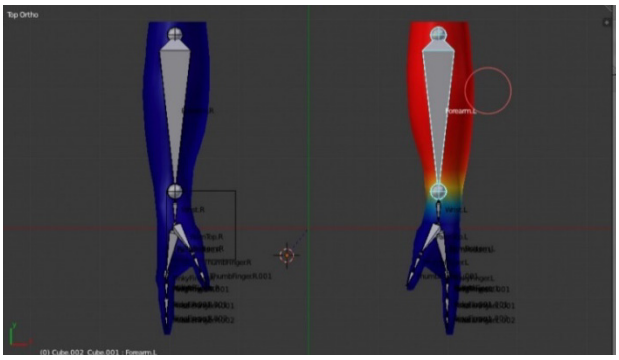

Fig. 17. Weighing both Bones and Mesh

Рис. 17. Взвешивание костей и сетки

From Fig. 16 it is clear that without the use of constraints, the bones will not be able to function properly.

Weighing down of the mesh and armature: is the fusion of the 3D mesh and the bones of the armature. The structure of the bones, rigging, extrusion, and how they work was shown, but without the fusion of the mesh, the 3D object will end by separating it from the armature. This defeats the purpose of creating an arm, as if the flesh moving separately from the bones. So that's why we have to bond or fuse both the skin and bones together to make that look realistic. Weighing is the process of joining both the bones and skin together, this is done automatically. Fig. 17 shows the weighing process of joining both the bones and skin together, this is done automatically. Fig. 18 shows the steps taken from the beginning until when things are put in the scene: start, concept of design, creation of the 3D mesh of hand, creation of bones, rigging, skinning, animate/simulate, render, and put in the scene. The simulated object that was created has a representation of both a logical and physical perspective. When looking into it we can see the perspective of the logical aspect in the director and hand simulated object which acts intertwine with each other. The director gets information from the "script". It is a text file which contains at least thirty dimension 30 D vectors and also instructs the hand to pose in a 30 D vector, one after the other in the given *direct*() method. Which means the director influenced object calls the set_pose() function of the simulated hand object and allows a 30 D vector to pass through it. The hand object rest on the pose told by the vector by switching the configuration of the bones. Once the pose of the hand is given, the director object "captures the pose" by recording the settings of all the bones of the hand object, and makes a keyframe in blender. This process will be repeated for each pose in the script/text file until the end of the text file. By creating the sequence of keyframes, we create an animation. The hand pose is given by *set_pose*() function and sent to the *set_skeleton_pose*() function of the skeleton function and passes the 30 D vector through it. The *set_skeleton_pose*() function, then receives the pose from the *set_pose*() functions of every limb while passing only the required 30D vector needed for each limb to set its given pose.

When it's done, the call to take up a pose, comes from higher level, in this object. The limb class is made up of the Limb sub-class. This class has the *bone_extrude*() method. This function receives the parameters as the parent bone object, the child bone after extrusion, and the location tuple.

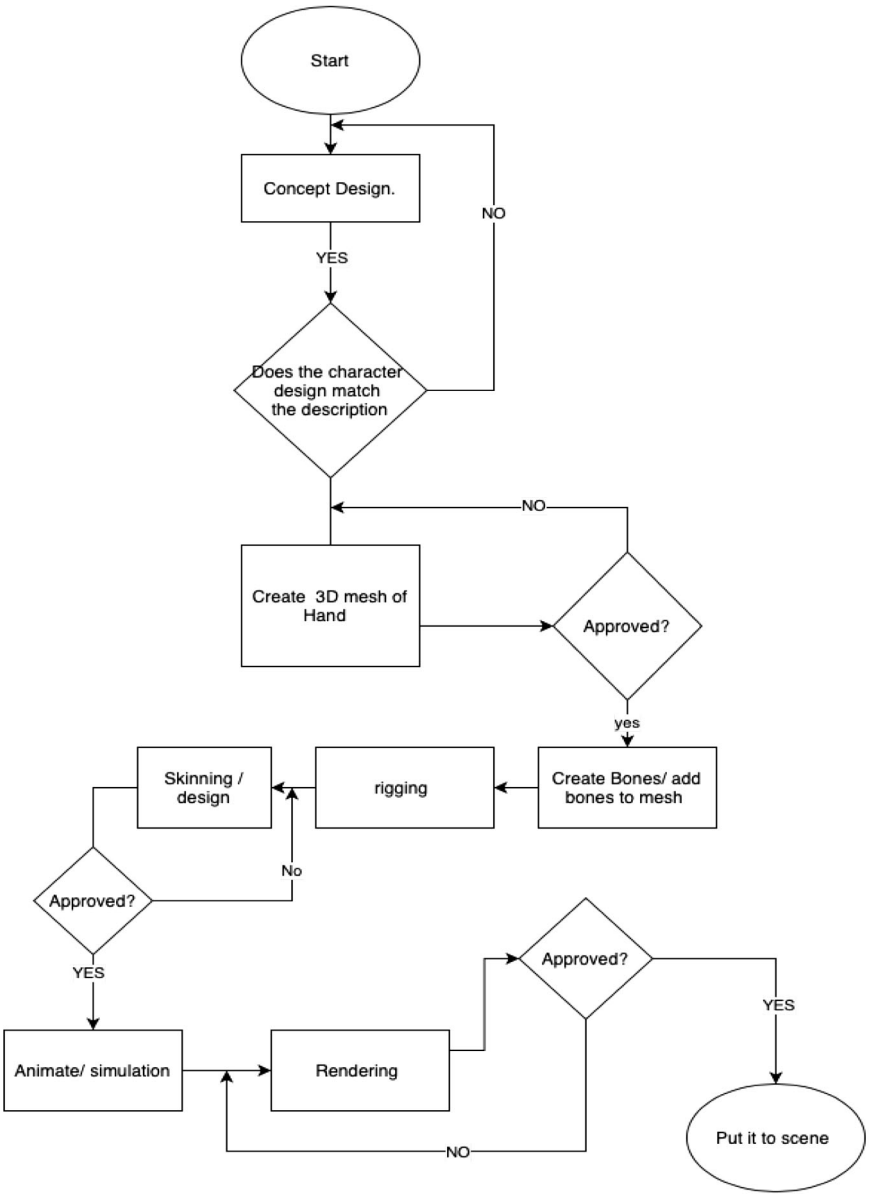

Fig. 18. Flowchart of the whole process

Рис. 18. Блок-схема всего процесса

From this specific method, a new bone is extruded from the joint of the parent bone, we do some synchronization operations to ensure that the newly created bone and the child bone object will pass as a parameter into our code referring to the same bone and we set the free end of the newly extruded child bone to the *x*, *y*, *z* coordinates that are specified in the parameter. The sub-classes of the given Limb class have different attributes, to which the object of each bone of that limb is corresponding.

The bone objects consist of different attributes such as their name, and their location and rotation, which are related to the parent bone instead of a global location and rotation, or a location and rotation connecting initial orientation of the bone. They also share such attributes as their parent bone object and different variants of their children bones. They are labelled as NULL when the bone object is created or modified when the bone is actually extruded by the *bone_extrude*() method in the Limb class. Bone objects can also change the rotation and position of the bone. In the sub-class of the limb class, it also has a *create_XYZ*() method, e.g., *create_arm*() in the arm class. This method is made from the buildup of the Limb class, once the class attributes have been made. This method is responsible for the creation of the limb. It does so by starting at the bone that would be the highest in the bone hierarchy of all the bones in the limb as seen in the upperarm bone in the arm, and calling the *bone_extrude*() method in the Limb class, passing as arguments, the appropriate bone objects.

The *create_XYZ*() method is also needed to make the required Joint class objects by passing the bone objects during instantiation.

The Joint class is used to set up the constraints between the given bones. A joint object takes into account the parameters of the child and parent bone objects, and pairs between the bones that are chosen to set up the needed constraint. The object also takes the min and max location and rotation values that are needed for setting up the constraint. The constructor of the object calls the *set_constraint*() method, which builds up the constraints in Blender. The joint consists of 2 main joints which includes the hinge and swivel joints. In the class each joint will create _XYZ() included in each limb. Following the human anatomy, the hinge joints can be found in different parts of the body such as the elbow, knee, fingers, and ankles. looking at the *create_arm*() function in the arm class, the hinge joints are used for setting up the constraints leading to the elbow joint and the finger joints. Sequence diagram in Fig. 19 shows setting the pose of a rigged arm by the director. Fig. 20 shows the class diagrams of the final design.

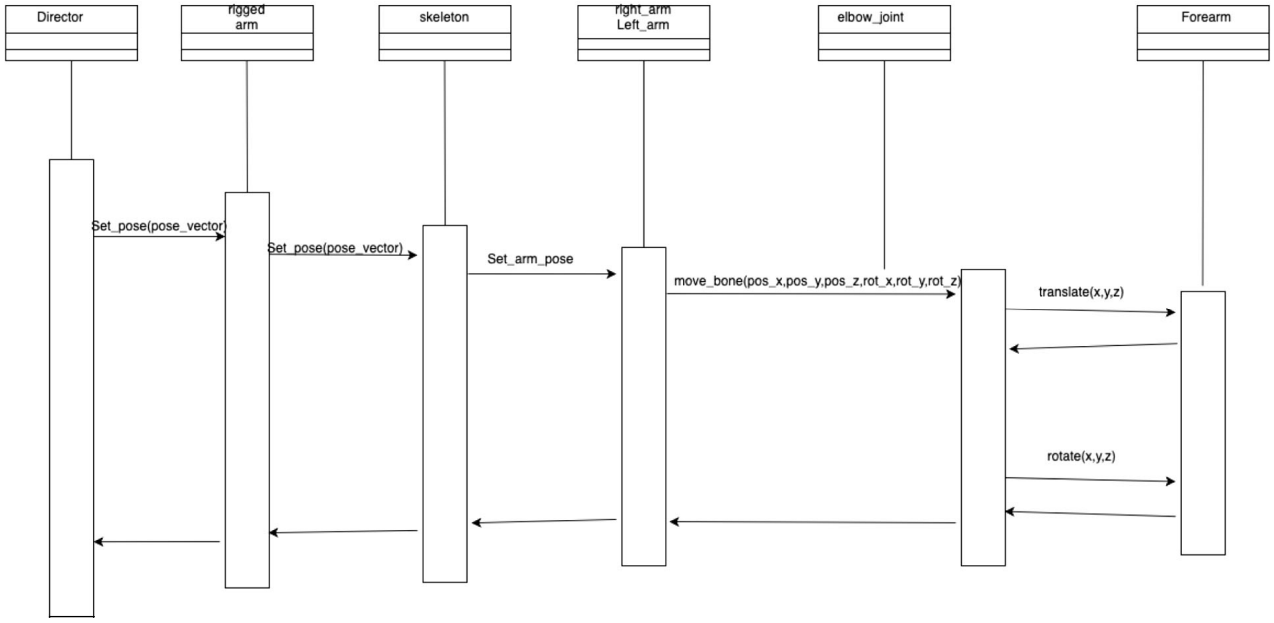

Fig. 19. The sequence diagram of bone setting by the director to the hand mesh and the skeleton

Рис. 19. Схема последовательности установки кости директором на сетку руки и скелет

Fig. 19 shows the sequence of how each bone is set by the director to the hand mesh all the way to the skeleton, while Fig 20 shows the design in UML form of the rigged arm. The UML diagram shows a class named the joint class that has different types of joints. If joint obj is selected, each joint passes the same constraint type, and uses the given min and max for each constraint which are the rotation and location constraints. Instead of passing a child bone and a parent bone to the constructor, the child bone is only used for the smaller bone as each bone can tell where the parent bone is. The joint object in the new prototype is used as an attribute of the limb. The initial constraints that the bones are subject to are defined within the director, and use the *create_XYZ*() methods. Few modifications were made for the DOF of the joint to function properly following the pattern of the limb.

- For everything to work properly, the bone was not controlled from the joint, calling the rotate() or translate() function directly of the selected bone obj from the set_pose obj of a selected limb, instead it is moved through joints. When moving the forearm, the function takes the move_bone funct from the elbow joint. The elbow move_bone is called as the attribute of a forearm object. The move_bone funct from the joint connects to the needed bone attribute method to translate or rotate. This helps in simplifying the bones movement. When moving a bone, the best way is calling the arm individually. This is the best method of connecting a joint object instead of calling a function to rotate and translate the bone differently.

- To influence the DOFs of the joint following the whole settings of the limb, the bone class stays as an attribute called associate_joint. This is the attribute that will be used by the set_pose() method in the limbs in order to move bones in the mesh, e.g., the forearm bone will be connected to the elbow joint as it's a linked joint also the upper arm will have the shoulder as it's a linked joint just like in the human arm.

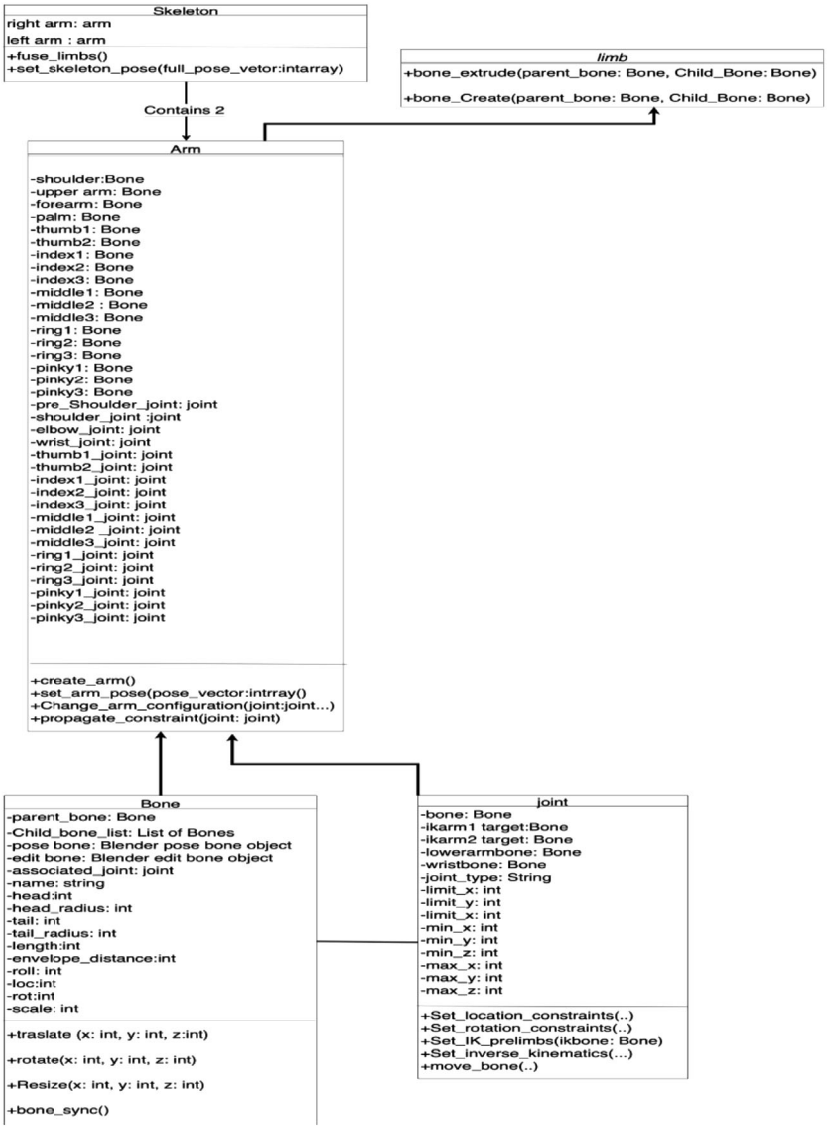

Fig. 20. Final design class diagram

Рис. 20. Окончательная диаграмма классов проекта

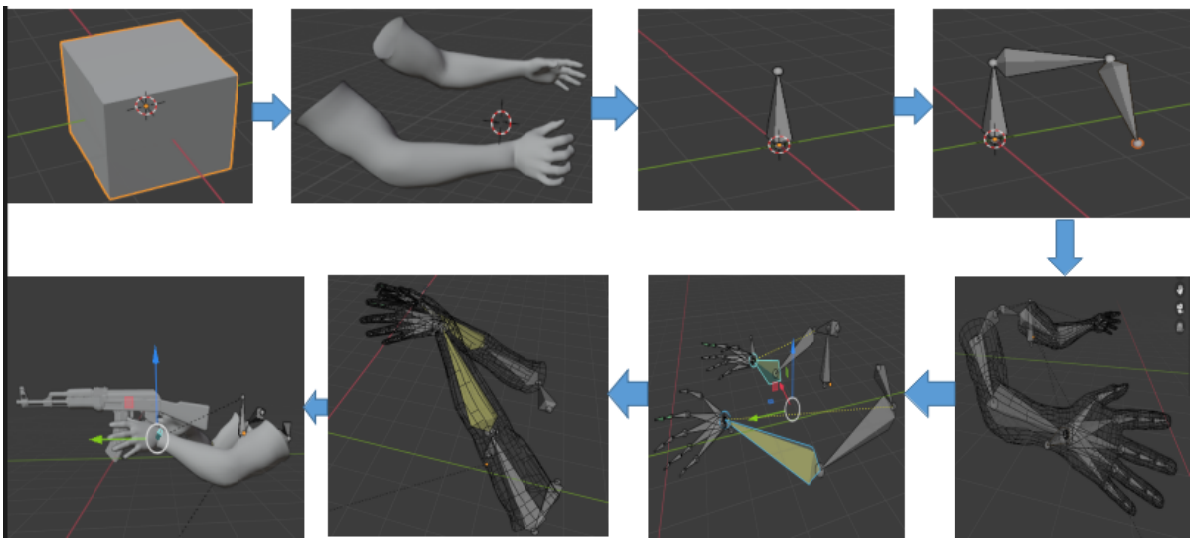

Fig. 21. The steps of the whole process

Рис. 21. Этапы всего процесса

Fig. 21 shows the whole process of modelling and addition of the riffle, starting from the cube, modelling the two arms and hands, making bones, extruding and duplicating bones, applying inverse kinematics, and weighing all together.

Simulation of the arm and hand in blender

Physics simulation is used as the process of bringing life to objects, e.g., cartoons, video games, flight or driving simulators, where the following are needed:

Key frames: a key frame is used to put a marker which stores the value in time, and can define the position of a cube set as 4 m on a frame, allowing the animator or simulator to set the time for objects to allow them to move.

Fig. 22 shows when a key frame is set, it is possible to set up the time on different marker then play back everything as one, like a movie. They are different types of keyframes: Keyframe, white/yellow diamond, is a normal keyframe; Breakdown, small cyan diamond, is a breakdown state. e.g. for transitions between key poses; Moving Hold, dark gray/orange diamond, is a keyframe that adds a small amount of motion around a holding pose. In the Dope Sheet it will also display a bar between them; Extreme, big pink diamond, is an extreme state, or some other purpose as needed; Jitter, tiny green diamond, is a filler or baked keyframe for keying on ones, or some other purpose as needed.

The design after implementing everything includes: Pose of the hand, adding a riffle, rendering, adding skins, adding bullet simulation, and background

Fig. 23 shows the first prototype and the final design after putting everything together..

Fig. 22. Visualization of keyframe Set up in blender

Рис. 22. Визуализация настройки ключевого кадра в Блендере

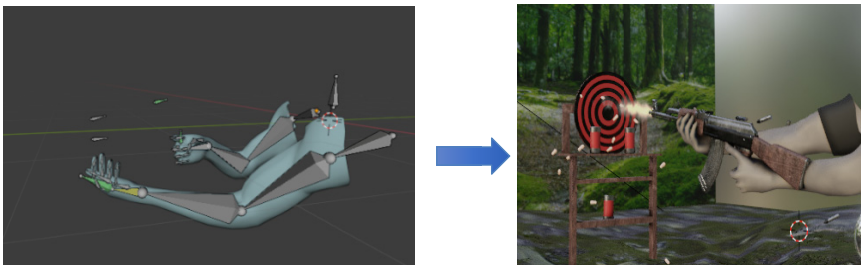

Fig. 23. Before and After

Рис. 23. До и после

Design flaw: The original prototype had problems with its design, constraints and rigging because the calculations were wrong and the hinge joint wasn't included. From the first design diagrams it's obvious that it doesn't have any skin color and the gun setup wasn't included but in the final design this issue was fixed.

Additions on features: The final design has new features which includes a forest as a background, an AK 47 rifle, bullets, better rigging, better animation, and better pose.

Fig. 36 shows a use case diagram showing the future scope of the whole project when integrated into a First-Person Shooter FPS game, which can be used to train biathlon sportsmen and army soldiers for precise shooting.

Conclusion

The whole process of modelling and simulation of two anthropomorphic manipulators were done and illustrated in Blender, starting from the cube, modelling the two arms and hands, making bones, extruding and duplicating bones, applying inverse kinematics, and weighing all together was discussed and illustrated. This work was intended for showing how two robotic arms and hands can work together in a cooperative task and as a case study the task of shooting with an AK 47 rifle, which was

added to the virtual environment with a chart and a background to demonstrate the simulated process of firing a rifle without even touching it. This work is intended to be integrated into a First-Person Shooter FPS game, which can be used to train biathlon sportsmen and army soldiers for precise shooting.

References

1. Blain J. M. (2018). The complete guide to Blender graphics: computer modeling & animation. CRC Press.

2. Urrea C., Saa D. (2020). Design and Implementation of a Graphic Simulator for Calculating the Inverse Kinematics of a Redundant Planar Manipulator Robot: Semantic Scholar. Applied. Sciences. vol. 10(19), p. 6770.

3. Al Akkad M. A. (2014). Exploiting two ambidextrous robotic arms for achieving cooperative tasks, Vestnik ISTU, no. 4, pp. 134-139.

4. Al Akkad M.A. Complexity Reduction for two Human-like Ambidextrous Robotic Hands Control, International Siberian Conference on Control and Communications SIBCON, IEEE. Pp. 1-7.

5. Anders M. (2010). Blender 2.49 scripting: extend the power and flexibility of Blender with help of Python; a high-level, easy-to-learn scripting language. Packt Publishing.

6. Assaf, E. (2016). Rigging for games: a primer for technical artists using Maya and Python. CRC Press, Taylor and Francis Group.

7. Pieper D., Roth B. (1969). The kinematics of manipulators under computer control. In Proceedings of the Second International Congress on Theory of Machines and Mechanisms, pp. 159-169.

8. Pitarch E. P., Omar A. A., Abdel-Malek K., Yang J. (2007). Virtual human hand: grasping strategy and simulation. PhD thesis, Universität Polytechnic de Catalunya.

9. Wadsworth C. T. (1983). Clinical Anatomy and Mechanics of the Wrist and Hand. Journal of Orthopedic and Sports Physical Therapy. Vol. 4, no. 4, pp. 206-216.

10. Taylor C. L., Schwarz R. J. (1955). The Anatomy and Mechanics of the Human Hand. Artificial Limbs, Vol 2, no. 2.

11. Tolani D., Badler, N. I. (1996). Real-time inverse kinematics of the human arm. Presence Teleoperators & Virtual Environments, MIT Press.

12. Craig J. (2005). Introduction to Robotics Mechanics and Control. 3rd edition. Prentice Hall.

13. LaValle S. (2006) Planning Algorithms. Cambridge University Press.

14. Mullen T. (2012). Mastering Blender, 2nd edition. Blender Store.

15. Villar O. (2017). Learning Blender: a hands-on guide to creating 3D animated characters. Addison Wesley Professional, 2nd ed.

* * *

Моделирование двух двуруких антропоморфных манипуляторов для выполнения совместных задач

Г. Ч. Убох, студент, ИжГТУ имени М. Т. Калашникова, Ижевск, Россия
М. А. Аль Аккад, кандидат техничнских наук, доцент, ИжГТУ имени М. Т. Калашникова, Ижевск, Россия

Статья посвящена разработке роботизированной руки при помощи 3D-моделирования. Применен программный продукт Blender, представляющий собой программу для 3D-моделирования на языке программирования Python. Программному обеспечению Blender было отдано предпочтение по сравнению с программным продуктом Maya, поскольку это свободное ПО, которое подходит студентам для выполнения проектов, обладающее при этом такими же характеристиками, оно более доступно, и моделирование получается более реалистичным. Роботизированная рука проектировалась после подробного изучения руки и кисти человека. Были выведены выражения для описания кинематики роботизированной руки. Моделирование показывает, например, как движется предплечье при стрельбе из пистолета и куда направляются пули, это было сделано при помощи анимации по ключевым кадрам и игры. Моделирование двух роботизированных предплечий и кистей при стрельбе из АК-47 по центру мишени было реализовано на языке Python и ПО Blender. Целью работы являлась интеграция в игру First-Person Shooter FPS, которую можно применять при подготовке биатлонистов и солдат для отработки точности стрельбы. Прямая и обратная кинематика применяется для обеспечения движения установки без поломок и деформирования при выборе положения звеньев для формирования единого узла с последующим добавлением оружия и падением гильз при достижении цели. После этого файл экспортируется.

Keywords: Blender, роботизированная рука, моделирование, обратная кинематика, наладка, анимация.

Получено: 05.07.22