DOI: 10.5748/18CONTECSI/PSE/ESD/6726

AN IMPLEMENTATION GUIDE TO PROCEDURAL ANIMATION

Vitor Novaes Cantão - UFPA - UNIVERSIDADE FEDERAL DO PARÁ - Orcid: https://orcid.org/0000-0001-5925-6164

Sandro Ronaldo Bezerra Oliveira - UFPA - UNIVERSIDADE FEDERAL DO PARÁ - Orcid: https://orcid.org/0000-0002-8929-5145

This work has the general objective of providing a guide for the implementation of procedural animation techniques, with more focus on character animation. Help beginner game developers interested in procedural animation techniques, guide them with a step-by-step approach, and contribute as a future reference. Identify procedural animation techniques based on three main criteria, select the methods that matched those criteria, new and more detailed research was conducted to be implemented within the Unity Engine, this knowledge became sections within the application guide. An Implementation Guide to Procedural Animation The construction of a guide built from the choice of procedural animation techniques based on three criteria: relevance in the gaming industry, significant return from its application, and ease of implementation. Step-by-step approach focused not only on the application itself but also on the reader's reflection about the implementation process The reader can also use the guide as a future reference, as it contains sections about how the methods work, when to apply them, technique complexity, and use in real-life scenarios.

Keywords: Procedural animation, Application Guide, Games, Inverse Kinematics, Unity

An Implementation Guide to Procedural Animation

ABSTRACT: The rise of computer graphics gave birth to a new generation of videogames where unparalleled graphics and meticulous animation became the standard. Furthermore, as the industry grew, the interest of new game developers and new tools have expanded as well. In the field of videogame animation, procedural animations yield results that would be otherwise impossible or extremely time-consuming, providing more vivid and game world-connected outcomes. This work aims to introduce new game developers to procedural animation techniques by presenting a guide with different methods and sections, containing: how to use, when to apply, and real-world scenario usage. The processes consisted of the selection of the techniques based on criteria such as industry relevance, outcome quality, and ease of implementation. Each method was studied and applied, resulting in the guide sections containing discussions about the implementation complexities, limitations, and pre-requisites.

Keywords: Procedural Animation, Application Guide, Games, Inverse Kinematics, Unity.

Acknowledgments: This work belongs to the SPIDER / UFPA research project (*http://www.spider.ufpa.br*).

1. INTRODUCTION

This section presents the introduction to the work, starting with context and the motivation behind its realization. Subsequently, this section presents the justification for having done so and the goals through this work. Finally, this section reports the development methodology and the work structure.

1.1. Context

In its 50-year history, video games have grown to become one of the most noticeable entertainment mediums (ZACKARIASSON & WILSON, 2012). In its early days, animation in games consisted of swapping a few 2D images, known as sprites, on a static background, given the limited hardware. Starting in the 90s, however, 3D graphics generation became quite popular, driven by the impressive graphics on newer systems, as seen in franchises like Super Mario 64 (WOLF, 2008).

With the transition from 2D to 3D, animations became increasingly complex, given the additional dimension, the possibilities for displacement and interaction in a 3D world, and the increase in computing power with the rapid hardware growth. Expectations for massive game productions continue to grow each year, with increasingly realistic graphics, larger worlds, and especially the quality of animation becoming even more meticulous and closer to reality.

However, despite the recent democratization of game development, mainly by the availability of powerful game engines to the general public, such as Unity and Unreal, many practices and techniques are still kept under industry domains, evidenced by the few veteran authors who share their knowledge (GREGORY, 2009). Given this context, this paper aims to present some relevant procedural animation techniques and democratize their access through an application guide.

1.2. Motivation

The motivation for this work is primarily personal, born from a desire to learn more about this theme and implement different methods to build a knowledge base to put the learnings into practice in actual game productions in the future.

The creation of an application guide arises as an opportunity to democratize and popularize the debate about the theme. This work aims to contribute to the learning of other developers, serving as an introduction to this theme.

1.3. Justification

The game market in Brazil, as in much of the world, is on a rapid rise (ZACKARIASSON & WILSON, 2012). Because of this, it is natural that there is an increasing search for game developers. However, in Brazil, there are few universities and colleges that offer more specific courses related to games. Because of this, the professionals in this area are partly self-taught (RUGGILL, MCALLISTER, & NICHOLS, 2012), learning from the abundant amount of material available on the internet, such as tutorials, videos, and articles. However, there is still a lack of more advanced content, especially in Portuguese.

Therefore, this work aims to help beginner game developers interested in procedural animation techniques, guide them with a step-by-step approach, and contribute as a future reference. The choices of methods presented in this work reinforce this idea, serving as an introductory knowledge base.

1.4. Objectives

Procedural animation allows the creation of animations that would once have been impossible to create traditionally or impractical for time reasons. In this way, its use in the video game industry allows increasing the quality, realism, and interaction with the world of big productions. Moreover, the use of procedural animations by independent developers can result in less effort in content creation.

Because of this, this work has the general objective of providing a guide for the implementation of procedural animation techniques, with more focus on character animation.

Specific objectives include:

- Introduce the concepts of procedural animation as a basis for learning multiple techniques;
- Identify procedural animation methods that have relevance, good practical results, and a relatively low level of complexity;
- Exploring the techniques in a way that guides the reader to understand them practically, through the building of the thinking behind their application;
- Serve as a reference guide for future reference.

1.5. Methodology

The first step was a search to identify procedural animation techniques based on three main criteria: relevance within the industry and use in games, i.e., the method must have a high degree of applicability; significant return, i.e., given its degree of complexity, a notable improvement in animation is expected; and ease of implementation, since the structure of the guide is in a step-by-step format, aimed at a reader who is a beginner in the theme.

After that, the second step was to select the methods that matched those criteria. Thus, for each technique chosen, new and more detailed research was conducted to be implemented within the Unity Engine. This implementation step was fundamental for a greater understanding, laying out the pros and cons and the specific implementation difficulties of the tools used. Later, this knowledge became sections within the application guide.

Subsequently, the structure of the application recommendations was made based on the lessons learned from the previous steps. Finally, the final step was to outline the step-by-step guide (CANTÃO, 2021).

1.6. Paper Structure

- Section 1 - Work Introduction. It presents the context, its motivation, its objectives, and the methodology for its construction;
- Section 2 - Theoretical basis behind this work. It presents the fundamental concepts, being essential for the understanding and application of the techniques in the subsequent section;
- Section 3 - Presents the recommendations for the application of the procedural animation methods, including multiple inverse kinematics techniques and real-time Bleendtree interpolation animation;
- Section 4 - Conclusion of the work. It presents the overview of the work, contributions of the work, limitations, and future work.

2. THEORETICAL FOUNDATION

This section presents the theoretical background necessary for the application of procedural animation methods. First, the section discusses the concepts involved in the term procedural animation. After that, there is an explanation of the concepts related to the applied methods. Finally, this section introduces the tool used in the application.

2.1. Procedural Animation

Traditionally, the animation process is a tedious task that involves specifying numerous keyframes (Figure 1) for various degrees of freedom to achieve the desired motion (BRUDELIN, 1996). Keyframing methods provide tools to manipulate joint angles only at the lowest level, meaning the animator must explicitly consider high-level interactions based on their experience and skill.

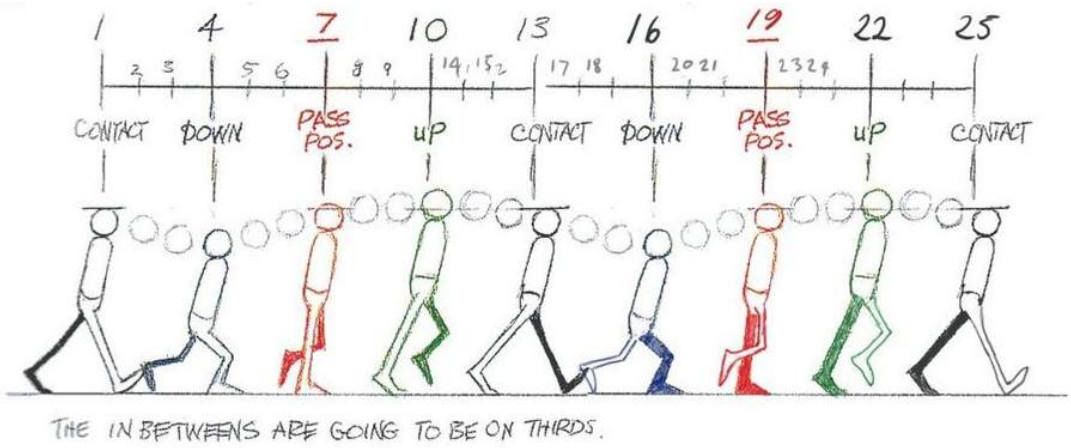

Figure 1 – Keyframes of a walk cycle.
Source: (WILLIAMS, 2009)

Often associated with the term random generation, procedural content generation is a contrast to the traditional method of content creation, where the artist manually fabricates each asset in the game. Typically, procedural generation involves a combination of content produced by an artist for greater control, along with the use of algorithms and pseudo-randomness. The origin of this technique in games was in the genres of Role Playing Game (RPG) and Roguelike (GOLDENBERG, BENHABIB, & FENTON, 2020). Figure 2 exemplifies a procedurally generated map in the game Rim World.

Figure 2 – Example of procedural generation. Randomly generated game map in the game Rim World.
Source: Author (2021)

Thus, the term procedural animation implies a type of digital animation generated in real-time. Its application within games expands from character animation (as in the game Rain World illustrated in Figure 3) to particle systems (such as snow, fire, smoke), cloth simulation (such as flags and clothes), and ragdoll physics, which seeks to simulate a lifeless character colliding with objects and the scene in a natural way.

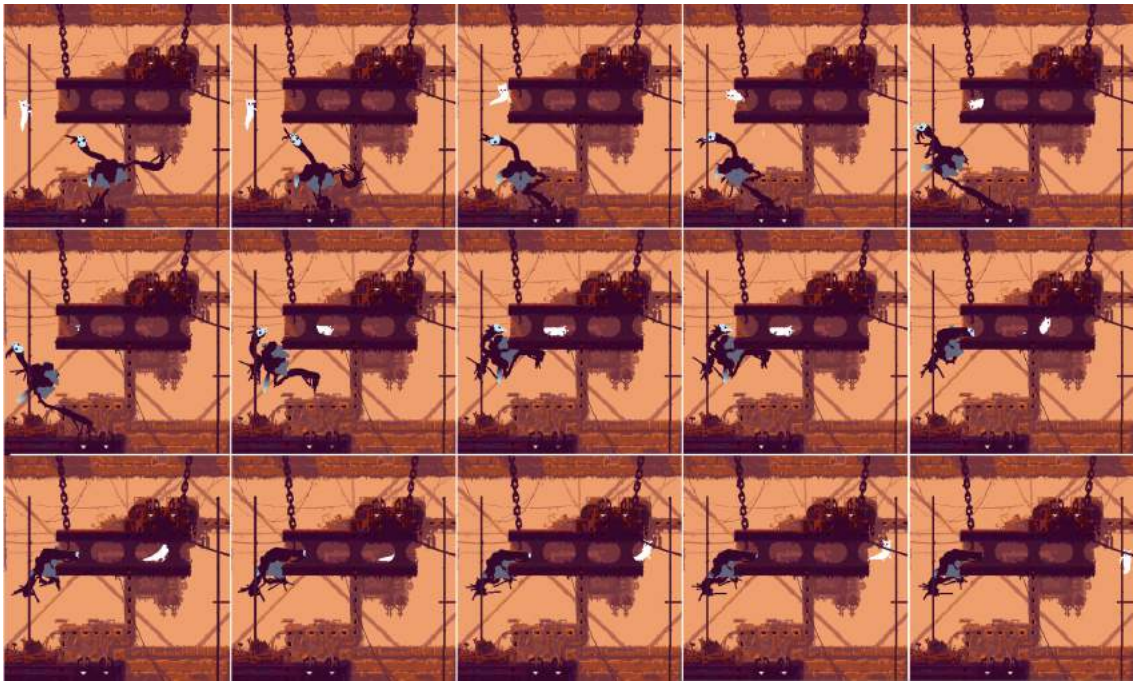

Figure 3 – Rain World game that uses procedural animation to bring characters to life to react to the terrain.
Source: Author (2021)

2.2. Procedural Animation

The following subsections will present the concepts of the methods analyzed in this conference paper.

2.2.1. Forward Kinematics

Before starting the discussion on the Forward Kinematics and Inverse Kinematics, it is crucial to clarify the use of some terms, since the concept of both is related to their use in robotics (Figure 4), in addition to animation, namely:

- Joint: Also known as axes, joints are moving parts of a robot, or 3D model, that cause motion between two links;
- Link: Inflexible part that connects two joints. It is also known as Bone in the animation field;
- Kinematic Chain: Set of joints and their links that compose a system;
- Base: Immobile part that begins the kinematic chain;
- End-Effector: The last part of a kinematic chain. Within character animation, the end-effector is commonly the hand or foot of the model.

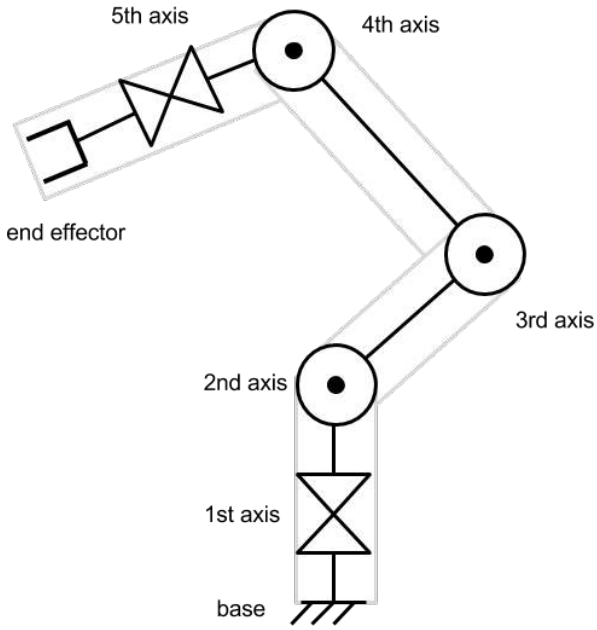

Figure 4 – Robotic arm schematic.
Source: Author (2021)

Kinematics is the study of the geometry of motion. It is an area of physics that studies the motion of bodies and objects without regard to the forces that acted on the object to cause it to move (BEGGS, 1983).

Forward Kinematics (FK), in turn, is the use of kinematics to compute the end-effector position based on the parameters of its kinematic chain. The concept of FK may be intuitive to animators and those familiar with Game Engines, since when moving and rotating objects that are related, the result is effectively the application of the FK concept, as exemplified in Figure 5. It is important to note that after the rotation is applied, all child objects have their rotation and position based on the rotation and position applied to their ancestor.

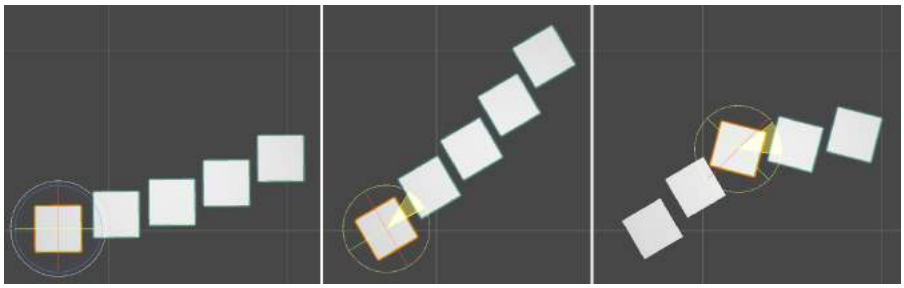
Figure 5 – Illustration of the FK concept on related objects in Unity.
Source: Author (2021)

2.2.2. Inverse Kinematics (IK)

By understanding the concept of the Forward Kinematics, it's possible to comprehend Inverse Kinematics as the inverse process (KUCUK & BINGUL, 2006): in FK, the result is the end-effector position from the joints angle, while in IK, given a coordinate at which you want to position the end-effector, the result is the joints angle of the kinematic chain. Figure 6 illustrates this concept.

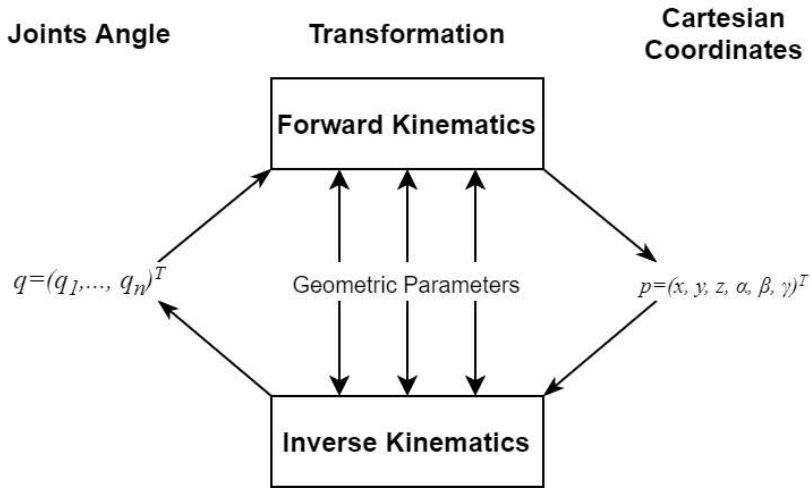

Figure 6 – Comparison of Forward Kinematics and Inverse Kinematics.
Source: Author (2021)

Games and 3D animation uses the IK method to connect characters to the game world. An example of this is adjusting the model to be realistically on the ground. Figure 4 made use of this technique.

There are several solutions to the Inverse Kinematics problem, from analytical to numerical solutions. During the application of the IK technique, we will use the heuristic solution known as FABRIK (ARISTIDOU & LASENBY, 2011). The referring section explains the theory behind this technique, while the related work contains the step-by-step application (CANTÃO, 2021).

2.3. Unity

Unity is a 3D engine for creating games and products and has even been adopted by industries outside of video games, such as architecture, engineering, film, and automotive. Because it is free, powerful, and easily accessible, with hundreds of educational contents, its choice becomes ideal for the first application of procedural animation techniques.

As we will see in section 3, each method has a subsection containing prerequisites for implementing each technique. The reader must have a certain degree of knowledge and experience with Unity, varying according to the subsection. Thus, this section clarifies the terms used regarding the Unity engine since they will be used extensively during the application of the methods.

2.3.1. Unity Editor

This subsection presents the Unity engine editor (Figure 7), with an explanation of each item that makes up this editor below:

A. The Toolbar, located at the top, contains tools for manipulating the gameObjects (detailed further in subsection 2.3.2) contained in the Scene window (D), as well as play, pause, and step (to advance one frame) buttons;
B. The Hierarchy, located on the left, is a hierarchical representation of each gameObject present in the Scene. It reveals the relationship between gameObject, which, by having a parent-child relationship, is related to the concept of Forward Kinematics, seen in subsection 2.2.1;
C. The Game View simulates the rendered look of the game through the camera;
D. The Scene View allows you to visually navigate through the game scene, serving as a stage for the gameObjects;

E. The Project window stores the library of assets and scripts.

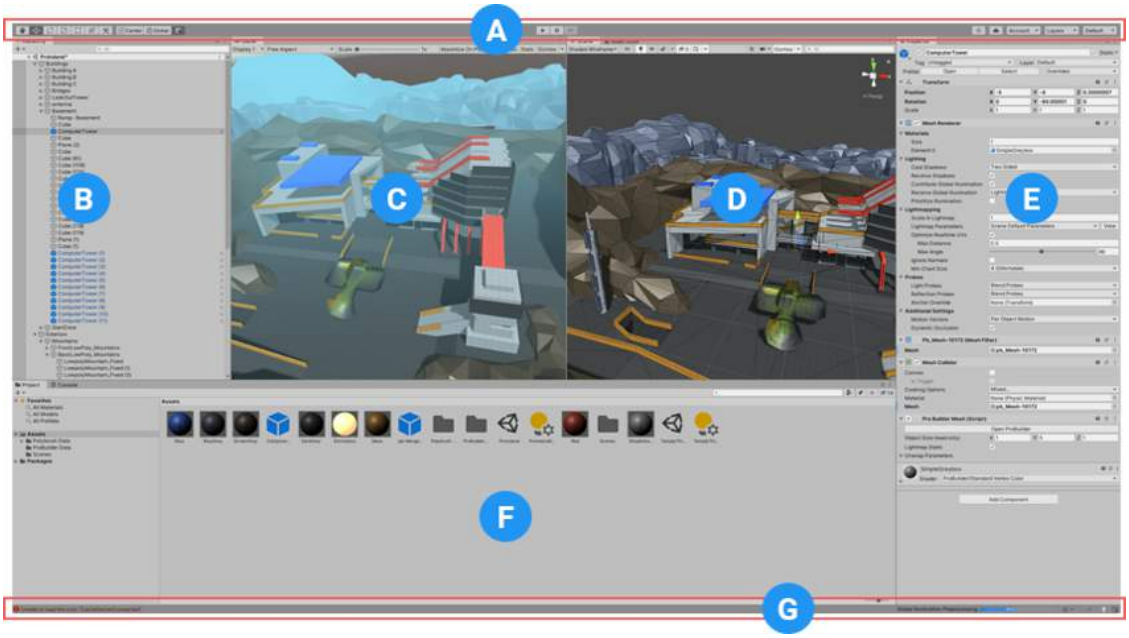

Figure 7 – Comparison of Forward Kinematics and Inverse Kinematics.
Source: (UNITY, Unity's interface Documentation, 2021)

2.3.2. Gameobject and other important terms

This section discusses the main Unity-related terms (UNITY, Unity's important classes, 2021) used during the recommendations for using procedural animation methods (seen in Section 3).

- **Gameobjects:** Unity's fundamental objects, which can represent characters, cameras, scenery, etc. Their functionality is composed of their components (scripts, for example);
- **Transform:** Component of a Game Object that allows the manipulation in space, with its position, rotation, and scale, besides its relation with other transforms;
- **Vector:** Classes to express and manipulate vectors;
- **Quaternion:** Class that provides methods to create and manipulate quaternions, representing absolute or relative rotations;
- **Collider:** Invisible component that defines the shape of a game object in terms of collision. It is essential for collision detection;
- **Rigidbody:** Component that makes the game object act under the control of physics and receives forces to move realistically.

3. Recommendations for the use of Procedural Animation Methods

This section presents the recommendation guide for the procedural animation implementation methods. Thus, that are four sections for each technique, namely:

- **What it is?:** This section introduces the technique and its basic concepts to the reader;
- **When?:** Illustrates scenarios in which the technique can be applied, contrasting with traditional methods of animation;
- **How:** Discusses the theoretical application and presents the algorithm/heuristic;

- **Real-world Use:** Illustrates a real-world use scenario where the technique is applied.

For reasons of conference page limitations, we selected the main criteria for understanding procedural animation methods. For additional information, such as prerequisites, the actual step-by-step application, complexity, and technical limitations, access the term paper (CANTÃO, 2021).

3.1. Methods choice

As discussed in section 2.1, there are numerous areas for applying procedural animation in games. Because of this, it is natural that there is a wide range of techniques. We chose the methods presented in this paper based on two main criteria: i) Relevance, in other words, their applicability within the industry and effectiveness of the method; and ii) Ease of implementation since this is a step-by-step approach, more complex techniques would require more knowledge from the reader.

3.2 FABRIK

The following subsections deal with the guidelines for using the FABRIK method.

3.2.1 What it is?

As in many computational problems, there are numerous solutions for Inverse Kinematics. It is not only important the problem correctness, but also its time complexity and ease of implementation.

Forward And Backward Reaching Inverse Kinematics (FABRIK) is a heuristic method for solving the Inverse Kinematics problem, presented by Andreas Aristidou in his paper (ARISTIDOU & LASENBY, 2011).

The idea of the algorithm, explored further in section 3.2.3, is to treat the problem of finding the position of each joint as a problem of finding a point on a line.

3.2.2 When?

It is possible to use the FABRIK method for most problems involving Inverse Kinematics. Nevertheless, it is an efficient solution, as can be seen in the comparison to other methods in Figure 8, while producing realistic visual results in the comparison in Figure 9.

	Number of Iterations	Matlab exe. time (*sec*)	Time per iteration (in *msec*)	Iterations per second
FABRIK	15.461	0.01328	0.86	1164
CCD	26.308	0.12356	4.69	213
Jacobian Transpose	1311.190	12.98947	9.90	101
Jacobian DLS	998.648	10.48051	10.49	95
Jacobian SVD-DLS	808.797	9.29652	11.50	87
FTL	21.125	0.02045	0.97	1033
Triangulation	1.000	0.05747	57.47	21

Figure 8 – Average results for a single kinematic chain with 10 joints.
Source: (ARISTIDOU & LASENBY, 2011)

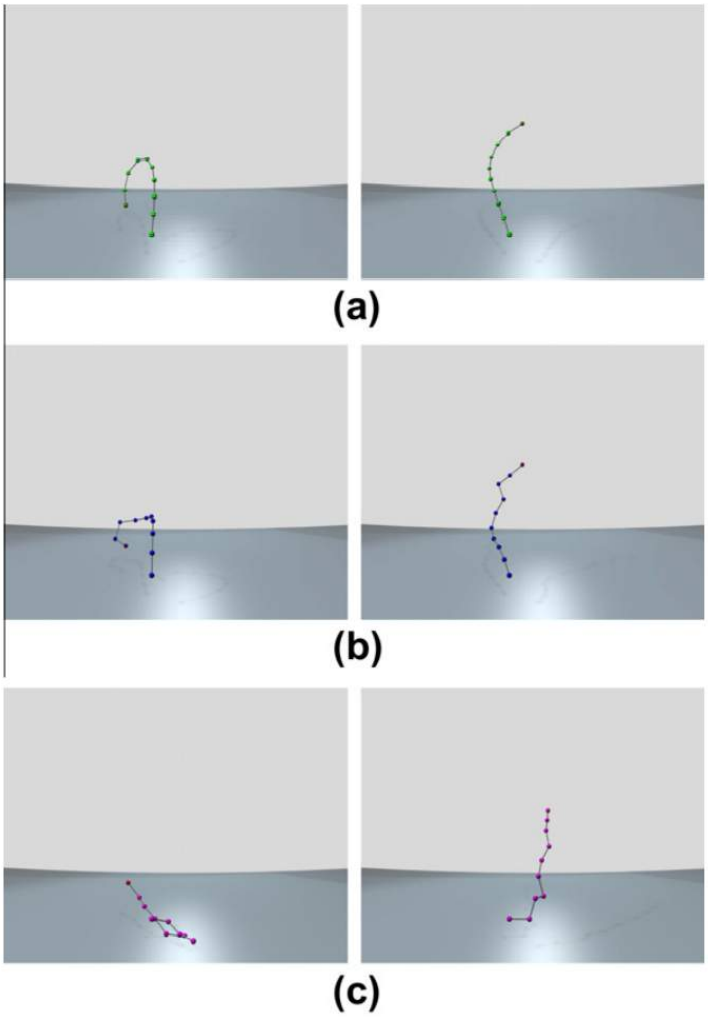

Figure 9 – Visual comparison between a) FABRIK, b) CCD, c) J. DLS.
Source: (ARISTIDOU & LASENBY, 2011)

3.2.3 How

As the name implies, FABRIK has two main parts: the Forward step and the Backward step. However, it is possible to skip the algorithm and get to a result quickly if the target is outside the range of the chain of joints.

To check if it is possible to reach the target, we add up the length of the Links, and if the result is less than the distance from the Start to the Target, we draw an imaginary line from the Start point to the Target. We then move each Joint to the imaginary line, preserving the length of each link. As a result, we will have the end-effector as close as possible to the target in just one iteration. Figure 10 demonstrates this process.

Otherwise, the target is within the range of the end-effector. We will discuss the main steps of FABRIK next.

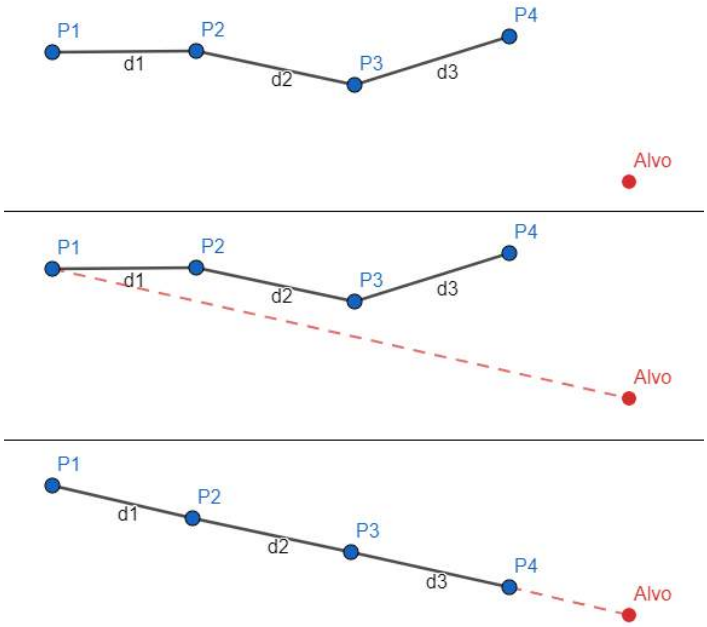

Figure 10 – Visual demonstration of the algorithm if the target is out of range.
Source: Author (2021)

3.2.3.1 Forward Step

In this step, the goal is to advance the end-effector to the target without changing the length of the links. To do this, we move the end-effector (P4) to the target position and draw a line from its new position to the previous Joint (P3), as shown in frames 1 and 2, respectively, of Figure 11.

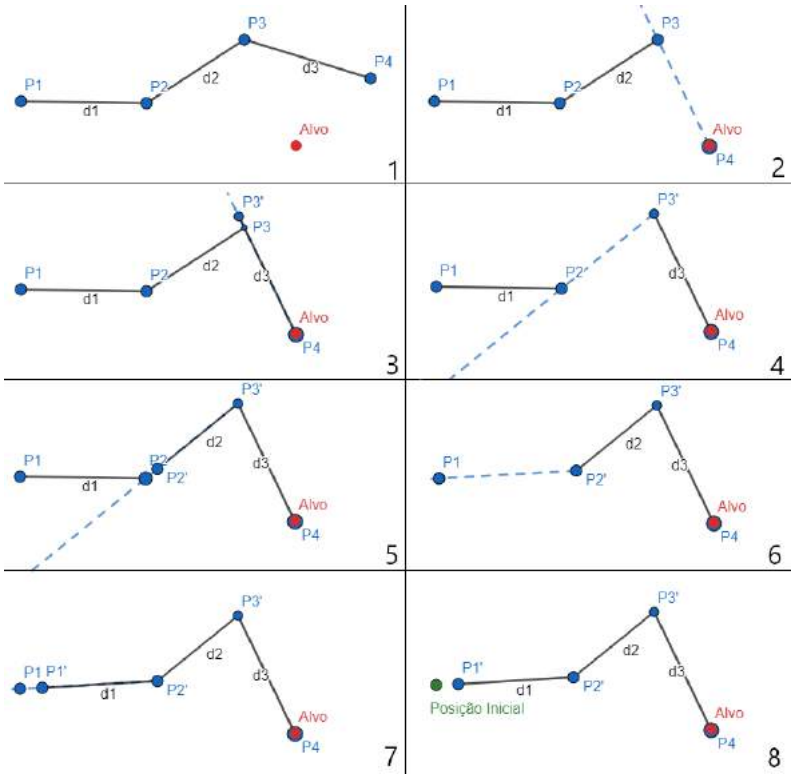

Figure 11 – Visual demonstration of the algorithm in the Forward step.
Source: Author (2021)

We move P3 along this line so that the length between P3 and P4 (d3) remains the same. Thus, giving rise to the point P3', as shown in Table 3 in Figure 12. With P3' in its position, we will do the same with the previous Joint (P2). Then, we draw a line between P3' and P2, moving P2 on the line to keep its length (d2). Similarly, we do this process until all links maintain the correct distance.

Table 8 in Figure 11 shows the final result of the Forward step. Note that P1' is no longer in its initial position. Imagining that the root joint (P1) is the base of a robotic arm, it would not be correct to move this point from its initial position. To solve this problem, we will look at the Backward step.

3.2.3.2 Backward Step

In this step, the goal is to return the base to the starting point in a very similar but inverse way to the previous step. We start by moving the base (P1') to the initial position and draw a line from its new position to the rear Joint (P2'), as shown in Table 2 of Figure 12.

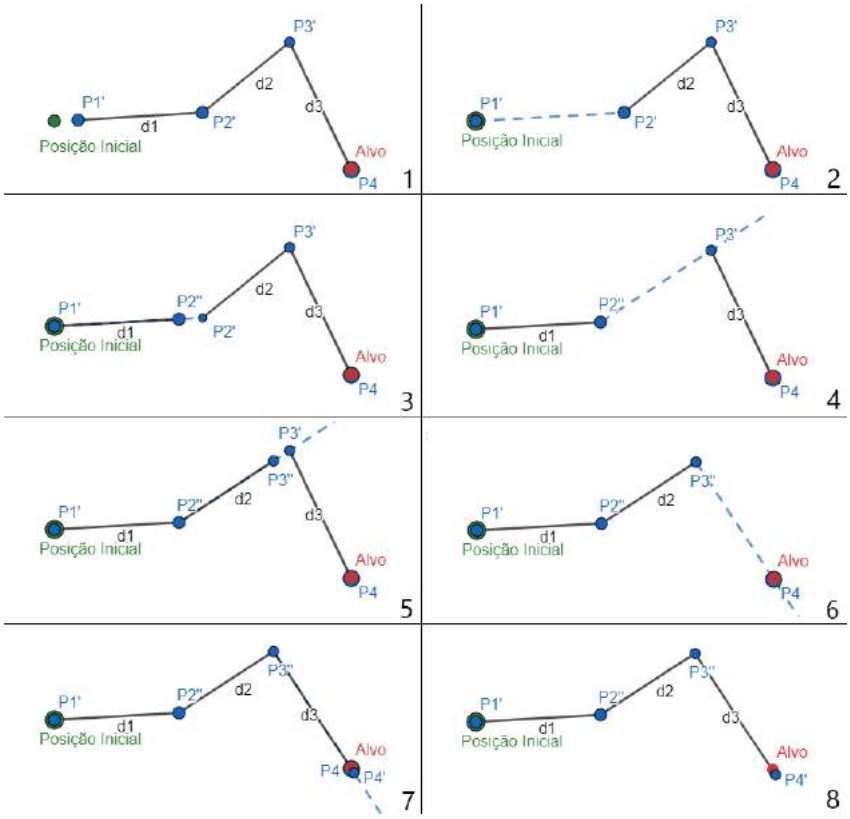

Figure 12 – Visual demonstration of the algorithm in the Backward step.
Source: Author (2021)

We move P2 along this line so that the length between P1' and P2' (d1) remains the same, giving rise to the point P2" shown in Table 3 of Figure 12. With P2" in its position, we will do the same with the subsequent Joint (P3'), we will draw a line between P2' and P3", displacing P3' on the line to keep its length (d2). Similarly, we do this process until all links maintain the correct distance, as in the previous step.

As seen in table 8 in Figure 12, the end of this step marks one iteration of the FABRIK algorithm, making the end-effector closer to the Target. Each subsequent iteration of the algorithm will reduce the distance from the target by setting an error margin. It's possible to achieve a good result with relatively few iterations.

3.2.3.3 Pseudocode

Following the logic described above, we present the pseudocode in Figure 13.

```
input : As posições do joints p_i para i = 1, ..., n,
        a posição do alvo t e
        a distancia entra cada joint d_i = |p_{i+1} − p_i| para i = 1, ..., n.
output: As novas posições dos joints p_i para i = 1, ..., n − 1.
initialization;
if target is in range then
    for i ← 1 to maxIteration by 1 do
        if distance to goal < offset then
            break;
        else
            forwardReach();
            backwardReach();
        end
    end
else
    stretchToTarget();
end
```

Figure 13 – Visual demonstration of the algorithm in the Backward step.
Source: Author (2021)

If the target is within range, we will do the forward and backward steps until we reach the maximum number of iterations, or the end-effector has reached the target. Otherwise, we will use the *stretchToTarget* method, following the idea shown in Figure 10.

The pseudocode of Figure 14 shows the *stretchToTarget* method in more detail. Where *r* is the distance between the target *t* and the initial position of the base *s*. We iterate from the second joint and assign its position to the previous joint position towards the target *t*, keeping the distance *d* from the link.

$$\mathbf{r} = \|t - s\|$$
$$\textbf{for } i \leftarrow 1 \textbf{ to } n \textbf{ by } 1 \textbf{ do}$$
$$\quad \mathbf{p}_i = p_{i-1} + (r * d_{i-1})$$
$$\textbf{end}$$

Figure 14 – *stretchToTagert* method pseudocode.
Source: Author (2021)

The *forwardReach* and *backwardReach* methods also follow the same idea presented earlier. In the pseudocode presented in Figure 15, we match the end-effector p_n to the target position *t*. With this, we iterate from the penultimate joint to the root joint, assigning the previous joint p_{i-1} the sum of the position of the current joint p_i, and the direction r_i by keeping the distance from the link d_{i-1} through distance multiplication.

$$
\begin{aligned}
&\mathrm{p}_n = t \\
&\textbf{for } i \leftarrow n-1 \textbf{ to } 1 \textit{ by } -1 \textbf{ do} \\
&\quad \mathrm{r}_i = |p_{i-1} - p_i| \\
&\quad \mathrm{p}_{i-1} = p_i + (r_i * d_{i-1}) \\
&\textbf{end}
\end{aligned}
$$

Figure 15 – *forwardReach* method pseudocode.
Source: Author (2021)

Similarly, in the backwardReach method shown in the pseudocode in Figure 16, we equalize the root joint p_1 to the position of the initial s. With that, we iterate in ascending order until the penultimate joint p_{n-2}, assigning to the subsequent joint p_{i+1} the sum of the position of the current joint p_i, and the direction r_i by keeping the distance from the link d_i through distance multiplication.

$$
\begin{aligned}
&\mathrm{p}_1 = s \\
&\textbf{for } i \leftarrow 1 \textbf{ to } n-2 \textit{ by } -1 \textbf{ do} \\
&\quad \mathrm{r}_i = |p_{i+1} - p_i| \\
&\quad \mathrm{p}_{i+1} = p_i + (r_i * d_i) \\
&\textbf{end}
\end{aligned}
$$

Figure 16 –backward*Reach* method pseudocode.
Source: Author (2021)

3.2.3.4 Real-world Use

Since it is a generalist algorithm within the Inverse Kinematics problem, we will, in the following sections, make more specific examples that will contain the use of this method in real situations.

3.3 Multiple End-Effectors FABRIK

The following subsections deal with the guidelines for using the Multiple End-Effectors FABRIK method.

3.3.1 What it is?

This technique is an expansion of the FABRIK algorithm, presented in subsection 3.2. The method allows the ability to handle more than one end-effector and, consequently, more than one target.

As Figure 17 illustrates, adding multiple end-effectors results in a number of chains equal to the number of end-effectors available. A greater number is possible if there are intermediate chains. When a Joint has more than one path or link, we call it a sub-base. The sub-base will indicate the beginning of a new chain.

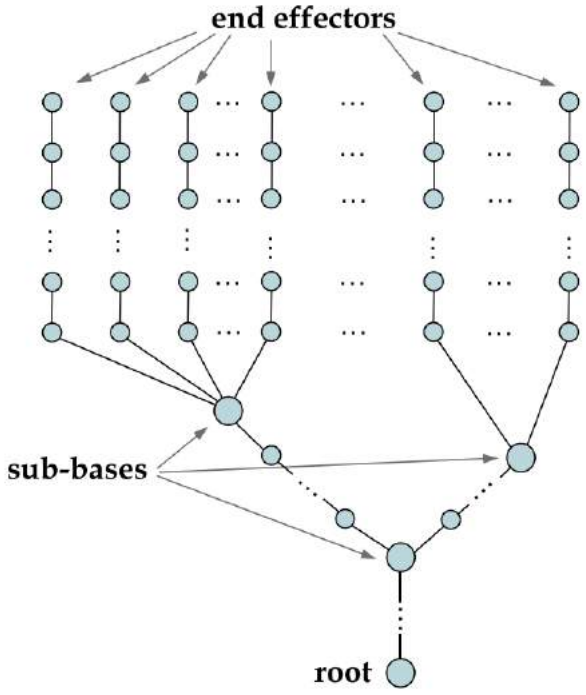

Figure 17 – Multiple end-effector kinematic chain illustration.
Source: (ARISTIDOU & LASENBY, 2011)

3.3.2 When?

In isolation, the simple FABRIK technique discussed in subsection 3.1 is sufficient to solve the Inverse Kinematics problem. However, in the real world, we have more complex models, such as multibody models.

The human body is an example of a multibody model made up of several kinematic chains. We can control each of them separately, but overall this will affect the body as a whole. Because of this, we need this technique to solve problems involving both multiple end-effectors and multiple targets (control points).

3.3.3 How

To illustrate how FABRIK works with multiple end-effectors, we will use a body with three kinematic chains, as shown in the first frame of Figure 18.

As in the FABRIK algorithm presented in the previous section, this method also has two parts. In the first part, we will apply the FABRIK algorithm starting from each end-effector towards the root joint, as shown in frames 2, 3, and 4 of Figure 18.

After that, for each subbase we will have the number of positions equal to the number of end-effectors of its chain. Since Joint D has two chains, it will have two positions resulting from the application of the algorithm (D_{sub1} and D_{sub2}).

Similarly, Joint B with three strings will result in the positions Bsub1, Bsub2, and Bsub3. The final position of each sub-base will be the centroid of all its positions, as illustrated in tables 5 and 6.

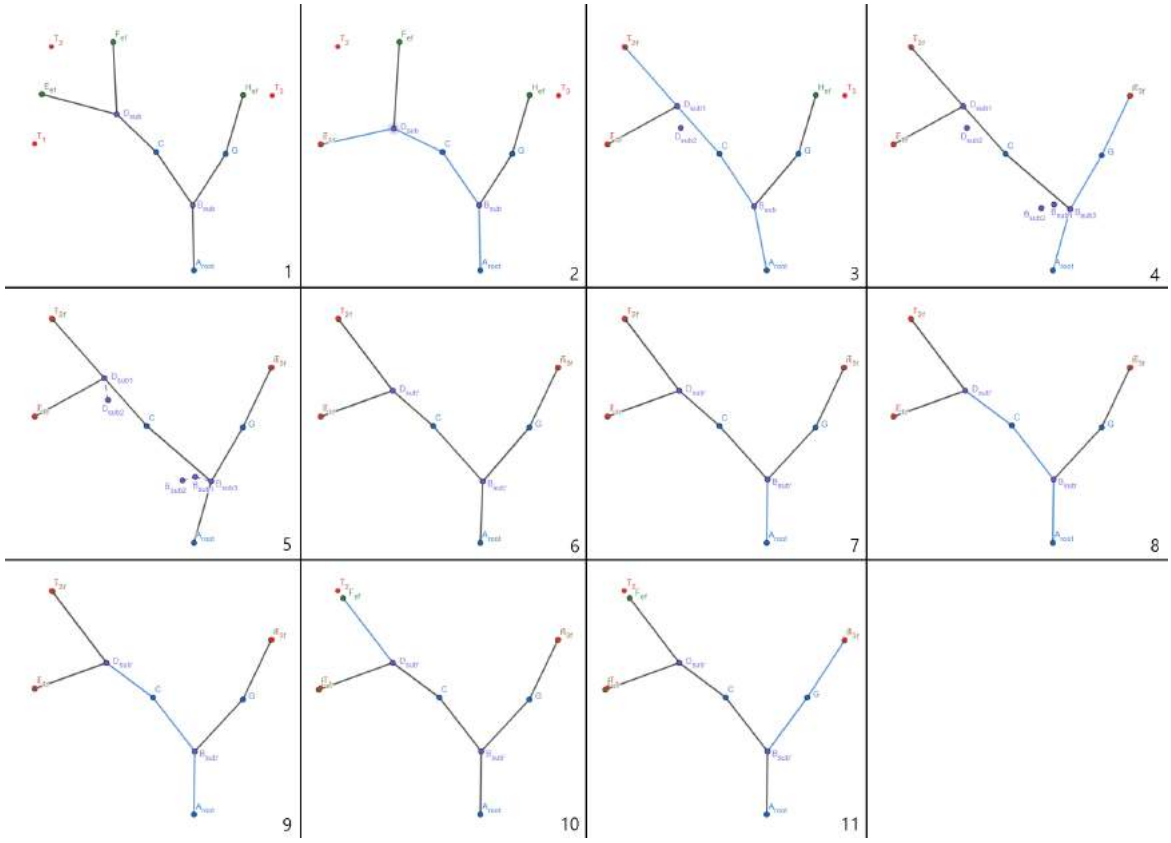

Figure 18 – "Multi-FABRIK" method step-by-step representation.
Source: Author (2021)

The second part of the algorithm will go in the reverse direction. Starting from the root joint we will apply FABRIK to each sub-base, as represented in tables 7, 8, and 9. After updating all sub-base chains, we apply the algorithm for each end-effector to its sub-base, as shown in charts 10 and 11. We repeat this process until either all end-effectors have reached their targets or there are no significant changes between their previous and current position.

To simplify the problem, we will discuss the multiple end-effectors method using only one sub-base. The reason is that working with more than one sub-base requires the use of algorithms with recursive features, which would turn more difficult to understand.

As can be seen in Figure 19, the algorithm has three parts:

1. **RootChainReach**: corresponds to the first part of the algorithm discussed earlier, where we start by applying FABRIK to each end-effector;
2. **SubBaseUpdate**: after applying the *rootChainReach* method, as seen in Figure 19, there will be N positions for the subbase equal to the number of end-effectors present. Here we will update its position by calculating the centroid of all generated positions;
3. **SubBaseChainReach**: corresponds to the second part of the algorithm discussed above. We start by applying FABRIK to each end-effector to its closest sub-base, using the updated value from the previous method.

```
input  : As cadeias FABRIK
output: As novas posições para cada cadeia FABRIK
initialization;
rootChainReach();
subBaseUpdate();
subBaseChainReach();
```

Figure 19 – Pseudocode of one iteration of the FABRIK algorithm for multiple end-effectors with a single sub-base.
Source: Author (2021)

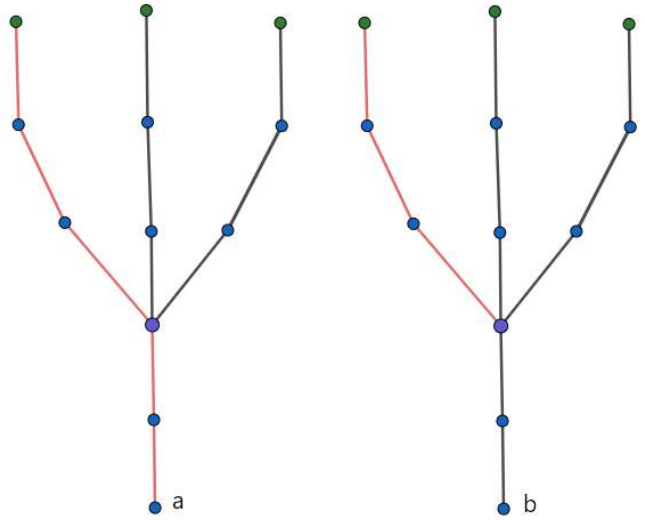

Figure 20 – a) represents the "rootChainReach" method range, while b) represents the "subBaseChainReach" range.
Source: Author (2021)

In the *rootChainReach* method, described in the algorithm in Figure 21, we iterate over each chain and apply the same logic as in the simple FABRIK discussed in section 3.2. However, instead of updating its position directly, we store it in a variable (hence the *compute* prefix). We will also store the position of the subbase for each iterated chain.

```
for each chain in chains do
    subPosition = [];
    if chain target is in range then
        chain.computeForward();
        chain.computeBackward();
    else
        chain.computeStretchToTarget();
    end
    subPosition = posição da subbase da chain;
end
```

Figure 21 – "RootChainReach" method pseudocode.
Source: Author (2021)

In the *subBaseUpdate* method, described in the algorithm of Figure 22, we will use the data of the subbase positions collected in the algorithm of Figure 21 and calculate its new position from the centroid of these positions. We will then apply the FABRIK algorithm (forward and backward) and update the joints' position.

```
subBase.positions.last() = CalculateCentroid();
subBase.ComputeForward();
subBase.ComputeBackward();
subBase.ApplyComputed();
subBase.UpdatePositionsArray();
```

Figure 22 – ”SubBaseUpdate” method pseudocode.
Source: Author (2021)

In the subBaseChainReach method, described in the algorithm of Figure 23, we will first update the position of each joint. After that, in a similar way to the algorithm in Figure 21, we will apply the simple FABRIK algorithm, but with a modified version of the method that will limit the reach to the subbase. At the end of each iteration, we will apply the change to each joint.

```
for each chain in chains do
    chain.UpdatePositionsArray();
    if chain target is in subbase to end-effector range then
        chain.computeForwardSubBase();
        chain.computeBackwardSubBase();
    else
        chain.computeStretchToTargetSubBase();
    end
    chain.applyComputedSubBase();
end
```

Figure 24 – ”SubBaseChainReach” method pseudocode.
Source: Author (2021)

3.3.4 Real-world use

Similar to the simple FABRIK, the heuristics presented here serve as a basis for realizing many animation techniques involving Inverse Kinematics. As discussed in section 3.3.2, this method is especially interesting for multi-body models, such as human characters, as can be seen in Figure 25.

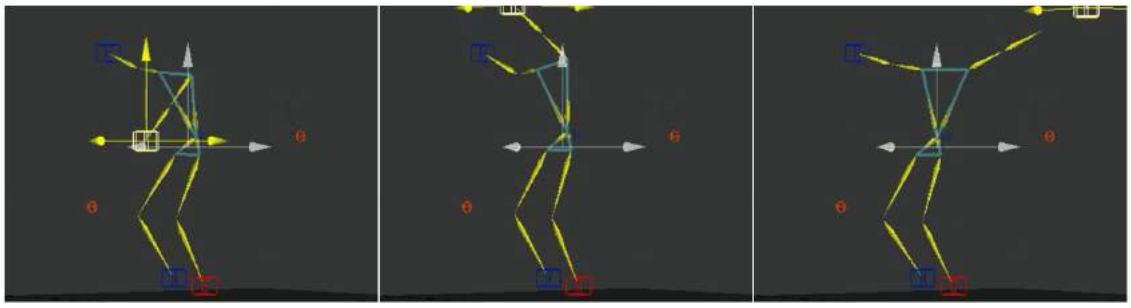

Figure 25 – Human skeleton in the FABRIC Engine 2.
Source: Author (2021)

Figure 25 demonstrates a real-world application using techniques involving the FABRIK method for multiple end-effectors. As we move the left hand, there is a change throughout the body, even though other control points exist.

3.4 Foot Inverse Kinematics

The following subsections deal with the guidelines for using the Foot IK method.

3.4.1 What it is?

In this section, we will apply the Inverse Kinematics (IK) concepts to real-world use scenarios. We will use the Foot IK technique on a humanoid model to improve the realism of its feet positioning relative to the terrain.

3.4.2 When?

Whenever you want to increase the realism of a character or creature concerning interactivity with the world around it, Foot IK is always welcome. In Figure 26, we can see the difference in the same scene with the technique on and off. Small details like this help build the immersion of a game.

Figure 26 – Example of the Foot IK technique on and off in *The Elder Scrolls Online* game.
Source: Author (2021)

3.4.3 How

Once we have an IK solution, the implementation of Foot IK comes down to moving the target to the correct height from the ground. To add even more realism, we also rotate the foot close to the contact surface. Figure 27 illustrates this idea.

```
input  : Posição do alvo
output: Nova posição do alvo
initialization;
if ground is in range then
    ikPosition = groundLevel;
    ikRotation = rotateToGroundNormal;
end
```

Figure 27 – Foot IK pseudocode.
Source: Author (2021)

The first step is to find the surface position to which we will move the foot. For this, we could use *Raycasts*. A ray works similarly to a vector, where we define a starting point and a direction. If its length is not defined, the line becomes a ray.

A *raycast* is the use of a ray, such as a beam of light from a laser, to collect collision information, such as distance, surface normal, and the collided object itself. It is a technique often used in shooting games, as exemplified in Figure 28.

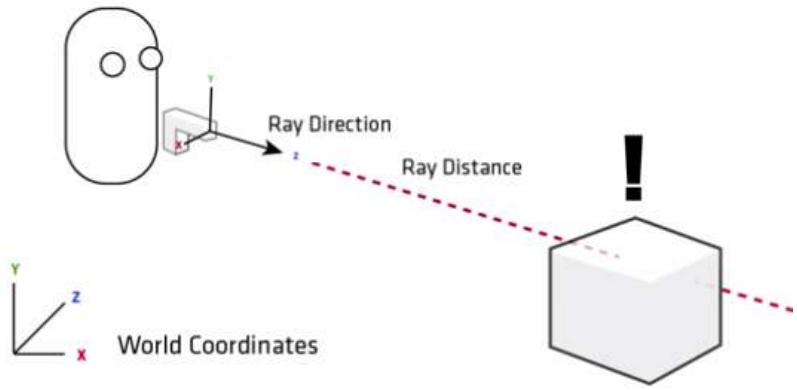

Figure 28 – Scheme of raycast used for projectiles.
Source: (MEDEIROS, 2021)

With this, we can verify the ground is in range condition. What remains is to change the target of the IK, as we saw earlier, to the position where the surface is. To add even more realism, we rotate the foot to be close to the surface using its normal, as shown in Figure 29.

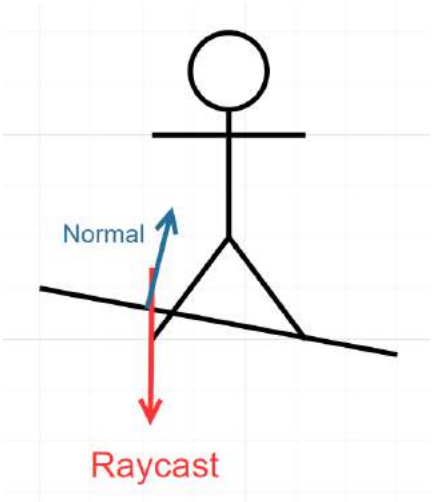

Figure 29 – Schematic demonstrating the raycast and normal vectors.
Source: Author (2021)

3.4.4 Implementation Preview

In Figure 30, we can see the comparison between the character on flat terrain (on the left) and with different terrain simulations in the images on the right.

Figure 30 – Example of recent and old games that use the Inverse Kinematics technique.
Source: Author (2021)

3.4.5 Real-world use

Both in the field of animations and games, there are several examples of media that use this technique, even becoming a solution expected by players in more modern games. Figure 31 illustrates the use in three very famous AAA games: Sekiro: Shadows Die Twice; The Legend of Zelda: The Wind Waker; and Tomb Raider, respectively.

Figure 31 – Example of recent and old games that use the Inverse Kinematics technique.
Source: Author (2021)

3.5 Hand Inverse Kinematics

The following subsections deal with the guidelines for using the Hand IK method.

3.5.1 What it is?

Just as in the previous section, when we covered a real-world application of Inverse Kinematics, here we will show another form of application: the use of IK to procedurally move a character's hands when approaching a wall.

3.5.2 When?

The Hand IK technique is very versatile, used whenever we desire to have character interactivity with objects around it. In this section, we will explore just one of many ideas of its use. For other real applications, see section 3.5.4.

3.5.3 How

This technique works in much the same way as the Foot IK problem from subsection 3.4. Once we have the IK solution, we will assign the corresponding target of the character's hand to an object. In this case, when approaching a wall, the character will procedurally place his hand on it. We will use only the right hand to illustrate.

input : Posição do alvo
output: Nova posição do alvo
initialization;
if *wall is in range* **then**
    ikPosition = closest point of wall;
    ikRotation = rotate palm to face wall;
end

Figure 32 – Hand IK pseudocode.
Source: Author (2021)

As we can see in Figure 32, we must first detect if a wall exists near the character. If so, we must verify that it is at a distance equal to or less than the arm length. For this, unlike the last section where we used *Raycasts*, we will use the *OverlapSphere* method.

The OverlapSphere method, illustrated in Figure 33, consists of returning objects that collide with a sphere, being either entirely in its area or only partially in contact.

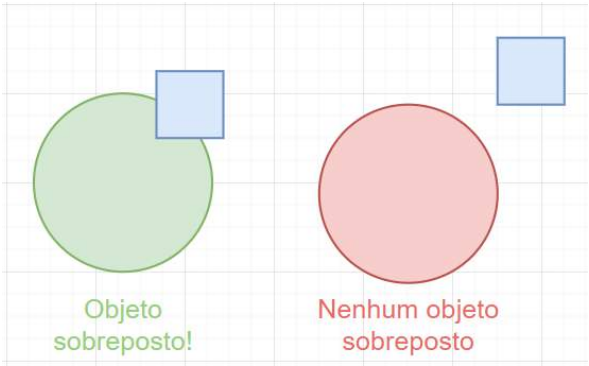

Figure 33 – 2D schematic illustrating the use of *OverlapSphere* to detect objects. Overlap in green, and no overlap in red.
Source: Author (2021)

Using the *OverlapSphere* method, we can check the condition *wall is in range*. With this, we move the hand to the closest point to the returned object. To add even more realism, we rotate the hand so that it appears to land on the wall.

3.5.4 Implementation Preview

Figure 34 illustrates the animation of the character moving closer and closer to the wall, resulting in different poses.

Figure 34 – Example of the Hand IK implementation result as the character moves closer to the wall.
Source: Author (2021)

3.5.5 Real-world use

The game Red Dead Redemption II is an extraordinary example of using this technique in a variety of situations. Figure 35 demonstrates the interaction of a player fetching supplies, where there is a whole process to open each compartment and realistically grab each object. Such a feat would be impossible to animate in time for all possible situations.

Figure 35 – Red Dead Redemption II. Animation in which the player opens a cupboard and individually collects each item, depending on the player's choice.
Source: Author (2021)

3.6 Interpolation Animation with Real-time Blendtree

The following subsections deals with the guidelines for using the Real-time Blendtree Interpolation method.

3.6.1 What it is?

Interpolation consists of estimating new data from existing data. Animation by interpolation follows the same idea, where we can create new animations based on two or more keyframes, as illustrated in Figure 36.

Figure 36 – Illustration of frames generated from already defined keyframes.
Source: Author (2021)

The interpolation technique is nothing new. When we use 3D modeling and animation software (such as Blender), we define keyframes and let the computer do the heavy lifting: generate new frames between the defined keyframes. However, the fascinating thing is to think of applying this technique procedurally. It is possible to generate new animations based on curves in real-time, using just a few keyframes. More than interpolating between keyframes, it is also possible to interpolate between animations generated procedurally.

In Unity, the blendtrees allow us to do precisely these combinations of animations, using variables that control this mixing, as we will see in the following sections.

3.6.2 When?

Animating using conventional means, although accurate, takes lots of effort. Because of this, this technique fits very well in situations where animating in a conventional way would be impractical because of time or in cases of simpler animations. The interesting part is that this technique is very versatile and can be used in countless scenarios, from characters to object animation or visual effects.

3.6.3 How?

Unity provides us with a powerful tool known as BlendTree, illustrated in Figure 37. Essentially, it is a method for combining and mixing different animations across one-dimensional or two-dimensional parameters. However, instead of using full animations, we will use single keyframes, interpolating them with different methods for creating animations.

To have more control over the interpolation, we will use runtime editable curves. The idea is to use a one-dimensional parameter in the BlendTree that will follow a curve for each animation. To do this, we will divide the curve into equidistant points by the number of animations in the BlendTree, having each point associated with a keyframe.

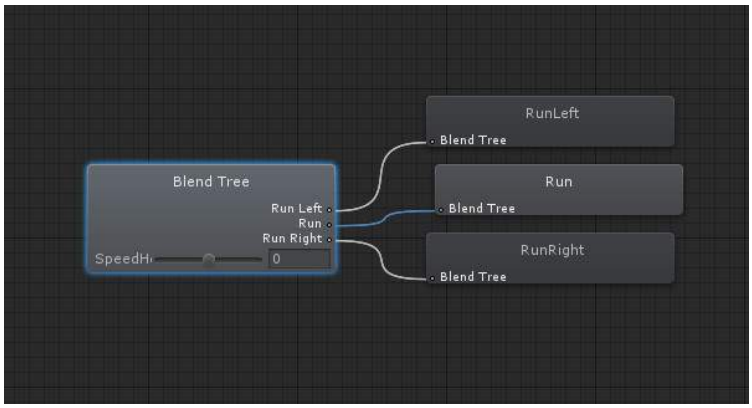

Figure 37 - BlendTree example with one-dimensional parameter.
Source: Author (2021)

3.6.4 Implementation Preview

Figure 38 shows the frames generated from the animation with cubic interpolation. It is also possible to use this technique for more complex cases. Figure 39 illustrates the mixture of two BlendTrees, one for walking animation, and one for running animation. Therefore, we can generate more frames between the frames generated between animations.


Figure 38 – Walk cycle with Cubic interpolated frames.
Source: Author (2021)

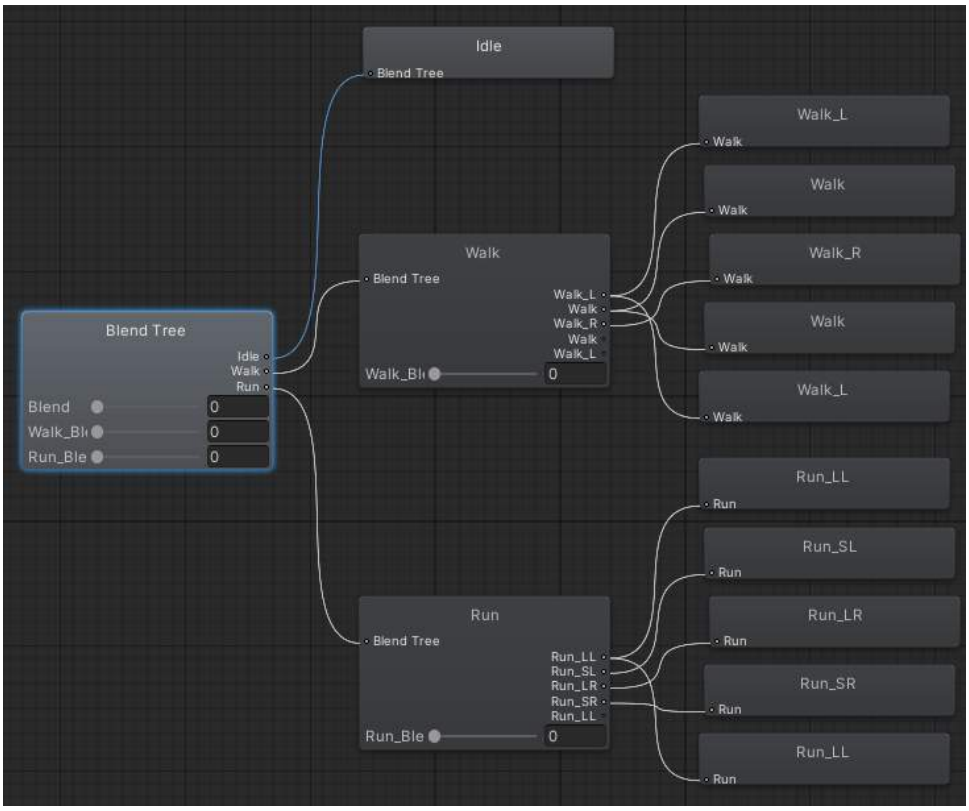

Figure 39 - Example of using multiple BlendTrees.
Source: Author (2021)

3.6.5 Real-world use

It is possible to use different curves to bring animations to life in distinctive ways. Because of this, this technique is very flexible. Its applications cover all kinds of animation, from characters to object animation or visual effects. But its main quality is efficiency, as with only a few keyframes, it is possible to create high-quality animations, which is very advantageous for smaller game studios. There is also the possibility to merge multiple BlendTrees in order to create more complex animations that depend on several variables.

The game Overgrowth, developed by only two people, is a successful example of this technique. By combining it with other animation methods, with only a few keyframes it was possible to achieve impressive levels of animation, illustrating its efficiency. As an example, we have the movement of the main character, made from only 13 keyframes, as illustrated in Figure 40.

Figure 40 - Example of keyframes used together with the technique to generate convincing animations.
Source: Author (2021)

4. CONCLUSION

This section presents the conclusion of the work. First, by highlighting an overview of the work, the contributions to the field of procedural animation, the limitations of the work, and finally, the proposals for future work.

4.1. Overview

The first section presented the introduction, starting with the work context, as well as its motivation. Then, its justification, the objectives of the work, the methodology in its realization, and its structure.

The second section presented the theoretical foundation, necessary not only to conceive the work but as a base for the practical recommendations of the following section. First, it introduced the procedural animation concept to contextualize and offer a broader vision before the application. Finally, it illustrated some more concrete theories used in techniques in section 3. It also presented crucial terms and engine concepts for the implementation.

The third section presented the recommendations for applying procedural animation methods. Methods that serve as a basis for more complex applications were first introduced, including the FABRIK algorithm and its expansion to multiple end-effectors. After this, it used the implementation of FABRIK to present the Foot IK and Hand IK methods. Finally, unlike the other methods using Inverse Kinematics, the Bleendtree Interpolation technique is presented.

4.2. Contributions

As the main contribution of this work, it is evident the construction of a guide built from the choice of procedural animation techniques based on three criteria: relevance in the gaming industry, significant return from its application, and ease of implementation.

In addition, the guide offers a step-by-step approach focused not only on the application itself but also on the reader's reflection about the implementation process. The reader can also use the guide as a future reference, as it contains sections about how the methods work, when to apply them, technique complexity, and use in real-life scenarios. Thus, the reader will have a solid foundation to understand and apply future procedural animation techniques.

4.3. Limitations

This paper exposes recommendations for multiple procedural animation techniques application, including sections that address when, how, and the complexity of the technique to be applied.

However, without an actual usage scenario for applying the techniques, the implementation nuances, such as unique difficulties of the inserted context, or alteration of the technique to fit some mechanics or limitations imposed by the game design, are lost and make it hard to create a more comprehensive guide.

In addition, the work lacks techniques that encompass other areas of procedural animation, such as particle systems, fabric and coat simulation, and rigidbodies dynamics.

4.4. Future Work

As a proposal for future work, there are several other methods of procedural animation for implementation, focusing on more areas besides character animation, such as particle systems, cloth simulation, and rigidbody dynamics.

Besides this, another proposal is the insertion of these methods presented within a real scenario, applying them to games that take advantage of the benefits and results of the techniques. This way, we can reveal the benefits of when, how, and where to use the methods.

REFERENCES

ARISTIDOU, A., & LASENBY, J. (2011). FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models* .

BEGGS, J. S. (1983). *Kinematics.* Hemisphere Publishing Corporation.

BRUDELIN, A. (1996). Procedural Motion Control Techniques for Interactive Animation of Human Figures. *Simon Fraser University*.

CANTÃO, V. (2021). *Uso de Métodos de Animação Procedural: Recomendações para Aplicação.*

GOLDENBERG, A., BENHABIB, B., & FENTON, R. (2020). A complete generalized solution to the inverse kinematics of robots. *IEEE Journal on Robotics and Automation*.

GREGORY, J. (2009). *Game Engine Architecture.* CRC Press.

KUCUK, S., & BINGUL, Z. (2006). Robot Kinematics: Forward and Inverse Kinematics.

MEDEIROS, J. (12 de July de 2021). *Raycast Schema*. Fonte: https://medium.com/@miguel.araujo/raycast-what-the-hell-is-that-6d36b3c8dd8b.

RUGGILL, J., MCALLISTER, K., & NICHOLS, R. (2012). *Inside the Video Game Industry: Game Developers Talk About the Business of Play.* Routledge.

UNITY. (19 de April de 2021). *Unity's important classes*. Fonte: https://docs.unity3d.com/Manual/ScriptingImportantClasses.html.

UNITY. (18 de April de 2021). *Unity's interface Documentation*. Fonte: https://docs.unity3d.com/Manual/UsingTheEditor.html.

WILLIAMS, R. (2009). *The Animator's Survival Kit--Revised Edition: A Manual of Methods, Principles and Formulas for Classical, Computer, Games, Stop Motion and Internet Animators.* Faber & Faber, Inc.

WOLF, M. (2008). *The Video Game Explosion: A History from PONG to Playstation and Beyond.* ABC-CLIO.

ZACKARIASSON, P., & WILSON, T. (2012). *The Video Game Industry: Formation, Present State, and Future.* New York: Routledge.