UNIVERSITY
of
GLASGOW

Computing Science

Positioning Articulated Figures

Stéphane Etienne

Submitted in fulfillement of the requirements for the degree of
Doctor of Philosophy

Acknowledgments

Many people contributed in some way or other to the work presented in this thesis. I will not be able to thank everyone as this page would certainly be too small to contain all their names. My apologies to those of you I could not mention here.

First, I wish to thank Dr John Patterson without whom this thesis would certainly not have come into existence. I also want to thank Mr Inteak Kim and Mr Yu Jinhui for all our discussions. These have had a major bearing on this work.

I want to thank everyone who helped me to prepare, or were part of, the comparative study presented in Chapter VI. I am specially grateful to Dr Stephen Brewster, Dr Mark Dunlop, Dr Phil Gray and Prof. Chris Johnson who all had a major influence on the work presented in that chapter.

I want to thank David Christie, Huw Ewans, Lorna Love, David Manlove, Michelle Montgomery and Susan Spence for agreeing to read this thesis and correct my grammar and spelling. Also, I want to thank Jean-Christophe Nebel for accepting to bind the thesis.

In conclusion I want to thank EPSRC for funding MIME (Making It Move Easily), the project for which I was employed by the University of Glasgow and which enabled me to complete this work.

Abstract

Many animation systems rely on key-frames or poses to produce animated sequences of figures we interpret as articulated, e.g. the skeleton of a character. The production of poses is a difficult problem which can be solved by using techniques such as forward and inverse kinematics. However, animators often find these techniques difficult to work with.

The work, presented in this thesis, proposes an innovative technique which approaches this problem from a totally different direction from conventional techniques, and is based on Interactive Genetic Algorithms (IGAs).

IGAs are evolutionary tools based on the theory of evolution which was first described by Darwin in 1859. They are derived from Genetic Algorithms (GAs) themselves based on the theory of evolution. IGAs have been successfully used to produce abstract pictures, sculptures and abstract animation sequences.

Conventional techniques assist the animator in producing poses. On the contrary, when working with IGAs, users assist the computer in its search for a good solution. Unfortunately, this concept is too weak to allow for an efficient exploration of the space of poses as the user requires more control over the evolutionary process.

So, a new concept was introduced to let the user specify directly what is of interest, that is a limb or a set of limbs. This information is efficiently used by the computer to greatly enhance the search. Users build a pose by selecting limbs which are of interest. That pose is provided to the computer as a seed to produce a new generation of poses. The degree of similarity is specified directly by the user. Typically, it is small at the beginning and increases as the process reaches convergences.

The power of this new technique is demonstrated by two evaluations, one which uses a set of non expert users and another one which uses myself as the sole but expert user. The first evaluation highlighted the high cognitive requirement of the new technique whereas the second evaluation showed that given sufficient training, the new technique becomes much faster than the other two conventional techniques.

For these evaluations, solutions to the problem of forward and inverse kinematics were implemented. For forward kinematics, a widget called a joint ball was used as the manipulation tool. The problem of inverse kinematics was tackled in a different manner from conventional techniques, resulting in the implementation of a fast and effective algorithm.

This work used a humanoid for the articulated figure. It is made of nineteen limbs and has thirty degrees of freedom. Volumes such as cubes, spheres and cylinders were used to flesh out the skeleton. A new technique was also designed to render cylinders effectively.

Preface

The purpose of producing poses

To animate articulated figures or robots, animation packages rely on key-frames. A key-frame describes the position or pose of the robot at a particular time step.

Consequently, much work has been devoted and is still devoted to improving posing (positioning) systems. Two main techniques are being used by these systems. These are forward and inverse kinematics. However, animators still find the task of posing an articulated figure hard.

Hypothesis

In this thesis, I argue that a completely innovative positioning system which relies on an interactive genetic algorithms type interface with direct control by the user is a more powerful interface and will allow animators to produce poses faster than conventional positioning systems.

Description of the innovative technique

Interactive genetic algorithms (IGAs) have been successfully used previously to produce abstract pictures or animated sequences of images, plants, etc. They are based on genetic algorithms which are powerful search and optimisation tools. Genetic algorithms are themselves based on the theory of evolution which was first described by Darwin in 1859.

Conventional techniques assist the animator in producing poses. With an IGA instead, the animator assists the computer in producing poses. The IGA explores the space of poses and its search for a given pose is guided by the user. However, poses are not abstract objects and animators would not produce poses by randomly exploring the space of poses, as this takes far too long. Consequently, IGAs do not offer enough control for animators to assist the computer efficiently.

As a result, a new concept was introduced to let the user specify directly what is of interest. This information is being used efficiently by the computer to greatly enhance the search. Instead of specifying the goodness of fit of a particular pose produced by the computer, the animator directly selects the joint configurations which are of interest. From this selection the computer constructs a new pose which is used as a seed to proceed with the search. The user can also specify how far the target pose deviates from its predecessor.

Verification of the hypothesis

To verify the hypothesis, it was decided to perform an evaluation. Although a licence for a posing system using forward kinematics was available when this work was being performed, there was no such licence for a posing system using inverse kinematics. As a result, and also to decrease the side effect of using a different interface, the two most common techniques were also implemented.

For forward kinematics, a widget called a joint ball, which allows users to work on two angles at the same time, was used as the sole manipulation tool.

The problem of inverse kinematics was tackled in a different manner from conventional techniques, resulting in the implementation of a fast and effective algorithm.

Since no expert users were available for an evaluation of these techniques, non expert users were used instead in the hope that results could be generalised. Unfortunately, there were too few participants and the variability between them was too big to be able to obtain significant results. However, this evaluation highlighted the fact that these implementations were not perfect but could be improved. Also, it was felt that given sufficient training, the new technique would perform better than the other techniques.

Consequently, another study was performed in which I was the sole but expert user. Such an evaluation has already been performed in the past, and since variability amongst expert users may be assumed to be less important, the hope was that results based on a single expert subject would generate useful evaluative data.

Results

The outcome of the study were as follows:

- First, it was shown that given sufficient training, the generator will indeed allow animators to pose articulated figures faster than conventional techniques
- However, there was evidence that the generator is a lot more mentally demanding an requires a lot more training than conventional systems

Contents

List of Figures

Chapter I

Introduction

1 The use of computers in the production of films and cartoons

The use of computer animation in the production of animated sequences for cartoons and films is steadily increasing. This is a potentially highly profitable area of business. Huge amounts of money are invested by firms such as DreamWorks, Time-Warner, Walt Disney to conduct research into what is still not feasible. The use of computer during the process of film production greatly aids the quality of the resulting films. For example, one just has to remember the special effects used in films before and even during the eighties. One of the first computerised effects-made films was Blade Runner which was made in 1982. It already used computerised special effects. The use of computers also opens new universes, which were difficult not to say impossible to produce without computers. In particular, Star Wars and Jurassic Park would have been impossible to produce without computers. The use of computers also increases the quality of films and cartoons. Since cartoons are of great complexity, the use of computers during the production process do not incur such a great speed-up but it allows cartoons to be edited at a much lower cost than they used to be.

2 Previous work

In the computer animation literature, the word *positioning* seems to be preferred over the word *posing*. In this thesis, these words describe the same process and were used interchangeably.

Most animation applications rely on key-framing, a technique in which poses or key-frames are specified in time and position. Thus positioning an articulated figure (that is a robot) is part of a longer process used to animate it. Since techniques to position articulated figures can also be used to animate them, most of the work has

been devoted to how to animate articulated figures. In the first part of this chapter, the techniques used to animate an articulated figure are reviewed before focusing on the techniques used to position it.

Although a lot of research has been performed in the area of computer assisted animation of articulated figures [Stu86, Gir91, WMS88, NMT85, Tha88, dJAGAN76, CCP82, Cal88], it is still an active area of work.

In this thesis and also as usually done in the computer animation literature, the term *computer animation* is used to mean *computer assisted animation*. That is, the computer is used as a tool to aid the animator to produce animations. *Computer animation* might imply that it is the computer which produces animations, with none or virtually no external help. This meaning does not apply in this work.

2.1 History of computer animation

Twenty five years ago, computer scientists started to model human figures [Csu75, dJAGAN76] to study ergonomic problems. In the seventies, real computer animation began [BS79]. Researchers started to model actors by means of spheres, cylinders and other simple drawing primitives [Kno81]. Simple interpolation techniques, based on spline mathematics and represented in parametric form [HS85, Stu84, KB84, SB85], and some motion capture techniques such as rotoscopy[1] [NMT85, Tha88] were developed during this period. To bring more interactivity to these systems, a new techniques from the field of robotics, based on kinematics, started to appear [BTT90, BT92, Zel82]. In the mid-eighties, researchers started to use the laws of physics, called dynamic systems [WMS88, Wil87a, Gir86, AGL87, Hah88, BOK80, AG85, Wil87c] to simulate motion with a great deal of realism. Even though computer animation has gained even more realism, a great deal of work still remains to produce convincing animations.

2.2 Review of computer animation

Computer animation is a vague term. In computer animation, there are animations or simulations of natural phenomena like fire, clouds, etc. In this thesis, this aspect of computer animation will not dealt with. The discussion will focus on the animation of three dimensional actors. These actors are bodies of 3D articulated rigid limbs or body parts. Therefore, we are nor interested in two dimensional animation at this stage, nor are we interested in the animation of actors having a single body part or actors having flexible body parts. Although our domain of study has been considerably narrowed, it is still too vague. Animation systems can be separated into low-level and high-level animation systems. In modern computer animation systems, these two levels may be combined together in a single interface. It results in a gain of productivity, speed and

[1] Rotoscopy involves reproducing an animation by first recording the data from a real figure producing the animation we want to reproduce. Cameras have to be used for this purpose.

effectiveness. The animators first specify the animation at a high level using tools such as scripts and adjust finer details at a later stage by the use of techniques involving direct motion control [BC89, Gre91, HS85, MC90, vO91, Wil87b, Zel82, Zel85]. In low-level animation systems, the animation is described in terms of intermediate frames, rotation angles and translations or forces and torques. In high-level animation systems, script languages, behavioural animation and task oriented animation are used instead [BS79, Cal88, CCP82, NMT85, Stu86, TP88].

2.2.1 Labanotation

Labanotation, a method of specifying animation has only been used in the early eighties. Badler and Smoliar [BS79] made a careful and thorough study of this notation. This notation was chosen after concluding that

> The digital representations of human movements involve an explosive amount of data, most of which would probably be ignored in any given investigation. Movement notation systems, designed to record human movement in symbolic form are a more fruitful area of investigation.

The purpose of Labanotation is to describe the position and trajectories of a set of points in space. It was developed in 1928 by Rudolph Laban and used in choreography (Fig. I.1). This notation appears to be well suited for choreography, the domain it was developed, but it appeared to be of limited use in computer animation. Two main drawbacks were identified. First, the script which has to be written to specify an animation is rather difficult to understand and moreover tends to become large as the animation gets long. Second (and maybe paradoxically) the resulting script is always under-specified: several different animations may be specified by using exactly the same script. This is not at all a problem in choreography where the imprecision allows the choreographers to bring their own personal touch to the final result but the need for determinism is predominant in computer animation. As a result, the Labanotation has now been abandoned in computer animation.

2.2.2 Kinematics

To animate articulated figures, forward and inverse kinematics can be used.

2.2.2.1 Forward kinematics: In forward kinematics, the animation is specified in terms of rotations and translations. These operations are applied to each joint of the body to perform a given task. The computer then calculates the necessary frames to display the animation by interpolating the given information. This technique is computationally light but requires that the animator specifies a set of rotation and translation vectors at each time step. This is usually far too much to ask of a professional animator. As a result, it is usually not used in today's animation packages.

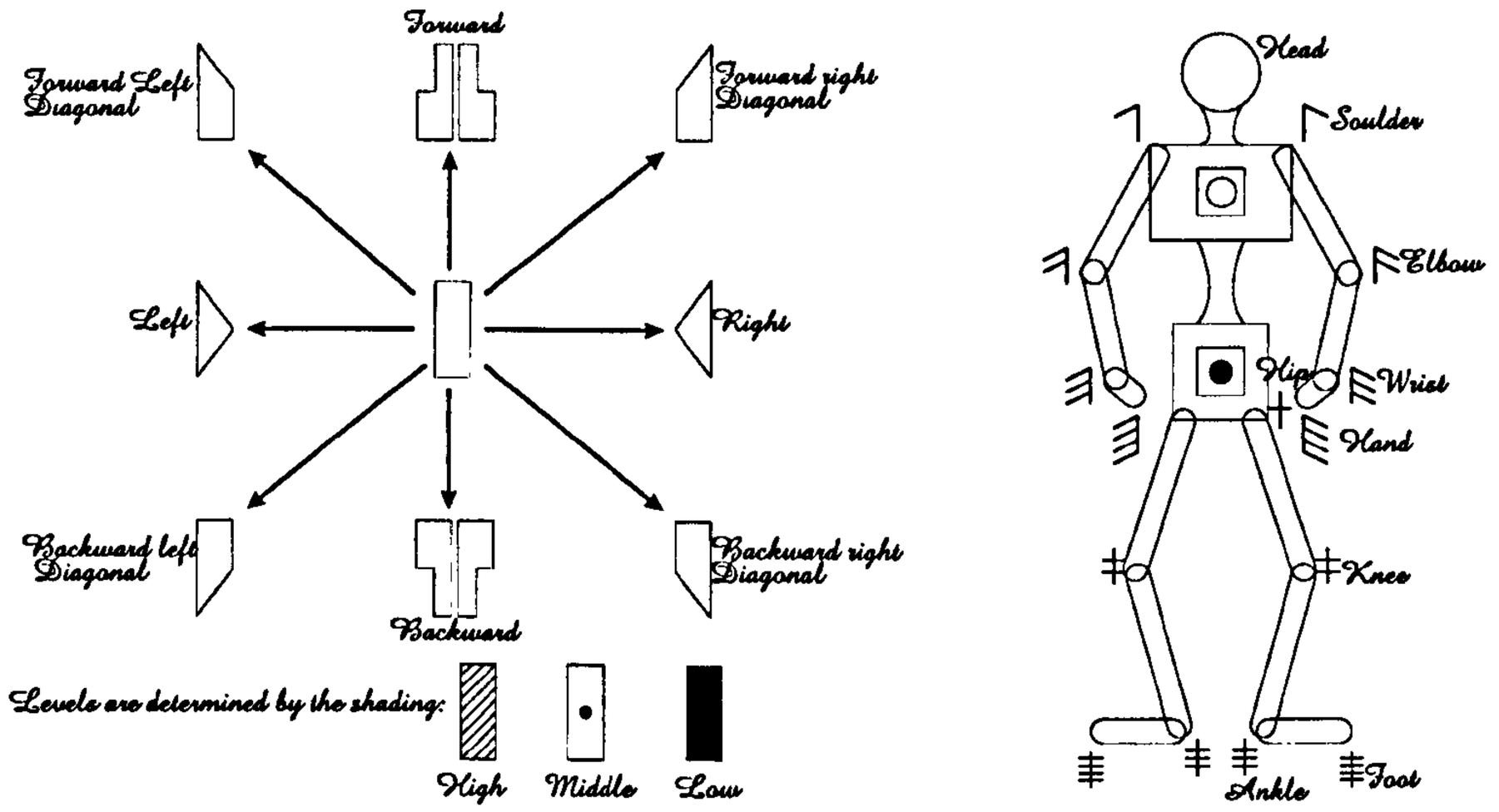


Figure I.1: Labanotation

The Labanotation is used as a script to specify the motion to achieve. A set of symbols can be specified at different time intervals.These, unfortunately, inherently lack any accuracy.

2.2.2.2 Inverse kinematics: In the animation literature, the word kinematics alone is frequently used and usually refers to the inverse aspect of kinematics. With inverse kinematics[BT92, CCP82, Cal88, Dai88, Stu86, Wil87b, KB82, Kor82, JU85, BKK+85], constraints, such as the initial positions and final positions, also called goals, of one or more body part, also called end-effectors, have to be specified (Fig. I.2). The computer will then compute the necessary rotations and translations to bring the end effectors to their required positions. Once the necessary rotations and translations have been obtained, forward kinematics is used to interpolate along the time dimension. Usually, the problem to be solved will be under-constrained, so several motions may satisfy the constraints specified by the user. Optimisation methods have been implemented to try to work out the best of these. In particular, genetic algorithms have been successfully used by Miller [MP94] and Davidor [Dav91a] to solve this problem.

2.2.3 Dynamics

Like kinematics, dynamics may be divided into two sub-techniques. These are forward and inverse dynamics.

2.2.3.1 Forward dynamics: The production of realistic animations with key-framing is still difficult, because dynamic systems use physical laws to produce animations, computers are used to simulate reality. Jane Wilhelms[Wil87c] provided the following definition for dynamics:

Figure I.2: Inverse kinematics

With inverse kinematics systems, the initial position and usually one goal to reach by one effector (but there could be more) have to be specified. Here the goal is the black ball and the end effector is the tip of the hand. Rotations of every limb is automatically calculated by the application.

Dynamics refers to the description of motion as the relationship between forces and torques acting on masses. If we treat the objects modeled in computer graphics as masses and apply forces and torques to them, we can use physics to find out the motion these masses should undergo. This motion should mimic the motion that would actually occur to such masses in the real world, hence dynamics simulates the motion, rather than just animating it.

As a result, the generated animations should be highly realistic. With forward dynamics, dynamic equations of motion which describe how masses will move under the influence of forces and torques have to be set up [Bar87, Hah88, dJAGAN76, Wil87a, Wil87c, WMS88, Wil91]. The main drawback of this method is that obviously the animator has to specify all forces and torques to apply at each body part. The equations are then solved to produce the animation. The fact that all forces interact with each other makes this process time-consuming although fast recursive formulations such as the Armstrong formulation [AG85] have been made available. To reach interactive times, some people have used simplified algorithms [vO90, Ove94]. Due to the fact the physics are simulated, it is now possible to simulate collision effects realistically [Bar87, Dai88, Hah88, MP89, MW88, Wil87b, WMS88]. The animator just has to let the computer make the computations and wait for the results.

2.2.3.2 Inverse dynamics: With forward dynamics, animators have to specify forces and torques. This is not an intuitive approach to producing animations, thus the need for inverse dynamics [Wil91, Hah88]. Like inverse kinematic systems, users are solely required to position a set of end-effectors with a set of goals to reach. The computer tries to work out the necessary forces and torques at each time step to perform the task. However, computation times needed for animating even a simple articulated figure are usually far too big for the technique to be usable. Inverse dynamics can only be used with truly simple models and over a short period of time.

Animation of articulated figures such as a humanoid for instance is well out of reach.

2.2.4 Hybrid systems

In an effort to ease the use of dynamics and to make them interactive, work has been performed on hybrid methods which use concepts like kinematics, knowledge-based systems, scripts, libraries of motions, in combination with dynamics [AGL87, ADH89, BC89, Cal88, FW88, GM85, Gir91, Gre91, IC88, vO90, RH91].

2.2.5 Rotoscopy

Rotoscopy is an old animation technique [NMT85, Tha88]. Rotoscopy involves reproducing an animation by first recording the data from a real figure producing the animation we want to reproduce. Cameras or more sophisticated devices may be used for this purpose. Since the technique is 2D based, it is not well suited to 3D animation. As a result, it is not used very much.

2.2.6 Motion capture

For the last few years, special hardware has been built to capture the motion of a human or another animal. The price of such devices is high and they are difficult to calibrate. However, when well calibrated, results are impressive and nowadays, it is the easiest and the fastest technique to animate an articulated figure. Literature in this area [BN93, J.96, HM95, HM96, MTD96, SSK96] is scarce and difficult to find. It seems that most of the research has been undertaken by private companies which prefer to keep the results for themselves as an obvious asset over competitors.

2.2.7 Key-framing

Key-framing has been one of the first techniques to be used in 3D computer animation [Stu84, KB84, SB85]. It comes directly from the schools of cartoon films [Las87, TJ81, PW94] such as the well known school of Walt Disney. In Walt Disney, when a new cartoon film has to be made, the story is story-boarded first (Fig. I.3). Then, the most experienced animators draw the most important frames. Due to their special importance, these frames are called key-frames. There is usually one key-frame every twelve frames. After this, other animators, less skilled, draw what are called breakdown frames. There are usually one breakdown frame every four frames. To finish the animation, the missing frames are drawn by even less skilled animators. These frames are called in-between frames or in-betweens for short. This stage, which is called in-betweening, is time-consuming, so it is usually sub-contracted to small firms located in parts of the world where labour is cheap but maybe not of such a good quality. Reasonably enough, early work in computer animation attempted to

automate this stage. Unfortunately, the problem is not as simple as it might sound and is a lot more intricate in 2D animation than it is in 3D animation.

Figure I.3: The making of cartoons

The making of cartoons is a complex process which in particular involves skillful artists to draw key-frames. Less skillful artists will draw the in-betweens. Key-framing systems in 3D animation comes from this concept, except that the computer replaces less skilled artists.

This technique is particularly interesting for us because it relies entirely on key-frames or poses to animate an articulated figure. Unfortunately, to produce complex animations, a large number of key-frames may have to be specified. Positioning an articulated figure is not an easy task. Although, because of its simplicity, this method is still the most widely used [BMW87, ADH89, BN88, NMT85, NTD88, Stu84, KB84, SB85].

2.2.8 Gait systems

Inverse kinematics are also used in combination with gaits to produce typical motions such as walks, runs, etc [BTT90, RH91, GM85, MZ90, Gir86]. A gait describes a sequence of positions or states which put together will perform a cyclic motion. A set of gaits are usually assembled together and synchronised to achieve periodic motion. Thus the gait of a leg representing a walking motion is described by the foot being lifted from the ground, moved forward in the air and placed back onto the ground. Once a gait has been computed, it can be easily reused. Usually, because only the position of a limb is known (such as the foot), inverse kinematics is used to compute the position of other limbs. Gaits are not limited to inverse kinematic systems. A key-framing system could also use gaits to achieve greater re-usability.

2.2.9 Motion controllers

Instead of using gait systems, motion controllers or motor controllers have also been used [RH91, vO91, MZ90, BC89, Zel82, AGL87, Zel85, Wil87b, Gre91, HWA+91, GT95]. A motion control is like a state machine which is used to animate an articulated figure to produce the desired motion. Usually, a controller operates on a single joint but some implementations are able to deal with many joints at a time [Sim94b, Sim94a].

This is a particular fruitful area for dynamic systems. Animating an articulated figure using inverse dynamics is nearly impossible for complicated figures such as a humanoid. One solution is to generate a state machine or motion controller which will decide which forces and torques to apply depending upon input parameters, the task to achieve and the current state. At the beginning, one or many controllers are randomly generated. They are evaluated and rated according to how well they performed the task. Random search is used to generate better motion controllers. This is a time consuming process so really powerful machines are required and it can still takes hours or days to compute even simple motion controllers. However, once they have been computed, they can theoretically be used whenever they are necessary. These motion controllers are reliable, often capable of handling well, totally unexpected situations.

3 Analysis

3.1 Disadvantages

All techniques used to animate an articulated figure exposed so far suffer from a few disadvantages which will be summarised here.

3.2 Kinematics and dynamics

Inverse kinematics and inverse dynamics (assuming the latter is not too computationally expensive) are good tools to edit existing motions [BT92]. They are also good at producing short motions such as grasping a chair, etc. When producing long motions such as walks, runs, etc, they have to be used in combination with techniques like gaits, motions controllers or key-framings.

3.3 Rotoscopy and motion capture

Rotoscopy and motion capture require special hardware and assume that the entities from which the motion will have to be captured do exist. As a result, these techniques are out of reach for most potential users and they have only been used to animate humanoids.

3.4 Gait and motion controllers

Gait systems require the use of another technique such as inverse kinematics or motion controllers. Finding the right motion controllers is usually slow. Furthermore, users have no control over the resulting motion. Adding constraints or using a motion editing system is the sole alternative.

3.5 Key-framing

Key-framing systems are usually simple to use. However, complex animations require many key-frames henceforth the production of key-frames must be as easy and fast as possible. Key-framing systems do not normally handle interactions with the environment. Thus, resulting motions which are not normally acceptable can be generated (Fig. I.4). As many key-frames as necessary will have to be generated to handle these types of problems.

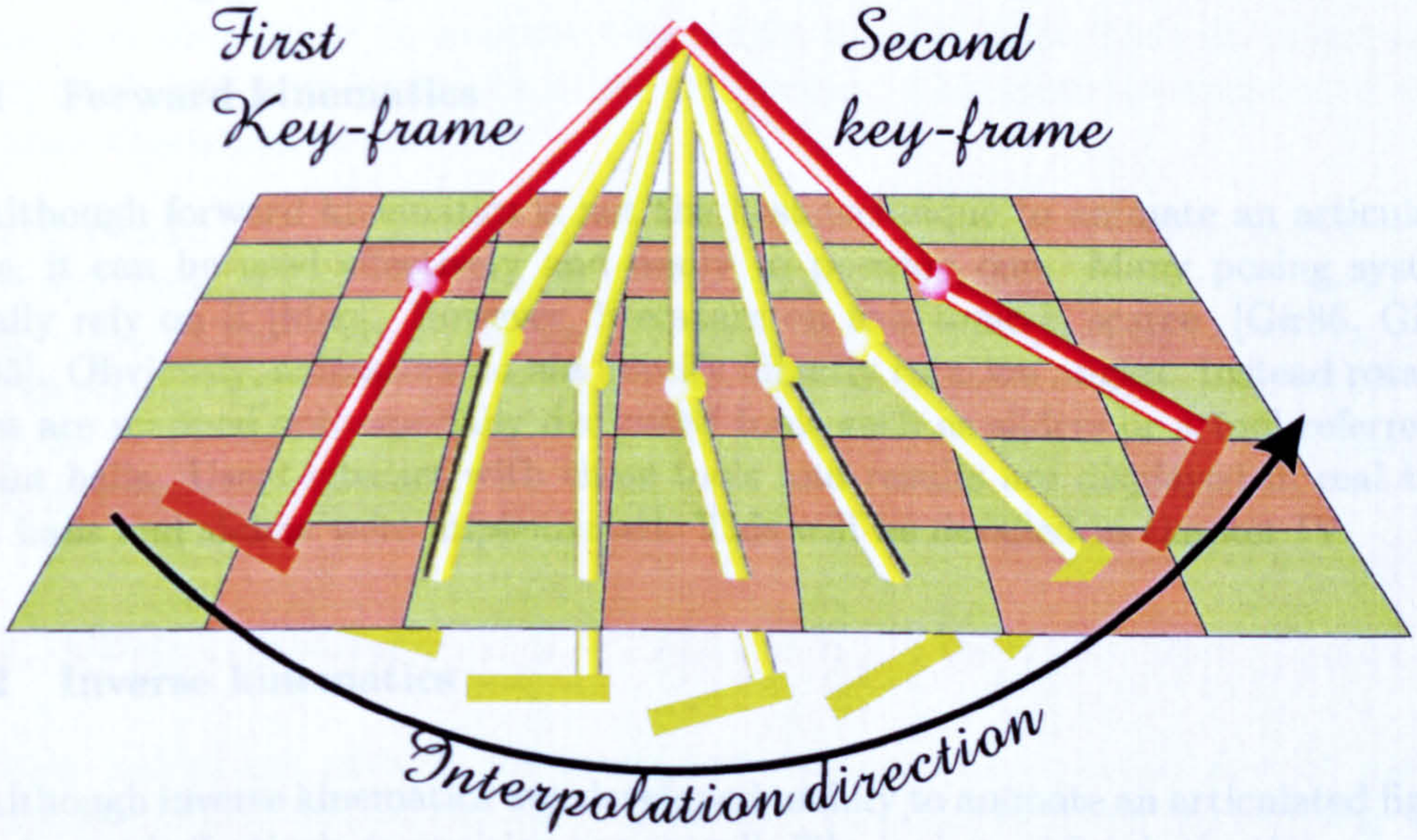


Figure I.4: Handling floor collision

Common interpolation techniques are not intelligent enough to detect and handle collisions. As a result, when a collision happens, the animator usually has to produce one or more key-frames to produce a correct animation and restart the interpolation process.

Animating hands is a complex process [RG91, MTLT88, ST94, LK95]. Key-framing systems are usually not suited for this type of problem. Hands are mainly used to grasp objects. Goal-directed systems such as inverse kinematics are more appropriate.

4 Positioning articulated figures

In modern animation systems, computers are used to aid animators to produce animated sequences of images. In October 1994, I started to work as a research assistant, at the University of Glasgow, on a project called MIME (Make It Move Easily, funded by EPSRC). The goal of the project was to animate articulated figures such as a humanoid by letting the burden of the animation process fall onto the computer. Unlike modern animation systems, the goal was to create a system in which the animator assisted the computer to produce animated sequences.

For this purpose, we used a new concept at the time called *interactive genetic*

algorithms [Daw86, ST90, Sim91, TL91, STH91, Ven95]. The details of this part of the project will be detailed later. However, our first attempts were unsuccessful and it was concluded that the only way to achieve the initial goal was to sub-divide the process of animating an articulated figure in three separate parts. This thesis will focus entirely on the first part which deals with the problem of positioning an articulated figure.

4.1 Existing techniques

4.1.1 Forward kinematics

Although forward kinematics is not the best technique to animate an articulated figure, it can be used effectively and easily to position one. Many posing systems actually rely on it [Mac]. However, literature on this topic is scarce [Gir86, Gir87, GM85]. Obviously, animators do not specify directly rotation angles. Instead rotation angles are mapped onto specially dedicated tools such as sliders or a tool referred to as joint balls. Users interact with these tools and results are displayed in real time. Joint balls and sliders were implemented. This will be detailed in chapter IV.

4.1.2 Inverse kinematics

Although inverse kinematics was developed mainly to animate an articulated figure, it can be used effectively to position one as well. The main problem is the time required to compute rotation angles from one position to another. Although computer speeds have greatly improved since this technique was first used, interactive work might still be out of reach for real-time interaction with a complex articulated figure with modern personal computers. Computing rotation angles is not all. The articulated figure still needs to be rendered and displayed. This is also a time consuming process.

The work presented in this thesis resulted in the construction of a new technique to position articulated figures. Since it needed to be evaluated against conventional posing systems and inverse kinematics in particular, a system capable of doing inverse kinematics was implemented. In 1982, Korein & Badler proposed a faster solution to the problem of inverse kinematics. From this article, a new technique which would tackle the problem of inverse kinematics in a totally different fashion, was devised. This technique could solve the problem of inverse kinematics at a much lower cost than conventional techniques. The implementation of this solution enabled users to interact with a humanoid in real time. Although the rendering system was efficient, most of the computation time was spent in rendering and displaying the robot. This will be detailed in chapter IV.

4.1.3 Inverse dynamics

Inverse dynamics were used by David Forsey and Jane Wilhelms [FW88] to position an articulated figure. This work was done based on the assumption that since dynamics are based on physical laws, positioning an articulated figure using dynamics would be more intuitive and therefore faster than using inverse kinematics.

However inverse dynamics imply the users have to specify weights, friction coefficient, etc. In their research, David Forsey and Jane Wilhelms used the volumes used to represent their robot to evaluate the weight of each limb. Default values which usually work well were set for the other parameters. This gross approximation might invalidate this whole work.

At the time, their technique was not interactive although a fast recursive formulation was used. No evaluation was performed to verify that inverse dynamics were indeed better at positioning articulated figures that inverse kinematics. This still has to be demonstrated. This is a common problem in computer animation where people devise new techniques but carry no effective study to demonstrate the power and weaknesses of their technique.

4.1.4 Other techniques

Other techniques such as rotoscopy and motion capture could also be used to position articulated figures but there is usually no point in doing that. Using these techniques, animating articulated figures is as easy as positioning them.

4.1.5 Proposition

Constructing poses constitutes one of the main tasks of most animation systems. Even techniques which do not usually rely on poses, such as gaits, could be adapted to use poses to their advantage. The main problem with the production of poses is that there is no easy technique to produce them. Common drawbacks to conventional techniques are they are too slow, too cumbersome to use or are not fast enough to interact with an articulated figure in real-time.

For my PhD, I decided to investigate the potential of an entirely new technique to produce poses. Not all tasks that professional animators might want were implemented as these were not felt to be necessary to test the new concept. Thus the possibility of specifying multiple constraints was not implemented.

With conventional techniques, the computer is used to assist animators along the pose production process. In the thesis, I argue that a system in which the user assists the computer in trying to produce poses is more efficient than conventional positioning systems. For this purpose, an interface derived from the field of interactive genetic algorithms was used [Daw86, ST90, Sim91, TL91, STH91, Ven95].

In Chapter II, the origin of the technique is presented. Chapter III will be spent detailing the implementation and the interface of the technique. To be able to evaluate the technique, two conventional techniques were implemented: forward and inverse kinematics. These implementations will be described in chapter IV. Chapter V will focus on the preparation, the results of the evaluation and the evaluation itself. In chapter VI, the thesis is concluded and potential future work is presented.

Chapter II

The articulated figure

1 Introduction

The actor is the key object in an animation system. A badly designed actor will produce bad animations. Rendering is just as important: If it is insufficiently representative, the animator cannot evaluate efficiently the quality of a particular pose or animation. If it is too detailed the computer will spend too much time rendering frames thereby making real-time evaluation impossible. This chapter will present all the main techniques used both in modeling and rendering and present the ones which have been chosen for this work.

2 Actor design

An actor can be represented by any components, simple or complex, rigid or flexible and fixed or movable. A posing system is only used to change the position or shape of the actors in a scene. In this chapter, we are solely interested in the posing of *3D rigid articulated figures.*

A lot of work has already been done on the modeling stage of human and other bodies [Stu84, NTD88, NMT85, Cal88, TP88]. In [Stu84], David Sturman made a comprehensive definition of a rigid articulated figure and also specifies some features that a rigid articulated figure model should own:

The information stored in a model is an important aspect of any animation system. One simple way to define a model is as a set of rigid objects jointed at nodes, organised hierarchically into an articulated body. At each node or joint, a 3D transformation matrix controls the position of the portion of the body below that joint. Transformations matrices are nested in accordance with the body structure. The position of the model at any one instant is determined solely by the transformation matrices. The only

intelligence contained in the model is the topology of the body parts and the degrees of freedom (DOFs) at each joint. Alone the model is a static entity. To make the model move, the animator uses the animation system to control the 3D transformation values at each joint. The "rigid object" stipulation allows scaling of the body parts (using the joint matrices) but not bending or changing their basic geometries.

Starting from this definition, we see that a *3D rigid articulated figure* is an actor made of segments or limbs or body parts all connected together by means of joints. The figure is said to be in *3D* because it is represented in three-dimensional space. The word *rigid* is also used to say that each body part has a fixed length and a fixed shape. This means that objects like deformable balls and cartoon characters cannot be modeled using this representation. Though one of the final goals of this research was to assist cartoon characters animation, special effects like stretching and squashing were not implemented. To evaluate our animation system, a humanoid was implemented. A very simplified skeleton was used as a model. It is made of exactly 19 segments. Such a skeleton is shown in figure II.1.

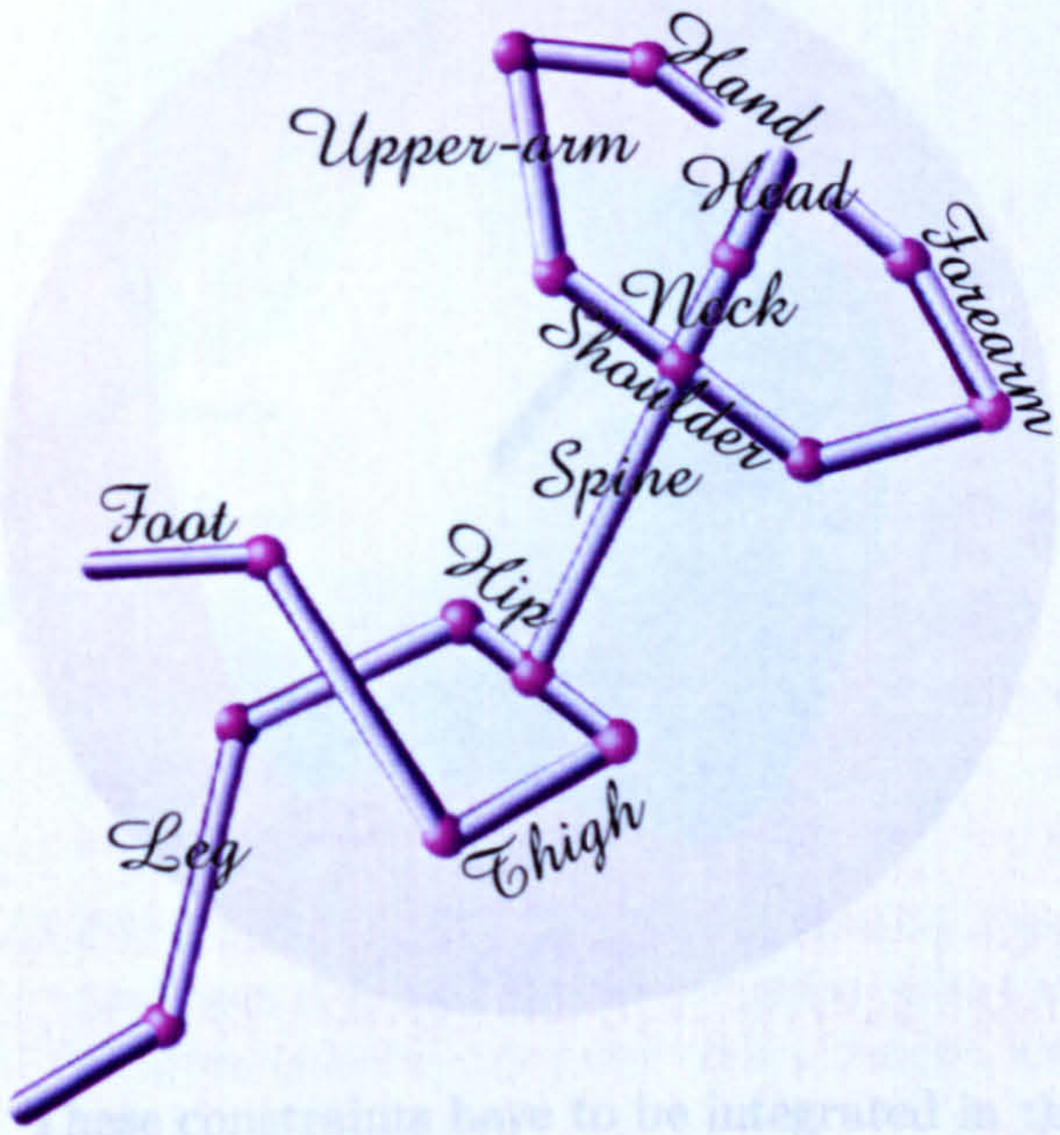


Figure II.1: Example of a simple skeleton with only 30 DOFs

Example of a simple skeleton with only 30 degrees of freedom (DOFs) and 19 limbs. The skeleton is represented in the seated position with the hands at the back of the head.

2.1 The model file

Models describing articulated figures are first written manually into text files. One of these text files is specified when the application is launched.

2.1.1 Tree structure

To represent a given actor, a tree structure made of joints and limbs is specified. Each node specifies a body part usually referenced as limb and each link specifies a

joint. The tree is an n-tree. Each parent body part can have from 0 to n children.

Each limb has a name which is sometimes displayed as a hint to the user. It does not make sense to have the starting coordinates of the limbs outside their parent's limb so starting coordinates of all children have to be on their parent's limb. Most of the time, joints will be located at the beginning or at the ending coordinates of their corresponding limbs. The interpreter of the posing system is not case-sensitive.

2.1.2 Degrees of freedom

If a limb was permitted to move in all directions, the path described by the extremity of the limb would lie on the surface of a sphere in which the centre is the joint position and the radius is the length of the limb. Obviously, no limb of any kind can perform such motions. They are limited in some ways so the path described by the tip of the limb always lies in a limited area on the surface of a sphere II.2.

Figure II.2: Area of permissible motions

If the joint is at the centre of the sphere, and the length of the limb corresponds to the radius of the sphere, then the area in which the tip of the limb can move is described in green

These constraints have to be integrated in the posing system. An accurate specification of the area where each limb is allowed to move would be too cumbersome to specify and not useful for our own purposes. Korein used polygons for which each vertex lay on a unit sphere to approximate the area of allowable motions [KB82, Kor82, JU85]. This was only an approximation since the valid area also depends on a type of movement called twist and on the positions of surrounding limbs. This area also varies amongst individuals. To check if a position was valid, an algorithm was devised to check the position was inside the valid area. Although probably fast, this algorithm could not be as fast as a cruder approximation of this area, an approximation was chosen here. Korein also stated that the implementation of his algorithm was hard. Their work was also meant to be used to simulate human beings for the Jack^{tm} system [Bad86, BPW93], an animation system developed by Badler & al.

We are not interested in simulating an accurate human figure but solely in producing believable postures. Every object in a scene has six DOFs. These count for the translations along the three Euclidean coordinate axes and the rotations around them. For an articulated figure, all limbs except the one at the top of the tree, the root limb, are glued at their joint position. In other words, they are not allowed to move away from their joint positions. Therefore, DOFs dealing with translations can simply be ignored. Only DOFs involving rotations need to be taken into account.

There are three different categories of DOFs [NTD88] which directly relate to three different types of motion. These are called *flexion*, *pivot* and *twist* motions. They are defined as follow:

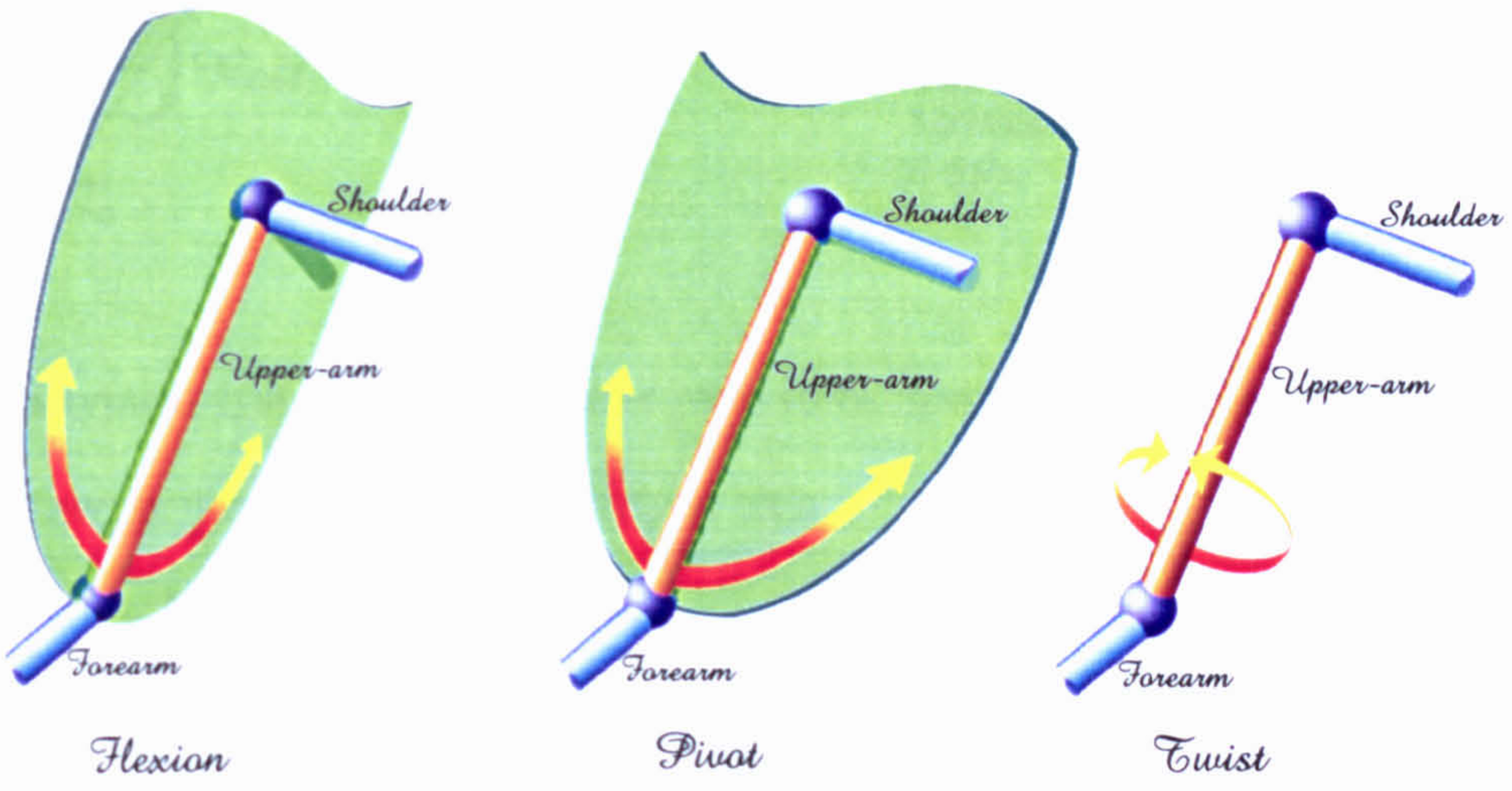


Figure II.3: Degrees of Freedom (DOFs)

Flexion motions are the most common. Some joints can also produce pivot motions and a few others allow twist type motions.

- ❒ **Flexion:** The flexion is a rotation of the limb which is influenced by the joint and causes the motion of all limbs linked to this joint. This flexion is carried out relative to the joint point and a flexion axis which has to be defined.
- ❒ **Pivot:** The pivot makes the bending axis rotate around the limb which is influenced by the joint. The pivot axis is the axis perpendicular to the flexion axis and the axis of the limb.
- ❒ **Twisting:** Twisting causes a torsion of the limb which is influenced by the joint. The direction of the twisting axis is found similarly to the direction of the pivot.

In engineering, another notation called the yaw notation is more often used. It is described in Hoggar's book ([Hog92], page 208). DOFs are not expressed relative to the joint but in scene coordinates. This makes it easier to deal with some problems such as the non-commutativity of the matrix multiplication. The order in which the matrix multiplication is made is of major importance to the result obtained.

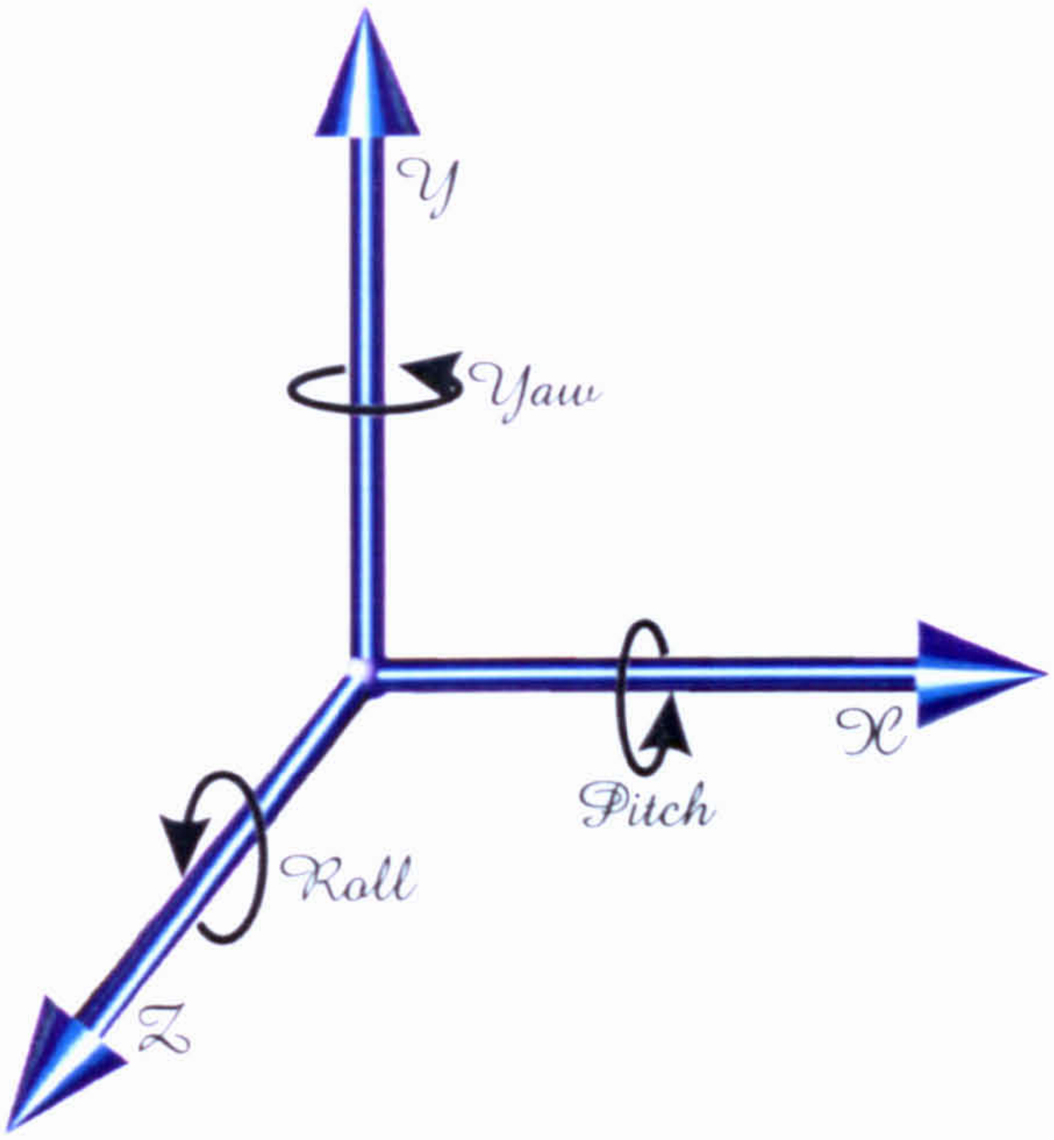


Figure II.4: The yaw notation

The Yaw axis is vertical, the Roll axis is horizontal and the Pitch axis is for the depth

In a typical systems, the rotations have to be specified in the following order:

1. **Yaw**, or heading, around a vertical axis
2. **Pitch**, around a horizontal axis
3. **Roll**, around an axis along the depth of the scene

The first notation was the one used. Like other common animation systems, it was found to be the most appropriate for our purposes. When a limb is created, its main axis, its flexion axis and their directions are specified. They are optional. If they are not specified, the Y axis is assumed for the axis of the limb and the X axis is assumed for the flexion axis. Possible axis are only Euclidean coordinates axis X, Y and Z axes. Arbitrary axes were not worth the difficulty to implement them. The X axis is horizontal and points toward the right, the Y axis is vertical and points upward, the Z axis is horizontal and points inward. The $+$ and $-$ signs are used to specify the direction of the axis, the $-$ meaning that the direction is inverted. The default direction is $+$. Thus, for the joint at the hip for the left leg of a humanoid, the description would look like something like this:

```
Axis -Y
FlexionAxis -X
PivotDof 0 80
FlexionDof -20 160
TwistDof -90 90
```

Once the flexion axis has been specified, the three types of DOFs are specified. There are all optional. If they are not specified, it is assumed that the motions corresponding to a given DOF are not allowed. For each DOF, the minimum and

maximum angles, in degrees, describing the area in which the limb is allowed to move must be specified. Angles are specified in degrees and are specified in the clockwise direction.

Dealing with the DOFs in this way, a valid area would look like something like this:

Figure II.5: Model of the area of permissible motions

The technique used to model degrees of freedom results only in simple areas such as this one to approximate the area of permissible motions.

2.1.3 Representing the model

To render the articulated figure, a few primitive colours are used. These are:

Figure II.6: Colours used

A set of shades for different primary colours are pre-computed and stored in a dedicated colour map to speed up rendering.

The different shades are pre-computed and stored in dedicated colour ramps to allow for fast rendering.

To display the figure on the screen, several techniques may be used most of which are discussed and reviewed in [Stu84, NTD88, NMT85, Cal88, TP88].

- **Wire framing:** The simplest of these techniques is simply to draw a line segment to represent each limb. It is fast but unfortunately, resulting pictures are far from being convincing and crucially lack the capability to represent twists.

- **Volumes:** Volumes may also be used. Pictures produced are a lot more convincing and any types of posture can be efficiently represented. However, compared with the wire-frame technique, rendering is also a lot more time consuming. Fortunately, fast rendering techniques have been built which allow rendering at a reasonable speed and thus allow interactivity.

- **Surfaces:** Surfaces may also be used and it is certainly the best of the available techniques to represent an articulated figure. A surface is made of patches which will match as close as possible the shape of the real figure being model (e.g. a human). The number of polygons necessary to represent an articulated figure is likely to be overwhelming for a good representation. Polygons approximation techniques, which gather polygons in a single one to produce simpler surfaces but close enough to produce pictures of apparently the same quality, do exist [HDD+93, CVM+96, KT96, LKR+96, AS96] but the number of polygons is likely to stay large. Several techniques are also available to render objects made of surfaces. The simplest one is the flat shading technique. It is fast but it does not produce pictures of very good quality unless polygons are very small. Gouraud shading is another technique which can produced images of relatively good quality. Basically, colours are computed at each vertex of the polygon to shade. They are then interpolated to approximate the illumination model. Phong shading is the last of the usable techniques [BW86]. Instead of computing colours at each vertex of the polygon to shade, normal vectors are computed. These are then interpolated and the relative shading intensity is deduced at each pixel. It produces images of even better quality but it is also the slowest of the three. Fast algorithms for converting surfaces to polygons are available [LCWB80, Cla, LR80, Kau87]. Although the best pictures are produced by using surface representations, the rendering process is also much more time consuming than with the two previous techniques. It would be difficult to reach real-time animation without some special hardware and this hardware was not available while this research was taking place. Furthermore, producing the original surfaces and making them move along with their corresponding limb is not an easy task and is a time consuming problem to solve.

It was eventually decided to represent articulated figures by means of volumes. Several volumes may be used to represent an object onto a screen:

- **Spheres:** Spheres can be used to represent a body on a screen. Efficient algorithms to display spheres have already been implemented [Kno81, Pat93]. Knowlton does not exactly render spheres: he displays a pre-computed sphere instead of a real one. For this reason, the algorithm is said to work in a $2^{1/2}$ space only. A wide range of pre-computed spheres are also necessary. Patterson's algorithm [Pat93] is said to suffer from a parabolic approximation of the real sphere equation but this defect is barely noticeable. To represent an ar-

ticulated figure solely by means of spheres would require a large collection of spheres. This counterbalances the efficiency of the algorithms.

- **Cylinders:** Another way to represent a body is to use cylinders. Cylinders are more time consuming to produce than spheres but they can be efficiently used to replace a collection of spheres. James Blinn [Bli89] developed an efficient algorithms to render tubes. The technique is based upon tubes decomposition into polygons at the eye and light silhouettes II.7. If the axis of the cylinder (or tube) is perpendicular to the viewing direction (both sides are hidden), the illusion is perfect. Otherwise the polygons quickly become noticeable (the polygons are clearly visible). As a result, a fast cylinder algorithm was implemented. It is detailed in Appendix II. Basically, this algorithm is derived from Patterson's fast sphere algorithm. The silhouette points on the visible face of the cylinders are first computed. Their colour is calculated at the same time. The main surface is then rendered by sweeping these points along the main axis of the cylinder using forward mapping. It might not be as fast as Blinn's tube algorithm but it does not have any visual defects.

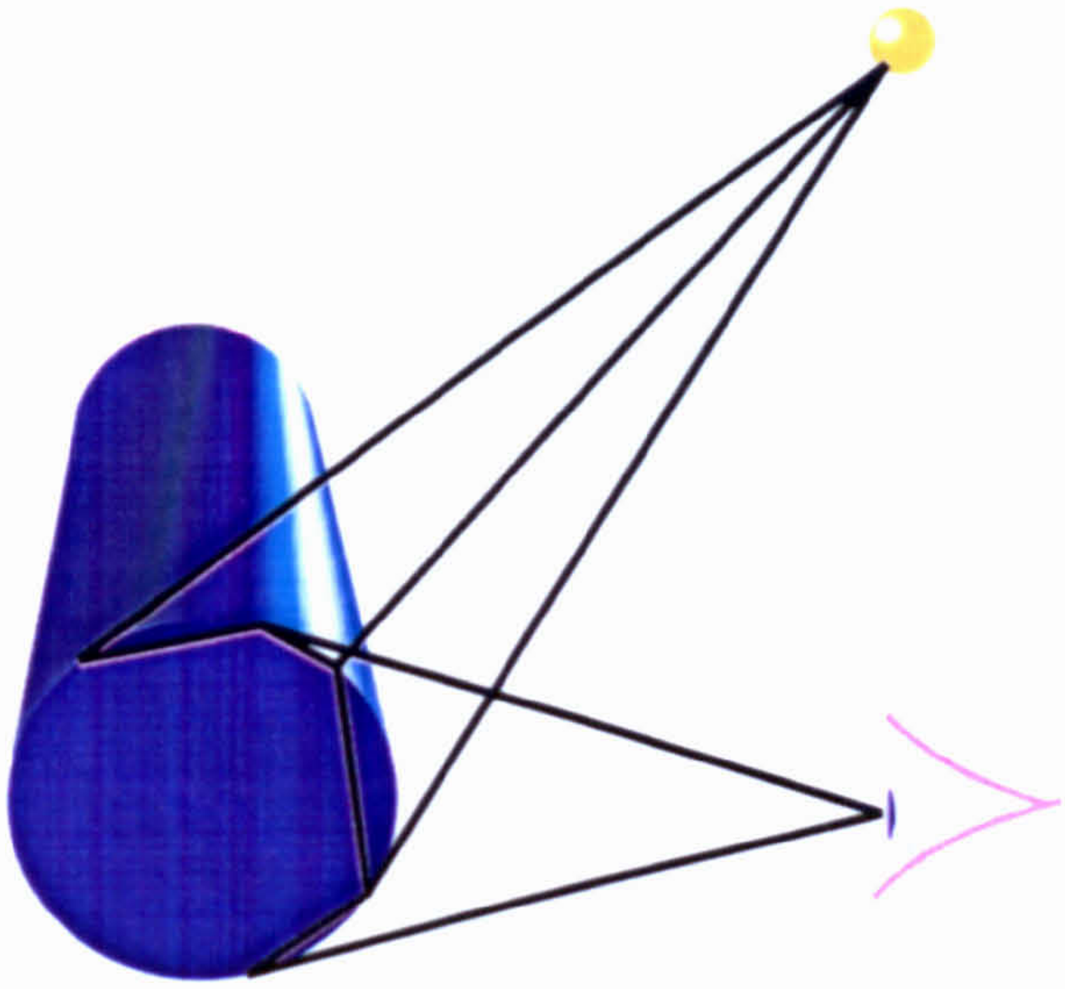


Figure II.7: Blinn's tube technique

Using the silhouette and light vectors, four polygons are generated and rendered using conventional techniques to produce the illusion of looking at a cylinder.

- **Cones:** Cones were also implemented. The algorithm used to render cylinders was adapted for cones. Rendering is fast although the algorithm could not be optimised as much as with cylinders.

- **Cubes:** Most pictures which have been produced in the computer animation literature to represent articulated figures hitherto were mostly made of a collection of cubes. Nice pictures can be produced with little processing power. This solution has become a standard in the area until now. Cubes have also been used in the representation of the humanoid.

- **Ellipsoids:** Ellipsoids are another class of objects which would be nice to use. However, to my knowledge, no efficient algorithm is available for the time being to render such $3D$ objects. Herbisons-Evans details an algorithm used to

render ellipsoids [HE80, HE82] for their animation system. However, the representation for such objects is only in $2D$ and so can only be extended to a $2^{1/2}$ representation. This was not found to be satisfying for this animation system.

To render articulated figures, the following primitives have been implemented:

- **sphere**: The position, radius and colour need to be specified.
- **cylinder**: The positions of the two extremity points, the radius and colour need to be specified
- **cone**: The position of the origin and the end of the cone, the radius and colour have to be specified
- **cube**: Four points forming three orthogonal vectors and the colour have to be specified. Only four points are necessary to make a cube. Specifying these points is not easy and requires a bit of practice.

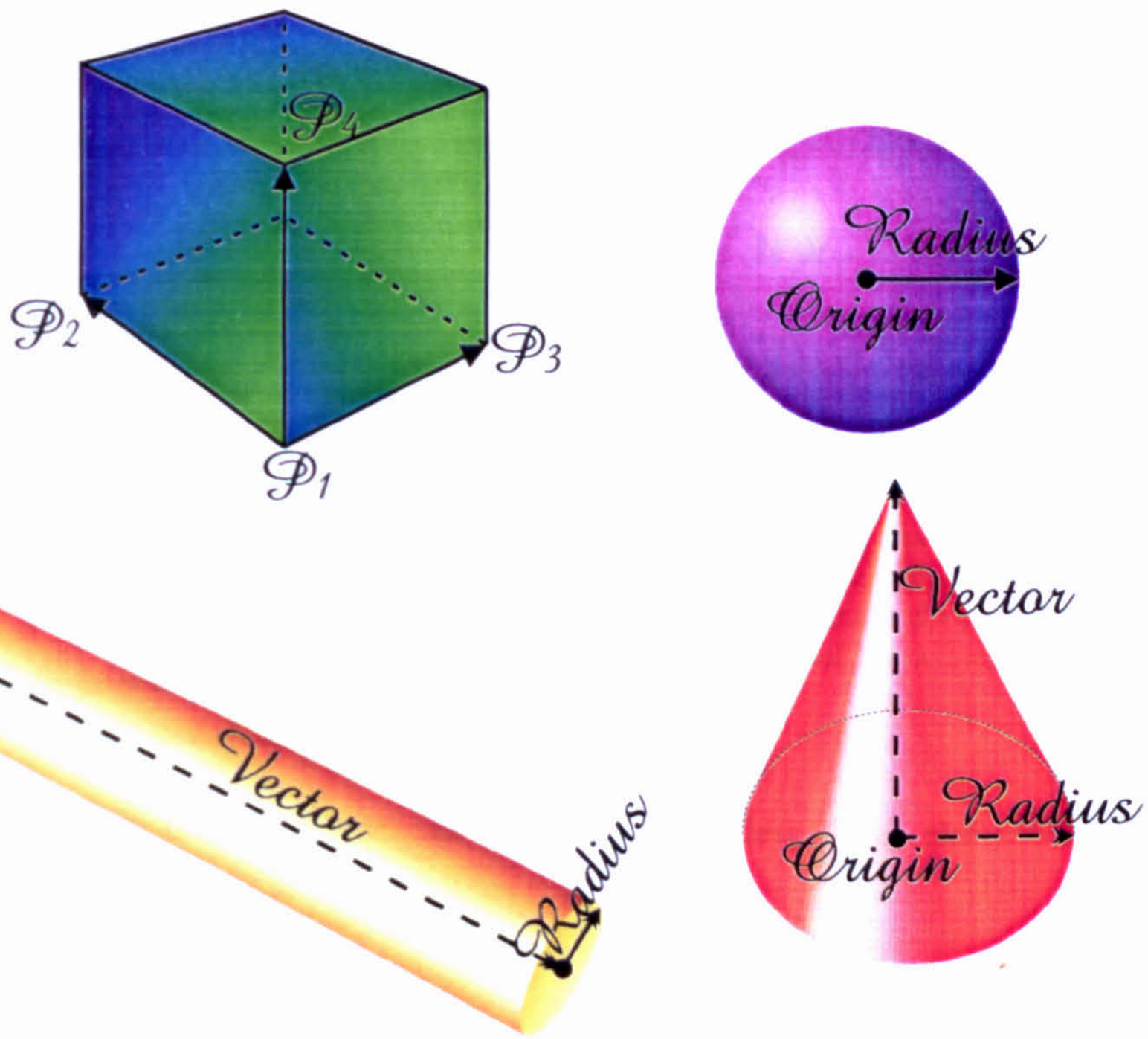


Figure II.8: Volumes used

Four different volumes are being used. Fast renderers have been implemented for each of them to allow for real time animation production.

2.1.4 Exploiting similarities

Articulated figures usually have many similarities. For instance, the right and left sides are identical apart from the fact that coordinates on one axis (usually the X

axis) are reversed. These similarities can be exploited by positioning and animation systems. For a positioning system, model files specifying articulated figures can be smaller and some mirroring functions can be implemented.

For this purpose, a special directive has been implemented (Fig. II.9). It is called *Mirror*. This directive is specified just before the joint directive. All the children limbs of the limb being mirrored are also mirrored. Thus specifying that the *thigh* is mirrored will also mirror the left leg and the foot. This directive is followed by two words, which are usually *left* and *right*. They are used as a prefix for each name of the limbs being mirrored. Thus when the articulated figure will be mirrored, the *LeftThigh*, the *RightThigh*, etc will be created. The axis of the limb is used to direct the mirroring operation. For example, if the axis of the limb is the X axis, the X coordinates of the top limb being mirrored are inversed.

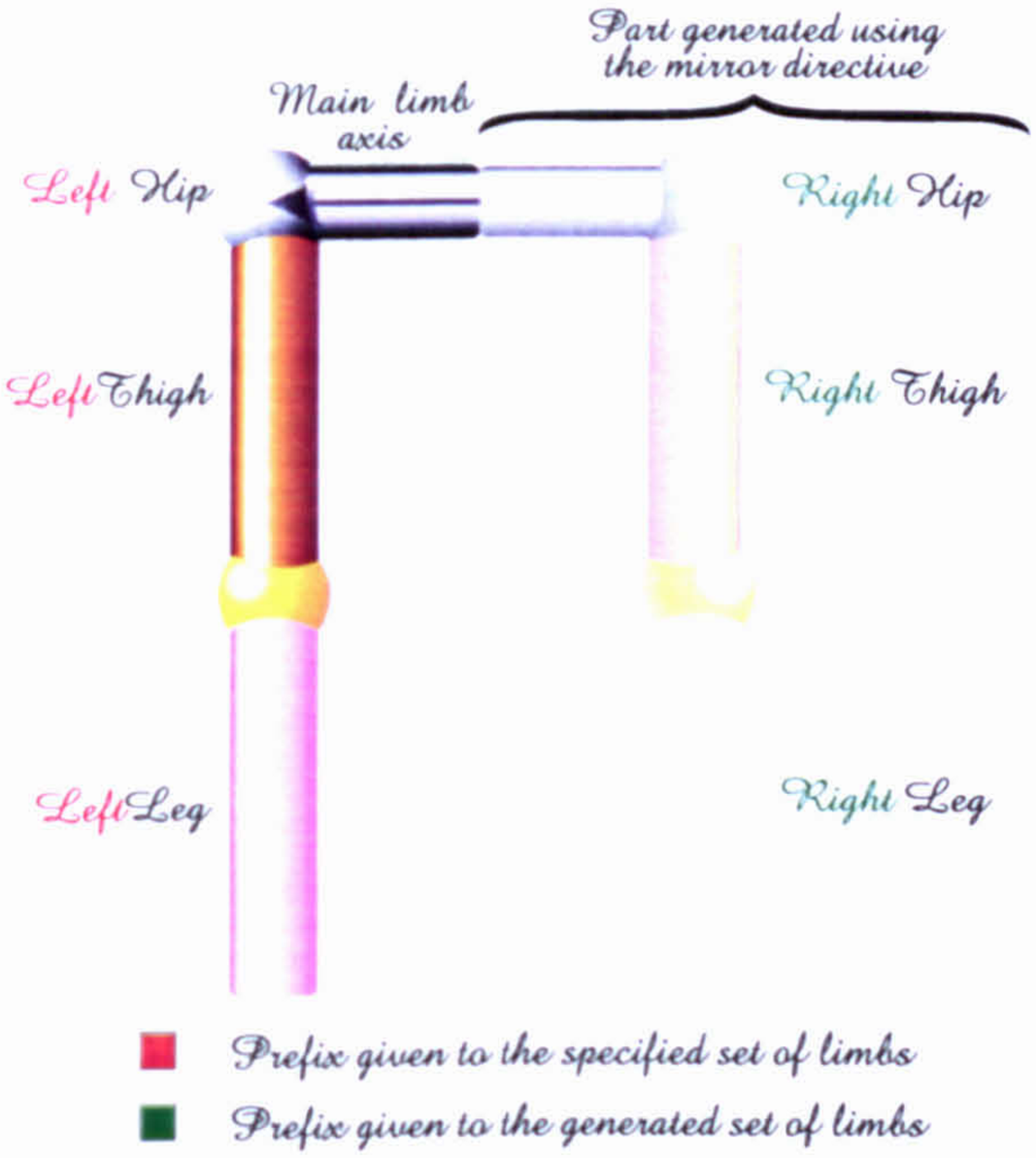


Figure II.9: Mirroring a set of limbs

When a situation like the one shown on the left occurs, the *mirror* directive can be used. Only one side has to be described, the other one being automatically deduced by the computer. Furthermore, it will allow the animation to take advantage of this information to ease further the job of the artist.

To conclude, Figure. II.10 shows the piece of code describing the model used in this thesis.

2.1.5 The problem of the spine

The spine of the human body is made of 33 vertebrae. To produce the best animations possible, Monheit and Badler [MB91] argue that an accurate modeling of the human spine and torso becomes necessary. In their work, they implemented a kinematic model of the human spine and torso. A careful study of how the spine is also allowed to move was also performed. However, the goal of this work was more to simulate the human spine and torso rather than animate it. Our purpose is not simulation of the human body so we do not need to use that many vertebrae. In fact, to model well enough the human spine, only three limbs would be required. The

```
top {
 axis x
 0 0 0 0 0 0
 mirror right left joint 0 0 0 hip {
     axis x
     0 0 0      −16 0 0
     sphere −12 0 0 12 grey
     joint −16 0 0 thigh {
      axis −y
      flexionaxis −x
      0 0 0 0 −45 0
      pivotdof 0 80
      flexiondof −20 160
      twistdof −90 90
      cylinder 0 0 0 0 −45 0 8 orange
      sphere 0 −45 0 10 orange
      joint 0 −45 0 leg {
          axis −y
          flexionaxis −x
          0 0 0 0 −55 0
          flexiondof −160 0
          cylinder 0 0 0 0 −25 0 6 pink
          cylinder 0 −20 0 0 −45 0 5 pink
          joint 0 −55 0 foot {
           axis −z
           flexionaxis −x
           0 0 0 0 0 −30
           flexiondof −20 20
           pivotdof −30 30
           cube −6 0 −30 −6 9 −30
           6 0 −30 −6 0 8 yellow
          }
      }
     }
}
joint 0 0 0 spine {
     axis y
     flexionaxis x
     0 0 0 0 50 0
     pivotdof −50 50
     flexiondof −70 45
     twistdof −45 45
     sphere 0 0 0 8 orange
     cylinder 0 0 0 0 12 0 11 orange
     sphere 0 15 0 10 orange
     cylinder 0 20 0 0 32 0 15 orange
     sphere 0 35 0 10 orange
     cylinder 0 40 0 0 52 0 19 orange
     mirror right left joint 0 50 0 shoulders {
      axis x
      0 0 0 −25 0 0
      cylinder −25 0 0 0 0 0 7 grey
                                          joint −25 0 0 upperarm {
                                              axis −y
                                          flexionaxis −x
                                              0 0 0 0 −33 0
                                              pivotdof 0 180
                                              flexiondof −60 80
                                              twistdof −90 90
                                              sphere 0 0 0 8 orange
                                              cylinder 0 0 0 0 −33 0 5 orange
                                              sphere 0 −33 0 7 orange
                                              joint 0 −33 0 forearm {
                                               axis −y
                                          flexionaxis −x
                                               0 0 0 0 −25 0
                                               flexiondof 0 160
                                               cylinder 0 0 0 0 −25 0 4 pink
                                               sphere 0 −25 0 6 pink
                                               joint 0 −25 0 hand {
                                                   axis −y
                                               flexionaxis −x
                                                   0 0 0 0 −12 0
                                                   flexiondof −45 45
                                                   pivotdof −70 70
                                                   cube −2 −12 −4 −2 0 −4
                                                   2 −12 −4 −2 −12 4 yellow
                                               }
                                              }
                                          }
                                         }
                                         joint 0 50 0 neck {
                                          axis y
                                          flexionaxis x
                                          0 0 0 0 17 0
                                          flexiondof −45 20
                                          pivotdof −45 45
                                          twistdof −90 90
                                          cylinder 0 17 0 0 0 0 8 orange
                                          joint 0 17 0 head {
                                              axis y
                                              0 0 0 0 12 0
                                              sphere 0 12 −5 16 pink
                                              sphere 0 16 −3 16 yellow
                                              cube −3 30 −18 3 30 −16
                                                   −3 33 −18 −3 16 −12 yellow
                                              sphere −6 16 −17 4 blue
                                              sphere 6 16 −17 4 blue
                                              cube −2 11 −24 −2 19 −17
                                                   2 11 −24 −2 10 −21 red
                                              cube −5 6 −20 −5 7 −20
                                                   5 6 −20 −5 6 −16 red
                                          }
                                         }
                                     }
                                    }
                                   }
                                  }
                                 }
                                }
```

Figure II.10: Description of a humanoid

This humanoid was the articulated figure used during this research. Other creatures could be easily implemented as well.

position of the the different objects along the spine would be interpolated to simulate many more limbs.

In the model presented here, the spine is made of only one limb. Positions achieved with the system were judged to be satisfying enough to demonstrate the hypothesis. To improve the realism, a curve could have been used to pose the different objects used to represent the torso (spheres and cylinders). This has not been performed but is considered for future work.

Figure II.11: Juxo, the lamp

A lamp was also implemented to show that this work did not focus solely onto humanoids. This lamp was called Juxo since the name Luxo is trademarked.

2.1.6 The hands and the feet

Animating hands is a complex process [RG91, MTLT88, ST94, LK95]. We decided that a positioning systems was not appropriate. As a result, the hand is modeled by a single limb segment and is represented by a cube.

Some applications may require an accurate model for the feet. This was not the case here. Consequently, a single cube was used to render each foot.

Chapter III

Origin of the technique

1 Richard Dawkins's Biomorph

One of the most famous arguments of the creationist theory of the universe is the eighteenth-century theologian William Paley's saying:

> Just as a watch is too complicated and too functional to have sprung into existence by accident, so too must all living things, with their far greater complexity, be purposefully designed.

Like many people, Richard Dawkins, a professor of zoology at Oxford University was not impressed by William Paley's argument. In 1986, he wrote *The Blind Watchmaker* [Daw86]. He argued that there is not need for an intelligent upper being and that the watchmaker is simply nature and its tool is called evolution.

To have a watch produced by sheer luck is virtually impossible. However nature and the principle of evolution which has been revealed for the first time by Darwin in 1859 is far more than just sheer luck [Dar59]. In simple terms, Darwinian evolution involves three major concept: *Reproduction*, *mutation* and *survival of the fittest*. The combination of the three can perform amazing feats, such as producing human beings who will invent watches after billions of years of evolution.

To show how powerful evolution is, Dawkins implemented Biomorph, a program which produces forms made of small line segments. The number of line segments, their length and direction was defined by a structure which we can call a chromosome. To make the program simpler, reproduction involved only one parent. Thus, the advantage of sexual reproduction was simply ignored and only mutation was used to evolve these forms. Although the program did not use all the ingredients of evolution, it amazingly fulfilled the expectations of his creator. Many forms which resembled shapes of different animals were produced. Dawkins also produced all the letters of the alphabet.

2 Mutator model

In 1990 and 1991, Steven Todd and William Latham worked on a program which they called mutator. Steven Todd is a computer scientist working at IBM whereas William Latham is an artist. Together, they tried to bring the best of both worlds in a single project. In [ST90], they provided a definition of Mutator:

> Mutator assists an artist to create computer sculptures. The artist makes a series of judgments of examples presented by the computer. For a major part of the create process the artists focuses on aesthetic considerations, freed from the mechanics of computer interaction and form realization.

As mentioned in [ST90], Mutator was first used to produce sculptures but in later versions [TL91], it was also used to produce animations. These animations were simple as creatures could only grow in size. Creatures had a life cycle in which they were first born, grew and died. They also moved along a simple path in space.

At the initial stage, the user needs to create a structure defined as a vector which gathers all the parameters necessary to create a sculpture. Typical parameters are:

- **sphere:** the number of spheres in the sculpture
- **ribs:** the number of items
- **grow:** shrinkage and expansion of the elements
- **bend:** bend stack
- etc

Parameters values are ranged to avoid the production of useless sculptures. In effect, Mutator is used to browse an n-vector space.

The simplest form of mutator presents the user with nine forms: an original sculpture and eight mutations. The user simply selects the favorite sculpture and a new set of eight sculptures is produced. At the first generation, the original sculpture is set to an initial parameter vector. The user is allowed to set the intensity of the mutations to permit fast initial exploration of a wide space, followed by fine tuning.

In a more complex form, Mutator allows the user to make *marriages*. Users select two parents which are then used to create the children by mixing the parameters of the two parents.

3 Interactive Genetic Algorithms

In 1975, John H. Holland published an article called *Adaptation in natural and artificial systems* [Hol75]. In this book, he described how Darwinian evolution could

be used to solve searching problems. These type of algorithms were called *Genetic Algorithms (GAs).* Genetic algorithms use three main concepts: *selection, reproduction* and *mutation*, the selection being the mechanism of the survival of the fittest. To perform the selection, the computer needs to evaluate how good or how fit a solution (an individual) is compared to other solutions (individuals). For this purpose, an objective or fitness function is provided. For each individual, this function will provide a value (the objective value). The goal of the algorithm is then to minimize or to maximize the objective value depending upon what has to be achieved. Appendix A explains in more detail how genetic algorithms work.

Interactive Genetic Algorithms (IGAs) refer to the type of algorithms where the objective function is the user. With this type of algorithm, it is the user who has to provide the objective value.

The mutator model and IGAs are quite similar. They were more or less developed at the same time. With the mutator, not so much emphasis is put on the biological aspect of evolution. IGAs were completely derived from John H. Holland's GAs and henceforth have a more theoretical background.

3.1 Fields investigated

Genetic Algorithms have been used in many areas such as robotics, plane modeling, etc [Dav91b]. In comparison, IGAs have been applied to just a few fields.

3.1.1 Pictures and objects

Mutator was used to produce abstract sculptures [ST90, TL91, STH91].The artist used aesthetic considerations to select one or two sculptures which are then used as seeds for the next generation. Although generated sculptures do not recall anything known, they are amazing and resulting pictures are extremely appealing.

In 1991, Karl Sims implemented the first true IGA for which, to demonstrate the capabilities, he applied to produce 3D plant structures, 2D abstract images, solid textures and abstract animations [Sim91]. To be more accurate, a genetic algorithm was not used but instead a concept referred as genetic programming. Instead of producing individuals (images, 3D plants, etc), genetic programming [Koz92, Koz94] is used to generates programs. These programs were written in Lisp. It is particularly difficult to write a non-working program in Lisp therefore it is particularly well suited to this type of problem. Once generated, the programs are executed to produce the individuals, these being images, solid textures or animations. The instructions provided were also purposely limited so that, for example, the IGA used to produce images could only produce useful images. Set of functions are also provided so that visually interesting solutions could be produced. This is particularly true for the production of images. For the production of plants, like conventional GAs, a parameter set called a chromosome was used. It contains 21 elements or genes. It tells information such as

how fast a segment grows, when it should generate buds, in which direction, etc. A program is then executed which interprets the parameter set and generates the corresponding plant. To produce images, solid textures and animations, Lisp programs are first generated, then executed. To produce something, programs usually need inputs. For images, the input is the position of the pixel to colour. For solid texture, the input is the position of the voxel to colour. For animation, the information is the position of the pixel plus the time value. Implementation of such programs is not difficult. The choice of the set of functions is very important though. [1]

More recently, evolutionary techniques were successfully applied to generate 2D and 3D textures [TH95, Pet97].

3.1.2 Virtual creatures

A few years later, Karl Sims published two papers, one in Siggraph '94 [Sim94b] and one in Artificial Life '94 [Sim94a]. These papers described how artificial creatures with a brain and a body were evolved. Brains were made of neurons and bodies of rectangular cubes of different size. The initial population of creatures was made of randomly generated creatures of which more than half simply did nothing. Every creature was rated on their capability to achieve a given task (e.g. move forward, follow a point, etc). A genetic algorithm was used to evolve these creatures onto a connection machine with hundreds of processors. Physical laws were used to simulate the real world in which creatures were put in. This was the most time consuming process. The system was not interactive and could run for days.

One year later, Ventrella published an article [Ven95] describing how the motion produced by artificial creatures could be guided interactively by a user. His creatures were made of rigid segment linked by joints. Only the joints were allowed to move and no constraints (Degrees of freedom) were enforced. Creatures were also allowed to change their shape. Apparently, it did not produce interesting results apart for a scheme where the body of the creatures had to be symmetric. Motion was specified by means of sine wave functions whose amplitudes, frequencies and phases could vary. Animation and rendering was fast enough to allow for interactive work to proceed. Like conventional IGAs, users selected pleasing results which were used to produce the next generation. To get useful results, a few constraints such as maintaining the head at a given height and insisting for locomotion had to be enforced. In the background, the genetic algorithm was generating many simulations and only the ones which fulfilled the constraints were displayed.

[1] I personally implemented an IGA which also used genetic programming but I did not have access to all the functions that Karl Sims used (noise functions particularly). As a result, generated pictures were far less varied and interesting. I also could not afford the computing power he was using on so programs tended to be much shorter and simpler.

3.2 Limits

Originally, the plan was to animate an articulated figure using an IGA. We were quite optimistic in being able to achieve this goal since the examples detailed in the computing science literature looked impressive.

At first, limbs were allowed to move freely in the areas defined by the degrees of freedoms (DOFs) over a short time span. A set of short animated sequences were produced which could be played by the user. The user selected the preferred ones and these were used to produce a new set of animations. The system quickly produced interesting animations but the user had virtually no control over what was going on. We realised that the size of the space was far too big (about 10^{640}) for an interactive system to be able to search through it efficiently, no matter how powerful the search algorithm is. During this period, an automatic user was also implemented to try to work out what would be the ideal size of the population, the number of generations, the most useful operators and parameters for the search algorithm. The computer produced a randomly generated target animation. It then compared each animation and selected the closest ones. A simple least square technique was used to evaluate how close an animation was to another one. Trials were made with a populations of nine and twelve individuals. Higher than that, the screen becomes too small to be able to display everything properly. I realised that the process never reaches the target animation. On average, when the search process stopped, it was still half way to the target. Usually, after forty generations, the convergence speed was too slow. So at the same time, the search process was converging too fast because all the genetic material needed to reach the target disappeared too soon and it was also converging too slowly because forty generations are far too many generations for an interactive system. To animate his artificial creatures, Ventrella [Ven95] experienced the same type of problems: many generations were necessary and the user had virtually no control over how a particular creature was achieving its task. This made the system quite interesting from an artificial life point of view but not so useful from an animator point of view. The same conclusion could also be drawn for Karl Sims's evolving creatures [Sim91, Sim94b]. In his system, users were simply not there. Apart from ensuring that some constraints were fulfilled, there was no means by which a user could interact with the system and guide the search.

It was concluded that an IGA is not appropriate to animate an articulated figure. Since the main problem was related to the size of the search space, the first thing to do was to make it smaller. So instead of generating animations from scratch, the system was divided in three major parts, the first one to produce poses, the second one to produce short motions and the last one to produce fully complex animations (Fig. III.1).

An IGA was implemented to produce poses. For the first generation, random poses were computed and displayed onto the screen. The user was able to grade each pose according to how close they were to a target pose. The higher the grade was the higher the likeliness of the corresponding pose was to be chosen to produce the next generation. This was just a more powerful selection mechanism.

Figure III.1: An animation system in three parts

To make the animation system as viable as possible, it was divided into three separate parts: the production of poses, the production of simple motions such as walks, runs, etc and the production of fully complex animations. Each part would use what has been produced by the previous part of the animation system.

Trying to make the best use of the system, this was found to be far from satisfying. First, providing an objective value was like marking an exercise, that is an intellectually demanding task. Second, control over the evolution was poor. It was difficult to say the least to reach a target pose. Third, there was no way by which users could directly tell the computer what had to be used. It would have been much faster if the user was allowed to select the parts of interest.

As a result, it was concluded that for an IGA to be efficient, four rules have to be fulfilled:

- ❒ The size of the space to search should not be too big
- ❒ The user should be good at grading one particular solution
- ❒ The user should not know what makes a good solution
- ❒ The user should not look for an accurate target solution

If these four rules can be fulfilled, an IGA will be a really powerful and enjoyable tool to use. Otherwise, the matter can be quite different.

In the cases where an IGA has been used successfully, most of these rules were fulfilled. If the size of the search space is really big, the user can definitely not afford to look for a particular solution. If, on the contrary the size of the space is small, then target solutions can be reached.

In successful examples previously described, what was produced was usually so abstract that the user did not know what made a solution good. Grading or selecting pictures was an easy task for users. Users were not encouraged to look for a target solution, but instead to guide the evolution process in pleasing directions. The system developed by Ventrella [Ven95] was different, but he had to resort to a great deal of constraints such as maintaining the head at a given height. He also had to use simple creatures to keep the search space small, and even though resulting animations were rather abstract and control over them was poor.

For the purpose of positioning an articulated figure, a simple IGA just cannot succeed:

- The size of the search space is usually too large (unless the articulated figure is very simple).
- To provide an objective value is difficult, users know what is good and what is not (e.g. part of the arm but not the legs).
- When animators want to pose an articulated figure, they do not want to be pulled by the evolution process in directions which do not interest them although poses produced that way may look quite interesting.

Chapter IV

Generator

1 Principle

Having gained experience from previous unsuccessful trials, an innovative technique to position articulated figures, which could prove to be more powerful than existing techniques, was designed. Although it was proven that a true Interactive Genetic Algorithm (IGA) could not be used to position an articulated figure, letting the user directly select good limb positions is powerful concept. The newly produced pose can then be mutated to produce another population of poses and thus converge towards a target pose.

A definition of this new technique might be:

> The generator is an evolutionary technique for which genes are clearly identifiable by the user and the cross-over process (i.e. the reproduction process) is explicitly performed by the user. Mutation is then applied to produce a new population of individuals.

The previous chapter concluded by stating four rules which have to be fullfilled for an IGA to be useful at a problem at hand. Three similar rules can be stated for the generator:

- Genes can be made clearly identifiable to the user
- Particular values for theses gene can be made easily selectable by the user
- There should not be too many such genes

If these three rules can be fullfilled, then the generator can be used to solve the problem at hand. If on the contrary one of these rules cannot be fullfilled, the generator should not be used. A mutator or IGA type interface will perform better if the four rules for this type of interface can be fullfilled.

2 Producing a pose

2.1 Definition

A pose is the term used to describe an articulated figure in a particular position. Internally, that pose is defined by a set of limb configurations which are held in a dedicated structure.

An articulated figure is represented by a tree for which each node corresponds to a joint. Each node has a corresponding matrix which is used to specify the current position of the limb relative to its parent. To find out the position of a given limb in the virtual world, all matrices from the root node to the corresponding limb have to be multiplied. Multiplication order is important. The resulting matrix is obtained by multiplying joint matrices together starting from the most distal limb back to the root of the tree:

$$\begin{cases} M_i = j_i \times M_{i-1}, & \text{if } i > 0 \\ M_0 = j_0, & \text{otherwise} \end{cases}$$

where M_i is the resulting matrix at joint i
j_i is the local transformation matrix at joint i

To render the current limb, the resulting matrix has to be applied onto the objects representing that limb. New coordinates are derived and the rendering process can take place. This design allows to change easily the configuration at one joint without having to recompute anything else in the tree describing the articulated figure.

2.2 Choice of a positioning scheme

Given an articulated figure, there are an infinite number of possible poses. The size of a space always affects the powerfulness of a search algorithm; the smaller a space is, the faster it can be searched through. Consequently, the space of poses has to be kept as small as possible.

There are many ways to browse a space and it depends a lot on its encoding. For example, the space of visible colours can be represented using the RGB encoding. It could also be represented using the HSV encoding. Although there is no noticeable difference to the human eye, the space using the HSV encoding is somewhat smaller than the one using the RGB encoding. Consequently, it should also be much faster to search through [1]. Similarly, the set of poses might be represented in several different ways.

In the tree describing the articulated figure, at each node (joint), there is a matrix which defines the configuration of the joint and hence of the different limbs down

[1] It shoud be noted that shorter spaces are not always easier to browser as the new representation may make the browsing harder

that part of the tree. Thus, the search space could simply be arrays of matrices, one matrix for each joint. However, this would not be efficient as a matrix contains a lot of information such as scaling and translations which are not used. It would also be difficult to check the matrices do not break the constraints imposed by the degrees of freedoms (DOFs). Alternatively, an array of cells specifying the direction of the limb plus another angle for the twist could be used. Even better, three angles are sufficient to encode for any position for a single limb. One angle would be used for flexion, one for pivot and one for twist. Thus flexion & pivot would be polar coordinates. It is possible to reduce the parameter set even further. In some cases two different angles for the flexion and the pivot will result in the same position. This is redundant. Furthermore, solutions are not evenly distributed, that is solutions close to each other in the parameter's space might produce poses which look much more different; this is said to increase the chaotic nature of the search space. In practice, some positions would be a lot more difficult to obtain than others.

2.2.1 Use of a hyper tessellated sphere

To remedy to these problems, a tessellated hyper sphere was used. A tessellated sphere is made of a set of points that are evenly spaced from their neighbours [2]. Hence there is no redundancy of information and no position is more difficult to achieve than others. The sphere has a radius of one and every points lie on its surface. A point is a 3D vector and specifies only one direction. Points are numbered so a joint configuration is made of a number and of a twist angle. Henceforth, only two numbers are necessary. A vector of these joint configurations specifies one pose. Knowing the direction of a point and a twist angle, the corresponding matrix is easily obtained. This is explained later on.

To generate a pose, an innovative search technique, which intensively involves the user, is used. At the beginning, of the search, the computer does not know what has to be generated therefore the search is being performed on the entire part of the space defined by the DOFs. As the search progresses, it is narrowed to reach convergence.

To ensure the search will be optimal, several tessellated spheres are being used. Some have very few points, whilst some have many. An ordered chain of these spheres is used to make an object, the hyper tessellated sphere. The first sphere of the chain is the one with the fewest points and the last is the one with the most points. At the beginning of the search, since it has to operate on a big part of the search space, one of the crudest decomposition levels is used. As the search progresses, it also gets more and more focused. Finer decomposition levels are used accordingly. The number of levels of decompositions were carefully taken to ensure that all possible positions were representable. Seven levels of decomposition appeared to be sufficient for our purposes.

[2] This is actually not quite right. Above a given number of points, algorithms able to produce evenly spaced points are not known. However, the tessellated sphere used was found to be good enough

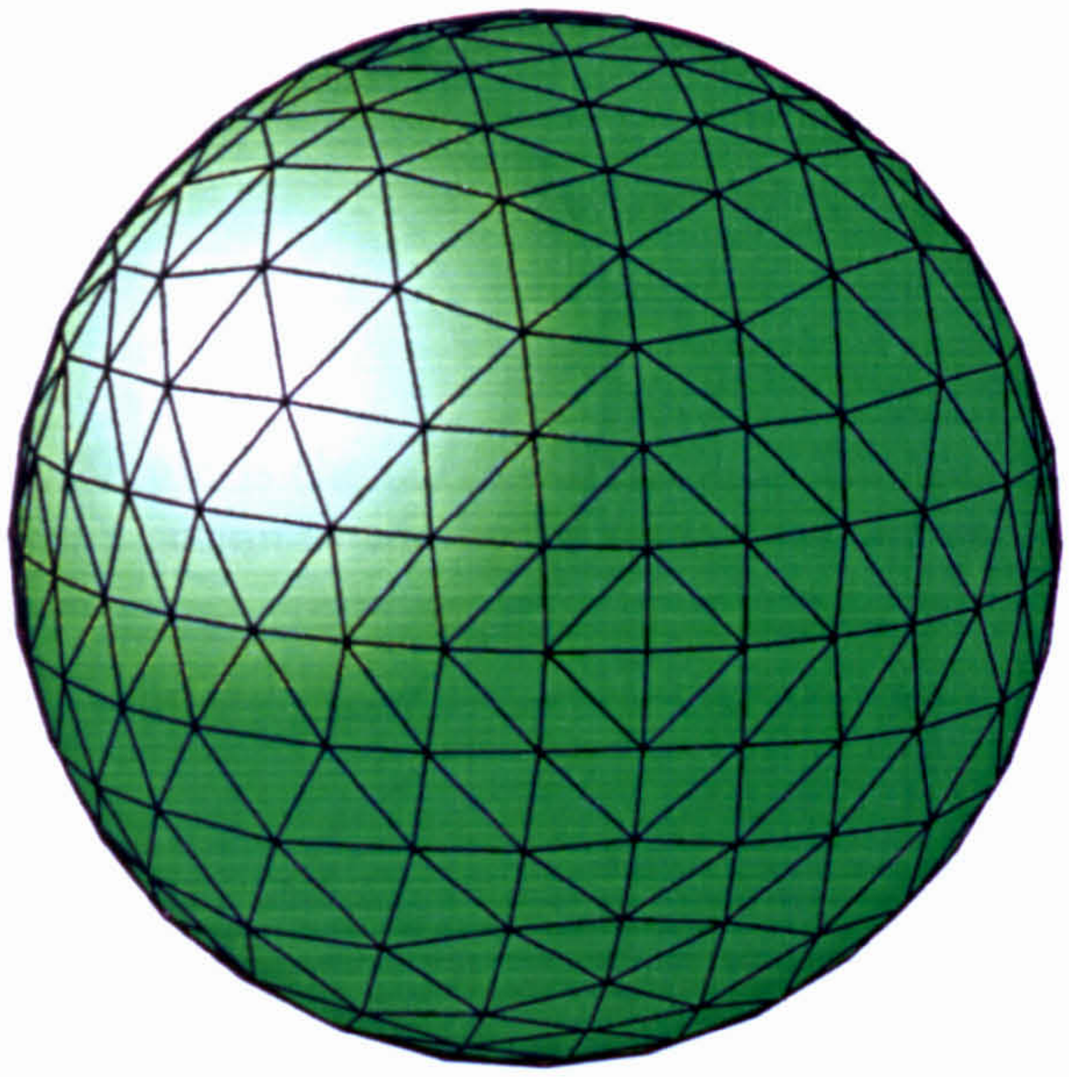


Figure IV.1: Tessellated sphere

A tessellated sphere produces a set of points which are more or less equally spaced, thus avoiding redundancy.

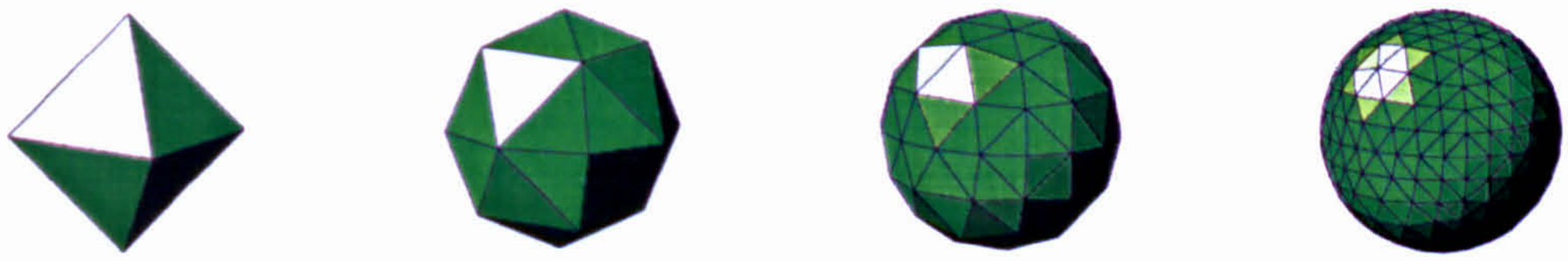


Figure IV.2: Hyper tessellated sphere

A hyper tessellated sphere is made of a set of tessellated spheres. Each sphere corresponds to a decomposition. Here are shown the first 4 decomposition levels. Seven levels are being used by the application.

For twist, although the search space is much smaller, a hyper tessellated circle has been used. The generation of the tessellated hyper circle and its use are similar to the generation and the use of the tessellated hyper sphere. At joints where only one degree of freedom is used for Flexion & Pivot, a hyper tessellated circle is used as well.

2.2.2 How points are computed,saved

To produce the hyper tessellated sphere, an algorithm which was provided by a net user, Mike Castle [Cas94], was used. This algorithm was used to produce a tessellated sphere of any required detail. The algorithm starts by using a set of eight unit triangles (Initially, there are 8 triangles and 6 points). They describe two pyramids whose vertices lie on the surface of the unit sphere. To tessellate it, the algorithm takes in turn every triangle, and for each triangle computes the midpoint of each edge. These points are then projected back onto the unit sphere. So for each triangle, four new triangles are created at each iteration. The number of triangles is multiplied by four at each iteration and the number of new points is half the number of triangles. Thus

the number of points is

$$\begin{aligned} Number\ of\ points &= 6 + \frac{8 \times 4^n}{2} \\ &= 6 + 4^{n+1} \\ &= 6 + 2^{2 \times (n+1)} \end{aligned}$$

where n is the current iteration. The iteration number corresponds to the level of decomposition of the hyper tessellated sphere.

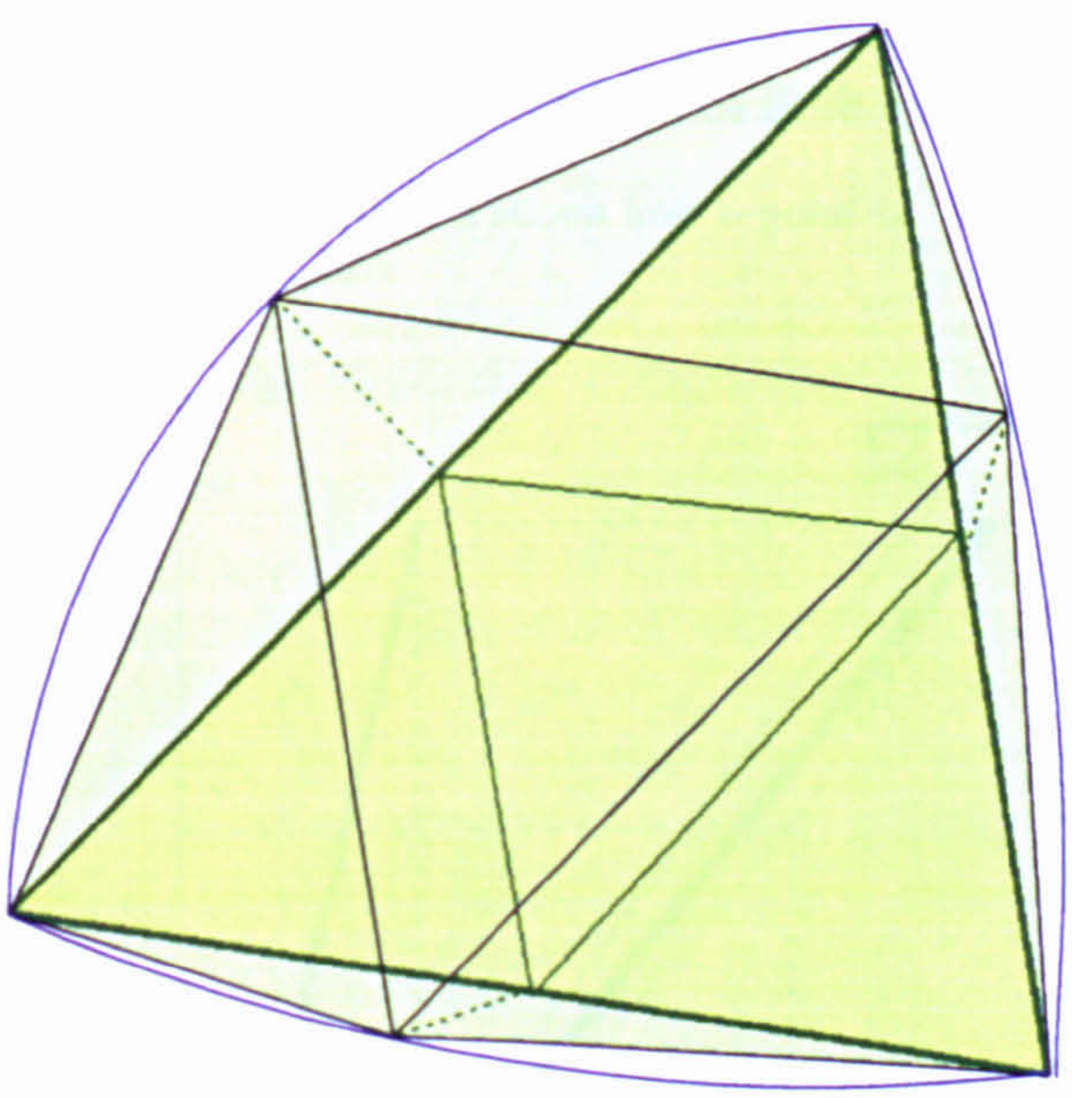


Figure IV.3: Tessellating a triangle

To tessellate a triangle, the mid-points of each side are computed. Four new triangles are obtained. Coordinates of the mid-points are then projected back onto the surface of the sphere.

Producing the hyper tessellated sphere is slow. As a result, points of the hyper tessellated sphere are not calculated at run-time, but are retrieved from a file instead.

2.2.3 How angles are computed

Computing the tessellated hyper sphere is not sufficient. These points are used to move limbs in different directions. So it is necessary to check that the positions are valid, in other words that any new direction lies in the area defined by the DOFs constraining the corresponding joint.

To check a point describes a direction that is within the DOF constraints, its angles are needed. The first angle describe the rotation on the XY plane, the second one the rotation on the YZ plane. The main axis is the Y axis, the flexion axis is the X axis, and the pivot axis is the Z axis. If the point is valid, it needs to be projected into the limb coordinate system:

Although the computations could be performed each time a point is retrieved, it would be inefficient. It is not too time consuming but thousands of these calculations may have to be performed in a flash of a second. So instead, they are performed when the hyper tessellated sphere is built, that is when the application is launched.

```
CPoint3D CLimb::Transform(CPoint3D Point) const
{
  CPoint3D TransformedPoint;

  Point *= AxisSign;                          // Multiply by the sign of the limb axis
  TransformedPoint[Axis] = Point[YY];                    // Y is the main axis
  TransformedPoint[3 - (Axis + FlexionAxis)] = Point[XX];     // X is Flexion
  TransformedPoint[FlexionAxis] = -FlexionAxisSign * Point[ZZ];       // axis
  return TransformedPoint;
}
```

Figure IV.4: Transform a point in limb space

The above method shows how a point in the tessellated sphere space is projected into the limb space.

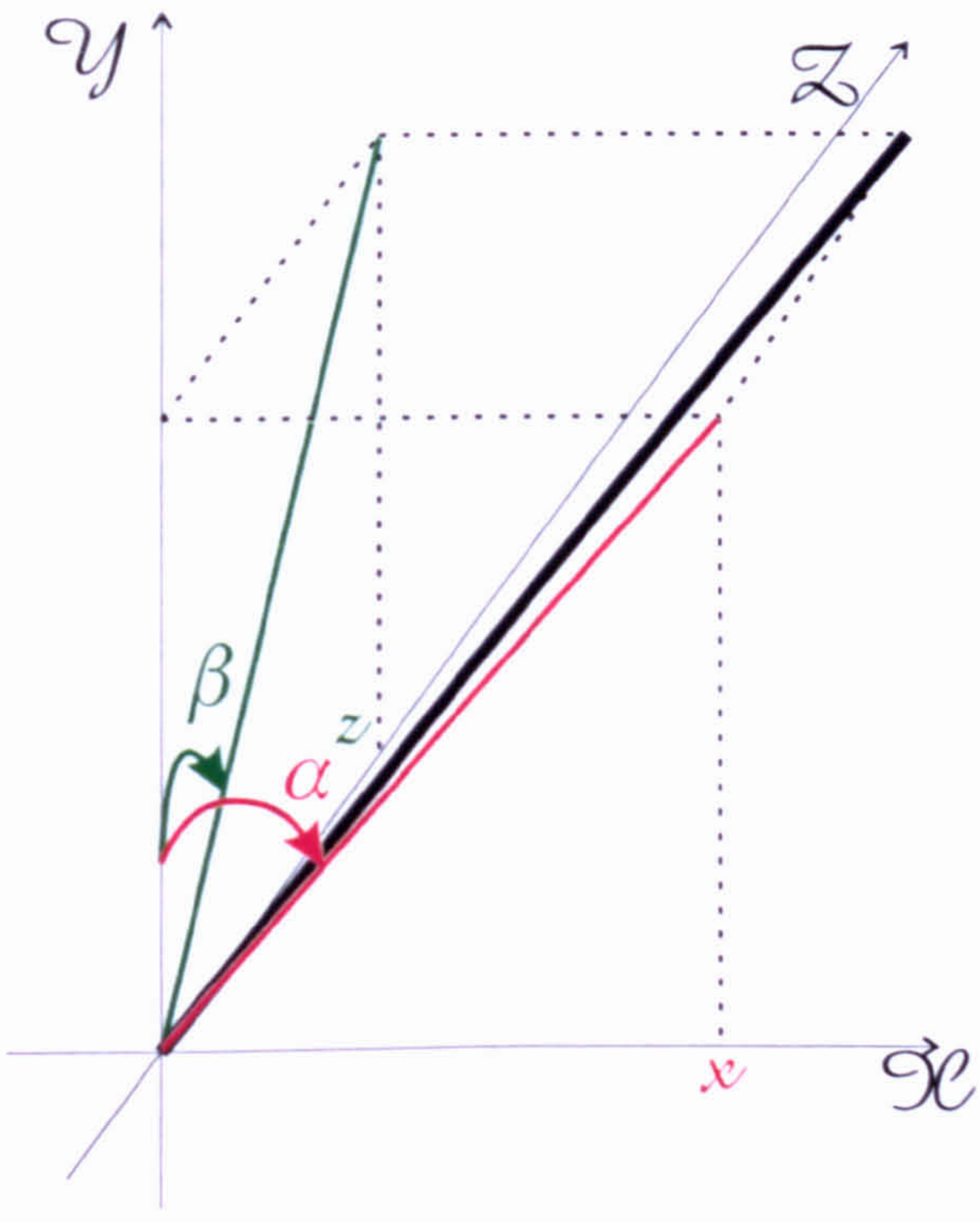


Figure IV.5: Getting angles from a direction vector

This figure shows where the angles(α and β) are and how they can be calculated knowing the x and z coordinates of the direction vector.

To compute the angles of a given point (Fig. IV.5), the x and z coordinates values of a given point correspond to:

$$\begin{aligned} x &= sin(\alpha) \\ z &= sin(\beta) \end{aligned}$$

where α corresponds to the flexion angle in radian
β corresponds to the pivot angle in radian

Knowing x and z, getting the angles in straightforward:

$$\begin{aligned} \alpha &= sin^{-1}(x) \\ \beta &= sin^{-1}(z) \end{aligned}$$

where sin^{-1} is the inverse of the sine function.

2.2.4 Rotating limbs

From the above, it can be seen that the sign of the y coordinate is not taken into account. These points are used to specify the new direction of the limbs, but there are not quite the new directions themselves. This is due to an implementation detail. To rotate a limb in a given direction, quaternions [Sho85, Sho87, Ple89, WJ93] are being used. Quaternions were discovered by Sir William Rowan Hamilton in October 1843. They are efficient and well suited to solve rotations problems.

To build a quaternion, two vectors have to be specified. The first vector represents the source and the other one is the center of rotation. if θ is the angle between the two vectors, the quaternion will rotate any points along the plane defined by these two vectors by twice that angle (Fig. IV.6). Therefore, to access any point on the surface of a sphere, if the source is the point at the top of the sphere, the center of rotation needs to lie anywhere in the top half part of the sphere. Hence, what is required is not an entire tessellated sphere but just a tessellated half sphere. So the y coordinate value can only be positive.

This also means that if a limb can only move in the area defined by angles α, β with $\alpha < \beta$, we are only interested in points lying in the area defined by $\frac{\alpha}{2}, \frac{\beta}{2}$.

To generate the matrix which will move the limb in the position specified by the direction vector and a twist angle, we first need to compute the quaternion specified by the direction vector. This is done as follows:

$$Q = (-A \bullet V, A \times V) \tag{IV.1}$$

where A is the main axis of the corresponding limb
V is the direction vector after having been projected into the limb space

The first part of the quaternion is the dot-product of A by V, that is the cosine of twice the angle between A and V. The second part is the cross-product of A by V, that is the sine of twice the angle between A and V times the axis of the quaternion normalised to unity. The quaternion is then transformed into a matrix using the following method:

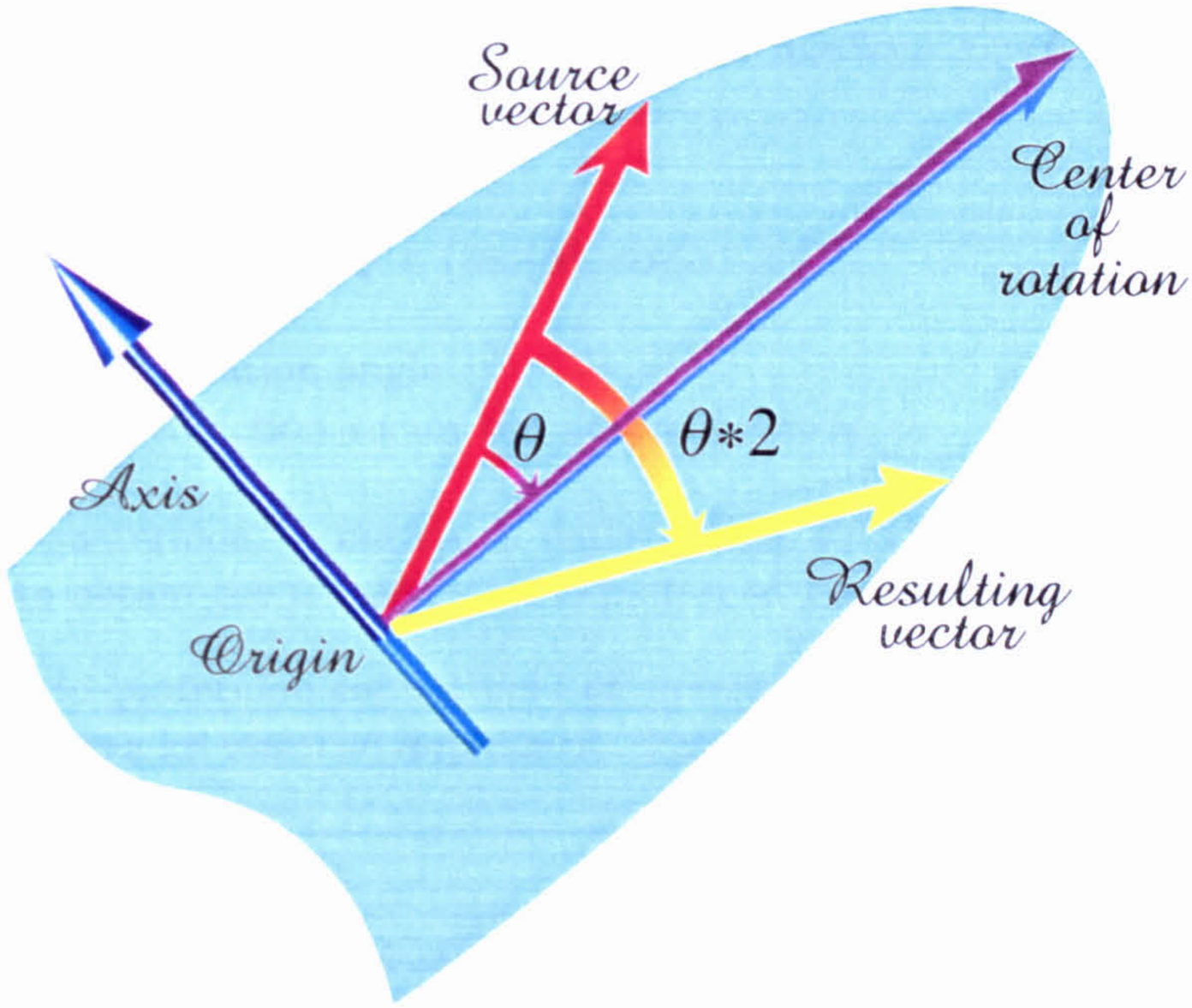


Figure IV.6: Computing a quaternion

A quaternion can be computed using the dot-product and the cross-product of the source vector and the center of rotation. If ω is the angle between the source vector and the center of rotation, the resulting quaternion will rotate objects twice this angle. Thus, rotating the source vector will produce this resulting vector.

$$\begin{aligned}
X_2 &= Q_x \times 2 \\
Y_2 &= Q_y \times 2 \\
Z_2 &= Q_z \times 2 \\
X_Y &= Q_x \times Q_y \times 2 \\
X_Z &= Q_x \times Q_z \times 2 \\
Y_Z &= Q_y \times Q_z \times 2 \\
W_X &= Q_\alpha \times Q_x \times 2 \\
W_Y &= Q_\alpha \times Q_y \times 2 \\
W_Z &= Q_\alpha \times Q_z \times 2
\end{aligned}$$

$$\begin{bmatrix}
1 - Y_2 - Z_2 & X_Y + W_Z & X_Z - W_Y & 0 \\
X_Y - W_Z & 1 - X_2 - Z_2 & Y_Z + W_X & 0 \\
X_Z + W_Y & Y_Z - W_X & 1 - X_2 - Y_2 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \tag{IV.2}$$

where Q_α is the angle part of the quaternion(the result of the dot-product)
$Q_{(x,y,z)}$ are the coordinates of the axis of rotation of the quaternion
(the result of the cross-product)

Now, only the twisting transformation is left to be added to the newly obtained

matrix. Again, this is done using a quaternion. The direction vector (normalised to unity) the limb is pointing to (relative to its parent) times the sine of the twist rotation angle becomes the axis of the quaternion. Mathematically, the quaternion is computed as follows:

$$Q = (cos(\alpha), sin(\alpha) \times D) \tag{IV.3}$$

where α is the twist rotation angle

D is the unit direction vector the limb is pointing to

From this quaternion, a matrix is derived (Eq. IV.2) and is multiplied by the previous one to obtain the transformation matrix at the current joint.

Since a single quaternion cannot hold all necessary transformations, an algorithm using matrices was a lot easier to implement. This is why all calculations are eventually stored in a matrix.

2.2.5 Structure of alternatives

A class has been specially built to hold all the necessary information:

```
class CAlternative {
        Point;              // Centre of rotation
        FlexionAngle;       // Corresponding flexion angle
        PivotAngle;         // Corresponding pivot angle
        Number;             // Identification number
        Level;              // Level of decomposition
        Neighbours;         // Closest neighbours at each level
}
```

Figure IV.7: Structure of an alternative

This structure contains all the necessary information to generate a quaternion. It also contains the angles of the rotation thus making it easy to check that the rotation will be allowed by the DOFs.

- **Point:** The points defined by its 3D coordinates.
- **FlexionAngle:** The angle on the XY plane
- **PivotAngle:** The angle on the YZ plane
- **Number:** The index of this alternative
- **Level:** Which hyper tessellated half sphere
- **Neighbours:** The set of neighbours at the different levels

2.3 Getting a first point

The interface of the technique resembles in many respects the interface of conventional interactive genetic algorithms. Typically, a set of nine poses for a given articulated figure is produced and rendered in a dedicated window. This means that nine is the size of the population. This is very small, therefore we need to make the best use of it.

For a given limb, the computer needs to show it in as many different positions as possible to ensure the space is well explored. For this purpose, when trying to find a set of possible positions for a given limb, the search starts at the crudest decomposition level. All the points at this level which lie in the area described by the DOFs of the corresponding limb are selected. If the number of points selected is higher than the size of the population, the selection stops, otherwise the search continues but at a finer detail until the number of points selected is equal or higher than the size of the population.

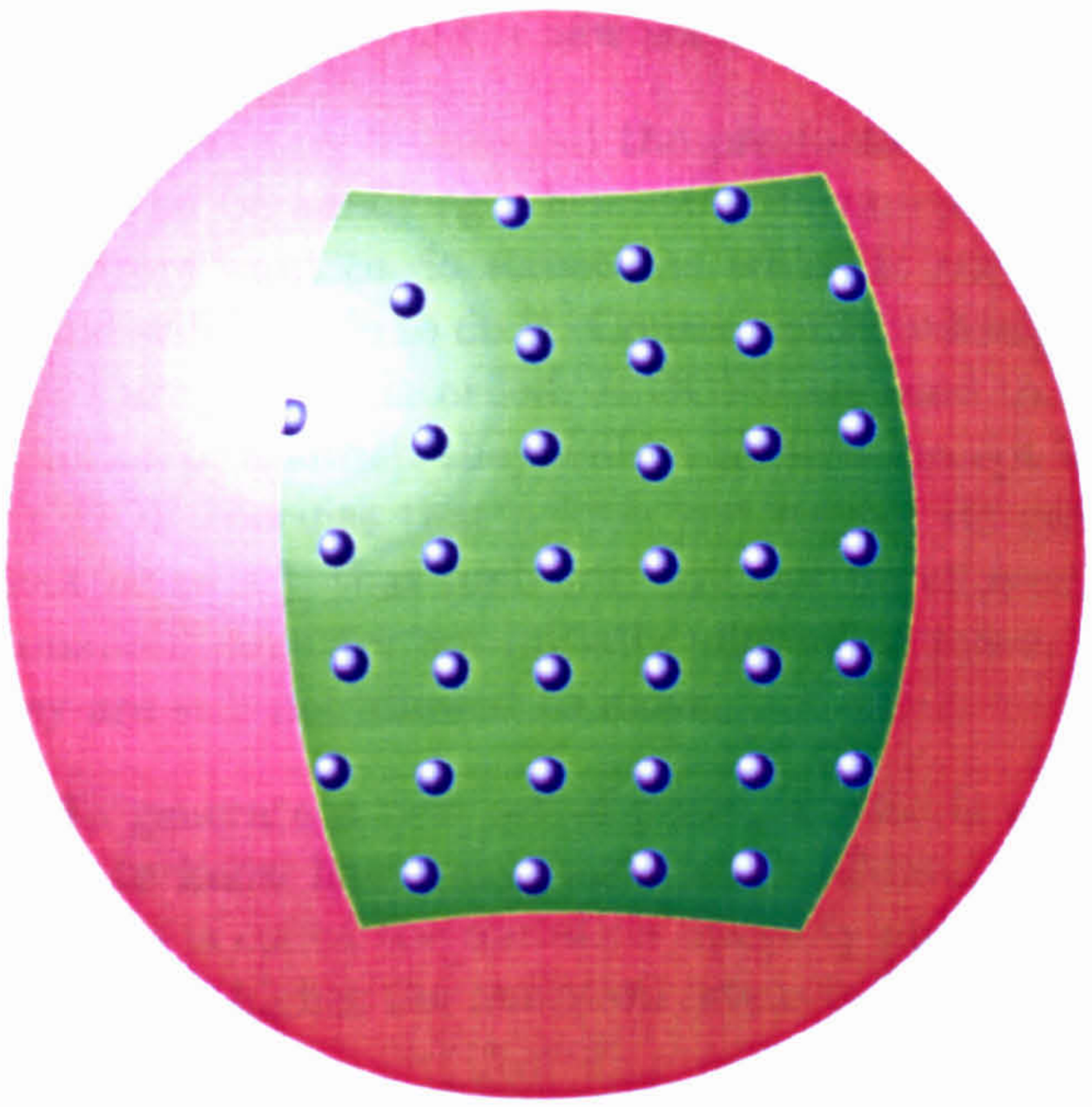


Figure IV.8: Valid alternatives

To move a limb, a sufficient number of valid alternatives has to be reached first. The process starts at the coarsest level and each valid alternative is selected. If not enough alternatives have been retrieved, the process is restarted at one finer level until a minimum number of alternatives has been obtained.

When enough configurations for each limb have been obtained, they are used to produce the set of poses. At first, this used to be a random process (that is, to define one joint, every configuration for that joint had an equal chance of being used). However, it was clear that some configurations were more important than others (some configurations are more likely to be found in poses at some joint than others). For instance, configurations involving only the flexion axis were more useful than configurations involving both flexion and pivot. Configurations belonging to coarser level of decomposition also tended to be more important.

So, a procedure was implemented so that these configurations were more likely to be selected than others. For this procedure to work properly, configurations had first to be sorted. The first sorting criterion is whether the solution alternative involves

only a flexion motion. The second criterion is the level of decomposition, the crudest solution alternatives being first.

When debugging this code, to check that the sorting procedure was working properly, solutions were not displayed randomly but from the first to the last. Surprisingly, this simple scheme was much more powerful than a pseudo-random selection. Random selection is one of the main driving force for interactive genetic algorithms but, here, it just shuffles a clearly recognizable order. This order is always symmetric and eases considerably the search for good solution. This is a physiological phenomena, patterns clearly appears from the order. Pseudo-random selection simply destroys these patterns and henceforth considerably harden the user's task.

2.4 Getting next points

Once an articulated figure has been positioned, it is used as a seed to produce a new population of poses. At the limb level, the problem is that knowing the position of the limb, how to get a new set of positions.

DOFs are used to forbid the production of impossible poses. However, enforcing DOFs can be annoying when some uncommon poses have to be produced. So, when animators want to do something which is normally not allowed by the DOFs, they should still be able to do it. Consequently, when generating a new population of poses, DOFs are partly discarded. If DOFS are used to restrict limbs of going too far in one direction or another, they will be ignored, except if it is the first generation. However, if a DOF specifies that a limb just cannot do certain type of motions (e.g. flexion, pivot or twist), that limb will still not be allowed to do them. For example, for the humanoid, hips are not initially allowed to move. When generating new populations, they are still not allowed to move.

To generate a new set of poses, a pose is used as a seed but the computer still needs to know how much variation is allowed to generate the new set of poses. The variation is called the mutation intensity and it is specified by the user by means of a slider. The higher the mutation intensity is the more different generated poses will be from the seed.

The mutation intensity is then translated into a maximum distance from the limb positions of the seed. For each joint position, the most important solutions are selected to be displayed later on. The process is divided into two stages:

1. At the first stage, all the positions which distance is smaller than the maximum distance are stored into a temporary array.

2. Starting from the coarsest level of decomposition, all positions which belong to this level are stored into an array until the size of the array is greater than the population size (Fig. IV.9).

The procedure in Fig. IV.9 will return a set of positions which can then be used to

position the corresponding limb. Again to make the best use of this set, first entries are displayed first.

```
/*
 * Retrieve all alternatives which are close enough
 */
for (i = GetNbAlternatives() − 1; i >= 0; i−−) {
        P2 = Alternatives[i]−>GetPoint();
        Dist = Distance(P, P2);
        if (Dist < MaxDistance) AllValids += Alternatives[i];
}

/*
 * Retrieve alternatives at the coarsest level
 */
Level = MaxLevel;
for ( i = AllValids.GetNbElems(); i > 0;)
        if (AllValids[−−i]−>GetLevel() == Level)
                ValidAlternatives += AllValids[i];

/*
 * Retrieve at finer levels until there are enough
 */
while (Level > 0 && GetNbValidAlternatives() < IdealNumber) {
        Level−−;
        for ( i = AllValids.GetNbElems(); i > 0;)
         if (AllValids[−−i]−>GetLevel() == Level)
                ValidAlternatives += AllValids[i];
}
```

Figure IV.9: Sorting alternatives

Alternatives are used by order of *importance.* So they are sorted, flexion only and coarsest alternatives first.

where *Alternatives* is the array of positions
AllValids is the temporary array containing all positions within the required distance
ValidAlternatives is the array containing the most important positions within the required distance

3 Genetic structure

Although the technique which has been implemented is quite different from an IGA, the vocabulary has been kept.

3.1 Gene

Each gene is associated to a limb. It defines the configuration of this limb. The structure of the gene is approximately like this:

```
class CGene {
        FlexionPivot;          // Pointer onto a CAlternative object
        Twist;                 // Specify the configuration of the joint
        FlexionDof;            // Minimum and maximum flexion
        PivotDof;              // Minimum and maximum pivot
        TwistDof;              // Minimum and maximum twist
        ValidFlexionPivots;    // Set of flexion & pivot to choose from
        ValidTwists;           // Set of twists to choose from
        PreviousAlleles;       // Configuration already selected
}
```

Figure IV.10: Gene structure

A gene completely specifies a joint configuration. It also contains the DOFs of that joint so that it can make sure the configurations it builds are valid. The last three attributes specify what are the valid alternatives and the one which have already been selected. They are shared by all genes.

The first two attributes specify the configuration of the associated limb. The next three attributes define the DOFs of the corresponding joint. When a particular position has been used, its corresponding alternative number is added into the *PreviousAlleles* attributes which behaves like a bucket. Thus, when new configurations are needed the ones inside this bucket will not be selected again. When all possible solution alternatives have been selected, the bucket is emptied. *ValidFlexionPivots* is an array containing the set of valid positions to choose from to produce a new configuration. *ValidTwists* is the same except that the set of valid positions is only for twists. Both attributes are shared by all genes.

3.1.1 The mutation process

There are two types of mutation:

1. Mutations that only concern flexion & pivot.

2. Mutations that only concern twist

Working on all three types of rotation at the same time is not efficient at all. First, there is a combinatorial explosion and second it makes it a lot more difficult to work out what is useful and what is not. The type of mutations that is enabled at any one time is specified by the user.

When a gene is being mutated, it looks in *ValidFlexionPivots* and *ValidTwists* to see whether there is any valid positions and if any, chooses the first one which has not been selected yet. The ID of the solution alternative selected is then added to the bucket (*PreviousAlleles*).

3.2 Chromosome

The chromosome is a structure which holds the set of genes. It is also in charge of interpreting the genes and of generating the phenotype. A gene contains all the information necessary to produce the configuration of its associated joint, that is the matrix which defines the position and direction of the corresponding limb and its children. The phenotype is simply the set of joint configurations, that is an array of matrices.

The field *FlexionPivot* of each gene points onto a solution alternative. For this solution alternative, the X axis stands for the flexion axis, the Y axis stands for the main axis of the limb and the Z axis stands for the pivot axis. Joints usually use different axes. So, to produce the phenotype, the chromosome takes all its genes in turn, and for each, if there is a *FlexionPivot* solution alternative, it is projected into the limb space and transformed into a rotation matrix. If there is also a *Twist* position, it is translated into a rotation matrix which is multiplied by the previous one. The result completely specifies the configuration of the corresponding joint (Fig. IV.11).

```
for (i = 0; i < GetNbGenes(); i++) {
      Allele = Genes[i]->Allele;
      if (Allele->FlexionPivot)
       Matrix = Skeleton->Limb[i]->TransformToQuaternion(Allele->FlexionPivot);
      else
       Matrix = Identity();
      if (Allele->Twist)
       Matrix *= ToMatrix(Allele->Twist);
      Limbs[i]->SetJointMatrix(Matrix);
}
```

Figure IV.11: Building the phenotype

For each gene, flexion, pivot and twist rotations are gathered in a rotation matrix which is given to the corresponding joint.

3.2.1 The mutation process

Using a conventional GA, mutation is only used to create new genetic material, in the hope it will be useful. It is not the main driving force. The latter is the reproduction process which brings good individuals together to produce even better individuals. In the original population, most of the genetic material is already there, so mutation is mainly used to bring back genetic material which may have disappeared.

With an IGA, things are different. The original population is too small to contain all the possible genetic material. As a result, mutation is typically much higher and as important as the reproduction process.

Using the generator, mutation is the only driving force (apart from the user). Therefore, we have to make the best use of it. When a chromosome has to be mutated, an array of boolean flags, one for each gene, is used. If the flag is enabled, the

corresponding gene is mutated. With the generator, unless specified not to do so, there is no point in not mutating a gene. This would simply be a waste of space since the resulting limb position would be the same as the seed. Consequently, all flags are enabled and all genes are mutated. However, it can sometimes be a hindrance. When trying to produce a pose, some limbs will be positioned faster than others simply because the ideal position was found faster. In the next generations, there is no point in mutating these limbs. If they are still mutated, it makes the job of the user harder. For example, if the arm is already set but the position of the hand is not quite perfect, it is more difficult to work on the position of the hand if the position of the arm changes. So, the user is allowed to enable or disable the body parts which will be allowed to move.

```
void Mutate(const aInt& Masks)
{
        for (i = 0; i < GetNbGenes(); i++)
                if (Mask[i]) Genes[i]->Mutate();
}
```

Figure IV.12: Mutation

For some reasons, some genes are not allowed to mutate. So, an array of booleans, one for each gene, is specified. If the boolean associated to a gene is set, that gene is mutated.

4 Search Engine

The search engine is inherited from the first implementations when the application relied on a GA to pose an articulated figure. As a matter of fact, it can still be used by a true GA with no modification at all. With the generator, most parameters are simply not used.

In input, the search engine is given a population of chromosomes. The goal of the search engine is to produce another population of chromosomes. Normally, each chromosome also has a fitness value so that the fittest individuals will be used to generate the output population. The selection of individuals to produce new individuals could be performed in several ways, so a variety of genetic selection mechanisms have been implemented. The default one is called tournament selection and it is this one which is used by Generator. Typically, when the search engine needs a chromosome, this selector is allowed to choose between n chromosomes and returns the best.

With Generator, a chromosome is extracted from the seed pose which is used to create the input population. So the genetic selector is not important, it will always return the same thing, that is a copy of the seed chromosome.

Using a conventional GA, a variety of genetic operators may be used to produce the new population, one or more being able to operate at the same time. Each operator

has a probability of being used. With the generator, only the mutation operator is being used. It requires one chromosome in input and will produce a single chromosome in output.

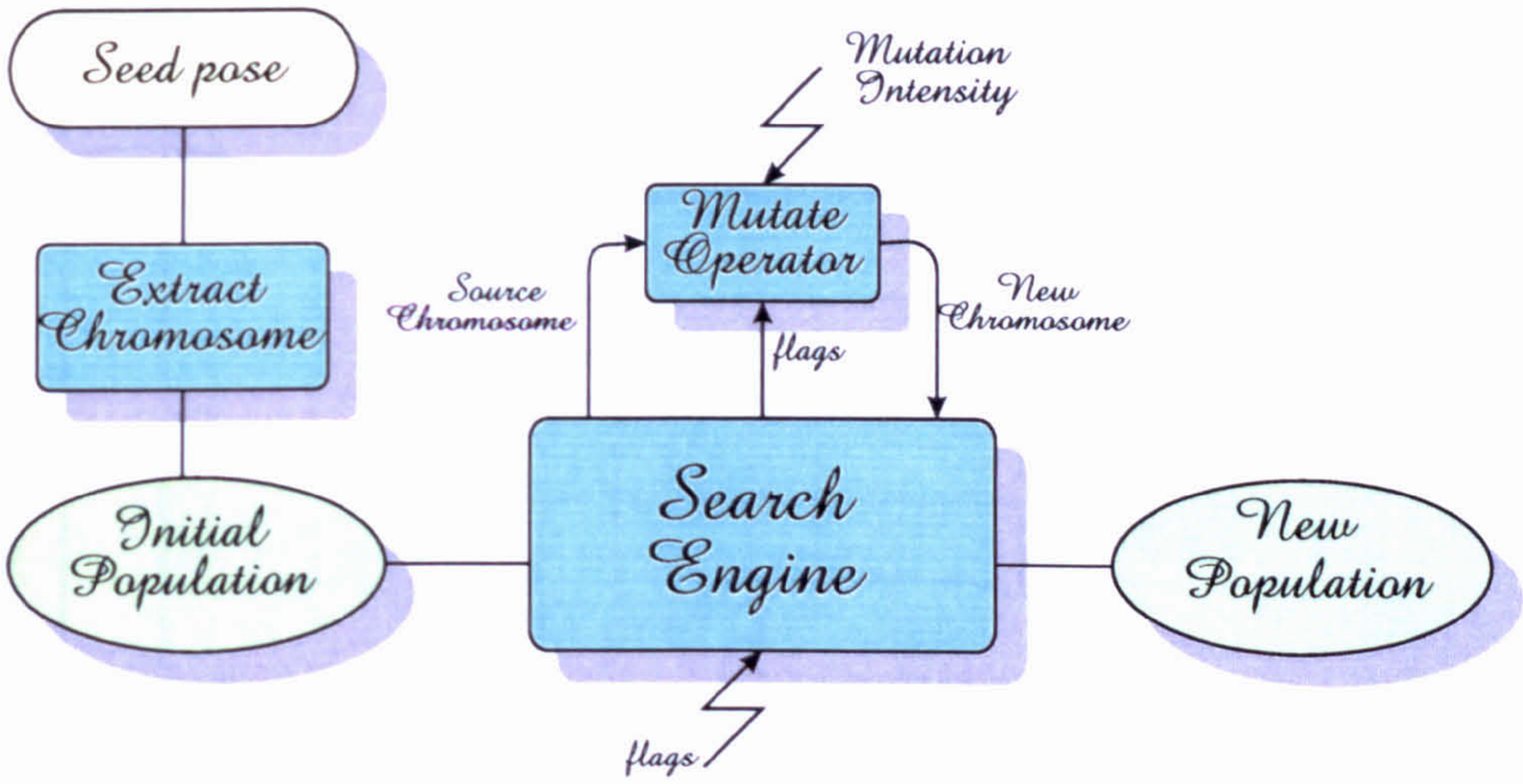


Figure IV.13: Search engine

A population of chromosomes made from the seed pose is specified to the engine which will produce a new population of chromosomes ready to be decoded and displayed.

Before the production of each new population, valid solution alternatives for each of the joints have to be selected and stored in an array shared by all genes (Fig. IV.14).

For the first generation, a dummy chromosome is created and each gene is called in turn to select a valid set of alternatives which is then stored in the array of valid solution alternatives. DOFs constrain the selection process, so that useless poses cannot be produced.

For the next generations, the poses built by the user are used as seeds. A chromosome is extracted from this pose, and each of its genes is asked to produce a new set of valid solution alternatives, given a maximum distance.

The application is now ready to produce the first or the next population of chromosomes. During the mutation process, each gene will retrieve the next unselected solution alternative. At the end, chromosomes will generate the phenotypes which will be used to produce the poses. These poses are then rendered and displayed.

5 Interface

5.1 Producing the first population

When the application is launched, a set of nine standing articulated figures is displayed. Although there is no restriction to the type of articulated figure which can

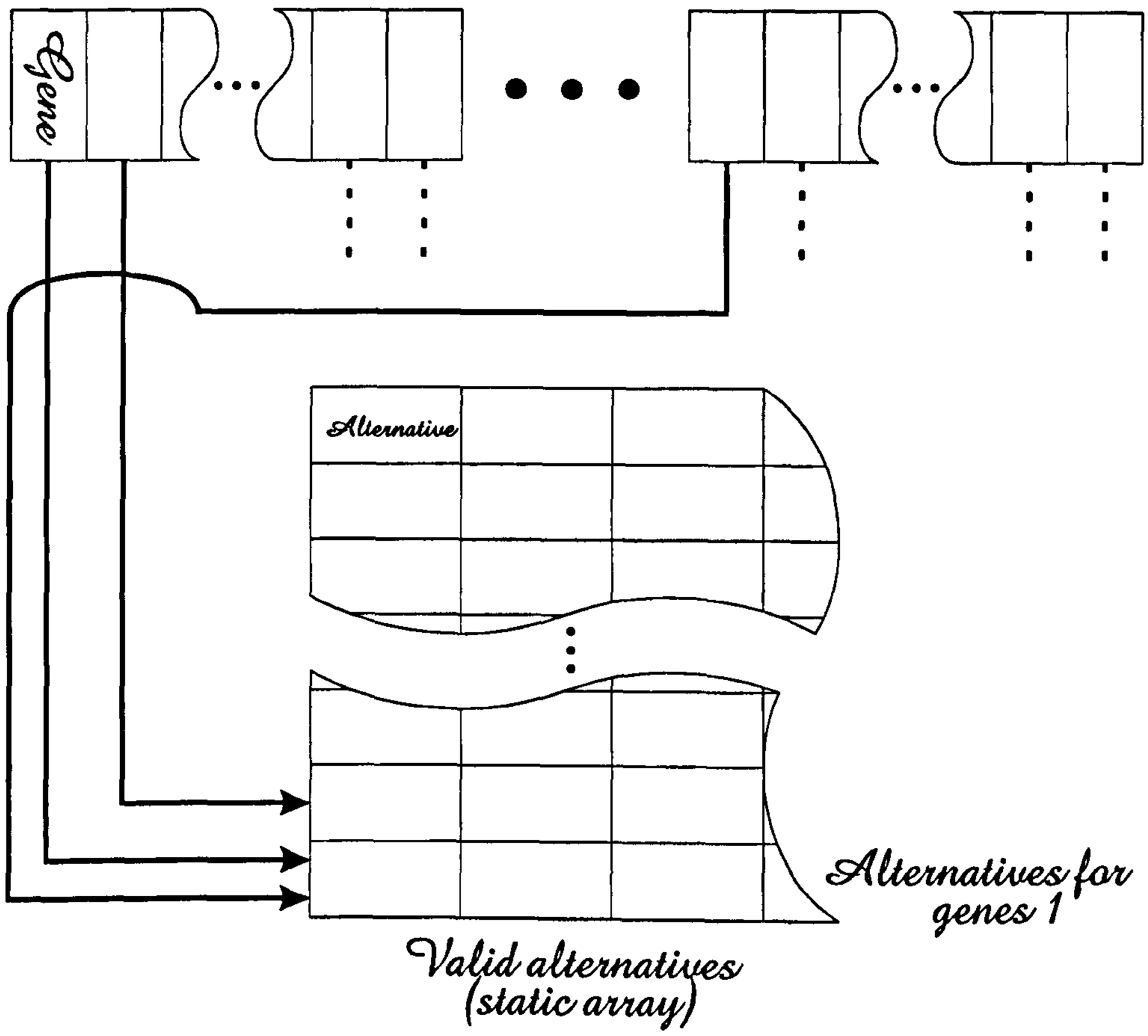


Figure IV.14: Array of valid alternatives

> Valid alternatives (that is the main ones which do not break constraints) for each gene are first selected. They are then stored in an array statistically declared, that is an array shared by all genes. Thereafter, each gene will retrieve an alternative and mark it as selected so that it will not be selected again.

be implemented, only a humanoid has been encoded to test the system.

The first generation will create a population of poses for which each limb position is sorted by order of importance, that is flexion and coarsest position appear first. DOFs are also used to avoid the generation of impossible or unlikely positions. For a given joint, the space of configurations is made of two sub-spaces which are the space of flexion&pivots rotations and the space of twist rotations. Although these two spaces could be searched at the same time, in practice, this is too inefficient. If the first space as cardinality n and the second has cardinality m, the joint space (the space of joint configurations) has cardinality $n * m$. This is a space of much bigger size, which is consequently much more difficult to search through. The user selects which space to search through and nearly always, the first generation is used to search the space of Flexion&Pivot type positions. They are the ones which carry the most information. If twists are necessary, they are used at the end.

After the poses have been generated, they are displayed (Fig. IV.15). The number of poses displayed corresponds to the size of the population. Nine poses was found to

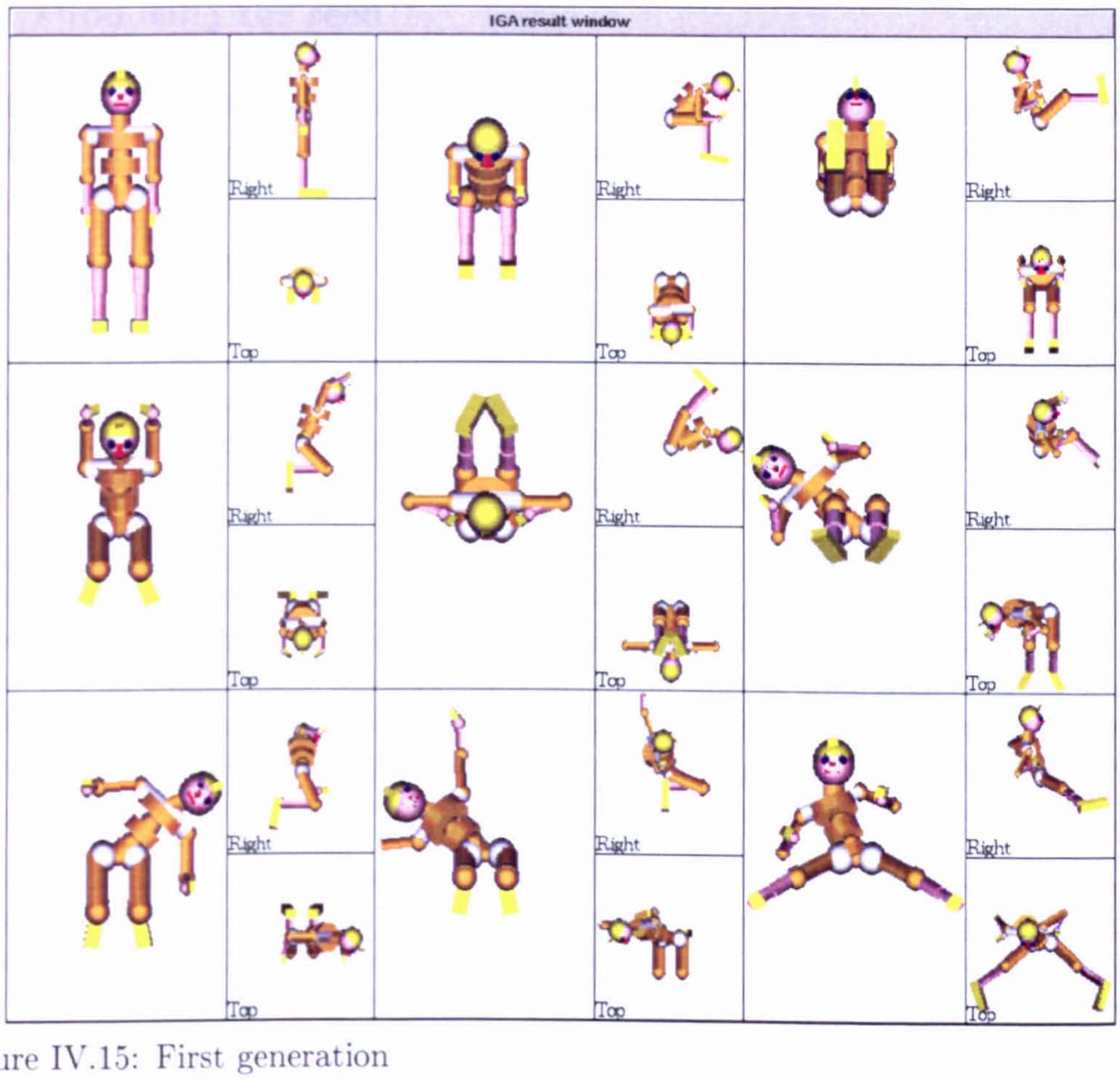


Figure IV.15: First generation

At the first generation, a set of nine poses are displayed. They all differ strongly from each other. This allows users to produce many types of pose very quickly using just what is being displayed.

be an ideal number, it is neither too small nor too big. If the size of the population is too small, the computer cannot display enough information to make the interface interesting. If it is too big, it is very difficult to make everything fit on the screen. With a bigger screen, it might be worthwhile to investigate the use of the interface with a bigger population.

Once built, the articulated figures are rendered on their own sub-window. These sub-windows are composed of three views. The main view displays the articulated figure at its original size. A camera is used when rendering the articulated figure. Rotating the camera allows the user to see the figure from different angles. Beside the main view, the articulated figure, viewed from the right and the top, is also displayed at half its original size. Side views are very helpful when some limbs are hidden or when it is difficult to work out their true positions.

5.2 Producing the seed

The next task for the user is to assemble a pose which will be used as a seed to generate the next population. Initially, this pose represents a standing figure.

5.2.1 Selecting body parts

At the first generation, some interesting positions have already been produced. There is also a great deal of variety and so most of the configurations that the user will be interested in will be there. The user can select them directly from the screen. Interesting joint configurations can be selected by clicking on the corresponding limb.

A 2D point is obtained. This point truly lies on the projection plane. So, by knowing the projected positions of each limb, it is easy to work out which one is the closest. However, this has a problem. It is not possible to work out which limb is in front. For example, what the user does when selecting a limb is to click on what is visible, that is its graphic representation. The projected skeleton is just a set of line segments. So, using it, the computer might select an equally close limb or even closer limb, but one which was hidden by the one the user truly wanted to select.

A better implementation would be to use what the user expects, that is the representation of the articulated figure. A ray starting from the position of the camera and cutting the projection plane at the position clicked by the user is easily obtained. This ray is in world coordinates and intersects the graphic representation of the limb to select. Because we do not have the coordinates of the limbs in this coordinate system (although they could be easily obtained), the ray is projected back in the articulated figure coordinate system.

Cameras are used to project objects composing the scene to obtain their new coordinates before rendering them. One might think that using the inverse of the projection matrix of the camera would project back projected points into their original position. However the projection matrix is not invertible[3]. So instead, the transformation matrix, that is the translation concatenated to the rotation of the camera, is used.

Applying the inverse of this transformation projects back coordinates to their original value:

$$\begin{aligned} P_s &= [0,0,0,1] \times M^{-1} \\ P_f &= C \times M^{-1} \end{aligned} \tag{IV.4}$$

[3]All points from a single line cutting the lens of the camera will be projected onto exactly the same point on the projection plane. There is no way the computer can work out the original position of a point if it only has its coordinates on the projection plane.

where C is where the ray cuts the projection plane
M is the transformation matrix associated with the camera
P_s will produce the position of the camera. So in fact, because it is already known, this operation is not necessary.
P_f is now just a point belonging to the ray which will enable us to find the equation of the latter.

So, the limb to select is the one which will cut the closest object representing it. An intersection algorithm was devised for each graphic primitive (sphere, cylinder, cube and cone). They are all based on some simple geometry principles such as the intersection of a line and a point, the intersection of two lines and the intersection of a line and a plane [Gla90].

The intersection of a point and a line is used with the sphere and is performed as follows:

The ray is represented by the following equation:

$$L_r(s) = P_r + V_r s \tag{IV.5}$$

then the closest point of the ray to the point is:

$$s = \frac{(P_r - C) \bullet V_l}{|V_r|^2} \tag{IV.6}$$

$L_r(s)$ will give the closest coordinate of the ray to the 3D point C. If C is the centre of the sphere and the distance between the $L_r(s)$ and C is less than the radius, then the ray intersects the sphere.

For the intersection of two lines, we have the following equations:

$$L_r(s) = P_r + V_r s \tag{IV.7}$$

$$L_c(t) = P_c + V_c t \tag{IV.8}$$

The closest point on the two lines is:

$$t = \frac{((P_c - P_r) \times V_r) \bullet (V_r \times V_c)}{|V_r \times V_c|^2} \tag{IV.9}$$

So if the second equation defines a cylinder or a cone, and P_c is one of the sides, and V_c is the unit vector of the cylinder or the cone respectively, the one thing to make sure is that the value of l lies between 0 and 1. Otherwise the ray might intersect the

line of the cylinder or the cone but not these objects themselves. s and t will define two coordinates and if the distance between them is smaller than the radius, the ray intersects these objects. For the cone, things are a bit trickier and we omit the details.

A cube is made of polygons and each of them describes a plane. To see if a ray intersects a polygon, we need to find the point at which the ray will intersect the plane and if this point is inside the polygon.

A plane is defined by two terms, let's say J_N and J_d. J_N is the normal of the plane and J_d indicates the position of this place. J_d is computed like this:

$$J_d = -P \bullet J_N \tag{IV.10}$$

where P is a point of the plane.

From the previous equation, we can replace P by a point somewhere on the ray:

$$(P_r + V_r s) \bullet J_N + J_d = 0 \tag{IV.11}$$

Solving for s, we get:

$$s = -\frac{J_d + P_r \bullet J_N}{V_r s \bullet J_N} \tag{IV.12}$$

A polygon of n vertices can be viewed as a set of $n-2$ triangles. So if P is a point inside a triangle which vertices are V_0, V_1 and V_2, then

$$\overrightarrow{V_0P} = \alpha\overrightarrow{V_0V_1} + \beta\overrightarrow{V_0V_2} \tag{IV.13}$$

and $\alpha >= 0$, $\beta >= 0$ and $\alpha + \beta <= 1$

The limb to select is the one whose representation was intersected by the ray and is the closest to the camera.

In some occasions, it may be difficult to select one limb that way. This is the case when the limb to select is pointing straight towards the camera. In such a case, neighbouring limbs are likely to be selected (Fig. IV.16). In some cases, limbs behind the one to select but closer to the ray defined by the mouse click will be selected. In other occasions, it might be interesting to select more than one limb at once. In such cases, the user simply draws a rectangle and all the body parts wholly inside it are selected. Using the camera, a three-dimensional rectangular region is obtained and all limbs lying entirely inside are selected. This rectangular region is defined by four lines, all parallel to the camera direction. To test if a limb is inside it, it is only necessary to check that its projected X and Y coordinates are inside the rectangle(Fig. IV.17). Any views, the main one or the two small ones beside it, can be used to select the

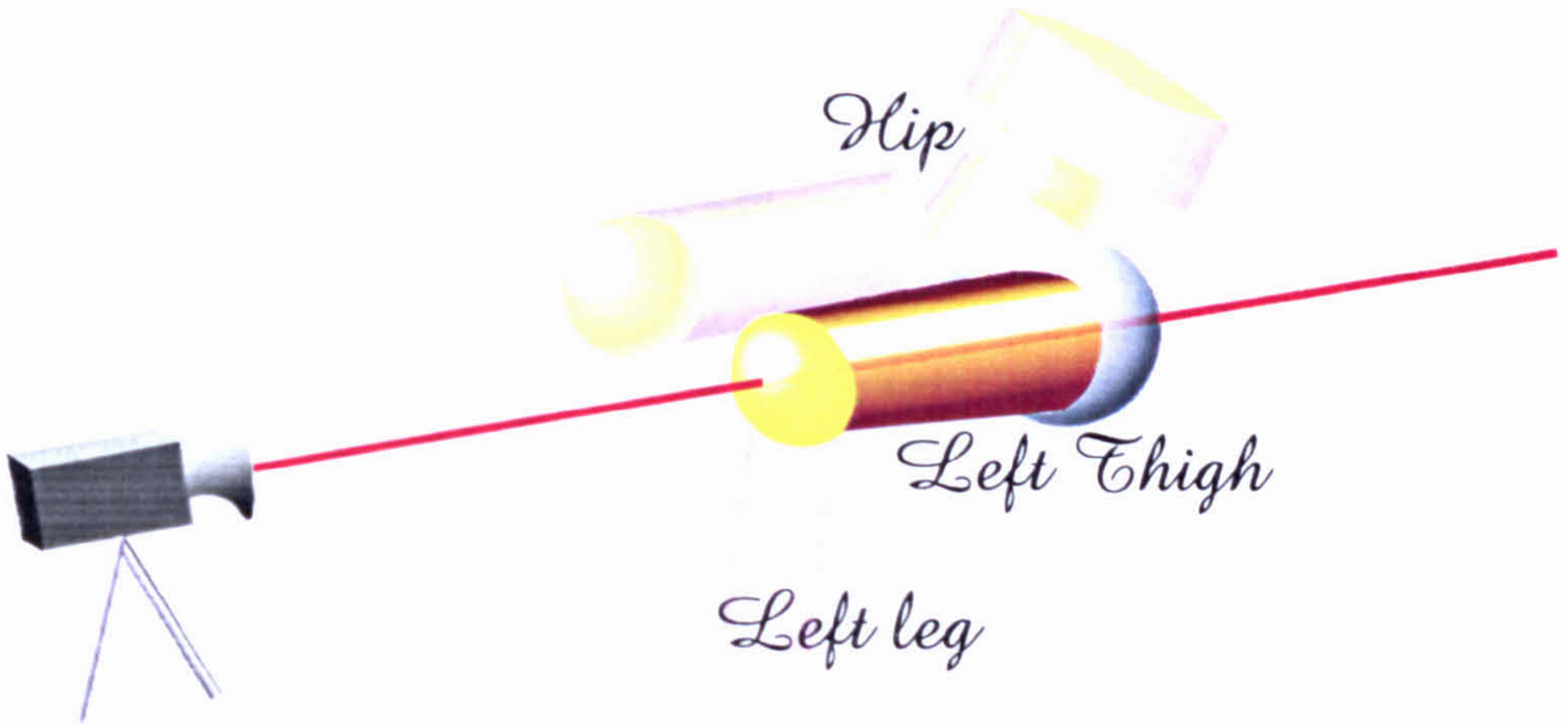


Figure IV.16: Selection ambiguity

In cases illustrated by the above picture, the selection is ambiguous. The Hip will probably not be selected because it is hidden. Although the user probably wanted to select the left thigh, it is likely that the left leg will be selected because it is in front. If the ray is closer to the hip, then it is this one which will be selected.

limbs. The small views are useful when it is difficult to perform a selection using the main view.

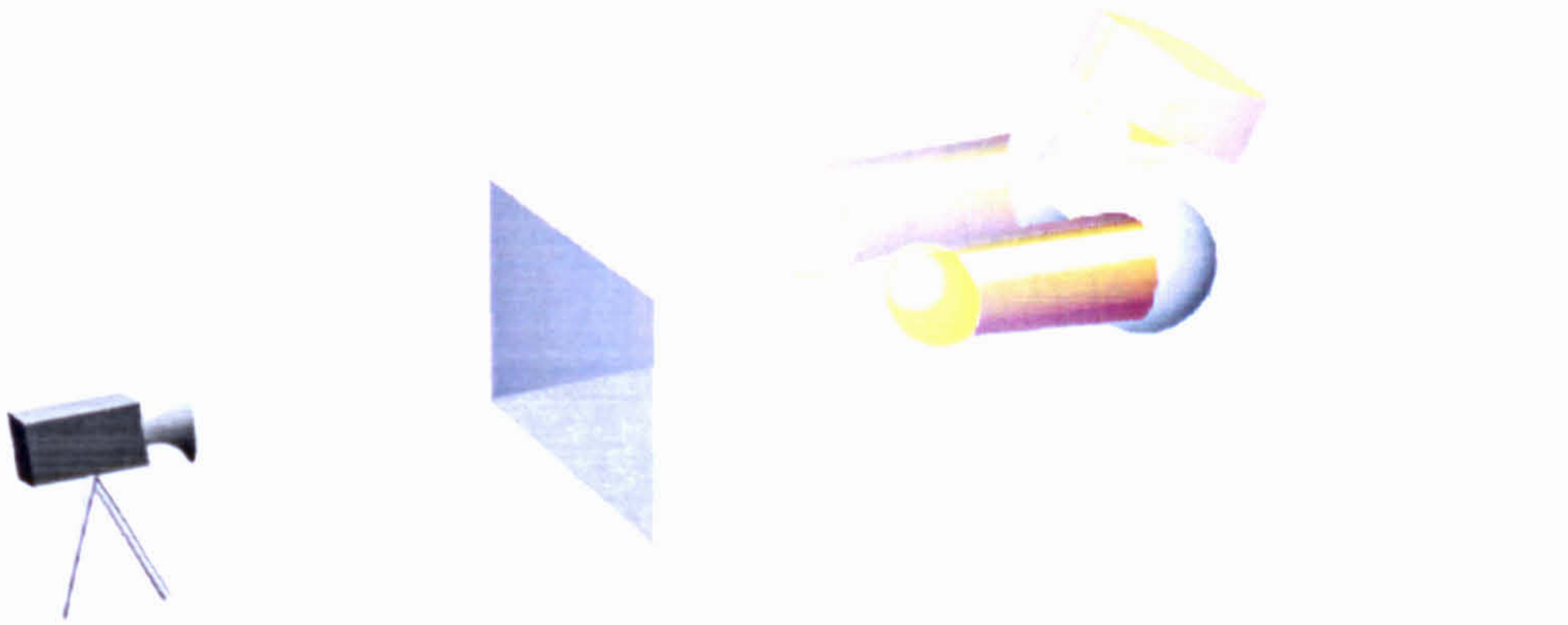


Figure IV.17: Selecting a group of limbs

To solve the problem shown in Fig. IV.16 and to allow for multiple selection, the user can draw a rectangle on the screen. A volume is deduced and all limbs lying entirely inside are selected.

When a limb is selected, it is copied in the seed pose. No confirmation is asked. The seed pose is then rendered and displayed. This pose is displayed in a separate window beside the main one (Fig. IV.18).

When selecting valid alternatives for joints, the process tries to retrieve a minimum of n positions, n being the size of the population. Rarely will it retrieve exactly n positions. Sometimes it will retrieve up to three times more. The selection mechanism works at a decomposition level granularity. First, it selects all alternatives which may

be displayed independently of their importance. Then it selects all alternatives at any given details, starting from the coarsest to the finest until there are at least n alternatives (Fig. IV.9). Consequently, it may select more alternatives than it was asked for. Some of these alternatives will not be displayed because they were judged not important enough. Although this is usually the case, they will still be required for some poses. To reach them, the user would have to build a seed pose with what there is and ask the computer to produce a new generation. Since this is a bit slow, the concept of pages was used. For each generation, three pages of poses are computed which can be accessed at any time. Thus, if a configuration cannot be found on the first page, it may well be on the second or third one (although rarely the third page is used). Pressing N brings the next page whilst pressing F brings the first page. Hot keys were used to increase the interactivity of the system. The implementation of this concept resulted in a great speed-up.

5.3 Producing next generations

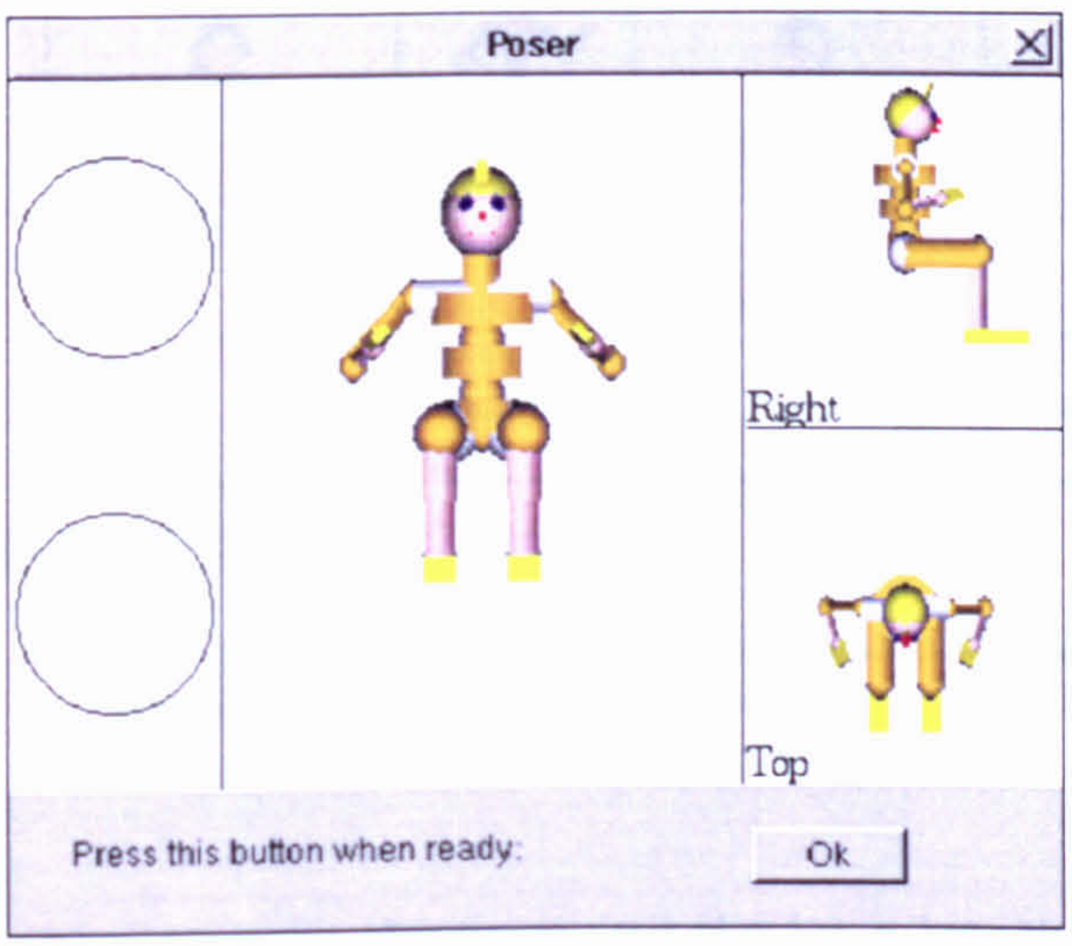


Figure IV.18: Pose builder

Using the set of poses generated in figure IV.15, this pose was constructed. It might represent a person sitting on some invisible chair and stretching the arms.

When the user is more or less satisfied with the seed pose (Fig. IV.18), the next generation can be produced. Because all poses are derived from the same seed, generated poses are all more or less similar to their creator. The higher the mutation intensity, the more different the generated poses will be. The mutation intensity is directly controlled by the user by means of a slider. To give an idea of how strongly limbs will be affected by this parameter, cones are rendered on the skeleton of the articulated figure and displayed onto a dedicated window(Fig. IV.23). They are used to show the area where the positions the computer will come up with will lie.

Using the first population, a good approximation of the target pose is easily produced. The second generation is usually used to search the twist rotations space.

In this example, the arms have to be rotated so that the hands point towards the hips and the right leg also has to be rotated so that its foot lies on top of the other leg. This will be done by selecting mutations which only perform twists. Because

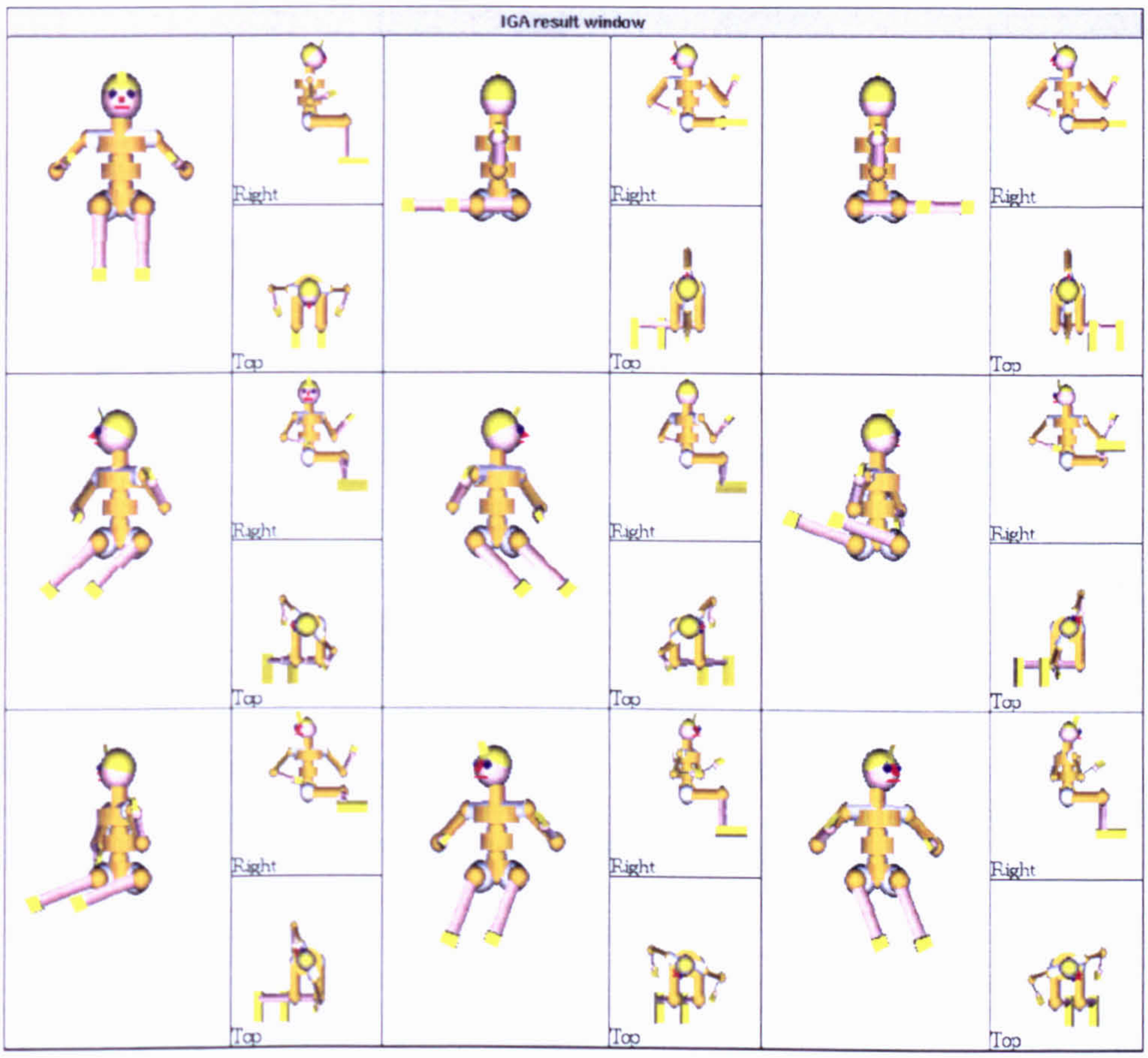


Figure IV.19: Second generation

This generation is for twist type rotations. Notice that the torso has also moved: this may be a hindrance because it makes it harder to see the positions of the arms. Only limbs shown in orange color can twist.

we want to twist the arms and the leg a lot, the mutation intensity is brought to its maximum. Once the second generation has been produced (Fig. IV.19), the next pose (Fig. IV.20) is derived.

6 Tuning

Usually, once the user is more or less satisfied with the current solution, a bit of fine tuning still has to be performed. For example, a set of limbs may not be bent enough, etc. It is also very nice to experiment with poses similar to the current one. One of the advantages of this interface is that often users were looking for a given pose but ended up producing another one, not because they could not produce the one they had originally in mind, but rather because they managed to create a pose which looks better than the one they were looking for. This purely subjective choice would not so easily be expressible using other more conventional posing systems.

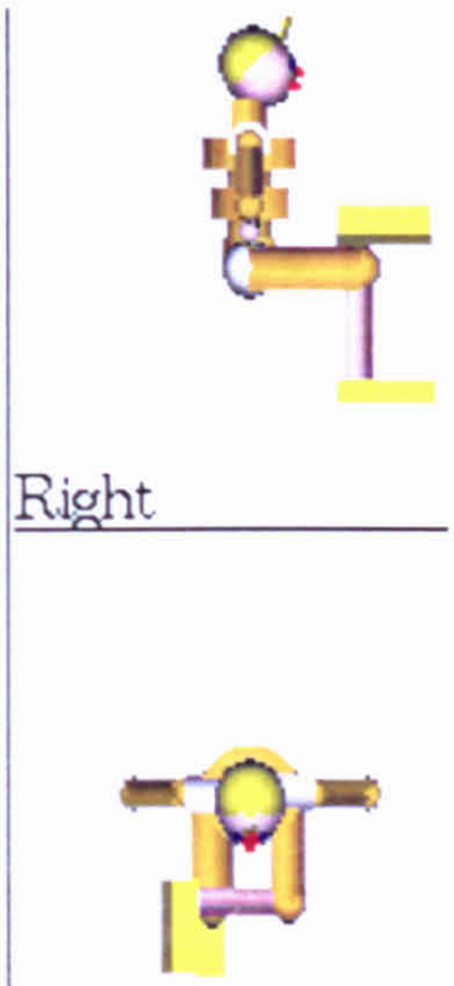
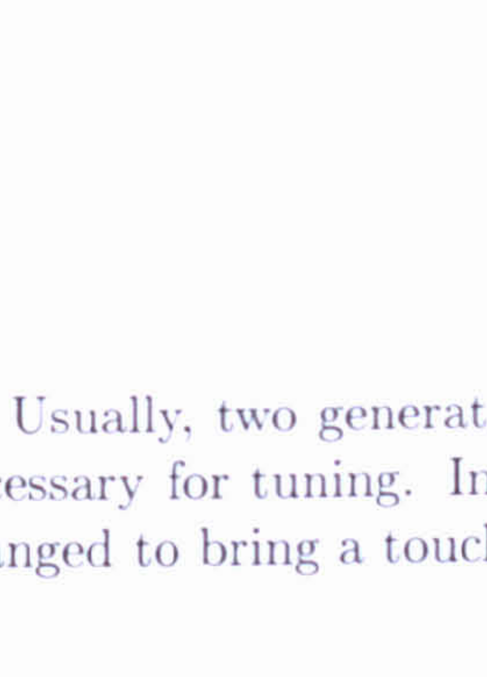


Figure IV.20: Second pose

Using the second generation, hands were brought towards the hips and the left foot now lies on top of the right leg.

Usually, two generations are required to produce a pose [4]. A third one might be necessary for tuning. In our example, the position of every single limbs was slightly changed to bring a touch of "naturalness" (Fig. IV.21).



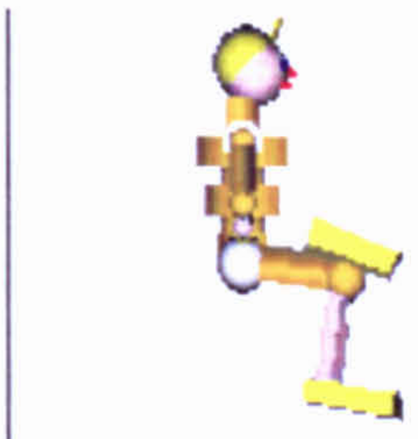


Figure IV.21: Third pose

To improve the realism, another generation was produced from which this pose was obtained.

In many animations, characters which stand completely still do not look natural. It is much better if they would move a bit. The pose which has been produced could be used in a sequence featuring a meeting between people. It might move a bit, use its hand, turn its head, move its feet. Using traditional techniques, this is still a lot of work. Using this model, a single generation (Fig. IV.22) can produce as many different poses as there are on the window, all usable for this type of animated sequence.

[4] If no twist is required, then one generation is generally enough.

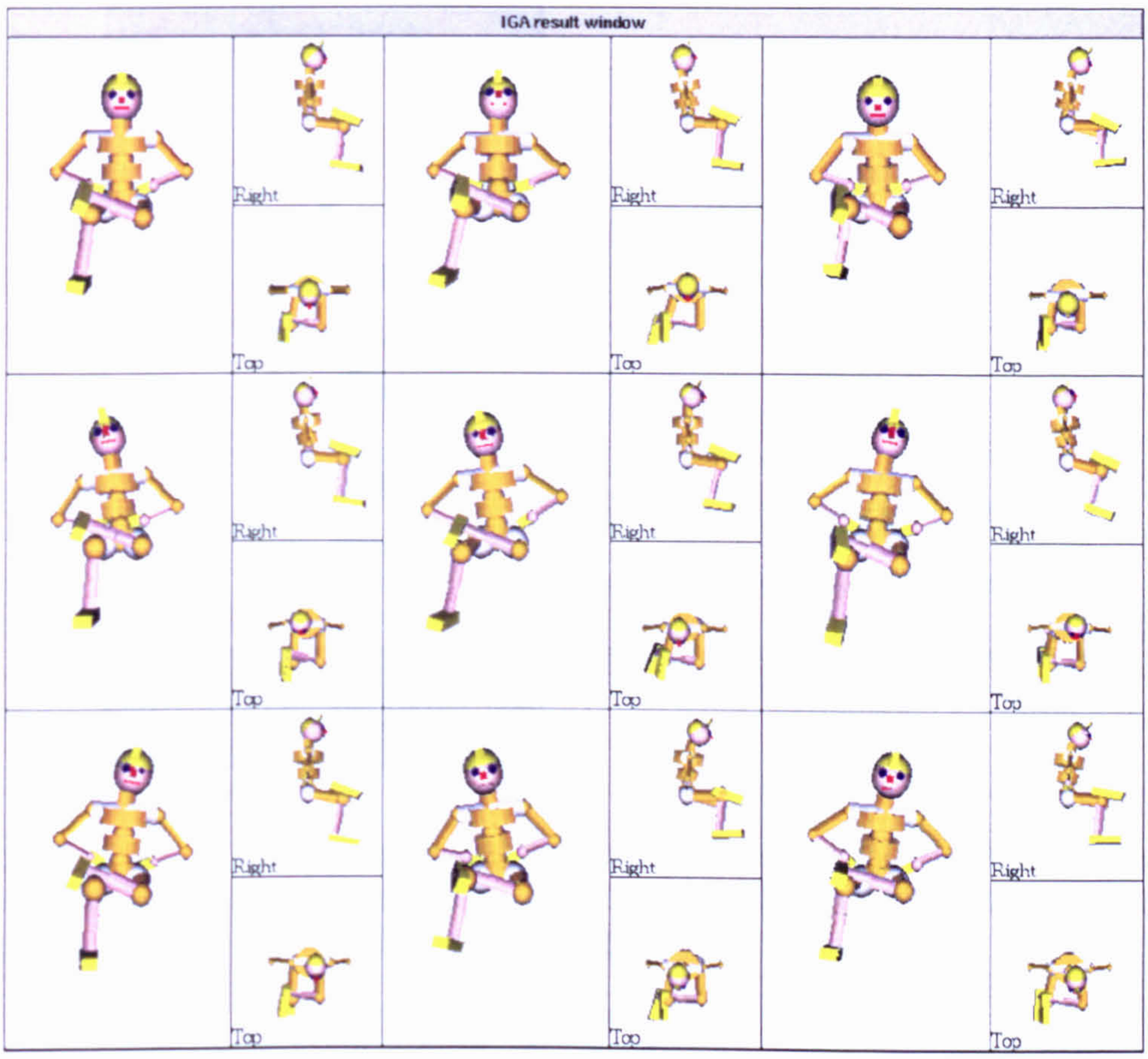


Figure IV.22: Fourth generation

Producing several poses slightly different from each other is useful for many animated sequences to bring the character to life.

6.1 Selecting part of the articulated figure

When mutating the seed pose to produce a new population, the computer will alter all limbs. This may be a hindrance. For example, if the torso is already correctly placed, moving the torso again will harden the selection of a correct arm position.

To overcome these problems, users can enable or disable limbs allowed to move using the same window already used to display the skeleton and its cones (Fig. IV.23). When the computer is asked to provide a new set of positions, only the enabled body parts will be altered. Selected body parts can also be completely reinitialised.

6.2 Virtual ball

When the articulated figures are rendered to their full size, a special camera is used. This camera can be rotated to view the scenes from different angles. An object called a *virtual ball* will move the virtual camera on the surface of a sphere with a

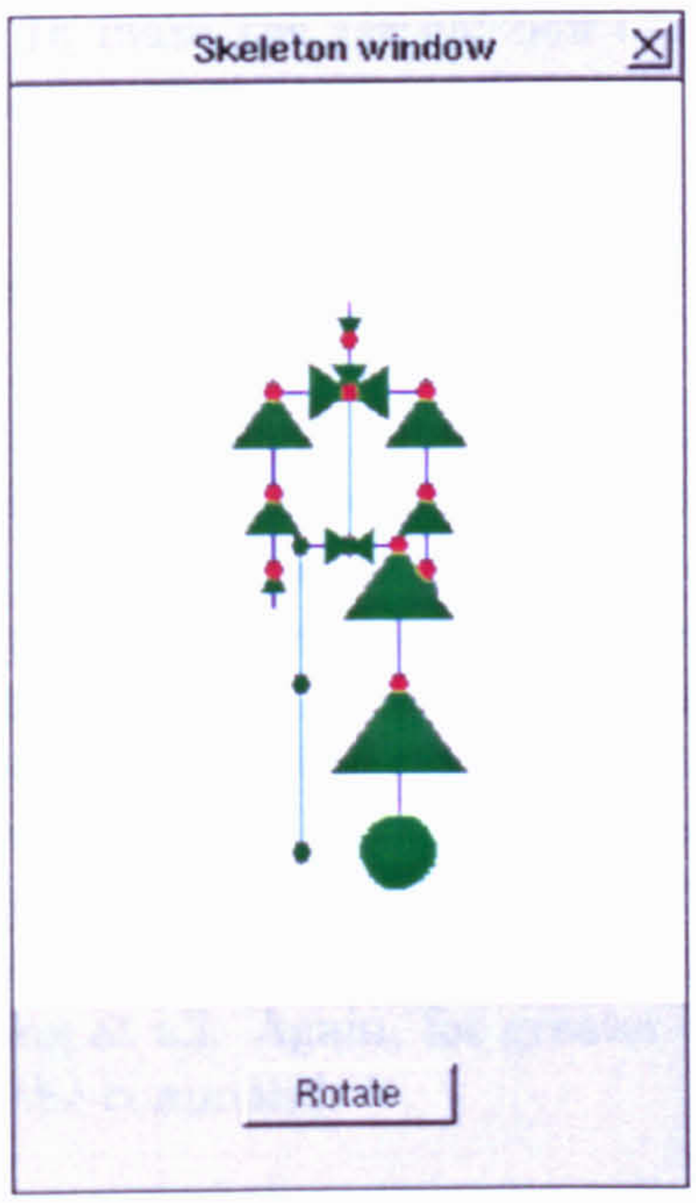


Figure IV.23: Skeleton window

To forbid some limbs to move, they have to unselected. Here, the spine and the right leg have been disabled. Cones show where the next alternatives will lie.

specified radius. It makes sure that the direction of view of the camera always points towards the centre of the scene, where the articulated figure lies. To bring the camera to a given position a quaternion is used (Fig. IV.24).

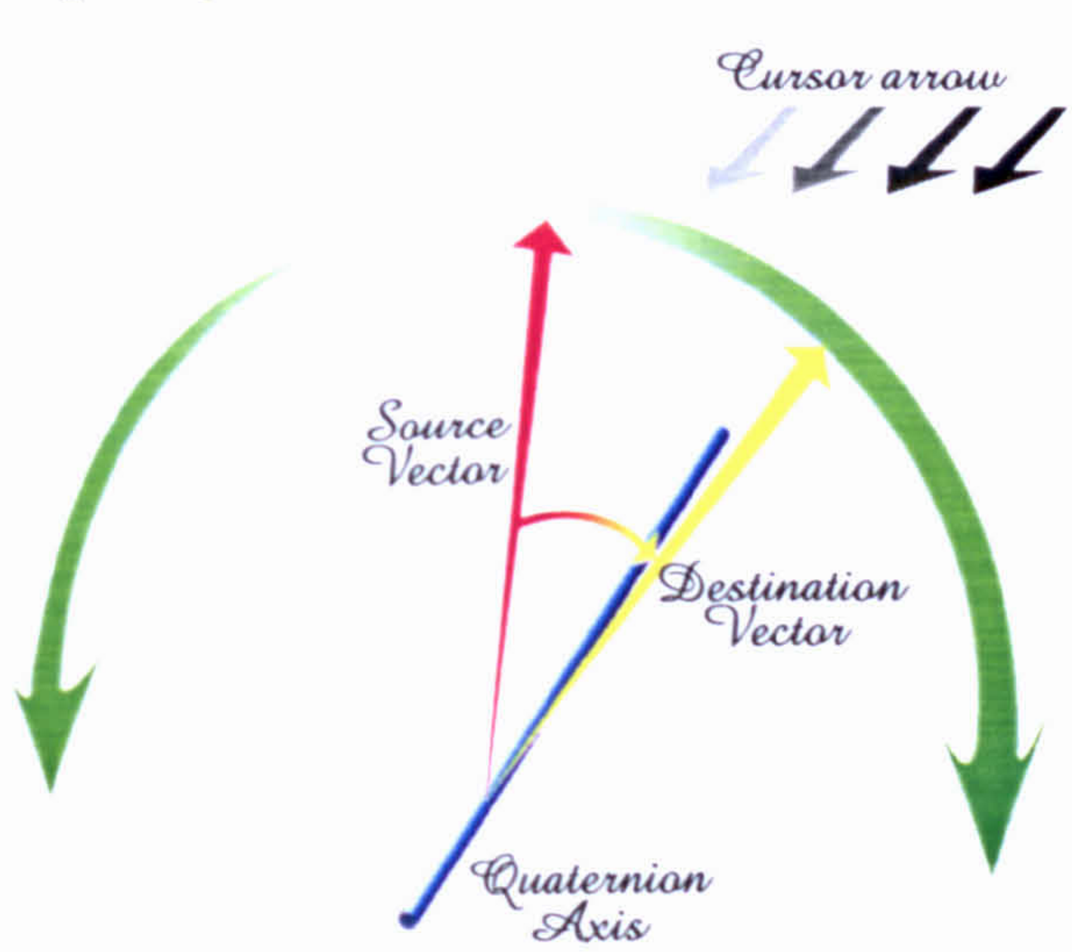


Figure IV.24: Virtual ball

Pressing first the mouse button down specifies the position of the source vector. Releasing it specifies the destination vector. With these two vectors, a quaternion is obtained which defines the new position of the camera and its orientation so that it appears that the object has been rotated.

This tool can be rendered on a canvas on its own. For more convenience, it can also use a canvas in which a scene is rendered. When this is the case, it is usually hidden. Because sometimes mouse buttons are already used, it is possible to specify a flag to this tool saying that it will obey mouse events when a key such as *shift* or *control* is pressed. In this application, the mouse was already used to select body parts. So, the virtual ball was made to obey mouse events only when the *control* key was pressed. So, pressing *control* and the left mouse button when the mouse is moved will rotate the scene. This tool was also implemented so that the outcome was intuitive to the user. Thus dragging the mouse to the right will rotate the articulated figure to the

right, etc. To make the virtual ball even easier to use, double clicking was used to reset the camera in the default view.

6.3 Undo operations

When trying to select a limb or a group of limbs, it is common to make mistakes. The computer tries to work out what the user wants to select but it may get it wrong. So, clicking on a limb might select another one. Before copying the selection into the seed pose, a confirmation could be asked for but this would drastically slow down the selection process. This would also be unnecessary most of the time. The obvious solution is to be able to backtrack. For this purpose, a simple *undo* command has been implemented. Before generating a new pose, the current one is saved. So, to undo an operation, the saved pose is copied into the seed pose. In fact, the containment of the seed pose and of the saved pose is exchanged. Thus, undoing twice is the equivalent of doing nothing at all. Again, for greater interactivity, a hot key (*Control-Z*) was used to activate the command.

Chapter V

Conventional techniques

1 Introduction

The design of a new technique is only the first part of a research work. In most areas, techniques will already exist. The next step is to evaluate how good this technique is, compared to existing ones.

To pose an articulated figure, two main techniques have already been developed. These are forward and inverse kinematics. There are some others and variations based on these two but to evaluate the generator against all possible techniques would have been infeasible.

An inverse kinematic system was implemented since an animation program using such a technique was not available at the time this research was taking place. However, LifeForm [Mac], a software which uses key-frames to animate articulated figures, was available. It also comes up with a positioning system which uses forward kinematics. However, a program using forward kinematics was implemented too, since using an entirely different interface from the other two techniques might bias the results of the comparative study.

From this, it is clear that the evaluation will be performed using a particular implementation of two known techniques. Some implementations are better than others, the ones presented here may not be perfect [1]. So, whatever the outcome of the evaluation is, it will just say that one particular implementation of a given technique is better than the implementation of another technique. This tends to prove that the first technique is better than the second one.

[1] And in fact, this study allowed me to realise that they were not perfect.

2 Forward kinematics

2.1 Review

Forward kinematics is a simple but powerful technique. An articulated figure is made from a set of segments called limbs connected together by means of joints. It is usually represented by a tree, where each node is a joint. At each joint, there is a transformation matrix. This matrix is used to rotate the associated limb and consequently all limbs down the tree connected to that one. It is also used to project the limb into the body coordinates. Quaternions were not used because either they would eventually have had to be converted into a set of matrices to take into account previous rotations and translations, or a set of quaternions and translations vectors would have had to be used and this would have quickly become more time consuming than simply using matrices. No other transformation such as scaling is allowed. There are three types of rotations:

- **Flexion:** This is a rotation of the limb which is influenced by the joint and causes the motion of all limbs linked to this joint. This flexion is carried out relative to the joint point and a flexion axis which has to be defined.

- **Pivot:** The pivot makes the bending axis rotate around the limb which is influenced by the joint. The pivot axis is the axis perpendicular to the flexion axis and the axis of the limb.

- **Twisting:** Twisting causes a torsion of the limb which is influenced by the joint. The direction of the twisting axis is found similarly to the direction of the pivot.

To position an articulated figure, the simplest solution is to specify the transformation matrix at each joint. Forward kinematics is about just that. Obviously, there is little chance that an animator, who is usually not a mathematician, will be able to specify directly a set of matrices for each joint. In a positioning system which uses forward kinematics, the interface translates animator instructions into a set of matrices which specify the new position of the articulated figure. The result is then rendered. Because calculations are simple, all this is easily performed in real time.

The interface is what will differentiate between a good positioning system and a bad one. To animate legged animals, Michael Girard implemented a key-framing system called PODA [Gir86, Gir87, GM85]. To produce key-frames, forward and inverse kinematics were used but he did not explain how the interface behaved. Thomas Calvert & al. implemented LifeForms^{tm}, an animation system for the Macintosh and other platforms[SC92]. Literature on this particular topic is scarce. Typical systems use sliders and another type of object which are refered here to as joint balls [Mac].

Experimentations were performed with three types of interface:

1. Three sliders, mapped to rotations around the X, Y and Z axis, were first used. The interface was neither easy nor intuitive.

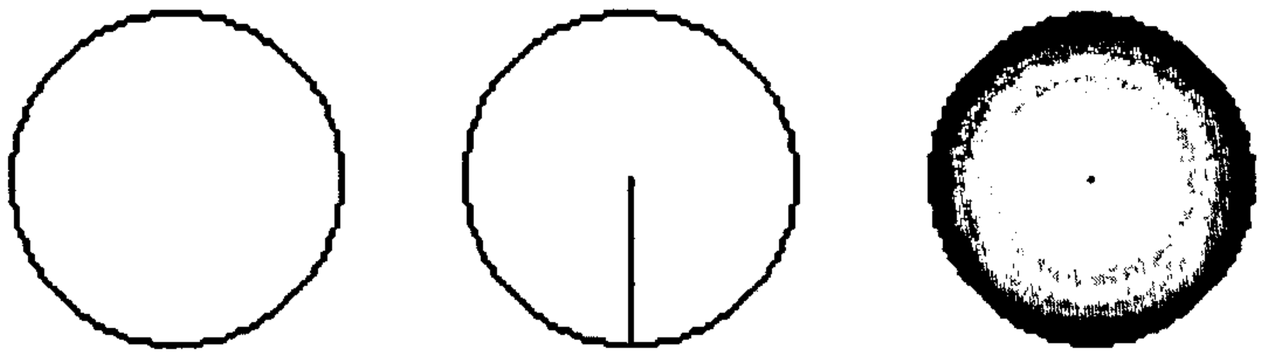


Figure V.1: Joint ball

The joint ball is shown in its three possible states:

1. Nothing is selected or not associated DOFs are enabled
2. Only one DOF is enabled (usually flexion or twist)
3. Both DOFs are enabled (flexion and pivot)

2. The sliders were then mapped to rotations around the *flexion*, *pivot* and *twist* axis instead. The intuitiveness of the interface improved but it was still rather difficult to use

3. Joint balls, a type of widget used by LifeForm, were implemented. A joint ball can deal with two angles at the same time. If only one angle has to be dealt with, then a joint circle is used instead. Interaction is intuitive and fast (Fig. V.1).

2.2 The interface

2.2.1 Selection

The window used to position the articulated figure is shown in Fig. V.2. The same window is used by all techniques, thus avoiding all bias due to a possible different environment.

In the main view, the articulated figure is rendered at its original size. A virtual ball is used so that the figure can be seen from different angles. By default, the camera is placed at the front of the scene. On the right hand side of the window, the same figure is rendered at 40% of its original size from two different angles. The view at the top uses a camera placed on the left hand side of the articulated figure, the one at the bottom uses a camera placed above the articulated figure. The two objects on the left hand side of the window are the joint balls.

Using this interface, only one limb can be moved at a time. In other words, the interface allows to work on only one joint configuration at a time.

Before trying to move one limb in one direction or another, the first thing to do is to select this limb. This is performed by double-clicking on the rendered limb either on the main view or on one of the side views. The same algorithm detailed in the previous chapter which uses a ray thrown from the camera and cutting through the limb to select is used.

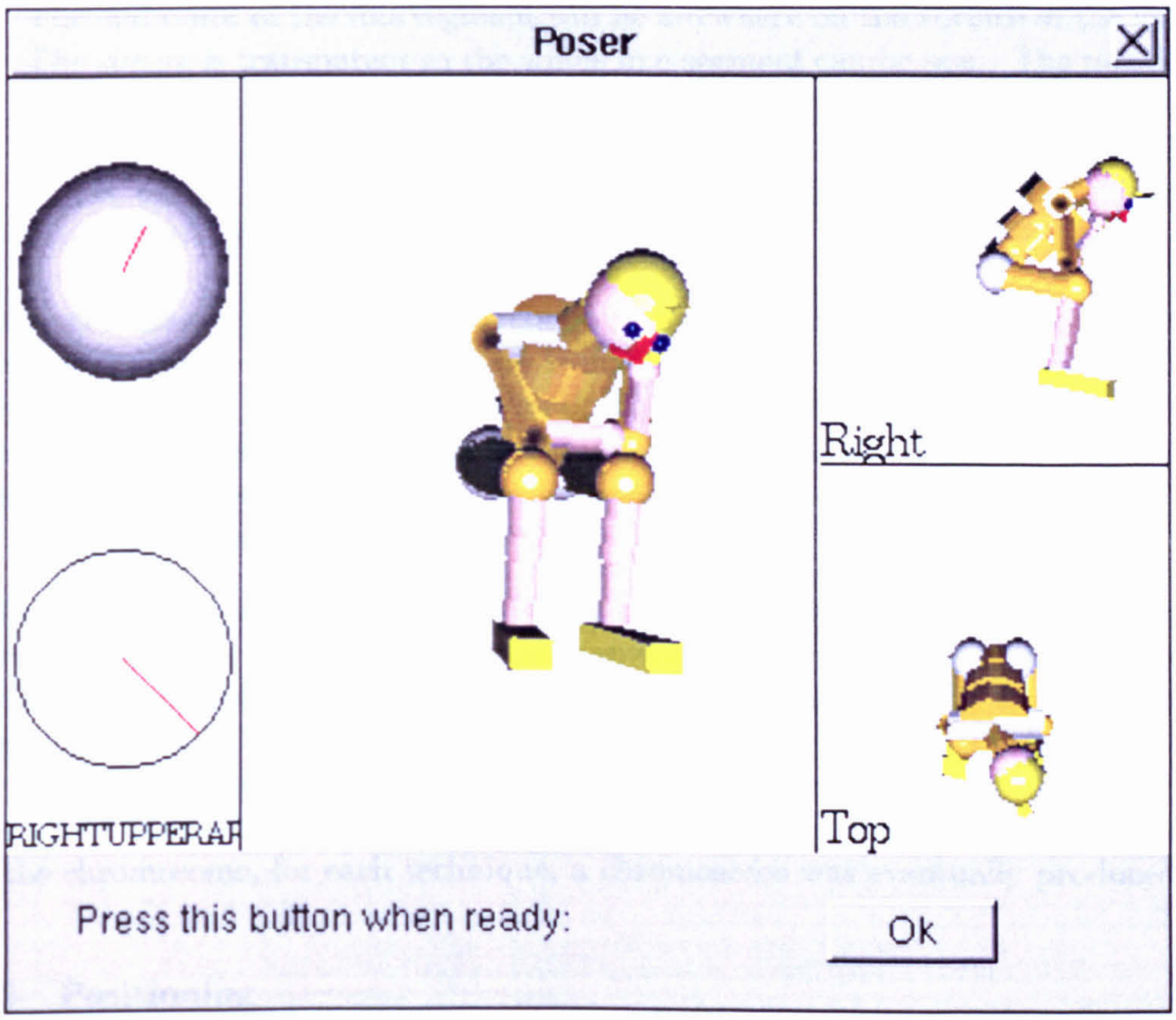


Figure V.2: Poses builder

The same window is used to produce by all three techniques to avoid any bias related to a different interface. The objects on the left are the joint balls. The figure is shown on the right from two different viewpoints.

2.2.1.1 Feedback to the user: The balls on the left hand side are called *joint balls*. The one at the top is used for flexion and pivot, the other one is used for twist. A joint ball operates with two degrees of freedom (DOFs). It has three modes of operations which depends on the DOFs (Fig. V.1):

- The two DOFs are empty (also said to be disabled), so the joint ball is disabled. It is represented by an empty circle.
- Only the first DOF is enabled (flexion DOF normally). The joint ball is represented by a circle and a red line segment inside. It is the size of the radius, and is drawn starting from the center of the circle. The line segment can be rotated around to produce the rotation angles. Angles can range from -180^o to 180^o. Mapping from the minimum to the maximum angle allowed was also tried, but this was found to be harder to use.
- Both DOFs are enabled. The joint ball is represented by a sphere. A line segment, the size of the radius, is drawn, starting from the centre of the sphere.

The end point of the line segment can lie anywhere on the surface of the sphere. The sphere is transparent so the whole line segment can be seen. The position of the end point defines two angles, one for each axis of the DOFs. Again, mapping from the minimum to the maximum angle specified by the DOFs was tried but it just made this tool less intuitive to use. So instead, angles can take values from the range -180^o to 180^o.

The conversion between the current joint configuration to these two widgets is easily performed. A joint configuration is made of a position on the hyper tessellated sphere for flexion & pivot and a position on the hyper tessellated circle for twist. The main axis for the tessellated sphere is the Y axis whereas it is the Z for this widget. So exchanging Y and Z coordinates and mapping onto the radius of the widget is the only thing which needs to be performed.

The use of the tesselated sphere is not a requirement of this implementation of forward kinematics. Neither it is for the implementation of inverse kinematics which will be detailed later on. During this work, an application was built which contained all three techniques. It was felt that it would be a nice feature to be able to switch from one technique to the other. Consequently, poses generated using one technique had to be made understandable to the other techniques. Since the common denominator was the chromosome, for each technique, a chromosome was eventually produced.

2.2.2 Positioning

Once the selection has been performed, the user just has to drag the line segments of the joint balls around to position the selected limb in the desired position. Each time the mouse is dragged, line segments representing rotations angle have to be updated. Using the circle, the position of the line segment is easily computed. It is the vector made of the mouse position relative to the center of the circle normalised to its radius. Using the sphere, the mouse produces X and Y coordinates. Using these, the Z coordinate is easily obtained:

$$Z = \sqrt{R^2 - X^2 - Y^2} \tag{V.1}$$

where R is the radius of the sphere.

At the same time, the articulated figure needs to be rendered again so that the user sees the result. For this purpose, the line segment is projected back from the space of the joint ball onto the hyper tessellated sphere space or hyper tessellated circle space. At this point, the position obtained corresponds just to a 3D point but not to a valid alternative. So because, a valid alternative is needed, it has to be retrieved. The positions on the tessellated sphere could be searched through sequentially but this would be too slow. Instead, the neighbours attribute of the alternatives structure is used to speed up the search. First the closest point at the highest or crudest level is selected. Then its neighbours at the next level of detail are searched through to

retrieve the closest to the position being searched. The closest one is then selected and the process is repeated one level of detail finer until the finest level of detail has been searched. In pseudo-code, this gives something like thisThe pseudo-code is:

```
/*************************************************************************
* GetAlternative:  Return the closest point on the hyper tesselated      **
*                  sphere or circle to Point                             **
*                                                                        **
* INPUT :  Point - Retrieve the closest alternative to                   **
* OUTPUT:func - The closest alternative                                  **
*************************************************************************/

CAlternative& GetAlternative(const iCPoint3D& Point) const
{
 unsigned int i, CurrentLevel;
 CAlternative *Best, *Current;
 long       Dist, BestDist;

 /*
  * Select the closest point at the crudest level of detail
  */
 Best = Alternatives[0];
 BestDist = Distance(Point, Best->GetPoint());
 for (i = 1; i < NB_CRUDEST_ALTERNATIVES; i++) {
     Dist = Distance(Point, Alternatives[i]->GetPoint());
     if (Dist < BestDist) {
      BestDist = Dist;
      Best = Alternatives[i];
     }
 }

 /*
  * Search through all level of details
  */
 CurrentLevel = Best->GetLevel() - 1;
 while (CurrentLevel != 0) {
     Current = Best;
     for (i = 0; i < Current->GetNeighbours()[CurrentLevel].GetNbElems(); i++) {
      Dist = Distance(Point, Current->GetNeighbours()[CurrentLevel][i]->GetPoint());
      if (Dist < BestDist) {
          BestDist = Dist;
          Best = Current->GetNeighbours()[CurrentLevel][i];
      }
     }
     CurrentLevel--;
 }
 return *Best;
}
```

Figure V.3: Alternative from 3D vector

> This sample code is used to retrieve the closest alternative on the tessellated hypersphere to a direction vector. The process starts by retrieving the closest alternative at the coarsest level of detail. It then proceeds by searching its neighbours at one level of detail finer, recursively, until the finest level of detail was searched through.

Even with forward kinematics, the chromosome structure is still being used. This is mainly to preserve implementation simplicity. The alternative retrieved is used to modify the gene encoding the selected limb. The chromosome is reinterpreted and the result displayed. Because most parts of the articulated figure have not moved, this part is specially optimised to avoid rendering the entire figure but just the parts which have moved.

3 Inverse kinematics

Sometimes called goal-directed positioning, inverse kinematics has become a popular technique to pose articulated figures in the past few years. Instead of having

to move every limb one by one, one has only to move the tip of a given limb called end effector, in a particular direction and the computer will move as many limbs as necessary for the end-effector to actually follow the path defined by the user.

3.1 Review

3.1.1 Inverse dynamics

Although inverse dynamics are not inverse kinematics, inverse dynamics have also been used to position articulated figures. Furthermore, the technique bears quite a few similarities with inverse kinematics, hence this is why it is discussed here.

In 1988, Forsey and Wilhelms [FW88] used inverse dynamics to pose an articulated figure, a humanoid in their example. Although inverse dynamics is a lot more computationally expensive than inverse kinematics, they started from the assumption that because inverse dynamics are based on physics, posing figures would be more intuitive to the user.

The mass of each limb of the articulated figure was automatically computed from the volume of each limb (limbs were represented using cubes). This avoided the effort to the user of having to specify each of them. A fourth order Runge-Kutta scheme was used for the dynamic computation. Goals had to be placed before the computation could take place and not before. This limited the interactivity of the system. Without entering in the details, the main drawback of the system, as one would have realised, is the time it takes to do the computations. On modern computers, the technique would still be too slow to allow for interactive work. By truly interactive, it is meant that if for example, the hand is grasped and moved around, the hand must move around in real time, i.e. something like 12 frames a second. In that system, the new position of the hand had to be specified before the computation could take place.

3.1.2 Analytical solutions

When trying to position a 3D chain, a set of equations can be obtained [KB82]. Solving these equations will produce the orientation of the different segments to achieve the goal specified.

If the goal is perfectly constrained, i.e. if the number of DOFs is equal to the number of constraints imposed by the goal parameters, then an analytic solution can sometimes be found [KB82]. The goal parameters are the position of the end effector or the area where it should lie, but also the orientation of the end effector, the limits at joints, etc.

Algebraic solutions have two main advantages over numerical techniques. First, they are performed more quickly and, second, they will find all possible solutions.

Unfortunately, they have been used with simple systems only and there is no guarantee that a solution will be found.

When the system is under-constrained (it is said to be redundant), that is if there are more DOFs than constraints, it is sometimes possible to find solutions analytically but it is usually better to fully constrain the system first and then to try to find an analytic solution.

Analytical techniques may produce more than just one solution and they do not ensure these are valid, since DOFs are ignored during the calculations.

3.1.3 Numerical solutions

With perfectly constrained systems, numerical solutions typically involve differentiating the constraints equations to obtain a Jacobian matrix J. From J, changes of the joint positions result in changes in the orientation of the limbs. To perform inverse kinematics, the Jacobian has to be inverted. Knowing the position of the end-effector, the Jacobian will produce the necessary changes in the joint positions. Inverting the Jacobian is computationally expensive. The complexity of the matrix greatly increases with the complexity of the articulated figure. Numerical solutions relies on iterative techniques to find a solution. At each iteration, all the entries of the matrix need to be evaluated.

Numerical solutions always converge towards one solution. Nothing is there to ensure the solution will not break joint limits. During the search, special care must be taken to ensure the solution is valid. This further adds to the computational cost of the technique. Since iterative techniques are used, the solution greatly depends of the initial estimate of the solution.

With redundant systems, a more general approach in which an objective function which has to be minimised is generally used. Possible objectives are the minimization of time, energy or displacement. The method of Lagrange multipliers is used [Whi72, uLM96]. With a system of no joint limits, it leads to a perfectly constrained system which can be easily optimised. The method will find all minima for the objective.

To solve for articulated systems with joint limits, minima that do not satisfy joint limits can be simply discarded. Another method is to eliminate the inequalities by adding a new variable and an equality constraint. If there are n DOFs, there will $2n$ additional variables to solve. Another way is to increase the value of the objective function when joints reach their limits. These are referred to as penalty functions. Another technique is to ignore inequalities until a joint limit is exceeded in which case the joint is brought back to a valid configuration. A discussion about these techniques can be found in [WC78].

In all cases, the minimisation of the objective function is a computationally expensive task.

3.1.4 The multiple constraints solution

In [BMW87, BMB86], Badler & al. evaluated the potential of a six DOFs input device to position an articulated figure amongst other things. The interface of their system was similar to the one presented by Forsey [FW88]. An end effector was selected and a position to reach was specified. An original and simple algorithm was used to solve the inverse kinematics problem. With a chain of n limbs where 0 is the root and $n-1$ is the end effector, the algorithm looks like that:

1. j = n − 1
2. orient segment j toward the goal
3. new goal = (old goal position) − (length of the segment j)
4. j = j −1
5. orient segement j toward the new goal
6. **if** (segment j was oriented toward the goal) OR
   (orientation is beyond joint limits) then **goto** step 4.
7. **if** (j is equal to 0) OR (distal joint of segment i has reached its goal) stop
8. **goto** step 1.

Figure V.4: Simple inverse kinematics algorithm

This algorithm relies on a simple recursive orientation scheme of the different limbs of the kinematic chain. Although Badler & al. describe this algorithm as being a simple and not clever implementation of inverse kinematics, it seemed that it achieved what was asked from it. May be an idea to investigate further.

We clearly see that, as the complexity of the articulated figure increases, the number of iterations might become huge. However, although they wrote "it was not an efficient or clever solution to the problem of inverse kinematics", it seems that it did the job. Maybe it is an alternative to this common problem worth investigating further.

3.1.5 The workspace solution

In [Kor82, KB82], James Korein developed a positioning system which involved the computation of pre-computed workspaces to build a perfectly constrained kinematic system. The latter could then be solved analytically. A workspace is defined as the volume which can be reached by a segment and all segments connected to that one until the end-effector. For example, a chain made of three segments q_1, q_2 and q_3, their workspace is W_1, W_2 and W_3 respectively (Fig. V.5). To reach a point p, this point needs to be in workspace W_1. If not, it is unreachable. If it is, q_1 needs to be rotated sufficiently so that p lies inside W_2. The process is repeated for the next limbs. The pseudo-code is:

Each adjustment requires finding the intersection between a workspace surface and a line or circle. The final links of the chain which comprise a perfectly constrained system can be solved analytically. The method requires pre-computation and storage of workspaces. Workspaces of high dimension requires a lot of memory and make the in-

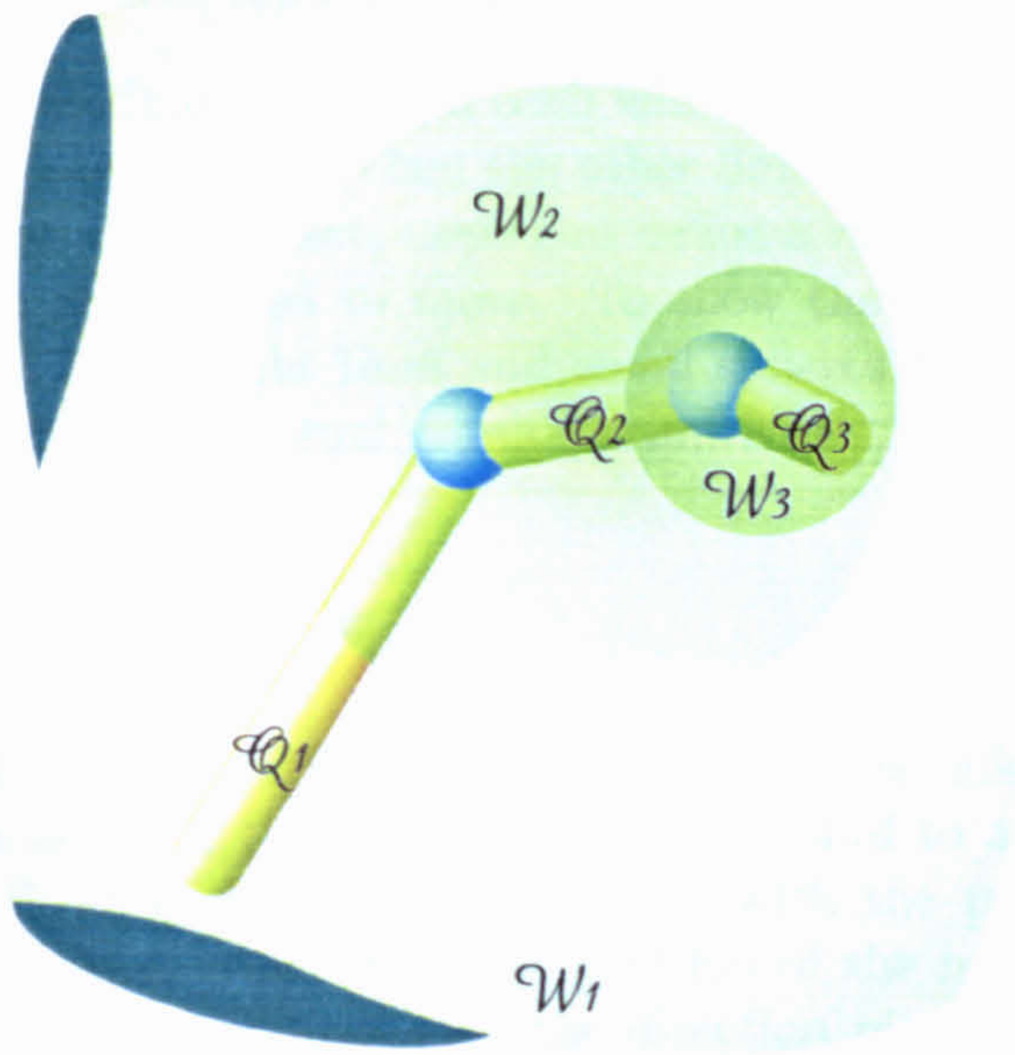


Figure V.5: Workspaces

A workspace defines the volume in which a limb and its children can move. Here W_1 represents the volume that Q_1, Q_2 and Q_3 can access. It includes W_2 and W_3.

if goal p is not in W_1
 then it is not reachable; give up.
Otherwise:
 for $i = 1$ to number of joints in the chain
 adjust q_i only as much as is necessary so that
 the next workspace W_{i+1} includes the goal p.

Figure V.6: Inverse kinematics using workspaces

Workspaces allow to determine if a goal is inside the reach of a kinematic chain. Adjusting joint configuration just enough so that the goal is still in the reach of the next sub-kinematic chain allows to eventually solve the inverse kinematic problem using analytical methods.

tersection between workspace surfaces and lines or circles harder and computationally expensive.

3.2 The interface

3.2.1 Selection

The window, already used for forward kinematics and displaying the seed pose, is used again for inverse kinematics (Fig. V.2). Again, this is to avoid all bias from using a different interface.

To select a limb, the user simply needs to double click on it. The same algorithm used for forward kinematics is used to find out which limb to select. To show that a limb has been selected, volumes of the limbs are rendered in reverse mode, that is

dark becomes light and vice-versa.

Sometimes, moving a limb will also move other limbs to allow the end effector to follow the mouse. When the other limbs are already positioned, this is annoying. To avoid this side effect, users can define a rectangular region around the group of limbs which are allowed to move. To show that the other limbs will not move, they are shown by straight lines and small spheres for limb segments and joints respectively instead of their usual graphic primitives.

3.2.2 Positioning

The end position of the selected limb is called the end effector. Once a limb has been selected, the user can drag it around to achieve a given pose. When dragging a limb with a mouse, we are left with the problem a finding the position of the next goal to reach from the position of the mouse. This is solved by using a plane which is perpendicular to the direction of view and containing the position of the end effector. The new position must also be somewhere on this plane. Knowing the position of the mouse and the direction of view, a ray, whith coordinates determined by equations IV.4, is thrown which will intersect this plane at the new position of the end effector (Eq. IV.12). Because the end effector usually does not completely reach the goal, either due to constraints making the goal unreachable or because of rounding errors in the computations, the plane may move as the end effector is being dragged around.

It is usually easier to place a limb using one view rather than using another one. For example, starting from a standing leg and trying to bring it in the seated position, it is easier to work when the articulated figure is seen from one of the sides rather than from the front. It is the role of the virtual ball to rotate the virtual camera. The pilot study before the evaluation helped to implement a powerful tool. Originally, when the virtual camera was being rotated, the articulated figure was displayed again only when the mouse button was released. Although it was felt that this was not a problem, most people could not predict the view they would end up with. Surely enough, experience would overcome this problem but it was not possible to train participants of the comparative study for long enough. As a result, I decided to redraw the figure as the virtual camera was rotated around. People had then no problem using the virtual ball. However, it requires some processing power that not all machines are capable of yet.

3.3 Implementation

3.3.1 Principle

Solving the problem of inverse kinematics usually involves finding the rotations at the different joints which will bring the end effector to the goal. As a result, the usual

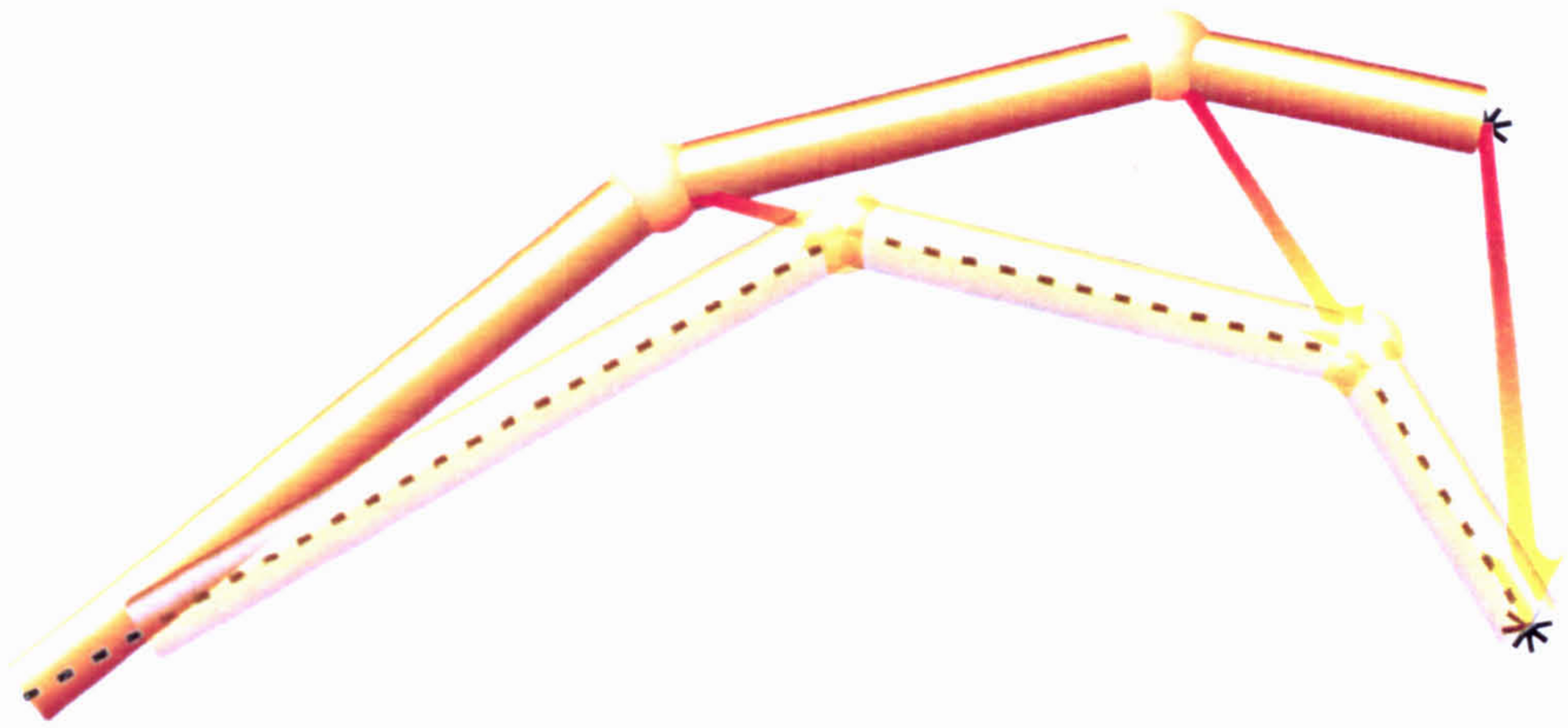


Figure V.7: Inverse kinematics using translations

The end effector is brought to the goal. The position of the beginning of the limb is recomputed. The parent limb is then called as the recomputed beginning of the limb a sub-goal.

way to think about this problem is in terms of rotations to apply at each joint to fulfill the goal.

Another approach to bring the end effector to the goal is to simply perform a translation. Translating the end effector enforces a rotation plus a translation of the parent joint. This translation enforces in turn another rotation and translation of its parent joint (Fig. V.7). Translations quickly become so small that they can be discarded. At the end, we are left with only rotations, the rotations of the joint needed to bring the end effector to the goal (Fig. V.8).

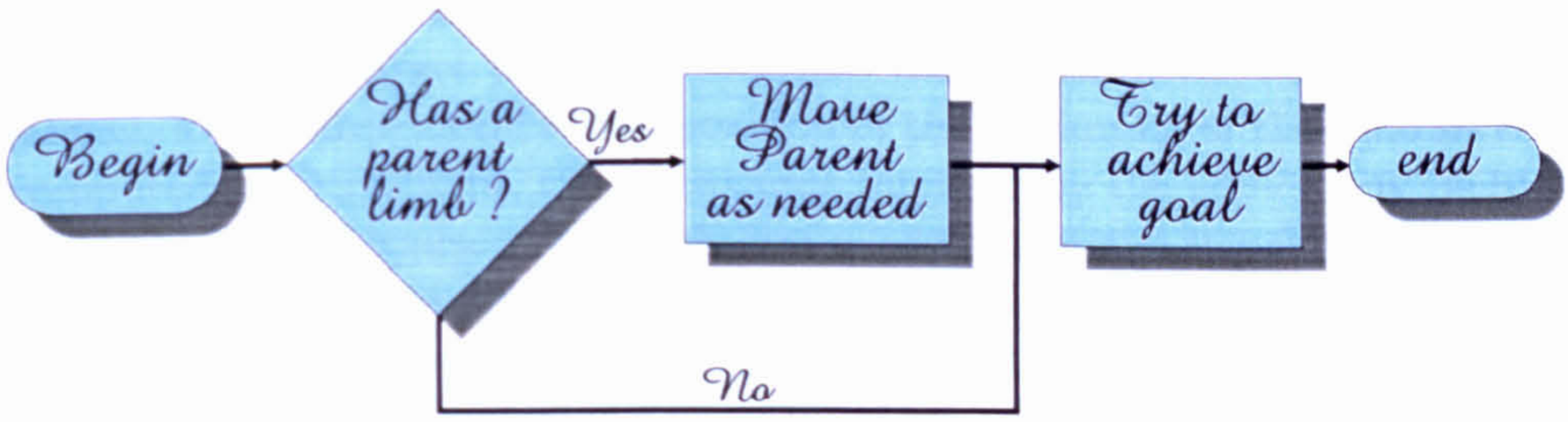


Figure V.8: New inverse kinematics algorithm

This algorithm is simple. Basically, if the current limb is too far to reach the goal and it has a parent, it asks the parent to achieve a new goal which is computed on the ideal position for this limb. Once the parent has done what it could, the current limb does what it can.

3.3.2 Degrees of freedom

3.3.2.1 Problem: Computations are straightforward. No inversion of a Jacobian is needed. No optimisation method is required. Consequently this method is much faster than conventional methods.

Unfortunately, this does not take into account DOFs. Experimentations with that particular implementation showed it was far from being practical. It was obvious that constraints imposed by DOFs had to be brought in.

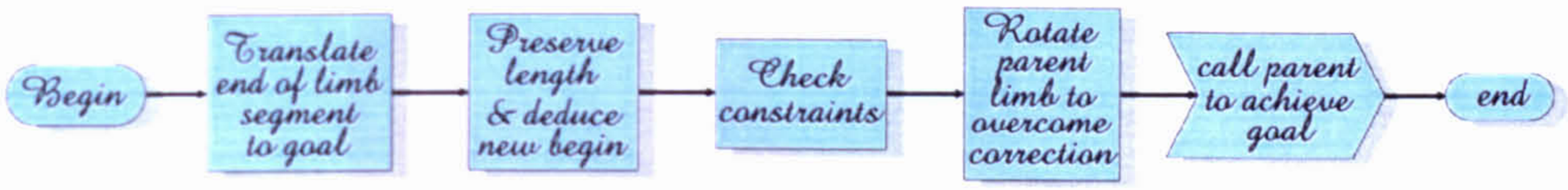


Figure V.9: Detailed first pass

During the first pass, the limb goes to the goal and makes any correction necessary. It then calls the parent limb so that the joint between the parent and the current limb lies where the current limbs would like it to be.

3.3.2.2 First pass (Fig. V.9): For this purpose, the technique works in two passes. In the first pass, the algorithm computes the ideal position for the beginning of the limb, the one which would not break the constraints. If this ideal position is different from the existing position, the parent limb (if any) is called to solve the new goal, that is to bring the end of its limb to the beginning of the current one. In the second pass, the parent joint has done the best it could to get closer to the goal it was given, so the current joint also does the best it can to get closer to the goal to achieve without breaking the constraints.

The first pass is only called if there is a parent limb. If there is no parent limb, that is if the current limb is the root node of the skeleton's tree (the hip with the humanoid), one solution is to translate the limb and so the entire figure. Trying it out, this was found to be annoying, so instead, nothing was done.

In the first pass, the end of the current limb (the end effector) is translated to the goal to achieve (the target position). Because the length of the limbs has to be kept unchanged, it probably modifies the position of the beginning of the limb so the latter is recomputed. At this point, the constraints imposed by the DOFs may be broken so these are checked. Because constraints are expressed in the hyper tessellated sphere space, limb positions are projected into the tessellated sphere space. Corresponding angles are computed and then checked against the DOFs. If an angle is too big or too small, it is brought back to the closest valid value. Angles are then used to compute the corresponding point in the hyper tessellated sphere space. It is then projected into the limb space. If constraints were broken, the end of the limb do not reach the goal anymore. So to overcome the problem, a rotation of the parent joint, which would bring the end of the current limb as close as possible to the goal, is performed.

This rotation is then applied to find what would be the ideal position of the beginning of the current limb. As seen in Figure V.10, a simple translation would not have worked correctly. Because this rotation only involves the parent joint, it has not altered the configuration of the current joint, so it is still valid. At the end, the parent limb is called to try to bring the beginning of the current limb in the required position.

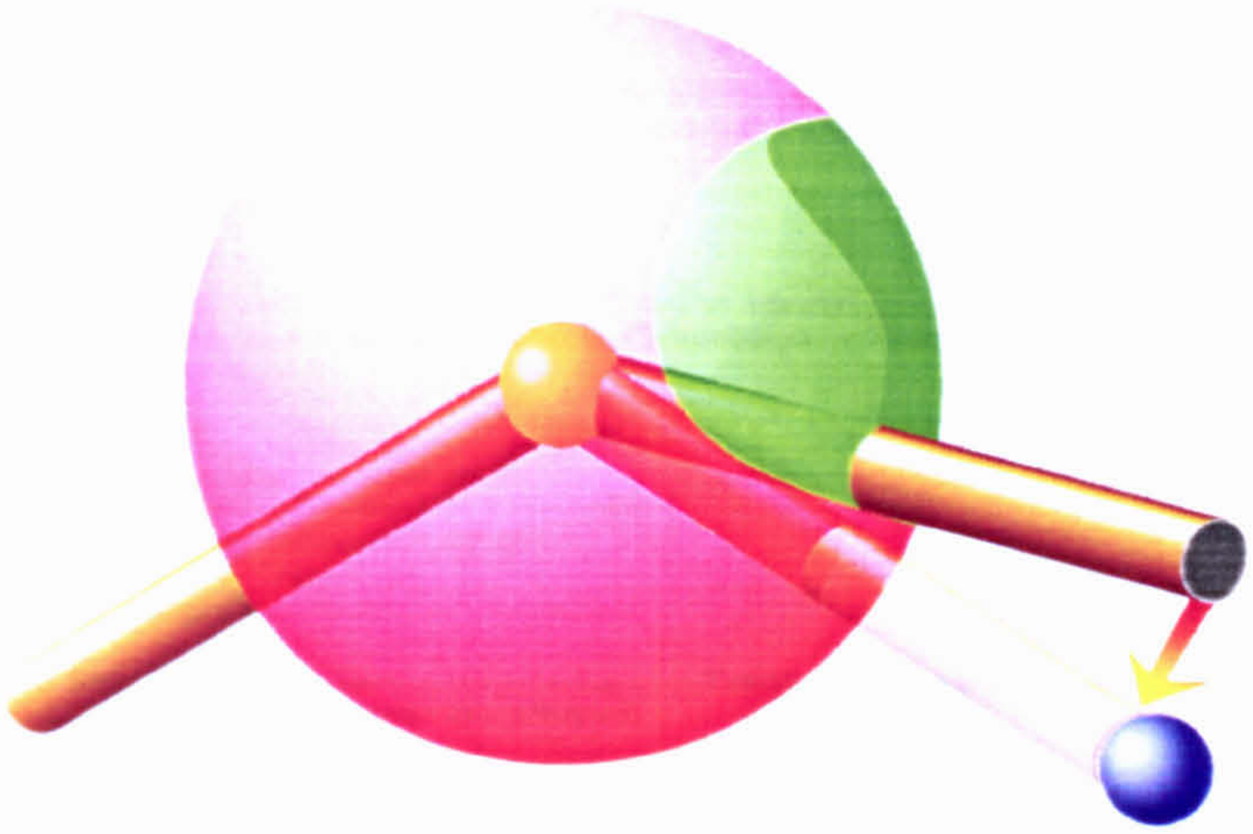


Figure V.10: Inverse kinematics and DOFs

When a limb is already at the edge of the allowed area, rotating at the joint configuration to reach the goal will certainly break the constraints.

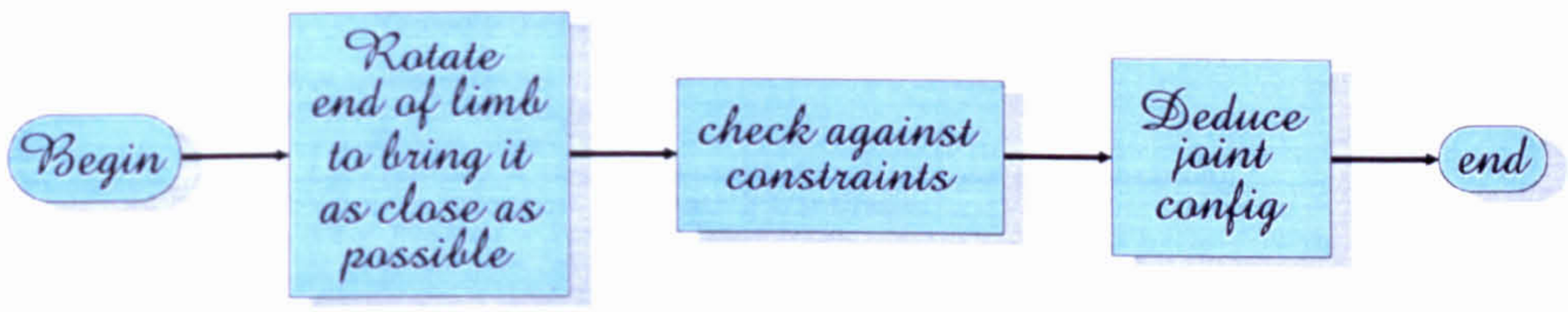


Figure V.11: Second pass

Once the parent limb has done what it could, the current limb has to try its best to achieve the goal. The end of the limb is rotated towards the goal and constrained are checked. The transformation matrix is deduced from that configuration

3.3.2.3 Second pass (Fig. V.11): After the parent limb has achieved the best it could, it may happen that it could not reach the required position. Rotating the end of the limb to bring it as close as possible to the goal might still break constraints. So, the segment made of the new start of the limb and the goal is projected into the tessellated sphere space. Angles are computed, corrected and the new vector produced. It is projected back into the limb space. The joint configuration, needed to bring the limb segment from the resting position to the current position, is then deduced.

3.3.3 Mapping on the hyper tessellated sphere

When positions are mapped onto the sphere space, there are not checked if they do exist amongst the set of pre-computed positions (unlike with forward kinematics). There is no point in doing this. Firstly, it might be too slow, secondly since there are only a finite number of points, it might look a bit jerky when the limb is dragged

```
void Reach(iCPoint3D& Position, int Threshold, CLimb* Caller)
{
    if (End == Begin) { // Zero size limb ? Let's call the parent
        if (Parent) Parent->Reach(Position, Threshold, Caller);
        EXIT_REACH;
    }

    // Joint is fixed ? Let's call the parent
    if (!isSelected || (FlexionDof.Empty() && PivotDof.Empty())) {
        if (Parent) Parent->Reach(Position + (TransBegin - TransEnd), Threshold, Caller);
        EXIT_REACH;
    }

    // We haven't finished yet have we ?
    if (Distance(TransEnd,Position) > Threshold) {
        // Compute desired position of current limb
        if (Position == TransBegin)
            IdealBegin += Position - TransEnd;
        else
            IdealBegin = Position + (TransBegin - Position) * Distance(Begin,End) /
                Distance(Position, TransBegin);

    // Set the end position of the parent limb
    if (Parent) {
        ParentTransformation = Parent->BringEndTo(IdealBegin);

        // Get the Inverse matrix and determine position in local space
        ParentTransformation = ParentTransformation *
            Parent->GetParentTransformation();
        Limb = Parent->GetChild(Parent->GetChildNo(this));
        if (!Null(Limb.JointPos)) {
            ParentTransformation = ParentTransformation.Translation(Limb.JointPos *
                Distance(IdealBegin,Parent->GetTransBegin()) /
                Normalise(Limb.JointPos)) * ParentTransformation;
        }
        INTO_LIMB_SPACE;
        if (!Null(LocalPosition/3)) {
            VALIDATE(CDof());
            LocalPosition = GetPoint(LocalPosition) * ParentTransformation);
            V1 = LocalPosition - Parent->TransBegin;
            V2 = Position - Parent->TransBegin;
            Q = QuaternionToRotate(V1, V2);
            IdealBegin = Rotate(Q, IdealBegin - Parent->TransBegin) + Parent->TransBegin;
            IdealBegin = (IdealBegin - Position) * Normalise(Begin - End) /
                    Normalise(IdealBegin - Position) + Position;
        }
        Parent->Reach(IdealBegin, TRESHOLD, Caller);
    }

        INTO_LIMB_SPACE
        if (!Null(LocalPosition / 3))
            VALIDATE(PivotDof);
        else
            EXIT_REACH; // This is suspicious, let's do nothing

        if (LocalPosition != Begin) {
            Q = QuaternionToRotate(End - Begin, iCPoint3D(LocalPosition) - Begin);
            if (Distance(GetPoint(End * Q * ParentTransformation), Position) <
                Distance(TransEnd, Position)) {
                Transformation = Q;
                hasMoved = TRUE;
            }
        }
        EXIT_REACH;
    }
}
```

Figure V.12: An innovative algorithm to the inverse kinematics problem

around [2]. Instead when the user releases the mouse button, for each joint segment which has moved, rotations are projected into the hyper tessellated sphere space. Closest positions are selected and configurations are re-computed. So when the user releases the mouse button, there is a slight change between the previous pose and the one newly displayed.

[2]Moving a limb using joint balls is always a bit jerky, although this does not seem to be too inconvenient

Chapter VI

Evaluation

1 Introduction

The design of an innovative technique is normally the first step of any research work. Usually, techniques already exist which try to tackle the same type of problem. The next step is to compare the new technique against existing techniques to work out which one is the best according to a few criteria (speed, ease of use, workload, etc) assuming the innovative technique is better than the previous ones (the hypothesis).

Chapter IV describes the generator, an innovative technique used to position articulated figures. A few techniques to position articulated figures have already been developed from which the main two, forward and inverse kinematics, were implemented (Chapter V). Thus, the hypothesis is:

> The generator, an innovative technique used to position articulated figures, requires less work and can position articulated figures faster than forward and inverse kinematics, two conventional techniques.

Ideally, this type of experiment would be performed using expert users. Unfortunately, finding professional animators who knew forward and inverse kinematics was not possible. It would also have been necessary to train them intensively to bring their knowledge of the generator to a level of expertise high enough to allow for a fair comparison.

So, non expert users were used instead in the hope that the results could be generalised to expert users. However, it quickly became obvious that some techniques required more training than others. Thus, some results would not able to be generalised to expert users. It was also felt that given sufficient training, the generator would become faster, although maybe not easier, than the other two techniques. So, a second experiment was conducted for which I was the sole participant. The new hypothesis to verify was:

> Given sufficient training, the generator will out perform forward and inverse kinematics at positioning articulated figures.

If results were clearly in favor of one technique or another, this would be an indication of the potential, or lack of it, of that technique.

2 First experiment

2.1 Design

2.1.1 Designing the tasks

Although this experiment involves the comparison of three techniques, the interest lies mainly in the comparison between:

1. The generator and forward kinematics
2. The generator and inverse kinematics

Some care has to be taken regarding the participants [JPBHC94]. There could be three sets of participants, one set for each technique. However, some participant will find the tasks easier to execute than others. If it can be ensured that participants of one set are as skilled as participants of the other sets (this is called participant matching), then the problem is solved. This is usually done by using big sets and randomisation. Unfortunately, the time allocated for this research would not have suffice to recruit enough participants and to perform the experiment.

Another solution is to make participants use more than just one technique, that is to have repeated measures within each participant experiment. Participants who will perform badly at one technique can also be expected to perform badly at the others. Performing this type of evaluation brings two problems:

- To be really effective, an evaluation must not be too long. Having each participant using all three techniques would have meant that most single evaluations would have lasted longer than an hour, that is somewhat too long for an evaluation. It is usually recommended than an evaluation last less than an hour [JPBHC94]. However, it was possible for each participant to use two techniques, so this scheme which was used instead. Evaluations lasted approximately fifty minutes.
- Unless the tasks being measured are completely unrelated, an order effect will appear. There are two types of order effects:
  - *The positive order effect*: The next task is somewhat similar to the first one and thus after having performed the first task, the next one appear much

easier. The first task will then be rated negatively compared to the second ones.

- ☆ *The negative order effect*: When performing the next task, the participant get tired and so find the next task more boring and harder than the first one. The first task will then be rated positively compared to the second one.

To cancel these side effects, the solution is to use two sets of participants. Both sets use the same techniques but in reverse order. Thus the order effect is counter-balanced.

Four sets of people were used in all, two to compare the generator against forward kinematics and two to compare the generator against inverse kinematics:

	First Part	Second Part
Group A	Generator	Forward Kinematics
Group B	Forward Kinematics	Generator
Group C	Generator	Inverse Kinematics
Group D	Inverse Kinematics	Generator

2.1.2 Choice of the participants

For this experiment, participants were inexperienced users of animation packages, although they knew how to use a computer. The only requirement was that they knew how to use a keyboard and a mouse.

Originally, it was planned that thirty-two people would take part in the study. Thus, there would be eight participants per group. Thirty-two represents a compromise between too few participants to get accurate results and too many which would take too long. From experts in statistics (personal discussion), I was also told that I would require at least twenty people to get sufficient results.

Since many people were required, it was decided to offer them five pounds as incentive to participate. This funding came from the research project (MIME, funded by EPSRC) I was working on.

The University of Glasgow hosts a conversion masters course in Information Technology which is followed by approximatively one hundred students. Due to their background, they are the perfect participants. Unfortunately, the evaluation study was carried out during the summer period where some had already left and the others were working on their summer project. Although thirty-two of them were targeted, only twelve took part in the study.

The Department of Computing of the University of Glasgow hosts a number of research students. Unfortunately, over four years there, I had the opportunity to present my work to many of them. They could not be allowed to take part in the study as knowing bits of my research may have influenced their performance. So, only

people who did not know my research interests were allowed to take part in the study. Eventually eight more people offered to participate. Due to time limitations, it was not possible to get more participants. Fortunately, there were just enough participants to validate the study.

2.1.3 Choice of the tasks

Participants who performed the evaluation were non-expert people. But ideally, they would have been experienced users. Some of the techniques are more difficult to master than others, and this is particularly true for the generator. Participants had to be brought to a level of expertise which would allow fair comparison. To make matters even worse, participants had to be trained not with a single technique but with two. Ideally, they would be trained as long as was necessary. However, if the training lasted too long, they would become bored even before the experiment started. Usually people can stay concentrated for a bit less that an hour. It is also recognised that such an experiment should not last more than an hour [HC97]. If this is not possible, then the experiment should be divided into several parts with enough time between each to allow participants to rest. An experiment with a single participant might thus be spread over several days.

To have several parts for each individual experiment would have considerably complicated this study. Most people would also not have been willing to take part, and the number of participants is already to the bare minimum. So, instead, a single evaluation had to be performed in less than an hour.

The task was about posing articulated figures. Some poses are more difficult to produce than others. It is important that these poses are easily understood by the participants. The computer can generate many possible poses randomly but most of them do not relate to anything known. Ideally, participants would be trained with as many poses as possible, and the evaluation would be performed with as many poses as possible as well. Poses should be typical poses that one would expect to produce using a positioning system, some easy and some harder.

All these problems can only be answered experimentally. So a pilot study involving four people was performed. The poses used for this experiment were poses than one could see in the literature or poses that people asked me to perform to demonstrate the technique. They were not chosen because they were easy or hard to produce using a particular technique.

2.1.4 Pilot study

Before performing an evaluation, it is necessary to sort out any problems which might arise during the experiment by making a pilot study. Usually two or three people are necessary. Four people were used here. The first two were of major help in solving a few basic problems and improving the interface a lot. Since further implementation

was carried out, again two people were used to ensure the system was ready for the evaluation.

Apart from improving the interface, the pilot study was also used to work out the ideal number of poses, the distances for each pose defining that two poses can be considered similar[1] and to write and test the training sheets.

Figure VI.1: Poses used for the evaluation

These are the poses which were shown to the participants of the study. They all resemble something everyone can relate to and are not difficult to produce using any of the techniques implemented.

It was realised than only three poses can be performed per technique, that is six poses in all. This is a small number; four poses would have been better but it would have made the evaluation too long for many participants. At the end of the pilot study, three poses were selected (Fig. VI.1). The first two poses were used to train the participants, the evaluation being truly performed on the last one. This scheme was also decided during the pilot study as it required on average two poses to get acquainted with any given technique. Each user built these poses twice, once for each technique they had to use.

2.1.5 Performing the experiment

2.1.5.1 Starting and ending the experiment: When users sit in front of the computer and try to perform a pose, there must be something which tells them when to stop. Producing the exact pose is extremely difficult. There is always a slight discrepancy which is not noticeable even to the practiced eye.

Letting the participants decide when to stop is not satisfactory either. They may stop early with one technique simply because they do not like it.

It is not much better to let the experimenter decide when to stop. A technique might, even unconsciously, be favored against the other ones.

[1] This is described in more details later on

As a result, the computer is assigned the duty to test when the user has produced a pose close enough to the target. For this purpose, the computer needs to be able to compare how similar a pose is to another one. Ideally, the judgment would mimic a human referee.

Humans find it difficult to spot small differences. They are much better are spotting big differences. Thus a pose which all limbs points in more or less the same direction as another limb will be virtually identical to a human. At the same time, a pose which is the exact replicate of another one apart from one limb which points into an altogether different direction will look entirely different to a human.

A pose is the exact replicate of another one if:

1. all joint configurations (rotation angles) are the same
2. all positions of the limbs in 3D space are the same

There are two problems with the first solution. First, it does not take into account the size of the limbs (the longer the limb is the bigger the differences are) and second, differences at a joint configuration alters the differences down the branches originating from that joint.

These problems are inherently solved by the second solution. Furthermore, it makes more sense to use this solution as it uses what the users sees, and not a representation which will have to be decoded.

The computer can accurately sum up any differences. Thus, the computer could be used to sum up distances between the end of all limbs and the target limbs. The number obtained would be a kind of similarity ratio. A similarity ratio of zero implies the two poses being compared are identical. A threshold could be worked out and used as a magic number. A distance below that threshold would indicate the two poses are similar enough.

However, this algorithm does not take into account the fact that the human eye is more receptive to big differences than small ones. Consequently, the computer was used to sum up all squared distances instead. At the end, the square root is returned because it simply looks like a more tractable number:

$$OverallDistance = \sqrt{\sum_{i=1}^{i=n} |A_i B_i|^2} \tag{VI.1}$$

where A_i is the coordinate of the tip of the limb i of the first pose.
B_i is the coordinate of the tip of the limb i of the second pose

This ensures that small distances have less weight than big ones. If the *overall distance* is below a given threshold, the computer decides that the user has reached

the pose. This number depends on the size of all limbs of the robot. With our humanoid, a distance of 100 appears to work most of the time. The size of all limbs being 561, this distance is approximatively to 18% of the size of all limbs.

Interestingly enough, some poses are easier to reach than others. For example, perfectly straight positions such as the military position are easier to reach than the ones where all limbs are astray. So the threshold actually depends on the target pose. For the three poses used in this evaluation, the thresholds were obtained experimentally during the pilot study. The differences are 80, 140 and 100 for the first, second and third pose respectively.

2.1.5.2 Description of the experiment: When the participant sits in front of the screen and starts the experiment, the computer first displays the pose to produce (Appendix 4). The participant is allowed to view it from multiple angles. The experimenter also makes sure the participant understands the pose before trying to construct it. When the participant is ready, the computer reinitialises everything and displays the robot in the standard military position. If this is the first pose, a training sheet is provided which explains how the technique works and what has to be done to reach the target pose. The experimenter also stands beside the participant to provide further recommendations and answer any questions. No recommendation was provided for the last pose as it would have invalidated the study. Also, participants were not allowed to ask questions for the same reason. When the pose is sufficiently close to the target pose, the computer stops the task by displaying that it is now finished. The experiment then continues by displaying the next pose.

2.1.6 What to evaluate ?

Most evaluations are about working out if there is a relationship between what are called *dependent* and *independent variables.* The dependent variables describe what is being measured and the independent variables define the conditions under which the experiment takes place. The assumption is that the scores of the dependent variables depend on the independent variables.

Here, the independent variables are the generator, inverse and forward kinematics. This section is about finding useful dependent variables.

2.1.6.1 Quantitative differences: The most useful information is probably the time one takes to achieve a given pose. The shorter it takes to achieve a given pose, the better the technique. Because the computer is in charge of starting and stopping the experiment, it is also in charge of measuring the time taken to accomplish the tasks.

The number of iterations it takes to fulfill a task were also recorded. An iteration is described as moving a limb from one position to another. The way one limb is moved

differs from one technique to the other so it is difficult to see if there is a relationship between techniques. However, it is fair to suspect that the more iterations there are the longer it takes. So the more iterations one technique requires, the less powerful it is. Reducing the number of iterations would be the obvious optimisation. On the contrary, a technique for which few iterations are necessary but is still slow implies the problem lay with the technique itself.

The number of errors is also a useful indication of the power of a given technique. Unfortunately, it was not possible to design an algorithm which would recognise errors. As a matter of fact, it happened that what I initially thought were errors from participants eventually turned out to be just another way to produce the required pose. Consequently errors have been ignored in this study.

2.1.6.2 Qualitative differences: An evaluation cannot be complete without an investigation of the qualitative differences between the techniques being compared. Qualitative differences refers to abstract concepts such as how hard a technique is to use. One technique could well be faster but harder than other techniques.

The workload associated with a given technique provides useful information to appreciate qualitative differences. Research in human factors is partly devoted to the study of workload assessment techniques. Many such techniques have been developed [Jex88, Egg88, Wil88, Rei88, Mes88a, Mes88b]. For a non expert, it is not easy to work out which one would be the most appropriate given a particular problem. Researchers at NASA developed the NASA-TLX(Task Load Index) [Sta88], a workload assessment technique. NASA-TLX has been successfully used in a wide variety of applications, ranging from flight simulators to laboratory tests of problem solving [DAS93, Dem93, RAK93, Bec92, Dud91, MK91, Bre94]. As a result, the NASA-TLX was also used in this study.

Hart & Wickens [SH90] defines workload as:

> the effort invested by the human operator into task performance; workload arises from the interaction between a particular task and the performer.

The basic assumption is that cognitive resources are required for a task and there is a finite amount of these. As a task becomes more difficult, the same level of performance can be achieved but only by the investment of more resources.

When trying to measure the workload associated with a particular task, researchers divide the workload in more precise categories, such as mental workload or physical workload. It was in an effort to standardize workload studies, that Hart & Staveland produced the NASA-TLX. With this method which is the result of many years of study, the workload is divided in six categories:

- **Mental demand:** How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc) ? Was the task easy or demanding, simple or complex, exacting or forgiving ?

- **Physical demand:** How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating, etc.) ? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious ?

- **Temporal demand:** How much time pressure felt due to the rate or pace at which the tasks or task elements occurred ? Was the pace slow and leisurely or rapid and frantic ?

- **Performance:** How successful the participants think they were in accomplishing the goals of the task set by the experimenter ? How satisfied were they with their performance in accomplishing these goals ?

- **Effort:** How hard did they have to work (mentally and physically) to accomplish their level of performance ?

- **Frustration:** How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did the participants feel during the task ?

Users rate the task they have just performed for each categories using a set of scales (Appendix D.1). An overall workload score, based on a weighted average of ratings on the six sub-scales, is then computed.

With the traditional TLX, there is also a paired-comparison between each category to derive weights for a user's subjective feeling of importance of that category. Individual scores are then multiplied by the weights and the average is computed. To speed up the application of the TLX, Byers & al. [JB89] proposed that the weightings were not needed; instead an average of the factor scores could be used to give the overall workload. They carried out a comparison of traditional TLX and 'raw' TLX (just the average). Their results showed no significant differences. They concluded (p. 484):

> RTLX (Raw TLX) is attractive for use because of its simplicity and essential equivalence with TLX. Because of its simplicity, we believe it has substantially greater potential in industrial and research settings than its predecessor. RTLX is recommended for use as a tool for multidimensional assessment of operator workload.

Consequently, The RTLX was used in place of the TLX.

A technique might be slow and difficult to use but it might still be an enjoyable tool. Professional animators work with this type of tool every day. It is obviously preferable if they enjoy using them. Brewster [Bre94] used a seventh category to measure the overall preference. This category was also added for this experiment.

Participants were asked to complete the first set of scales just after the first task had been completed to avoid that during the second experiment, participants forgot the impressions they had of the first technique. The different categories are abstract so it is difficult to judge them on their own. It would have made it easier to rate

each technique when the experiment had ended, but this was ruled out. Instead, participants were allowed to rectify what they had initially specified although this rarely happened. Instead the second ratings were always relative to the first ones. This resulted in different scores for the generator when it was compared against forward kinematics and when it was compared against inverse kinematics. This is acceptable because results need to be consistent only within each part. It does not matter if they differ from one part to the other.

It is not possible to prefer a technique without having tried it before. So an exception to the rule stating that participants must rate the different scales just after each experiment completed, was made to the overall preference. It was rated at the very end instead.

2.1.7 Documents relative to the study

2.1.7.1 Welcome form: Every country has rules to protect participants of a study like this one. Basically, participants are told the information gathered is anonymous and will be used solely for research purposes. What the experiment is about is explained and they are also told that they can leave at any moment should they decide to do so. Such a form is shown in appendix C

2.1.7.2 Training: Training is an important part of this study. Three scripts were written to explain, step by step, how each technique worked (Appendix 4). During the training, the experimenter also stood beside the participant to answer any questions and to make sure that everything was understood.

2.1.7.3 Rating sheets: The first page of these sheets explains what workload is and how it works (Appendix 2). Then a table is shown to explain the meaning of each category. Participants were allowed to consult it at any moment when rating the techniques.

2.2 Analysis

2.2.1 Choice of the statistical technique

A choice had to be made to decide which technique to use to assess the results. Although three techniques were being evaluated, they were only two experiments: The generator versus forward kinematics and the generator versus inverse kinematics.

A few techniques have been developed to deal with this type of problem, one of them being the *t-test*. The t-test compares two groups of scores. The t-test exists in two versions, the *related (correlated)* t-test and the *unrelated(uncorrelated)* t-test (also known as the student t-test). The related t-test compares two sets of scores from the

same group of participants to see if their means differ significantly. If the scores come from two separate group of people, the unrelated t-test has to be used. The latter version has a wider scope of application than its predecessor but may not provide so much useful results [HC97].

The t-test is powerful, simple and robust against noise. It also handles well small samples. Because of all these advantages, both related and unrelated t-tests have been used in this study.

Related and unrelated t-tests also come in two versions, the one-tailed and the two-tailed t-tests. If it is known that scores are in one direction, then the one-tailed t-test may be used. However, this is unlikely to be the case in most studies. It is also strongly recommend to use the two-tailed t-test even if the one-tailed t-test could be used. In this study, only the two-tailed t-test has been used.

The object of an evaluation is to reject the null hypothesis. The *null hypothesis* describes the assumption that the independent variable has no effect on the dependent variable. In this study, the independent variables are the generator, forward and inverse kinematics. The dependent variables are the speed, the workload attributes and the overall preference. The null hypothesis is rejected if results from the t-test are statistically significant. Statistically significant means the results are in the extreme 5%. They are said to be extremely statistically significant if they are in the extreme 1%. To be in the extreme n% means that if the null hypothesis was in fact true, the likeliness of these results happening by pure chance (or lack of it) is less that n%. In other words, the likeliness of making a mistake is less than n%.

At some point, we will have to work out if two paired groups of scores are correlated. If these scores would be plotted on a scattergram, one axis for each group of scores, they would be perfectly correlated if the scores describe a straight line. The Pearson correlation coefficient is a commonly used tool which provides two major pieces of information:

1. The closeness of the fit of the points of a scattergram to the best-fitting straight line through the scores.

2. Information about whether the slope of the scattergram is positive or negative.

Sometimes, the scores may not describe a line but a curve. In this case, the practice is to rank the scores. The problem with that is that some information will be lost (e.g. the distance from the line or the curve). The Spearman's Rho coefficient is another correlation tool. Before doing any calculations, it ranks the scores. So, it is commonly used in place of the Pearson correlation coefficient when the scores fit a curve instead of a straight line.

The Spearson's Rho correlation coefficient could be used all the time in place of the Pearson correlation coefficient but because it discards some information, it is better to use the latter when the scores on the scattergram describe a straight line. Here, everywhere a correlation had to be calculated, scores did not describe a curve but rather a straight line. As a result, the Pearson correlation coefficient was used.

2.2.2 Generator versus Forward kinematics

The first part of the analysis compares the generator and forward kinematics.

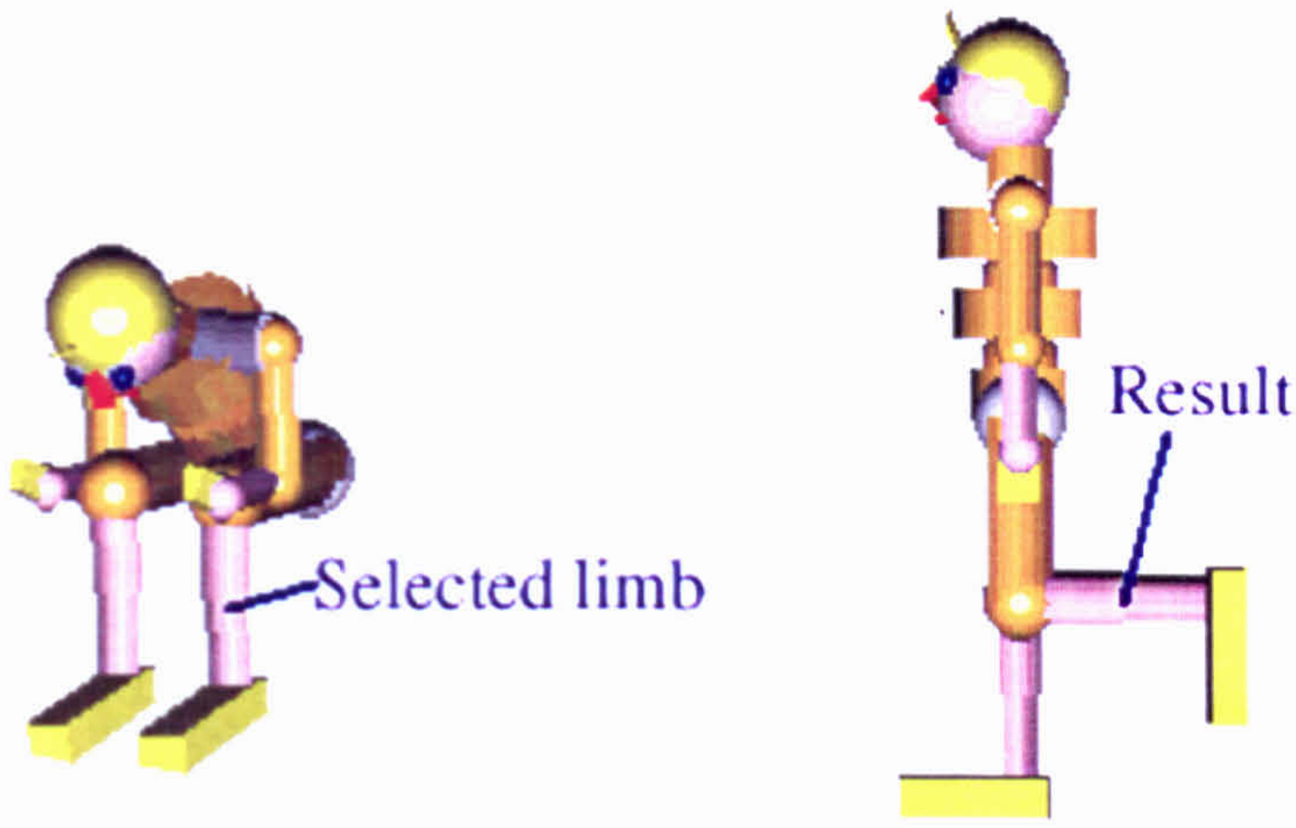


Figure VI.2: Understanding the generator

Starting from the standard military stance (not shown here) and clicking on the leg shown on the figure on the left will produce the configuration on the right. Some participants had trouble understanding this. It also requires some thinking to work out what the outcome of a selection will be.

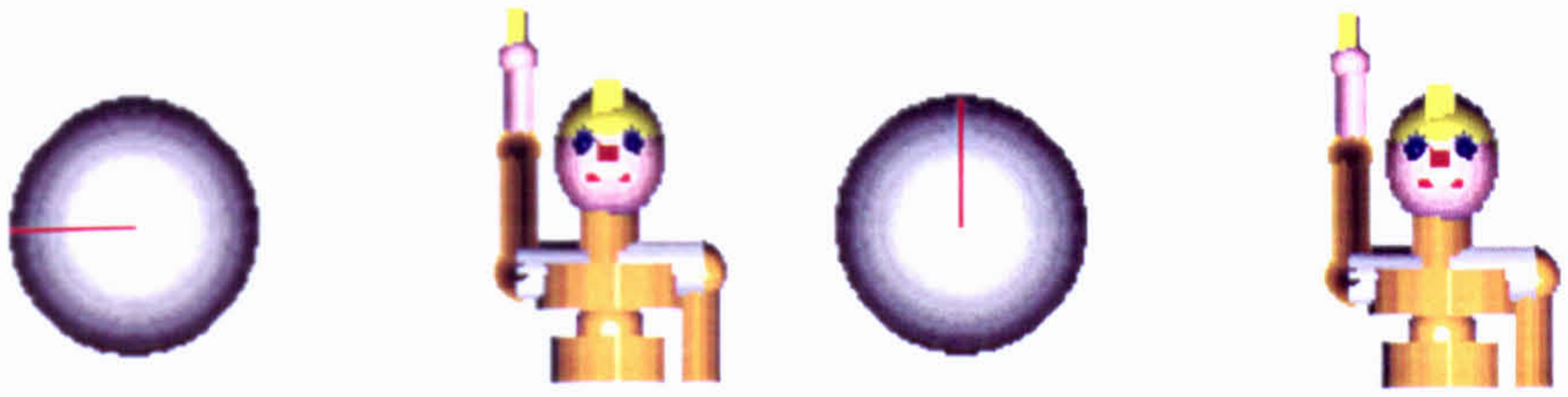


Figure VI.3: Using forward kinematics

As can be seen from this figure, although the right arm is in exactly the same position, it was brought there by two different paths. The first went through the pivot plane whereas the second one used the flexion plane only. Some participants found this situation very confusing.

2.2.2.1 Workload

2.2.2.1.1 Mental workload: Before the experiment took place, it was strongly suspected that the mental workload associated with the generator would be a lot higher than the mental workload associated with forward kinematics. With the generator, users have to look at each pose to find what they need to produce the required pose. To make matters worse, users have to workout the position of the limb relative to its parent and not the global position of the limb (Fig. VI.2). This is clearly more difficult than just using a tool to drag the limb in the required position. However, most

participants had problems achieving some positions using the joint balls. Basically, they are an infinite number of paths to rotate a limb by 180°. When users need to turn the limb in one direction by an angle of say 160°, they may take a wrong path. They get close at the beginning but are not able to get closer after a while (Fig. VI.3). Consequently, they had to think a lot to understand what was going on. As a result, although the mean was in favor of forward kinematics, the difference was not statistically significant (Fig. VI.4).

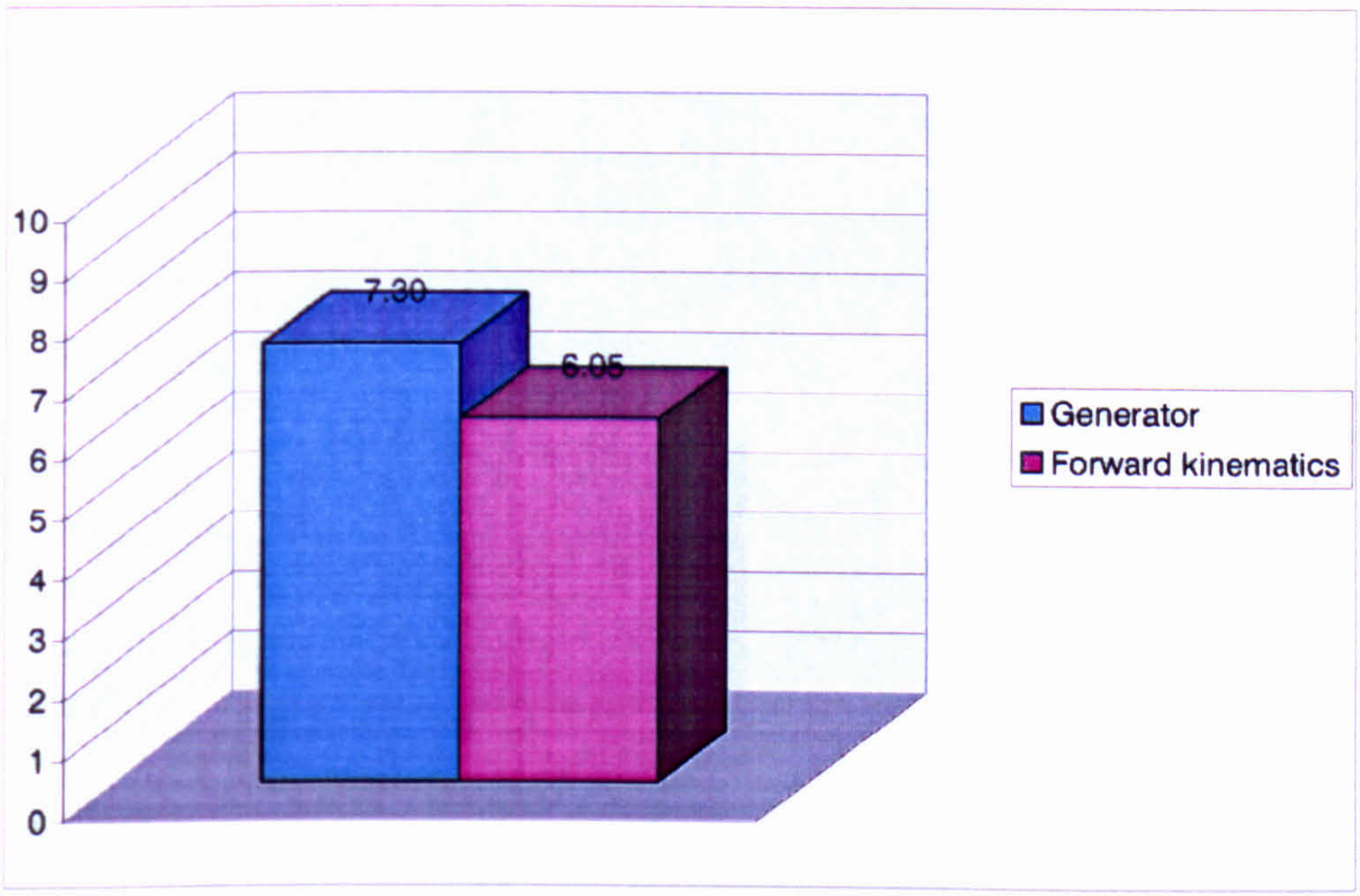


Figure VI.4: Higher mental workload with the generator

Although the mental workload was higher with the generator, the difference was not statistically significant.

2.2.2.1.2 Physical workload: Although dragging a mouse is not what one would call physically difficult, it is more difficult than just a single click. As a result, it was expected that the physical workload associated with forward kinematics to be higher than the one associated with the generator. Although the mean was in favor of this assumption, the variance associated with forward kinematics was too high for the difference to be significant.

2.2.2.1.3 Time pressure: Time pressure was a difficult category to comprehend. Time pressure is high when things are going to fast and this was not the case for anyone. They were virtually no difference amongst the two techniques.

2.2.2.1.4 Effort expanded: As a whole, participants found the generator harder to use than forward kinematics but the difference in the mean was not significant.

2.2.2.1.5 Performance level achieved: As a whole, participants also felt that they were better at using forward kinematics but the difference was not significant.

2.2.2.1.6 Frustration: It was expected that frustration would be higher for the generator since some participants had problems finding the right limb positions. However, the problem illustrated in Fig. VI.3 counterbalanced this effect and there was virtually no difference between the two techniques.

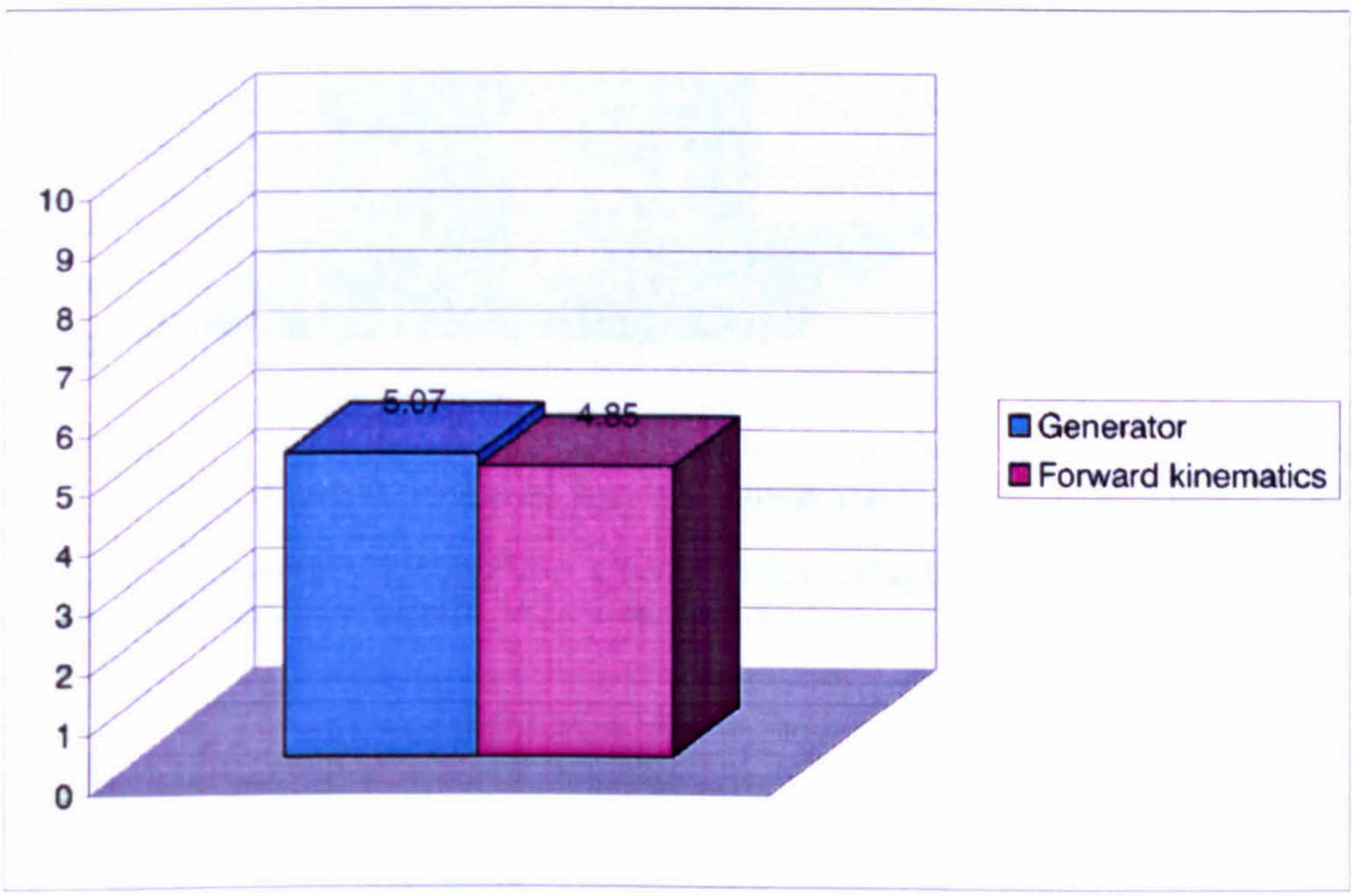


Figure VI.5: Forward kinematics's workload lower than generator's

Although the workload associated with forward kinematics was lower than the workload associated with the generator, the difference was not statistically significant

2.2.2.1.7 Workload: The workload was slightly higher for the generator but the differences were definitely not significant (Fig. VI.5).

2.2.2.2 Overall preference: As suspected however, the overall preference was strongly in favor of forward kinematics. There is a negative correlation of $t = -2.25$ between the overall preference of the generator and forward kinematics which is significant to the 5% level with a sample size of 10 (Fig. VI.6).

2.2.2.3 Speed: Interestingly enough, participants managed to produce poses faster using the generator although the difference between the two means was not significant (Fig. VI.7).

2.2.2.4 Number of iterations: Interestingly enough there was no difference between the number of iterations used in both techniques although the concept of iter-

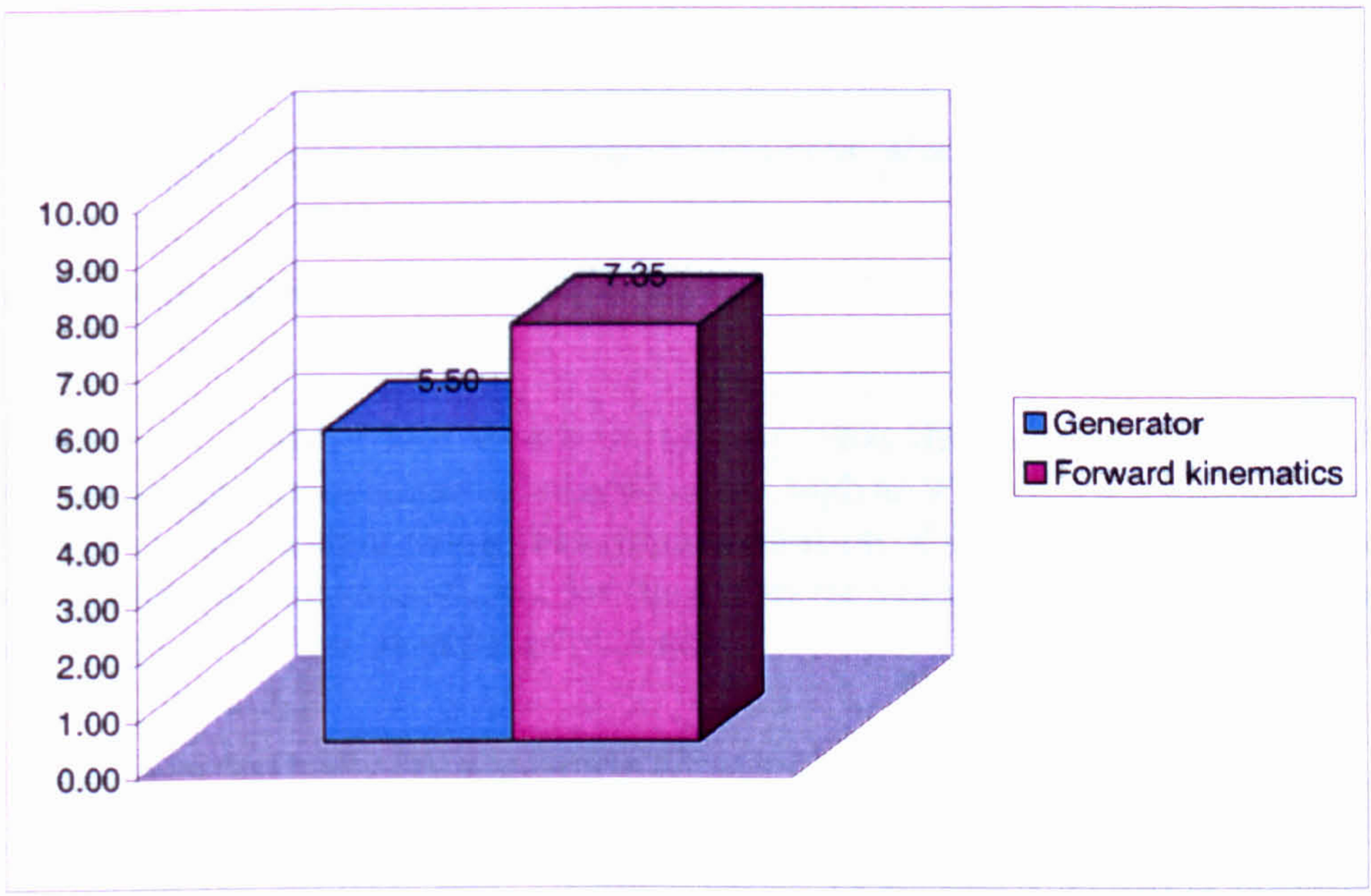


Figure VI.6: Participants preferred forward kinematics

Participants preferred using forward kinematics than using the generator. This difference was statistically significant to the 5% level

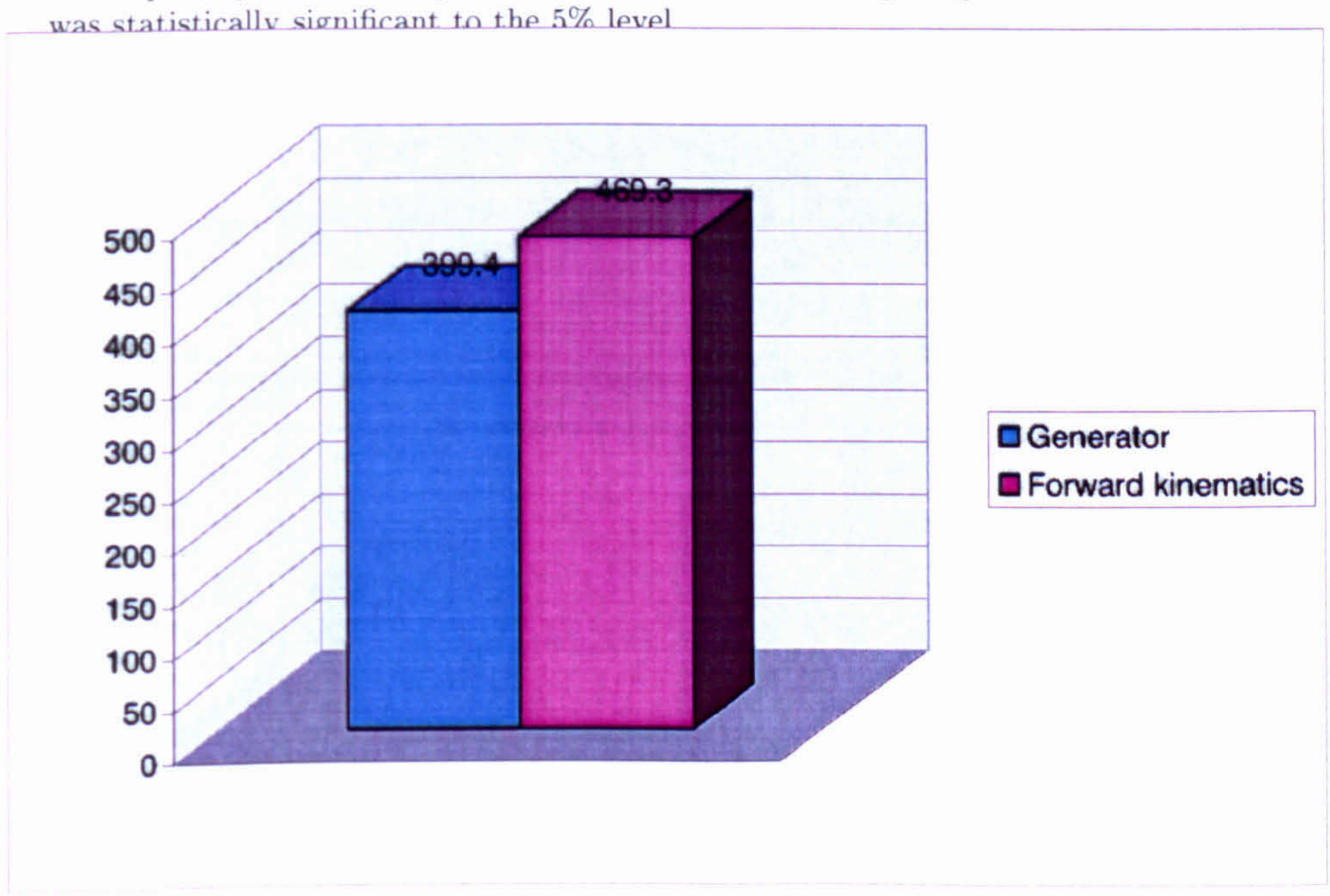


Figure VI.7: The generator faster than forward kinematics

Unexpectedly, participants constructed poses faster using the generator than using forward kinematics. However, this difference was not significant.

ation is quite different for one technique to the other. Basically, each iteration took a little less time with the generator as it did with forward kinematics, that is it took slightly longer to position a joint in the right configuration as it took to look for a correct configuration. These differences were not significant though.

2.2.3 Generator versus inverse kinematics

The first part of the analysis compares the generator and inverse kinematics.

2.2.3.1 Workload

2.2.3.1.1 Mental workload: It was expected that mental workload would be higher for the generator although maybe not as high as with forward kinematics. However since moving a limb using that implementation of inverse kinematics required a bit of practice, the mean was lower for the generator than it was for inverse kinematics. That difference was not significant though.

2.2.3.1.2 Physical workload: It was also expected that physical demand would be higher for inverse kinematics since users had to select and then un-select joints, rotate the camera, etc. Unsurprisingly, there was a negative correlation of $t = -2.82$ between the physical workload of the generator and inverse kinematics which is significant to the 5% level with a sample size of 10 (Fig. VI.8).

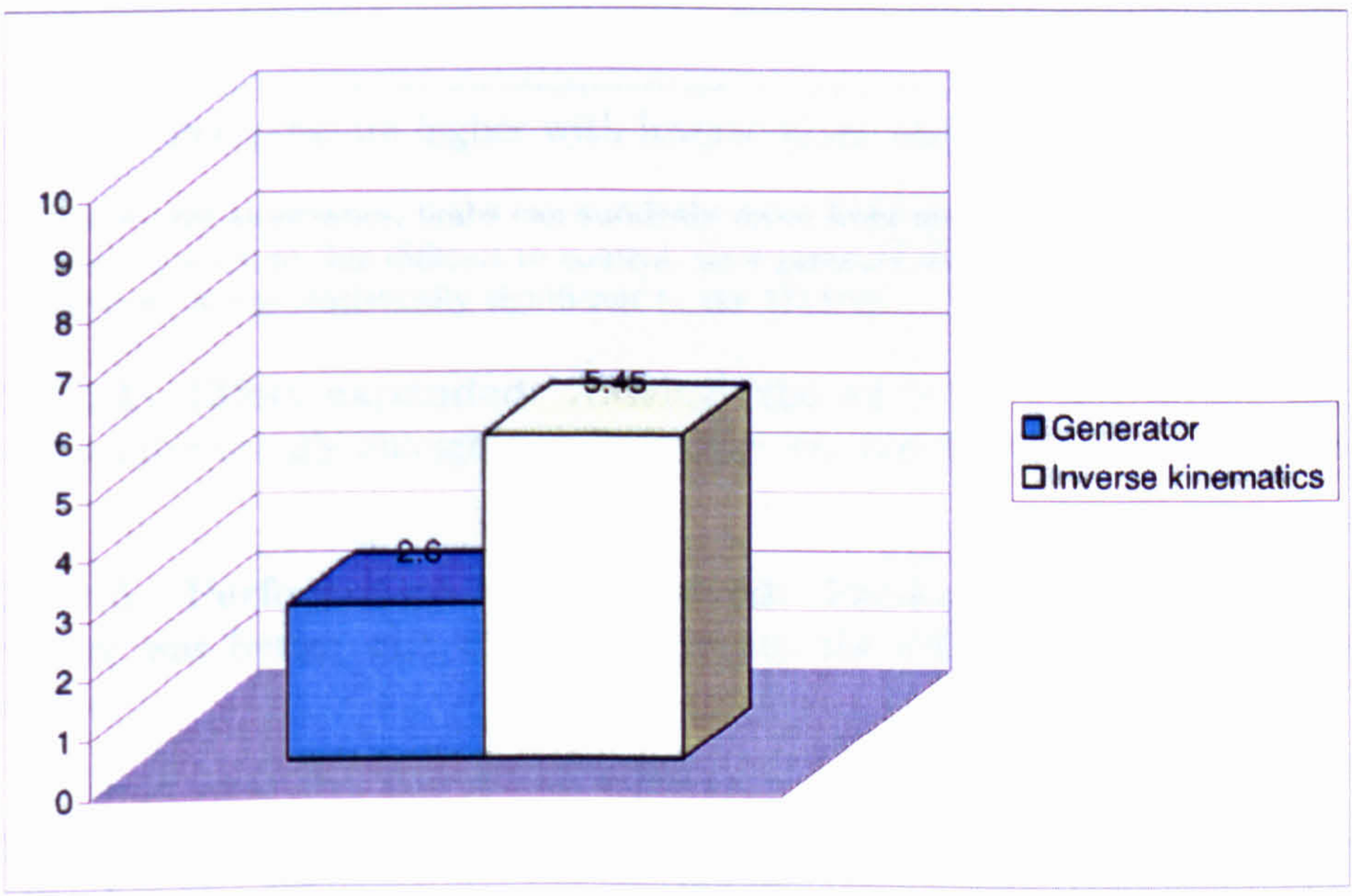


Figure VI.8: Physical demand higher with inverse kinematics

Not surprisingly, participants found inverse kinematics were a lot more demanding physically than the generator was. This difference was statistically significant to the 5% level.

2.2.3.1.3 Time pressure: When using inverse kinematics, limbs can jump from one position to another. This is difficult to control. As a result, participants gave high scores for this category to inverse kinematics. So there was a negative correlation of $t = -3.30$ between the time pressure of the generator and inverse kinematics which is extremely significant to the 1% level with a sample size of 10 (Fig. VI.9).

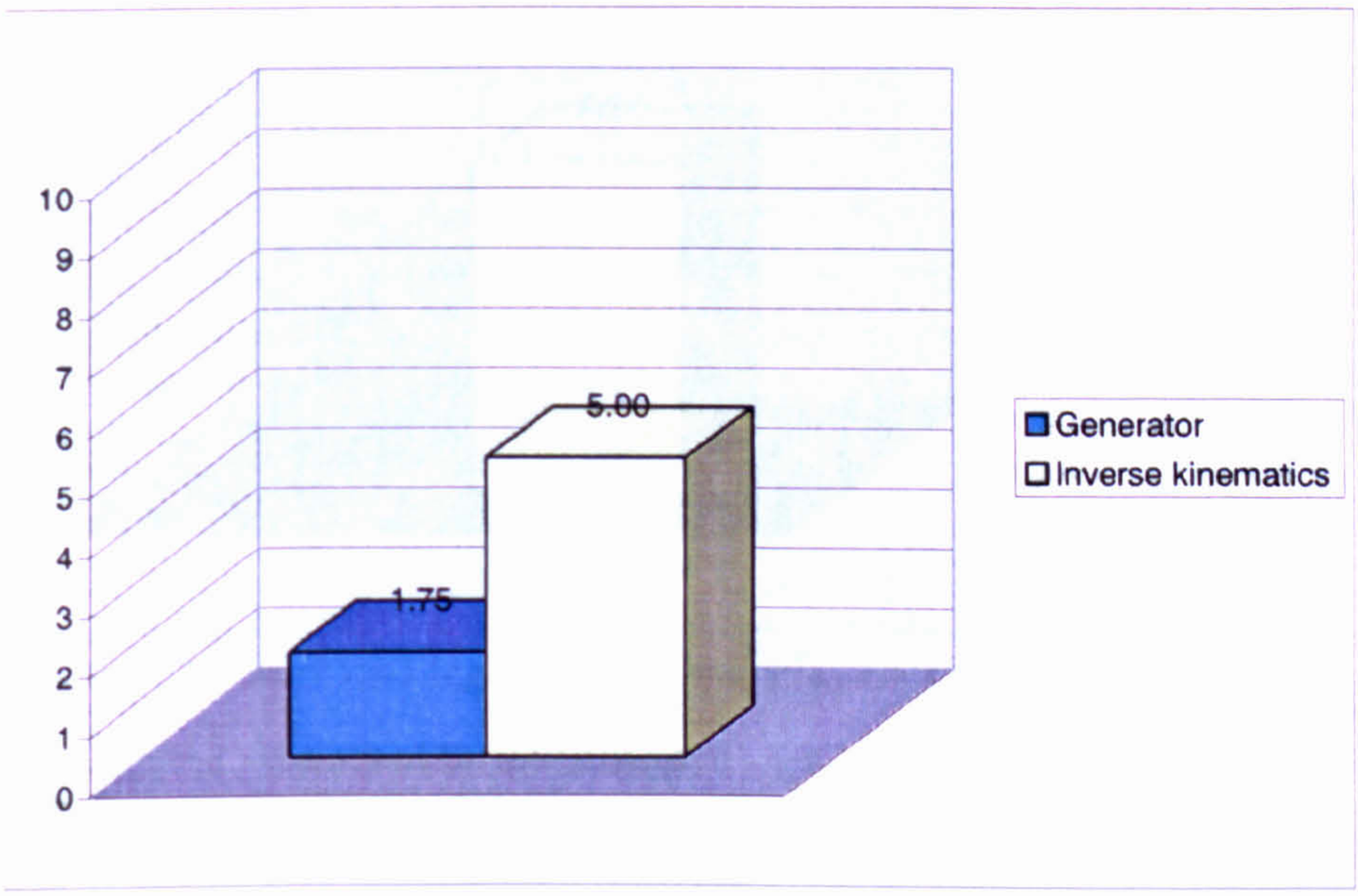


Figure VI.9: Time pressure higher with inverse kinematics

With inverse kinematics, limbs can suddenly move from one position to the other. As participants found this difficult to control, time pressure felt was higher than with the generator. It was statistically significant to the 5% level.

2.2.3.1.4 Effort expanded: Although the mean is slightly higher for inverse kinematics, interestingly enough, the difference was not statistically significant.

2.2.3.1.5 Performance level achieved: Participants also found that their performance was better with the generator but the difference was not statistically significant.

2.2.3.1.6 Frustration: As expected, participants found inverse kinematics more frustrating but the difference was not significant.

2.2.3.1.7 Workload: All factors combined, there was a negative correlation of $t = -2.37$ between the workload of the generator and inverse kinematics which is significant to the 5% level with a sample size of 10 (Fig. VI.10).

2.2.3.2 Overall preference: However although participants preferred the generator, the difference was not significant. It is because of too large a variance amongst

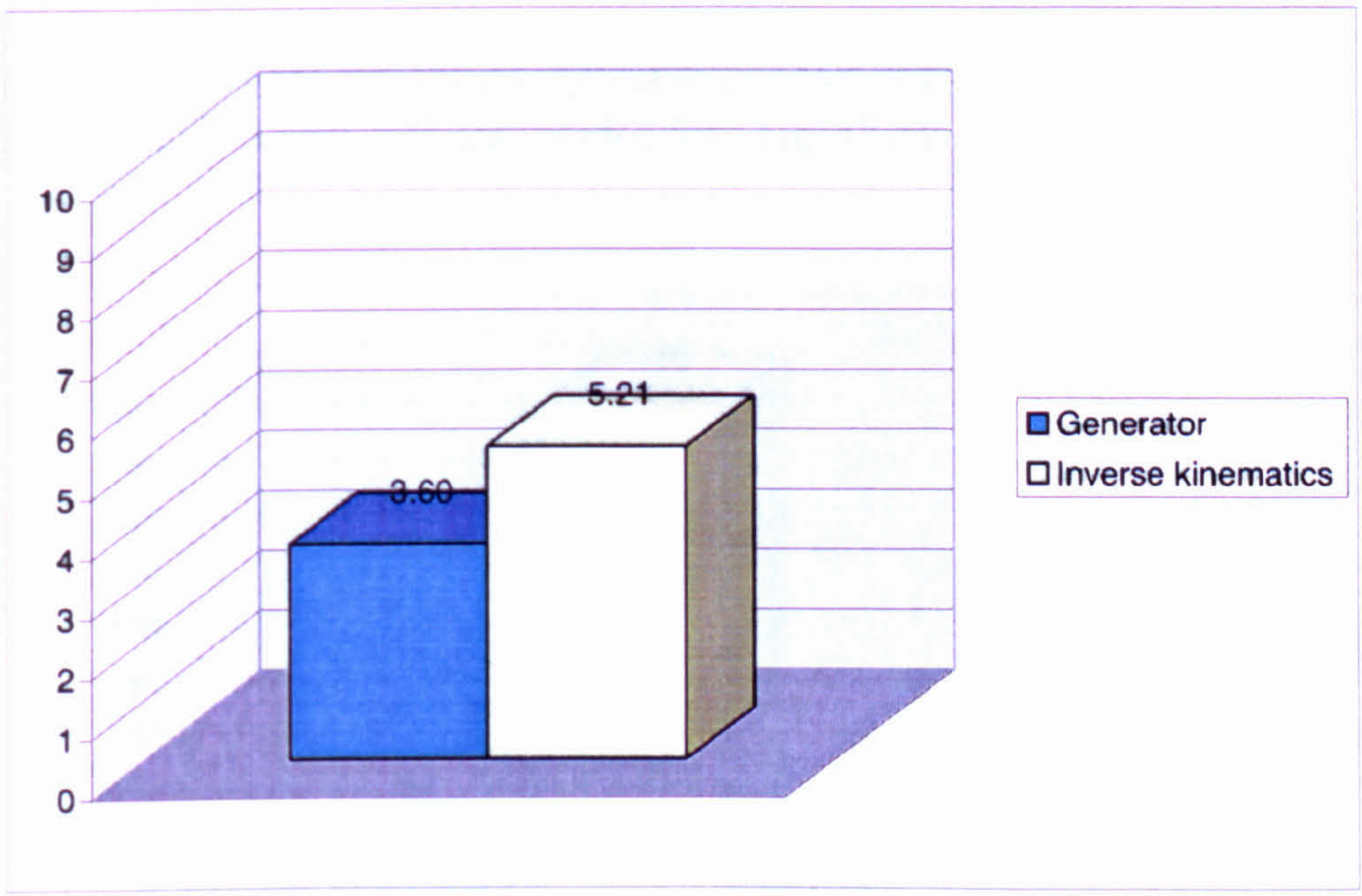


Figure VI.10: The workload higher for inverse kinematics

> The workload experienced by the participants was higher for inverse kinematics. This difference was significant to the 5%.

the scores of the inverse kinematics technique.

2.2.3.3 Speed: Participants produced poses faster using the generator however the difference between the two means did not reach statistical significance, although it was getting quite close to it (6% and needed 5%. Fig. VI.11).

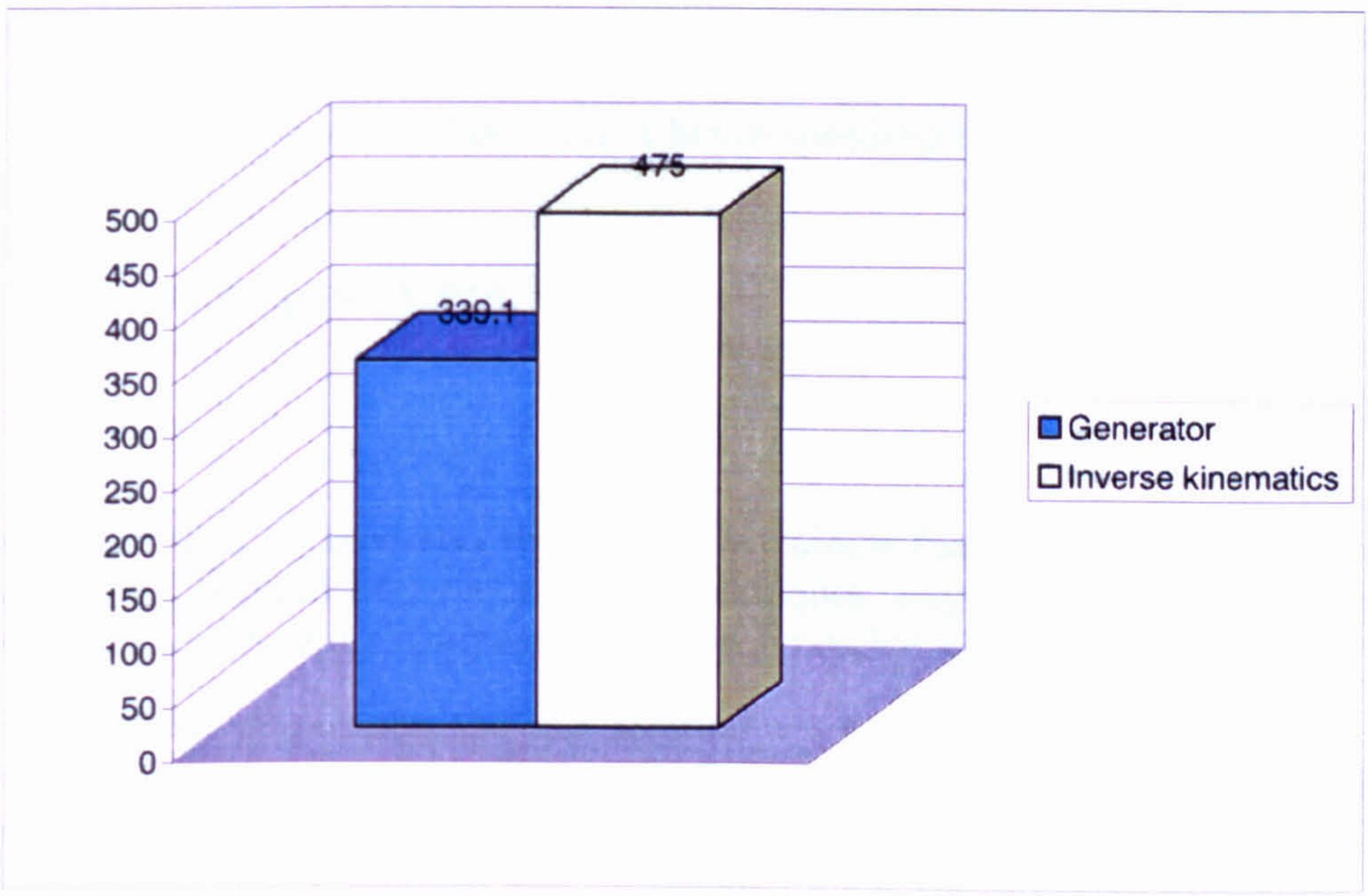


Figure VI.11: The generator faster than inverse kinematics

Although participants built poses quite faster using the generator than inverse kinematics, this difference was not statistically significant.

2.2.3.4 Number of iterations: There was no difference for the number of iterations used by both techniques. As a result, each iteration took slightly longer to complete using inverse kinematics than it did using the generator. However, the difference did not reach statistical significance.

2.2.4 Conclusion

With this implementation of the generator, forward and inverse kinematics, most differences were not statistically significant. A larger sample would have been necessary to get more significant results.

The hypothesis arguing that the generator will be easier to use than forward kinematics was rejected. Its opposite was also not proven. However, the preference of the participants in favor of forward kinematics was significant.

However, the hypothesis arguing that the generator will be easier to use than that implementation of inverse kinematics was proven.

There was no significant difference for the time taken and the number of iterations used to achieve the last pose. However, the Pearson coefficient showed there was a strong correlation to the 5% level between the time taken to achieve the last pose and

the number of iteration used. Since a finite number of iterations was required to achieve this particular pose, it implies that all the unnecessary iterations are errors. Thus the number of errors is strongly correlated to the speed of the technique. Errors happened when users selected joint configuration just to see if they fitted with what they were looking for. It can then be assumed that a better knowledge of the relationship between joints and limbs, and also a better knowledge of position of the joint configuration would decrease the number of errors and hence speed-up the technique.

3 Second experiment

3.1 Design

It was felt that the generator required more training than the other two techniques. Although people could improve using all techniques, improvements with kinematics would stagnate before improvements with the generator would.

To check this assumption, the only solution was to perform another evaluation but this time using one or more expert users. It can be safely assumed that variability amongst expert users is less than variability amongst non expert users. So this type of evaluation typically requires less particpants than the previous evaluation. An evaluation involving a single participant is thus allowable. Apart from me, no one was available. Theoretically, it would be possible to train someone to use all techniques but this would require a whole day or even longer to bring this person to my level of expertise. Consequently, it was decided to carry out an evaluation with myself being the sole participant.

Although this type of evaluation is uncommon, it has actually already been used. Results can only be interpreted as an indication that the hypothesis is indeed true. Another study should be carried out to confirm these results.

3.1.1 Improvements

During the first experiment, it was realised that the implementation of inverse kinematics could be improved a lot:

- One of the problems was due to the fact that the camera tended to be used a lot more than with other techniques. Rotating the camera to see the articulated figure from a different angle takes a bit of time. To avoid this, the arrow keys were mapped onto four different views. Thus, rotating the camera is both much faster and more accurate.

- Initially, side views were meant to visualise and select body parts, not to rotate them. This was changed so one could position body parts using any view. This made a major difference to the power of the technique. In fact, the figure needed

not be rotated anymore, all three views being sufficient. During the evaluation, no rotations were performed.

- When dragging body parts around, more limbs than expected might be involved in the process. This is really annoying and the solution to this problem was to draw a rectangle to select the parts which are allowed to move. This also takes a bit of time. Instead, it is now possible to type in the maximum number of limbs involved in the process. Thus typing 3, and dragging the hand around will also drag the forearm, the upper-arm but not the torso. This also made a major difference on the performance of the technique.

- Sometimes, a set of limbs will jump from one configuration to another. They are perfectly valid kinematically but it is annoying and difficult to control. Girard & al. [Gir91] solved this problem by using forward dynamics to interpolate from one position to the other so that smooth motion resulted. That solution would have been too costly to implement on the type of hardware this system was running on. Instead, a simple frame correlation scheme was implemented. Basically, if after asking the skeleton to reach for a goal, the new configuration does not bring the end effector closer to this goal, it is simply discarded. The motion is still not C^2 continuous but this improvement was satisfactory.

Unlike for inverse kinematics, no improvement could be seen to be made with forward kinematics. Although the use of joint balls necessitates a bit of practice, it is still fairly easy and intuitive.

After the first experiment, two major improvements for the generator were devised:

- As can be seen from the analysis of the results of the first experiment, the mental workload is high (although differences were not statistically significant). This is mainly due to the fact that users have to look for joint configurations and to compute mentally what would the outcome be if they selected them (Fig. VI.2). The technique was modified so that when a limb is selected, it is copied in the poses produced by the generator as well as the seed. Thus, the mental computation required would lessened.

- It is also common that when a limb is selected, it still needs to be moved a bit or twisted a bit. Generating a new population requires approximatively five seconds. So, it might be helpful if when a limb is selected, the user can also specify that it still needs to be mutated for flexion & pivot or twist. This could be done using the *control* or *shift* keys. For instance, if the user holds the shift key down whilst selecting a limb, the computer would copy that limb in the seed pose and display all poses of the main window again but with this limb slightly mutated. A mutation intensity of 20% usually works well. Unfortunately, there was not enough time to implement this scheme.

3.1.2 The dependent variables

There is no point in using the NASA-TLX anymore. The TLX can only be performed with a set of users and I was the only participant.

The time spent and the number of iterations used to construct a pose are still useful information. On top of this, the number of generations used and the number of page switch were also added. For each generation, the computer generates three pages of poses. If a joint configuration cannot be found in one page, it may be in the next ones. A page switch describes the change from one page to another. A drop in performance as the number of generations and page switch increased could then be expected. This would suggest further improvements.

3.1.3 Design of the experiment

For this second experiment, the computer randomly generated poses which had to be constructed using one of the techniques. There are two ways this experiment could have been performed:

1. Each technique is used in turn to produce a complete different pose each time. There is no order effect problems but with small samples, results might be affected by the fact that some poses are harder to produce than others.

2. Each pose produced is constructed by the three different techniques. Thus, if a pose is harder to construct, it will be harder for all techniques, not just one. However, the order effect reappears which needs to be dealt with.

The second option was used as it should produce more accurate results when small samples are used (it will allow to use correlated analysis which is always better than uncorrelated analysis). To solve the order effect, a simple circular queue scheme was used so that the first technique to construct a new pose changes at each new pose.

To make sure that statistically significant results would be obtained, forty poses had to be produced. I also trained for the evaluation by producing a bit more than fifty poses with each technique. Since the implementations of the generator and inverse kinematics had evolved, I needed to familiarise myself with them. Since the generator was first implemented, I produced several hundred poses with each technique. To train another person would have been difficult indeed.

The t-test is appropriate when two group of scores have to be compared. To compare three group of scores, another tool has to be used. The analysis of variance (ANOVA) can also be used in place of the t-test. The advantage of the ANOVA over the t-test is that it can be used to compare several group of scores. The t-test is simpler and displays more information than the ANOVA, this is why it was used. The ANOVA test exists in several versions: one way with uncorrelated scores, one way with correlated scores and multi-factorial or n-ways with uncorrelated scores. The

latter is used when the experiment involves several types of independent variables. Here, there is only one type of independent variable, that is, the technique used. If during this experiment, scores had also been measured under the effect of alcohol for example, the multi-factorial ANOVA would have been used. Since each technique is used to produce every single pose, the related/correlated ANOVA test can be used. It will produce more useful results than the unrelated ANOVA test.

3.2 Analysis

3.2.1 Time spent

As expected, results were statistically significant. In fact they were extremely significant. As can be seen from Fig. VI.12, on average, it took me 36 seconds to produce the poses using the generator. It took 55 seconds and 47 seconds using forward kinematics and inverse kinematics respectively. The differences between the generator, forward kinematics and inverse kinematics are extremely significant to the 1% according to the one way correlated ANOVA test.

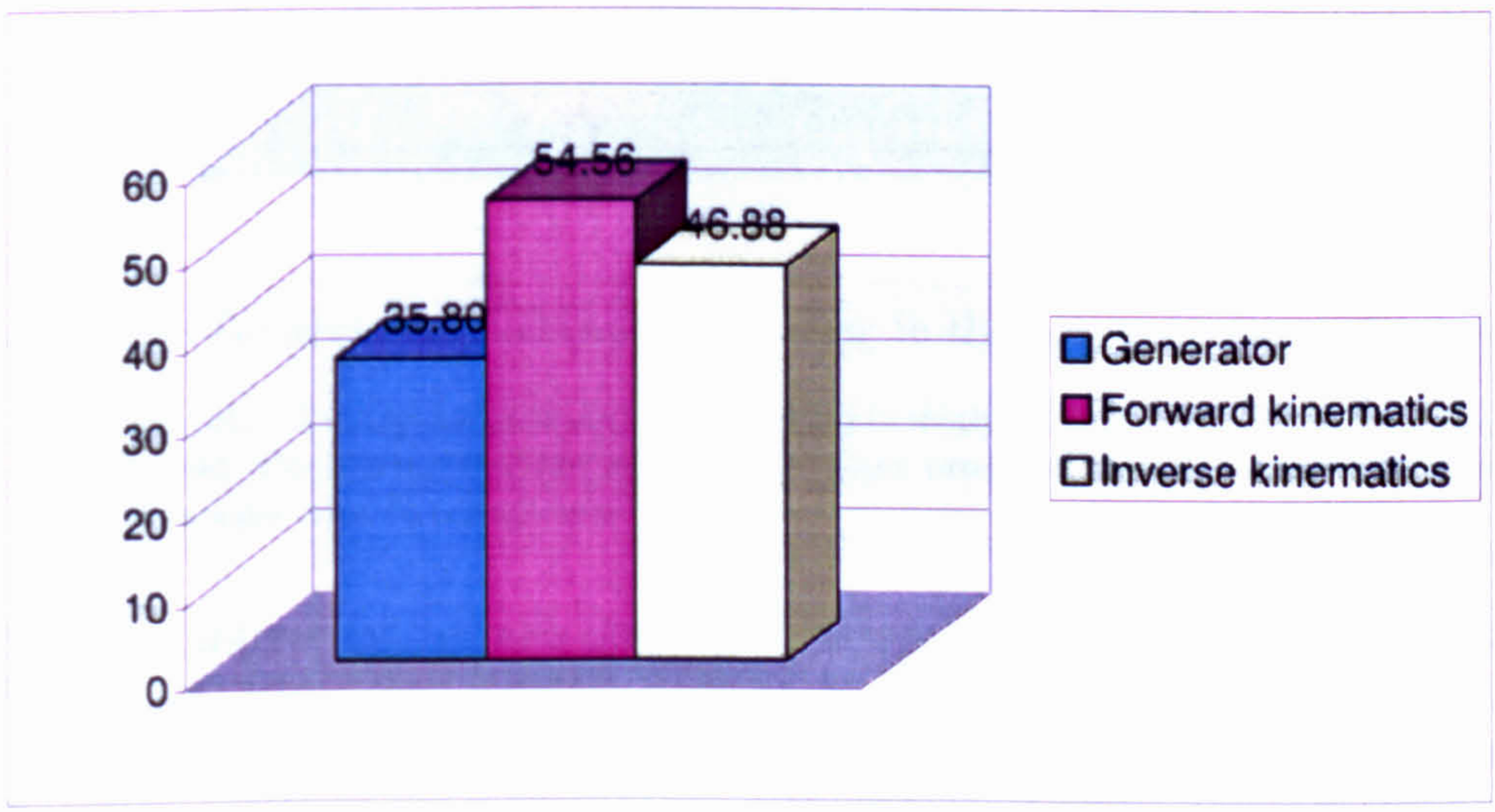


Figure VI.12: The generator faster than both forward and inverse kinematics

The generator was much faster than both forward and inverse kinematics. This difference was extremely significant to the 1% level.

Although I used the generator for a long time, I still had to search through poses to find configurations that I did not know well. I reckon that with more training I could do better and decrease the time necessary to produce a pose under 30 seconds. Interestingly, the last ten poses were constructed in just under 30 seconds whereas the first ten were constructed in slightly more than 46 seconds (Fig. VI.13). There is no such statistically significant evolution with forward and inverse kinematics.

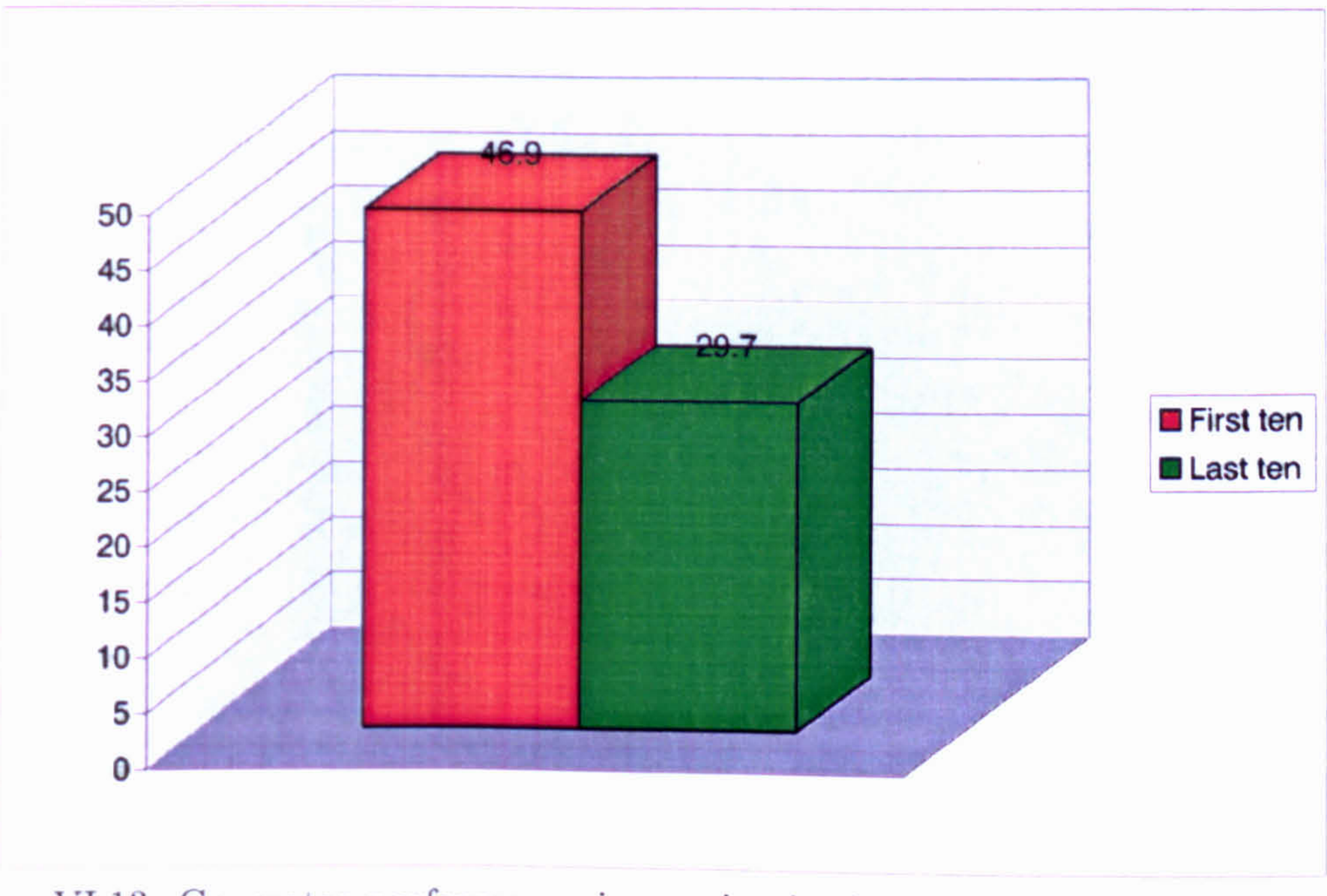


Figure VI.13: Generator performance improving in the course of the evaluation

> Poses produced using the generator improved consistently while the evaluation was being performed. Comparing the first and last ten poses produced, we see a clear difference which is significant to the 5% level.

It is also interesting to note that the new implementation of inverse kinematics performs a lot better than forward kinematics. Before this evaluation, I tried to use the previous version, and it was taking me on average twice as long to construct a pose.

It is important to note that the number of joints is rather small (seventeen). As the complexity of the skeleton increases, so would the power of inverse kinematics. However, for the purpose of entertainment, the complexity of the skeleton used was felt to be good enough.

3.2.2 Number of iterations used

Although the number of iterations used was slightly higher with forward kinematics (Fig. VI.14), the differences were not statistically significant at all. In other words, each iteration is performed faster using the generator than using the other two techniques (Fig. VI.15).

Figure VI.14: No difference in the number of iterations

> Although the number of iterations used was marginally higher using forward kinematics, this difference was not significant.

3.2.3 Iterations, generations and page switches

The Pearson correlation coefficient was calculated to work out if there was a relationship between the time spent and the number of iterations, generations and page switches used. Unsurprisingly, there was a strong positive correlation which was significant to the 1% level for each variable.

Interestingly enough, the correlation coefficient was higher with page switch and even higher for the number of generation used than for the number of iterations. However, the statistical significance of this discovery could not be assessed. This would suggest that the cost associated with a generation switch is higher than when there is no generation switch. This is not surprising as a new generation tends to create configurations that the user has not seen before. Hence, the user has to look at each pose instead of trying to remember where is the best configuration. Switching to another page also seems to affect the performance of the technique. This is not a surprise as the first page tends to be a lot more well known than the others. Furthermore, there

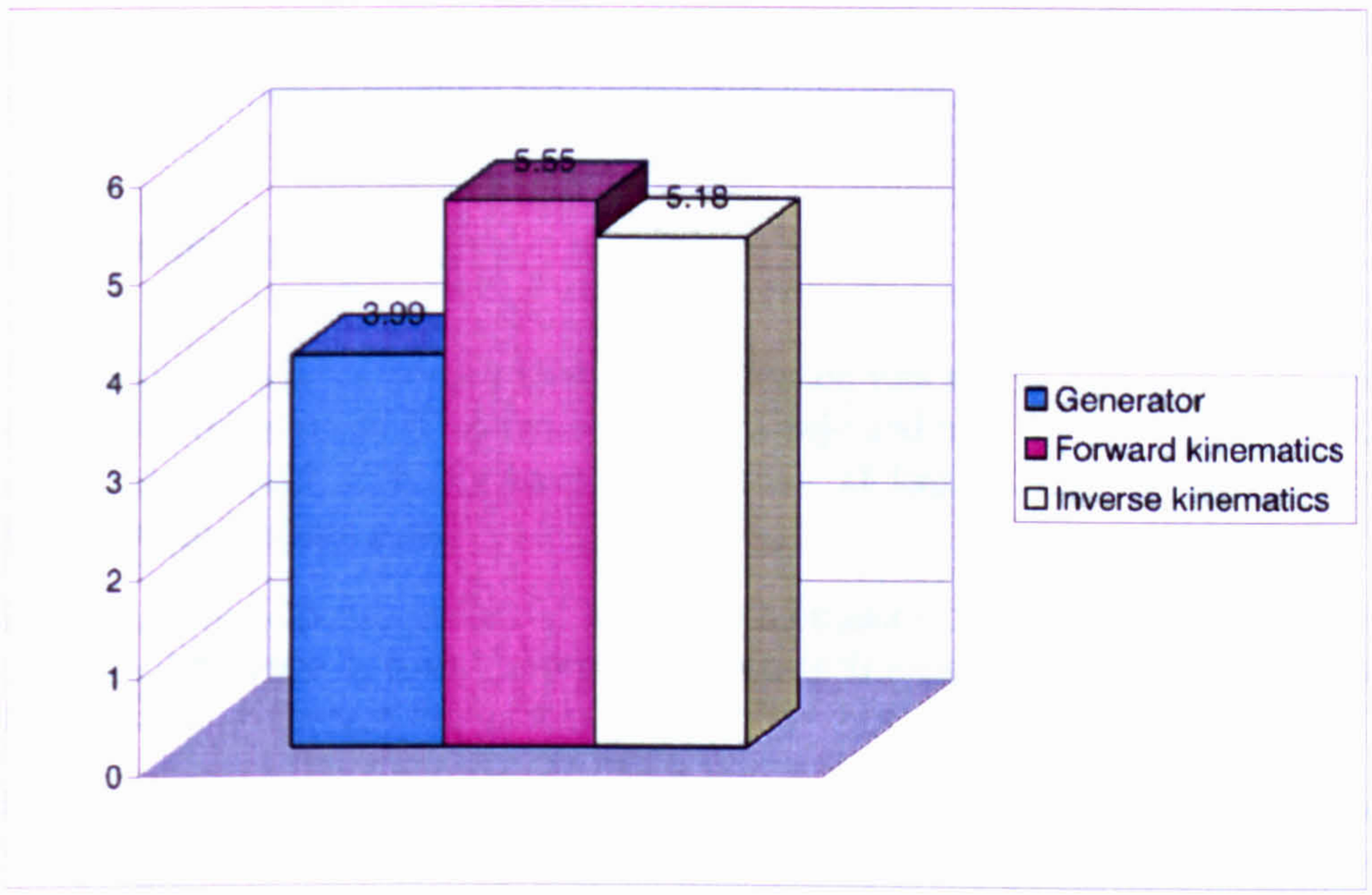


Figure VI.15: Less time per iteration using the generator

Each iteration was performed faster using the generator than using the kinematics. This difference was statistically significant to the 5% level.

is also a cost which is due to the rendering process. Although rendering is fast, a page switch takes on average three seconds on my workstation. A generation switch takes slightly more.

Before performing this evaluation, the generator was improved so that each joint configuration selected by the user is also copied in the pose produced by the computer. The reason behind this is simple: if the user has selected a particular configuration, the other configurations will not be used. They will just make the task of the user harder. Although its significance could not be assessed, this improvement had a cost in terms of errors and thus performance. Poses were redrawn in the background so that the user could carry on selecting other configurations. Ideally, this rendering process would have taken place inside another thread. However producing a multi-threaded application is a lot harder than producing one which uses a single thread. Instead, a scheduler in which poses were rendered when the application was idle was implemented. This works fine most of the time but if the user clicks somewhere while a pose is being rendered and move somewhere else, when the application will have finish rendering that pose, it will retrieve the mouse click event but at the current mouse position. As a result, a wrong joint configuration might be selected. This type of error is very costly. The user may not realise it soon enough which would result in an increase in the number of generations (A generation is necessary to create again the information lost). Unfortunately, it was not possible to measure how often these errors appeared. With training, it is possible to reduce these errors and their impact. This would explain some of the differences in performance between the first and last ten poses. In fact, it was felt that this improvement was more a hindrance than an

advantage since I knew so well the first pages. A truly threaded implementation should completely solve this problem.

4 Conclusion

This first hypothesis argued that the generator was easier and faster to use than the other two techniques. This hypothesis was rejected and, in fact, a bigger number of participants would probably have proven that, at least, the generator is harder to use than forward kinematics.

Although participants preferred forward kinematics over the generator, there was virtually no difference in speed between the two techniques. It is unfortunate that the implementation of inverse kinematics was not perfect.

However, since a training effect was clearly recognisable in the second evaluation, special care has to be taken before generalising these results. It may well be that given sufficient training, animators do not find the generator cognitively difficult for instance.

Furthermore, the small size of the samples and the great variability within them raises questions about these samples being truly representative of the population ther are coming from. For instance, another similar study may well draw different conclusions.

The second hypothesis was arguing that given sufficient training, the generator will help produce poses faster than using forward or inverse kinematics.

Using an improved version of the generator and inverse kinematics and using me at the sole but expert user, results seemed to prove that indeed, given sufficient training, the generator will perform better than the other two techniques. It is also interesting to note that inverse kinematics performed a lot better than forward kinematics. This comes more in line with what expert animators would expect.

There was a statistically significant improvement in terms of speed from the first ten to the last ten poses using the generator (Fig. VI.13). Although there was also an improvement for forward kinematics and a decrease in performance for inverse kinematics, these changes were not statistically significant. Analysing the logs from when I was using the system to train myself before the evaluation, all techniques improved (and these improvements are statistically significant) but improvements with kinematics were far from being as good as they were with the generator (Fig. VI.16). The time taken to generate a pose with the generator was nearly halved. This is another indication in favor of the assumption indicating the generator is faster than the other two techniques given sufficient training.

Although, I felt the mental workload is still higher with the generator than it is with the other techniques, it tends to diminish as my knowledge of the first few pages improves. What seems to be the most tiring is the process where the user has to

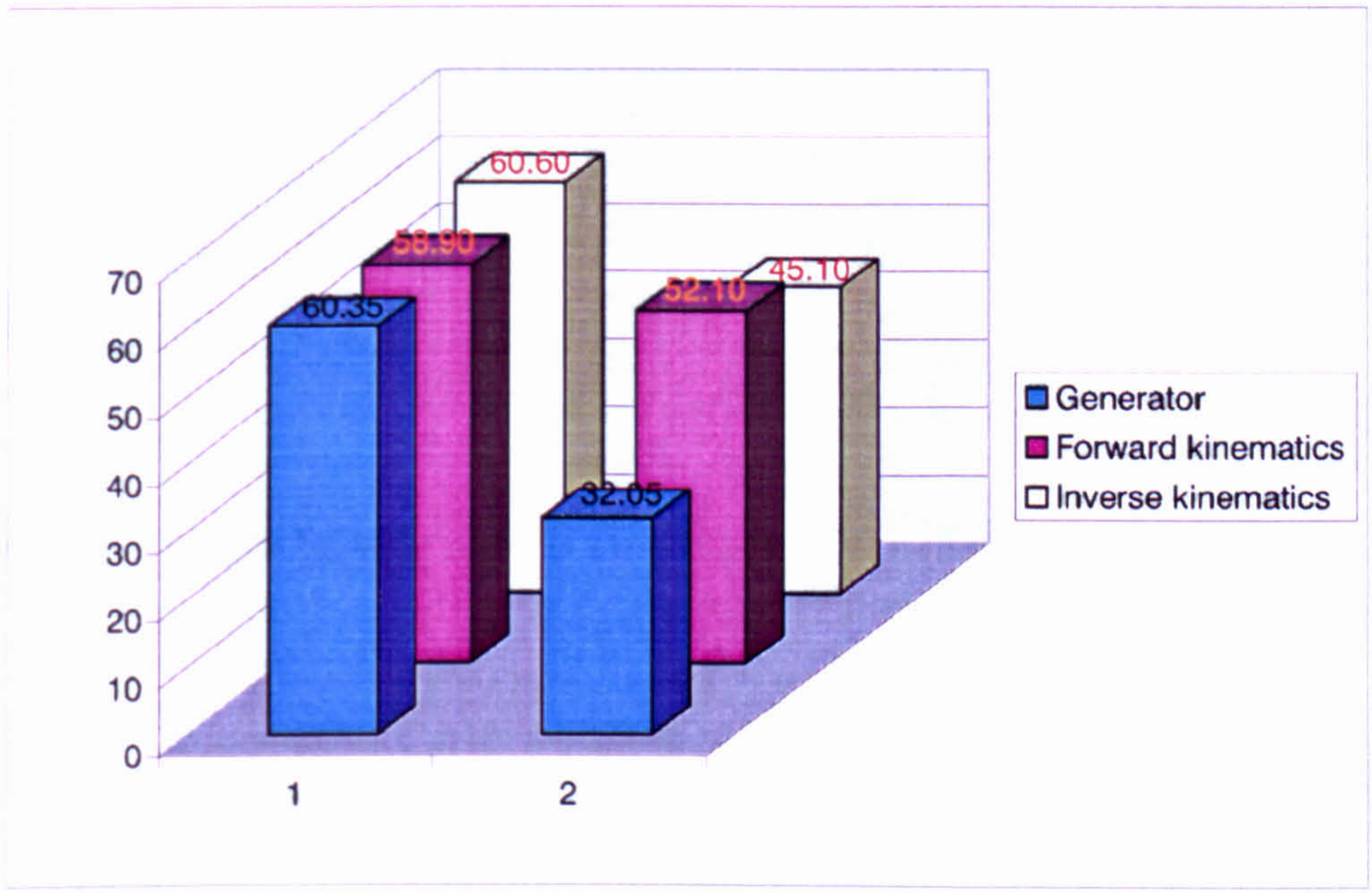


Figure VI.16: The generator improving more than other techniques

Although all techniques improved from the beginning of the training session to the end of the evaluation, improvements from the generator were a lot more significant. These results are all significant to the 5% level.

scrutinise each pose to find what is of interest. Scores for the other categories of the NASA-TLX would be fairly low. On the contrary, mental workload would be low with kinematics and physical workload higher than with the generator, the highest being with forward kinematics.

Chapter VII

Conclusion

1 Introduction

The work described in this thesis was performed when I started to work on a new generation of animation system. The initial goal was to use Interactive Genetic Algorithms (IGAs) to animate articulated figures.

The initial research was unsuccessful but enabled me to understand how such systems work and how their power can harnessed.

It became clear, however, that an IGA would never be capable of producing useful animations of an articulated figure (interesting animations could easily be produced) but the user had virtually no control over what was produced. This was mainly due to the overwhelming size of the space of possible animations.

The solution to this problem became to subdivide the animation system in three distinct parts. The first part was dedicated to the production of poses which could then be used in a key-framing animation system. This thesis was devoted to that first part.

2 Origin of the technique

When Richard Dawkins, a professor of zoology at Oxford wrote *The Blind Watchmaker* [Daw86], he also wrote a little program which would demonstrate the power of evolution. His program, which he called Biomorph, was used to produce forms made of small line segments. The number of line segments, their length and direction were defined by a structure similar to a chromosome. To make the program simpler, sexual reproduction was not used. Although this program did not use all the ingredients of evolution, complicate forms alike shapes of different animals were produced. Biomorph also managed to produce all the letters of the alphabet.

In 1991, Stephen Todd and William Latham [ST90, TL91] wrote Mutator, a program used to assist the artist to produce 3D images. Mutator is similar to Biomorph apart from the fact that the user can specify the mutation intensity and allows to make marriages. Forms which can marry are directly selected by the user. This is different from conventional genetic algorithms where the user has no control over the reproduction process.

During the same period of time, Karl Sims [Sim91] used genetic programming [Koz92, Koz94] with a Biomorph like interface to generate 3D plant structures, 2D abstract images, solid textures and abstract animations, all appealing to the human eye.

My research originated from this work and we were optimistic in be able to extend the applicability of this type of tool to animate articulated figures. However, as I previously mentioned, this did not work out and I focused instead on the production of poses.

Again, it quickly became clear that producing poses using these tools was hard. I concluded that four conditions have to be fulfilled before one chooses to use IGAs in an application:

- The size of the space to search should not be too big
- The user should be good at grading one particular solution
- The user should not know what makes a good solution
- The user should not look for an accurate target solution

Since these conditions were not fulfilled, it was not a surprise that that actual positioning system was not satisfactory.

I realised there was a much faster and efficient way to produce poses. The solution was to let the user directly selects useful joint configurations to produce a special pose (the seed pose), which could then be mutated to produce another set of poses which similarity to the original pose depended upon a mutation intensity specified by the user.

The thesis was devoted to the study of this innovative and powerful concept.

3 Design of the articulated figure

Before extending to the study of the technique itself, I need to talk about the articulated figure.

First, it is a 3D character made of rigid limbs each connected by means of joints. It is represented internally using an n-tree, that is a parent limb can have many joints/limbs or none. A few articulated figures were implemented, but all of the research work was performed using a humanoid.

Limbs are rigid, that is they may move but they are not allowed to change of geometry. Thus squash and stretch movements cannot not be simulated.

Given a limb connected at a joint, there are only three possible degrees of freedom (DOFs). These are the three possible rotations at the joint at which the limb originates. The root limb (the hip for the humanoid) has six DOFs, the same three rotations plus translations along three orthogonal axes.

Different notations have been used to specify the rotation axis. The notation which was chosen here is described by Thalmann &al. [NTD88] and relies on the user to specify the main axis of the limb plus a flexion axis. The pivot axis is automatically deduced. This notation has the advantage of being more meaningful and thus more understandable than other common notations. To facilitate the implementation, axis can only either be the X, Y or Z axis.

Some limbs may rotate around the three orthogonal axis. Some may rotate around just some of them whereas some may not be able to rotate at all. The amplitude of the rotation also varies from one limb to the other, from one individual to the other and even from one stance to another. To simulate this complex process would be impossible and entirely useless for our purposes since we are not interested in simulation but only in animation. Instead, DOFs for each rotation axis can be specified by the user when the robot is first built.

To represent the articulated figure, there was a choice of many techniques. Since accelerated hardware which allows fast rendering was not available at the time the research was carried out, volumes (cones, cubes, cylinders and spheres) were chosen instead. They can be rendered at a small computation cost whilst still preserving the shape of the robot. An innovative algorithm to render cylinders and cones was implemented so that rendering of the articulated figure allows interactive work on it.

4 Generator

During this research I developed an application which I called Generator. A definition was provided:

> The generator is an evolutionary technique for which genes are clearly identifiable by the user and the cross-over process (i.e. the reproduction process) is explicitly performed by the user. Mutation is then applied to produce a new population of individuals.

Three rules were also stated to work out if the generator can be used for a particular problem:

- Genes can be made clearly identifiable to the user
- Particular values for these genes can be made easily selectable by the user

❒ There should not be too many such genes

The interface is similar to the interface of an IGA. Basically, the computer generates poses which are then displayed in their own window. To get a better understanding of what has been generated, the figure is also shown from the top and left sides.

Unlike typical IGAs, the user does not specify an objective value relative to some kind of "goodness" of the pose, but directly selects joint configurations which are of interest. This selection is then used to produce a seed pose which is displayed on its separate window.

When the user is satisfied with the pose which has been produced, it can be mutated to produce another set of poses. New selections will ensure this process will eventually converge towards a target pose.

Flexion&pivot and twists rotation types are searched through separately. Since the formers define the main characteristics of the pose they are typically searched through first. A seed pose is produced for the second generation which is generally used to get any twist rotations which may be required. A third and maybe a fourth generation may be necessary to fine tune the result.

To allow for fast and efficient covering of the space of solutions, a hyper tessellated scheme was used. When the flexion&pivot rotations space is being searched through, a tessellated hyper-sphere is used. When the flexion or twist rotation space are searched through, a tessellated hyper-sphere is being used.

The process of generating a new set of poses works in two separate passes.

During the first pass, a set of alternatives (a point on the tessellated hyper-sphere or hyper-circle) is selected. This process first uses the coarsest tessellation(decomposition) level and goes on using finer tessellation levels until at least a given number (the number of poses displayed) have been retrieved. This automatically sorts the alternative by order of importance. It was noticed that the coarsest tessellation level an alternative belongs to, the more likely it is to be used to build a pose. An alternative is retrieved if it is valid, that is, if does not break any constraints (DOFs, distance from seed configuration). Since all valid alternatives of the current level of tessellation is retrieved, it is likely that a number of alternatives exceeding the required number has been selected. To be able to visualise these alternatives, the concept of pages was brought in. Basically, an alternative will not be displayed twice if there are some valid alternatives which have not been displayed yet. So, by viewing the second and the third page of poses, the user can see some of the alternatives which were selected by the computer but which were not shown on the first page.

During the second pass, the computer retrieves each alternative in turn, from the first to the last. Because alternatives have been sorted in the previous pass, they are also displayed sorted. Because the articulated figure is symmetric, resulting poses also look symmetric. This is of great help to the user as patterns naturally appears. It considerably diminish the time required to produce a pose. Once an alternative has been used, it is marked so that it cannot be displayed again until all selected

alternatives have been displayed. Thus it was possible to implement the concept of pages where joint configurations which did not appear on the first set(page) of poses would appear on the second or the third one.

The articulated figure which is initially described by the user is used to produce conventional animations. In some cases, it may happen that the user will want to do more than what is allowable. In other words, the user will want to break the constraints that were initially set. So, to solve this problem, DOFs were only enforced at the first generation. Further generations will ignore DOFs and thus constraints imposed by them may break. It was felt that it was the best solution versus allowing for difficult or even impossible motions if constraints were not strong enough.

To speed up the use of the interface, some commands of the generator (mutate, initialise, etc) were mapped onto hot keys.

There are two ways that the user can select a limb: it can be clicked on or a rectangle can drawn around it. All limbs entirely included in the rectangle are selected.

5 Implementation of common techniques

5.1 Forward kinematics

To evaluate the power, or lack of it, in the generator, it was decided to implement techniques commonly used to pose articulated figures like our humanoid.

The first and most simple of these techniques is referred to as forward kinematics. Widgets similar to the one used in LifeForm [Mac] were implemented so that two DOFs (flexion&pivot) could be handled at the same time. These were called joint balls. When testing with two other types of interface, we came to the conclusion that this one was by far the most powerful.

When a joint is selected, flexion&pivot use one joint ball and twist uses the other. If both DOFs can be moved (flexion&pivot), the widget uses a graphical representation of a sphere. Grabbing a point and moving it on the sphere will make the selected limb rotate accordingly. If only one DOF can move (flexion or twist), a circle is used instead. The principle stays the same. If no DOF is selected, the joint ball is drawn as an empty circle and the corresponding limb cannot be moved.

5.2 Inverse kinematics

Inverse kinematics is a complex and still a slow problem to solve. To allow for better interactivity than conventional techniques, an innovative algorithm was implemented.

It more or less originated from Korein [KB82, Kor82, JU85] which used the concept of workspaces to rotate limbs at the beginning of the kinematic chain just enough

so that the problem left by the fully constrained kinematic sub-chain can be solved analytically.

With the new algorithm, the end-effector tries to achieve the goal by translating to it. The new starting coordinates of that limb are deduced. They become the new goal to achieve for its parent. The parent proceeds in a similar manner until it is already in the right position or the root of the tree has been reached. At all steps, constraints imposed by DOFs are checked. Consequently, the parent may not have reached the goal the current limb was asking for. As a result, when the parent has done what it could, the current limb takes over again and tries to perform the best it can.

To get the best of this new technique, digits were mapped as hot keys specifying how long the kinematic chain is. Thus pressing on the key numbered "2" tells the algorithm that the current kinematic chain only has two limbs/joints. Also, a frame correlation scheme was used to avoid the jerkiness inherent to this technique.

6 Evaluation

6.1 First part

After the generator was implemented, it was not obvious to see which of the techniques (Generator, forward kinematics and inverse kinematics) was the best. To determine the power or lack of it of the generator relative to the other conventional techniques, I decided to perform a proper evaluation

6.1.1 Preparation

Ideally, the evaluation would have been performed using expert users ,that is, animators experienced with the computer technology. However, this was not feasible so instead unskilled people but familiar the computer technology had to be used.

The evaluation involved comparing the generator against forward and inverse kinematics. To counterbalance order effects, four groups of participants had to be used. To get statistically significant results, eight participant per group would have been necessary. However I did manage to get only five people per group and this was virtually the minimal number of participants if any results were to be obtained.

Participants were IT students who had already spent one year in the Department and research students who did not know anything about my research, this in an effort to avoid all bias in favour of one or the other technique.

All participants were asked to produce three poses, these being the same for everyone. For the first pose, a training sheet was provided. The second pose was used to get confident with the technique and the third one was used for the measurements. The experiment was divided into two parts. The first one involved producing these

poses using one technique and the second part involved producing the same poses but using another technique. These poses are typical poses that anyone can relate to and were actually suggested to me when demonstrating the system. These poses were selected on the assumption that they are not easier to perform using one technique or the other. These poses and the degree of difficulty of the tasks was obtained from a pilot study and were made so that one experiment does not last more than an hour.

The computer was given the responsibility to determine when the user has reached a pose sufficiently similar to the target pose. A simple least square calculation was used for this purpose. The computer was also used to measure the time taken to complete each task.

A simplification of the NASA-TLX [Sta88], the RTLX (Raw-TLX) was used to determine the workload associated with each technique. After a task was completed, the participant had to fill in scales the purpose of which was to determine the workload of the task just completed. At the end of the experiment, another scale, used to specify the overall preference, was added.

6.1.2 Results

The related t-test was used to analyse the data from this evaluation. It resulted that, most of the time, differences in the means were not statistically significant due a too small population as expected from the small number of participants.

6.1.2.1 Generator versus forward kinematics: None of the differences between the means of the workload attributes of the generator and forward kinematics were significant. This came as a bit of a surprise though, because I expected the generator to be so difficult to become accustomed to.

However, participants preferred to use forward kinematics rather than the generator and this difference was statistically significant. Although participants managed to produce poses faster using the generator, this difference was not statistically significant.

6.1.2.2 Generator versus inverse kinematics: Participants found the physical demand and the time pressure associated with inverse kinematics more demanding than with the generator. These differences were statistically significant. Not surprisingly, the workload associated with inverse kinematics was also higher and this difference was again statistically significant.

Participants also produced poses faster using the generator than using inverse kinematics but this difference was not quite sufficient to be statistically significant.

6.2 Second part

Before the first experiment even took place, I was convinced the generator required a great deal of training, and therefore participants would perform badly using the generator because I could not afford to give them the necessary training.

Although participants did not perform as badly as I had assumed they would, I decided to perform another experiment with, this time, an expert user. Unfortunately, the only expert user available was myself so results can only be taken in terms of indication and not facts. However, I felt this work would not have been complete without this experiment

6.2.1 Preparation

During the first experiment, I noticed the implementation for inverse kinematics was not perfect and could be improved. This is sad as it entirely changes some of the implications of the evaluation. It is important to keep in mind that these evaluations are not performed on the techniques themselves but on implementations of these techniques. There are good and less good implementations. Mine are probably not perfect.

Inverse kinematics was improved in three different ways:

- Rotating the articulated figure is not necessary anymore since selection and positioning can also be performed using side views.
- The length of the kinematic chain can be specified using digits from the keyboard as hot keys
- A frame correlation scheme solves the jerkiness inherent to the technique.

The generator was also improved so that selection is copied in the seed pose like before but also in the other poses which the computer had generated. This was meant to diminish the mental workload inherently associated with the technique.

This time, poses could not be selected in advance. Instead, the computer was used to generate poses randomly. Each technique was then used to produce them. The computer was also used to determine when a pose was sufficiently close to the target. At each pose, the order of the techniques was changed to counterbalance the order effect.

The speed, the number of iterations and the number of page switch for the generator were logged for analysis.

To make sure statistically significant results would be obtained, forty poses were used during this evaluation.

Before the evaluation took place and since I had to be an expert user, I trained on about fifty poses.

6.2.2 Results

Since the related t-test can only be used when two set of scores have to be compared, the correlated ANOVA test was used instead. The ANOVA test is similar to the t-test except that it allows one to compare many set of scores at the same time.

Differences in speed were extremely significant, favouring the generator. Although I did expect differences in favour of the generator, I did not expect such big differences. Furthermore, it is interesting to notice there was a significant difference between the first and last ten poses produced during this evaluation with the generator. Such a difference also exists with the other techniques but it is not so significant. Comparing logs from the training session and the evaluation, we can see that all techniques improved but the average time was nearly cut by two with the generator which confirms that the training is of primary importance with the generator.

7 Conclusion

7.1 Advantages

Given sufficient training, these results indicate the generator will be a more effective technique than inverse and forward kinematics. However, as the articulated figure becomes more complicated, inverse kinematics will become more powerful as it is possible to position several limbs at once using this technique.

7.2 Disadvantages

However, I agree with the participants of the first evaluation: forward and inverse kinematics are a lot more entertaining tools to use. The generator is mentally demanding as it requires the users to stay aware of many things happening at the same time at different location on the screen. This is of major importance as eventually this type of tool will be used by professional animators who have to work many hours with this type of tool. The more interesting to use they are the better it is.

The results also highlight the importance of the training, a training which may be costly in a real world situation. Other characters were built and I personally preferred to use inverse kinematics to pose them. Once used to them, no further training is required. On the contrary, the generator requires training with each different character.

To finish, here follows a table summarising the results of the work presented in this thesis:

What was proven	❒ The generator allows expert users to produce poses faster than using other conventional techniques. ❒ The generator facilitates the making of "breath of life" effects by allowing users to easily produce poses similar to a given one.
What was not proven	❒ The generator does not facilitate the production of poses. The production of poses seems to require more cognitive effort when using the generator rather than when using other conventional techniques.
What might be the case	❒ All these techniques have their own advantages and disadvantages. The implementation of a hybrid system cumulating the advantages of all techniques might be the way to go for to implement a powerful positioning system.

8 Future work

8.1 A better evaluation

If one would decide to invest further effort in the generator, the first thing to do would be to verify the results obtained during the second evaluation. Thus, an evaluation would have to be performed with one or more expert users, and not the person carrying out the evaluation.

8.2 Improving the technique

Copying joint configurations in the seed pose as in the poses produced by the computer definitely eased the building process. The current implementation does not use true multi-tasking. Thus poses are rendered in the background. This causes problems such as an interface slow to react and even selection of wrong limb configurations which inflict a consequent penalty on the time spent producing a pose.

A true multi-tasking implementation should solve these problems. It would then be interesting to study the power of such an implementation.

8.3 The making of a professional positioning system

As recommended previously, the implementation of a hybrid positioning system which would combine in a single application the generator, inverse and forward kinematics might the ideal solution. All techniques have their own advantages and disadvantages. By building a system which would use all of them, it should be possible to avoid the disadvantages while preserving the advantages of each technique.

Furthermore, what professional animators want to be able to do is to lock or "rubber-band" bones in particular positions or orientations. They will also want to be able to produce symmetric poses. For example, they should be able to specify that both hands are separated by a given vector (usually the current vector).

Also, the generator could also be used in combination with inverse kinematics. That would be a two layers positioning system. The generator would use inverse kinematics to produce the different individuals. Users would select one or more end effectors while still being able to assign constraints to some bones (positional or directional constraints) and use the generator to find a new position or orientation for the end effector(s) in space. For each individual, the generator calls the inverse kinematic system by specifying a set of constrains and the inverse kinematic system produces the pose which fulfils these constraints.

8.4 The generator as a browser for poses

Key-framing animation systems usually have to rely on many key-frames to produce animations. These are usually stored in a simple database with directories containing other directories or poses.

The generator could be used to implement a new type of search tool. Basically, the interface would be similar to the current one apart from another window displaying poses from the database. As the user builds the seed pose, the computer searches through the database to retrieve poses bearing similarities with the seed pose. The poses displayed could also be used by the selection process. If the pose is not in the library, at the end, the pose which the computer should have found is the seed pose. Because it was not in the database, it can be added to it so that it will be found the next time it is being looked for again.

Even if the building mechanism is fast, this mechanism guarantees a pose will be produced at least as fast.

8.5 An IGA/Generator for animating faces

In 1995, Patrick Lambourne [Lam95] used an IGA to alter the shape of a face and even to produces impressions such as happiness, sadness, etc. The program was rather successful although it was not heavily tested. I recalled that for an IGA to work, four

conditions have to be fulfilled. These were only partly fulfilled but it appears to be sufficient.

The generator should perform better that this IGA though. Thus, the generator could also be used to produce faces as key-frames and then try to animate them.

In conclusion, the generator could be used anywhere attributes defining a solution displayed on a screen are clearly identifiable and selectable. A seed can then easily be produced and evolved towards a target. The user has to have a clear understanding of this attribute and how it interacts with other attributes to produce the result displayed on the screen. If this is the case, there is no need to rely on reproduction to find this attribute as a selection scheme will do the job more effectively and much faster.

Appendix A

Advanced topics for genetic algorithms

1 Introduction

Genetic algorithms are a family of computational models inspired from natural evolution. Specific solutions are encoded using chromosome like data-structures. Recombination operators are applied to assemble together useful information.

2 Data structures

2.1 Chromosome

Our genetic patrimony is made of long chains of DNA, each one called a chromosome. Chromosomes are made of genes. A chromosome encodes one individual or solution. In this work, chromosomes were used to encode poses.

2.2 Genotype

Human beings have 21 chromosomes. This is called the genotype. With genetic algorithms, usually only one chromosome is used to encode an individual. Therefore, most of the time, the genotype and the chromosome are equivalent.

2.3 Phenotype

A chromosome encodes only one individual or solution. The individual, that is the result of the chromosome, is called the phenotype. In this work, phenotypes were poses.

2.4 Gene

A gene encodes one particular feature of the solution. In this work, a gene was used to encode the configuration at one joint. Genes from each chromosome were used to encode each joint configuration. So a chromosome encoded a whole pose.

2.5 Allele

The value of one gene is called the allele. The gene is like a variable in a program and the allele is its value. In this work, the allele is one flexion&pivot and one twist rotation.

2.6 Population

Genetic algorithms works on several chromosomes at the same time. At any one time, there is always a set of chromosome which is being worked on. This is the population.

3 Generation

The goal of a genetic algorithm is to evolve the population of chromosomes to find the ones which will encode the best solutions. This is achieved by producing a population of chromosomes and testing it. This process is performed several times. Each such iteration is called a generation.

4 Reproduction

At each generation, chromosomes of the current population are used to produce the next population of chromosomes. Good chromosomes are recombined together in the hope that better chromosomes will be produced. This is called reproduction. This process is performed using reproduction operators. Genetic algorithms may use several such operators.

5 Crossovers operators

The main recombination/reproduction operators are called crossover operators. There are several versions of this type of operator.

5.1 One-point crossover

The one-point crossover operator uses two chromosomes. These are usually of the same length although it does not have to be the case. A cutting point is chosen and two new chromosomes are generated. The first one is made of the first part of the parent chromosome plus the second part of the second chromosome and vice-versa for the second chromosome.

5.2 Two-point crossover

The previous operator is limited to only one crossing site. As a result, some of the features of the chromosomes cannot be recombined effectively ([Dav91b],page 47). The first solution to this problem is to use a two-point crossover (Fig. A.1) in which two crossing sites have to be chosen instead of one. In fact, by using a circular representation, the one-point crossover operator then becomes a special case of two-point crossover. Hence, the two-point crossover is a powerful generalisation of the one-point crossover. The two-point crossover operator also seems to be more powerful than the n-point crossover where $n > 2$.

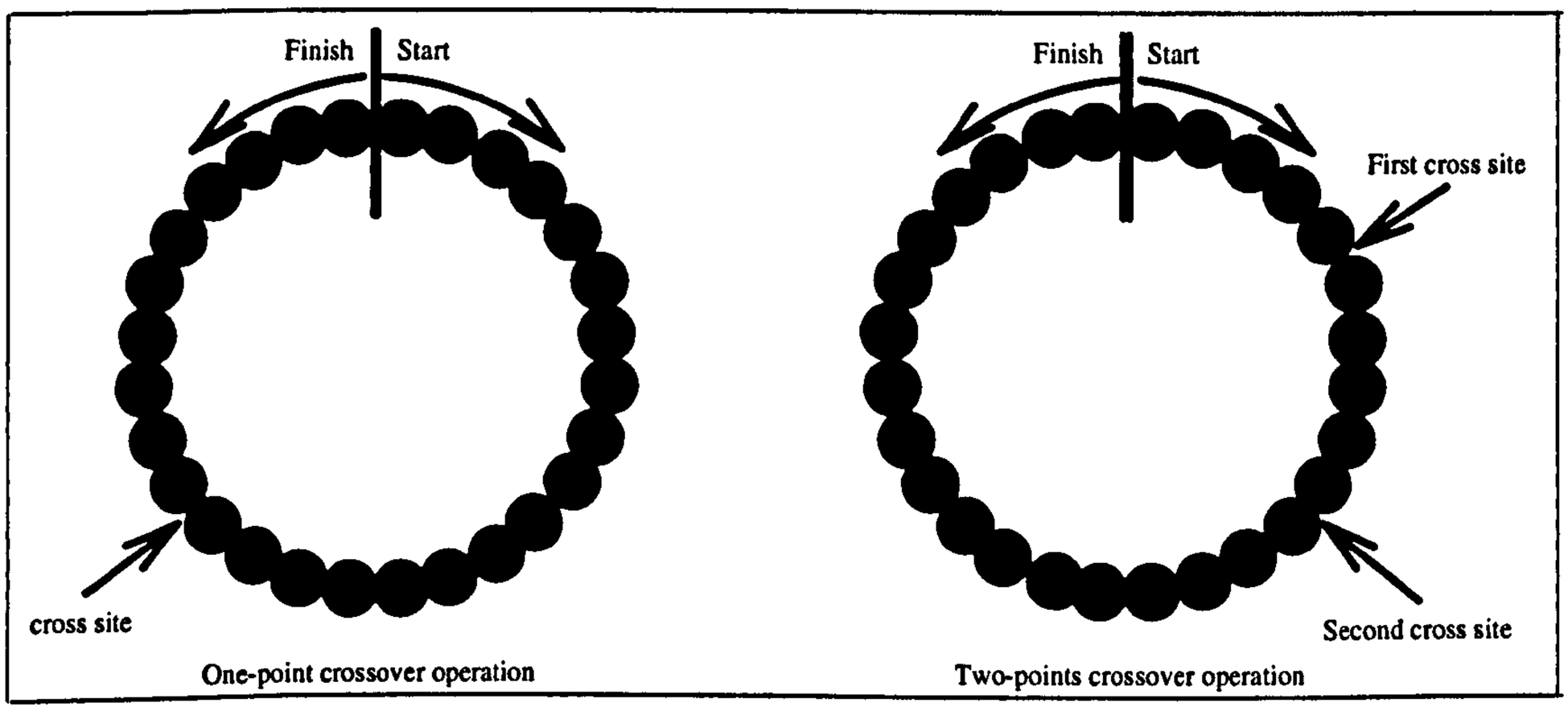


Figure A.1: The two-points crossover operator

This example shows first the one-point crossover and then the two-points crossover. As can be seen from this figure, the two-points crossover is only a generalisation of the one-point crossover.

5.3 Uniform crossover

The two-point crossover produces good results but unfortunately has some limitations. One solution is to use the uniform crossover instead. With this operator, for

each bit position of the two off-springs, a random selection is made to decide which parent contributes its bit value for which child. A crossover mask [DBM93], computed randomly, can be used to get the same results. If there is a one to one position in the crossover mask, then the corresponding gene value of the first parent is copied into the corresponding gene of the first child otherwise it is copied into the corresponding gene of the second child. e.g.:

Crossover mask	0	1	1	1	0	0	1
First parent	1	1	0	0	1	0	1
Second parent	0	1	0	1	0	1	0
First child	0	1	0	0	0	1	1
Second child	1	1	0	1	1	0	0

5.4 The blended crossover

The blended crossover operator is application-dependent but, due to the fact that it has been used successfully in the field of computer animation and particularly path motion optimisation [MP94], I will describe it here. This operator only makes sense if integer coding (also referred to as gray-encoding) is used. This different coding will be explained in detail later on. Using integer encoding, genes can have several values instead of two for the binary coding. Using one-point and two-point crossover operators, changes can produce discontinuities in some applications. One solution might be to avoid choosing crossing sites which are likely to contain discontinuities. Unfortunately, such requirements may not preserve continuity all the time. A better solution is to use the blended crossover which will avoid sudden changes in the genes (Fig. A.2). Briefly, changes are moderately propagated using a blending scheme (the further apart the genes are the smaller the changes) to smooth out changes occurring at one point.

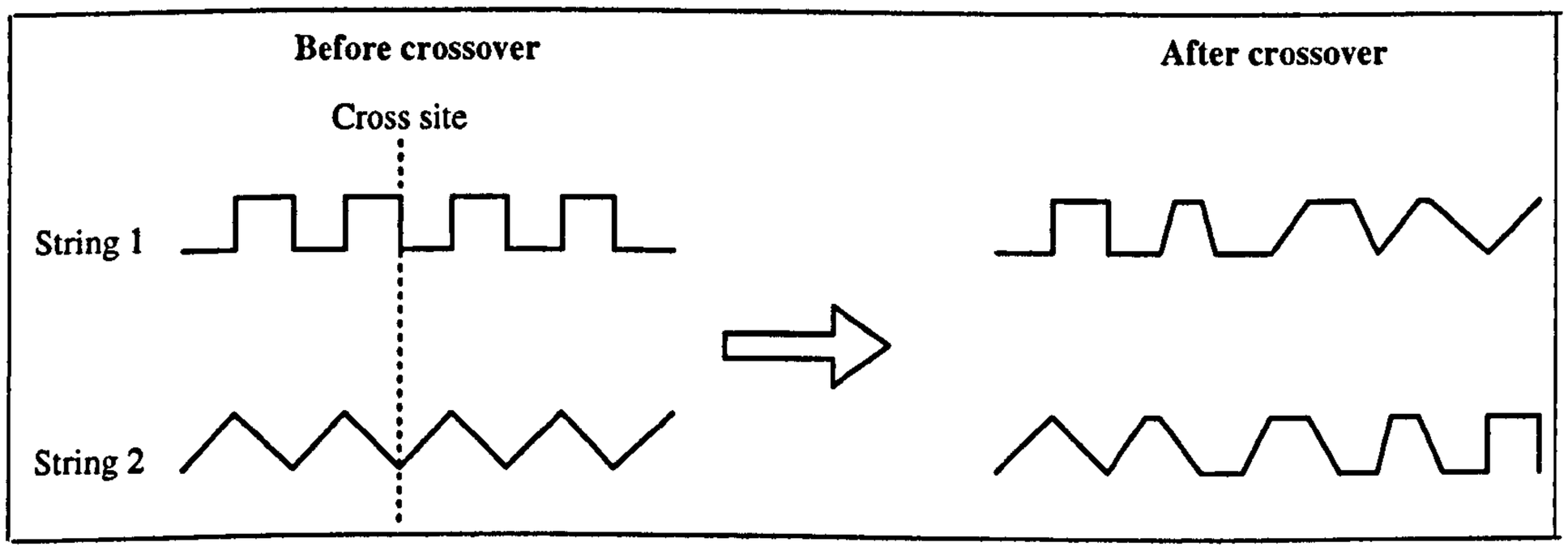


Figure A.2: The blended crossover

This figure shows the blended crossover in actions. The changes are slightly propagated around to diminish the disruptive effects of the usual crossover operators.

6 Other operators

Previous operators have biological origins. They are robust and have been used successfully in many areas. Nevertheless, in some applications, hybrid operators have proved to be more useful. Although less robust than conventional operators, they are general enough and have been used in many applications. Some of them are described in what follows.

6.1 Non-biological crossover operators

6.1.1 The analogous crossover operator

The main areas of applicability for the analogous crossover operator are robotics, path planning and computer animation[1]. When using the one-point or two-point crossover operators, one or more cross site positions have to be chosen. These are chosen randomly and independently of their corresponding genotype character. This is inconvenient in computer animation in which the use of such operators can bring discontinuities in the motion produced. The analogous crossover operator selects the crossing sites based on the similarity of the genotype character at this particular cross site. In other words, the crossing site will be chosen depending upon the similarity the genes of each chromosomes have at that particular position. Thus the closer one gene of the first chromosome is to the corresponding gene on the other chromosome the higher the probability that the gene position is to be selected as a crossing site (Fig. A.3).

6.1.2 The segregation crossover operator

The segregation crossover operator also provides some nice features for computer animation. It is only relevant when the genes composing the chromosomes are homogeneous and multi-valued. The usual crossover operator will use the same cross site for both chromosomes. With the segregation crossover operator, a first cross site is selected randomly. The alleles of each gene in the second chromosome are then compared against the alleles of the selected gene to find the nearest one. The locus of the genes with the nearest allele is chosen to be the second cross site (Fig. A.4). This operator has the capability of destroying the organisation within the genes and to eliminate poor alleles so that they will have no chance to come back to life later.

6.2 The inversion operator

In all genetics applications, the ordering of the genes within chromosomes is of prime importance. GAs works by recombining into a single chromosome the good

[1]See [Dav91a] page 78 for a detailed analysis of this operator

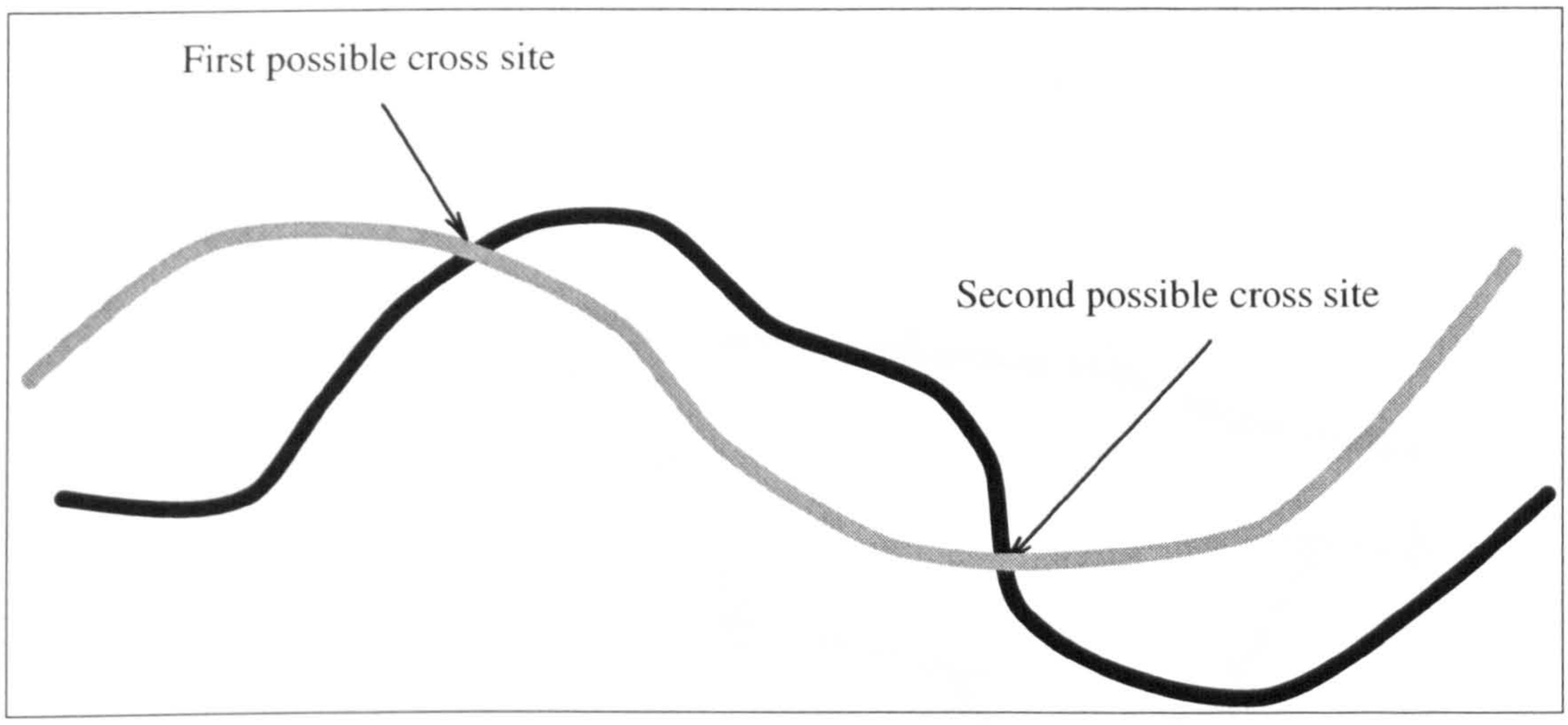


Figure A.3: The analogous crossover

With the analogous crossover, the cross sites are chosen dependently to the genotype character of the two chromosomes to mate. This may avoid some of the disruptive effects of the usual crossover operators.

features of two chromosomes. These good features are schemata[2]. Quite often these schemata do not belong to the same block (otherwise we would have a true building-block). Therefore it is fairly possible that they may be destroyed in further recombinations of different chromosomes. To avoid this inconvenience, reordering of the gene locus can be used to try to find the best ordering which will preserve these schemata. These operators also exist in nature where they are used to guide the search for better codings of the chromosomes.

The simplest of these operators is called the inversion operator. With this operator, two different genes are chosen randomly on a single chromosome. The block of genes between these two genes (including these genes as well) is then simply cut, reversed and pasted back again e.g.:

$$1\ 2\ \underset{\frown}{3}\ 4\ 5\ \underset{\frown}{6}\ 7$$

$$1\ 2\ 6\ 5\ 4\ 3\ 7$$

A thorough review of the different reordering operators was made by Goldberg ([E.G89], page 166).

This operator does not produce a new chromosome. It is still the same chromosome but ordered in a different manner. All the genes with this chromosome have kept their

[2]If schema contains only 0 or 1 (for a binary alphabet), then it is called a building block.

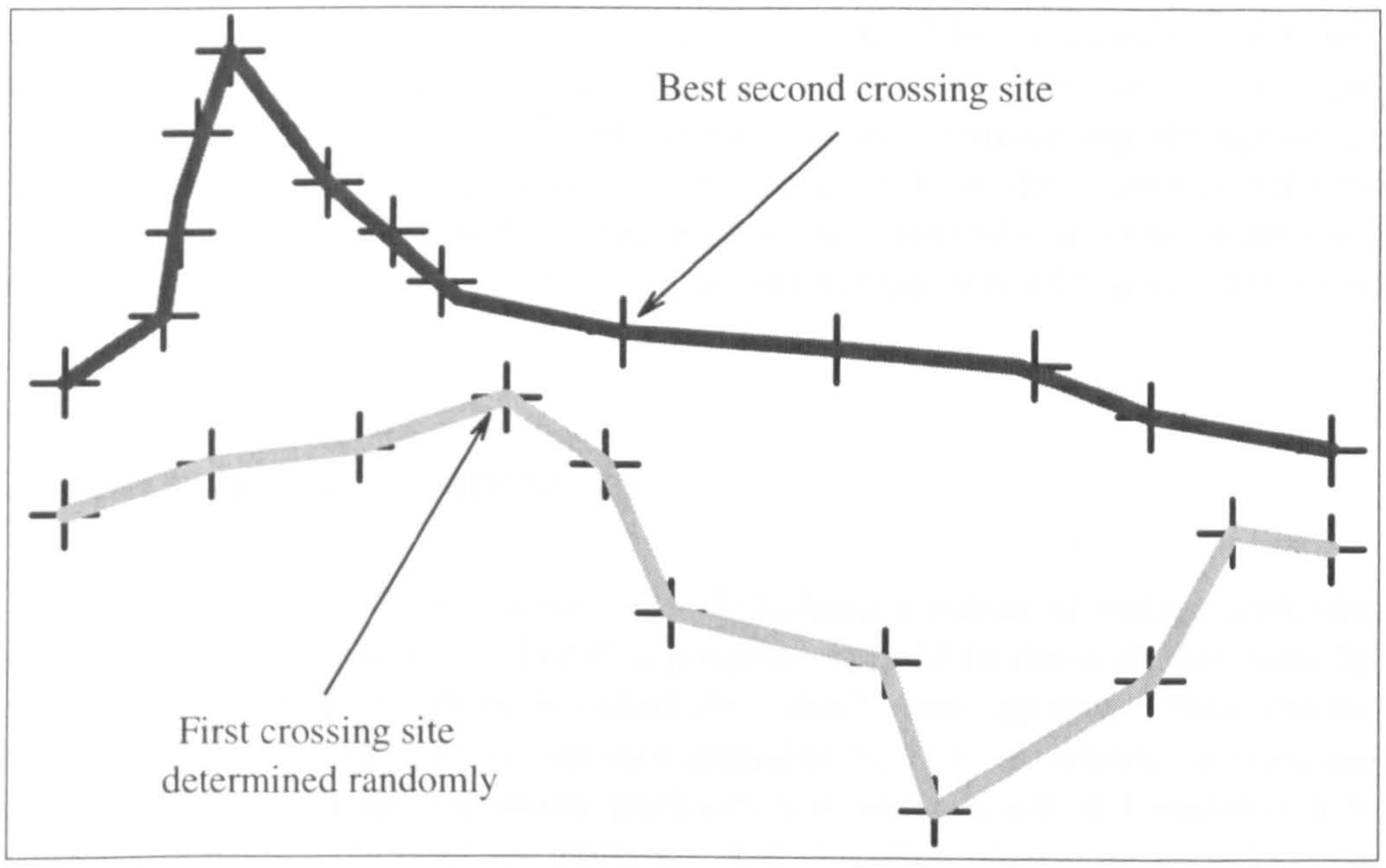


Figure A.4: The segregation crossover operator

The segregation crossover operator uses the same philosophy of the analogous crossover operator. But since it does not take into account the locus of the cross sites, it is more powerful than the other one. So cross sites with different locus may be obtained. If they are identical, it then results in a pure analogous crossover operation.

initial values but their positions have changed. The sole purpose of this reorganisation is to reset interesting schemata in a form which will make them difficult to be destroyed.

6.3 The addition and deletion operators

In most applications, the length of the chromosomes is specified at the beginning and cannot change during the search of the optimum solution. Nevertheless, in some other applications and particularly in computer animation, the length of the chromosomes can evolve whilst the search progresses. In some of these applications, it is even absolutely essential. To deal with this requirement two operators (the addition and deletion operators) were devised ([Dav91a], page 84). The mechanisms of these operators are very simple. The deletion operator simply selects randomly a gene to delete and the addition randomly selects a gene position where to add the new gene. Several policies can be used to choose the value of the new gene but only three of them are really interesting. These are the random, the duplication and the related schemes. With the random policy, the new allele is chosen completely randomly amid the set of possible alleles. With the duplication policy, the new allele is either duplicated from the previous or the next gene. With the related scheme, the mean of the previous and

of the next allele becomes the allele of the new gene. This interpolation scheme has nice properties specially in computer animation due to the fact that it permits the tuning of the search towards the finest details without bringing any disruption in the current motion path. More generally, it is also clear that this operator only makes sense if the allele-alphabet contains more than two elements, in other words it only makes sense if the GA does not deal with binary strings but with q-ary strings where $q > 2$.

7 The schema hypothesis

A schema [Hol75, E.G89] is a template describing a subset of strings with similarities at certain string positions. For this purpose, we add to the alphabet used by the allele another symbol (*) which is called the "don't care" symbol. Thus the binary alphabet is transformed into the ternary alphabet 0, 1, $*$. A schema is then said to match a particular string if at every position, a 0 matches a 0, a 1 matches a 1 and a $*$ matches either. Thus for an alphabet of k elements, there are $(k+1)^l$ possible schemata where l represents the length of the string.

It seems a bit surprising to want to augment the number of available possibilities (e.g. for a binary string of length 5, only $2^5 = 32$ solution alternatives exist and $3^5 = 243$ schemata are possible) but it facilitates the understanding of how GAs work and helps to determine the speed at which a GA can converge. For a given string of length l and an alphabet containing k elements, only k^l are possible. So for a population of n strings, there are between k^l and nk^l possible different schemata. What is interesting here is to compute the number of schemata which can be processed usefully in one generation.

The most widely quoted result here is Holland's estimate of $O(n^3)$ schemata usefully processed in a single generation. Simply stated, that means that despite the processing of only n strings, something like n^3 different schemata will be processed in a single generation. This result is so important that Holland called it *implicit parallelism*. However special care has to be taken about this result. To see where the problems are, the conclusive equation used by Goldberg [E.G89] is used.

$$n_s = \frac{(l - l_s + 1)n^3}{4} \tag{A.1}$$

where

n_s is the number of schemata s processed in a single generation
l is the length of the strings
l_s is the length of the schemata s
n is the size of the population

As can be seen from this equation, the result is first dependent on the length of the schema desired. It is also dependent upon the length of the string such as the

longer the schema is the smaller the number of successfully processed schemata is. This is quite normal and is due to the disruptive effects of the crossover and mutation operators. The longer the schema is the more likely it will be disrupted by one of these operators. Thirdly and more importantly is the role played by the size of the population. In fact the value of this parameter dominates the final result and this can be seen seen in the above equation. For the purpose of demonstration, Goldberg set the population equal to $k^{\frac{l_s}{2}}$ where k is the number of solutions in the alphabet. In conclusion, if this population is different the results may also be different but it seems quite difficult to guess what they could be. Finally, this is based on the assumption that a large number of different schemata are in the initial population and is largely true only at the first generation. After the first generation the population has already evolved in such a way that a great deal of unfit schemata have already disappeared permanently.

So this result suffers from a lot a problems which makes a large community of researchers dubious of its real efficiency and consequences. This equation states that something like $O(n^3)$ schemata are processed at each generation but what the users of GAs are really looking for is not a maximum of schemata to be processed in a single generation but the number of generations which will be necessary for a given problem before reaching the point of convergence. Is it fair to assume that by evaluating n solutions, we in fact implicitly evaluate n^3 solution alternatives? The answer to this question is not clear and is fairly problem dependent. Due to the lack of confidence I had in this result, I compared several implementations of GAs in different domains and computed the number of evaluations of solution alternatives they took before reaching the point of convergence. Surprisingly, it seemed that less evaluations were needed than this equation indicated. In other words, n evaluations of solution alternatives seemed to implicitly evaluate more than n^3 solution alternative. However, special care has to be taken with this observation [Dav91b]. First they were nearly all hybrid implementations of GA and they used large populations.

8 The epistasis problem

The epistasis problem is an advanced topic in genetic algorithm research but is of great importance to computer animation.

In biology, epistasis refers to the "masking" or "switching" effect among genes. A biology textbook says [HB93]:

> A gene is said to be epistatic when its presence suppresses the effect of a gene at another locus. Epistatic genes are sometimes called *inhibiting genes* because of their effect on other genes which are described as hypostatic.

In the FAQ of genetic algorithms [HB93], they define the epistasis problem seen from the artificial genetic research side:

When evolutionary computation (EC) researchers use the term epistasis, they are generally referring to any kind of strong interaction among genes, not just masking effects. A possible definition is:

Epistasis is the interaction between different genes in a chromosome. It is the extent to which the contribution to fitness of one gene depends on the values of other genes. Problems with little or no epistasis are trivial to solve (hill-climbing is sufficient). But highly epistatic problems are difficult to solve, even for GAs. High epistasis means that building blocks cannot form, and there will be *deception.*

When a chromosome is said to have a high epistasis, this means that many of its genes are dependent on other genes. A problem is said to be *deceptive* when small building blocks may give high fitness but their combination in a single chromosome is likely to result in rather a small fitness. These building blocks do not belong to the global optimum but just to some local ones. Combining one with another which does not belong to the same local optimum will result into another point in the search space which is likely to be worse than the two previous points.

For a normal GA, the detection of an epistasis problem is elusive because its effects can only be detected at the phenotypic level. The epistasis problem tends to use very long building blocks which makes the improvements on the fitness function difficult to obtain. A problem with no epistasis is a uniform problem suitable to be solved by a hill-climbing search method whereas a problem with a very high epistasis can only be solved by random search method. A problem with the mild epistasis is suitable for GAs. Therefore if a given problem has to much epistasis, it is convenient to work out a representation which will transform the original problem into another problem with mild epistasis [3].

A problem with a high epistasis is sometimes referred as a *GA-Hard problem.*

9 GA with small populations

Interactive Genetic Algorithms (IGAs) use very small populations. This may lead to many problems such as premature convergence or getting stuck into a local optimum [ree93].

The first condition that all populations have to meet is that every possible solution alternative in the search space is reachable, from the original population, only by crossover. By including the mutation operator, it is theoretically possible to reach any point in the search space but the search becomes more random and thus slower. The condition to be able to fulfill the first statement is that at least one instance of each allele exists at every locus in the whole population. In other words, if an allele does not appear at particular locus within a given population, then a subset of the space of solution alternatives will never be explored at all. Reeves then tried to compute the

[3] See in Davidor's book [Dav91a] page 33, 120 and 141

probability that at least one instance of each allele will appear at every locus in the whole population. He found that:

$$P^* = \left\{ \frac{q! S(M,q)}{q^M} \right\}^L \tag{A.2}$$

where

q is the size of the alphabet
M is the size of the population
L is the length of the strings
$S(M,q)$ is called the *Stirling number of the second kind* and can be written:

$$S(M+1,q) = S(M,q-1) + qS(M,q) \tag{A.3}$$

with $M \leq 1$, $q \leq 2$ and $S(M,1) = 1 \forall M$.

The implications of this result are somewhat startling. It means that for a binary alphabet, in a randomly generated population of a relatively small size, we can be virtually certain to have an instance of each allele at every locus. However for q-ary alphabet where $q > 2$, minimal population sizes become substantial even for short strings. For instance, for a confidence of 95%[4], the size of the population using an alphabet of size 8 and strings of length 10, is about 60 whereas the size of the population using binary alphabet and strings of length 30 $8^10 = (2^3)^10 = 2^30$ is only around 10.

The result appears to contradict the common belief that integer coding would produce better results and furthermore faster. This was not really unfounded. It was mainly based on the fact that most of the current implementations of the genetic algorithms use hybrid methods and in particular are using integer coding. This assertion seems to prove it is clearly a mistake. A recent article argues that a high-cardinality alphabet allow to sample more schemata but this is in fact at the cost of a much larger population.

The previous part has tried to compute the best population size to ensure a correct exploration of the space of solution alternatives. This has to be done for a randomly generated population. But if the population is deterministically generated, we can then be sure that it meets the primary conditions. In other words, we can ensure that all the alleles are represented at each locus and that a correct number of schemata are represented in the population. A technique had to be found which would automatically build a first population which would span correctly the space of solution alternatives [Smi93]. In other words, we might expect that the distance between each individual is nearly the same. This problem is a common problem of digital communication engineering which is solved by the use of error-detecting codes. These codes

[4]We want to be sure at 95% that all the instances of each allele are represented at every locus.

imply that all their elements are equally spaced. Some statistics have been generated for this paper and it appears that on average there are at least 20% more schemata of mid-order in systematically generated populations than in randomly generated populations. Nevertheless, these experiments were only based on binary strings. The experiments with q-ary strings where $q > 2$ were still in progress and whereas no results were provided for these alphabets they suggested that the advantages of a such mechanism should be even more beneficial.

Appendix B

Fast cylinders

1 Introduction

To represent a cylinder on a screen, we need the coordinates (P_1, P_2) of the segment representing the centre of the cylinder and also its radius R (Fig. B.1).

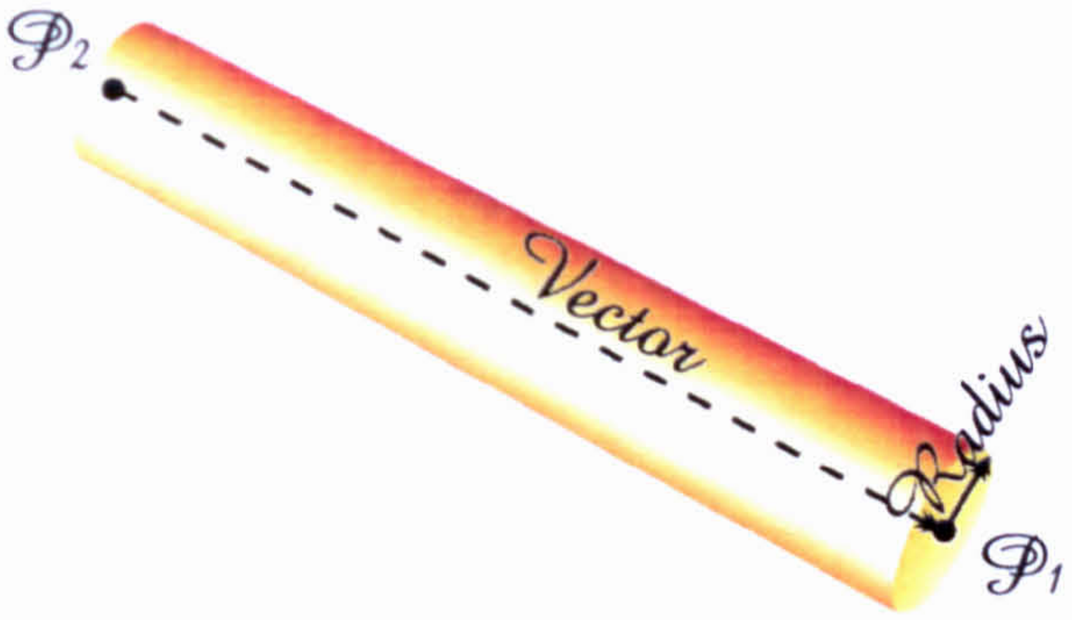


Figure B.1: Attributes describing a cylinder

To describe a cylinder, it is necessary to specify points P_1 and P_2 and the colour of the object.

The algorithm described here assumes an orthographic projection is used. Unrealistic images might be produced if perspective projection is used. Nevertheless, one may use a perspective projection on the condition that dimensions of the cylinder are scaled according to the camera position and state. Results would probably be good enough for cylinders distant enough to the camera. Otherwise, visual artifacts will become apparent.

When drawing a cylinder on the screen, three cases may occur (P_1 is at one end and P_2 is at the other):

1. P_1 is right in front of P_2. That is $P_{1x} = P_{2x}$, $P_{1y} = P_{2y}$ and $P_{1z} < P_{2z}$. The result is the same if P_2 is in front of P_1. Only one disc has to be drawn to

represent the whole cylinder. This disc will be located in P_1 or P_2 (depending on the Z coordinates) and will have R as radius.

2. P_1 and P_2 belongs to the same XY plane, that is $P_{1z} = P_{2z}$. Thus both sides are invisible. A single rectangle is used to represented the whole cylinder.

3. When it is neither of two previous cases, the side being the nearest to the observer is drawn, the other one being hidden. The side is represented by an ellipsoidal disc. A surface having the shape of the ellipsoidal disc is also swept along the cylinder.

2 First case: only one side is visible

Since a single disc has t be drawn, the resulting picture is simple to compute. The colour is computed first and the Bresenham's circle algorithm ([JFH90], page 81 and 99) is used to obtain the points on the perimeter of the disc. Since up to four points (two points for two scan lines) are computed at each iteration, the filling procedure is straightforward.

3 Second case: only the main body is visible

The resulting image is a bit trickier to obtain. First, the cylinder may point in any direction with the constraint that $P_{1z} = P_{2z}$ (Fig. B.2).

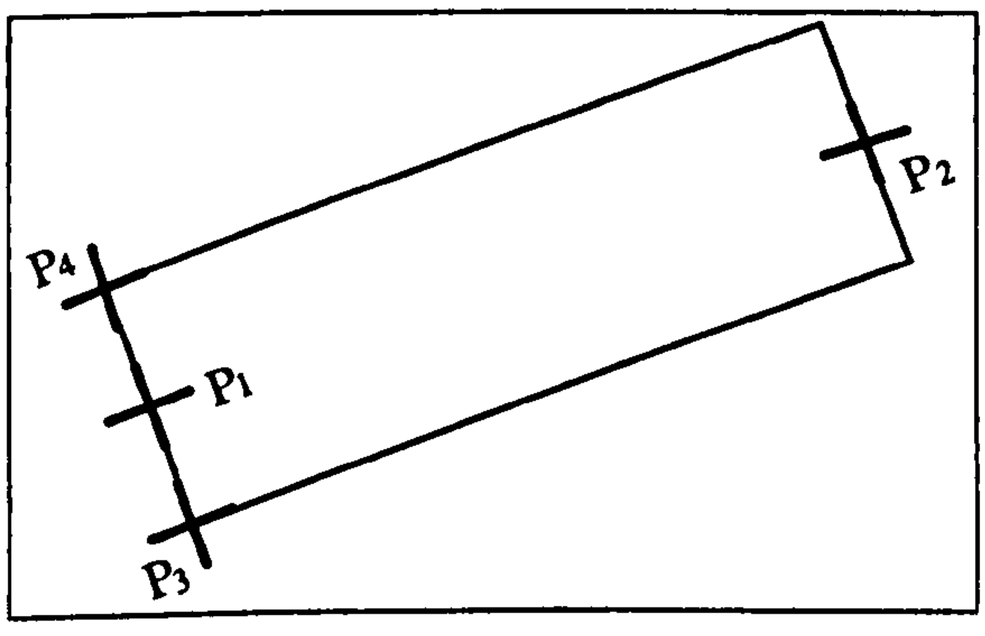


Figure B.2: Only the sweep surface is visible

The example shows a case where no side is visible. In this case, only the sweep surface has to be drawn.

P_3 and P_4 are first computed. Then, the Bresenham's line algorithm [JFH90], page 74) is used to display the sweep surface.

Let say that u is the normalised vector of $\overrightarrow{P_1P_2}$. Then

$$|\vec{u}| = (x, y)$$

So we have:

$$P_3 = \frac{r}{2}(-y, x) \tag{B.1}$$

$$P_4 = \frac{r}{2}(y, -x) \tag{B.2}$$

In the implementation, divisions by two will be replaced the much faster shifting operations. Since, all points lie on the same xy plane, their z component can simply be ignored.

Then, $\vec{u}$ is computed:

$$\vec{u} = \frac{\overrightarrow{P_1P_2}}{|\overrightarrow{P_1P_2}|}$$

and

$$|\overrightarrow{P_1P_2}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Computing the latter expression each time would be costly for a fast algorithm due to the square root function involved. Since it corresponds to the length of the cylinder which is known when the cylinder is created, it can be computed and stored with the data structure describing that cylinder.

To display the cylinder, the Bresenham's line algorithm could be used twice at each iteration first to compute the position of each point on the line segment P_3P_4 and again to sweep each point of the line segment P_3P_4 along the cylinder. A closer look to the algorithm reveals that the Bresenham's algorithm is called several times with exactly the same parameters. A better implementation would use the Bresenham's algorithm only twice in all, once for the points on the line segment P_3P_4 and once again to sweep these points along the cylinder. Thus, every points between P_3 and P_4 will have to be stored in a dedicated array. The structure associated with each point in this array will be made of the current x and y coordinates plus the colour of the pixel. If a z-buffer is being used, the z value for each point is also required. However, a problem remains. When displaying the surface, holes may appear each time the line algorithm changes both in x and y. The solution is to forbid such a thing by displaying the pixel each time either the value in x or y has changed. Thus, generated lines must be 6-connected.

This solution is still not complete though since the Z coordinate of each point on P_3P_4 still needs to be computed. This is achieved using the equation of a sphere:

$$X^2 + Y^2 + Z^2 = R^2 \tag{B.3}$$

Where

$X = x - x_c$
$Y = y - y_c$
$Z = z - z_c$
and $C(x_c, y_c, z_c)$ is either P_1 or P_2

Patterson [Pat93] used forward differencing to compute as quickly as possible each values of z.

Since the values for X,Y and r are known for each pixel, two possible solutions result for Z. These are:

$$Z = \sqrt{R^2 - X^2 - Y^2}$$

and

$$Z = -\sqrt{R^2 - X^2 - Y^2}$$

The problem here is that a square root operation is involved. Since this penalty would be too costly, the solution adopted here, which was proposed by Fuchs [FGH+85] and used successfully by Patterson, is to replace the square root operation by a division by R. Though this may be seen to be a coarse approximation, results produced that way always were convincing. Therefore:

$$Z = \frac{R^2 - X^2 - Y^2}{R}$$

and

$$Z = -\frac{R^2 - X^2 - Y^2}{R}$$

Points facing away from the observer (results from the second equation) are discarded. With the Bresenham's line algorithm, only $2R$ iterations will be performed whatever the direction of the cylinder. Thus, equation 3 can be simplified accordingly.

$$Z = \mathcal{F}_0(t) = \frac{R^2 - t^2}{R} \tag{B.4}$$

where t is a parameter which spans the range $-R$ to R. This is a parametric function which can be easily decomposed into an equation suitable for forward differencing.

$$\begin{aligned} \mathcal{F}_0(t+1) &= \mathcal{F}_0(t) - \frac{2t+1}{R} \\ \mathcal{F}_0(t+1) &= \mathcal{F}_0(t) + \mathcal{F}_1(t) \end{aligned} \tag{B.5}$$

similarly,

$$\begin{aligned} \mathcal{F}_1(t+1) &= \mathcal{F}_1(t) - \frac{2}{R} \\ \mathcal{F}_1(t+1) &= \mathcal{F}_1(t) + \mathcal{F}_2 \end{aligned} \tag{B.6}$$

Since at the beginning we have $t = -R$, so

$$\begin{aligned} \mathcal{F}_0 &= \mathcal{F}_0(-R) = 0 \\ \mathcal{F}_1 &= \mathcal{F}_1(-R) = \frac{2R-1}{R} \\ \mathcal{F}_2 &= -\frac{2}{-R} \end{aligned} \tag{B.7}$$

The implementation of these results are straightforward.

4 The common case

4.1 Displaying the disc

In this case, one disc and the sweep surface of the cylinder are visible. The disc of the cylinder which is visible is the one where its corresponding z value for P_1 or P_2 is the smallest one. The side is represented by an ellipsoidal disc.

Several implementation could have been used to draw this disc. In the first version of the algorithm, a bounding square was used in combination with a subdivision algorithm alike the Bezier algorithm to approximate the ellipse. One inconvenient of this algorithm was its computational cost when computing the z coordinates and when re-ordering the points afterwards. In the current version of the algorithm, each shade is also computed iteratively. That was not possible using the former version henceforth inflicting another penalty on the rendering time.

In another version, the Bresenham's circle algorithm was used to compute the real coordinates of each point. One problem was that I failed to find a suitable algorithm which would directly draw a circle in a three dimensional scene. Consequently, the circle was computed using the usual algorithm in 2D and then each point was transformed to get their real coordinates in 3D. Moreover, two problems remained which were related to the ordering of resulting vertices and the fact that several of these vertices had the same x and y values whereas in some other places, there were holes. Shading could also not be computed iteratively.

The current solution uses forward differencing to compute both positions and shades.

4.1.1 Computing the position of each point

To compute the position of each point, Bezier splines were used. A Bezier spline is expressed parametrically by:

$$P(t) = P \cdot C = [p_0 \cdots p_n] \begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} \tag{B.8}$$

where

n is the order of the Bezier spline
$p_0 \cdots p_n$ are the control points
$C_0(t) \cdots C_n(t)$ are the basis polynomials

The basis polynomial can be rewritten:

$$\begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} = \begin{bmatrix} m_0 0 & \cdots & m_0 n \\ \vdots & & \vdots \\ m_n 0 & \cdots & m_n n \end{bmatrix} \begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix} \tag{B.9}$$

If the C_i are the Bernstein polynomials ($C_i = B_i^n$) and if the spline is cubic (four control points), the resulting M matrix is:

$$M = \begin{bmatrix} 1 & -3 & 3 & 1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.10}$$

This is exactly what we have (Fig. B.3).

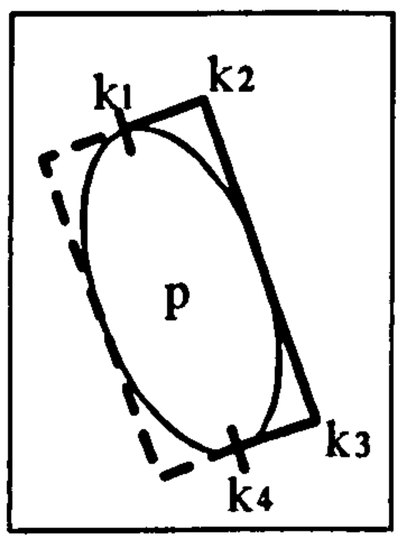


Figure B.3: How to compute the points of the ellipse

In this figure we can see that P is the centre of the ellipsoid and k_1, k_2, k_3, k_4 are the control points. Only one side is drawn at a time.

Rewriting $P(t)$ in the usual form, we have

$$P(t) = at^3 + bt^2 + ct + d \tag{B.11}$$

where

$$a = -k1 + 3(k2 - k3) + k4$$
$$b = 3(k1 - 2k2 + k3)$$
$$c = 3(-k1 + k2)$$
$$d = k1$$

Let's assume that we know $P(t)$. We then have

$$\begin{aligned} P(t+1) &= a(t+1)^3 + b(t+1)^2 + c(t+1) + d \\ &= P(t) + a + b + c + (3a + 2b)t + (3a)t^2 \\ &= P(t) + P_1(t) \end{aligned} \tag{B.12}$$

similarly,

$$\begin{aligned} P_1(t+1) &= a + b + c + (3a + 2b)(t+1) + 3a(t+1)^2 \\ &= P_1(t) + 6a + 2b + 6at \\ &= P_1(t) + 6a + 2b + P_2(t) \end{aligned} \tag{B.13}$$

and again

$$\begin{aligned} P_2(t+1) &= 6a + 2b + 6a(t+1) \\ &= P_2(t) + 6a \\ &= P_2(t) + P_3(t) \end{aligned} \tag{B.14}$$

If we start at $t = 0$, we then have

$$\begin{aligned} P_0 &= P(0) = d \\ P_1 &= P_1(0) = a + b + c \\ P_2 &= P_2(0) = 6a + 2b \\ P_3 &= P_3(0) = 6a \end{aligned} \tag{B.15}$$

To compute each point, $P_i = P_i + P_{i+1}$ have to be computed for each i between 0 and 2. The problem with this representation is that a single iteration is enough to get to the end. The step has to be made smaller. So if n points are needed

$$P\left(\frac{t+1}{n}\right) = a\left(\frac{t+1}{n}\right)^3 + b\left(\frac{t+1}{n}\right)^2 + c\left(\frac{t+1}{n}\right) + d \tag{B.16}$$

If we multiply this by n^3, we have

$$\begin{aligned} \mathcal{P}(t+1) &= n^3 P\left(\frac{t+1}{n}\right) \\ &= a(t+1)^3 + nb(t+1)^2 + n^2c(t+1) + n^3d \\ &= A(t+1)^3 + B(t+1)^2 + C(t+1) + D \end{aligned} \tag{B.17}$$

where

$$\begin{aligned} A &= a \\ B &= nb \\ C &= n^2c \\ D &= n^3d \end{aligned}$$

And from B.15:

$$\begin{aligned} \mathcal{P}(t) &= At^3 + Bt^2 + Ct + D \\ \mathcal{P}_0 &= \mathcal{P}(0) = D \\ \mathcal{P}_1 &= \mathcal{P}_1(0) = A + B + C \\ \mathcal{P}_2 &= \mathcal{P}_2(0) = 6A + 2B \\ \mathcal{P}_3 &= \mathcal{P}_3(0) = 6A \end{aligned} \tag{B.18}$$

In the current implementation, the divisor is the first number 2^n above $4R$ where n is a natural number and R is the radius of the cylinder, this in an effort to use shifting operations whenever possible.

Each point computed is stored into a dedicated array containing its position and colour. The way the colour is computed will be explained in the next section. Special care has to be taken to avoid redundant points. Thus, it is fairly possible that several different parametric points share the same resulting coordinates. To solve this problem, the forward differencing method was extended to make it more dynamic by selecting the best available step value between three possible values. The implementation of such an algorithm is simple. It just tries the first solution by taking the biggest step value and then it checks that no necessary intermediate solution has been discarded by comparing either the new column value or the new row value depending upon the direction of the cylinder[1]. If this is not the case (if a solution is missing), it tries the lower step size until the missing solution is found. Within each block, the code generated is significantly optimised to avoid redundant computations.

Before doing any of these computations, the first stage is to determine the coordinates of the $k_i s$. Knowing P_1 and P_2, u is computed as previously mentioned in Eq. 3.

[1]If the cylinder is more horizontal than vertical, then the points are stored in line order otherwise they are stored in column order. So if points are stored in line order, there must be at least one point per line

Then, the coordinates of k_1 have to be computed. The vector is obtained from $\overrightarrow{Ck_1}$, where C is either P_1 or P_2, is of length R and is perpendicular to the vector $\vec{u}$. If we call $\vec{v}$ the vector $\overrightarrow{Ck_1}$, then we have

$$u.v = u_x v_x + u_y v_y + u_z v_z = 0$$

From simple geometry and due to the orthogonal projection, it can be shown that $v_z = 0$. By deduction, there are only two possible solutions:

$$v_x = -u_y \; and \; v_y = u_x$$

and

$$v_x = u_y \; and \; v_y = -u_x$$

This solution works in every cases except when the $u_x = u_y = 0$ and u_z is the only value which is different of 0. Fortunately, this case cannot occur at this stage because it simply means that the cylinder can be represented by a single disc. The value of k_4 is obtained similarly.

k_2 and k_3 still have to be computed. Both $\overrightarrow{Ck_2}$ and $\overrightarrow{Ck_3}$ must be two vectors perpendicular to both the vector $\vec{u}$ and the vector $\vec{v}$. To obtain these points, we process like previously. Let state that l is equal to the vector Ck_2, then we have

$$u.l = u_x l_x + u_y l_y + u_z l_z = 0$$

$$v.l = v_x l_x + v_y l_y + v_z l_z = v_x l_x + v_y l_y = 0$$

$$l_x = \frac{v_y l_y}{v_x}$$

$$\frac{u_x v_y l_y}{v_x} + u_y l_y + u_z l_z = 0$$

$$l_y(\frac{u_x v_y}{v_x} + u_y) + u_z l_z = 0$$

$$l_z = l_y \frac{\frac{u_x v_y}{v_x} + u_y}{u_z}$$

Any solutions will satisfy what is being looked for so l_y is arbitrarily set to 1. Thus:

$$l_z = \frac{u_x \frac{v_y}{v_x} + u_y}{u_z}$$

and

$$l_x = \frac{v_y}{v_x}$$

The value of k_2 is then

$$k_2 = r\frac{l}{|l|} + k_1$$

and the value of k_3 is

$$k_3 = r\frac{l}{|l|} + k_4$$

The same procedure is used to compute the points on the other side.

4.2 Displaying the sweep surface

Each point has now been stored in the right order into an array ready to be used by the sweeping process. The Bresenham's line algorithm is used for this purpose. Since all the lines forming the sweep surface are parallel, the algorithm can work in a single pass. However, things are not that simple (Fig. B.4).

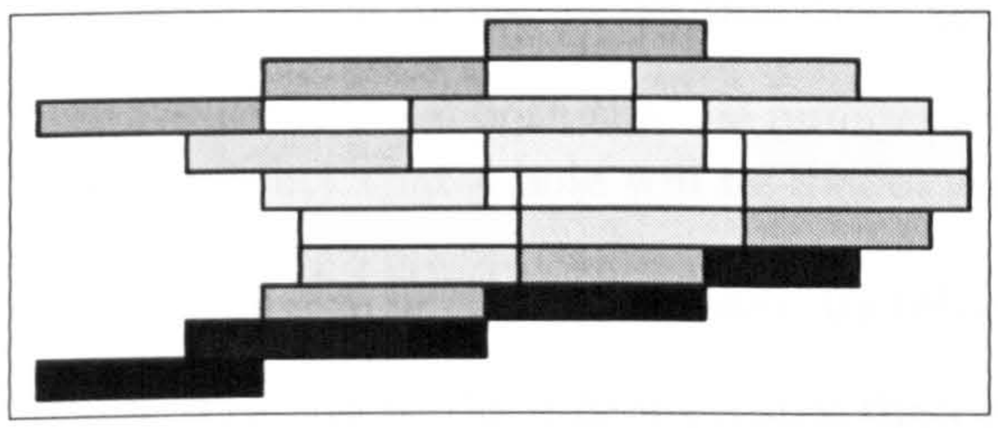


Figure B.4: Holes in the sweep surface

This example illustrates what a typical situation may be when drawing a cylinder. A technique has to be found to fill all these holes.

As can be seen from figure B.4, if only the simple Bresenham's line algorithm is used, then holes will appear. The problem is made even harder by the fact that these holes are of variable length and dependent upon the direction of the cylinder and of the relative position of each point. An example is used to illustrate the approach used to solve this problem. For purpose of simplicity, lines were drawn from left to right and from bottom to top. The direction of the cylinder is also more horizontal than vertical therefore there must be a single point per line (Fig. B.4). In the current implementation, lines were drawn in descending y order. Though, this is not necessary, this approach would be slightly different if this was not the case.

The process works in two parts. The first part takes place when the point is stored for the first time in the array whereas the second part takes place when displaying the sweep surface itself.

4.2.1 First part:

For each point added into the array, three different cases may occur:

- In the first case, the new point is added at the same column as the previous point. Therefore, each time the point will move one line up, this point will do it as well. So no special care has to be taken.

- In the second case, the new point is before the previous point (its x is below the one of the previous point). In this case no hole is produced as well. So, no special care needs to be taken. However, a number of pixels will be drawn many times. To improve the efficiency of the algorithm, a pixel should not plotted more than once. Unfortunately, no algorithm, which cost of ensuring such a requirements would not overcome its potential gain, was found.

- In the third case, the new point is located after the previous one (its x value is greater the one of the previous point). In this case, the size of the gap has to be measured (It is equal to the difference between the two x values) and stored with the position of the point and its colour.

In the first two cases, the size of the hole is set to zero.

4.2.1.0.1 Second Part: In this part, two different situations can occur.

- The next point computed stays on the same line. In other words, it is just the x value which increases. In this case, nothing special needs to be done.

- In the other case, both the line number and the column number change. There is a possibility that a hole will be made. Again, two different situations can occur

  - ☆ The size of the hole is zero, therefore no special care needs to be taken.

  - ☆ The size of the hole is greater than zero. A hole will then appear if nothing is done against it. What we have to do is to go one line up as intended, then come back of a number of pixels equals to the size of the hole and draw a horizontal line equal to the size of the hole + 1. This will have the effect of drawing the point and filling the hole at the same time.

5 Determining the colour

In this section, some of the formulae used are based on Patterson's article [Pat93]. Thus, they are not be described in details here. When displaying a cylinder on a screen, two parts have to be displayed. These are the sweep surface and one of the sides of the cylinder which can be represented either by a disc or by an ellipsoidal disc depending upon the situation. The ways colours are obtained for the two parts are quite different and so are described separately.

5.1 Displaying the disc

In the current implementation of this algorithm, a simple lighting model which uses a single light source positioned at infinity is being used. Hence, computations are easier and much faster than more complex model. However, the light source is not fixed. If this would be the case and pointing in the direction $\vec{u}$ where $|u| = (0,0,1)$, computation would be even simpler but resulting pictures may look too artificial. Since the light source is at infinity, light rays are parallel for every objects in the scene. Therefore the normalised value of the light vector is computed at the very beginning and used as a constant.

The lambert law formula for diffuse illumination at a point on a surface with unit normal N is

$$Colour = base\_colour + (K_d \times max(N \cdot L, 0) + K_a) \tag{B.19}$$

where

base_colour is the primary colour of the object
K_d is the proportion of light scattered by diffuse reflection
K_a is the proportion of ambient light
N is the normal vector at the intersection point between the ray and the surface
L is the unit vector towards the light source

This formula is certainly one of the simplest of the equations used to model light. However it is good enough for our purposes. It can even be simplified a little bit. The constant K_a which gives the proportion of the ambient light can be set to zero if it is ensured that the lowest colour available is equal to this constant. This is achieved by specifying that K_a is the lowest colour shade possible for the colour palette. The constant K_d then specifies the maximum number of shades available for a particular colour. Thus, Eq. B.19 becomes

$$Colour = base\_colour + (n \times max(N \bullet L, 0)) \tag{B.20}$$

where

base_colour is the primary colour of the object
n is the number of different shades available
N is the normal vector at the intersection point between the ray and the surface
L is the unit vector towards the light source

The only unknown is the normal vector N. For the disc of the cylinder, N is computing as following:

If P_1 is in front of P_2
then $N = \frac{P_2P_1}{length}$
else $N = \frac{P_1P_2}{length}$

where *length* is the length of the cylinder (P_1P_2). With $N = (x, y, z)$, the dot-product $N \cdot L$ is:

$$N \bullet L = xl_x + yl_y + zl_z \tag{B.21}$$

The colour value is obtained directly.

5.2 Displaying the sweep surface

The colour of each line composing the sweep surface still has to be computed. Since the light source is placed at infinity, these colours are the same all along their corresponding line.

Colours can be computed iteratively. If this could not be the case, then for each new point its value would have to be computed using equation B.20. This would inflict a big time penalty to the algorithm.

When displaying the cylinder, we know that two different cases can occur. Either no side is visible or only one side is visible. Since they have been implemented differently, they will be described separately.

5.2.1 When the no sides are visible:

To compute the colour of each line composing the sweep surface iteratively, we need to combine Eqs. B.4 and B.20 together. Again this is mainly based on Patterson's fast sphere algorithm [Pat93].

Combining Eqs. B.4 and B.20 together, we get:

$$\mathcal{F}(t) = base\_colour + \left(n \times max\left(\left(X, Y, \frac{R^2 - t^2}{R}\right) L, 0\right)\right) \tag{B.22}$$

where

X is the current x value of normal vector
Y is the current y value of normal vector
t is a parameter which span the range $-R$ to R
base_colour is the primary colour of the object
n is the number of different shades available
L is the unit vector towards the light source
R is radius of the cylinder

Simplifying for explanation purposes:

$$\mathcal{F}(X, Y, t) = base\_colour + (n \times max\,(\mathcal{F}_0, 0)) \tag{B.23}$$

and

$$\mathcal{F}(X,Y,t) = n\left(X, Y, \frac{R^2 - t^2}{R}\right) L \tag{B.24}$$

Rewriting $\mathcal{F}_0$

$$\mathcal{F}_0(X,Y,t) = n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l) \tag{B.25}$$

A parametric function with three parameters results. It can be divided into three separate functions:

$$\begin{aligned}
\mathcal{F}_x(X) &= n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l) \\
\mathcal{F}_y(Y) &= n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l) \\
\mathcal{F}_t(t) &= n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l)
\end{aligned}$$

The result of the modification of one parameter independently to the others can be studied. Since results are the same for $\mathcal{F}_x$ and $\mathcal{F}_y$, only $\mathcal{F}_x$ is described. $\mathcal{F}_t$ is described a bit later.

The parameter X can increase or decrease depending upon the direction of the cylinder. To simplify the analysis, only the case in which X values increases is shown. This will be easily generalised afterwards for decreasing X values.

$$\begin{aligned}
\mathcal{F}_x(X+1) &= n((X+1)l_x + Yl_y + \frac{R^2 - t^2}{R} l_z) \\
&= \mathcal{F}_x(X) + nl_x
\end{aligned} \tag{B.26}$$

Thus, each time X increases by one pixel, the constant $n \times l \bullet x$ only needs to be added to the function $\mathcal{F}$. Similarly, if X is decreasing, then $-n \times l \bullet x$ is used instead. For the Y parameter, the results are the same except that $l \bullet x$ is replaced by $l \bullet y$.

Now let's study Eq. B.25 when only the parameter t is changing.

$$\begin{aligned}
\mathcal{F}_{t0}(t) &= n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l_z) \\
\mathcal{F}_{t0}(t+1) &= n(Xl_x + Yl_y + \frac{R^2 - (t+1)^2}{R} l_z)
\end{aligned}$$

$$\begin{aligned}
\mathcal{F}_{t0}(t+1) &= n(Xl_x + Yl_y + \frac{R^2 - t^2}{R} l_z) - n\frac{2t+1}{R} l_z \\
\mathcal{F}_{t0}(t+1) &= \mathcal{F}_{t0}(t) - n\frac{2t+1}{R} l_z \\
\mathcal{F}_{t0}(t+1) &= \mathcal{F}_{t0}(t) + \mathcal{F}_{t1}(t)
\end{aligned} \tag{B.27}$$

Differencing a bit further,

$$\begin{aligned}
\mathcal{F}_{t1}(t+1) &= -n\frac{2(t+1)+1}{R} l_z \\
\mathcal{F}_{t1}(t+1) &= -n\frac{2t+1}{R} l_z - \frac{2nl_z}{R} \\
\mathcal{F}_{t1}(t+1) &= \mathcal{F}_{t1}(t) - \frac{2nl_z}{R} \\
\mathcal{F}_{t1}(t+1) &= \mathcal{F}_{t1}(t) + \mathcal{F}_{t2}(t)
\end{aligned} \tag{B.28}$$

At the beginning $t = -R$, so

$$\begin{aligned}
\mathcal{F}_{t0} &= \mathcal{F}_{t0}(-R) = n\left(Xl_x + Yl_y + \frac{R^2 - R^2}{R} l_z\right) = n(Xl_x + Yl_y) \\
\mathcal{F}_{t1} &= \mathcal{F}_{t1}(-R) = -n\frac{-2R+1}{R} l_z = n\frac{2R-1}{R} l_z \\
\mathcal{F}_{t2} &= \mathcal{F}_{t2}(-R) = -\frac{2nl_z}{R}
\end{aligned} \tag{B.29}$$

This is a parametric function which can be easily decomposed into equations suitable for forward differencing.

$$\begin{aligned}
\mathcal{F}_0(t+1) &= \mathcal{F}_0(t) - n\frac{2t+1}{R} L \\
\mathcal{F}_0(t+1) &= \mathcal{F}_0(t) + \mathcal{F}_1(t)
\end{aligned} \tag{B.30}$$

similarly,

$$\begin{aligned}
\mathcal{F}_1(t+1) &= \mathcal{F}_1(t) + n\frac{2}{R} L \\
\mathcal{F}_1(t+1) &= \mathcal{F}_1(t) + \mathcal{F}_2
\end{aligned} \tag{B.31}$$

Since at the beginning we have $t = -R$, so

$$\begin{aligned}
\mathcal{F}_0 &= \mathcal{F}_0(R) = base_colour + (n\frac{R^2 - t^2}{R}L) \\
\mathcal{F}_1 &= \mathcal{F}_1(R) = -n\frac{2R+1}{R}L \\
\mathcal{F}_2 &= n\frac{2}{R}L
\end{aligned} \tag{B.32}$$

The implementation of these equations into an algorithm is straightforward.

5.2.2 When only one side is visible:

To compute the colour of each line composing the sweep surface iteratively, Eqs. B.18 and B.20 are combined together.

$$\mathcal{F}(t) = base\_colour + (nmax\left((X_c - \mathcal{P}_x(t), Y_c - \mathcal{P}_y(t), Z_c - \mathcal{P}_z(t))\, L, 0\right)) \tag{B.33}$$

where

t is a parameter which span the range 0 to k
k is the number of iterations
$\mathcal{P}_x$ is the current x coordinate
$\mathcal{P}_y$ is the current y coordinate
$\mathcal{P}_z$ is the current z coordinate
(X_c, Y_c, Z_c) represents the centre of the disc
base_colour is the primary colour of the object
n is the number of different shades available
L is the unit vector towards the light source

Like in the previous paragraph, this can be simplified by just taking the important bits of the above equation.

$$\begin{aligned}
\mathcal{F}(t) &= base_colour + max\left(\mathcal{F}_0, 0\right) \\
\mathcal{F}_0(t) &= n\left(X_c - \mathcal{P}_x(t), Y_c - \mathcal{P}_y(t), Z_c - \mathcal{P}_z(t)\right) L \\
\mathcal{F}_0(t) &= n\left((X_c - \mathcal{P}_x(t))l_x + (Y_c - \mathcal{P}_y(t))l_y + (Z_c - \mathcal{P}_z(t))l_z\right)
\end{aligned} \tag{B.34}$$

Like in the previous paragraph, we only need to study the changes in one coordinate. This is easily generalised to other coordinates. In our example, only changes of the X coordinate have to be studied. Eq. B.34 can then be rewritten with the Y and Z coordinates fixed.

$$\begin{aligned}
\mathcal{F}_0(t) &= n\left((X_c - \mathcal{P}_x(t))l_x + Yl_y + Zl_z\right) \\
\mathcal{F}_0(t) &= n\left((X_c - (A_x t^3 + B_x t^2 + C_x t + D_x))l_x + Yl_y + Zl_z\right)
\end{aligned} \tag{B.35}$$

where

$$A_x = a_x$$
$$B_x = nb_x$$
$$C_x = n^2 c_x$$
$$D_x = n^3 d_x$$

and

n is the number of iterations

$$a_x = -k1_x + 3(k2_x - k3_x) + k4_x$$
$$b_x = 3(k1_x - 2k2_x + k3_x)$$
$$c_x = 3(-k1_x + k2_x)$$
$$d_x = k1_x$$

and

$k1, k2, k3, k4$ are the control points

Differencing B.35, we get:

$$\begin{aligned}
\mathcal{F}_0(t+1) &= n[\{X_c - (A_x(t^3+3t^2+3t+1) + B_x(t^2+2t+1) + \\
&\quad C_x(t+1) + D_x)\}l_x + Yl_y + Zl_z] \\
\mathcal{F}_0(t+1) &= n[\{X_c - (A_x t^3 + B_x t^2 - C_x t + D_x)\}l_x + Yl_y + Zl_z] - \\
&\quad n[A_x(3t^2+3t+1) + B_x(t^2+2t+1) + C_x(t+1) + D_x]l_x \\
\mathcal{F}_0(t+1) &= \mathcal{F}_0(t) - n[A_x(3t^2+3t+1) + B_x(t^2+2t+1) + C_x(t+1) + D_x]l_x \\
\mathcal{F}_0(t+1) &= \mathcal{F}_0(t) + \mathcal{F}_1(t) \qquad \text{(B.36)}
\end{aligned}$$

Rewriting $\mathcal{F}_1(t)$:

$$\begin{aligned}
\mathcal{F}_1(t) &= -n[A_x(3t^2+3t+1) + B_x(t^2+2t+1) + C_x(t+1) + D_x]l_x \\
\mathcal{F}_1(t) &= -n[(3A_x + B_x)t^2 + (3A_x + 2B_x + C_x)t + \\
&\quad A_x + B_x + C_x + D_x])l_x \qquad \text{(B.37)}
\end{aligned}$$

We can now difference $\mathcal{F}_1(t)$:

$$\begin{aligned}
\mathcal{F}_1(t+1) &= -n[(3A_x + B_x)(t^2+2t+1) + \qquad \text{(B.38)} \\
&\quad (3A_x + 2B_x + C_x)(t+1) + A_x + B_x + C_x + D_x]l_x \\
\mathcal{F}_1(t+1) &= -n[(3A_x + B_x)t^2 + (3A_x + 2B_x + C_x)t + A_x + B_x + C_x + D_x]l_x \\
&\quad -n[(3A_x + B_x)(2t+1) + 3A_x + 2B_x + C_x]l_x \\
\mathcal{F}_1(t+1) &= \mathcal{F}_1(t) - n[(3A_x + B_x)(2t+1) + 3A_x + 2B_x + C_x]l_x \\
\mathcal{F}_1(t+1) &= \mathcal{F}_1(t) + \mathcal{F}_2(t) \qquad \text{(B.39)}
\end{aligned}$$

Again differencing $\mathcal{F}_2(t)$, we have:

$$\begin{aligned}
\mathcal{F}_2(t+1) &= -n[(3A_x + B_x)(2(t+1)+1) + 3A_x + 2B_x + C_x]l_x \\
\mathcal{F}_2(t+1) &= -n[(3A_x + B_x)(2t+1) + 3A_x + 2B_x + C_x]l_x - n[(3A_x + B_x)2t]l_x \\
\mathcal{F}_2(t+1) &= \mathcal{F}_2(t) - n[2(3A_x + B_x)]l_x \\
\mathcal{F}_2(t+1) &= \mathcal{F}_2(t) + \mathcal{F}_3(t) \qquad \text{(B.40)}
\end{aligned}$$

Forward differencing is now finished. At the initialisation, the parameter t is equal to zero. So:

$$\begin{aligned}
\mathcal{F}_0 &= \mathcal{F}_0(0) = n[(X_c - D_x)l_x + Yl_y + Zl_z] \\
\mathcal{F}_1 &= \mathcal{F}_1(0) = -n[A_x + B_x + C_x + D_x]l_x \\
\mathcal{F}_2 &= \mathcal{F}_2(0) = -n[6A_x + 3B_x + C_x]l_x \\
\mathcal{F}_3 &= \mathcal{F}_3(0) = -n[2(3A_x + B_x)]l_x
\end{aligned}$$

The implementation of these equations is straightforward.

6 Conclusion

A fast algorithm to render cylinders was described. Many of the concepts used where first suggested by Fuchs [FGH+85] and successfully used by Patterson [PW94]. This algorithm assumes an orthogonal projection and a single light sources placed at infinity. It cannot use texture mapping but is sufficient to provide a useful representation of articulated figures.

Compared to Blinn's fast cylinder algorithm [Bli89], it does not suffer the visual artifacts when the algorithm is viewed from the sides. Using Blinn's algorithm, polygons become clearly visible. However, this algorithm is a great deal more intricate to implement.

Appendix C

Forms for the evaluation

1 Usability Evaluation

Thank you for agreeing to help evaluate the usability of different positioning techniques. I am interested in finding out which technique is the most appropriate to use to pose 3D articulated figures, a humanoid in this case.

Posing articulated figures is an important part of the animation process to produce films such as Toy Story, Jurassic Park, etc. I have implemented the three main techniques used.

During this evaluation, you will use two of these techniques. You will have a training session for each of the techniques in which you will learn how to produce one pose. Next, you will try out your skills to produce two other poses. I will measure how long it takes you to achieve these poses. So you should try to produce these poses as fast as possible.

When you have achieved a pose, the computer will tell you that it is finished. Sometimes, although you may believe that you have completed the task, the computer is not satisfied with what you have produced so far and does not say that it is finished. This is because you are missing some small details. So, because I am more experienced in the poses to achieve, I shall also be beside you to tell you what, but not how, you still need to do to reach the required pose.

After each session, you will have to fill in a short questionnaire about the technique you have just used. Please remember it is these techniques which are being evaluated and not you. The results of this evaluation may be published, but all the data recorded is anonymous. If you're not happy then you may stop at any time, and recordings and notes taken will be destroyed.

Tell me when you are ready and I will instruct you on how to proceed with the exercise. Do you have any questions ?

I have read the above and will be paid five pounds for participating.

Signature: _____ Date: __/__/97

2 Explaining the NASA-TLX

2.1 Introduction

Positioning or posing articulated figures is an important part of the animation process in the industry. The aim of this experiment is to work out which of the existing techniques is the best, This experiment will be divided into two sessions of half an hour each.

2.2 Workload tests

After each session, I will ask you to fill-in some tables. I am not only interested in finding out how fast you produced the required poses, I also want to find out about your experiences during the completion of the tasks. I am now going to describe how I will do this.

I am examining the "workload" you experienced. Workload is difficult to define precisely but easy to understand generally. The factors that influence your experiences in the positioning may come from the technique used itself, your feelings about your own performance, how much effort you put in, or the stress and frustration you felt. The workload contributed by these different factors may change as you get more familiar with the technique. The physical parts of workload are easy to measure but the mental ones are harder.

Since workload is something that is experienced individually by each person, there are no effective measures that can be used to estimate the workload of different activities. One way to find out about workload is to ask people to describe feelings they experienced, Because workload may be caused by many different factors, we would like to evaluate several of them individually. This set of 6 scales was developed for you to use in evaluating your experiences in different tasks. Please read the definitions of the scales carefully. If you have a question about any of the scales in the table please ask me about it. It is extremely important that they be clear to you. You may keep the descriptions with you during the experiment.

After each session, I will ask you to fill-in the 6 scales. You will evaluate the technique you have just used by marking each scale at the point which matches your experience. Each line has a description at each end. Please consider your response carefully. Consider each scale individually. Your ratings will play an important role in the evaluation being conducted, thus your active participation is essential to the success of this experiment, and is greatly appreciated. The last scale described on the sheet will not be used until after both halves of the experiment have been completed.

Rating Scale Definitions		
Title	**End points**	**Description**
Mental demand	*Low/High*	How much mental, visual activity was required? (e.g. thinking, deciding, calculating, looking, searching)
Physical demand	*Low/High*	How much physical activity was required ? (e.g.pushing, pulling, turning, controlling)
Time pressure	*Low/High*	How much time pressure did you feel because of the rate at which things occured? (e.g. slow, leisurely, rapid, frantic)
Effort expended	*Low/High*	How hard did you worked(mentally and physically) to accomplish your level of performance ?
Performance level achieved	*Poor/Good*	Hou successful do you think you were in doing the task set be the experimenter ? How satisfied were you with your performance
Frustration experienced	*Low/High*	How much frustration did you experience ? (e.g. were you relaxed, content, stressed, irritated, discouraged ?)
Overall preference	*Low/High*	Rate your preference for the two techniques. Which one made the task easier ?

Figure C.1: Workload scales

3 Sample marking sheet

4 Training sheets

4.1 Learning to use the generator

UNIVERSITY of GLASGOW

Computing Science

Posing a humanoid using the Generator

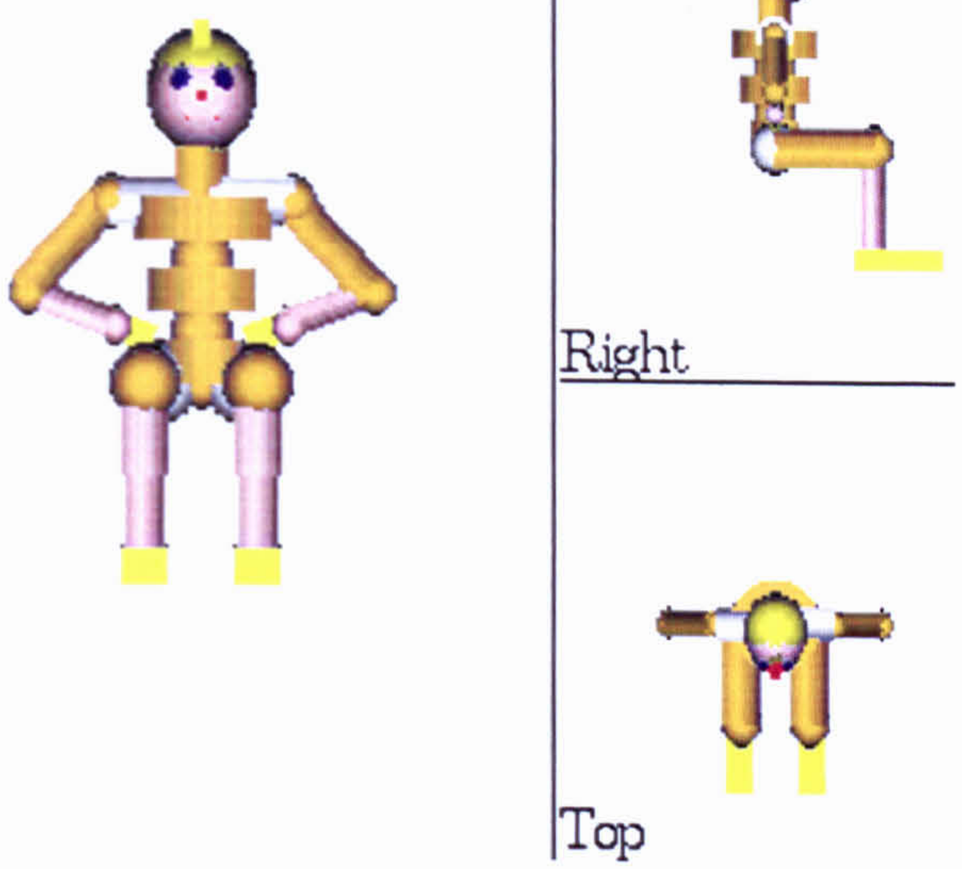
Stephane Etienne

April 30, 1998 .

The purpose of this evaluation is to evaluate how good different techniques are at posing (or positioning) articulated figures (a humanoid in this case).

For this purpose, we are going to try to produce the following pose. It might represent someone sitting on some invisible chair with the hands pointing toward the pelvis.

We are going to use a tool called **Generator**.



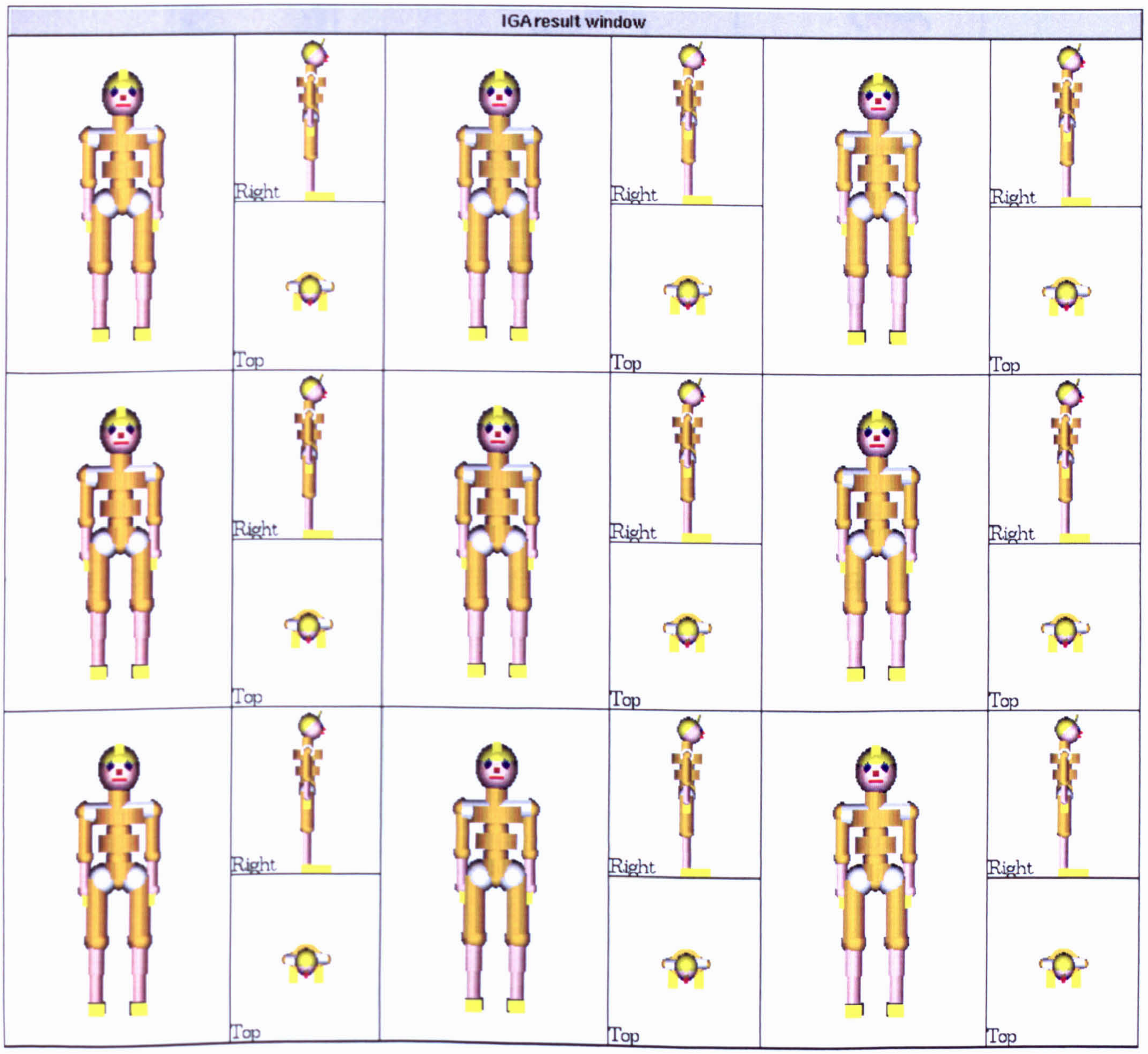
Colours have been selected to help you. Apart from the head, they all mean something.

- **Grey:** The corresponding limb cannot move
- **Pink:** Only flexion motions are possible (cf forearms)
- **Yellow:** Flexion and pivot motions are possible. The joint resembles part of a sphere
- **Orange:** Flexion and Pivot plus Twist motions are possible. For Twist, the limb rotates around it self.

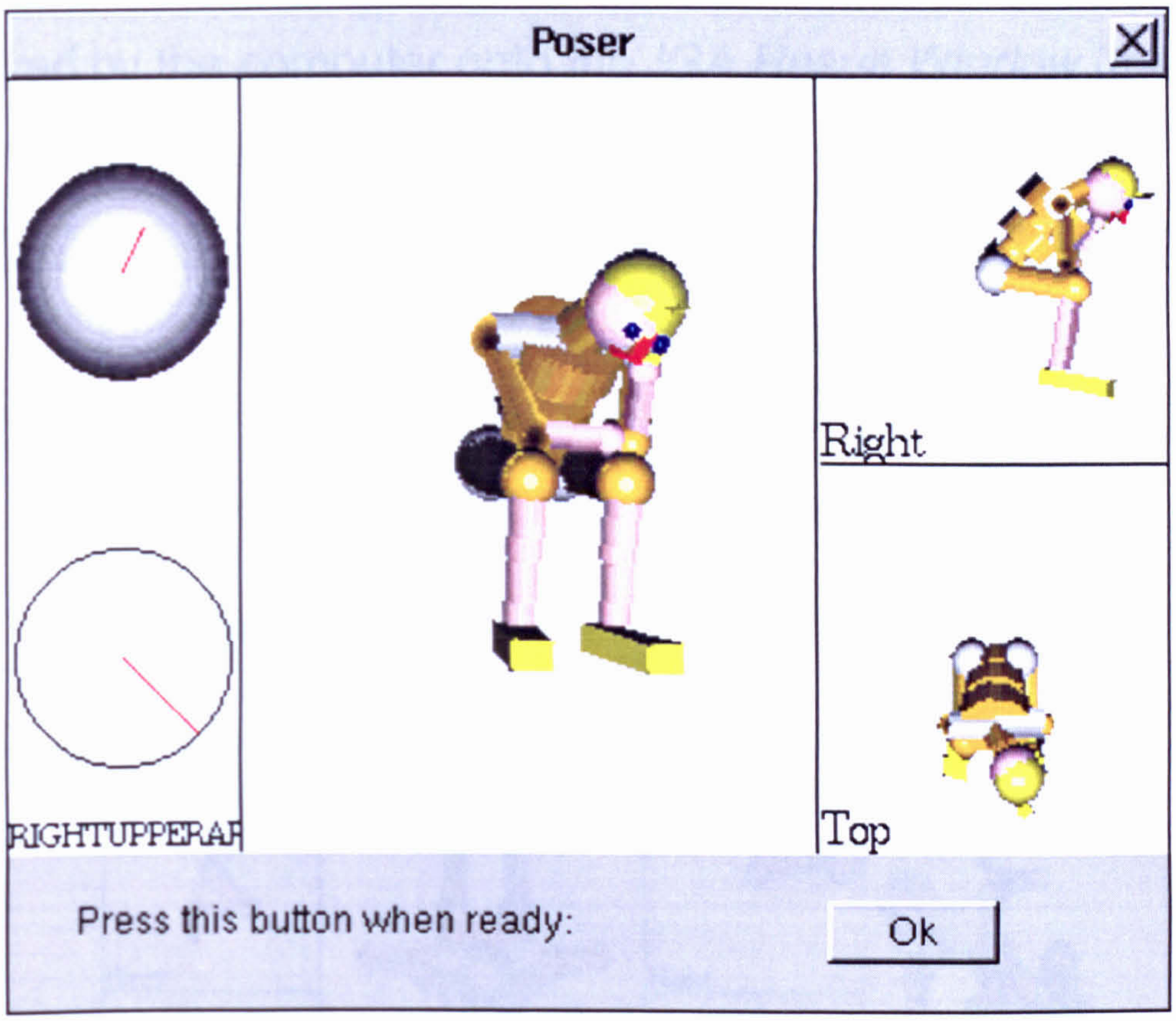
When you are ready, press on *Ok* at the bottom of this window. The computer then displays a pose that you will try to achieve. Take a careful lookat it. It is also displayed beside you on a sheet of paper. Ask me if you cannot find this sheet of paper. When you are ready, press *Ok*.

While you try to pose the figure, the computer will record all sort of informations which will enable me to work out how good the technique is. Try to produce the pose as fast as possible. When you will be close enough to it, the computer will tell you that it is fine and that you can stop.

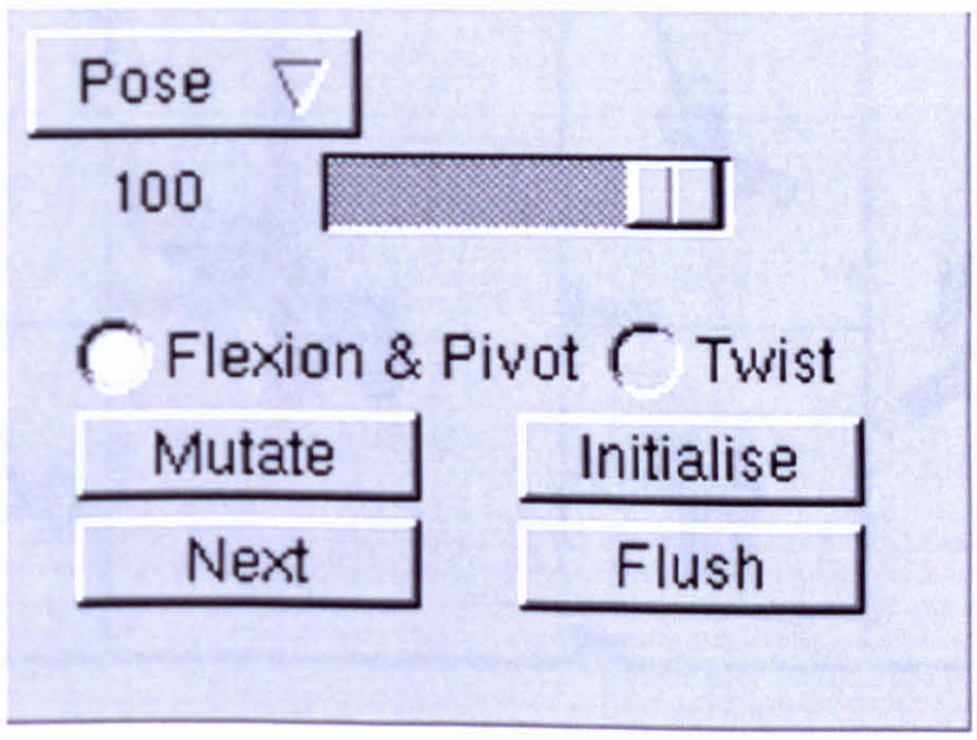
You should have in front of you a big window with nine standing humanoids. Beside each humanoid, you have different views of the same character. The one at the top shows the humanoid from the front, the one below shows the humanoid from the right hand side and the one at the bottom shows the humanoid from above.



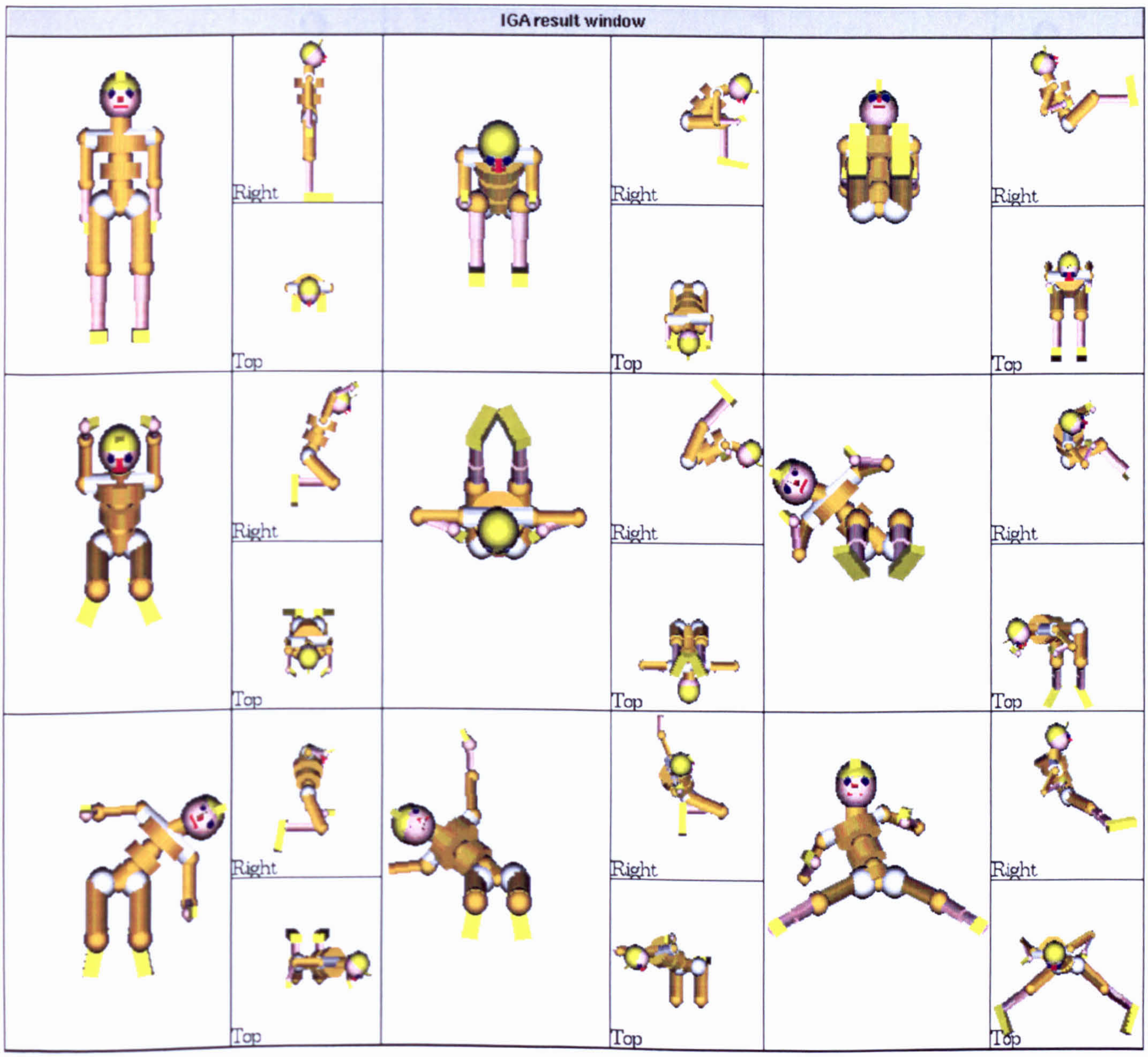
On the left of the screen, you should have a window called *Poser*. In this window, you should have a standing humanoid.



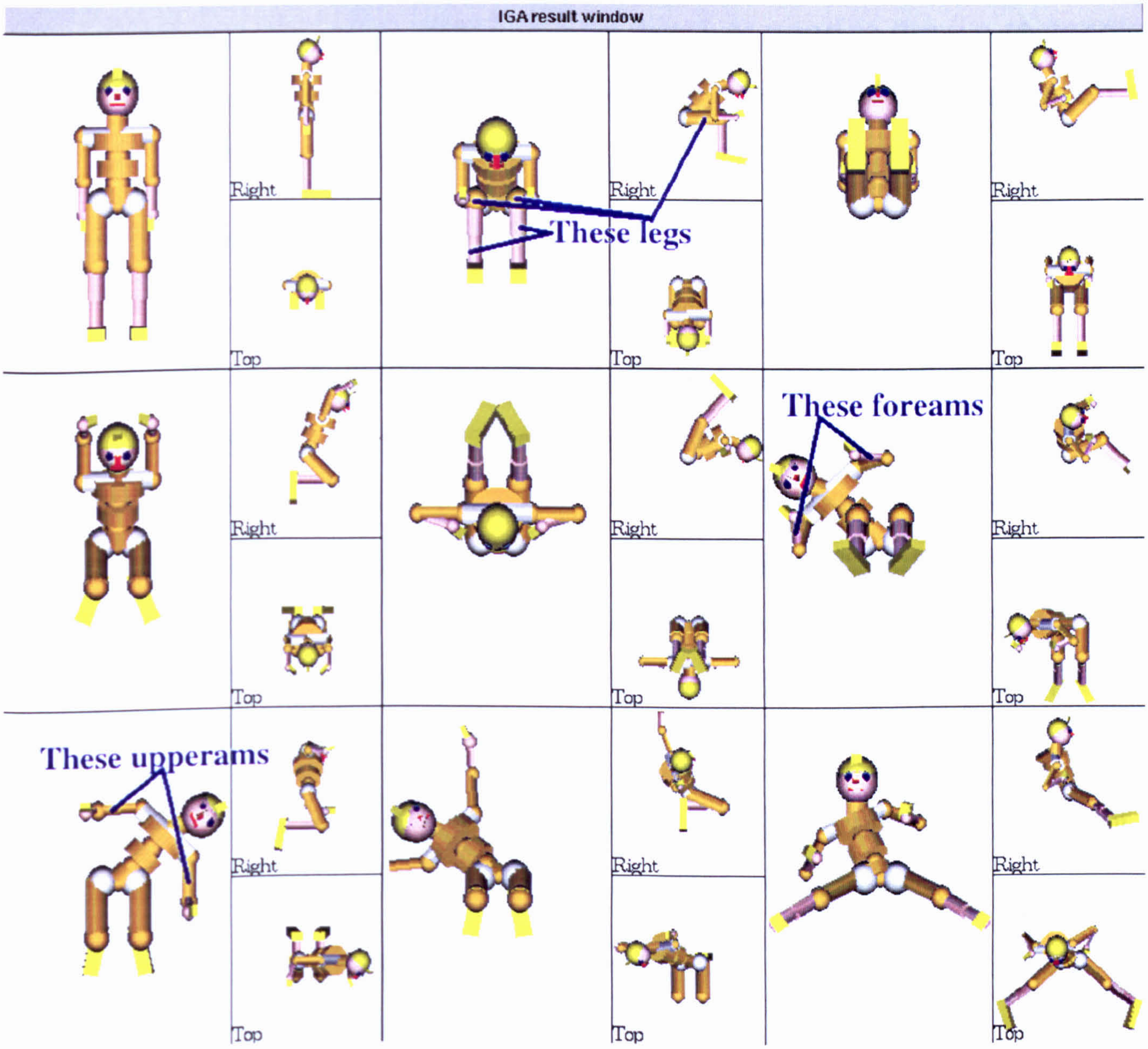
At the bottom, there is also a grey window called the *Main Window*. From this window, we are only interested in the second panel.



The first thing to do is to press the *Mutate* button on the panel. Alternatively, you can also press the *M* key. Nine poses are then produced and displayed by the computer onto the *IGA Result Window* (the window with the nine figures):



To produce the required pose, you have to select the relevant limb positions. These will be automatically copied onto the *Poser* window. For example, you will find:



To select a limb, you just have to click on it. If the limb is pointing right toward you, it may not work. In this case, either you use one the views beside it or you draw a rectangle around the limb.

If you want to select more than one limb, you just need to draw a rectangle around the groups of limb positions you want to copy and all limbs which lie entirely inside it will be copied.

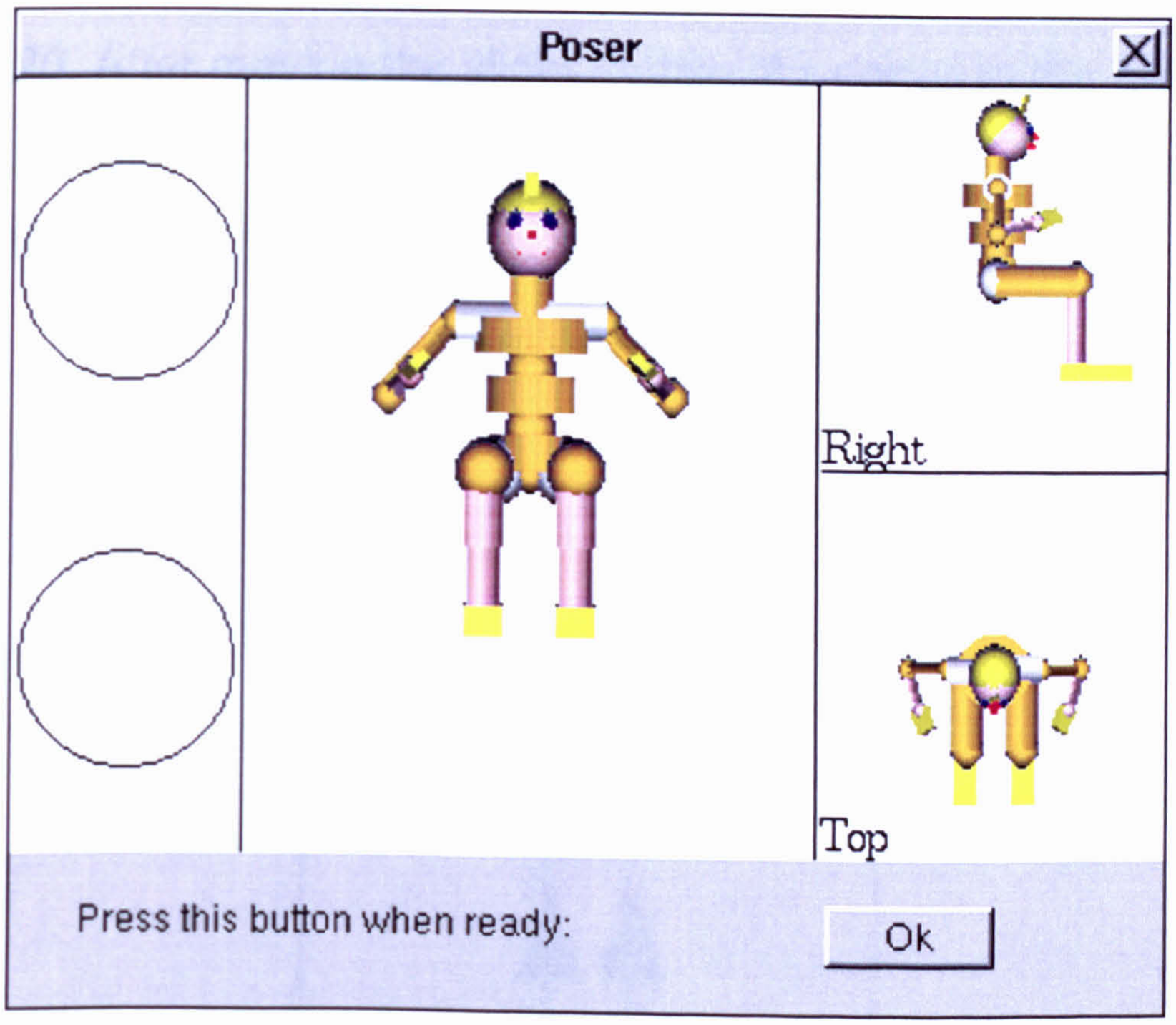
If you have made a mistake, you can undo the last operation by pressing *control-u*. Undoing twice just comes back to the original. Just try it out.

You may also want to rotate the humanoids to see them from a different angle. For example, if you want to rotate the humanoids from left to right, bring the mouse cursor at the level of the hip on the right side. Once there, press the *control* key, the left mouse button and drag the mouse cursor leftward. This will rotate the current figure. Once you are in the right position, release the mouse button and the *control* key. All other figures will be redrawn at this point.

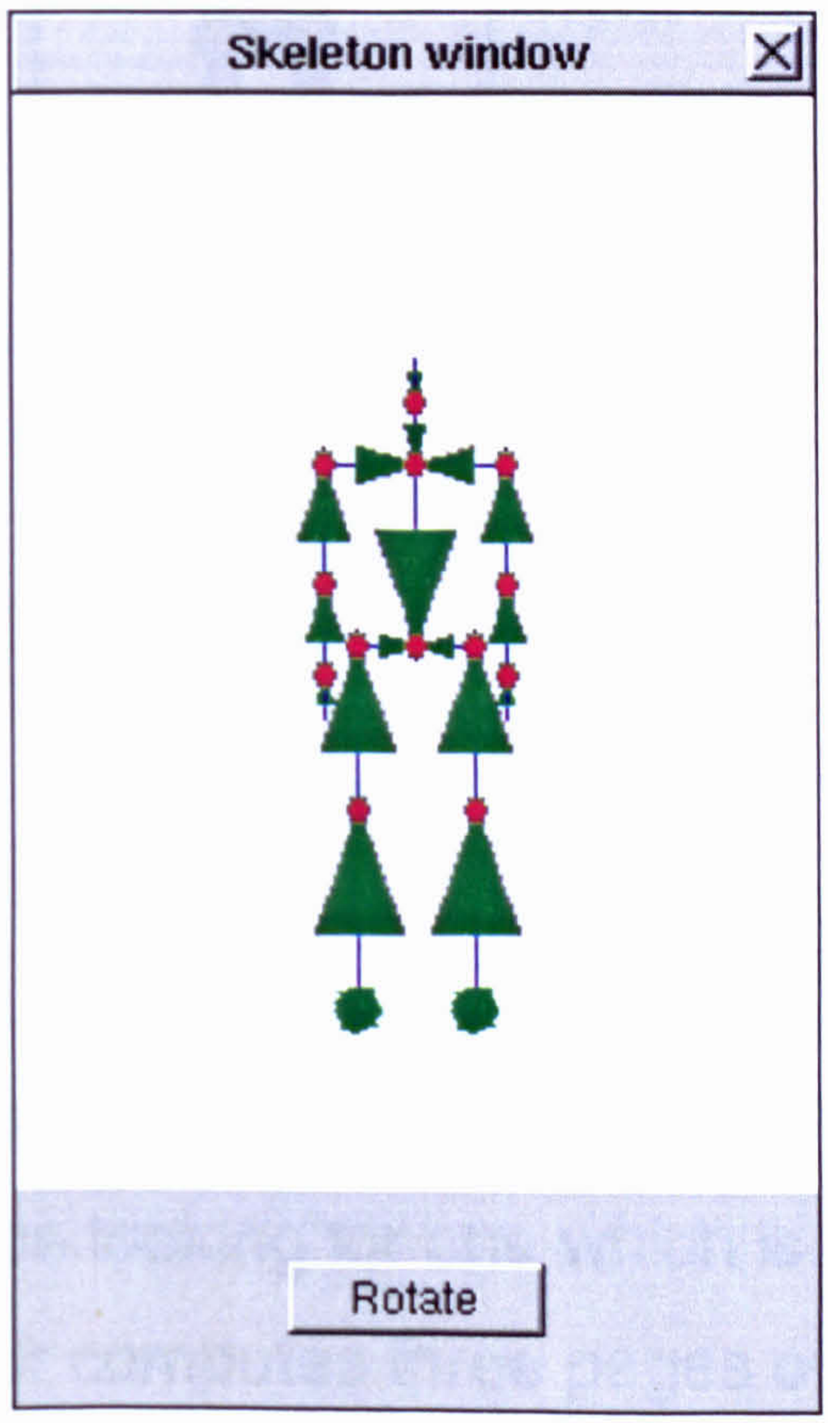
If you have problems rotating the humanoid, try to go trough the centre of the humanoid (the hip), the results will be more predictable.

To bring back the articulated figure in the original view (the front view), press the *control* key again, and double click.

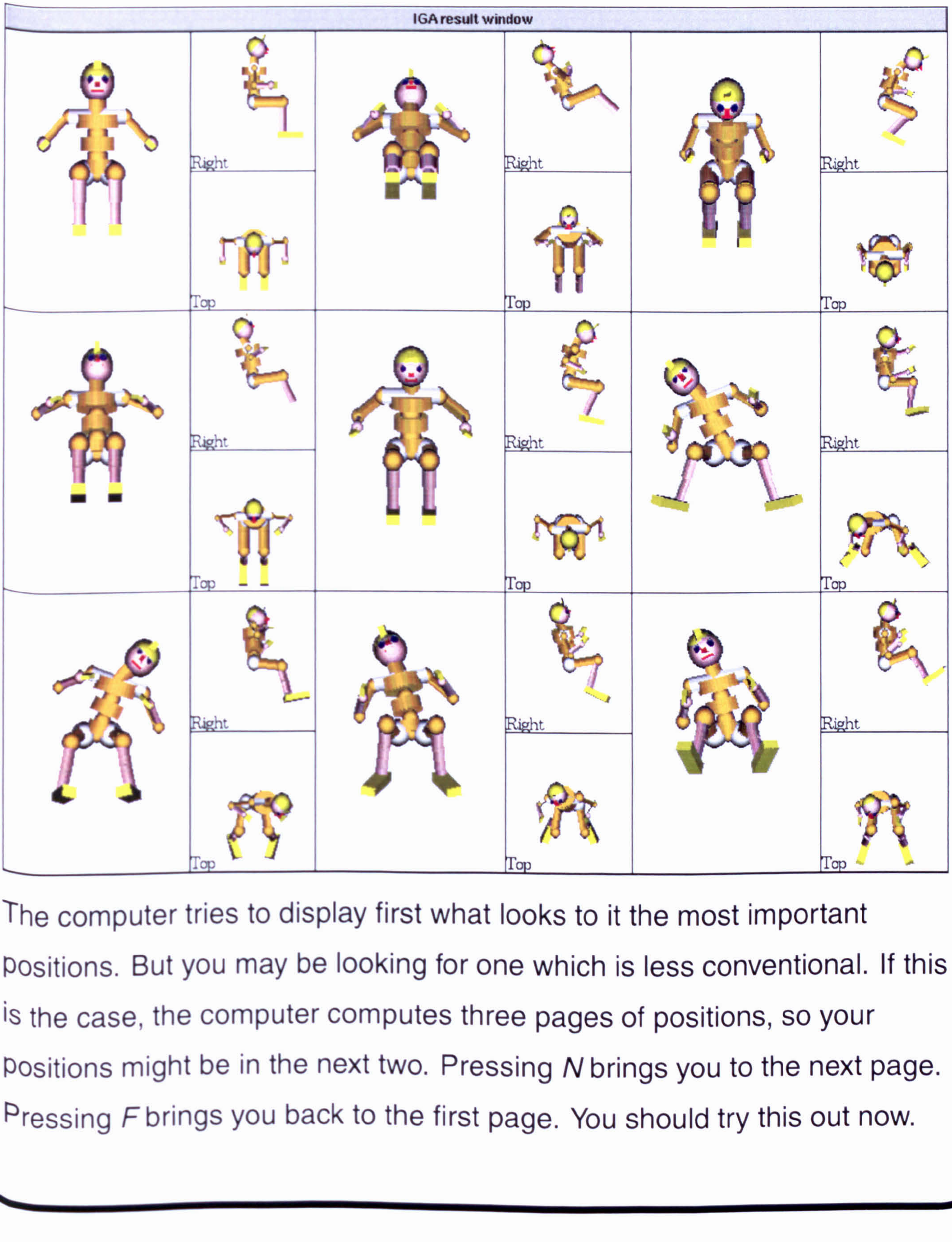
Once all limbs have been selected, you should end up with a figure looking like this:



In what will follow, the computer will use the pose displayed in the *Poser* window as a seed to generate a new set of positions. On the *Panel*, you also have a slider. It tells how strong the mutation will affect the seed pose. At *100*, the value it has at the moment, it will look very different from the seed pose. At *0*, all poses generated will be exactly like the seed pose. Although the pose we have at the moment is just perfect, bring the slider to a value of *20*. After moving the slider, notice the cones in the *Skeleton Window*, a window on the left hand side of the screen. They more or less describe the areas where new positions will lie.



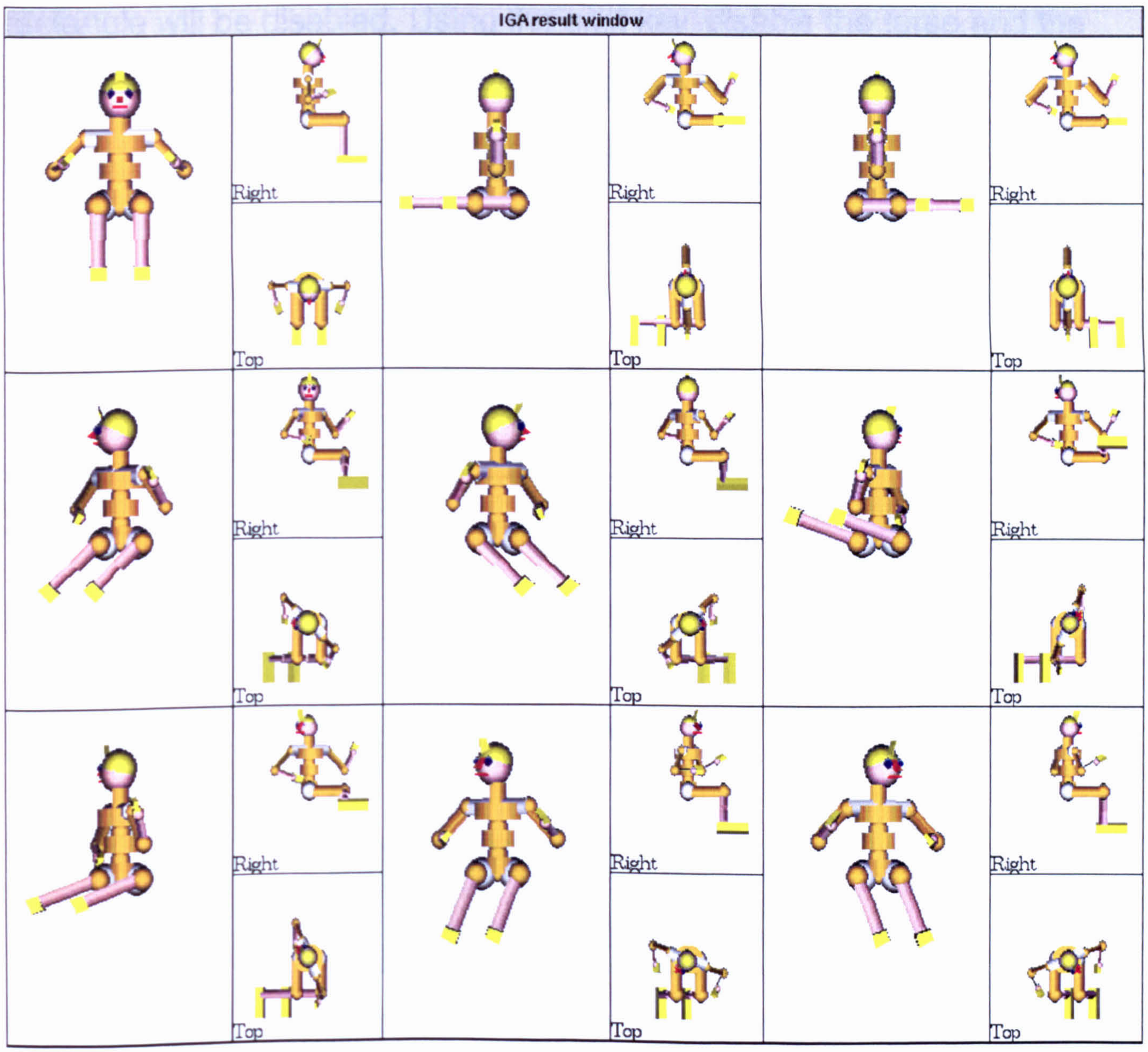
After pressing *Mutate* or the *M* key, a new set of positions are displayed. Notice how close they are to the original position.



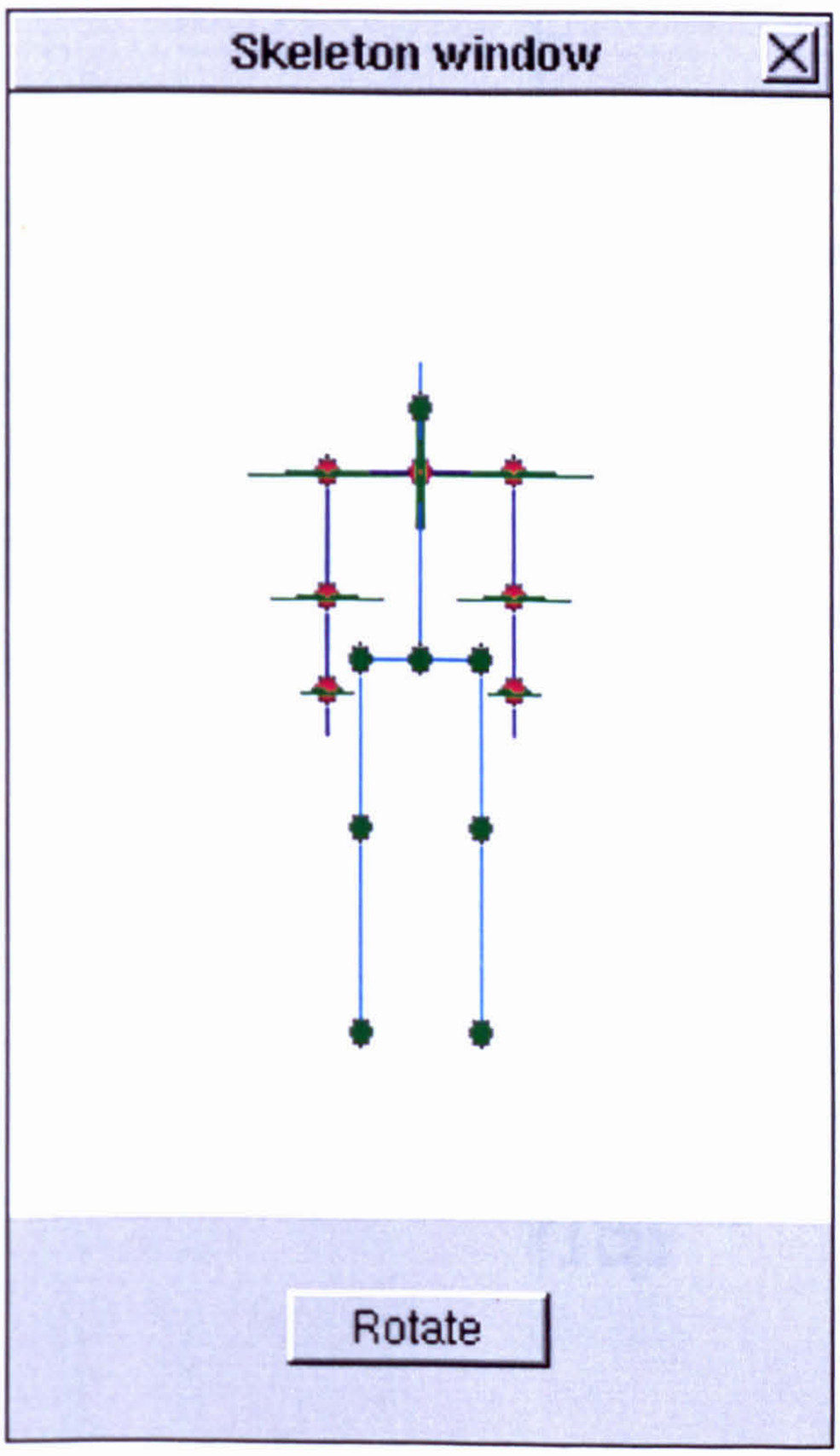
The computer tries to display first what looks to it the most important positions. But you may be looking for one which is less conventional. If this is the case, the computer computes three pages of positions, so your positions might be in the next two. Pressing *N* brings you to the next page. Pressing *F* brings you back to the first page. You should try this out now.

The next thing to do is to rotate the arms so that the hands point towards the hip. To perform this, you need to select *Twist* from the *Panel*. Because the changes are quite big, you should bring the slider back to *100*.

Pressing *Mutate*, you will get something like this:

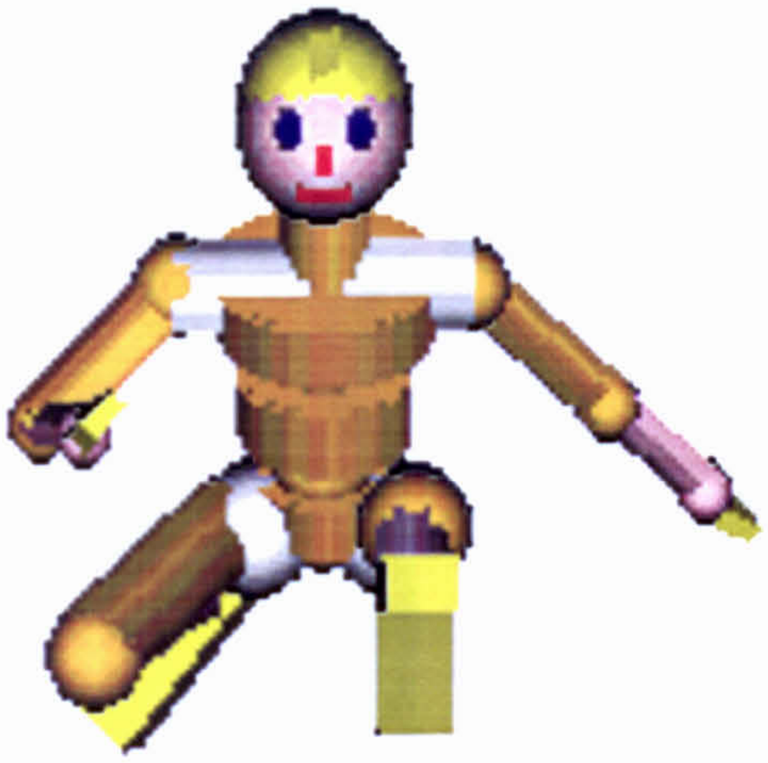


As not only the arms but also the torso and the legs have moved, you may find it difficult to see the interesting positions. So, to avoid this problem, you will draw a rectangle around the group of limbs forming the arms and the torso in the *Skeleton window*. If you press the *shift* key while dragging the mouse cursor to draw the rectangle, the group of limbs inside the rectangle will be disabled. Using the *shift* key, disable the torso and the neck. You should end up with something looking like this:



Mutating again, you can see that only the arms have been rotated. Select the one you think are the best (by clicking onto the arm, not the forearm !).

Before you go to the evaluation strictly speaking, you will try to produce the following pose. It represents a sportsman jumping over a barrier. I will be there to help you and answer your questions if any.

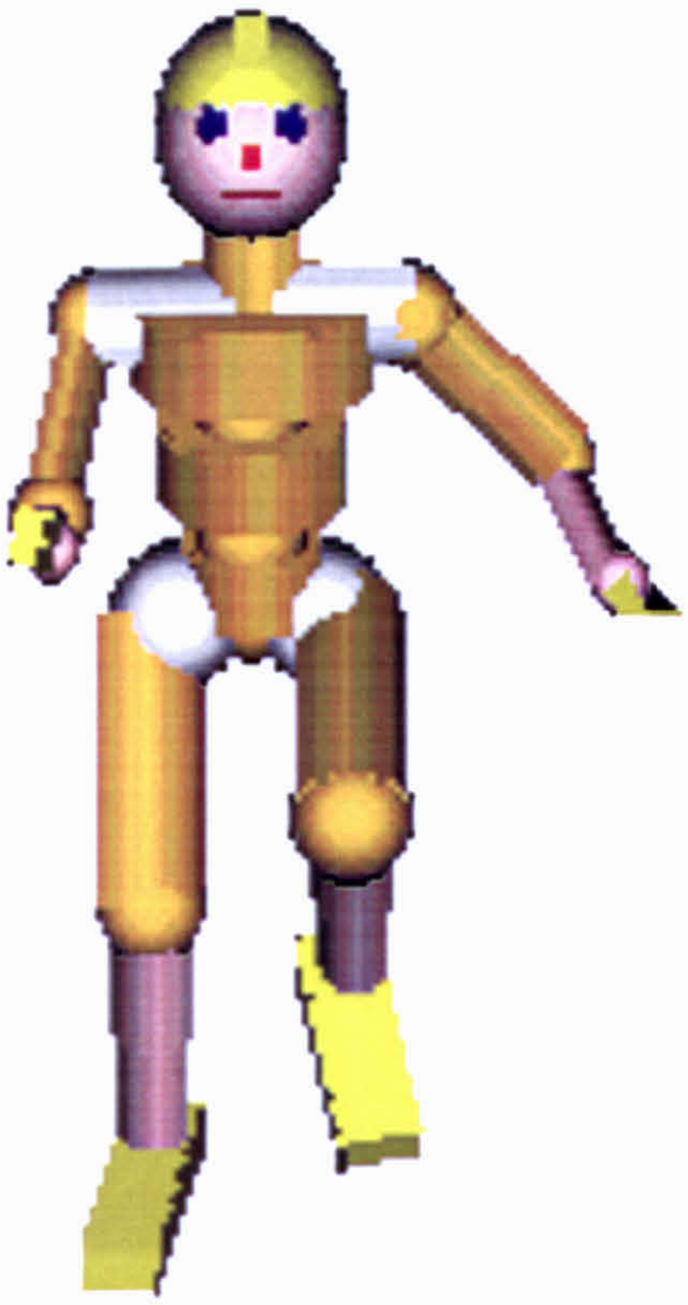


Right

Top

Now, so that I can evaluate how good the technique is, you will try to produce the following poses.

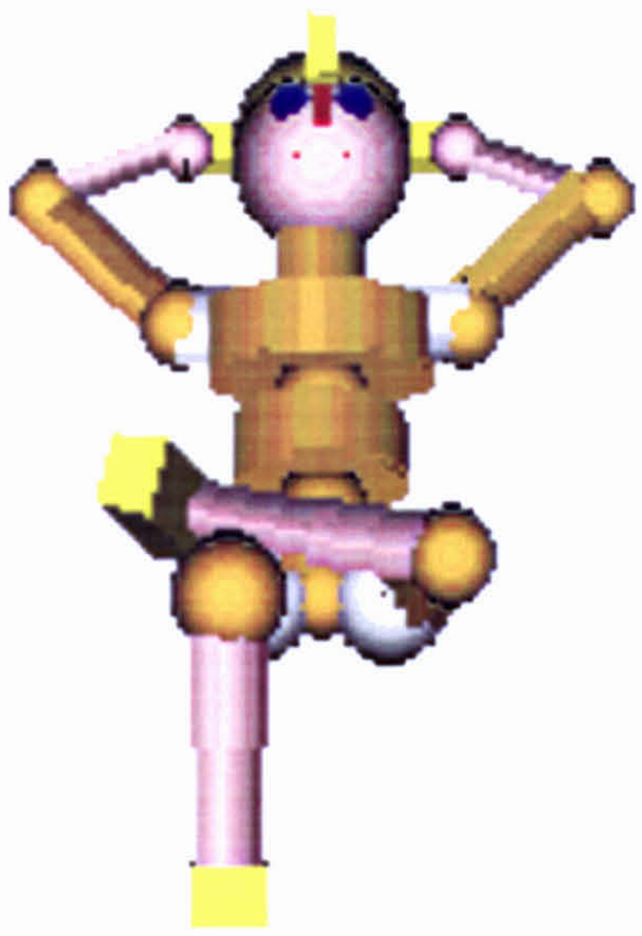
The first one represents a runner.



Right

Top

The second one represents someone lying on some invisible chair, the left leg resting on top of the other one, the hands at the back of the head.



Right

Top

4.2 Learning to use forward kinematics

UNIVERSITY
of
GLASGOW

Computing Science

Posing a humanoid using forward kinematics

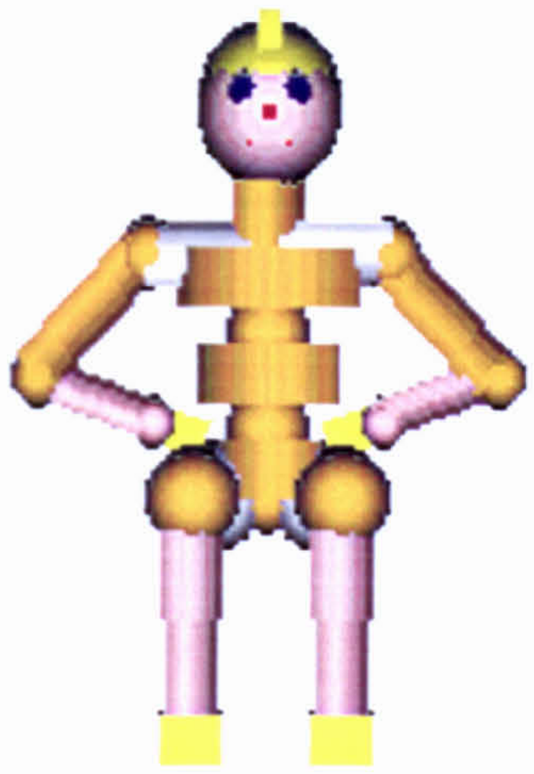
Stephane Etienne

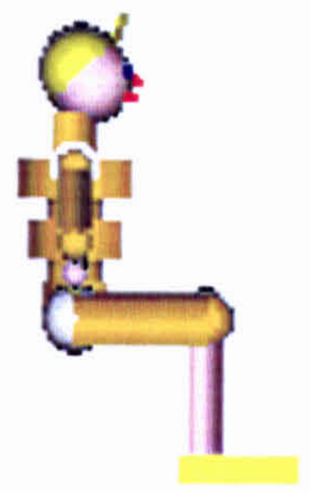
April 30, 1998

The purpose of this evaluation is to evaluate how good different techniques are at posing (or positioning) articulated figures (a humanoid in this case).

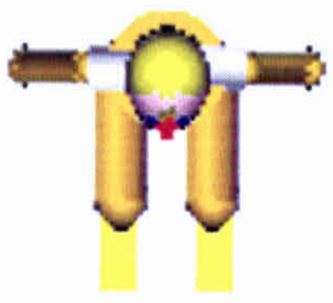
For this purpose, we are going to try to produce the following pose. It might represent someone sitting on some invisible chair with the hands pointing toward the pelvis.

The technique we are going to use is called forward kinematics.





Right

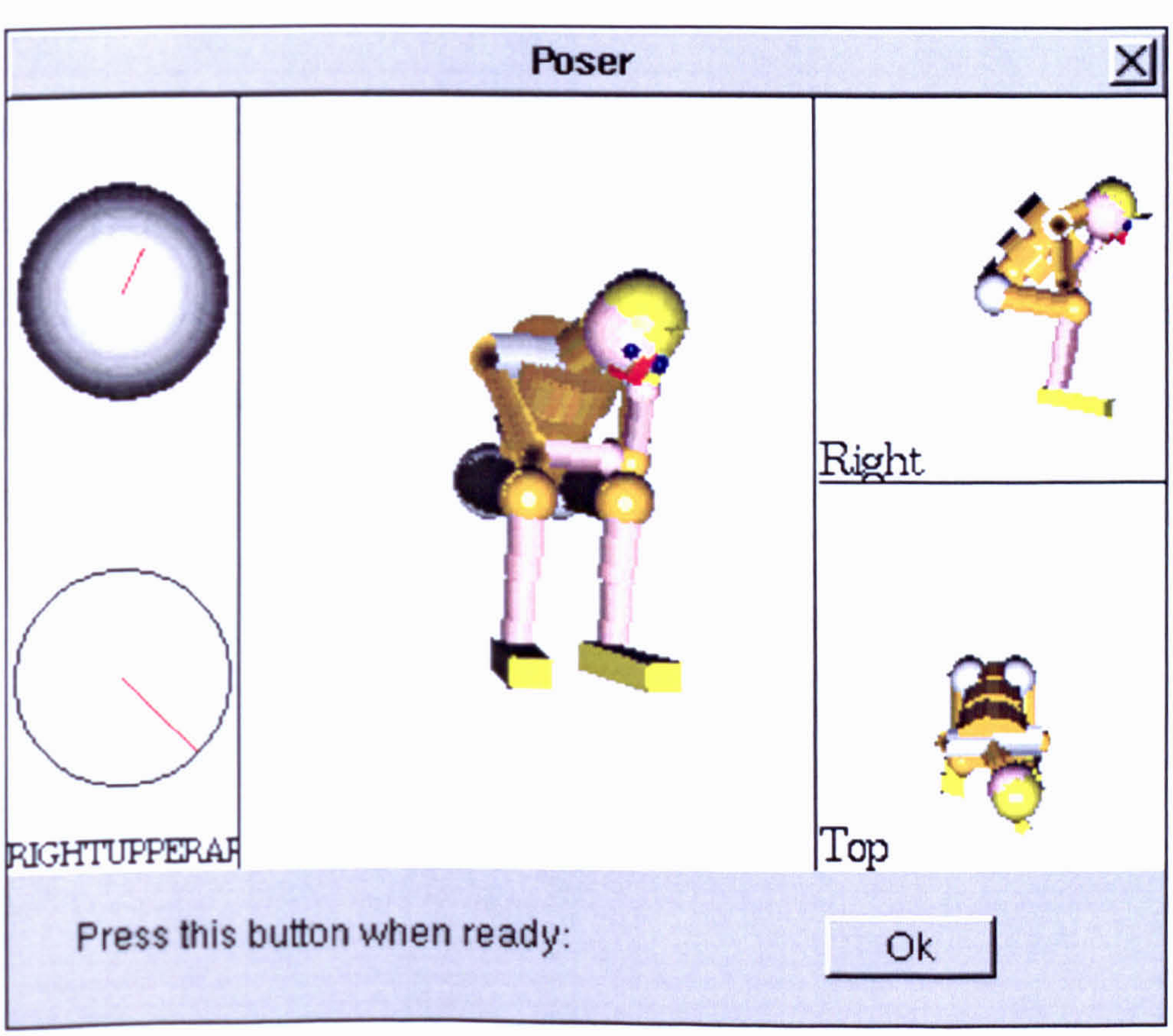


Top

Colours have been selected to help you. Apart from the head, they all mean something.

- ▷ **Grey:** The corresponding limb cannot move
- ▷ **Pink:** Only flexion motions are possible (cf forearms)
- ▷ **Yellow:** Flexion and pivot motions are possible. The joint resembles part of a sphere
- ▷ **Orange:** Flexion and Pivot plus Twist motions are possible. For Twist, the limb rotates around it self.

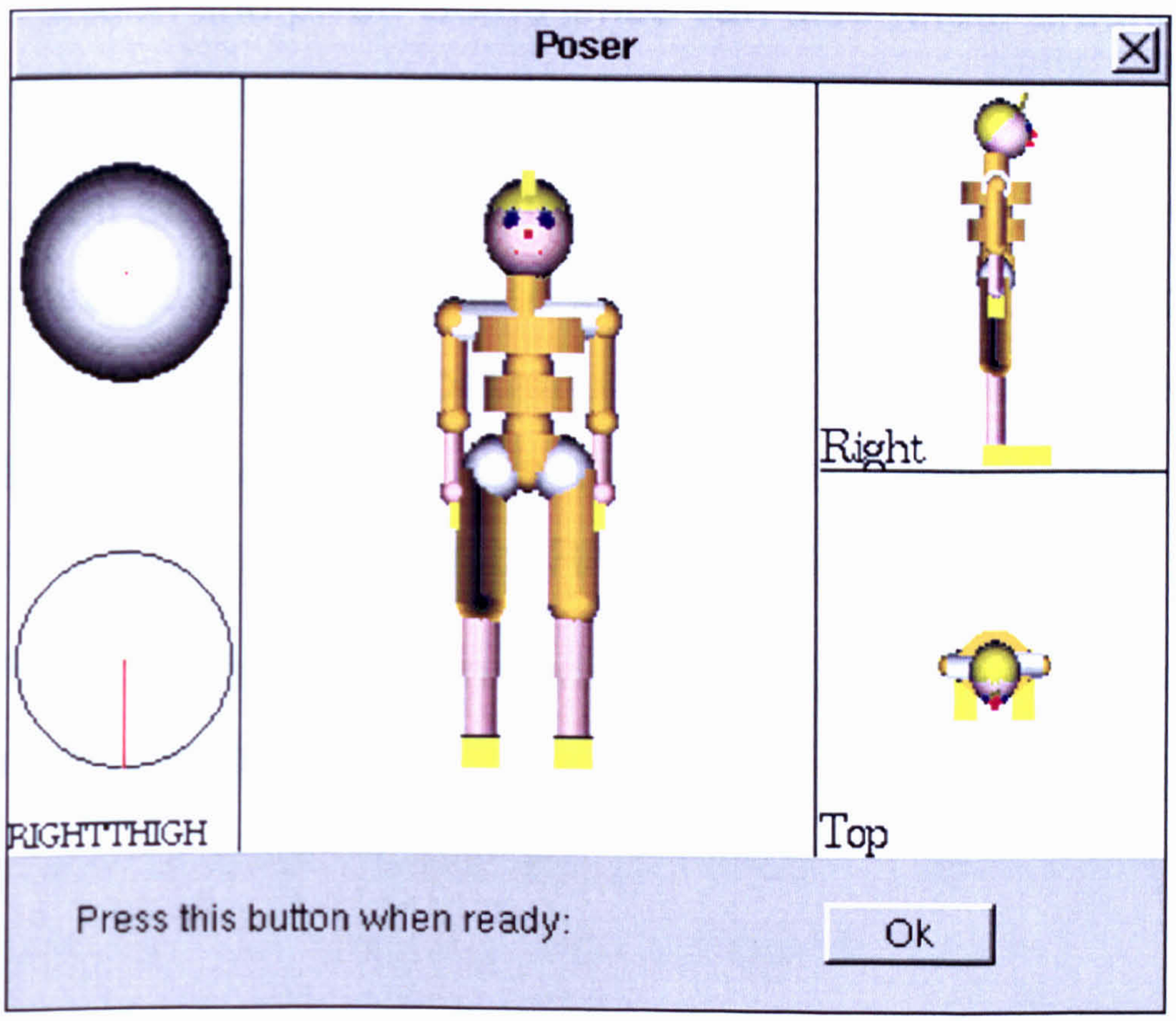
On the left of the screen, you should have a window called *Poser*. In this window, you should have a standing humanoid.



When you are ready, press on *Ok* at the bottom of this window. The computer then displays a pose that you will try to achieve. Take a careful lookat it. It is also displayed beside you on a sheet of paper. Ask me if you cannot find this sheet of paper. When you are ready, press *Ok*.

While you try to pose the figure, the computer will record all sort of informations which will enable me to work out how good the technique is. Try to produce the pose as fast as possible. When you will be close enough to it, the computer will tell you that it is fine and that you can stop.

To produce the required pose, we will start with the legs. Let's first select the right thigh of the humanoid (it is on your left). To do that, just move the mouse cursor to the thigh and double click the left mouse button. The thigh should highlight. You should end up with something like this:

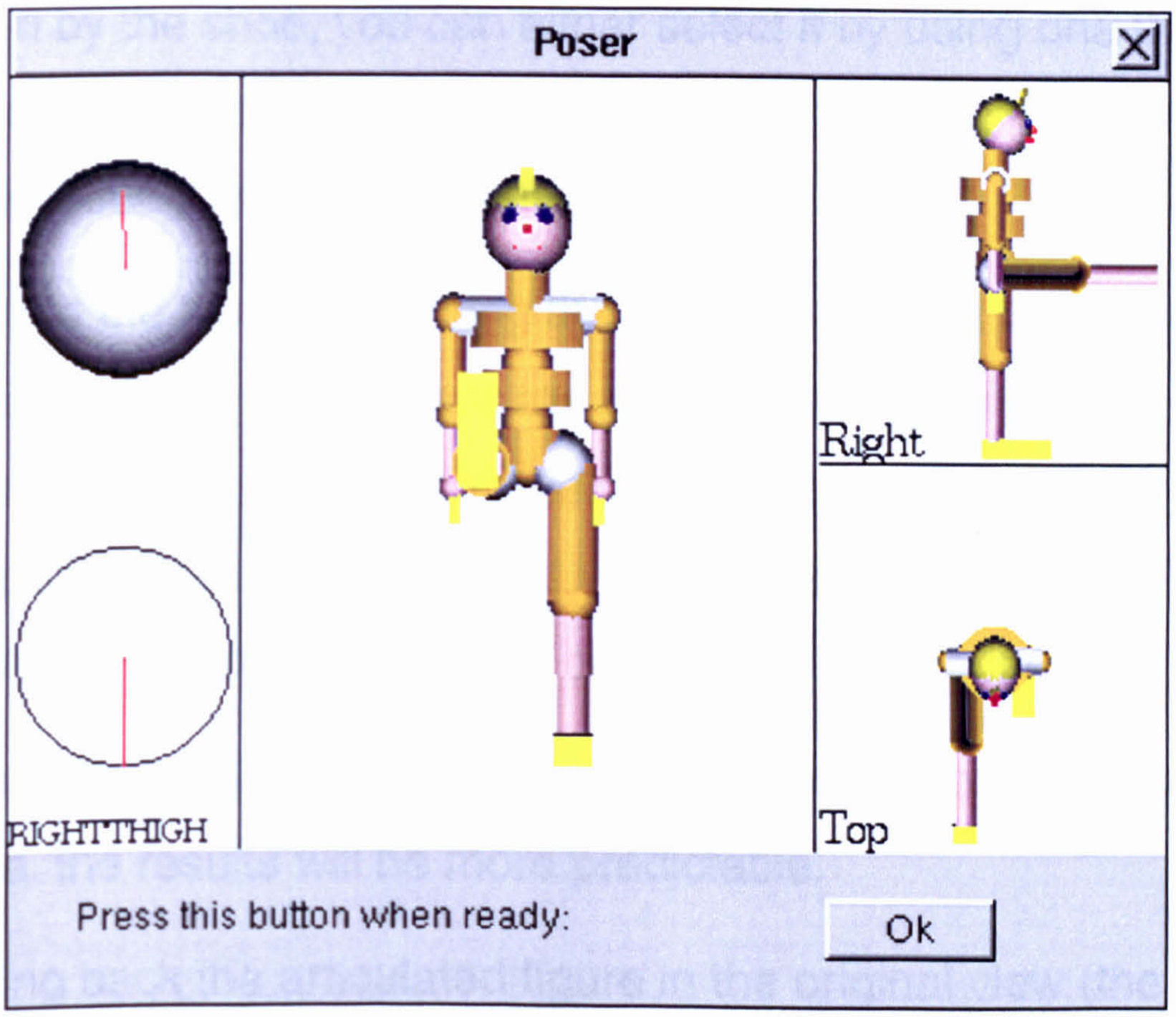


In animation, there are two types of motions. Limbs can be moved left to right and up to bottom. These types of motions are called flexion and pivot. Some limbs can also rotate around themselves. These are called twist motions.

On the left of the window, you can see two objects (a sphere and a circle). These are called joints balls. When you selected the thigh, the first object became a sphere with a red spot in the middle and in the second object, a line was drawn. At the bottom, the name of the limb (RIGHTTHIGH) was written.

The first object is for flexion and pivot motions whereas the second one is for twist motions.

Try now moving the red spot of the sphere upwards. You should see the right leg of the humanoid moving up as well. Try to place it so that it is horizontal and pointing towards you like this:



If you have made a mistake, you can undo the last operation by pressing *control-u*. Undoing twice just comes back to the original. Just try it out.

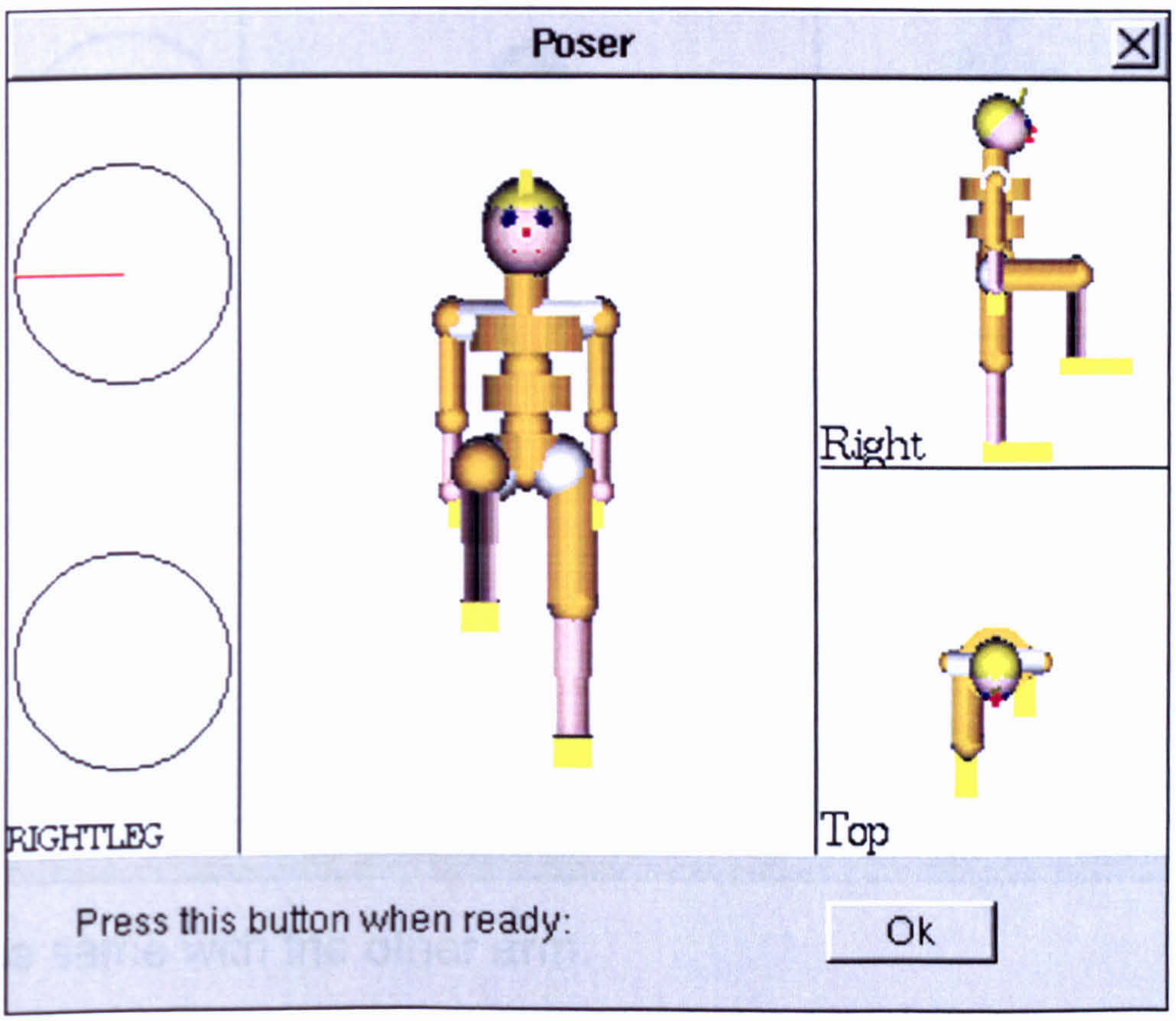
The next thing to do is to select the right leg. Because it is hidden by the shoe, you can either select it by using one the view on the right hand side of the window, or you can rotate the figure to see it from a different angle.

To rotate the humanoid, bring the mouse cursor at the level of the head. Once there, press the *control* key, the left mouse button and drag the mouse cursor downwards. This will rotate the figure. Once the right leg is visible, release the mouse button and the *control* key and double click on the leg.

If you have problems rotating the humanoid, try to go trough the centre, the results will be more predictable.

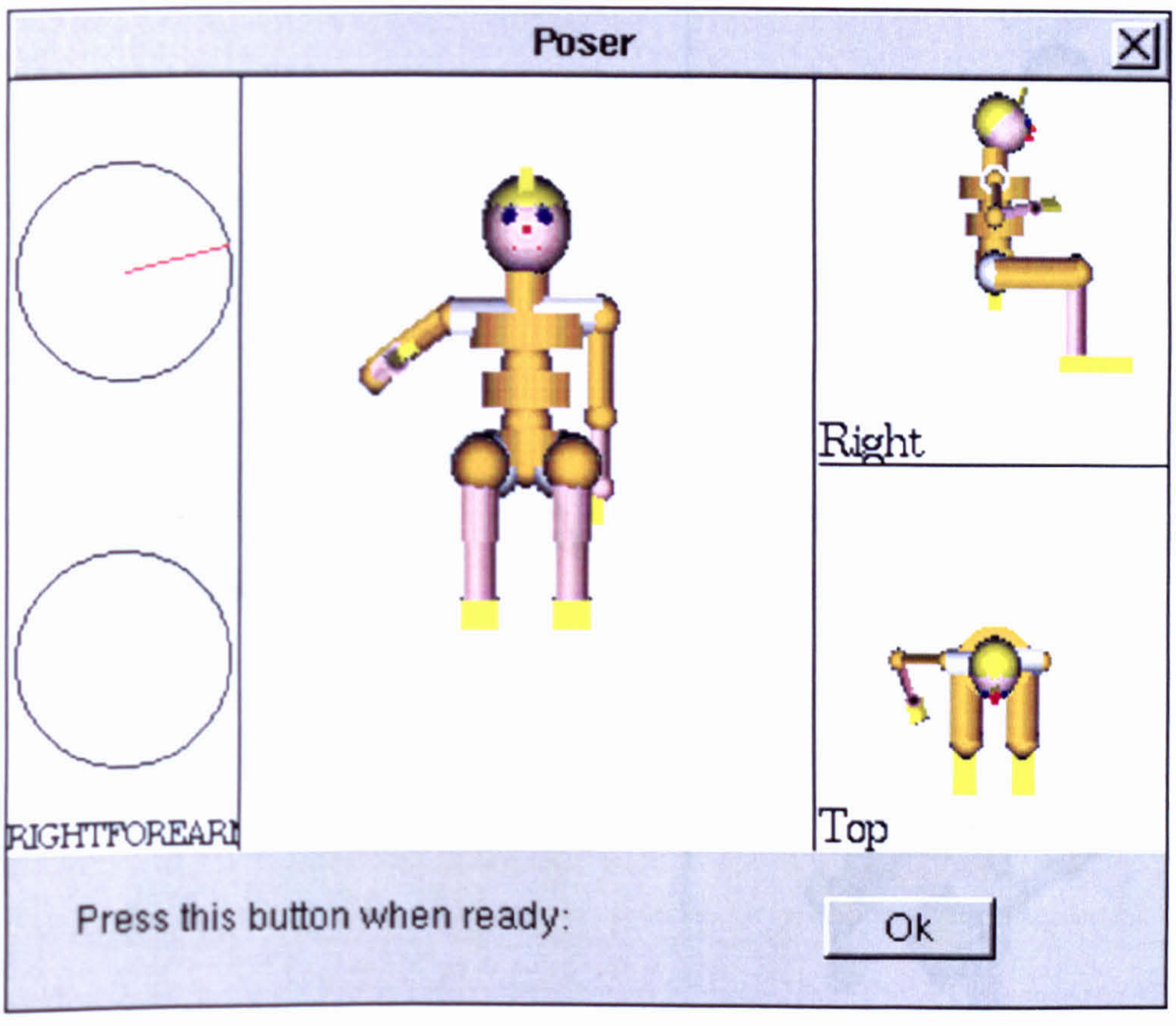
To bring back the articulated figure in the original view (the front view), press the *control* key again, and double click.

With the leg joint only, it is not possible to move left to right so pivot motions are disabled. As a result, only a circle with a red line appears at the top left of the window. Since twist is not allowed as well, there is no red line at the bottom. By moving the red line, try to bring the leg in a seated position so that it looks like this:



By now, you should be able to position the left leg in a similar position. I leave it as an exercise for you to do.

We now want to move the arms so that the hands points toward the hips. Working with the right arm, first select the upper arm and move it upwards, then select the forearm and bend it so that it looks like this:



Do the same with the other arm.

The next thing to do is to twist the upper arms so that the hands touch on the hip. The pose is now finished

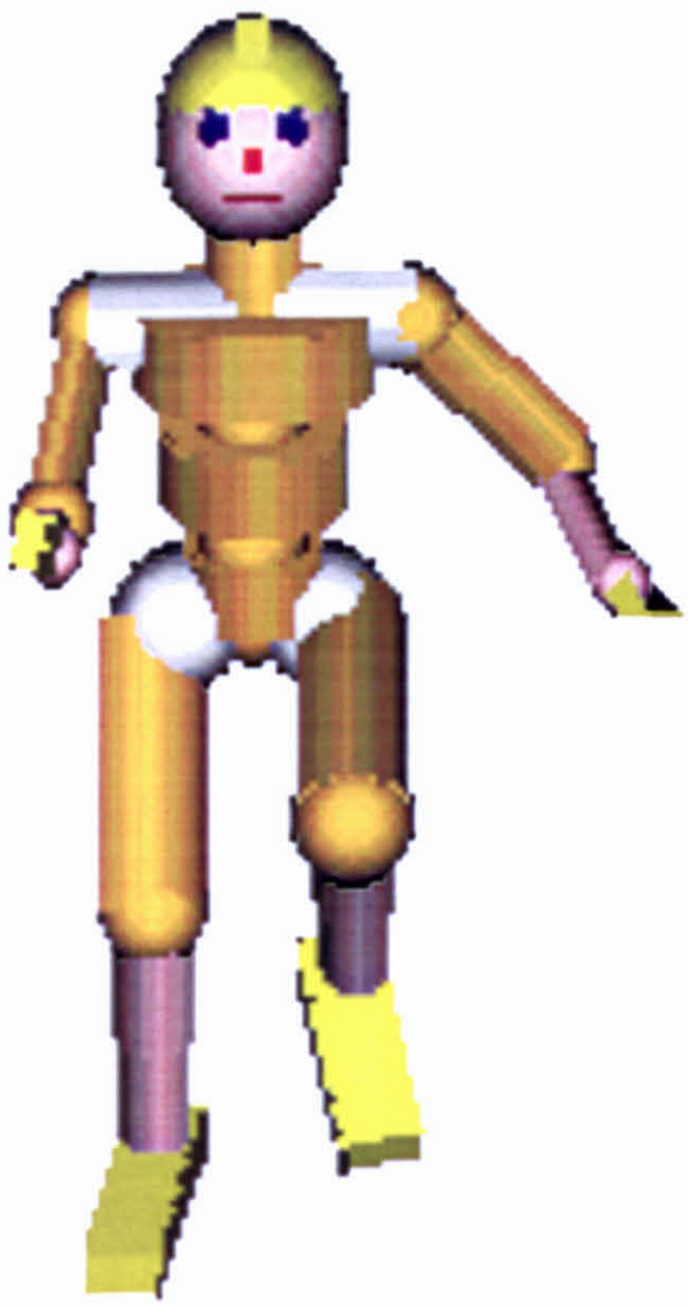
Before you go to the evaluation strictly speaking, you will try to produce the following pose. It represents a sportsman jumping over a barrier. I will be there to help you and answer your questions if any.

Right

Top

Now, so that I can evaluate how good the technique is, you will try to produce the following poses.

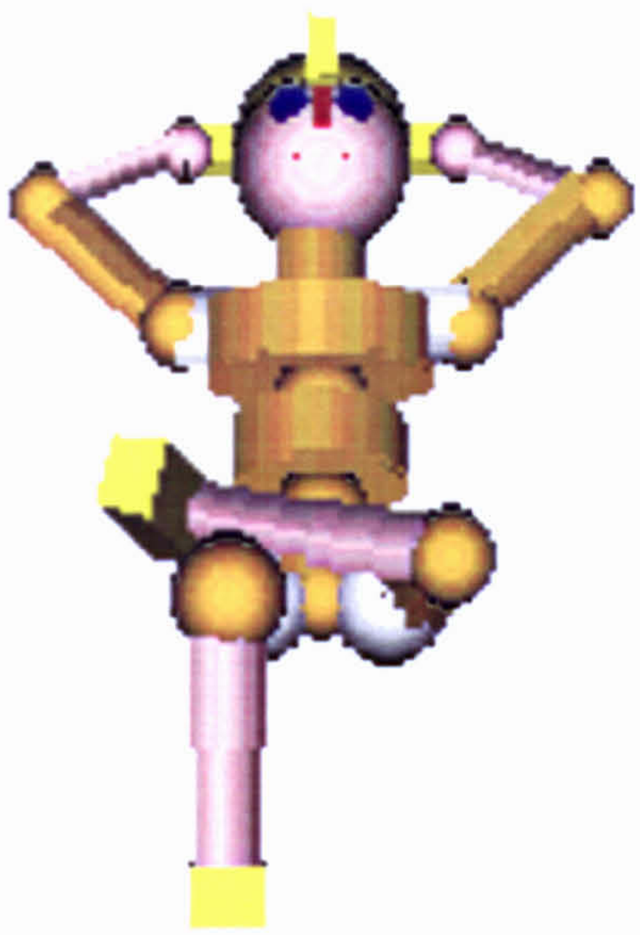
The first one represents a runner.



Right

Top

The second one represents someone lying on some invisible chair, the left leg resting on top of the other one, the hands at the back of the head.



Right

Top

4.3 Learning to use inverse kinematics

UNIVERSITY
of
GLASGOW

Computing Science

Posing a humanoid using inverse kinematics

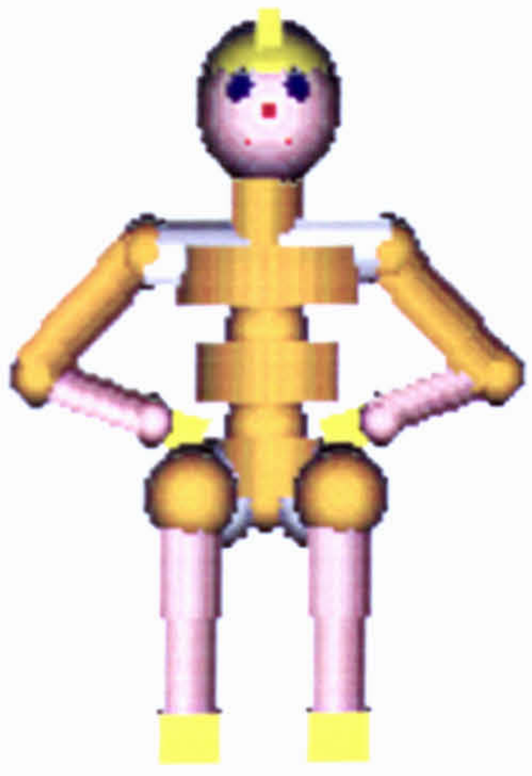
Stephane Etienne

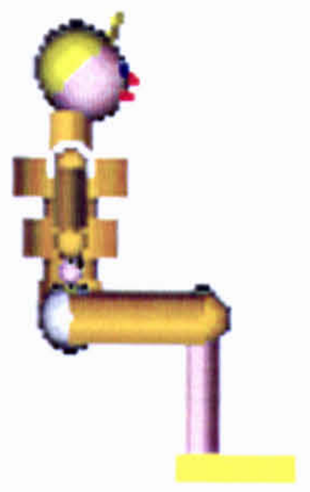
April 30, 1998

The purpose of this evaluation is to evaluate how good different techniques are at posing (or positioning) articulated figures (a humanoid in this case).

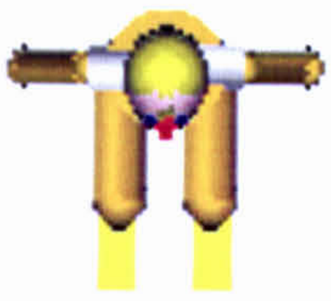
For this purpose, we are going to try to produce the following pose. It might represent someone sitting on some invisible chair with the hands pointing toward the pelvis.

The technique we are going to use is called inverse kinematics.





Right

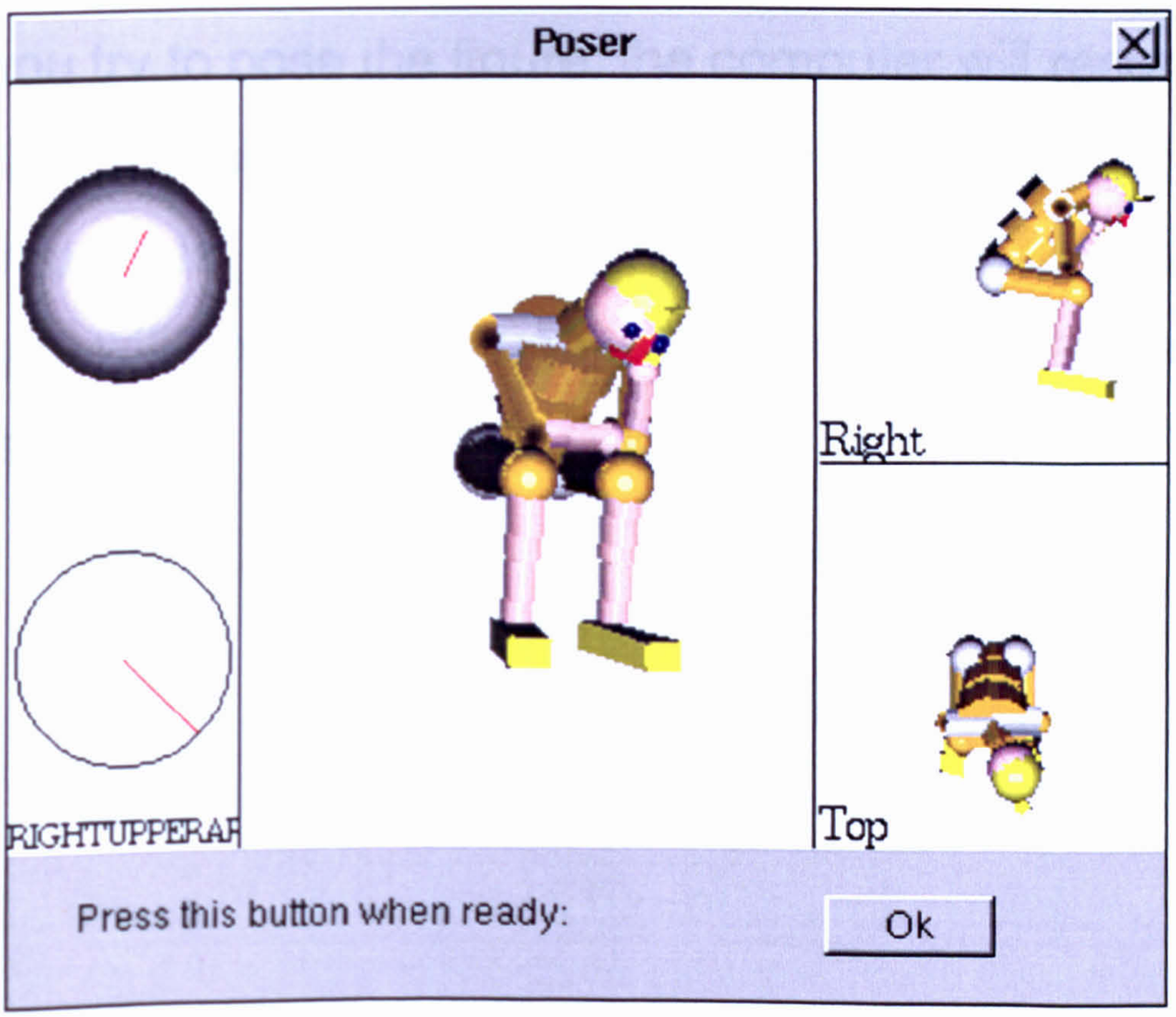


Top

Colours have been selected to help you. Apart from the head, they all mean something.

- **Grey:** The corresponding limb cannot move
- **Pink:** Only flexion motions are possible (cf forearms)
- **Yellow:** Flexion and pivot motions are possible. The joint resembles part of a sphere
- **Orange:** Flexion and Pivot plus Twist motions are possible. For Twist, the limb rotates around it self.

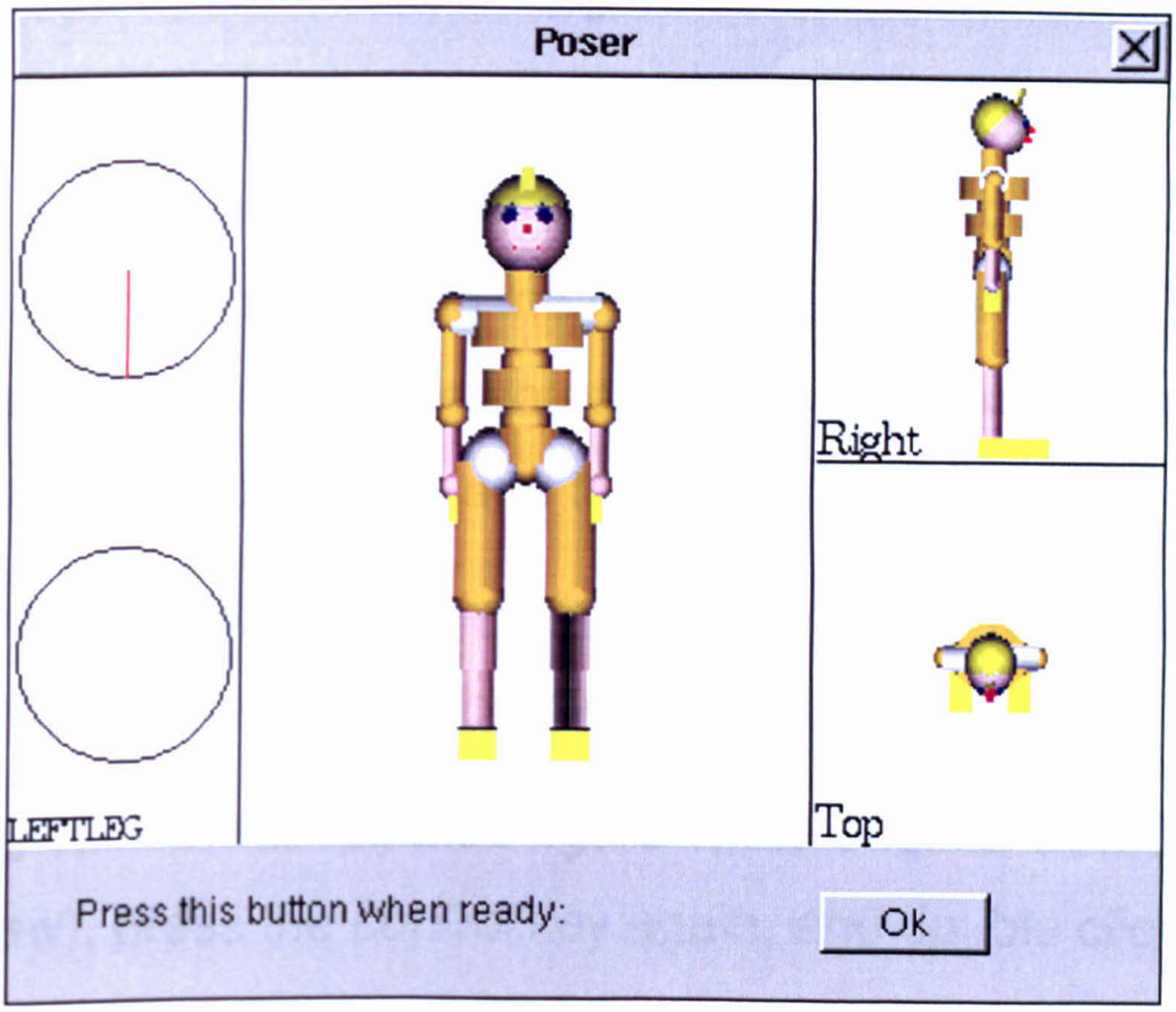
On the left of the screen, you should have a window called *Poser*. In this window, you should have the a standing humanoid.



When you are ready, press on *Ok* at the bottom of this window. The computer then displays a pose that you will try to achieve. Take a careful lookat it. It is also displayed beside you on a sheet of paper. Ask me if you cannot find this sheet of paper. When you are ready, press *Ok*.

While you try to pose the figure, the computer will record all sort of informations which will enable me to work out how good the technique is. Try to produce the pose as fast as possible. When you will be close enough to it, the computer will tell you that it is fine and that you can stop.

To produce the required pose, we will start with the left leg of the humanoid. The first thing you need to do is to select the left leg. For this purpose, bring the mouse cursor on top of the left leg and double click the left mouse button. The leg will highlight. The name of the limb will also be written at the bottom left of the window. The window should look like this:



Note if a limb you are trying to select points toward you, it may not work. You will have to rotate the figure or to use one of the views on the right hand side of the window to select it.

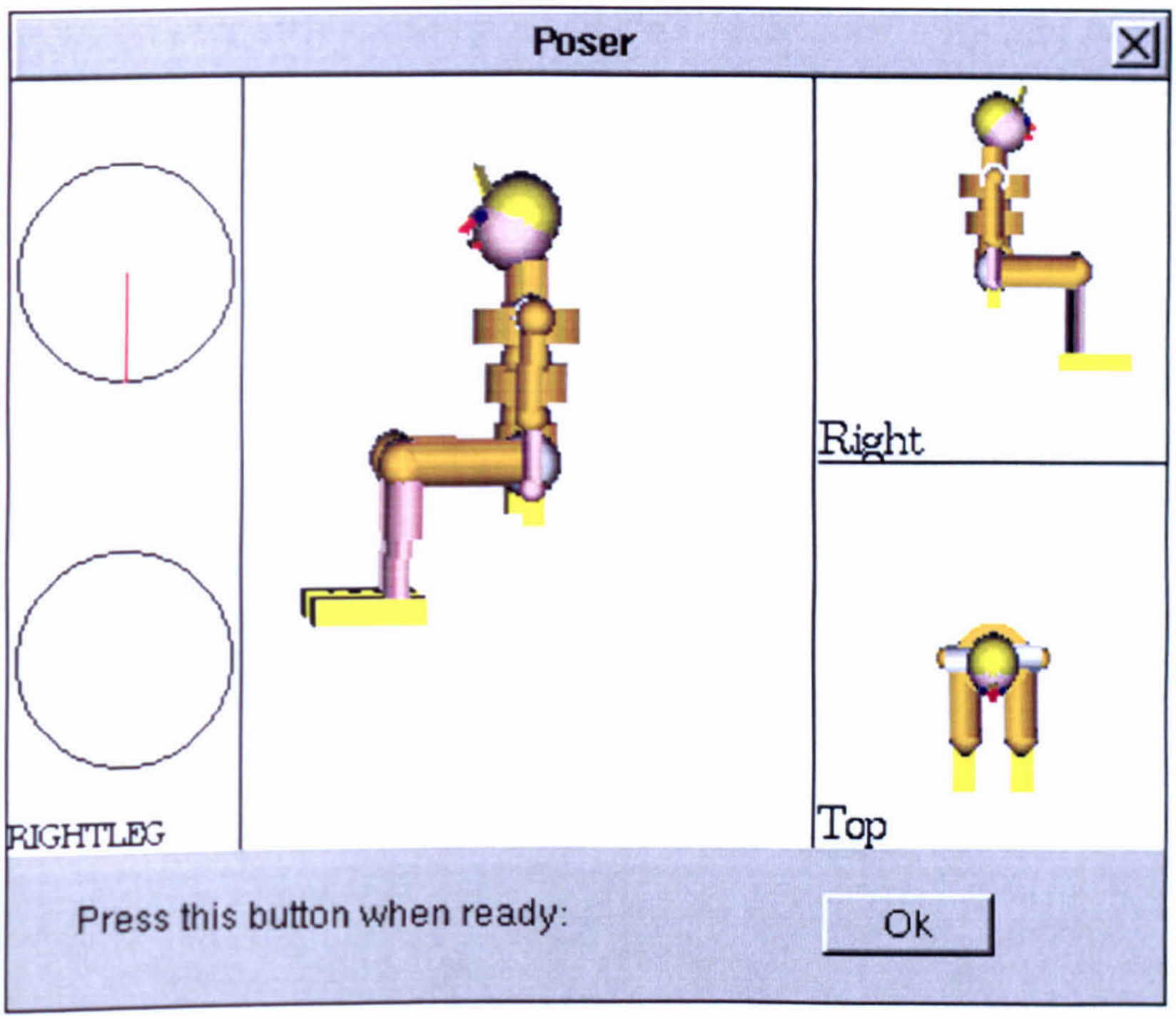
The next thing to do is to bring the leg in the seated position. Doing it with the humanoid facing you would be too difficult. Instead you should rotate the figure to see it from the side.

To rotate the humanoid, bring the mouse cursor at the level of the hip on the right side. Once there, press the *control* key, the left mouse button and drag the mouse cursor leftward. This will rotate the figure. Once you are in the right position, release the mouse button and the *control* key.

If you have problems rotating the humanoid, try to go trough the centre, the results will be more predictable.

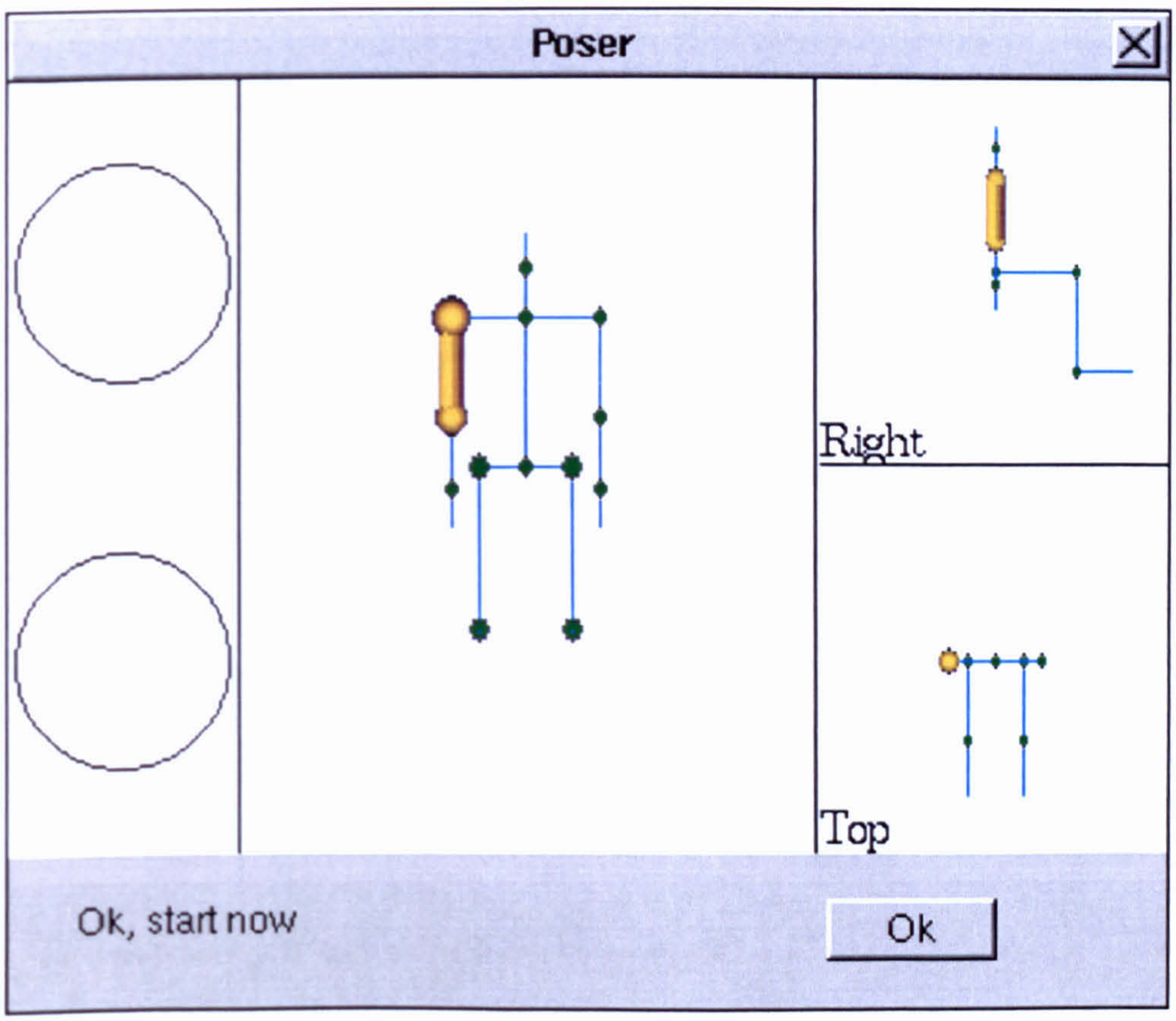
To bring back the articulated figure in the original view (the front view), press the *control* key again, and double click.

You can now move the leg by dragging it around with the mouse. You will see as the end of the leg tries to follows the mouse cursor, the thigh will move as necessary. The leg might slightly be tilted inwards or outwards. This does not matter. We will correct this defect later on. Try now moving the other leg so that it looks like this:



If you have made a mistake, you can undo the last operation by pressing *control u*. Undoing twice just comes back to the original. Just try it out. Coming back in the original view, bring the legs in the correct position.

Select now the left hand of the figure and try to move the entire arm so that the hand points toward the pelvis. You will probably find this impossible to do as it makes the torso move and the upper arm does not want to go in the position you want it to go. So, undo the changes by pressing *control u* and draw a rectangle around the upper arm while pressing the *shift* key. The window should look like this:

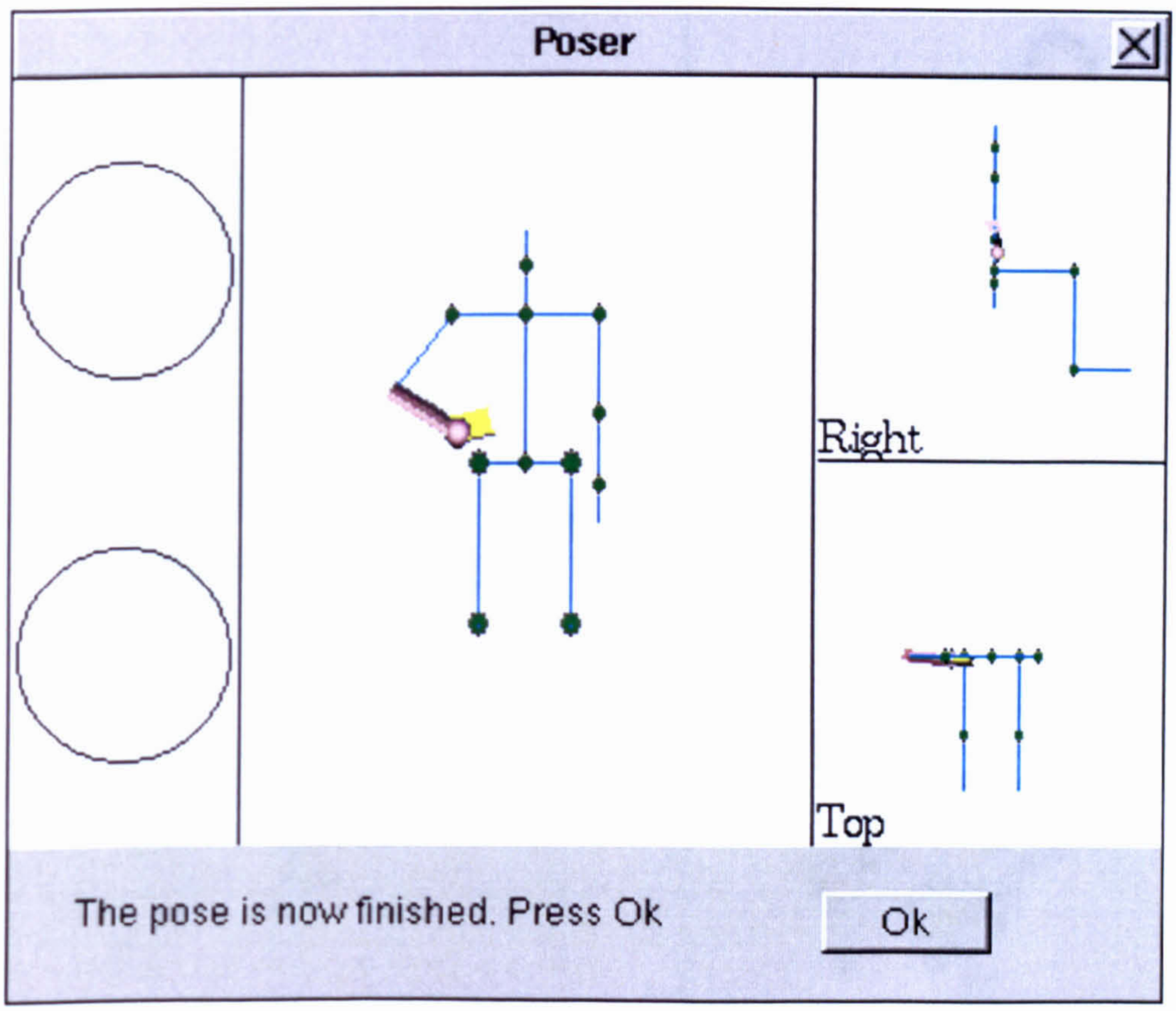


All the limbs which lie entirely inside the rectangle are enabled, the others are disabled. If no limb is enabled, the computer assumes that you want everything to be enabled. Thus, clicking only once will enable everything. You should now find it easier to pose the arm in the required position.

Once this is done, bring the upper arm in the right position.

Then, enable the forearm and the hand only, and move them by dragging the hand so that the hand points toward the hip.

You should end up with something like that:



Do the same thing with the other arm. The pose is not finished

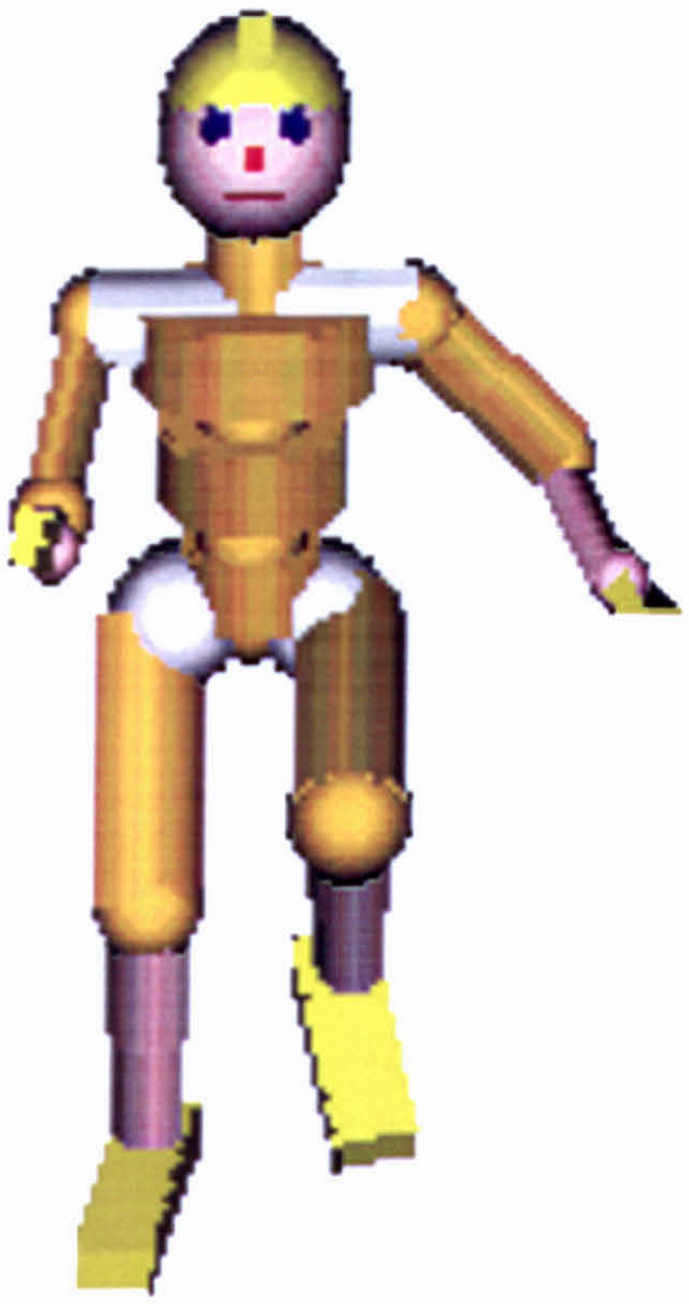
Before you go to the evaluation strictly speaking, you will try to produce the following pose. It represents a sportsman jumping over a barrier. I will be there to help you and answer your questions if any.

Right

Top

Now, so that I can evaluate how good the technique is, you will try to produce the following poses.

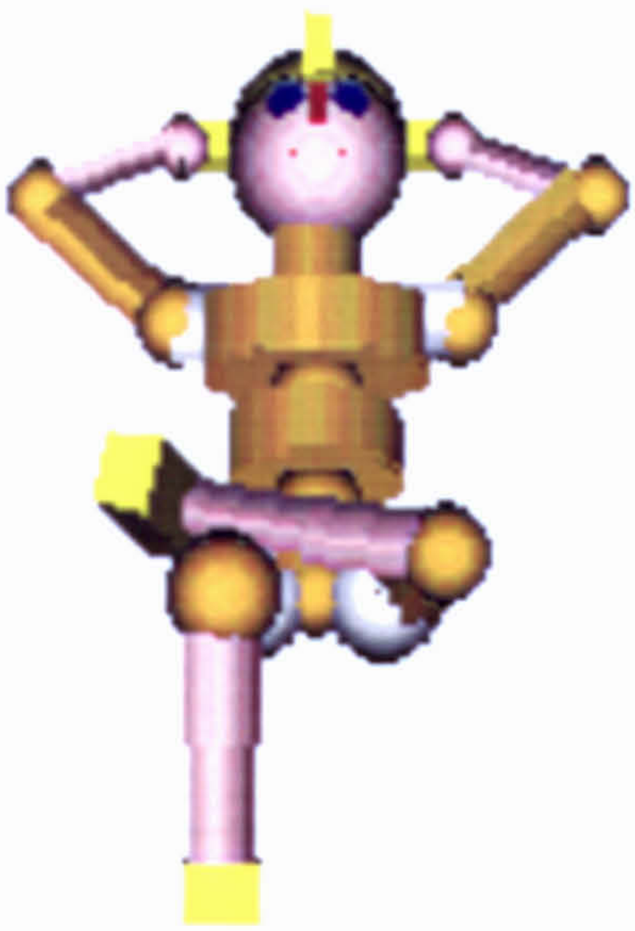
The first one represents a runner.



Right

Top

The second one represents someone lying on some invisible chair, the left leg resting on top of the other one, the hands at the back of the head.



Right

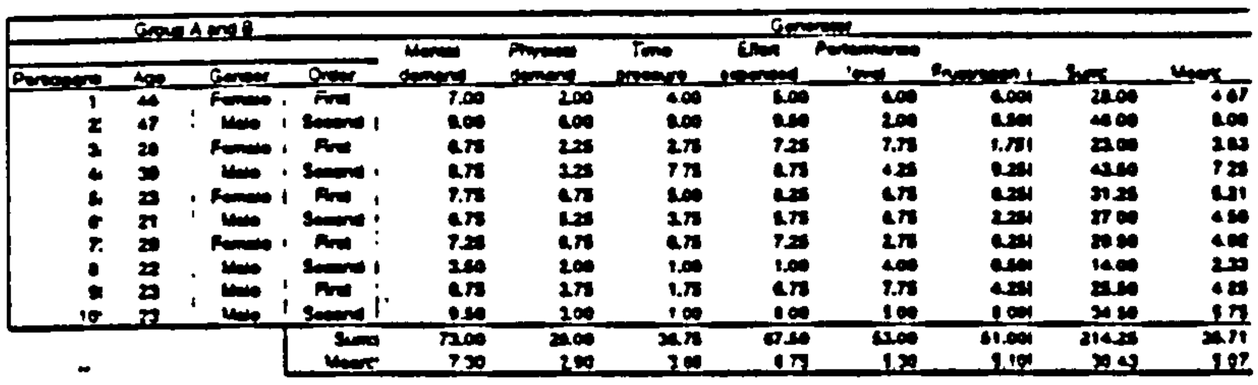
Top

Appendix D

Raw data and analysis

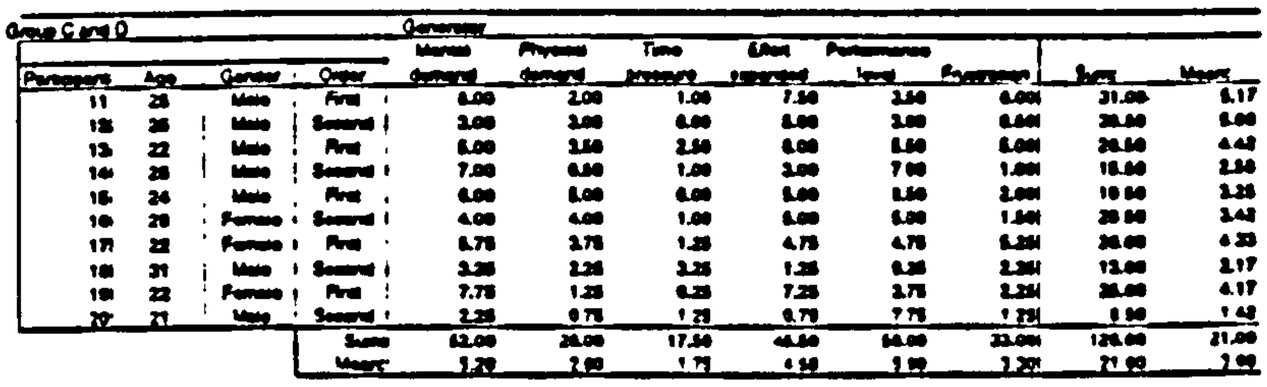
1 First evaluation

1.1 Workload raw data



Groups A and B				Generator							
Participants	Age	Gender	Order	Mental demand	Physical demand	Time pressure	Effort expended	Performance level	Frustration	Score	Mean
1	44	Female	First	7.00	2.00	4.00	5.00	6.00	6.00	28.00	4.67
2	47	Male	Second	9.00	6.00	8.00	9.50	2.00	8.50	46.00	8.00
3	28	Female	First	6.75	2.25	2.75	7.25	7.75	1.75	23.00	3.83
4	30	Male	Second	8.75	3.25	7.75	8.75	4.25	9.25	43.50	7.25
5	23	Female	First	7.75	6.75	5.00	8.25	6.75	6.25	31.25	5.21
6	21	Male	Second	6.75	5.25	3.75	5.75	6.75	2.25	27.00	4.50
7	29	Female	First	7.25	6.75	6.75	7.25	2.75	6.25	29.50	4.92
8	22	Male	Second	3.50	2.00	1.00	1.00	4.00	0.50	14.00	2.33
9	23	Male	First	6.75	3.75	1.75	6.75	7.75	4.25	25.50	4.25
10	23	Male	Second	0.50	2.00	1.00	0.00	1.00	0.00	24.50	3.75
			Sums	73.00	28.00	36.75	67.50	53.00	51.00	314.25	36.71
			Means	7.30	2.80	3.68	6.75	5.30	5.10	30.43	5.07

Groups A and B								
Mental demand	Physical demand	Time pressure	Effort expended	Performance level	Frustration	Score	Mean
4.00	6.00	7.50	7.50	5.00	6.00	30.00	6.50
7.00	2.00	8.00	7.00	3.00	8.00	36.00	6.33
6.75	2.25	2.75	8.25	6.75	1.25	22.50	3.75
6.75	3.75	1.25	6.25	6.25	7.25	29.00	4.83
6.25	6.75	5.00	6.75	6.75	6.75	38.75	6.13
5.75	3.25	1.75	2.25	2.75	6.75	21.00	3.50
6.75	6.75	6.75	8.75	6.25	7.75	41.50	6.92
6.00	1.50	1.50	4.00	5.50	1.00	19.50	3.00
3.75	7.25	3.75	7.25	9.25	5.75	25.50	4.75
0.50	3.00	2.00	1.00	3.00	4.00	22.50	3.75
60.50	43.50	41.25	58.00	58.50	47.50	301.25	50.54
5.05	4.35	4.13	5.80	5.85	4.75	30.13	4.98



Group C and D				Generator							
Participants	Age	Gender	Order	Mental demand	Physical demand	Time pressure	Effort expended	Performance level	Frustration	Score	Mean
11	25	Male	First	8.00	2.00	1.00	7.50	3.50	8.00	31.00	5.17
12	26	Male	Second	3.00	3.00	6.00	5.00	3.00	8.00	28.50	5.00
13	22	Male	First	5.00	3.50	2.50	6.00	5.50	5.00	26.50	4.42
14	28	Male	Second	7.00	0.50	1.00	3.00	7.00	1.00	19.50	3.50
15	24	Male	First	6.00	5.00	6.00	5.00	3.50	2.00	19.50	3.25
16	28	Female	Second	4.00	4.00	1.00	5.00	5.00	1.50	20.50	3.42
17	22	Female	First	6.75	3.75	1.25	4.75	4.75	5.25	26.00	4.33
18	31	Male	Second	3.25	2.25	3.25	1.25	0.25	2.25	13.00	2.17
19	22	Female	First	7.75	1.25	6.25	7.25	3.75	2.25	25.00	4.17
20	21	Male	Second	2.25	0.75	1.75	0.75	7.75	1.75	8.50	1.42
			Sums	62.00	28.00	17.50	48.50	56.00	33.00	126.00	21.00
			Means	3.20	2.80	1.75	4.50	5.60	3.30	21.00	3.50

Groups C and D								
Mental demand	Physical demand	Time pressure	Effort expended	Performance level	Frustration	Score	Mean
8.50	5.50	5.00	6.00	2.00	8.50	43.50	7.00
8.00	7.00	7.00	8.50	3.00	7.00	41.50	6.92
1.00	6.00	2.00	2.00	7.00	2.00	19.00	3.00
8.00	1.00	8.00	8.00	6.00	8.00	36.00	6.00
3.00	8.00	7.00	7.00	6.00	8.00	34.00	5.75
7.00	5.50	1.00	6.00	6.00	1.00	26.00	4.33
6.75	6.75	4.75	6.75	3.75	6.50	35.50	5.92
8.00	2.25	6.75	1.25	8.25	2.50	19.25	3.21
7.25	8.75	6.25	8.75	1.75	4.50	43.50	7.25
2.75	1.75	2.25	1.75	5.75	2.75	14.50	2.42
56.25	54.50	50.00	58.50	48.50	44.00	222.75	37.13
5.63	5.45	5.00	5.85	4.85	4.40	31.75	5.29

Figure D.1: Workload Data

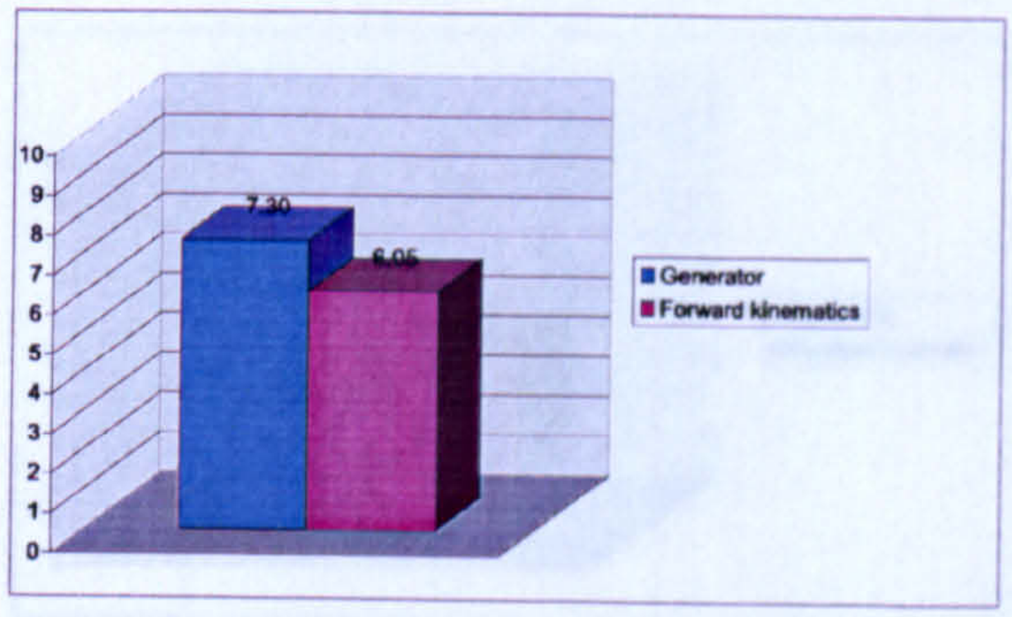
The first set of values is for the generator with forward kinematics. The second set if for the generator with inverse kinematics

Page missing

1.2 Mental demand

t-Test: Mental demand

	Generator	*Forward*
Mean	7.3	6.05
Variance	2.83055556	2.06666667
Observations	10	10
Pooled Variance	2.44861111	
Hypothesized Mean Difference	0	
df	18	
t Stat	1.78622066	
P(T<=t) one-tail	0.04545992	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.09091984	
t Critical two-tail	2.10092367	



t-Test: Mental demand

	Generator	*Inverse*
Mean	5.2	5.625
Variance	4.15	6.33680556
Observations	10	10
Pooled Variance	5.24340278	
Hypothesized Mean Difference	0	
df	18	
t Stat	-0.4150184	
P(T<=t) one-tail	0.34151662	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.68303324	
t Critical two-tail	2.10092367	

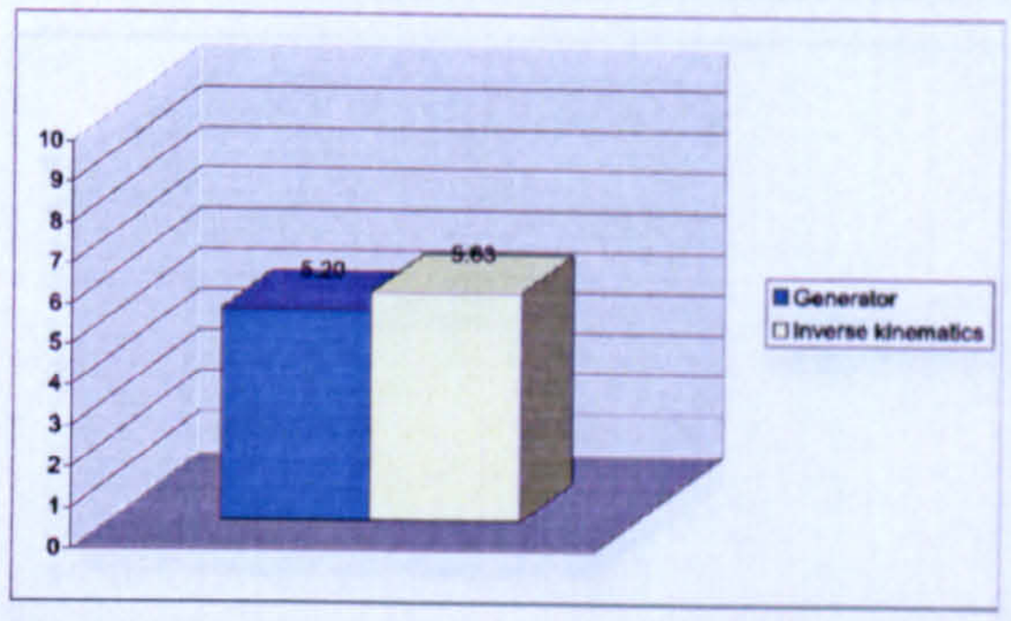


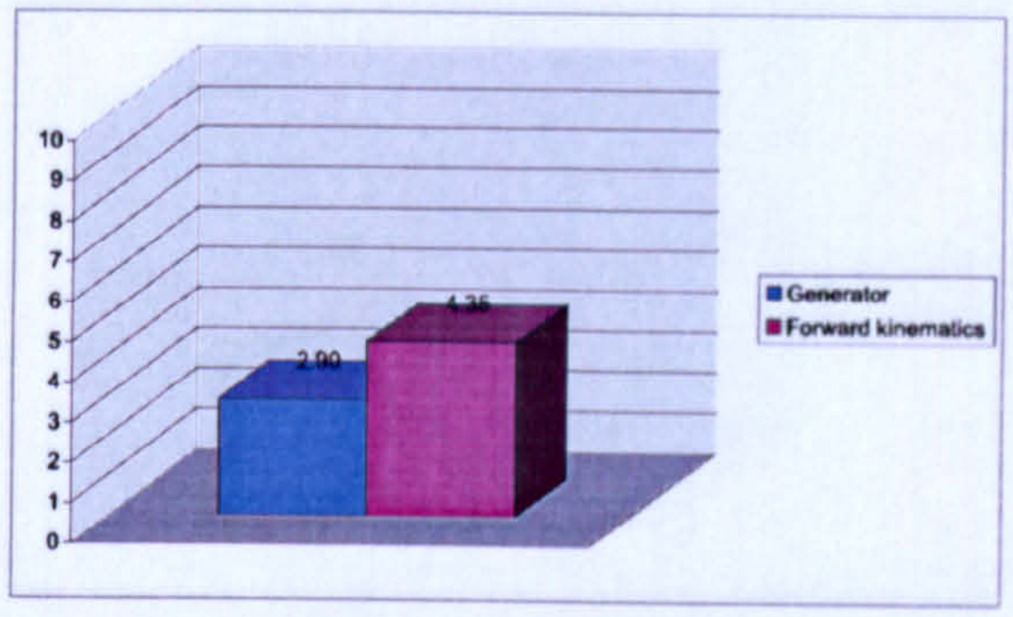
Figure D.2: Mental demand

Differences in the means are not statistically significant.

1.3 Physical demand

t-Test: Physical demand

	Generator	*Forward*
Mean	2.9	4.35
Variance	3.03055556	8.54444444
Observations	10	10
Pooled Variance	5.7875	
Hypothesized Mean Difference	0	
df	18	
t Stat	-1.3477443	
P(T<=t) one-tail	0.09723002	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.19446004	
t Critical two-tail	2.10092367	



t-Test: Physical demand

	Generator	*Inverse*
Mean	2.6	5.45
Variance	2.23888889	7.96944444
Observations	10	10
Pooled Variance	5.10416667	
Hypothesized Mean Difference	0	
df	18	
t Stat	-2.8207685	
P(T<=t) one-tail	0.00566053	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.01132106	
t Critical two-tail	2.10092367	

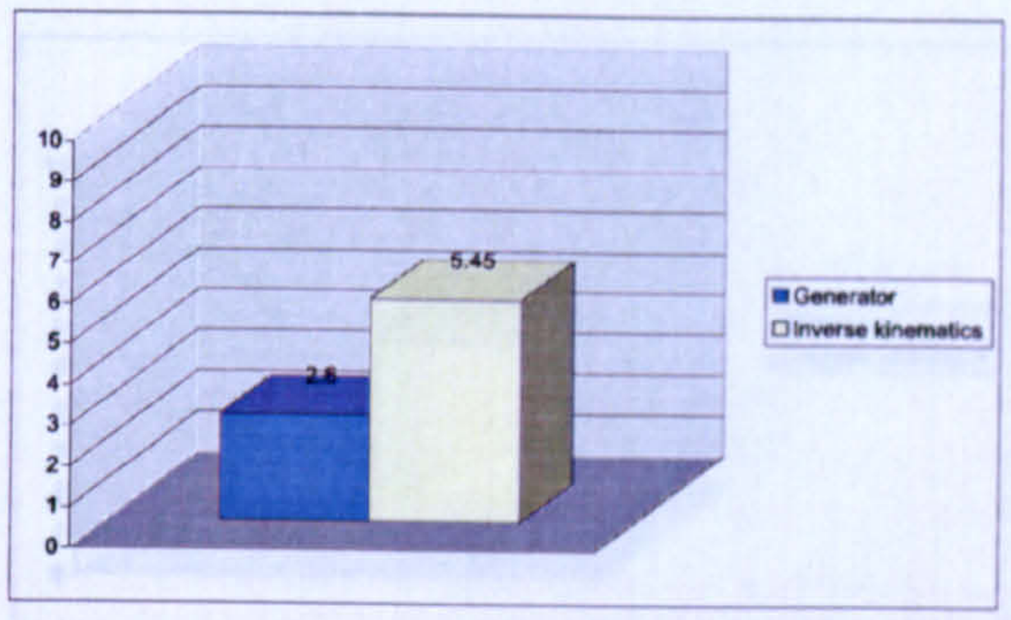


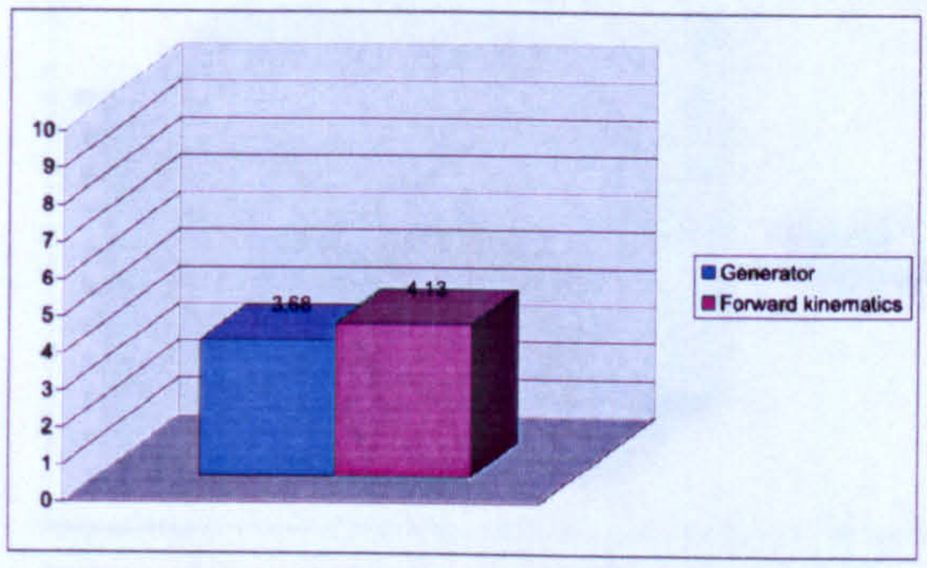
Figure D.3: Physical demand

Participants found that inverse kinematics were a lot more demanding physically than the generator was. This difference was statistically significant to the 5% level.

1.4 Time pressure

t-Test: Time pressure

	Variable 1	*Variable 2*
Mean	3.675	4.125
Variance	8.25069444	7.79513889
Observations	10	10
Pooled Variance	8.02291667	
Hypothesized Mean Difference	0	
df	18	
t Stat	-0.3552478	
P(T<=t) one-tail	0.36326635	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.72653269	
t Critical two-tail	2.10092367	



t-Test: Time pressure

	Variable 1	*Inverse*
Mean	1.75	5
Variance	3.15277778	6.52777778
Observations	10	10
Pooled Variance	4.84027778	
Hypothesized Mean Difference	0	
df	18	
t Stat	-3.3031875	
P(T<=t) one-tail	0.0019767	
t Critical one-tail	1.73406306	
P(T<=t) two-tail	0.00395339	
t Critical two-tail	2.10092367	

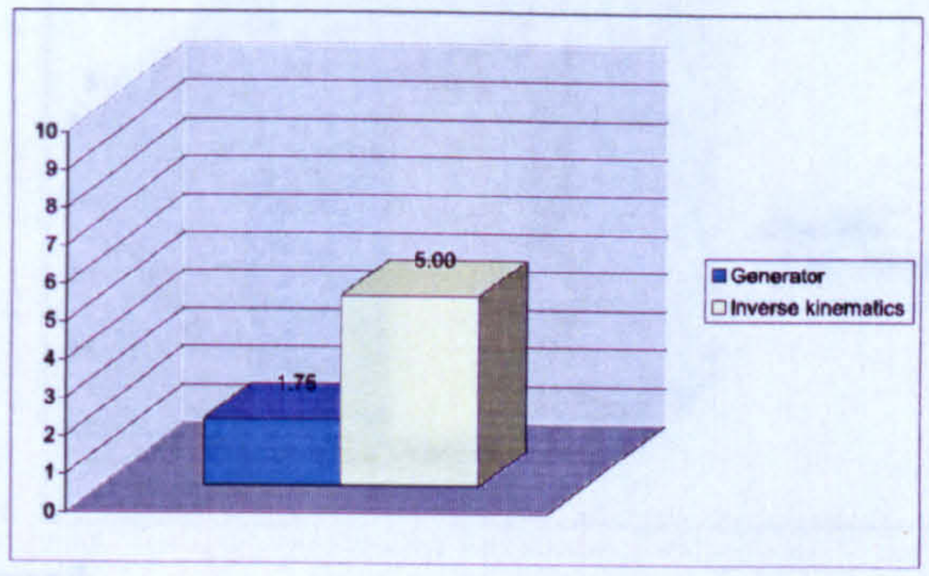


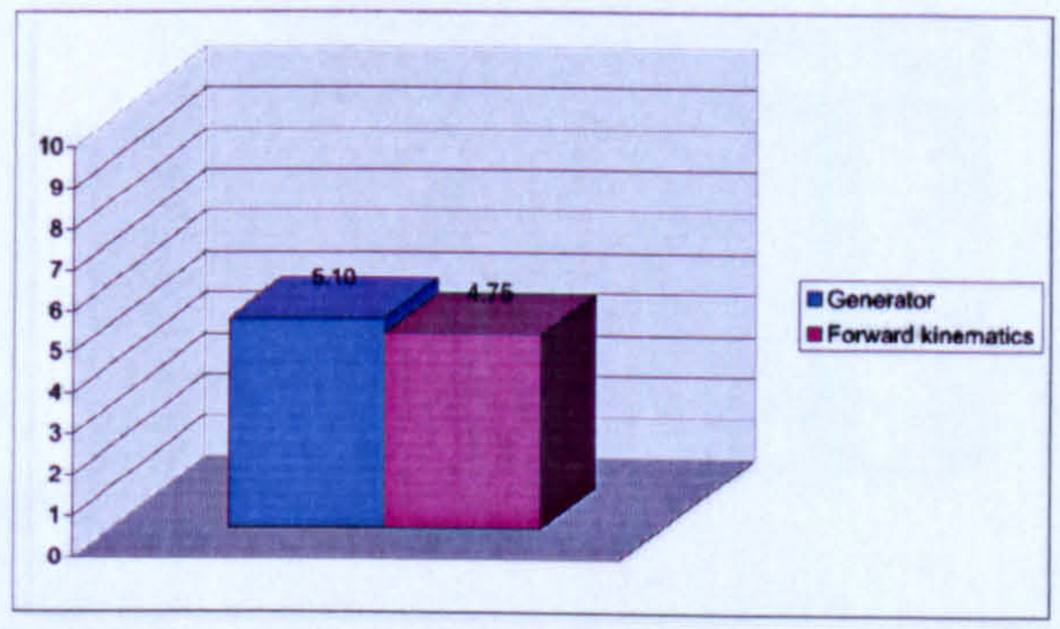
Figure D.4: Time pressure

The time pressure related to inverse kinematics was somewhat higher than the one related to the generator. This difference is statistically significant.

1.5 Frustration

t-Test: Frustration experienced

	Generator	*Forward*
Mean	5.1	4.75
Variance	8.030556	8.527778
Observations	10	10
Pooled Variance	8.279167	
Hypothesized Mean Difference	0	
df	18	
t Stat	0.271994	
P(T<=t) one-tail	0.394361	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.788722	
t Critical two-tail	2.100924	



t-Test: Frustration experienced

	Generator	*Inverse*
Mean	3.3	4.6
Variance	4.538889	4.766667
Observations	10	10
Pooled Variance	4.652778	
Hypothesized Mean Difference	0	
df	18	
t Stat	-1.347635	
P(T<=t) one-tail	0.097247	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.194495	
t Critical two-tail	2.100924	

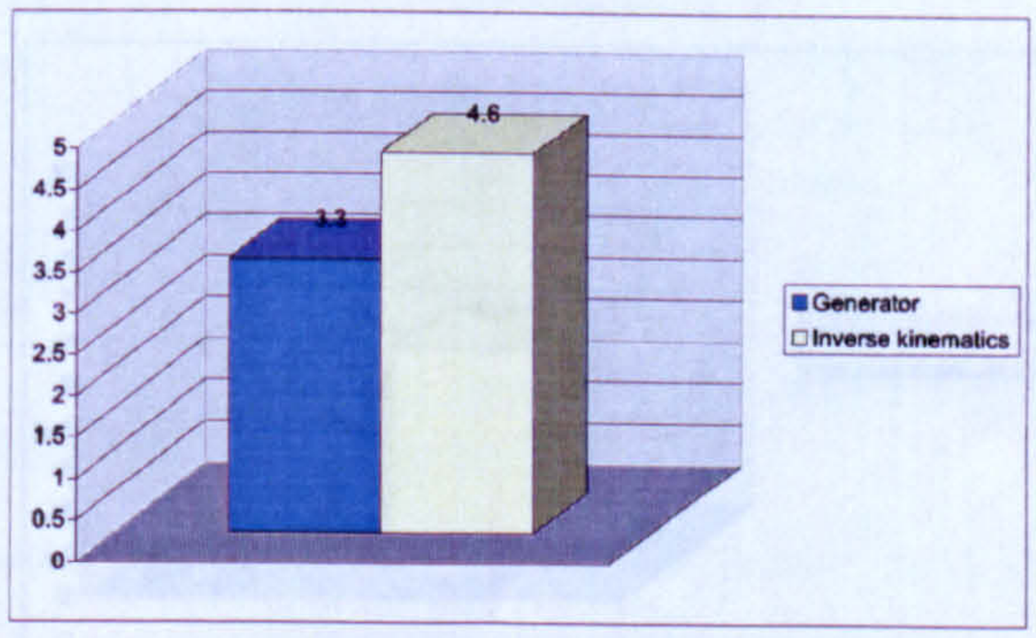


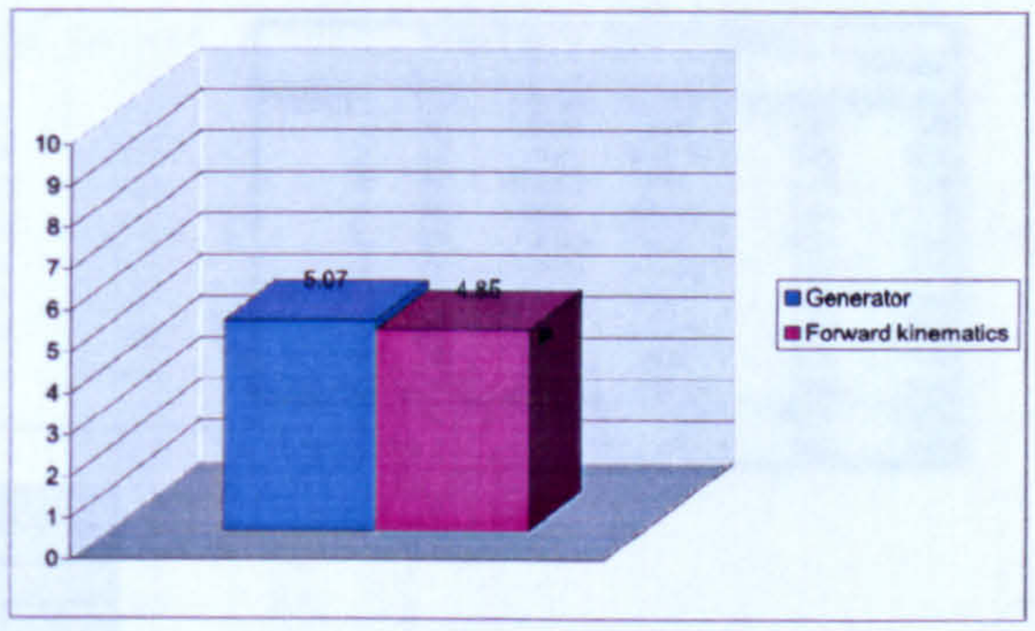
Figure D.5: Frustration experienced

Differences in the means are not statistically significant.

1.7 Workload

t-Test: Workload

	Generator	*Forward*
Mean	5.070833	4.854167
Variance	2.669001	1.849248
Observations	10	10
Pooled Variance	2.259124	
Hypothesized Mean Difference	0	
df	18	
t Stat	0.322335	
P(T<=t) one-tail	0.375456	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.750913	
t Critical two-tail	2.100924	



t-Test: Workload

	Generator	*Inverse*
Mean	3.6	5.2125
Variance	1.576852	3.157581
Observations	10	10
Pooled Variance	2.367216	
Hypothesized Mean Difference	0	
df	18	
t Stat	-2.343504	
P(T<=t) one-tail	0.015393	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.030785	
t Critical two-tail	2.100924	

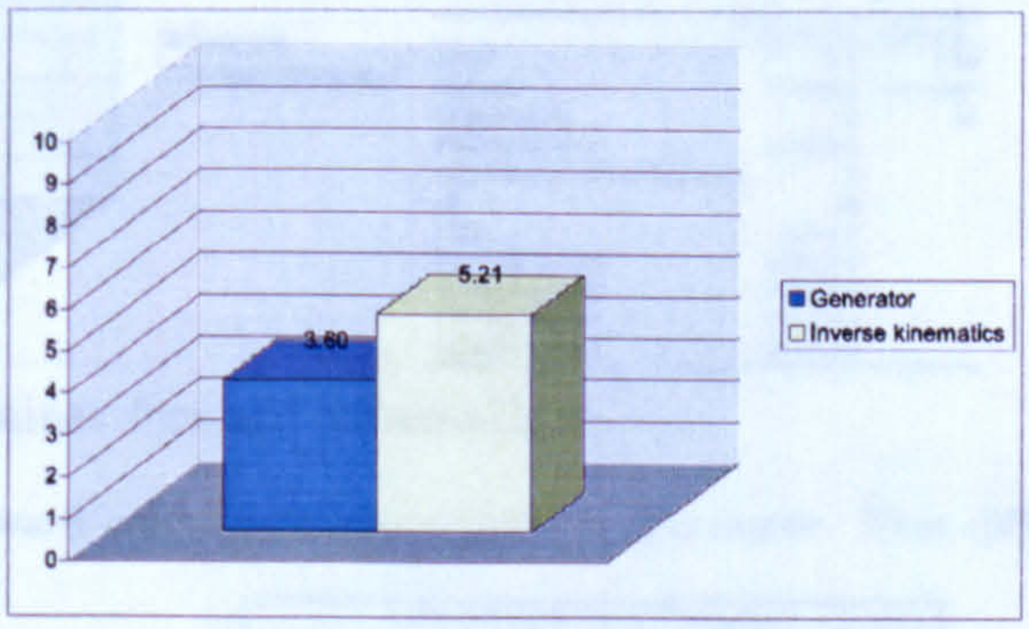
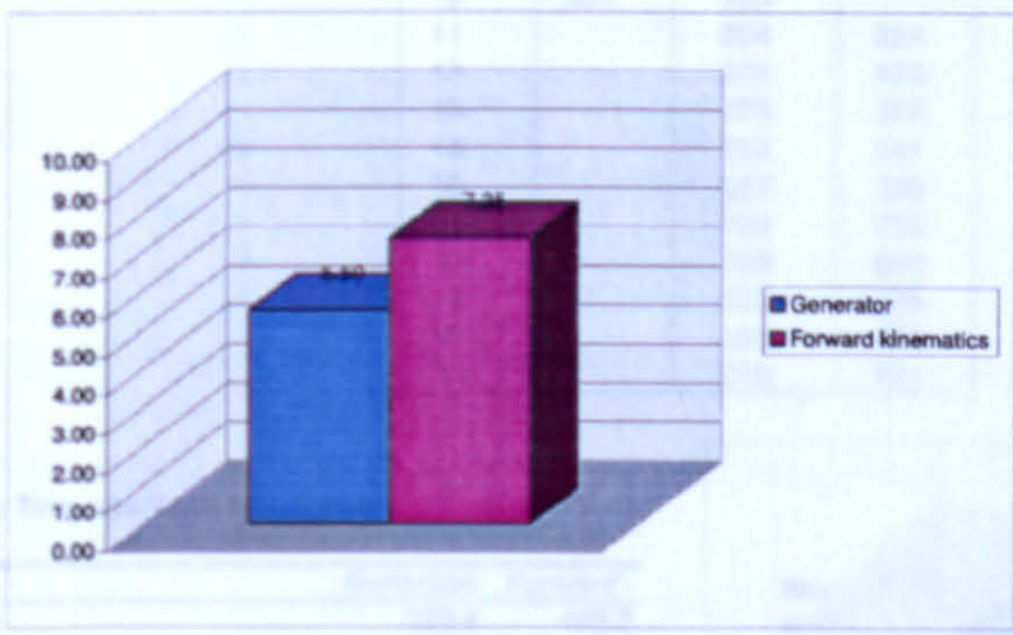


Figure D.7: Workload

Participants found inverse kinematics somewhat more demanding to use than the generator. This difference was statistically different.

1.8 Overall preference

Group A and B - Overall Preference					
					Forward
Participant	Age	Gender	Order	Generator	Kinematics
1	44	Female	First	6.00	7.00
2	47	Male	Second	4.00	6.00
3	28	Female	First	6.75	8.25
4	39	Male	Second	2.25	7.75
5	23	Female	First	2.75	7.25
6	21	Male	Second	7.25	4.25
7	29	Male	First	6.25	8.25
8	22	Male	Second	9.50	9.00
9	23	Male	First	5.75	8.75
10	23	Male	Second	4.50	7.00
			Sum:	55.00	73.50
			Mean:	5.50	7.35



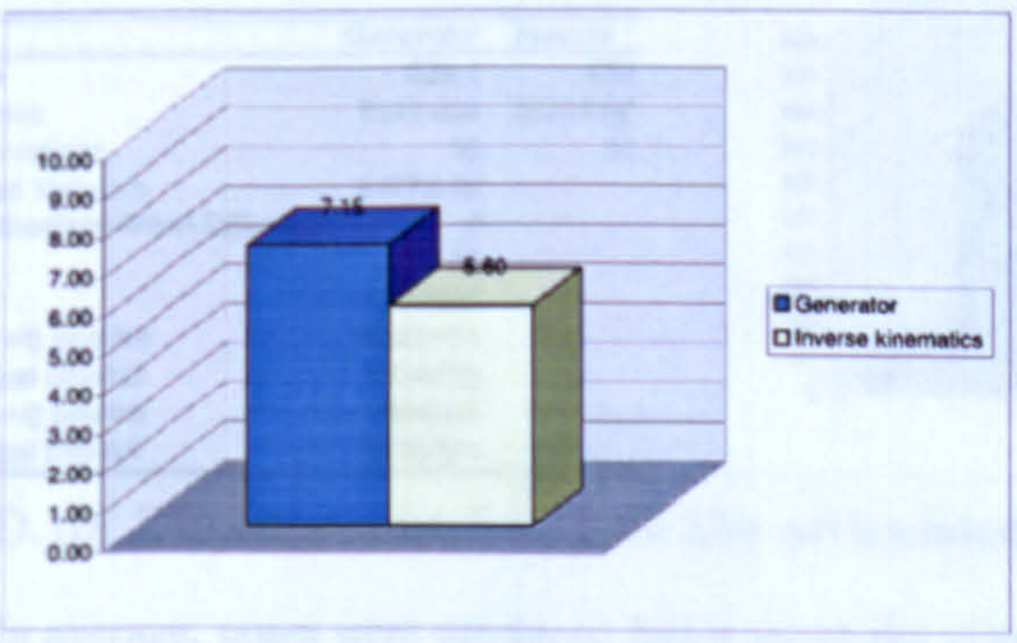
t-Test: Overall preference

	Generator	*Forward*
Mean	5.5	7.35
Variance	4.763889	2.016667
Observations	10	10
Pooled Variance	3.390278	
Hypothesized Mean Differenc	0	
df	18	
t Stat	-2.24667	
P(T<=t) one-tail	0.018722	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.037445	
t Critical two-tail	2.100924	

Figure D.8: Overall preference against forward kinematics

Participants preferred to use forward kinematics rather than the generator. That difference was statistically significant.

Group A and B - Overall Preference					
					Inverse
Participant	Age	Gender	Order	Generator	Kinematics
11	25	Male	First	7.00	4.00
12	25	Male	Second	5.50	8.50
13	22	Male	First	6.00	7.00
14	28	Male	Second	9.00	1.00
15	24	Male	First	5.00	8.50
16	28	Female	Second	7.50	5.50
17	22	Female	First	6.75	4.75
18	31	Male	Second	7.25	3.25
19	22	Female	First	8.75	6.75
20	21	Male	Second	8.75	6.75
			Sum:	71.50	56.00
			Mean:	7.15	5.60



t-Test: Overall preference

	Generator	*Inverse*
Mean	7.15	5.6
Variance	1.947222	5.711111
Observations	10	10
Pooled Variance	3.829167	
Hypothesized Mean Differenc	0	
df	18	
t Stat	1.771188	
P(T<=t) one-tail	0.04673	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.09346	
t Critical two-tail	2.100924	

Figure D.9: Overall preference against inverse kinematics

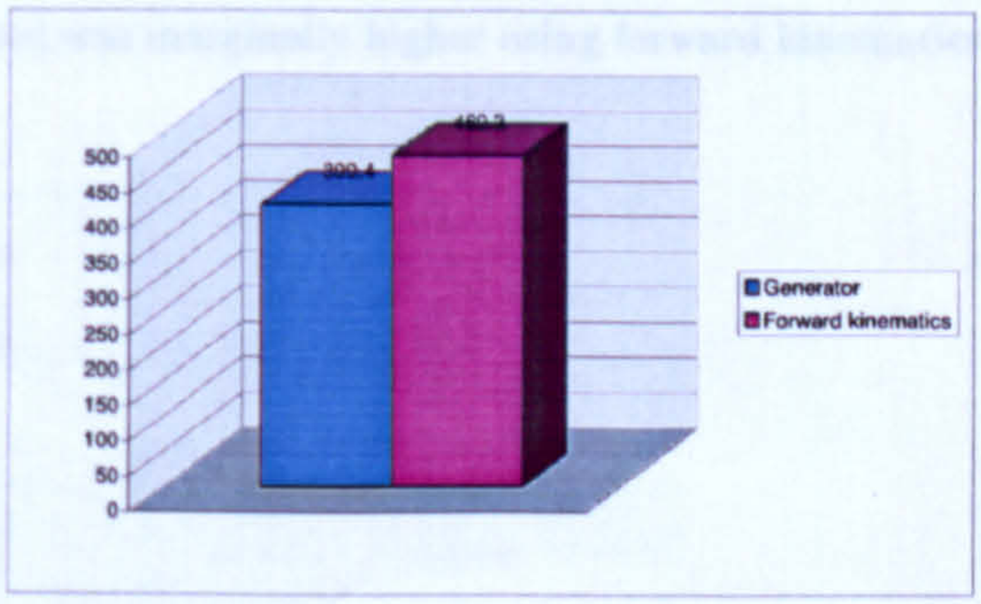
Participants preferred to use the generator rather than the inverse kinematics but that difference was not statistically significant.

1.9 Timings

	Forward	Generator	Inverse
1	394	654	
2	692	715	
3	363	220	
4	808	318	
5	587	404	
6	395	293	
7	469	317	
8	348	198	
9	336	455	
10	301	420	
11		394	824
12		276	433
13		533	253
14		318	341
15		357	339
16		423	235
17		255	626
18		325	525
19		234	453
20		276	721

t-Test: Time spent posing a figure

	Generator	*Forward*
Mean	399.4	469.3
Variance	29493.82	29051.57
Observations	10	10
Pooled Variance	29272.69	
Hypothesized Mean Differenc	0	
df	18	
t Stat	-0.913547	
P(T<=t) one-tail	0.186515	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.37303	
t Critical two-tail	2.100924	



t-Test: Time spent posing a figure

	Generator	*Inverse*
Mean	339.1	475
Variance	8299.656	39244.67
Observations	10	10
Pooled Variance	23772.16	
Hypothesized Mean Differenc	0	
df	18	
t Stat	-1.970925	
P(T<=t) one-tail	0.032154	
t Critical one-tail	1.734063	
P(T<=t) two-tail	0.064308	
t Critical two-tail	2.100924	

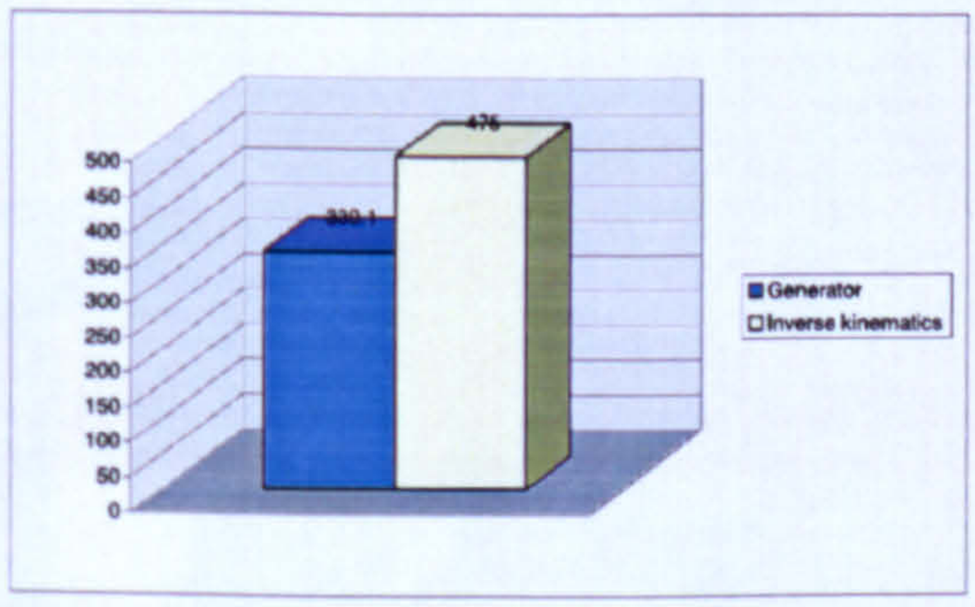


Figure D.10: Times required to pose the articulated figure

On average, poses were produced faster using the generator but differences in the means were not statistically significant.

1.10 Number of iterations

	Forward	Generator Inverse
1	27	22
2	23	47
3	20	14
4	38	26
5	20	21
6	17	22
7	23	32
8	28	17
9	25	30
10	14	29
11		27 37
12		24 18
13		22 18
14		33 16
15		36 15
16		13 24
17		18 15
18		15 19
19		15 26

t-Test: Two-Sample Assuming Equal Variances

	Generator	*Forward*
Mean	26	23.5
Variance	87.11111	44.72222
Observatio	10	10
Pooled Var	65.91667	
Hypothesiz	0	
df	18	
t Stat	0.688537	
P(T<=t) on	0.249947	
t Critical or	1.734063	
P(T<=t) tw	0.499893	
t Critical tw	2.100924	

t-Test: Two-Sample Assuming Equal Variances

	Generator	*Inverse*
Mean	22.55556	20.88889
Variance	67.27778	51.11111
Observatio	9	9
Pooled Var	59.19444	
Hypothesiz	0	
df	16	
t Stat	0.459531	
P(T<=t) on	0.326018	
t Critical or	1.745884	
P(T<=t) tw	0.652035	
t Critical tw	2.119905	

Figure D.11: Number of iterations

Although the number of iterations used was marginally higher using forward kinematics, this difference was not significant.

2 Second evaluation

2.1 Training

	Generator	Forward kinematics	Inverse kinematics
1	65	56	46
2	52	61	36
3	42	71	111
4	29	46	64
5	38	40	76
6	65	76	111
7	46	43	43
8	74	102	36
9	65	59	40
10	43	71	46
11	56	70	34
12	97	48	41
13	50	55	42
14	54	59	34
15	157	52	113
16	33	30	41
17	30	42	45
18	42	52	94
19	72	50	51
20	97	95	108
21	76	36	49
22	88	64	53
23	76	106	57
24	89	96	71
25	59	53	89
26	45	49	69
27	52	55	56
28	35	56	107
29	42	32	42
30	48	33	44
31	47	115	112
32	53	80	113
33	43	52	77
34	39	74	51
35	28	53	52
36	61	72	102
37	47	65	78
38	75	91	106
39	42	41	32
40	35	54	42

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Generator	40	2187	54.675	340.6865
Forward	40	2455	61.375	436.5481
Inverse	40	2614	65.35	790.2846

ANOVA

Source of Varia	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between G	2328.617	2	1164.308	2.228314	0.112262	3.073765
Within Gro	61133.25	117	522.5064			
Total	63461.87	119				

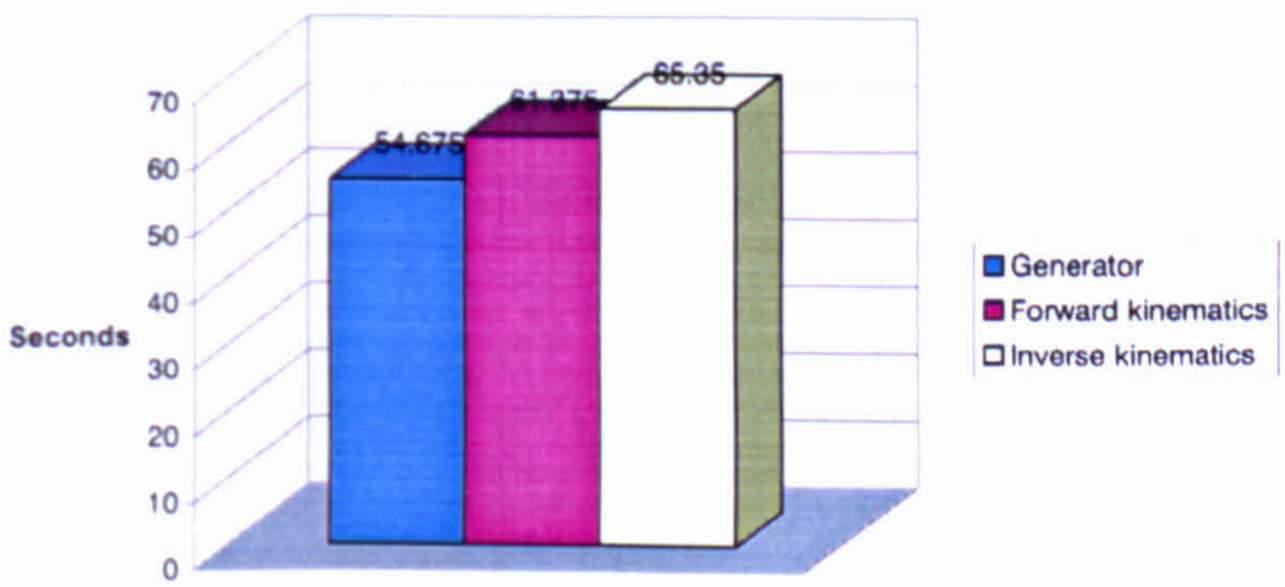


Figure D.12: Training times

These are the logs from my training session. Although *the generator performs better, it* is not by much. Considerable improvement were achieved while performing the second evaluation.

2.2 Timings

	Generator	Forward	Inverse
0	26	40	30
1	23	58	33
2	99	78	88
3	48	48	34
4	16	51	27
5	29	56	31
6	86	86	45
7	31	46	39
8	50	49	34
9	61	54	35
10	39	66	54
11	43	49	58
12	24	72	51
13	48	118	125
14	21	50	54
15	55	55	80
16	35	58	55
17	22	33	36
18	39	23	35
19	13	53	41
20	19	52	35
21	42	53	88
22	31	71	28
23	42	42	37
24	23	46	49
25	31	59	30
26	44	44	35
27	37	60	43
28	30	59	35
29	43	66	67
30	21	41	30
31	33	46	43
32	32	55	67
33	41	58	40
34	23	59	61
35	26	44	43
36	50	67	53
37	26	62	50
38	22	37	55
39	40	39	27
40	4	34	21

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Generator		1468	35.80488	316.211
Forward		2237	54.56098	257.9024
Inverse		1922	46.87805	414.4598

ANOVA

Source of Varia	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between G	7290.26	2	3645.13	11.06179	3.9E-05	3.071776
Within Gro	39542.93	120	329.5244			
Total	46833.19	122				

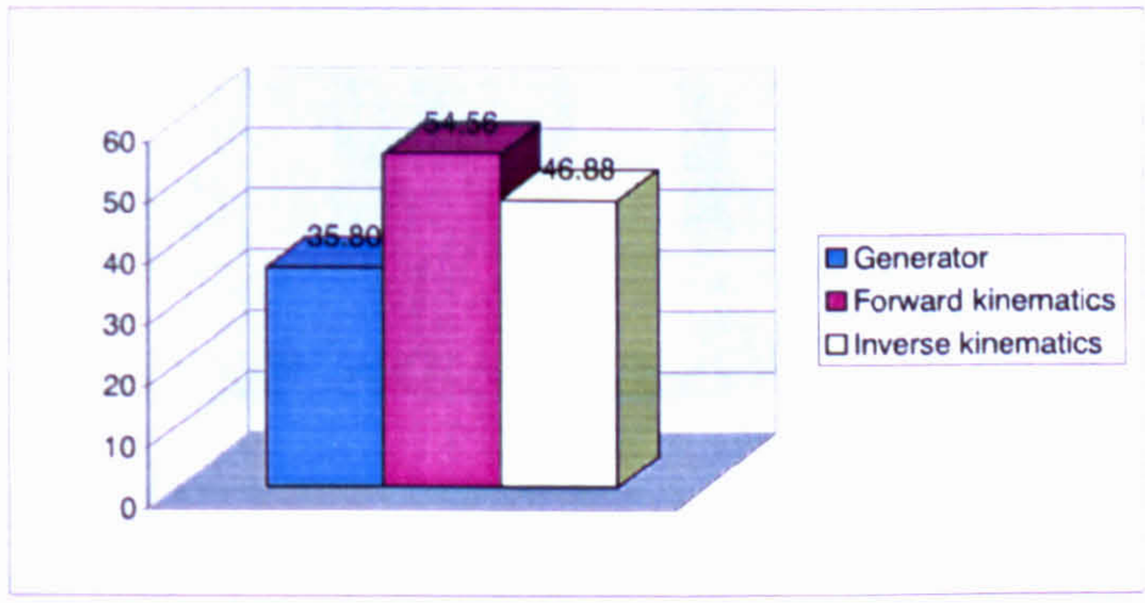


Figure D.13: Time to pose the articulated figure

Poses were produced considerably faster using the generator. Differences in the means were extremely significants.

2.3 Number of iterations

	Generator	Forward	Inverse
0	8	8	6
1	8	10	8
2	12	13	13
3	11	8	9
4	6	9	5
5	7	9	5
6	14	12	8
7	8	12	8
8	10	12	4
9	13	11	7
10	8	15	10
11	11	9	9
12	6	10	8
13	9	17	20
14	12	8	11
15	12	11	13
16	9	11	10
17	6	5	8
18	6	4	6
19	7	8	10
20	6	7	8
21	11	12	14
22	13	15	5
23	10	9	7
24	8	10	10
25	10	12	8
26	10	8	8
27	10	10	8
28	7	10	8
29	13	9	14
30	6	6	7
31	8	8	8
32	10	13	10
33	10	12	10
34	5	10	11
35	9	7	10
36	12	13	11
37	9	10	13
38	7	6	10
39	8	9	8
40	3	5	5

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Generator	41	368	8.97561	6.42439
Forward	41	403	9.829268	7.945122
Inverse	41	371	9.04878	9.247561

ANOVA

Source of Vari	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between G	18.35772	2	9.178862	1.165961	0.315128	3.071776
Within Gro	944.6829	120	7.872358			
Total	963.0407	122				

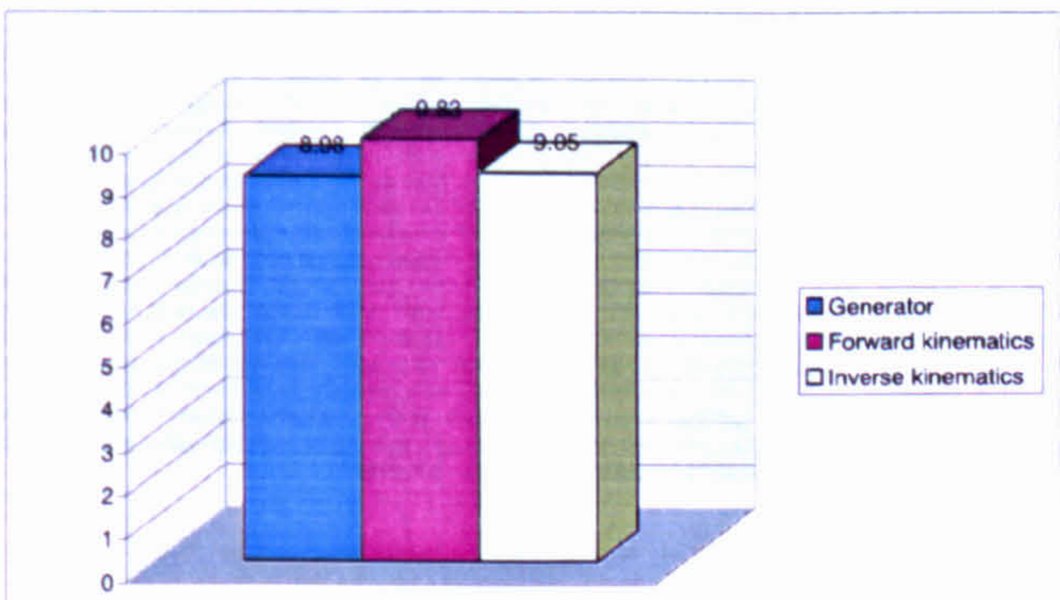


Figure D.14: Number of iterations

There was no significant difference in the number of iterations necessary to produce a pose.

2.4 Correlation between time, iterations, etc

Pose No	Time spent	Iterations	Generator	Page Switc	T	I	G	P	T*T	I*I	G*G	P*P	T*I	T*G	T*P
0	26	8	1	2	-9.804878	-0.97561	-0.853659	-0.463415	96.13563	0.951814	0.728733	0.214753	9.565735	8.370018	4.543724
1	23	8	2	2	-12.80488	-0.97561	0.146341	-0.463415	163.9649	0.951814	0.021416	0.214753	12.49256	-1.873885	5.933968
2	99	12	3	4	63.19512	3.02439	1.146341	1.536585	3993.623	9.146936	1.314099	2.361095	191.1267	72.44319	97.1047
3	48	11	2	2	12.19512	2.02439	0.146341	-0.463415	148.721	4.098156	0.021416	0.214753	24.68769	1.784652	-5.651398
4	16	6	1	1	-19.80488	-2.97561	-0.853659	-1.463415	392.2332	8.854253	0.728733	2.141582	58.93159	16.9066	28.98275
5	29	7	1	1	-6.804878	-1.97561	-0.853659	-1.463415	46.30637	3.903034	0.728733	2.141582	13.44378	5.809042	9.958358
6	86	14	4	5	50.19512	5.02439	2.146341	2.536585	2519.55	25.2445	4.606782	6.434265	252.1999	107.7359	127.3242
7	31	8	2	3	-4.804878	-0.97561	0.146341	0.536585	23.08685	0.951814	0.021416	0.287924	4.687686	-0.703153	-2.578227
8	50	10	3	3	14.19512	1.02439	1.146341	0.536585	201.5015	1.049375	1.314099	0.287924	14.54134	16.27246	7.616895
9	61	13	5	6	25.19512	4.02439	3.146341	3.536585	634.7942	16.19572	9.899465	12.50744	101.395	79.27246	89.1047
10	39	8	2	3	3.195122	-0.97561	0.146341	0.536585	10.2088	0.951814	0.021416	0.287924	-3.117192	0.467579	1.714456
11	43	11	2	2	7.195122	2.02439	0.146341	-0.463415	51.76978	4.098156	0.021416	0.214753	14.56573	1.052945	-3.334325
12	24	6	1	2	-11.80488	-2.97561	-0.853659	-0.463415	139.3551	8.854253	0.728733	0.214753	35.12671	10.07733	5.470553
13	48	9	2	3	12.19512	0.02439	0.146341	0.536585	148.721	0.000595	0.021416	0.287924	0.297442	1.784652	6.543724
14	21	12	1	1	-14.80488	3.02439	-0.853659	-1.463415	219.1844	9.146936	0.728733	2.141582	-44.77573	12.63831	21.66568
15	55	12	2	3	19.19512	3.02439	0.146341	0.536585	368.4527	9.146936	0.021416	0.287924	58.05354	2.809042	10.29982
16	35	9	2	3	-0.804878	0.02439	0.146341	0.536585	0.647829	0.000595	0.021416	0.287924	-0.019631	-0.117787	-0.431886
17	22	6	1	2	-13.80488	-2.97561	-0.853659	-0.463415	190.5747	8.854253	0.728733	0.214753	41.07793	11.78465	6.397383
18	39	6	2	3	3.195122	-2.97561	0.146341	0.536585	10.2088	8.854253	0.021416	0.287924	-9.507436	0.467579	1.714456
19	13	7	1	1	-22.80488	-1.97561	-0.853659	-1.463415	520.0625	3.903034	0.728733	2.141582	45.05354	19.46758	33.37299
20	19	6	1	1	-16.80488	-2.97561	-0.853659	-1.463415	282.4039	8.854253	0.728733	2.141582	50.00476	14.34563	24.5925
21	42	11	2	3	6.195122	2.02439	0.146341	0.536585	38.37954	4.098156	0.021416	0.287924	12.54134	0.906603	3.324212
22	31	13	2	3	-4.804878	4.02439	0.146341	0.536585	23.08685	16.19572	0.021416	0.287924	-19.3367	-0.703153	-2.578227
23	42	10	2	2	6.195122	1.02439	0.146341	-0.463415	38.37954	1.049375	0.021416	0.214753	6.346222	0.906603	-2.87091
24	23	8	1	2	-12.80488	-0.97561	-0.853659	-0.463415	163.9649	0.951814	0.728733	0.214753	12.49256	10.93099	5.933968
25	31	10	2	3	-4.804878	1.02439	0.146341	0.536585	23.08685	1.049375	0.021416	0.287924	-4.92207	-0.703153	-2.578227
26	44	10	2	4	8.195122	1.02439	0.146341	1.536585	67.16002	1.049375	0.021416	2.361095	8.395003	1.199286	12.5925
27	37	10	2	2	1.195122	1.02439	0.146341	-0.463415	1.428316	1.049375	0.021416	0.214753	1.224271	0.174896	-0.553837
28	30	7	2	2	-5.804878	-1.97561	0.146341	-0.463415	33.69661	3.903034	0.021416	0.214753	11.46817	-0.849494	2.690065
29	43	13	2	2	7.195122	4.02439	0.146341	-0.463415	51.76978	16.19572	0.021416	0.214753	28.95598	1.052945	-3.334325
30	21	6	1	2	-14.80488	-2.97561	-0.853659	-0.463415	219.1844	8.854253	0.728733	0.214753	44.05354	12.63831	6.860797
31	33	8	1	2	-2.804878	-0.97561	-0.853659	-0.463415	7.867341	0.951814	0.728733	0.214753	2.736466	2.394408	1.299822
32	32	10	2	2	-3.804878	1.02439	0.146341	-0.463415	14.4771	1.049375	0.021416	0.214753	-3.89768	-0.556811	1.763236
33	41	10	2	3	5.195122	1.02439	0.146341	0.536585	26.98929	1.049375	0.021416	0.287924	5.321832	0.760262	2.787626
34	23	5	1	1	-12.80488	-3.97561	-0.853659	-1.463415	163.9649	15.80547	0.728733	2.141582	50.9072	10.93099	18.73885
35	26	9	1	2	-9.804878	0.02439	-0.853659	-0.463415	96.13563	0.000595	0.728733	0.214753	-0.239143	8.370018	4.543724
36	50	12	3	5	14.19512	3.02439	1.146341	2.536585	201.5015	9.146936	1.314099	6.434265	42.93159	16.27246	36.00714
37	26	9	1	2	-9.804878	0.02439	-0.853659	-0.463415	96.13563	0.000595	0.728733	0.214753	-0.239143	8.370018	4.543724
38	22	7	2	2	-13.80488	-1.97561	0.146341	-0.463415	190.5747	3.903034	0.021416	0.214753	27.27305	-2.020226	6.397383
39	40	8	3	3	4.195122	-0.97561	1.146341	0.536585	17.59905	0.951814	1.314099	0.287924	-4.092802	4.809042	2.251041
40	4	3	1	1	-31.80488	-5.97561	-0.853659	-1.463415	1011.55	35.70791	0.728733	2.141582	190.0535	27.15051	46.54372
	35.80488	8.97561	1.853659	2.463415					308.4985	6.267698	0.759072	1.273052	31.26353	11.77632	14.99286
													0.71098	0.769559	0.756545

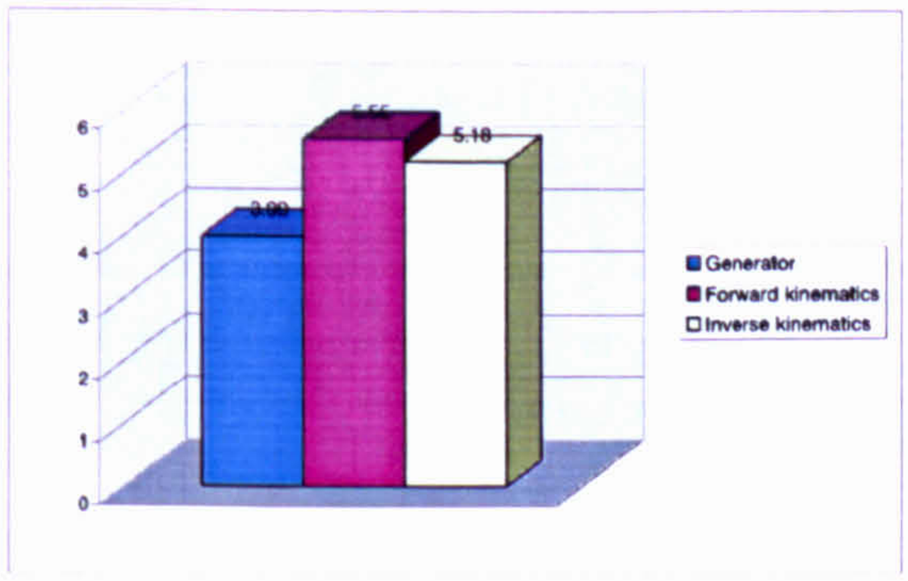


Figure D.15: Correlation between times, iterations, etc

To find if there was a correlation between time, number of iterations, number of generations and number of page switch, the Pearson coefficient was computed manually. Excel^{TM} was of great help here.
Below is shown the time per iteration for each technique.

2.4 Correlation between time, iterations, etc

Pose No	Time spent	Iterations	Generator	Page Switc	T	I	G	P	T*T	I*I	G*G	P*P	T*I	T*G	T*P
0	26	8	1	2	-9.804878	-0.97561	-0.853659	-0.463415	96.13563	0.951814	0.728733	0.214753	9.565735	8.370018	4.543724
1	23	8	2	2	-12.80488	-0.97561	0.146341	-0.463415	163.9649	0.951814	0.021416	0.214753	12.49256	-1.873885	5.933968
2	99	12	3	4	63.19512	3.02439	1.146341	1.536585	3993.623	9.146936	1.314099	2.361095	191.1267	72.44319	97.1047
3	48	11	2	2	12.19512	2.02439	0.146341	-0.463415	148.721	4.098156	0.021416	0.214753	24.68769	1.784652	-5.651398
4	16	6	1	1	-19.80488	-2.97561	-0.853659	-1.463415	392.2332	8.854253	0.728733	2.141582	58.93159	16.9066	28.98275
5	29	7	1	1	-6.804878	-1.97561	-0.853659	-1.463415	46.30637	3.903034	0.728733	2.141582	13.44378	5.809042	9.958358
6	86	14	4	5	50.19512	5.02439	2.146341	2.536585	2519.55	25.2445	4.606782	6.434265	252.1999	107.7359	127.3242
7	31	8	2	3	-4.804878	-0.97561	0.146341	0.536585	23.08685	0.951814	0.021416	0.287924	4.687686	-0.703153	-2.578227
8	50	10	3	3	14.19512	1.02439	1.146341	0.536585	201.5015	1.049375	1.314099	0.287924	14.54134	16.27246	7.616895
9	61	13	5	6	25.19512	4.02439	3.146341	3.536585	634.7942	16.19572	9.899465	12.50744	101.395	79.27246	89.1047
10	39	8	2	3	3.195122	-0.97561	0.146341	0.536585	10.2088	0.951814	0.021416	0.287924	-3.117192	0.467579	1.714456
11	43	11	2	2	7.195122	2.02439	0.146341	-0.463415	51.76978	4.098156	0.021416	0.214753	14.56573	1.052945	-3.334325
12	24	6	1	2	-11.80488	-2.97561	-0.853659	-0.463415	139.3551	8.854253	0.728733	0.214753	35.12671	10.07733	5.470553
13	48	9	2	3	12.19512	0.02439	0.146341	0.536585	148.721	0.000595	0.021416	0.287924	0.297442	1.784652	6.543724
14	21	12	1	1	-14.80488	3.02439	-0.853659	-1.463415	219.1844	9.146936	0.728733	2.141582	-44.77573	12.63831	21.66568
15	55	12	2	3	19.19512	3.02439	0.146341	0.536585	368.4527	9.146936	0.021416	0.287924	58.05354	2.809042	10.29982
16	35	9	2	3	-0.804878	0.02439	0.146341	0.536585	0.647829	0.000595	0.021416	0.287924	-0.019631	-0.117787	-0.431886
17	22	6	1	2	-13.80488	-2.97561	-0.853659	-0.463415	190.5747	8.854253	0.728733	0.214753	41.07793	11.78465	6.397383
18	39	6	2	3	3.195122	-2.97561	0.146341	0.536585	10.2088	8.854253	0.021416	0.287924	-9.507436	0.467579	1.714456
19	13	7	1	1	-22.80488	-1.97561	-0.853659	-1.463415	520.0625	3.903034	0.728733	2.141582	45.05354	19.46758	33.37299
20	19	6	1	1	-16.80488	-2.97561	-0.853659	-1.463415	282.4039	8.854253	0.728733	2.141582	50.00476	14.34563	24.5925
21	42	11	2	3	6.195122	2.02439	0.146341	0.536585	38.37954	4.098156	0.021416	0.287924	12.54134	0.906603	3.324212
22	31	13	2	3	-4.804878	4.02439	0.146341	0.536585	23.08685	16.19572	0.021416	0.287924	-19.3367	-0.703153	-2.578227
23	42	10	2	2	6.195122	1.02439	0.146341	-0.463415	38.37954	1.049375	0.021416	0.214753	6.346222	0.906603	-2.87091
24	23	8	1	2	-12.80488	-0.97561	-0.853659	-0.463415	163.9649	0.951814	0.728733	0.214753	12.49256	10.93099	5.933968
25	31	10	2	3	-4.804878	1.02439	0.146341	0.536585	23.08685	1.049375	0.021416	0.287924	-4.92207	-0.703153	-2.578227
26	44	10	2	4	8.195122	1.02439	0.146341	1.536585	67.16002	1.049375	0.021416	2.361095	8.395003	1.199286	12.5925
27	37	10	2	2	1.195122	1.02439	0.146341	-0.463415	1.428316	1.049375	0.021416	0.214753	1.224271	0.174896	-0.553837
28	30	7	2	2	-5.804878	-1.97561	0.146341	-0.463415	33.69661	3.903034	0.021416	0.214753	11.46817	-0.849494	2.690065
29	43	13	2	2	7.195122	4.02439	0.146341	-0.463415	51.76978	16.19572	0.021416	0.214753	28.95598	1.052945	-3.334325
30	21	6	1	2	-14.80488	-2.97561	-0.853659	-0.463415	219.1844	8.854253	0.728733	0.214753	44.05354	12.63831	6.860797
31	33	8	1	2	-2.804878	-0.97561	-0.853659	-0.463415	7.867341	0.951814	0.728733	0.214753	2.736466	2.394408	1.299822
32	32	10	2	2	-3.804878	1.02439	0.146341	-0.463415	14.4771	1.049375	0.021416	0.214753	-3.89768	-0.556811	1.763236
33	41	10	2	3	5.195122	1.02439	0.146341	0.536585	26.98929	1.049375	0.021416	0.287924	5.321832	0.760262	2.787626
34	23	5	1	1	-12.80488	-3.97561	-0.853659	-1.463415	163.9649	15.80547	0.728733	2.141582	50.9072	10.93099	18.73885
35	26	9	1	2	-9.804878	0.02439	-0.853659	-0.463415	96.13563	0.000595	0.728733	0.214753	-0.239143	8.370018	4.543724
36	50	12	3	5	14.19512	3.02439	1.146341	2.536585	201.5015	9.146936	1.314099	6.434265	42.93159	16.27246	36.00714
37	26	9	1	2	-9.804878	0.02439	-0.853659	-0.463415	96.13563	0.000595	0.728733	0.214753	-0.239143	8.370018	4.543724
38	22	7	2	2	-13.80488	-1.97561	0.146341	-0.463415	190.5747	3.903034	0.021416	0.214753	27.27305	-2.020226	6.397383
39	40	8	3	3	4.195122	-0.97561	1.146341	0.536585	17.59905	0.951814	1.314099	0.287924	-4.092802	4.809042	2.251041
40	4	3	1	1	-31.80488	-5.97561	-0.853659	-1.463415	1011.55	35.70791	0.728733	2.141582	190.0535	27.15051	46.54372
	35.80488	8.97561	1.853659	2.463415					308.4985	6.267698	0.759072	1.273052	31.26353	11.77632	14.99286
													0.71098	0.769559	0.756545

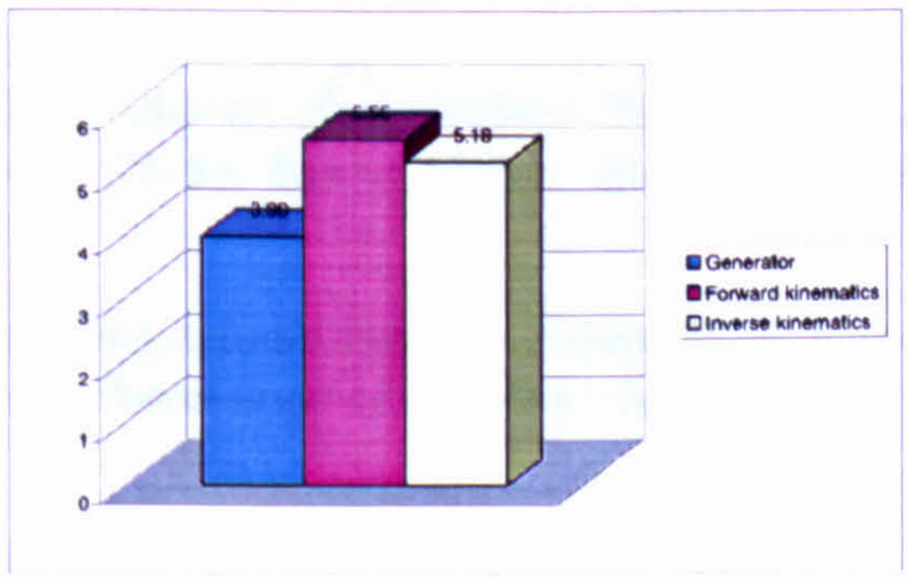


Figure D.15: Correlation between times, iterations, etc

To find if there was a correlation between time, number of iterations, number of generations and number of page switch, the Pearson coefficient was computed manually. Excel^{TM} was of great help here.
Below is shown the time per iteration for each technique.

Bibliography

[ADH89] Bruno Arnaldi, Georges Dumont, and Gerard Hegron. Dynamics and unification of animation control. *The Visual Computer*, 5(1/2):22–31, March 1989.

[AG85] William W. Armstrong and Mark W. Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1(4):231–240, December 1985.

[AGL87] William W. Armstrong, Mark Green, and Robert Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6):52–61, June 1987.

[AS96] Maria-Elena Algorri and Francis Schmitt. Mesh simplification. In J. Rossignac and F. Sillion, editors, *Eurographics '96 Conference Proceedings*, volume 15, pages 77–86. Eurographics, Blacwell Publishers, 1996.

[Bad86] Norman I. Badler. Animating human figures: Perspectives and directions. In M. Green, editor, *Proceedings of Graphics Interface '86*, pages 115–120, May 1986.

[Bar87] David Baraff. Non-penetrating rigit body simulation. 1987.

[BC89] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. In Jeffrey Lane, editor, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 233–242, July 1989.

[Bec92] Ami B. Becker. Effects of aircraft noise on vigilance performance and perceived workload. In *Proceedings of the Human Factors Society 36th Annual Meeting*, volume 2, pages 1513–1517, 1992.

[BKK+85] Norman I. Badler, Jonathan D. Korein, James U. Korein, Gerald M. Radack, and Lynne Shapiro Brotman. Positioning and animating human figures in a task-oriented environment. *The Visual Computer*, 1(4):212–220, December 1985.

[Bli89] James F. Blinn. Optimal tubes. *IEEE Compter Graphics & Applications*, pages 8–13, 1989.

[BMB86] Norman I. Badler, Kamran H. Manoochehri, and David Baraff. Multi-dimensional input techniques and articulated figure positioning by multiple constraints. In Frank Crow and Stephen M. Pizer, editors, *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pages 151–169, October 1986.

[BMW87] Norman I. Badler, Kamran H. Manoochehri, and Graham Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, June 1987.

[BN88] Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 309–315, August 1988.

[BN93] Granieri J. P. Badler N., Hollick M. J. Real-time control of a virtual human using minimal sensors. *Presence*, 2(1):82–86, 1993. MIT.

[BOK80] N. I. Badler, J. O'Rourke, and G. Kaufman. Special problems in human movement simulation. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 189–197, July 1980.

[BPW93] Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, New York, 1993. ISBN 0-19-507359-2.

[Bre94] Stephen Anthony Brewster. *Providing a Structured Method for Integrating Non-Speech Audio into Human-Computer Interfaces.* PhD thesis, University of York, 1994.

[BS79] N. I. Badler and S. W. Smoliar. Digital representation of human movement. *ACM Computing Surveys*, 11:19–38, March 1979.

[BT92] Ronan Boulic and Daniel Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. In *Computer Graphics forum*, volume 4, pages 189–202, december 1992.

[BTT90] Ronan Boulic, Nadia Magnenat Thalmann, and Daniel Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6(6):344–358, December 1990.

[BW86] Gary Bishop and David M. Weimer. Fast Phong shading. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 103–106, August 1986.

[Cal88] Tom Calvert. The challenge of human figure animation. In *Proceedings of Graphics Interface '88*, pages 203–210, June 1988.

[Cas94] Mike Castle. Sphere tesselation starting from a octahedron. *Alt.Comp.Graphics*, 1994.

[CCP82] T. W. Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. *IEEE Comput. Graphics and Appl. (USA)*, 2:41–48, November 1982.

[Cla] James H. Clark. A fast algorithm for rendering parametric surfaces. pages 27–32.

[Csu75] Charles Csuri. Computer animation. In Anthony P. Lucino, editor, *Computer Graphics (SIGGRAPH '75 Proceedings)*, volume 9, pages 92–101, June 1975.

[CVM+96] Jonathan Cohen, Amitabh Varshney, Dinesh Manocha, Greg Turk, Hans Weber, Pankaj Agarwal, Frederick P. Brooks, Jr., and William Wright. Simplification envelopes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 119–128. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[Dai88] F. Dai. Collision-free motion of an articulated kinematic chain in a dynamic environment. *IEEE Computer Graphics and Applications*, 9(1):70–74, January 1988.

[Dar59] C. Darwin. *On the Origin of Species.* John Murray, London, 1859. *The first edition. No bibliography on life would be complete without this entry!*

[DAS93] Mark W. Scerbo David A. Sawin. Vigilance: Where has all the workload gone? In *Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting*, volume 2 of *VISUAL PERFORMANCE: Attention and Workload*, pages 1383–1387, 1993.

[Dav91a] Yuval Davidor. *Genetic Algorithms and Robotics. A heuristic strategy for optimization.* World Scientific, 1991.

[Dav91b] Lawrence Davis. *Handbook of genetic algorithms.* Van Nostrand Reinhold, 1991.

[Daw86] Richard Dawkins. *The Blind Watchmaker.* Harlow Longman Scientific & Technical, 1986.

[DBM93] David R. Bull David Beasley and Ralph R. Martin. An overview of genetic algorithms. *University Computing*, 15(2 & 4):58–69 & 170–181, 1993.

[Dem93] William N. Dember. The rate of gain of perceived workload in sustained attention. In *Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting*, volume 2, pages 1388–1392, 1993.

[dJAGAN76] J. Garcia de Jalon A. Garcia-Alonso N.Serrano. Interactive simulation of complex mechanical systems. pages 1–28, 1976.

[Dud91] Helen J. Dudfield. Colour head-up displays: Help or hindrance? In *Proceedings of the Human Factors Society 35th Annual Meeting*, volume 1, pages 146–150, 1991.

[E.G89] David E.Goldberg. *GENETIC ALGORITHMS in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.

[Egg88] F. Thomas Eggemeier. *Human Mental Workload*, chapter Properties of Workload Assesment Techniques. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[FGH+85] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 111–120, July 1985.

[FW88] David R. Forsey and Jane Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. In *Proceedings of Graphics Interface '88*, pages 8–15, June 1988.

[Gir86] Michael Girard. Interactive design of 3D computer-animated legged animal motion. In Frank Crow and Stephen M. Pizer, editors, *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pages 131–150, October 1986.

[Gir87] Michael Girard. Interactive design of 3D computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, June 1987.

[Gir91] Michael Girard. *Constrained optimization of articulated animal movement in computer animation*, pages 209–232. Morgan Kaufmann, 1991.

[Gla90] Andrew S. Glassner. *Graphics Gems.* Academic Press, Inc, 1990.

[GM85] Michael Girard and Anthony A. Maciejewski. Computational modeling for the computer animation of legged figures. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 263–270, July 1985.

[Gre91] Mark Green. *Using dynamics in computer animation: control and solution issues*, pages 281–314. Morgan Kaufmann, 1991.

[GT95] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of Muscle-Actuated locomotion through control abstraction. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 63–70. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 299–308, August 1988.

[HB93] Joerg Heitkoetter and David Beasley. The hitch-hiker's guide to evolutionary computation: A list of frequently asked questions (faq). Freely available on ftps sites, rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/, 1993. About 90 pages.

[HC97] Dennis Howitt and Duncan Crammer. *An Introduction to Statistics for Psychology. A complete Guide for Students.* Prentice Hall/Harvester Wheatsheaf, 1997.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 19–26, August 1993.

[HE80] D. Herbison-Evans. Rapid raster ellipsoid shading. *Computer Graphics*, 13(4):355–361, February 1980.

[HE82] D. Herbison-Evans. Real-time animation of human figure drawings with hidden lines omitted. *IEEE Computer Graphics and Applications*, 2(9):27–33, November 1982.

[HM95] Nakagaki Y. Hirose M., Deffaux G. A study on data input of natural human motion for virtual reality system. In *ICAT/VRST'95*, pages 245–251. ACM- SIGCHI, November 1995.

[HM96] Nakagaki Y. Hirose M., Deffaux G. Development of an effective motion capture system based on data fusion and minimal use of sensors. In *VRST'96*, pages 117–123. ACM-SIGGRAPH and ACM-SIGCHI, July 1996.

[Hog92] Stuart G. Hoggar. *Mathematics for Computer Graphics.* Cambridge University Press, 1992.

[Hol75] John H. Holland. *Adaptation in natural and artificial systems.* Ann Arbor: The University of Michigan Press, 1975.

[HS85] Pat Hanrahan and David Sturman. Interactive animation of parametric models. *The Visual Computer*, 1(4):260–266, December 1985.

[HWA+91] David Haumann, Jakub Wejchert, Kavi Arya, Bob Bacon, Al Khorasani, Alan Norton, and Paula Sweeney. An application of motion design and control for physically-based animation. In *Proceedings of Graphics Interface '91*, pages 279–286, June 1991.

[IC88] Paul M. Isaacs and Michael F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4(6):296–305, December 1988.

[J.96] Bers J. A body model server for human motion capture and representation. *Presence*, 5(4):381–392, 1996. MIT.

[JB89] A.C. Bittner & S.G. Hill J.C. Byers. *Advances in industrial ergonomics*, chapter Traditional and raw task load index (TLX) correlations: Are paired comparisions necessary ?, pages 481–485. Taylor & Francis, 1989.

[Jex88] Henri R. Jex. *Human Mental Workload*, chapter Measuring Mental Workload: Problems, process and promises. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[JFH90] Steven Feiner James Foley, Andries van Dam and John Hughes. *Computer Graphics. Principle and practice.* Addison Wesley, 1990.

[JPBHC94] Helen Sharp Jenny Preece, Yvonne Rogers, David Benyon, Simon Holland, and Tom Carey. *Human Computer Interaction.* Addison-Wesley, 1994.

[JU85] Korein JU. *A Geometric Investigation of Reach.* MIT Press, Cambridge, MA, 1985.

[Kau87] Arie Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 171–179, July 1987.

[KB82] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed animation of articulated structures. *IEEE Computer Graphics and Applications*, 2(9):71–81, November 1982.

[KB84] D. H. U. Kochanek and R. H. Bartels. Interpolating splines for keyframe animation. In S. MacKay, editor, *Graphics Interface '84 Proceedings*, pages 41–42, 1984.

[Kno81] K. Knowlton. Computer-aided definition, manipulation and depiction of objects composed of spheres. *Comput. Graphics*, 15:352–375, December 1981.

[Kor82] James Korein. Using reach descriptions to position kinematic chains. In *Proceedings of the Fourth National Conf. Canadian Society for Computational Studies of Intelligence*, pages 79–84, May 1982.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[Koz94] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[KT96] Alan D. Kalvin and Russell H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Applications*, 16(3):64–77, May 1996. ISSN 0272-1716.

[Lam95] Patrick Lambourne. 3d facial deformation. Master's thesis, University of Glasgow, 1995.

[Las87] John Lasseter. Principles of traditional animation applied to 3D computer animation. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 35–44, July 1987.

[LCWB80] J. Lane, L. Carpenter, T. Whitted, and J. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of the ACM*, 23(1):23–34, 1980.

[LK95] Jintae Lee and Tosiyasu L. Kunii. Model-Based analysis of hand posture. *IEEE Computer Graphics and Applications*, 15(5):77–86, September 1995.

[LKR+96] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hughes, Nick Faust, and Gregory Turner. Real-Time, continuous level of detail rendering of height fields. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 109–118. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[LR80] J. Lane and R. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Trans. Pattern Analysis Machine Intell.*, 2(1):35–46, 1980.

[Mac] Macromedia. *LifeForm - User manual.* Macro media.

[MB91] Gary Monheit and Norman I. Badler. A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications*, 11(2):29–38, March 1991.

[MC90] Claudia L. Morawetz and Thomas W. Calvert. Goal-directed human animation of multiple movements. In *Proceedings of Graphics Interface '90*, pages 60–67, May 1990.

[Mes88a] N. Meshkati. *Human Mental Workload*, chapter An Eclectic and Critical Review of four Primary Mental Workload Assessment Methods: A guide for developing a comprehensive model. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[Mes88b] N. Meshkati. *Human Mental Workload*, chapter Towards Development of a Cohesive Model of Workload. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[MK91] Don Malzahn Mary Klein. Effects of physical ability on working memory requirements, keystroke rate, and subjective workload of computer input devices. In *Proceedings of the Human Factors Society 35th Annual Meeting*, volume 1, pages 261–265, 1991.

[MP89] Claudio Mirolo and Enrico Pagello. A solid modeling system for robot action planning. *IEEE Computer Graphics and Applications*, 9(1):55–69, January 1989.

[MP94] David P. Miller and Richard E. Parent. An articulated limb motion planner for optimized movement. *The Journal of Visualisation and Computer Animation*, 5(2):85–123, April-June 1994.

[MTD96] Boulic R. Molet T. and Thalmann D. A real time anatomical converter for human motion capture. In *in Boulic R. and Hegron G. (eds) Eurographics workshop on Computer Animation and Simulation'96*, pages 79–94. Springer- Verlag Wien, July 1996.

[MTLT88] Nadia Magnenat-Thalmann, Richard Laperriere, and Daniel Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings of Graphics Interface '88*, pages 26–33, June 1988.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 289–298, August 1988.

[MZ90] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 29–38, August 1990.

[NMT85] Daniel Thalmann Nadia Magnenat-Thalmann. *Human Modeling and Animation*, pages 121–134. Springer-Verlag, 1985.

[NTD88] N.Magnenat-Thalmann and D.Thalmann. Construction and animation of a synthetic actress. In D. A. Duce and P. Jancene, editors, *Eurographics '88*, pages 55–66. North-Holland, September 1988.

[Ove94] C. W. A. M. Van Overveld. A simple approximation to rigid body dynamics for computer animation. *The journal of visualisation & computer animation*, 5:17–36, 1994.

[Pat93] John W. Patterson. Fast spheres. In R. J. Hubbold and R. Juan, editors, *EUROGRAPHICS '93*, volume 12, pages 61–72, September 1993.

[Pet97] Philip Ray Peterson. A genetic engineering approach to texture synthesis. Master's thesis, SFU Computing Science School, April 1997.

[Ple89] Daniel Pletinckx. Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, 5(1/2):2–13, March 1989.

[PW94] J.W. Patterson and P.J. Willis. Computer assisted animation: 2d or not 2d? *The Computer Journal*, 37(10):829–839, 1994.

[RAK93] Gregory M. Corso Robert A. King. Auditory displays: If they are so useful, why are they turned off? In *Proceedings of the Human Factors and Ergonomics Society 37th Annual Meeting*, volume 1, pages 549–553, 1993.

[ree93] *Using genetic algorithms with small populations*, volume 5, July 1993.

[Rei88] Gary B. Reid. *Human Mental Workload*, chapter The Subjective Workload Assessment Technique: A scaling procedure for measuring mental workload. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[RG91] Hans Rijpkema and Michael Girard. Computer animation of knowledge-based human grasping. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 339–348, July 1991.

[RH91] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 349–358, July 1991.

[SB85] Scott N. Steketee and Norman I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phasing control. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 255–262, July 1985.

[SC92] Thecla Schiphorst and Thomas Calvert. LIFE FORMS — making discoveries in dance — an interactive tool for composition used by merce cunningham. In *Proceedings of the 1992 Western Computer Graphics Symposium*, pages 87–98, April 1992.

[SH90] C. Wickens S.G. Hart. *Workload assessment and prediction*, chapter MANPRINT, an approach to systems integration, pages 257–296. New York: Van Nostrand Reinhold, 1990.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 245–254, July 1985.

[Sho87] Ken Shoemake. Quaternion calculus and fast animation. *SIGGRAPH 1987 Tutorial*, pages 101–121, 1987.

[Sim91] Karl Sims. Artificial evolution for computer graphics. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 319–328, jul 1991.

[Sim94a] Karl Sims. Evolving 3d morphology and behavior by competition. In R.Brooks & P.Maes, editor, *Artificial Life IV Proceedings*, pages 28–39. MIT Press, 1994.

[Sim94b] Karl Sims. Evolving virtual creatures. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22. ACM SIGGRAPH, ACM Press, jul 1994. ISBN 0-89791-667-0.

[Smi93] Robert E. Smith. Adaptively resizing populations: An algorithm and analysis. Tcga, The Clearinghouse for Genetic Algorithms, February 1993.

[SSK96] Standfield S Semwal S. K., Hightower R. Closed form and geometric algorithms for real-time control of an avatar. In *Proceedings of VRAIS'96*, pages 177–184. IEEE, 1996.

[ST90] William Latham Stephen Todd. Mutator, a subjective human interface for evolution of computer sculptures. *IBM United kingdom Scientific Centre Report*, 1(248), December 1990.

[ST94] R. M. Sanso and D. Thalmann. A hand control and automatic grasping system for synthetic actors. In *Computer Graphics Forum*, volume 13, pages 167–177. Eurographics, Basil Blackwell Ltd, 1994. Eurographics '94 Conference issue.

[Sta88] S. G. Hart & L. E. Staveland. *Human Mental Workload*, chapter The developement of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[STH91] William Latham Stephen Todd and Peter Hughes. Computer sculpture design and animation. March 1991.

[Stu84] D. Sturman. Interactive keyframe animation of 3-D articulated models. In S. MacKay, editor, *Graphics Interface '84 Proceedings*, pages 35–40, 1984.

[Stu86] David Sturman. A discussion on the development of motion control systems. May 1986.

[TH95] C. Thorborrow and A. Hodden. Genetic programming for easy 3d texture generation. In *Eurographics UK Chapter, 13th Annual Conference*, pages 107–113, March 1995.

[Tha88] Daniel Thalmann. Human modelling and animation. pages 1–25, 1988.

[TJ81] F Thomas and O Johnson. *Disney Animation - The Illusion of Life.* Abbeville Press (New York 1981), 1981.

[TL91] Stephen Todd and William Latham. Artificial life or surreal art. *IBM UK Scientific Centre Report*, June 1991.

[TP88] D. Tost and X. Pueyo. Human body animation: a survey. *The Visual Computer*, 3(5):254–264, March 1988.

[uLM96] Linear-Time Simulation using Lagrange Multipliers. David baraff. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 137–146. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[Ven95] J. Ventrella. Disney meets darwin - the evolution of funny animated figures. In *Computer Animation '95*, April 1995.

[vO90] C. W. A. M. van Overveld. A technique for motion specification in computer animation. *The Visual Computer*, 6(2):106–116, March 1990.

[vO91] Cornelius W. A. M. van Overveld. Building blocks for goal direction motion. In *Eurographics Workshop on Animation and Simulation*, pages 41–54, 1991.

[WC78] David A. Wismer and R. Chattergy. *Introduction to Nonlinear Optimization.* North Holland Press, 1978.

[Whi72] D. E. Whitney. The mathematics of coordinated control of prosthetic arms and manipulators. *Trans. ASME, Journal of Dynamic Systems, Measurement, and Control*, pages 303–309, December 1972.

[Wil87a] Jane Wilhelms. Dynamics for everyone. *IEEE Computer Graphics and Applications*, 7(6), June 1987.

[Wil87b] Jane Wilhelms. Toward automatic motion control. *IEEE Computer Graphics and Applications*, 7(4):11–22, April 1987.

[Wil87c] Jane Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[Wil88] Glenn F. Wilson. *Human Mental Workload*, chapter Measurement of Operator Workload with the Neuropsychological Workload Test Battery. Elsevier Science Publishers. Amsterdam (North Holland Press), 1988.

[Wil91] Jane Wilhelms. *Dynamic experiences*, pages 265–279. Morgan Kaufmann, 1991.

[WJ93] Wenping Wang and Barry Joe. Orientation interpolation in quaternion space using spherical biarcs. In *Proceedings of Graphics Interface '93*,

pages 24–32, Toronto, Ontario, Canada, May 1993. Canadian Information Processing Society.

[WMS88] Jane Wilhelms, Matthew Moore, and Robert Skinner. Dynamic animation: interaction and control. *The Visual Computer*, 4(6):283–295, December 1988.

[Zel82] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, November 1982.

[Zel85] David Zeltzer. Towards an integrated view of 3-D computer character animation. *The Visual Computer*, 1:249–259, January 1985.