TASK QUARTERLY 11 No 1–2, 71–86

EMOT – AN EVOLUTIONARY APPROACH TO 3D COMPUTER ANIMATION

HALINA KWAŚNICKA AND PIOTR WOŹNIAK

Institute of Applied Informatics, Wroclaw University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wroclaw, Poland
Halina.Kwasnicka@pwr.wroc.pl

(Received 18 December 2006)

Abstract: Key-framing and Inverse Kinematics are popular animation methods, but new approaches are still developed. We propose a new evolutionary method of creating animation – the EMOT (Evolutionary MOTion) system. It enables automation of motion of animated characters and uses a new evolutionary approach – Gene Expression Programming (GEP). Characters are controlled by computer programs, an animator providing the way of motion's evaluation. GEP works with a randomly selected initial population, using directed but random selection. Experiments have shown that the proposed method is capable of developing robust controllers.

Keywords: Gene Expression Programming, computer animation, simulation, motion

1. Introduction

The paper concerns the problem of automation of computer animation of characters. Animation is understood as specification of motion in such a way that a given entity performs actions and expresses thoughts and emotions, important for the related story. In the paper, we focus on entities (characters) consisting of a set of stiff blocks with a number of joints.

Recently, most animation is created using the Key Framing technique. This technique requires high qualifications of animators: an animator has to take care about the general nature of an animated character, and simultaneously, about all visualization details and the physical credibility. Thus, even partial automation of the process is desirable. We observe intensive study in this area and a number of different approaches being used.

Usually, an evolutionary paradigm is used for developing geometrical and physical models rather than animation itself (*e.g.* the Karl Sims study [1]). Our approach shows that a new evolutionary technique, Gene Expression Programming (GEP) [2], could be useful in automation of movement generation that would meet animators' expectations.

The proposed solution, the EMOT (Evolutionary MOTion) system, allows us to automate the motion of an animated character. Characters are controlled by computer programs (controllers); an animator has to provide the way of motion's evaluation

(fitness function) but GEP searches acceptable controllers. GEP, as all evolutionary methods, works with a randomly selected initial population, using directed but random selection. Naturally, it does not guarantee proper solutions but experiments have shown that the proposed method is capable of developing robust controllers. The method is rather general and it can be applied to various animation tasks.

The paper is structured as follows. The following section is a presentation of problems connected with animation, briefly reviewing useful approaches and techniques. GEP is introduced in Section 3, where the coding schema and genetic operators are presented as well. Section 4 is devoted to the developed computer system, EMOT (Evolution of MOTion). The results of experiments with two animated characters are described in Section 5. A short Summary concludes the paper.

2. 3D animation – a short overview

Animation is not only simulation of physics or of an animal behavior, but also an art [3]. Often real biological and physical features are violated when animators try to tell a story due to the personality of a character and the course of narration. Using an evolutionary approach to the behavior of an animated character allows us to optimize its behavior taking into account the physical features, but what about characters' expressivity? How can we incorporate expressivity in the fitness function?

A number of different objects can move in animation, but characters are only those, which express thoughts, emotion and action important for the related story. When designing motion, animators must care for individuality and emotion of an animated character. Not only living organisms can be used as characters in animation, *e.g.*, a shoal of fish is alive but it is perceived more as a mechanical natural force than as a character.

Special effects such as rain, water waves, a flying plane, *etc.* are yet another category of animation. There are a number of commercially successful techniques – such effects can be obtained by physical simulation. In the present paper, we focus on a specific kind of character animation: animation of a set of stiff blocks with a number of joints. An additional assumption is that our character (blocks with joints) forms an acyclic graph.

2.1. Basic rules of animation

Most of the rules for 'perfect' animation were developed in the early 20th century, especially in the Disney Studios. Frank Thomas and Ollie Johnston, in their book entitled *The Illusion of Life: Disney Animation* proposed twelve basic principles of animation:

Timing – The tempo of moving an object gives meaning to the object and prompts why the object is moving, *e.g.* a quick wink denotes that the object is worried or stimulated, a slow one – that our character is tired and asleep. Additionally, a character's quickness creates a subconscious assumption about its mass and strength in a given scene.

Slow in and out – Movement between extreme positions cannot start and stop violently, *e.g.* a limb achieving a position slows down or gradually accelerates starting to move.

Arcs – In the real world almost every move of a character (or a part thereof) is performed along an arc. Biological joints are usually rotary.

Anticipation – Actions of an animated character usually happen in three phases: a preparatory phase, movement, and a final phase. Anticipation concerns preparations. A jumper usually sags his knees before jumping, which is justified by physics. Looking for an object usually comes before reaching for it.

Exaggeration – Exaggeration of movement, emotions or even constitution of the character's body focuses the attention of spectators and helps in understanding the story.

Squash and stretch (deformations) – This rule describes a way in which animated characters react to movement or being subjected to forces. A ball falling down is flattened when it touches the playing field (squash), but after a rebound the ball is lengthened in direction of movement (stretch). These deformations allow spectators to imagine the physical features of an animated object.

Secondary action – Living organisms have a natural tendency to simultaneously perform a number of actions. In general, a basic activity occurs (required for a basic task, *e.g.* our character walks), but less important, secondary activities accompany it (*e.g.* the walking character looks around). Thus, the character becomes more natural and interesting.

Follow through and overlapping action – The sequence rule (follow through) is similar to anticipation but it concerns the third phase of movement – the finish. It usually consists in moving something and returning to a neutral placement. Overlapping action means that one action comes from a previous one.

Straight-ahead action and pose-to-pose action – These are two alternative ways of creating animation. Forward animation is creating a starting frame and then adding subsequent frames up to the final one (when the action is stopped). The other way is developing frames in a number of key moments and then building the intermediate frames.

Staging – Clear and easy-to-understand presentation of the idea of animation. Action should be presented in a manner easy to understand.

Appeal – This means attractiveness, all that we like to see. It covers character creation as well as the scenario.

Solid drawing – The ability of an animator to create realistic, three-dimensional objects (including characters).

The basic rules of animation presented above concern various aspects of animation (see Table 1). Some of them concern artistry and we do not try to automate artistic elements. We are only interested in the rules connected with the physical aspects of animation.

2.2. Overview of methods

Simplification of animators' work is the main feature of the newly developed animation methods. Animation techniques, apart from 'technical' details of animation, allow animators to focus on the behavior and nature of animated characters instead of solid geometry [4].

Key framing. Key framing is the oldest animation method – the main animator creates figures of characters in some characteristic positions, while the remaining

Table 1. Rules of animation and their aspects

Rules of animation	Aspect
Squash and stretch (deforming) Follow through and overlapping action Slow in and out Arcs	Physical
Secondary action Anticipation Timing	Physical and artistic
Straight ahead action and pose-to-pose action Staging Exaggeration Appeal	Artistic
Solid drawing	Not relevant to computer animation

frames are drawn by less skillful animators in a kind of 'manual interpolation' [5]. Later, 2D animation was made using computers to interpolate basic, characteristic positions [6]. Further development allows to transform key framing into 3D animation [7, 8].

IK – Inverse Kinematics. Inverse Kinematics [9] is another well-known technique. It allows finding end positions of an animated element, *e.g.* hand or foot, using a coordinate system as a referrence set. Forces and moments that cause movement are omitted [9]. A number of methods were developed to transform position and velocity from the character set (joints angles) to the Cartesian system [10, 11].

Physical simulation: dynamics. Systems based only on kinematics are intuitive but the resulting animation is often unrealistic due to gravitation or inertia. Natural movement can be achieved by taking into account forces and momentum. Physical simulation methods can be divided into two groups: with or without constraints. The latter approach is more popular; it mainly consists in parameter adjustment. The former approach requires presentation of features in the form of constraints [12–15].

Behavioral techniques. Behavioral systems can be seen as particle systems in which a number of moving objects are controlled by the same set of relatively simple rules. The entirely system manifests complex and complicated moves ([16–18]).

Optimization methods. Recently, optimization methods of computer animation have been developed. They can be divided into three groups:

- energy minimization [19],
- space-time constraints [20, 21],
- evolutionary algorithms, such as Genetic Programming [1, 5, 22], evolution of computer programs able to control motion.

3. Evolutionary Computation and GEP

Evolutionary **C**omputation (EC) covers a number of methods of good search skills, incl., **G**enetic **A**lgorithms, **E**volutionary **S**trategies, **G**enetic **P**rogramming. They are nature-inspired and often used searching through the solution space as an optimization methods [3, 23, 24].

The evolutionary approach uses a population of potential solutions. The better individuals are chosen as parents for the next population's generation according to an assumed selection method. The quality of individuals is measured using a defined fitness function that reflects how good a solution is coded into an evaluated individual. Stochastic processes imitating those of natural evolution, such as mutation and crossover, are used as a way of differentiating individuals (potential solutions) in the evolved population. The process of artificial evolution used in the EC paradigm is a stochastic but directed (by selection) process that finds better and better solutions in the course of evolution. We can stop it at any time and use the best solution found so far, if we find it satisfactory. If the solution is not good enough, we can run the evolution process once more.

The most popular evolutionary approaches are Genetic Algorithms (GA) [23] and Genetic Programming (GP) [24, 25]. In GA, individuals (potential solutions) are coded as linear chains, usually - but not always - as bit strings of constant length. GP uses the same paradigm as GA but it evolves different structures: instead of linear chains (of bits or real numbers), potential solutions are coded into trees, each individual having a different size, which requires special genetic operators (mutation and crossover).

3.1. GEP

Gene Expression Programming (GEP) [2] is a kind of GP using linear structures as individuals: parse trees are coded into linear strings of constant length. An individual – a genotype represented as a linear structure – is easily decoded into a phenotype, called an Expression Tree (ET). It allows the use of simple, traditional genetic operators. GEP offers similar functionality as GP but it is easier to evolve. In GEP, the process goes according the following pseudocode:

```
Pseudocode 1: Evolution in GEP
Generate of an initial population (random selected chromosomes)
Express of chromosomes (decoding genotypes into phenotypes)
Repeat for each individual in the population
    Fitness calculation (it requires calculation the expressions or running
    the programs)
    Checking the stop conditions
    If stop condition occurs do
        Return the best solution
        STOP
    else do
        Store the best solution
        Reproduction process (creation of the next population's generation)
            Select individuals according to assumed selection method
            Perform mutation and crossover operators
    end if
end repeat
```

3.2. Representation of individuals

Let us consider the following expression:

$$\sqrt{(a+b)(c-d)}. \tag{1}$$

It can be easily presented as the expression tree (ET) shown in Figure 1, where Q denotes a square root. This ET is a phenotype of expression (1).

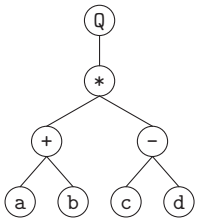

Figure 1. ET representing expression 1

The genotype of this expression can be represented by Chromosome 1.

```
Chromosome 1:
    01234567
    Q*+-abcd
```

The second row represents the chromosome itself but the first row contains the positions (numbered from *zero*) of the corresponding function/terminal symbols in the chromosome.

A genome of an individual in GEP is called a K-expression (from the Karva language) [2]. K-expression can be easily created from an Expression Tree by reading the tree from left to right and from bottom to top. It is worth mentioning that this representation differs from those used in GP, *viz.* post- and prefix ones.

The inverse process – converting a K-expression into an expression tree (ET) – is simple and efficient. A genome in GEP consists of a head and a tail. The head may contain symbols representing functions and terminal symbols. The tail contains only terminal symbols. The head's length, h, is determined depending on the expected complexity of the problem (it defines the maximal size of coded trees). The length of the tail, t, is a function of h and the maximal number, n, of arguments of the functions considered in the K-expression:

$$t = h \cdot (n-1) + 1. \tag{2}$$

Let us consider a chromosome consisting of the alphabet $\{Q, *, /, -, +, a, b\}$. We can see that $n = 2$ and further assume that $h = 10$. We calculate $t = 11$. The length of the chromosomes is equal to $10 + 11 = 21$. An example of such a chromosome can be as follows:

```
Chromosome 2:
    012345678901234567890
    +Q-/b*aaQbaabaabbaaab
```

The ET coded by this chromosome is presented in Figure 2 using only the first 10 symbols. In GEP, individuals may consist of a number of genes, each gene having the same length and coding one subtree. Subtrees form more complex structures. An example of a 3-gene chromosome (with three encoded subtrees) follows:

```
Chromosome 3:
    012345678012345678012345678
    -b*babbab*Qb+abbba-*Qabbaba
```

Depending on the problem under consideration, subtrees can be evaluated separately (*e.g.* when the problem has a number of outputs) or be combined into a complex structure in a predefined way and evaluated as a single structure.

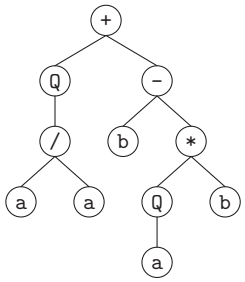

Figure 2. ET encoded in the Chromosome 2

3.3. Genetic operators in GEP

Evolved individuals are mutated and crossed according to the assumed probabilities. Genetic operators have to assure a proper structure of each gene (chromosome).

Mutation can occur in any place in the chromosome, but in the head each symbol can be changed into any symbol and in the tail – only into a terminal symbol.

GEP offers three kinds of *crossover*:

- *1-point crossover*: parents' chromosomes are crossed at a randomly chosen point producing two offspring;
- *2-point crossover*: as above, but two points of cutting are randomly selected;
- *gene crossover*: whole, randomly selected genes are exchanged between two parents chromosomes.

Besides mutation and crossover, three kinds of transposition are defined in GEP:

- *insertion sequence* (IS): a short fragment of the chromosome starting from a function or a terminal symbol is copied into any position in the head of any gene other than the zero position (a root of ET). Symbols of the head in which the sequence is copied are shifted to the right and the last symbols are removed so that the length of the chromosome remains unchanged;
- *root insertion sequence* (RIS): a short fragment of the chromosome starting from a function symbol can be copied at the beginning of the head of any gene in the chromosome (into a subtree root). The whole head is shifted to the right overwriting the last symbols of the head, so that the length of the chromosome remains unchanged;
- *transposition of genes* (GT): the whole selected gene is removed from its original location to the beginning of the chromosome.

3.4. EMOT – an overview of the system

Figure 3 presents a general scheme of EMOT. From an animator's point of view, the GEP, scene service, physical simulation subsystems, and the simulator itself can be treated as black boxes.

The system always has to know the geometry of a scene and the dynamics of each character involved in the simulation – this is EMOT's precondition. The model of a character contains the geometry, mass and inertia of particular joints, a description of the character's possibilities (*i.e.* ranges of joints' angles, forces and momentum of muscles, location of sensors of strength pressure). These values are constant during the animation process. Each sequence of movement has to be evaluated using a fitness

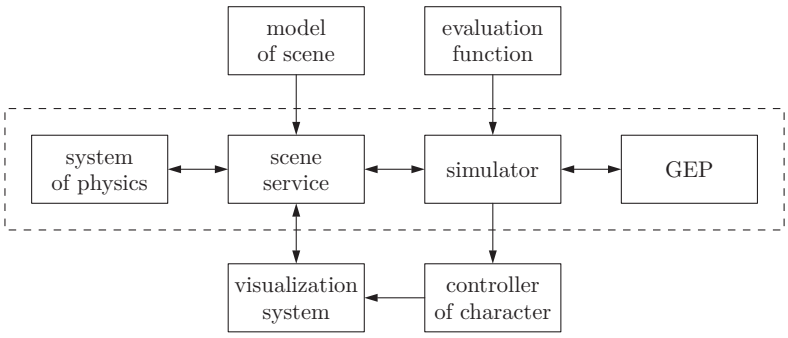

Figure 3. A diagram of the EMOT system

function defined by an animator. The best control program (*i.e.* the best individual) is the system's output.

The system of physical simulation takes care of natural appearance of the animation. In the proposed system, we have used the Open Dynamics Engine (ODE) software package developed by R. Smith. A character is represented as a set of rigid blocks connected by joints, the rest of thr scene, – *e.g.* walls, floors, – are treated as geometrical elements of infinite mass (stationary but interacting with animated characters). Joints moves are limited and each joint is propelled by an angular motor that gives the demanded relative angular velocities to all elements joined with the joint. The velocity is achieved by applying force (each angular motor a has defined power).

A controller of character is calculated (developed) in each step of the physical simulation and it is a list of demanded angular velocities for all joints. From the physical point of view, the resulting move looks quite natural because ODE detects collisions and reactive forces. The pseudocode of simulating moves generated by a character's controller is as follows:

```
Pseudocode 2: Simulation of moves
t=0
while t < time limit do
    make controller-program (calculate demanded angular velocities)
    simulate dynamics with moving ahead delta_t
    t <- t+delat_t
end while
```

3.5. GEP in designing control engines

The EMOT system uses a standard schema of GEP [2]. Each degree of freedom of each joint has to be controlled, so we must generate a control program for each of them. Therefore, a number of genes has to equal the number of degrees of freedom of all joints of the animated character. We assume that each gene codes a control program for one degree of freedom of a character's joint, which means that each gene is an independent program (see Figure 4).

A set of functions and a set of terminal symbols are presumed. Initial experiments allow us to choose adequate sets of functions and terminals. A set of functions consists of the basic mathematical operators and a predefined *IfNeg* function: {+, −, ∗, /, *IfNeg*}, where '/' means 'safety division'. The *IfNeg* function (with three arguments) returns:

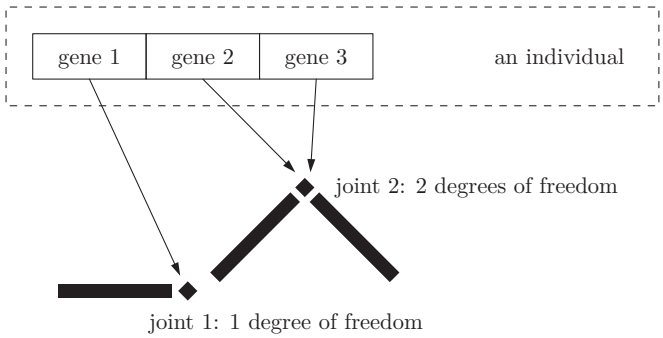

Figure 4. A coding schema

- the value of the second argument (subtree) if the value of the first argument is negative,
- the value of the third argument (subtree) if the value of the first argument is positive or *zero*.

A set of terminals consists of internal variables (position, velocity, angles of joints, external forces, time, *etc.*) taken from the simulation system of dynamics when the program-controller is running and from random real values.

Each of the mini-programs is calculated (run) separately but they usually use state variables of the whole character model. Therefore, subprograms have a tendency to generate mutually dependent moves when it is more efficient than a series of separate moves. A fitness value is calculated for each program on the basis of statistics returned by the dynamic simulation system. The fitness value can be divided into two parts: satisfying the main task and a reward for style. For example, if we try to evolve a character that should move to a given point X, the distance between the character and point X at the end of simulation is a measure of the task's fulfillment. EMOT is able to produce different ways of moving (as some characters crawl rather than jump), so we should take into account the style of movement by enlarging or diminishing the fitness value as a reward for style. A number of categories can be distinguished in stylistic evaluation, namely:

- safety – a penalty for collisions with other objects,
- time – a penalty for inefficient behavior (*e.g.* selection of a longer route or quick movement after long inactivity),
- attaining equilibrium – a reward for remaining in a neutral position at the end of the movement (*e.g.* a character should stand and not lie),
- other – depending on the kind of generated movements.

It is worth mentioning that the initial *raw fitness* should be transformed into *normalized fitness*. In some cases, the raw and *standardized fitness* are the same. *Standardized fitness* means that a better individual has a lower fitness value (we try to minimize it). An ideal individual should have standardized fitness equal to zero. *Adjusted fitness* (f_{adj}) is given by the following equation:

$$f_{\mathrm{adj}} = \frac{1}{1+f_{\mathrm{std}}}, \tag{3}$$

where f_{std} is standardized fitness.

Normalized fitness is adjusted fitness divided by the adjusted fitness of the whole population:

$$f_{\text{norm}} = \frac{f_{\text{adj}}}{\sum_{i=1}^{M} f_{\text{adj}_i}}, \tag{4}$$

where M denotes the size of the population.

Normalized fitness is used in the selection process. EMOT uses *n-tournament selection* [23, 25].

4. Examples and experiments

Two tasks have been chosen for experiments. One, called *Jumper*, is used to study the suitability and efficiency of the approach proposed in EMOT. The other example, a spider we have called *Madzia*, is slightly more complicated task. The obtained results are very interesting.

4.1. Jumper

The *Jumper* example has been selected due to its clarity. The task of EMOT was to teach *Jumper* relocate to a defined place. The *Jumper* model is fully three-dimensional: it consists of five elements (a head, a two-part leg, a bearing and a base) joined by four joints (see Figure 5). Each joint has one degree of freedom and is controlled by the developed program. *Jumper*'s head moves right and left, both parts of the leg move vertically, and the bearing can turn in the plane of base, so that *Jumper* can move in any direction. We have defined *Jumper*'s task as moving to the defined point and standing in such a way that the center of the base is exactly at the defined point.

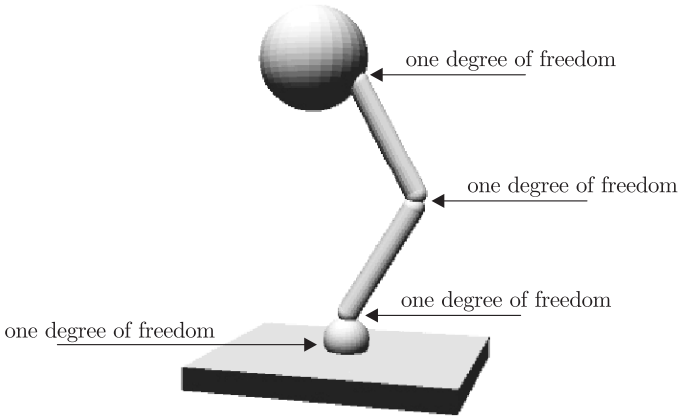

Figure 5. Jumper

A number of fitness functions were tested. The best fitness function found consisted of the following constraints:

1. the main aim: distance between the center of *Jumper*'s base and the desired position at the simulation's end;
2. style: a weighted sum of values ascribed for particular subtasks:
   (a) a penalty for each collision of the head with any element of the scene,

(b) a penalty for too long idleness,
(c) a reward (exactly – a lack of penalty) for a neutral position at the simulation's end.

The used fitness function is presented in the form of Pseudocode 3.

Some values were normalized into the range of [0;1]. We did not want to punish individuals who performed two subtasks relatively well, but failed in the third, therefore we cut fitness at the defined level. Violating the safety constraint (striking the head) caused the worst evaluation. Safety moves were evaluated taking into account tempo, the final position and remaining motionless after achieving the goal.

```
Pseudocode 3: The pseudocode and the fitness function used for Jumper
              evaluation
normalized distance = distance / initial distance
phase = sqrt(recent generation / max number of generation)
style = 0
if safety rules are violated then
  style = style +1
else
  style = style + timing         (do not be inactive)
style = style +sum of angles     (initially sum of angles is equal to zero)
style = style +sum of velocities (stop after achieving the goal)
evaluation = normalized distance + phase * style
GEPFLOAT CEvolution:: GEPFitness(int generation)
{
GEPFLOAT dist, phase, style;
Dist = (lpPhysics -> GetDistance(ID_1,ID_2))/dOriginalDistance;
Phase = sqrt((double)generation/MAX_GENERATIONS)
Style = 0.0;
if (lpPhysics->iGetCollisions(lpPhysics->BodyByID(5),
  lpPhysics->GeomByID(10000)) ||
  lpPhysics->iGetCollisions(lpPhysics->BodyByID(5),
  lpPhysics->GeomByID(10)))
  style += 1.0;
else
{
  style += 0.33*clamp(0.0, 1.0, lpPhysics->GetTiming());
  style += 0.33*clamp(0.0, 1.0, lpPhysics->GetAngleSum());
  style += 0.33*clamp(0.0, 1.0, lpPhysics->GetVelocitySum());
}
return dist+style*phase;
}
```

Taking into account that evolution with simulation is time-consuming, we ran EMOT ten times for 50 generations with the same parameters. All parameters used in the experiments, as well as sets of functions and terminals, are collected in Table 2.

All ten runs ended in success in that EMOT found adequate controllers, but different controllers were produced in particular runs. An exemplary animation of the best individual is shown in Figure 6. Some statistics are shown in Figure 7 (averages of ten runs). A common feature of all solutions (controllers) found is physical correctness, simplicity and surprisingly natural movement.

The length of genes' head influences the size of Expression Trees. Experiments were performed for lengths from 3 to 50. The best individuals were produced for the lengths of 10 and 15 (0.96 and 0.95 average fitness, respectively, see Figure 8),

Table 2. Parameter values used in the experiments

Aim	Controller that relocates *Jumper* from one to another defined point
Set of terminal symbols	t (time) $k0$, $k1$, $k2$, $k3$ (angles of joints) *Oxxx.px*, *Oxxx.py*, *Oxxx.pz* (coordinates of object *xxx*) *Vxxx.x*, *Vxxx.y*, *Vxxx.z* (velocity of object *xxx*) Random constants
Set of functions	$+$, $-$, $*$, $/$, *IfNeg*
Parameters of GEP	Size of population 100 Number of generations 50 Length of a gene head 10 Tournament size 6 Mutation 0.01 Transposition IS 0.1 Length of IS 1, 2, 3, 4 Transposition RIS 0.1 Length of RIS 1, 2, 3, 4 Gene transposition 0.05 1-point crossover 0.2 2-point crossover 0.5 Gene crossover 0.1 Mutation of constant 0.05 Tuning of constant 0.9 Range of constants -6.0 – 6.0
Other parameters	Maximal time of simulation 20.0 Simulation step 0.01 Number of animation slots 2000

while for head lengths equal to 3 and 50 the average fitness equalled 0.34 and 0.32. Too short a head makes the evolution of proper controllers impossible. At the same time, a longer head increases the size of the search space and the task becomes more difficult. The number of successes (per cent) is shown in Figure 9.

Working with GEP, we realized that the population size was an important issue. We tested populations of 10, 30, 50, 70, 100, 200, 300, 500, 1000 and 2000 individuals. Populations of 50 or less individuals were incapable of evolving acceptable solutions. Meaningful changes were observed for a population of 70 individuals. Evolution of 100 or more individuals did not generate substantially better controllers. It seems that the size of 100 individuals is a good balance between the quality of solution and efficiency of GEP.

It is difficult to find the required minimal number of generations due to the use of a scaling parameter for style evaluation in the fitness function (see the *phase* parameter in Pseudocode 3). With small values of P, the assumed scaling parameters given by Equation (5) causes the fitness function to prefer the role of style before suitable controllers are evolved (*i.e.* controllers able to achieve the main goal). At the same time, too long evolution takes into account mainly the styles of evolved controllers, potentially producing controllers that assure smart moves but miss the

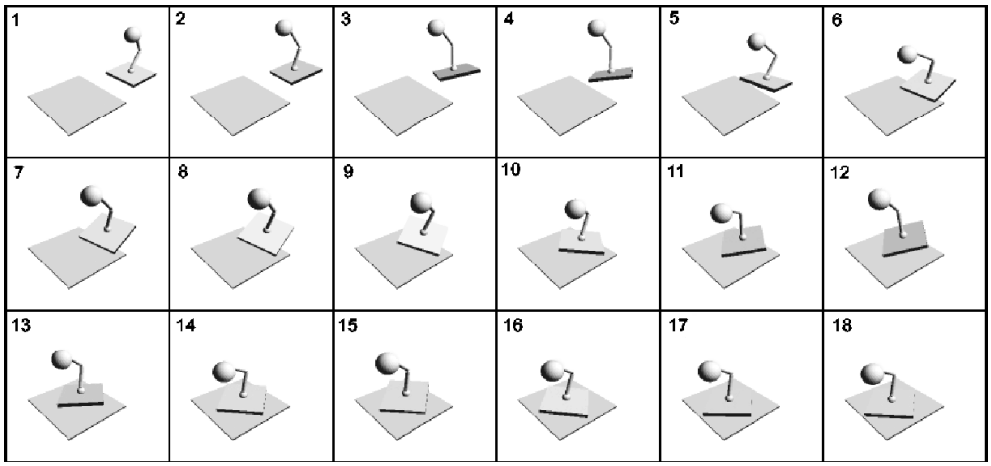

Figure 6. The best *Jumper*, 50th generation of the third run. *Jumper* accessed the desired point precisely after three jumps and remained stable

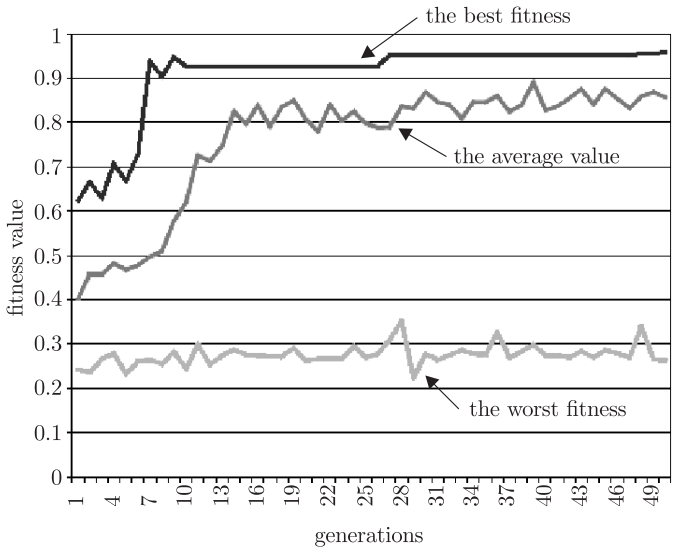

Figure 7. Normalized fitness values in generations (average from 10 runs). The best possible one is one, the worst – zero

assumed goal. Small values of P (10–20) are not suitable, the best results being obtained in 50–200 generations:

$$Phase = \sqrt{\frac{p}{P}}, \tag{5}$$

where p is a number of recent generation, and P is the maximal assumed number of generations.

4.2. The Spider

In this experiment we used a simple model of a spider. Our spider, called *Madzia* (in Polish a diminutive female first name), consists of a trunk and four legs, each

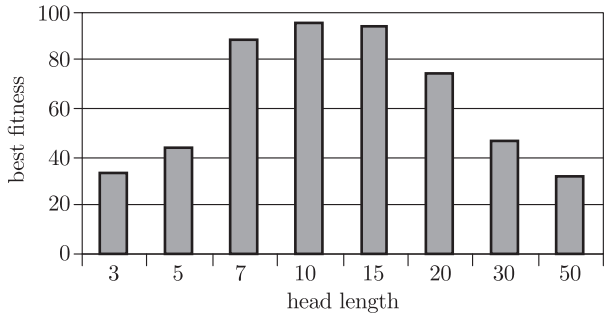

Figure 8. The best fitness values (an average from 10 runs) depending on the length of a head

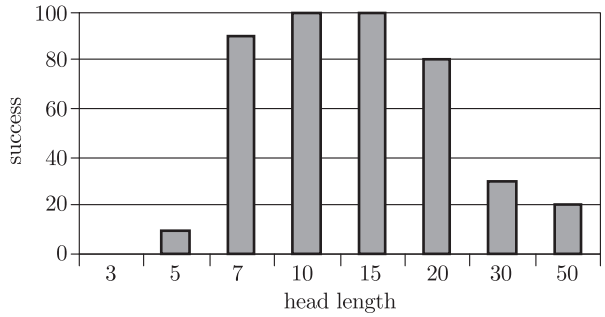

Figure 9. The number of successes (per cent) depending on the length of a head

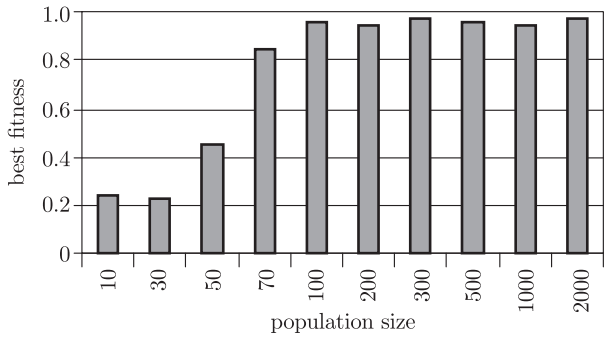

Figure 10. The best fitness values (an average from 10 runs) depending on the population size

with two segments, joined by 12 joints, each joint having one degree of freedom. The main task of *Madzia* was to go strictly over the box. The used fitness function consisted of two parts: a measure of distance and a safety rule (*Madzia*'s trunk could not touch the floor). *Madzia*'s model had more degrees of freedom than *Jumper*'s model, making the search space more complex and the evolution process (together with simulation) more time-consuming. Early experiments had shown that *Madzia* tried to reach the box by jumping rather than walking. In order to force it to walk we set *Madzia*'s mass and strength of muscles very carefully so as to prevent it from jumping.

An exemplary result is shown in Figure 11. *Madzia* amazed us by keeping her equilibrium relatively well but, instead of moving directly over the box, approaching the box so that she could reach it with one of her front legs. Then, she pushed the box

directly under her body (trunk). Having shifted the box a bit further than required, she moved back a little and achieved her goal. This behavior is completely compatible with the assumed aim and the used fitness function, but it came unexpected. In a way it was caused by our mistake: we had set the mass of the box too small compared with *Madzia*'s and she discovered it.

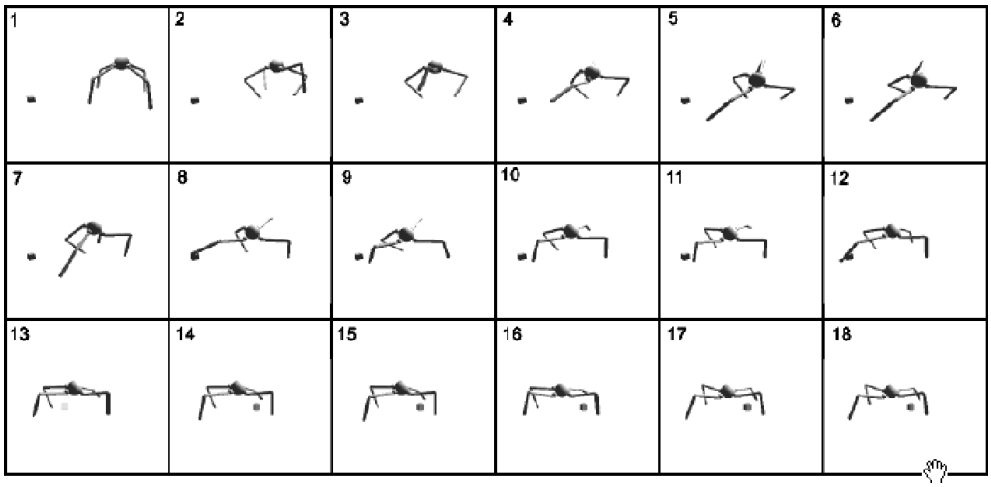

Figure 11. Madzia tries to stand up over the box

In our opinion it is a very natural result generated by artificial evolution. Artificial evolution surprises us by discovering solutions that people have troubles to find.

5. Summary

A number of techniques of automated animation have been developed recently, supporting animators' work. Using dynamic simulation for proper presentation of physical properties of objects and characters produces good results but is very difficult. Majority of approaches require parts of control programs to be written manually.

In the present paper we have proposed to use controllers of characters' moves assuring proper values of angular velocities of joints. Characters are built from rigid blocks connected by joints. Controller programs are not developed manually but by means of Evolutionary Computation, viz. Gene Expression Programming (GEP). Characters controlled by programs evolved using GEP are simulated in a dynamic environment. Controller programs are evaluated on the basis of simulation results, using manually written fitness functions.

Our method generates moves that are credible from the physical point of view, allows for animation goals to be specified and creates animation that is quite natural. The technique is not limited to one type of animation, depending only on the imagination and skills of the author of the fitness function. The results are visually attractive, the moves are fluent and natural, through effects of artificial evolution. As demonstrated by the *Madzia* example, the result may be surprising. We have shown GEP to be an approach useful in supporting automated animation. The presented examples have a relatively small number of degrees of freedom. The possibility of

scaling the method to a higher number of degrees of freedom seems to be a good starting point for future work.

The EMOT system has its disadvantages. The generated controllers are not flexible and a program generated for one move is sensitive to the initial condition and cannot be used for other moves. An evolution of controllers stressing their skills rather than defined tasks seems to be a proper direction of future work, another direction being extension of the method to include 'animation news', *e.g.* objects' deformation.

References

[1] Sims K 1994 *Computer Graphics, Proc. SIGGRAPH'94*, USA, pp. 15–22
[2] Ferreira C 2001 *Complex Systems* **13** (2) 87
[3] Ventrella J J 1995 *Proc. Computer Animation '95*, USA, IEEE Comp. Soc., pp. 35–42 http://www.ventrella.com/Alife/Disney/disney_0.html
[4] Terzopoulos D 1999 *Communications of the ACM* **42** (8) 32
[5] Gritz L and Hahn J K 1995 *J. Visualization and Computer Animation* **6** 129
[6] Burtnyk N and Wein M 1971 *J. SMPTE* **80** 149
[7] Leffler S J, Reeves W T and Ostby E F 1990 *J. Visualization and Computer Animation* **1** (1) 33
[8] Helser A 1988, Electronic document http://www.cs.unc.edu/~helser/291/Animation_using_GAs.htm
[9] Magnenat-Thalmann N M and Thalmann D (Eds) 1996 *Computer Animation*, Prentic Hall, USA
[10] Maciejewski A A 1985 *Proc. 12th Ann. Conf. Computer Graphics and Interactive Techniques*, ACM Press, USA, pp. 263–270
[11] Baerlocher P 2001 *Inverse Kinematics Techniques for the Interactive Posture Control of Atriculated Figures*, PhD Thesis, Ecole Polytechnique Federale de Lausanne, France
[12] Barzel R and Barr A H 1988 *Proc. SIGGRAPH'88*, ACM SIGGRAPH, USA **22**, pp. 179–188
[13] Baraff D 1993 *Eurographics'93, State of the Art Reports*, USA
[14] Baraff D 1989 *Proc. 16th Ann. Conf. Computer Graphics and Interactive Techniques*, ACM Press, USA, pp. 223–232
[15] Hahn J K 1988 *Computer Graphics* **22** (4) 299
[16] Reynolds C W 1982 *Computer Graphics* **16** (3) 289
[17] Reynolds C W 1987 *Computer Graphics* **21** (4) 25
[18] Reeves W T 1983 *ACM Trans. Graphics* **2** (2) 91
[19] Witkins A, Fleischer K and Barr A 1987 *Computer Graphic* **21** 225
[20] Witkins A and Kass M 1988 *Computer Graphic* **22** 159
[21] Ngo J T and Marks J 1993 *Proc. 20th Ann. Conf. Computer Graphics and Interactive Techniques*, ACM Press, USA, pp. 343–350
[22] Reynolds C W 1994 *Proc. 4th Int. Conf. Simulation of Adaptive Behavior*, in "From Animals to Animats 4", MIT Press, USA, pp. 401–410
[23] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley
[24] Koza J R 1992 *Genetic Programming: On the Programming of Computers By Means of Natural Selection*, MIT Press
[25] Kwaśnicka H 1999 *Evolutionary Computation in Artificial Intelligence*, Oficyna Wydawnicza Politechniki Wroclawskiej, Poland (in Polish)