Direct and Indirect Representations for Evolutionary Design of Objects

Juraj Plavcan[1] and Pavel Petrovic[2]

[1] Faculty of Mathematics, Physics, and Informatics,
Comenius University, 84248 Bratislava, Slovakia,
[2] Department of Computer and Information Science
NTNU, 7491 Trondheim, Norway,
ppetrovic@acm.org

Abstract. We follow up on the work of Ebner[1] in studying representations for evolutionary design of objects. We adopt both the method and the simulation framework, and perform more thorough experiments. We design new representations, both direct and indirect, and compare their performance to the original work. We design and make use of a specialised system for distributed computing that integrates smoothly with the EO library[5]. First, we confirm the results of Ebner with VRML scene graphs representation. Next, we demonstrate how both of the new representations based on triangular mesh perform significantly better. Finally, we study and improve the performance of the distributed system that we utilised to run our experiments on tens of computing nodes.

1 Introduction

Evolutionary Design is a promising application area of the Evolutionary Computation (EC) with a large and yet to be discovered potential. Steadily more intricate and specialised designs are needed in the various technological fields, in the scale ranging from the space applications down to the nano level. In order to utilise EC for the design of parts and objects, effective ways of encoding the 3D shapes into genotype representations must be provided and evaluated. Straight-forward direct mapping of unit cells of a 2D or 3D grid into bits of a genotype suffers from huge search spaces, very localised search with operators that cannot span across local extremes, and inflexibility of the search in focusing on the most relevant locations while covering the large even areas with only a few data items. For instance, consider an application of evolving a can opener. Close attention must be paid to the shape of the sharp blade, while the handle can consist of one large cylinder. The genotype representation should be able to dedicate many genes/alleles to the blade and cover the handle with only a few. This distribution need not to be known in advance and should be discovered by the evolutionary algorithm.

Previously, EC has been applied to design various shapes. For example, Robinson et.al. [8] evolve structures for satellite boom with passive vibration

control, i.e. shapes capable of cancelling unwanted vibration by the means of object topology. In [3], Hamda et.al. use indirect representations based on Voronoi diagrams, dipoles, and bars to represent 2D shapes for Topological Optimum Design. Hornby [4] is advocating for reuse of the design components (modules) represented by parts of genotype in particular by applying Lindenmayer systems, an instance of what they define as *generative representations*, where elements of genotypes are reused in their translations to phenotypes. In addition to reusing modules within the same single design, authors reuse modules across individual designs within design families. Ebner [1] addressed the challenge of indirect representations by adopting scene graphs as genotype representation of the shape of objects. Our work repeats Ebner's experiments and moves ahead by comparing them against two new representations. His representations are based on geometric 3D objects as building blocks that do not allow effective use of material and possibility to describe shapes accurately enough. We instead encode objects using a triangular mesh in two different ways: directly, by encoding the mesh node coordinates, and by generating the coordinates through a series of spacial transformations in a manner similar to Ebner's VRML representation. The second of our representations is generative.

Evolutionary experimental runs, demand extensive CPU resources. The runs would take very long to complete on a single computer. We therefore chose to develop a distributed version of the algorithm. The evolutionary engine runs on a master node, which is utilising the available slave nodes to evaluate the individuals in the population (the specific shapes). The evolutionary objective function measures the performance of shapes by simulation in a popular open-source body physics simulator ODE.

In the following sections, we describe the details of the ODE simulator, explain the scene graphs as used by Ebner, introduce the representations based on triangular mesh, unveil the details of our evolutionary algorithm, discuss the distributed system for fitness evaluations, present the results of the experiments and summarise the paper in conclusions.

2 Simulation

Physical simulation usually works with models of a continuous nature, where the state of the objects in the simulated world changes continuously. On the contrary, simulation of bank transactions typically has a discrete nature. In computer simulation, the continuous model is often approximated using a discrete model with small time steps: either as *fixed-increment time advance*, or as *next-event time advance*. We use the former, where the state of the system changes only in the instantaneous moment between short time intervals of constant duration.

2.1 Open Dynamics Engine

Several simulation engines have been developed (for instance Newton Dynamics™ and Havok Physics™). As Ebner, we adopted the open-source project Open Dy-

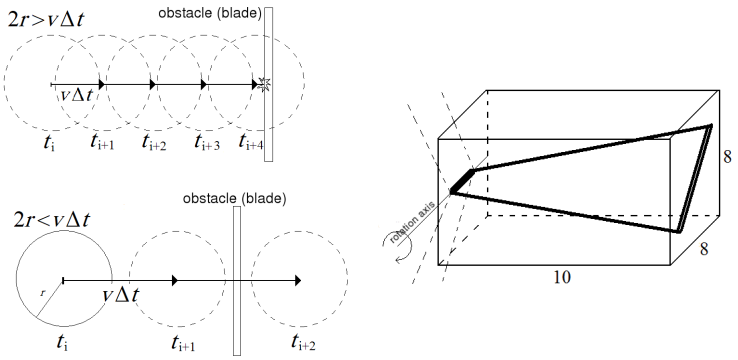

Fig. 1. Relation between the maximum feasible step size and wind velocity (left), reference plane and bounding prism for the rotor blades (right).

namics Engine [10] as the simulating platform. ODE is used for interactive simulation in real time in computer games and robotic simulators (SimRobot [6], UCHILSIM [12], Ubersim [2], Webots [7]) and it prioritises the speed and stability over accuracy and correctness, however, it provides a realistic environment, where the performance of representations can be studied and evaluated. We use ODE for simulation of stream of air particles passing through and colliding with the blades of a wind turbine and causing its rotation in result.

ODE simulation consists of two main components, *Collision Detection Subsystem* and *Physics Engine Subsystem*. The latter works with rigid bodies, objects defined by their position, orientation, speed (linear and rotational), weight, and centre of gravity. The bodies can be isolated or connected with each other using joints to form complex objects. The former is responsible for maintaining the mutual position of the bodies, and for avoiding their overlapping. This requires further attributes, such as volume, shape, 'softness', and 'flexibility'. During a contact, a special type of joint (contact joint) is created for the time period of one simulation step. All joint types together determine the interactions of the simulated bodies and their outcome. The geoms can have simple shapes – box, sphere, capped cylinder, or more complex shapes – defined by a set of triangles that together form a triangular mesh. The joints between the bodies are of different types (ball, hinge, slider, fixed) and have corresponding various degrees of freedom.

Due to the discrete time intervals used, the bodies are "teleported" in time steps. This influences the size of the air particles, which must collide with the blades of the rotor, see figure 1, and as a consequence the speed of the simulation.

2.2 Simulation model

The rotor consists of three blades (each rotated by 120°), attached as one compound object to horizontal axis of rotation (represented by a capped cylinder)

using a frictionless hinge joint. We enforced a constraint for the size of the blades, which ought to fit into a prism with dimensions 10x8x8 (figure 1). Given a specified wind strength (particle speed, weight and density), we observed the maximum angular velocity ω_{MAX}, the blade weight and its distribution (i.e. the distance of the blade's centre of gravity from the axis). Using these variables, we estimated the maximum reached kinetic rotational energy as:

$$E_{MAX} = \frac{1}{2} I \omega_{MAX}^2 = \frac{1}{2} 3 m r^2 \omega_{MAX}^2.$$

The air particles were flying inside of a wind corridor with a circular profile. This cylinder had its bases parallel to the rotational plane. The particles of radius of 0.1 units were generated at the start of the wind corridor with starting velocity $v_0 = 10$. In each step, two forces were applied on the particles: the wind force $F_{wind} = 0.3$ with a vector parallel to the direction of the wind. The resistant force F_d with a vector parallel to the particle's speed vector, opposite orientation, and size linearly proportional to the particle speed:

$$\mathbf{F} = \mathbf{F_{wind}} + \mathbf{F_d} = \mathbf{F_{wind}} - \alpha \times \mathbf{v},$$

where $\alpha = 0.02$ is a friction coefficient and $\mathbf{v}$ is the speed vector. Thus the particles with starting speed $v_0 = 10$ will continue to move with constant speed and direction until they collide with a blade of the rotor, or reach the end of the corridor. At the collision, the particle gives up part of its kinetic energy in favour of the rotational energy of the rotor, and leaves the point of contact according to the law of reflection. Consequently, the wind force will act on the particle to slowly change its speed vector and direct it in the direction of the wind, accelerating to its maximum speed until it leaves the wind corridor or collides again with a blade of the rotor.

The parameters of the model include the size of the wind corridor and the number of air particles. Marc Ebner used a constant-size wind corridor with 100 particles, which were moved to the start of the corridor automatically when reaching the end of the wind corridor. Given the limited total number of steps in the simulation, and the complexity of the blades, we believe this number is too low.

We attempted a more accurate model of the wind. We used smaller time step (0.01 as compared to Ebner's 0.1) and more particles. We optimised the model by variable-length corridor (with the base radius 10.5). The length depending on the "depth" of the rotor. The starting base was placed 0.1 units ahead of the foremost point of the rotor, and the ending base was touching the very rear point of the rotor. The number of particles in cube unit of the corridor was constant (0.7), which in practice meant 500 - 2000 particles in total. The challenge was in generating wind with stable (regular) strength. We tried two approaches:

1. In each step of the simulation, all particles were active. Immediately after leaving the corridor, the particles were restarted. This approach achieves an almost constant total wind energy (sum of the kinetic energies of the particles) during the whole simulation. A disadvantage are large variations

in the number of restarted particles in each step, which result in unwanted resonance and conflict with the criterium for terminating the simulation.
2. in each simulation step, a constant number of particles were started at the start of the corridor. The particles leaving the corridor were erased from the simulation. The number of starting particles in each step was determined empirically so that the number of particles in a cube unit was approximately 0.7. However, leaving this number constant throughout the simulation leads into a large variation in the total number of particles in the simulation and thus as well in the total wind energy. The number of particles depended extensively on the shape of the blades, and therefore was not comparable.

We chose the following compromise: we determined the total number of particles as in the first of the two approaches. However, we limited the maximum number of particles entering the corridor in each step ($restarts_{max}$). If too many particles started to leave the corridor, they had to wait in a backup stack until the number of particles arriving at the end of the corridor decreased under $restarts_{max}$ threshold. This limit was dynamically adjusted depending on the state of the stack: when the stack was growing above safe threshold (20), the limit increase was linearly proportional to the stack size. When it was shrinking (or empty), the limit was decreasing at constant rate. This implementation of "even" wind proved to be stable for different shapes of blades – the wind energy varied little and the wind resonance was balanced.

For a fixed set of wind parameters and specific rotor, there is a maximum angular velocity v_{MAX} it can reach. The goal of the simulation is to determine this speed, which together with the mass and its distribution implies the energy that the turbine can acquire from the wind. Accordingly, we derived the simulation termination criterium: the simulation executes until the angular rotor velocity converges, however always at least 500 and at most 2000 time steps. We consider the velocity converged when the averages over four different history windows vary less than δ.

In order to reduce the time required for the angular speed convergence, the rotors were assigned a nonzero starting kinetic energy, estimated as the half of the maximum kinetic energy reached by any shape currently present in the population. The rotors with lower performance decelerated, those with a higher performance accelerated. See figure 2 left for the effects of this optimisation.

3 Scene Graphs

Marc Ebner focused in his work on scene graphs and compared two existing representations: Open Inventor used in a 3D graphics library and VRML used for virtual reality modelling. Both representations have a tree structure. Internal nodes represent transformations, while leaves represent simple objects: sphere, capped cylinders or boxes. The resulting transformation of the leaf object in VRML representation is defined by gradual (in that order) application of transformations on the path from the tree root, while in the Open Inventor representation, the transformation nodes can be in terminal nodes too, and

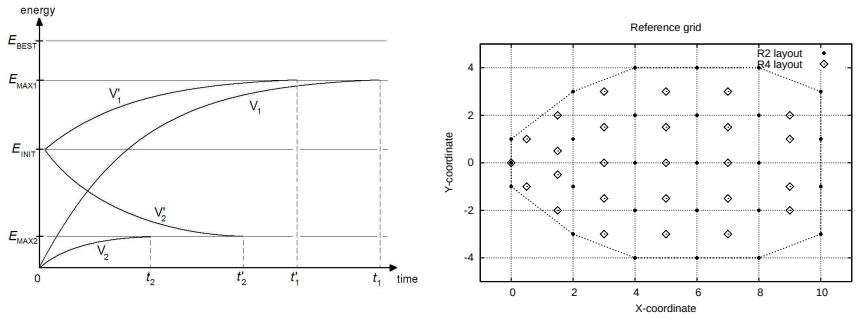

Fig. 2. Simulation time optimisation through initial rotor kinetic energy assignment (left). In most cases, the simulation time is decreased: original time t_1 has been decreased to t_1'. In some cases, the time increases: t_2 to t_2'. Layout for the blade for R_2 and R_4 (right).

the transformations are influenced by special nodes (separators). The resulting transformation in Open Inventor is the composition of the transformations in the order of searching the tree using depth-first search. The separators work as local accumulators and isolate the rest of the tree from the transformations within its subtree; the transformations in the subtree of a separator are applied only to the leaves inside of its subtree.

Ebner applied the two representations to represent the genotypes for the shape of the blade of the wind turbine and used the standard GP for evolving them in a population of 50 individuals, with the tournament selection of size 7, which is known in general to suggest a strong vulnerability to local optima (and we confirmed this also empirically in our early experiments). The parameters of transformations (translation vector, rotation axis and angle) and the leaves (the size of the placed objects) were initialised randomly and did not change during the evolution. This is in contrast with Ebner who used evolutionary strategy to evolve these values, while we preferred manual tuning so that the CPU cycles were used more efficiently during the runs. The standard tree crossover and mutation has been applied with the probability of 50%. Ten evolutionary runs of 200 generations were performed.

Ebner showed that VRML representation outperforms the Open Inventor representation significantly. He argued that the subtrees in VRML representation correspond to specific parts of the blade and thus the recombination and mutation of the root node in the subtree influences only these parts of the blade. The mutations in Open Inventor influence transformations of other subtrees and thus other parts of the blade.

4 Rotor Representations

4.1 Indirect VRML Representation with Objects R_1

We performed experiments with four different representations, R_1 - R_4. In the first, an individual is represented by a tree with branching factor 2–3. Maximal

initial depth of tree is 4. Maximal size of tree is 100 nodes (inner nodes and leaves). Tree nodes are chosen from the set F

$$F = \{R_0, T_0, R_2, T_2, R_3, T_3\}, where$$

$$R_i(x, y, z, \alpha) \in (0; 3) \times (0; 1) \times (0; 1) \times (0; 2\pi), i = 0, 2, 3$$

$$T_i(d_x, d_y, d_z) \in (0; 3) \times (0; 1) \times (0; 1), i = 0, 2, 3$$

R_i represents rotation along axis (x, y, z) of angle α. T_i represents translation along vector (dx, dy, dz). R_0, T_0 are leaves and represent final rotation, and translation of capsule (building block of blade) respectively. Node types R_2, T_2 have two child nodes, R_3, T_3 three. Internal node parameters are initialised randomly and later modified by the mutation operator. The rotor (phenotype) is constructed in ODE environment by depth-first traversing of the corresponding tree (genotype). For every tree leaf, there is one capsule placed. Its position is transformed according to information located on the path from root of the tree to this leave. Capsules, which are not linked to the rotation axis of the rotor directly or via other capsules, are ignored and do not participate in simulation. If the final rotor blade is exceeding the dimensions of the bounding prism (10x8x8), then the individual's fitness is set to zero. Otherwise the simulation is started and kinetic (rotational) energy of rotor after completion of simulation represents the individual's fitness.

4.2 Direct Representation R_2

$$(\alpha, (z_1, b_1), (z_2, b_2), \ldots, (z_{26}, b_{26})), \alpha \in R(0; 1), z_i \in R(-4; 4), b_i \in \{0, 1\}$$

The basis for our direct representation is a reference grid (see 2 right) containing 26 vertices. Coordinates (2D) of the vertices are fixed for all individuals. We set the total number of vertices and their position in reference grid by an expert guess.

Genotype defines elevation of each vertex relative to the reference grid. The elevation is limited, maximum +4 and minimum -4 units. Genotype contains presence bit for each vertex – 0 means the vertex is ignored, 1 means the vertex is part of the blade's shape definition (active vertex). Thus shape of the blade is defined by triangular mesh over elevated active vertices of the reference grid. The last information present in the individual's genotype is an angle of skewness of the blade – the reference grid is rotated by this angle along x-axis.

During the initialisation and mutation of the individuals, dimensions of the corresponding blade are checked against the dimensions of the bounding prism (10x8x8). In the case the blade stretches out of this prism, the elevations of conflicting vertices are adjusted so that the whole blade fits inside the prism. We used Delaunay triangulation to determine a triangular mesh (set of triangles) from the set of active vertices. The triangular mesh defines the front face of the blade. For simplicity, the thickness of blades is constant (0.1 units).

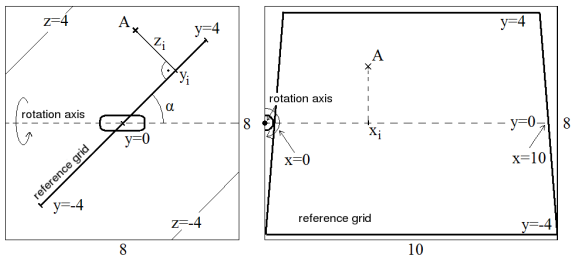

Fig. 3. Direct representation R_2.

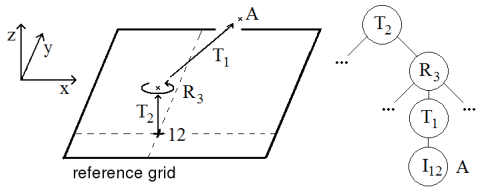

Fig. 4. Transformation in indirect representations R_3, R_4.

4.3 Indirect Representation R_3, R_4

Our indirect representations of blade's shape are based on ideas of both Ebner's VRML representation, and our direct representation R_2 using triangular mesh. We use reference grid with 26 vertices (their position in grid is modified compared to R_2 to prevent the final blade to stretch out of the bounding prism too often, figure 2). All vertices are transformed in space using a transformation tree which represents the whole genotypical information. The tree has a constant number of leaves 26. Leaf contains an index of a vertex from the reference grid (all leaves together cover all vertices). Inner nodes of the tree are either rotation or translation nodes. Branching factor of the tree is 1–3. The transformations on the path from the root of the tree to a leaf were applied in this order on the corresponding vertex of the reference grid. Our first approach to indirect representations – R_3 used fixed triangular mesh – the set of triangles covering vertices of the reference grid was fixed. This resulted in minor complications when vertices moved relatively too far from their original locations. The second indirect representation R_4 solves this problem by projecting already transformed vertices into XY plane. Delaunay triangulation provides set of triangles covering these projected vertices. Moreover, in R_4, the final positions of vertices are restricted so that their projections could not lay outside the boundary displayed in figure 2 right. Another difference between R_3 and R_4 is in the implementation of mutation and crossover operators. R_3 did not check the dimensions of final blade, therefore individuals that represented blades, whose dimensions did not meet our criteria, were also present in population, but during their evaluation in ODE simulator they received fitness zero.

R_4 genetic operators check the dimensions of projection of transformed vertices and also 3D dimensions of actual blade. In case the dimensions exceed limits, the mutation or crossover are tried again (maximum 5 times). If all attempts fail, the original individual (parent) is copied into the next generation instead. The consequence of this modification (R_3 vs. R_4) is the absence of zero-fitness individuals (blades that do not meet size restrictions) in population with R_4. This made the comparison between direct R_2 and indirect R_4 more appropriate, as R_2 adjusts elevation of vertices of reference grid so that they never exceed bounding prism (population with R_2 thus never has zero-fitness individuals).

5 Evolutionary Algorithm

The initial population of 200 individuals was created using the RHH method as in the work of Ebner. We used tournament selection of size 2, weak elitism, and ran for 200 generations. Common parameters were tuned to $p_{cross} = 0.3$, $p_{mut} = 0.7$.

In the case of R_1, the standard subtree crossover, and four different mutations were used: *BranchMutation* replaces a subtree with random content, *PointMutation* replaces a tree node with another tree node of compatible arity, *CollapseSubtreeMutation* replaces a random subtree with a new leaf, and *ExpansionMutation* replaces a leaf with a randomly generated subtree. The mutation probabilities were 0.5, 0.3, 0.1, and 0.1, respectively in the same order as above. In the case of R_2, we used the standard one-point crossover and simple average crossover operating on skewness angle. Since the points are topologically ordered, the crossover corresponds to geometric crossover, cutting the blade along a straight line. Because only some of the points of the grid are present in the individual phenotypes, this slight bias towards straight lines is partially compensated. We believe it is important to exchange the large parts of the blade using the crossover while the mutation with a higher rate tunes the correct details of the shape. Mutation operators were of four types: *AngleMutation* adds Gaussian noise to the skewness angle, *GaussAllMutation* alters the heights z_i using Gaussian $N(0, \delta)$, *Uniform3Mutation* alters three randomly selected z_i by a uniform noise from interval (-4,4), *PresenceMutation* toggles the presence of three randomly selected vertices. The skewness angle always remains in the interval $(0,90^0)$, and all heights z_i in the interval (-4,4). The probabilities of the mutations were 0.1, 0.5, 0.3, 0.1 in the order as above. The operators are implemented so that the resulting individual fitted into the bounding prism. In the cases of R_3 and R_4, we use a kind of swap crossover operator, while paying attention to the leaf nodes, which must form a permutation of the 26 vertices (at most three transformation nodes are swapped). Four mutation operators were used: *ContentGaussMutation*, and *ContentUniformMutation* apply Gaussian and uniform noise to all parameters in a single selected node, *SwapSubtreesMutation* swaps two random subtrees, and *SwapLeavesMutation* swaps two random leaves. The ratios used were 0.4, 0.2, 0.3, and 0.1, respectively. The experiments were implemented with the universal open-source evolutionary library EO. We only

made small modifications of the engine in the way the user's call back function is called in order to allow for the distributed evaluation of the objective function.

6 Distributed Evaluation

To perform the experiments, we utilised the idle CPU time of computers in the student laboratories and of several powerful server machines. The evolutionary algorithm was running on a master node, which communicated with the slave nodes at an application layer using MySQL database. In each generation, the master program sent a set of individuals to be evaluated into a database table, and waited until the whole population became evaluated.

The nodes retrieved one individual each, ran a simulation and stored the computed fitness back to the database table. In order to increase the utilisation of the computational nodes, and because of their varying performance, some individuals could be evaluated simultaneously by several nodes. This occurs when all individuals have either been evaluated or are already assigned to some node and there are still some nodes available. This cures the situation when a difficult individual is assigned to a node with low performance, which would otherwise block the progress of the whole evolution for tens of minutes, leaving all remaining nodes idle. Close to the end of each generation, the extra available computational nodes spread over the whole set of the remaining individuals evenly, prioritising those that are already being tried for the longest time.

7 Results

After many testing runs focused on tuning the parameters and understanding the underlying mechanisms, we performed 11 evolutionary runs for each of the four representations. Each representation required about 1 year of total single-CPU time. Figure 5 plots the best and average fitness for all runs against the number of generations. The actual number of evaluations was slightly lower in the case of direct representation due to different sequence of mutation application, but the trend and the result is the same as in the plot against generations. All three triangular mesh representations (both direct and indirect) clearly outperform the original representation based on VRML and capped cylinders. The best result is achieved with R_3, the first version of the indirect representation (VRML) with triangular mesh (significantly better than direct representation, t-test with $t = 4.463$), and the trend is more promising. However, this is likely due to the capability of R_3 of accumulating mass by creating "folds", and due to the possibility of exploiting additional areas close to the rotation axis, which are not part of the search space of the direct representation. This option was removed from otherwise identical R_4, which performed worse than R_2, the direct representation with triangular mesh, even though still better than the original R_1.

There are couple of observations, which can explain lower performance of the original VRML representation with capsules. The initial population contained

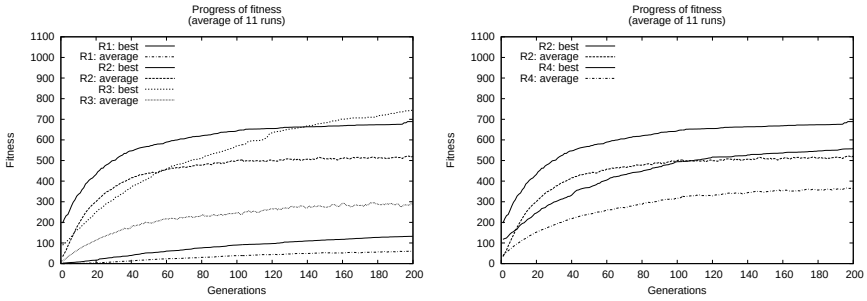

Fig. 5. Plot of the best fitness for R_1 - R_3 (left) and R_3, R_4 (right).

smaller trees, which in turn generated small blades accumulating little kinetic energy. This contributed to lower capacity of the evolution to explore the space with good solutions as it needed to spend more efforts on finding the proper region of the search space. In addition, an important aspect is the shape and size of the cutting edge of the blades. Using capsules implies thick blades with large friction against the air during the rotation. The figure 6 depicts the evolved shapes. Note their similarity to water turbines, rather than air turbines, which is due to our model that is not intended to be physically plausible. Note also how the evolution exploits the possibility to create folds to accumulate the weight.

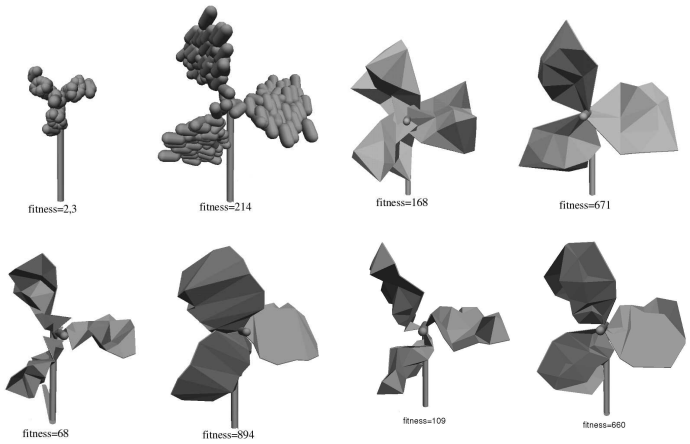

Fig. 6. Examples of evolved blades for the four representations R_1 – R_4, the best of the first and last generation.

Part of the genome in the direct representation was the angle of skewness of the reference plane. This variable converged to values around 10°, runs with indirect representation used a fixed 18° skewness.

8 Conclusions and Future Work

We follow up on the work of Marc Ebner in studying representations for evolutionary design of objects. We make a completely independent re-implementation and confirm his main results. We design two new representations, where the shape is defined as a set of triangles (triangular mesh). We demonstrate that the representation is more suitable for this problem and in general as it allows higher flexibility, better accuracy, and interesting genotype representations. We design both a direct and indirect (*generative*) representations, both of them performing significantly better than the original representation. We design a distributed system that utilises idle CPU time of computational nodes in student laboratories.

The future work will focus on studying new indirect representations that could outperform direct representations. A large number of possibilities exists, and remains only for a creative mind of the experimenter as we make the source-code publicly available [11], some more details of the experiments can be found in [9]. The same simulation framework can be evaluated with other problems of evolutionary design, and perhaps enabled with active controllers for the mutual co-evolution of robot morphologies and controllers.

References

1. Ebner, M.: Evolutionary Design of Objects Using Scene Graphs. Proceedings to Genetic Programming: 6th European Conference, EuroGP, Springer (2003) 47–58.
2. Go J., Browning B., Veloso M.: Accurate and flexible simulation for dynamic,vision-centric robots. In Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'04 (2004).
3. Hamda H., Jouve F., Lutton E., Schoenauer M., Sebag M.: Compact Unstructured Representations for Evolutionary Topological Optimum Design. Applied Intelligence, 16 (2002) 139–155.
4. Hornby, G.S.: Generative Representations for Evolutionary Design Automation. PhD Thesis, Michtom School of Computer Science, Brandeis University, MA (2003).
5. Keijzer, M., Merelo, J.J., Romero G., Schoenauer M.: Evolving Objects: a general purpose evolutionary computation library. Artificial Evolution (2001) 231–244.
6. Laue T., Spiess K., Rfer T.: SimRobot – A General Physical Robot Simulator and Its Application in RoboCup. In A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi (Eds.), RoboCup 2005: Robot Soccer World Cup IX, No. 4020, Lecture Notes in Artificial Intelligence. Springer (2006) 173–183.
7. Michel O., Cyberbotics Ltd. Webots^{TM}: Professional Mobile Robot Simulation. Int. Journal of Advanced Robotic Systems, Vol. 1 Number 1 (2004) 39–42.
8. Robinson G., El-Beltagy M., Keane A.: Optimisation in Mechanical Design. In Bentley, P. (ed.), Evolutionary Design by Computers, Morgan Kaufmann (1999).
9. Plavcan J., Petrovic P.: Distributed Evolutionary Experiments with Representations for Evolutionary Design. IDI NTNU Technical Report (2007).
10. Smith R.: Open Dynamics Engine Users Guide (2006).
11. RoboWiki: Evolving Shapes (2007), http://robotika.sk/projects/evodesign/.
12. Zagal J.C., del Solar J.R.: UCHILSIM: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes in Artificial Intelligence, Springer (2004).