evoVision3D: A Multiscale Visualization of Evolutionary Histories

Justin J. Kelly(✉) and Christian Jacob(✉)

Department of Computer Science, University of Calgary,
2500 University Dr NW, Calgary, AB T2N 1N4, Canada
{kellyjj,cjacob}@ucalgary.ca
http://www.ucalgary.ca

Abstract. Evolutionary computation is a field defined by large data sets and complex relationships. Because of this complexity it can be difficult to identify trends and patterns that can help improve future projects and drive experimentation. To address this we present *evoVision3D*, a multiscale 3D system designed to take data sets from evolutionary design experiments and visualize them in order to assist in their inspection and analysis. Our system is implemented in the Unity 3D game development environment, for which we show that it lends itself to immersive navigation through large data sets, going even beyond evolution-based search and interactive data exploration.

Keywords: Evolutionary computation · Multiscale · Visualization · Game engine

1 Introduction

It is said that history is the greatest teacher. Sometimes in order to move forward one must review past decisions and choices in order to identify common trends and patterns to predict future outcomes. This historical evaluation is especially valuable in interactive evolutionary algorithms [6] and genetic programming [12], where users review past experiments and trends in order to improve and refine their selection algorithms and fitness evaluations. However, evolutionary systems often produce very large data sets filled with complex relationships, making it difficult for a human to effectively process. Additionally, there are times when a system's requirements can suddenly change, rendering previous evaluations insufficient and forcing the user to begin their review from scratch. To address these issues we present *evoVision3D*, a multi-level visualization environment, displaying complex evolutionary data in a 3-dimensional, immersive scene (Fig. 1). In this paper we will explore *evoVision3D*'s features and how we have expanded upon *evoVersion*, an evolutionary data tracking and synchronization tool, we have developed earlier [10].

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 942–951, 2016.
DOI: 10.1007/978-3-319-45823-6_88

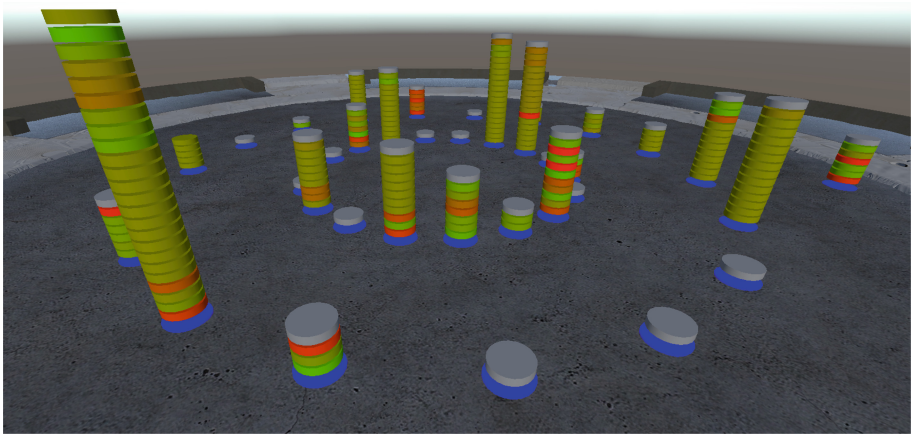

Fig. 1. Example of an evolutionary design workspace in 3D virtual reality using *evoVision3D*: columns represent experiments, disks denote populations, whose colours depict average fitness.

2 Related Work

With the advent of highly capable video game development environments, such as Unity 3D [4], it has become more and more common to use game engines for scientific research and visualization [15]. Taking advantage of advanced visualisation libraries and built-in physics engines, there is substantial opportunity for their integration into professional research. In this paper, we present related work in the areas of (1) evolutionary visualization, (2) VR technology and (3) *evoVersion*, one of our previous systems implemented in the Unity 3D Game Engine.

2.1 Evolutionary Visualization

Building upon the foundation laid by *evoVersion*, we draw inspiration from previous works. We combine a node-ring graph visualization [9] with a multiscale visualization model [14] to display data efficiently and with a dense arrangement of visual information without becoming overwhelming. We organize each session into a set of discrete generations represented by a series of stacked rings (Fig. 1).

We have drawn inspiration from the *EvoShelf* system, which applies techniques normally found in photo management software, organizing evolutionary data in a manner reminiscent to programs such as iTunes™ [8]. We use a similar modular design, providing a flexible and plug-in friendly environment. As we will demonstrate, the straightforward presentation of data makes searching through larger populations smoother and less cumbersome. *evoVision3D* differs from *EvoShelf* due to our use of 3D visualization rather than 2D with respect to result presentation and navigation (see Sect. 3.1). With *evoVision3D* we provide a tool to coordinate collaborative development among multiple users, rather than just one unsynchronized account.

We expand on Daida et al.'s work on mapping expression trees to a circular 2D grid, which provides a simple visualization, facilitating the identification of trends and patterns across a genetic programming session [7]. In comparison, *evoVision3D* enables data inspection, filtering, and analysis across multiple experiments. A visual analytics interface for evolutionary data has been discussed in [13]. More traditional 2D scatterplots are used to inspect and categorization data. In contrast, *evoVision3D* expands the data presentation to 3D and across multiple evolutionary sessions.

2.2 VR Technology and 3D Game Engines

Following *evoVersion* [10] and Shepherd's genome browser [15], *evoVision3D* is built using the Unity 3D game engine [4]. With the recent increase in public availability for professional-quality game engines and their active developer communities, these engines have proven to be an extremely valuable asset in the development of visualization systems. A notable example of this is Unity's built-in support for virtual reality systems such as the Oculus Rift [3], allowing for easy integration of these systems into immersive data display solutions.

2.3 *evoVersion*

evoVersion is a system designed to collect, store and visualize interactive evolutionary data. *evoVersion* utilizes the iterative storage methodology of software version control systems such as Git [2] and Subversion [1] and applies it to evolutionary computation in order to record, organize and analyze the resulting data. It consists of three primary components: interactive selection, data storage and basic 3D visualization. The interactive selection component handles user-driven evaluation and evolution of the phenotype population. The data storage component records all iterations of the population on a remote SQL server. The visualization component takes the data stored in the database and visualizes histories of evolutionary designs in a column-based format (Fig. 2). *evoVision3D* builds upon this system and focuses on improving the functionality and performance of the visualization component with regard to the existing data collection and storage mechanics while using the data sets produced by *evoVersion* as the primary data source.

3 The *evoVision3D* System

evoVision3D seeks to build upon the visualization scheme seen in *evoVersion* and provide the user with intuitive and efficient means of visualizing complex evolutionary data sets from various evolutionary experiments. To achieve this *evoVision3D* combines *evoVersion*'s data arrangement with an additional set of features in order to further assist the user as they examine the data visualization space. These features include: (1) a spatial arrangement of the visual data representations, (2) a multi-level abstraction of data, (3) genealogy tracing for specific elements, (4) similarity filtering, and (5) a set of dynamic interface panels summarizing key details and statistics of a given element.

3.1 Data Arrangement

In *evoVision3D*, each of a user's evolutionary experiments are treated as a distinct event with a series of discrete populations arranged in ascending historical order. Each evolutionary experiment, or session, is represented by a single vertical column (Fig. 1). The height of this column reflects the number of generations during that experiment, allowing the viewer to easily determine which sessions were the most active. Session columns are arranged in a spiral pattern, growing from previous sessions towards the center to the newest around the outer edge.

3.2 Multiscale Abstraction

A common problem encountered when visualizing these kinds of data sets in 3D is the limits of computer memory and rendering capabilities, making it impractical to fully render each individual phenotype at once when dealing with larger data sets. To address this challenge, *evoVision3D* utilizes multiscale abstraction of the data sets to reduce the computational overhead incurred during the visualization, improving both load times and frame rate significantly. Similar to [5], *evoVision3D* uses multiple levels of visualization. Each level differs in terms of breadth and detail of the data portrayed in order to collect both general and specific details with regards to evolutionary design histories.

Based on a hierarchal storage structure, our system currently operates on multiple levels of detail: sessional, generational, and individual (Fig. 2). The sessional level data is represented as a series of stacked disks arranged to form a column. Each of these disks represents a single generation of that session. Disks are arranged in ascending order of creation, placing the first generation at the bottom and the most recent generation at the top. The color of the disk denotes the average fitness of the entire population at that point in time during the experiment. Each color lies along a linear gradient between red and green, where

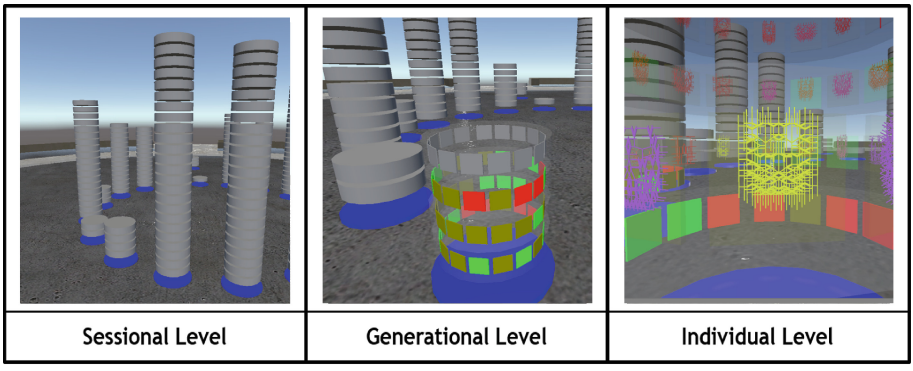

Fig. 2. The three levels of scale used in *evoVision3D* in descending order of detail. Automatic scale transition is triggered by the user approaching a specific object. (Color figure online)

red denotes a fitness of 0 (worst rating) and green represents a fitness of 100 (best rating). Alternatively, an object colored gray either has yet to be evaluated or has had its coloring toggled off by the user. By observing the color of each disk within the column the user can get a general feel for the quality of a session (Fig. 1).

At the generational level (Fig. 2b), each disk allows the user to view a series of nodes arranged in a ring within the disk. Each node, depicted as a distinct 3D object, represents a single member of a population. The color of each node represents the specific fitness of its corresponding element. This gives a more detailed breakdown of a generation's population without the need to render each individual phenotype, while also providing a quick visual summary of their fitness ratings.

At the individual level (Fig. 2c) the system renders the individual phenotypes of the elements within a generation. The nodes from the generational level lose their transparency and a representation of that element's phenotype is rendered inside the node. This presumes that a visual representation is available for each element. This allows the user to see a depiction of the element in combination with its fitness, represented by the color hue.

Each of these levels are rendered dynamically on demand. This keeps the memory overhead for the system minimal, while also reducing the amount of content loaded when the visualization engine initializes. This allows the system to maintain a high degree of efficiency even when rendering large data sets. The transition between each level of detail can be both manually and automatically triggered as needed. Automatic transitions are triggered based on the user's position relative to the session columns in the scene. A generation disk enters the generational level of detail when the distance between their position and the center point of a given disk is less than a user-defined value (Fig. 3). When this distance once again becomes greater than this user-set value the disk will return to its previous sessional level of detail (an opaque colored disk). The individual level of detail is triggered when the distance between the user and a disk is less than the radius of a generation's disk. In order to automatically trigger this transition the user enters the column in question, providing a 360 degree panoramic view of the local population. As with the generational level, the disk returns to its previous level of detail when the user exits the column space of that particular session.

Manual transitions can be invoked through key strokes at any time and will set the entire scene to a specific level of detail without regard to the user's position in the scene. These manual modes can be useful when trying to identify trends at a certain level of detail, allowing the user to navigate through and inspect the visualization space. This allows the system to maintain a minimal amount of wait time to load each scene.

3.3 Genealogy Tracing

One of the key aspects to evolutionary systems is their application of iterative development. New elements are derived from pre-existing elements through a

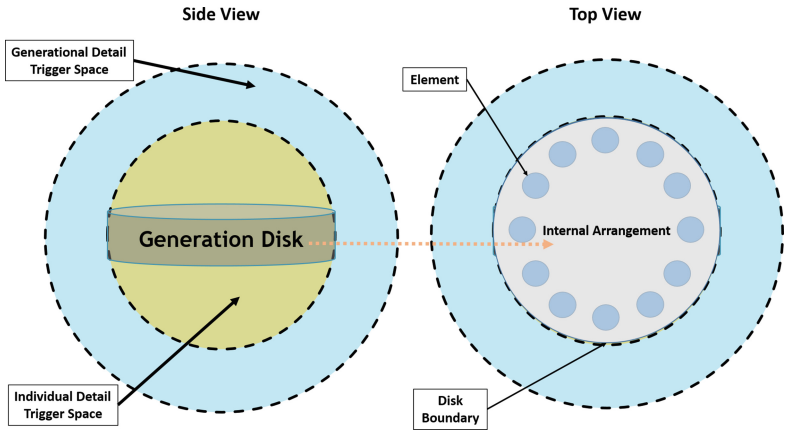

Fig. 3. An illustration of the conceptual boundaries used to trigger scale transitions in the visualization space. As the user approaches a generation disk its visual models become more and more detailed.

combination of crossover and mutation operations. Mutation is the process of applying a random modification to an existing genotype, while crossover is the act of combining two or more genotypes to produce a new child that shares a part of each parent's genotype combined together [10]. It is therefore quite valuable to maintain an understanding of an element's ancestry in order to help identify trends and patterns created through inheritance. In *evoVision3D* one can trace through a targeted element's genealogical history in order to identify and visualize the relationship it shares with its ancestors (Fig. 4). Serving as a filter, the genealogy trace removes all objects not related to the selected element from the scene and sets all remaining nodes to the individual detail level. The system then procedurally generates a series of line segments, where each line represents the relationship between a parent and its children. These connections produce a 3-dimensional family tree for the selected element, allowing the user to observe the genetic changes that culminated in the production of the target element. This operation uses a breadth-first expansion down the generations, allowing the user to examine the connections from more recent generations to older generations.

3.4 Similarity Filtering

Similarity filtering allows the user to identify what sections of a user's experiment set occupy the same genotype neighborhood. It allows the user to select an element and calculate its similarity to all other elements present in the scene. All elements whose similarity falls below a user-set threshold are filtered out, leaving only those individuals that have significant similarities to the selected element (Fig. 5). The colors of these individual nodes and the encompassing generational disks are changed from visualizing fitness to instead reflect the

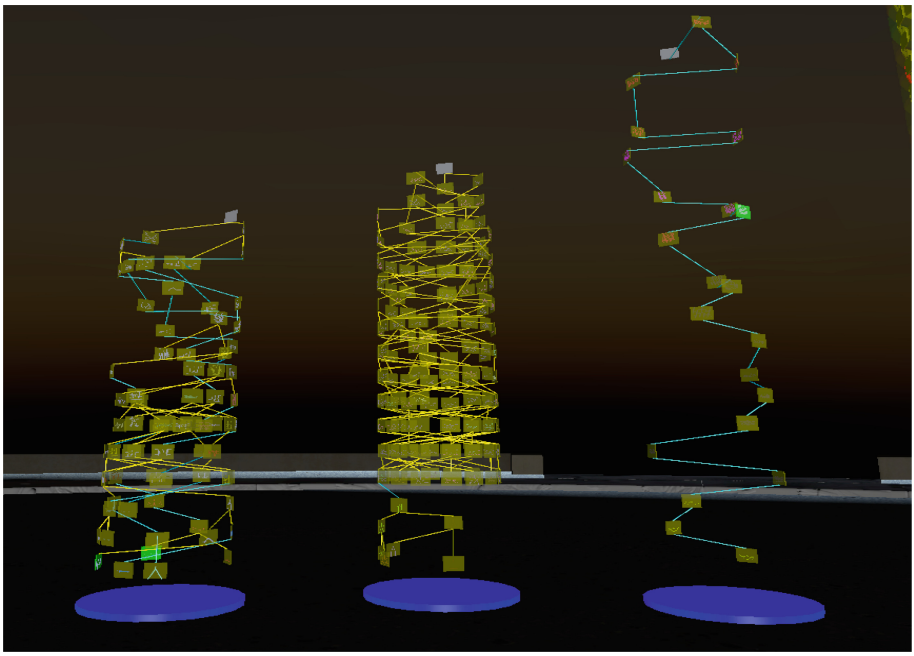

Fig. 4. Visualizing an element's genealogy: line segments illustrate the relationship between parent and child (from top to bottom in each column). Cyan lines indicate a mutation while yellow lines denote a crossover relationship, allowing for quick identification of development patterns within a session. The left column shows a mix of mutation and crossover, the middle session evolved mostly by crossovers, whereas only mutations created the individuals in the session on the right. (Color figure online)

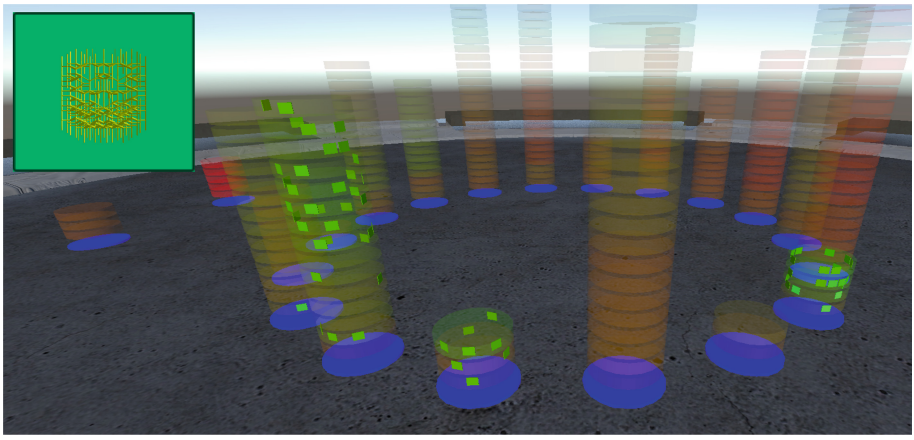

Fig. 5. Filtering the scene based on genetic similarity. The chosen phenotype is presented in the top left of the screen and only the nodes that have reached a user-defined degree of similarity are rendered in the scene. (Color figure online)

degree of similarity to the chosen element. Green denotes high similarity, while red denotes low similarity.

3.5 Dynamic Summary Panel

The multiscale representation and color encoding provide an effective summary of an element's phenotype and fitness. *evoVision3D* supplements this by providing three dynamic interface panels with a summary of an object's data and statistics. This function dynamically loads the data of any selected object in the scene, producing a summary of the underlying data, a close up view of its phenotype (if applicable) and a comparison of its fitness compared to all other objects in the scene (Fig. 6).

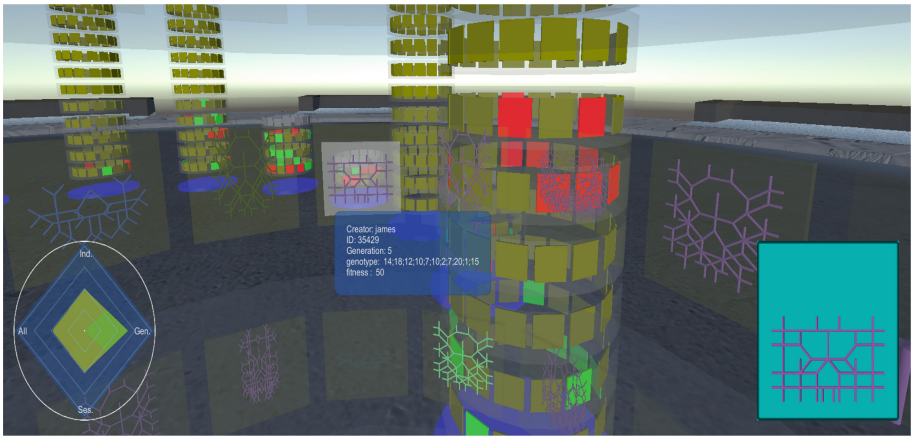

Fig. 6. Panel overlay. By selecting an object in the visualization space a summary view of key information is displayed, including summary data (center), phenotype viewer (to the right) and fitness statistics (to the left).

Data Panel. The data panel serves as a summary of an item's data as stored in the SQL database. Appearing next to the user's cursor, it consists of a translucent back panel and a textual output of key information. The information changes depending on the detail level. At the sessional level, a generation's disk displays high level information such as the population size at that generation and the ID number of the associated session. Alternatively, at the individual and generational levels the panel instead displays information for the now visible individual.

Phenotype Viewer. Displayed in the bottom right corner of the screen (Fig. 6), this panel allows the user to quickly check an element's phenotype without having to trigger the individual level of detail. This is useful for inspecting individual elements while operating on the generational level of detail, without the need

to reposition the camera while still allowing the user to rotate and magnify the viewer space. This can assist in the identification of patterns shared by elements such as, for example, a certain fitness range via manual observation and selection.

Radial Glyph. Expanding on *evoVersion*'s radial fitness rings [10], we added a glyph display contained inside a circular boundary, depicted in the bottom left corner (Fig. 6) This panel consists of two primary components: a measurement guide and a data map. The measurement guide is a translucent blue diamond that serves as a form of relative measurement for the given data. Each of the four points on this diamond represents a specific attribute. Starting from the topmost point and moving clockwise these points measure the fitness of the current target (Ind), the average fitness of the current target's generation (Gen), the average fitness of the target session (Ses) and the average fitness for every single session currently in the scene (All). The respective values are plotted along the line between the center of the guide and its corresponding corner. The higher the value, the closer its point is plotted to the outer edge of the guide. The points are connected to form an irregular n-sided shape that, for example, represents the target's fitness rating compared to other elements in the scene. This model of representation can easily be expanded to include more than four values.

4 Conclusion and Future Work

evoVision3D is a promising avenue for visualizing and exploring evolutionary histories. Implemented in the Unity game engine [4], it provides a robust suite of navigation and filtering tools for evolutionary data inspection and analysis. This level of performance also makes VR support viable for the system as a whole, allowing for a new avenue of immersive visualization to be explored. At this point our future work is threefold: the expansion of existing features, integration of gesture control to support immersive VR interaction and extending use of the system to more complex evolutionary systems in order to test its viability in the context of dense evolutionary data visualization [11] and collaborative coevolution [16]. We plan to expand upon the current color encoding and radial glyph graphs used to analyze and compare the fitnesses of individual elements and generations. We are also working on a flexible search tool capable of visualizing the similarity between population elements in terms of both genotype and phenotype features to assist in identifying commonalities and trends within the data set. The addition of gesture control to supplement peripheral systems such as the Oculus Rift Virtual Reality Headset [3] makes navigation through the virtual space more natural and intuitive, providing a logical alternative to the traditional mouse and keyboard interaction. Finally, in order to further reduce the computational overhead and allow for rendering of even larger scenes we plan to apply a dynamic octree implementation similar to that seen in Shepherd's genome exploration system [15] to further improve performance.

References

1. Apache subversion; enterprise-class centralized version control for the masses. https://subversion.apache.org/
2. git -local-branching-on-the-cheap. https://git-scm.com/
3. Oculus. https://www.oculus.com/en-us/
4. Unity 3d game engine. https://unity3d.com/
5. Stolte, C., Tang, D., Hanrahan, P.: Multiscale visualization using data cubes. IEEE Trans. Vis. Comput. Graph. **9**(2), 176–187 (2003)
6. Coello, C.A.C., Van Veldhuizen, D.A., Lamont, G.B.: Evolutionary algorithms for solving multi-objective problems, vol. 242. Springer, New York (2002)
7. Daida, J.M., Hilss, A.M., Ward, D.J., Long, S.L.: Visualizing tree structures in genetic programming. Genet. Programm. Evolvable Mach. **6**(1), 79–110 (2005)
8. Davison, T., von Mammen, S., Jacob, C.: *EvoShelf*: a system for managing and exploring evolutionary data. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 310–319. Springer, Heidelberg (2010)
9. Etemad, K., Carpendale, S., Samavati, F.: Node-ring graph visualization clears edge congestion. In: Proceedings of the IEEE VIS Arts Program (VISAP), pp. 67–74
10. Kelly, J., Jacob, C.: evoVersion: visualizing evolutionary histories. In: IEEE Congress on Evolutionary Computation, CEC 2016. IEEE (2016) (in print)
11. Koçer, B., Arslan, A.: Transfer Learning in Genetic Algorithms (2012)
12. Koza, J.R.: Genetic Programming: On The Programming of Computers by Means of Natural Selection, vol. 1. MIT Press, Cambridge (1992)
13. Lutton, E., Fekete, J.D.: Visual analytics of EA data. In: Proceedings of the 13th Annual Conference on Genetic And Evolutionary Computation - GECCO 2011, pp. 145–146 (2011)
14. Miller, R., Mozhayskiy, V., Tagkopoulos, L., Ma, K.L.: EVEVis: a multi-scale visualization system for dense evolutionary data. In: 2011 IEEE Symposium on Biological Data Visualization (BioVis), pp. 143–150 (2011)
15. Shepherd, J.J., Zhou, L., Zhang, Y., Zheng, J., Tang, J.: Exploring genomes with a game engine. In: Proceedings - 2013 IEEE International Conference on Bioinformatics and Biomedicine, IEEE BIBM 2013, vol. 169(207890), pp. 26–30 (2013)
16. Yang, Z., Tang, K., Yao, X.: Large scale evolutionary optimization using cooperative coevolution. Inf. Sci. **178**(15), 2985–2999 (2008)