Evolving 3d model interpretation of images using graphics hardware

Fredrik Lindblad Peter Nordin Krister Wolff

Complex Systems Group
Department of Physical Resource Theory
Chalmers University of Technology
SE-41296 Göteborg, Sweden
{nordin, wolff}@fy.chalmers.se

Abstract - We present a novel approach for 3d-scene interpretation with numerous applications, for instance in robotics. The models are rendered using 3d graphics hardware and DirectX. Both artificial and real images were used to test the system. More than one target image can be used, allowing stereoscopic vision. These experiments present results of interesting generalization.

I. INTRODUCTION

Vision, or interpretation of images, is one of the tasks that are easily performed by the human brain, but that are very cumbersome for computers. A program that could quickly interpret an image in terms of three-dimensional models would be very useful for robotics. With a 3d model that matches the scene that a robot sees, it could understand its environment better and move smarter. Other applications could be image compression. Since most real images and movies depict a three-dimensional world, a representation in the same number of dimensions could possibly be the most effective choice [1]. Graphics processors nowadays can in general handle 3d graphics. Since they are specialized in the tasks involved in rendering scenes, they are much faster than the central processing unit. This motivates the use of graphics hardware when calculating how good a candidate matches the target image. Using graphics hardware restricts us to simply use some kind of difference between target and rendered image as the fitness, but on the other hand we expect to be able to perform those calculations very rapidly. In this experiment we use evolutionary algorithms [2] to search for 3d-models corresponding to given target images.

II. METHOD

The learning method is a conventional steady-state linear GP algorithm. The method of steady-state tournament selection is used to select individuals to breed. This implies that there are no well-defined generations but a successive change of the population [2] and [3]. Four different individuals are randomly picked from the population and get to compete against each other in pairs. Their performance is evaluated using a fitness function and the two winners get to breed. The offspring, produced by recombination and mutation, replace the two loser individuals in the population.

A. Representing the genotype

Its objects and its general state represent each individual. The objects are stored in a graph. There are two types of nodes, which are positions and objects. A position is a relative translation and rotation that affect every object in its sub tree. It also contains the type of connection that it forms. The four available types are 'free object' (no constraints), 'near parent', 'flat on ground' and 'flat on top of parent'. The plane y = 0 is taken to be the ground. The three size scalars, a color and the type of object represent an object. Currently, 'block' is the only type that is available, but other could be added, e.g. cylinders and spheres. Every position has a child node, which is an object, and every object has a parent position. An object can have any number of children, of which all are positions defining the relative position for each sub tree. So, in the graph there are tree structures with positions at the first level, objects on the second and so on. The reason to use a graph was to make shared sub trees possible. If a shape is present more than once in a scene, then once one of the instances is found, it can duplicate by having another object point to the same place. Currently, this feature is not used, because neither initialization nor genetic operators create shared sub trees. In addition, each individual has a list of entry points in the graph. These represent the set of independent tree structures that are the unrelated compound objects of a model. Review the connection type available for the positions, 'free object' and 'flat on ground' can be associated with the top position of an unrelated object, while 'near parent' and 'float on top of parent' are valid for relative positions of dependent objects. The general state of the individual is the color of the ground, the ambient emissive colors in the scene and the direction and color intensity of a directional light (a light source at infinite distance). This diffuse color was included to get shaded objects and shadows. Thinking about how a human interprets an image, it seems that a lot of information of the 3d structure of a scene is mediated by shading and shadows.

B. Rendering the phenotype

The individuals are rendered using the DirectX 8 SDK for Windows and MSVC++. This way, the 3d graphics hardware is easily employed. The high speed of such hardware is the main motivation for doing 3d interpretation of images the way presented in this text. The acquired speed using hardware dated year 2000 is of order 1000 renderings per second, including intermediate calculations. Two functions carry out the rendering, one that prepare the scene by creating a linear list of objects with their absolute transformations and one that invokes the hardware layer. This is separated for two reasons, one is that the hardware renders the scenes asynchronously so some optimization can be reached by reordering the steps in the algorithm and the other is that sometimes an individual is rendered twice in a row, as we will see later. Shadows on the ground are also depicted. This is done manually calculating the projections of the objects on the ground in accordance with the direction of the diffuse light source. This method was chosen rather than the standard automatic technique of making shadows, which involves stencil buffers and shadow volume rendering. With this method, the diffuse light of the scene, which creates errors in the color and would be inappropriate for the present application, affects shadowed surfaces.

C. Fitness

The fitness is a weighted sum of the norm difference between the rendered individual and the target image, and the complexity of the model. The first term could be considered as the actual error of the phenotype, while the second depend on the genotype and induces some kind of parsimony pressure. Currently, the $L_1 - norm$ is used. The idea of the complexity is to promote solutions that are simpler and more realistic. The complexity is in turn a sum of the total volume of the objects in the model and for each connection (position node) a cost is added that depend on the type of connection. The cost is for instance higher for 'free object' since a freely floating object is rarely found in reality. The dependency of the total volume is motivated both by the preference of few object rather than many and by a tendency the system demonstrated without this pressure, namely to find negative solutions of a scene. This means that instead of putting a block where there is one, it puts blocks in the void and leaves a hole at the block. When calculating the mean difference between individual and target, not all pixels are sampled, but only every n:th in the x and y direction. Skipping pixels was introduced because the difference calculation turned out to be time costly in relation to the rendering. If the hardware could do this step too, much time could be saved. Maybe that it is possible somehow with the present hardware, e.g. by using bitblt operations to subtract the two squares and at the same time contract them to a single pixel with antialiasing turned on. You can in any way question this skipping of pixels. Just rendering a smaller image would be even faster. This is true, but the images are already rendered rather small (100-200 pixels wide) and at this level the size of the rendered images doesn't affect the overall speed of the system very much. To make some use of all the bits rendered, the offset of the sampling is chosen randomly for each difference estimation. This results in a slightly stochastic fitness value, but takes all bits into account.

D. Selection

The selection is based on standard tournaments with four contestants. When having more than one target image (stereoscopic vision), one of the targets is randomly chosen for each tournament. Hence the fitness of the contestants and the outcome of the tournament depend on the current point of view. This, together with the pixel skipping at sampling mentioned above, add some randomness to the outcome. This makes the fitness of the best individual fluctuate and may have some implications on the evolution, as the fitness is not an absolute value. When the winners have been identified, they replace the losers and the children are altered using the genetic operators, mutation and crossover. Crossover is applied once with some probability, while mutation is repeated and for each iteration there is some other probability to continue the loop. In other words the number of mutations applied obeys an exponential distribution. This way multi step mutation can occur, though less probable the single mutations. This is feasible, at least theoretically, as it allows competitive, local minimum-escaping offspring to be created.

E. Initial individuals and genetic operators

E.1 Initial individuals

The general state is more or less randomly set. Diffuse light pointing downwards is made more probable. The ambient color is set to a fixed value and is never changed during evolution. This reduces the number of free parameters without limiting the set of possible states, since only the relation between the ambient and the diffuse color is in fact of interest. The color of a pixel is given by the following equation:

$$c_{pix} = [i_{amb} + i_{diff} \cdot \cos\alpha] \cdot c_{obj} \quad (1)$$

Here, i_{amb} and i_{diff} denote the ambient and diffuse intensities, c_{obj} is the color of the object and α is the angle of inclination for the diffuse light onto the surface. You can see that a transformation to any ambient intensity can be achieved with no output effect on the phenotype if you change the diffuse intensity and all the object's colors appropriately at the same time. The intensity of the diffuse light is initialized at random, color with less saturation (more gray) colors being more probably selected. Finally, a small given number of objects are added in the same way as for mutation, described below.

E.2 Mutation

There are currently five types of mutation. When mutation is invoked, all types occur with some probability independently of each other. The first one adds an object. Its color is picked randomly and its position is chosen to be a random point within the viewing frustum (the set of visible points). The rotation is also random as well as the size is also selected randomly, but its expectance being equal to a given portion of the screen, that is far objects are more likely to be large. The object is not related to another so it will be a separate tree and the type of object will be 'free object' or 'flat on ground'. The second changes the position or the object at a random node in the graph. This is done with a bell-shaped distribution. Whenever a size or a position is altered, the objects are made to comply with the type of connection. This for instance restricts the motion of an object being 'flat on top of parent'. The third attaches an independent object to another one. This typically means taking an object standing on the ground and putting it on top of another object. The fourth picks a node randomly. If it is an object, it removes that including the whole sub tree. If it is a position, it detaches the sub tree, making it independent. This is the inverse of attach. The fifth and last mutating operation randomly adjusts the general state of the individual, i.e. the diffuse light intensity and direction.

E.3 Crossover

Crossover is done as normal for tree representations. Two nodes are randomly selected in the to children and the sub trees are swapped under the condition that there is enough space in both individuals for the resulting trees. Otherwise no change is made.

F. ADDITIONS

F.1 Local minimization

The first generic addition to the algorithm is local minimization performed along with the evolution. The idea is to let the evolution take care of the rough and non-analytical search and leave the search for local minima to a quadratic approximation procedure. Or, using terminology of nature, treat the structure of the individual and the rough changes of parameters as the genotype and think of the fine-tuning as part of the dynamics of the phenotype, as the learning during life. When a creature is born, its constitution gives it the limits, between which it, when it grows, finds the best solution. It learns to use its body as good as possible. With this as a motivation, local minimization was introduced. At each step a number of individuals grows on step. For each step an individual grows, a parameter within it is randomly chosen and an approximating quadratic function is estimated. Then the parameter is changed according to the minimum of the quadratic function. Another idea of local minimization is to let the offspring grow before it is exposed to tournament. This way a child which is a success when it comes to structure, but its parameters are not fine-tuned to the new structure, can have time to grow by do this fine-tuning and become rightfully competitive against its predecessors. To include this, each individual has an age that is set to zero when it is initialized or born and that increased when it learns by local minimization. A minimum age for tournament participation is also defined.

F.2 Temporally connected demes

The second generic addition is temporally connected demes. Demes normally in this field refer to a set of populations, between which a slow migration of individuals takes place or crossover operations are done across the borders. The point is to make several paradigms evolve at once. It is difficult to get more than one surviving paradigm, or species, in a single population, since crossover normally is very destructive for unequal individuals. The relation of the demes can be described as geographical. In this project, the problem of dominating paradigms was attacked slightly differently. A set of demes was created, but in addition to geographical connection for migration, temporal one-way connections were introduced. This was supposed to model the different stages of evolution. By making a chain or a tree of populations you can simultaneously have one at the dawn of life and the others at other stages of progress. Several ways to design the migration from an earlier stage to a later was tried. The chosen method is to simply replace the whole population at one position if an earlier stage connected to it has better individuals. The earlier stage is then replaced with a randomly picked forerunner. That way the populations are trickled down in the tree each time a population outruns one of its successors. The top or origin populations that result empty are reinitialized. This was thought to allow several species at one time, as the temporal structure lets you have several populations at each stage. It also addresses another tendency of evolutionary algorithm, namely the need to re-run the program. When a fairly good result is found, this prevents future experimental behavior, since experimental individuals have to compete with specialized ones. By pushing good species downward and thus keeping some demes at earlier stages, we hope to allow more experimental behavior to be maintained throughout the process. And by repeatedly reinitialize some deme; we could escape the risk of getting stuck in a local minimum. The program is constructed to reinitialize the origin demes from time to time even if no outruns occur, to assure exploration of new, independent paths.

F.3 Color sampling

Moving to the application specific additions, we begin by looking at color sampling. The idea is that given a model containing some objects we can actually choose the best color for each object by calculating the mean of the colors in the target image for all pixels where the object is visible in the rendered image. The color of the ground could the same way be taken as the mean of all pixels where no object resides. To accomplish this, the individuals can be rendered in two modes, one with normal colors and one with colors that supply the program with the necessary information. In this mode, the diffuse color of all objects is set to $(1, 0, 0)$ (red, green and blue components) and the ambient is set to $(0, n_1, n_2)$ where n_1 and n_2 correspond to the index of the node where the object is situated. With diffuse intensity $(1, 0, 0)$ and ambient intensity $(0, 1, 1)$ the resulting color will be $(\cos\alpha, n_1, n_2)$. By decoding the index for each pixel you know which object it should affect. Then contribution can be calculated using the color equation along with the value for $\cos\alpha$, the current values for the diffuse and ambient intensities and the color of the corresponding pixel in the target image. The same is done for the ground.

F.4 Directed mutation

Directed mutation rely on the application specific property that you don't just know the error of the individual but also the distribution of the error over the image. Therefore you can favor mutations acting to change the individual where the error is large, and that way hopefully speed up the search. The image is divided into sub squares. For each square the error is calculated and the index of an object that is visible in the square is stored. Also, the accumulated error is stored looking at the squares as a linear list beginning at the top left corner and ending at the bottom down corner. With the accumulated error, random coordinates that correspond to the error distribution can be easily produced. For mutating an individual, this information can be used to make for instance creation of an object where the error is large more likely to occur. The indices associated with each error square are used for mutation that include existing object, for instance when changing the size.

G. Pre-processing real images

When dealing with real images you have to think of how the scene is transformed to a bitmap. There are two main kinds of transformation, the resulting two-dimensional shape -the projection and the resulting color of each pixel -the intensity transformation. The first kind includes the aspect ratio of the bitmap, the type of projection (perspective, flat) and the field of view (the zoom). In this project it is assumed that these parameters are known. The second kind can be troublesome if the value of the colors in the bitmap is not proportional to the intensity in the depicted scene. This restriction is imposed by the way resulting colors are calculated. Of course, when taking a picture the colors are limited to an interval. There is an absolute black and an absolute white level. But this is modeled in the 3d hardware, since it has that same kind of interval itself. If the color isn't proportional to the intensity, but the transformation is known, the inverse transformation could of course be applied to the image to compensate that before starting the evolution. Here, with assume proportional color. If some transformation was unknown it could be parameterized and exposed to the evolution. However, for practical applications it is reasonable to think that those parameters are once and for all calculated by hand.

III. EXPERIMENTS

Ten different runs were performed, each with a unique setup. In all the runs the maximum number of nodes in each individual's graph was 30 and the maximum number of independent objects in was 15. The amount of memory allocated for the node data of each individual was 10000 bytes. The width of the images was 100 pixels and one fourth of the pixels were visited when calculating difference and sampling colors. Here a selected subset of the settings common for all runs is mentioned. In the runs with 'fixed light' marked the direction and intensity of the diffuse light were correctly initialized and the left unchanged. In all runs but one included multiple, temporally connected demes. Six demes were used; one origin connected to two second evolutionary stage demes. Those two were in turn connected to three final stage demes. In the run with local minimization, only the three final stage demes applied aging (or growing). The real image used for target in the last two runs was slightly manipulated. The background was replaced by the floor texture using the clone stamp tool in PhotoShop.

IV. RESULTS

In this section we present images of best individuals produced during evolution in our experiments.

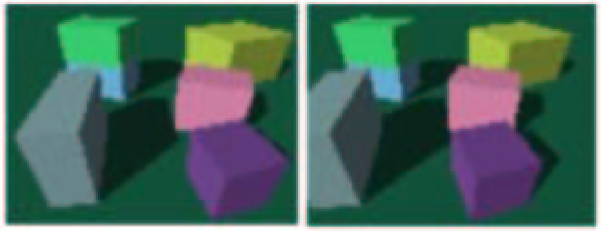

Fig. 1. Target images of run 1 to 8.

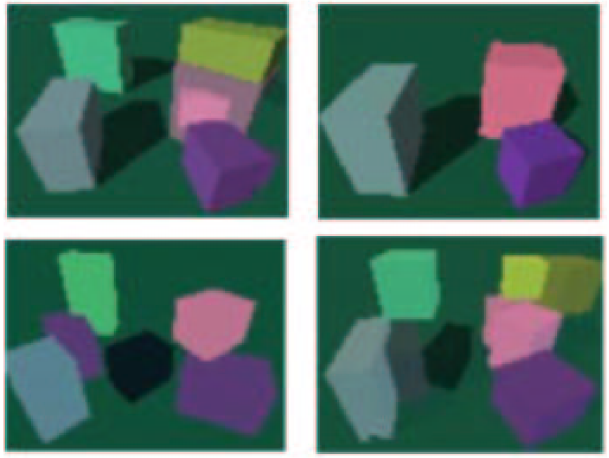

Fig. 2. Images of the best individuals of run 1 through run 4 (from top left corner to bottom right corner).

The system has difficulties finding the right values for the diffuse light. This prevents it from using the information carried by shadows and shading, which was the intention of including diffuse light. Therefore, a couple of runs were made with fixed diffuse light to see if it then seems to use this information. In fig. 2, you can see that both run 1 and 2 has indeed successfully oriented three blocks each.

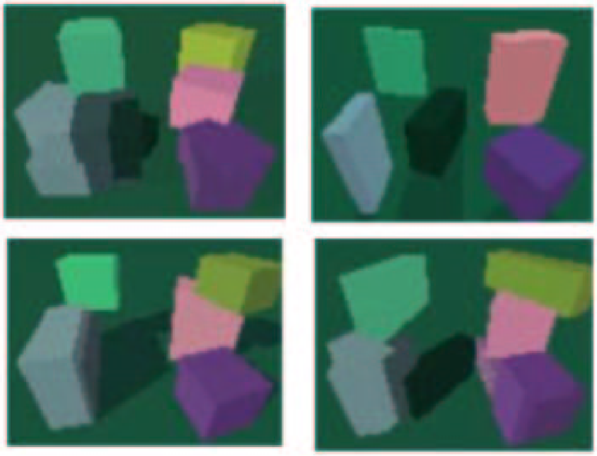

Fig. 3. Images of the best individuals of run 5 through run 8 (from top left corner to bottom right corner).

On the other hand, when the diffuse color is not fixed, the system unlikely finds values even near the correct answer. Only run 7 and 9 has actually the light coming from the right direction. So getting the light right need not lead to victory. The ones that fail have to compensate this in two ways. First, a block in the target image must be represented by at least two blocks, to be able to get both dark and light sides without the aid from the diffuse light. Second, shadows must be mimicked using dark and small blocks (e.g. run 4, 5 and 8). Both these phenomena of course lead to a worse interpretation of the image.

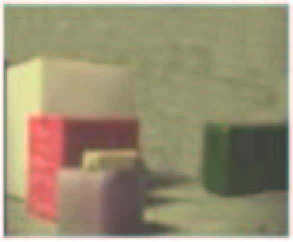

Fig. 4. Target image of run 9 and 10

The fitness value can of course not be compared between runs with and without color sampling, since the sampling always minimizes the error for a given model. What we should compare is the ability to interpret the target image. You could think that sampling colors would contract the search space and make the system better. On the other hand, since color sampling is so powerful and, in a sense, forgiving (or smoothening), you could fear that it actually inhibits the search. The computing time of all runs were all approximately one hour, so we could compare the results to see if any setup seems to give better results.

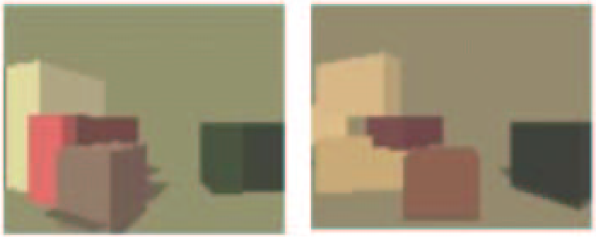

Fig. 5. Images of the best individuals of run 9 (left) and run 10 (right).

In the case of color sampling, looking at the images in fig. 2, 3 and 5, the runs using this feature seem to have reached a slightly better interpretation than the other. As an example, runs 9 and 10 differ only in the color-sampling mode and run 9 looks much better than run 10 even though only half the time was spent.

The graphs in fig. 6 exhibit fluctuations to a degree not usual for evolutionary algorithms. This have to do with the mentioned randomness included when calculated the difference but the difference in fitness depending on which image is in focus when using stereoscopic vision gives perhaps the most important contribution. The fluctuations may impose problems to the algorithm, or they may be beneficial, making the fitness fluctuate as in nature. In the best fitness, you can see phases with smaller and larger fluctuation respectively. This could possibly correspond to individuals being more and less specialized to one point of view.

V. DISCUSSION

The system show ability to interpret images as three-dimensional models. The problem of understanding an image is difficult to formulate, which makes it suitable for evolutionary algorithms. Evolutionary algorithms would be suitable applied to the evolution of interpreting algorithms. That is of course when genetic programming has developed to scope such complex problems. Still, the system in this project may lead to other applications than artificial vision. One application could be image compression using 3d models [4]. An interesting thing with this application is connected to the power of the human vision. When evolving, the system displays individuals being tested on the screen. Also, you can always see the best individual rendered. This way, since humans are so good at quickly interpreting images, you can get a good idea of the process. You can for instance confirm that one kind tends to dominate a whole population, which could be treated as a reason to introduce demes. The failure to find the correct diffuse light illustrates one of the problems of combinatorial search, which evolutionary algorithms have to deal with. The diffuse light, being part of the general state of the individual, affects all parts of the image when altered. The properties of the objects, on the other hand, only locally affect the individual. The diffuse light could be compared with the top node of a tree-based representation in a genetic programming algorithm.

VI. SUMMARY

In this paper a system that uses an evolutionary algorithm to interpret images three-dimensionally was presented. The fitness is related to the difference between the target image and the rendered image of an individual. Hardware for 3d graphics was used to be able to render around 1000 images per second. The system can partially interpret both artificial and simple real images. The concept works and there may be other application, for instance image compression.

References

[1] F. W. DePiero and M. M. Trivedi (1996). ”3-D Computer Vision Using Structured Light: Design, Calibration, and Implementation Issues.” *Advances in Computers* (vol. 43, p. 243-278).

[2] W. Banzhaf, P. Nordin, R. E. Keller and F. D. Francone (1998). ”Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.” San Francisco: Morgan Kaufmann Publishers, Inc. Heidelberg: dpunkt verlag.

[3] P. Nordin (1997). ”Evolutionary Program Induction of Binary Machine Code and its Applications.” Muenster, Germany: Krehl Verlag.

[4] P. Nordin and W. Banzhaf (1996). ”Programmatic Compression of Images and Sound.” In J. R. Koza, D. E. Goldberg, D. B. Fogel and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference* (pp. 345-350). Stanford University, CA, USA: MIT Press.