Algovision

Trying to visualize intuition

or

Discourse on Algovision Goals

Luděk Kučera
Algovision.org
and
Faculty of Mathematics and Physics
Charles University
Prague
and
Faculty of Information Technologies
Czech Technical University
Prague

September 16, 2021

Disclaimer: This text is a very rough presentation of Algovision philosophy. The text is not finished (other cases will be added to the text and other algorithms will be added to Algovision) and not polished from the point of the language and style, definitely not intended as an article for a conference presentation or journal publication. I spent my mental forces to write and finish in a one-man style the first relatively stable version of Algovision that has currently about 105000 lines of code and about 5000 paragraphs and sentences of the guide text, and even if I feel that the design principles of Algovision should be made open to interested colleagues, I am too week now to prepare a text as nice as I would like to have.

1 Introduction

Roughly speaking, Algovision is a collection of animations (or, if you want, visualizations) of algorithms. The field lived its golden times sometimes in 1995-2020 (or perhaps a slightly narrower time range if we want to use the word "golden").

Nowadays only few such systems is surviving, e.g., VisuAlgo of Steven Halim (National University of Singapore) a visualizations of David Galles (University of San Francisco). The goal of Algovision is not a continental balance, but an effort to use visual means to dissemination of ideas that, I believe, are not sufficiently elaborated in other algorithm visualization systems.

The main driving engine of Algovision and the feature that is different from other systems is that I am trying to pass the intuition and view of the algorithm author directly to students by using visual tools. It is difficult to explain the Algovision philosophy as a general principle, and hence the present text is a casuistry that illustrates certain features of particular Algovision algorithm tours and uses them to make the general principles easier to understand.

The cases presented here can be divided into four categories:

- **Intuition visualization** is the basis of the Algovision philosophy. I am trying to discover the algorithm author intuition and *show it* to a student - yes, to show it using proper visual means and tools. The intuition (or the algorithm background idea) is what a reader has in his or her head when, after a long time spent with an article, he or she says "Oh, I see, I *understand* it".

- **Visual proofs**: Constructions presented in correctness and termination proofs can sometimes be made clear if presented not as words, but as images.

- **Visual remainders**: If a teacher teaches certain theme sufficiently many times, he or she also knows a number of ways how to *misunderstand* the topic or to overlook something quite important. Certain successful examples of visual reminders are presented to make facts that were often overlooked or overheard so obvious that it is essentially impossible to miss them.

- **Visual object games** Games with visual objects explaining algorithms are also quite useful.

2 Intuition visualization

Already in 80's some authors were trying to use computer graphics to show temporal features of algorithms. E.g., a 1983 video R. M. Baecker, Sorting out sorting, presented at ACM SIGGRAPH, an 1984 article M. H. Brown and R. Sedgewick, "A system for algorithm animation", Computer Graphics 18(3), 177-186.

A true boom of algorithm animation started in late 90's with Java and similar systems and a mouse, windows, icons available not only on Macintosh, but also in Microsoft systems. However, late 00's saw a decline of algorithm animation, see, for example, a 2007 article R. Ben-Bassat Levy a M. Ben-Ari, "We work so hard and they don't use it: acceptance of software tools by teachers", 12th SIGCSE Conference. "We" were creating animation, "they" were teachers that did not exhibit much interest.

My personal explanation is the following: one late afternoon in February 2005 I had arrived to Pittsburgh exactly the day of Super Bowl - Pittsburgh Steelers played with Seattle Seahawks. The town was full of "Steelers Go" and it was my social obligation to watch the match in my hotel room. I saw fast and strong young men running and falling and using a strange ball, everything was very dynamic. Unfortunately I had absolutely no idea of rules of American football and I was even unable to figure out who won and I had to go down to the lobby to see happy Pittsburghers to find out that Steelers won (21-10).

If a student watches an algorithm animation, he or she sees nice circles, rectangles and letters becoming red, green, big, small and moving in the screen, but only someone who knows rules (i.e., how the algorithm works) sees some system in a fantastic dynamic spectacle, the others just watch an interesting Brownian motion. Not a big help if the animation is intended to make you know the algorithm. No surprise that "they don't use it".

How such an ordinary animation was made: already for years and centuries, data in scientific articles or textbooks are presented using graphics - e.g., matrices, graphs in the sense of Excel (columns, etc.) or graphs in the sense of the graph theory (with nodes as circles and edges as line segments), figures of geometric bodies in 2D and 3D, etc. The standard approach to al-

gorithm visualization is to show simply how a straightforward visualization of the algorithm data evolves during the computation.

But a successful visualization should bring much more. A researcher, trying to find a new computational method, has usually a general idea of possible approach to problem solving in his or her head and needs a lot of imagination, good knowledge of the field; logical analysis is not as important. Like dreams, such ideas can have a form of images, observed by the author's internal sight. Sometimes the idea is generally known, but only one researcher succeeds to use it, in other cases the most important had been to see the problem in a completely different way (and the rest was simple and straightforward).

When the goal is reached, the joy of the scientific work finishes: the result starts to be transformed to the form of an article that is concise, clear, logical, publication ready a free of any trace of the original intuition. And Algovision tries not to present only the final dead result, but also some of the images the author had in the head at the beginning.

It seems that our students like the way Algovision visualizes algorithms, and one of the prominent researchers of the golden age of algorithm animation, now working in a big data visualization, told me about Algovision dynamic visualization of Dijkstra's algorithm (see below) something like "If we had done it in this way, we would not have finished with it".

Algorithms have one quite useful feature that Algorithm uses to fulfill its goals (and which is perhaps a reflection of the original author's idea): we *prove* properties of more complex algorithms, which means proving that it halts and gives the desired result. The main tool of termination and correctness proving are *invariants* - statements that remain valid during the computation and imply what should be proved.

Given an invariant, it is not usually too difficult to find the proof. However, finding an invariant is a big challenge that is not algorithmically solvable in general.

I strongly believe (together with many of my colleagues) that a "knowledge of the invariant" is the same as (or very close to) "understanding the algorithm" and can also be taken as an embodiment of a vague term "author's intuition", and hence in many cases the main goal of Algovision is to illustrate and visualize the invariant(s) of the problem (e.g., see the description of Dijkstra's algorithm visualization in this section).

2.1 Binary search trees - key allocation

A sequence 54 53 52 55 55 59 62 71 72 79 (the net income of Intel in $ billions in years 2011-2020) must be *read* to find out that the income was growing only in the second half of the decade. In the financial sector, it is absolutely natural to present data as a (column) graph, where the trend can be *seen immediately*.

However, most of visualizations of binary search trees that are available in the web present the key allocation by writing numbers to nodes, which is comparable to the numerical presentation of a sequence in the previous paragraph. Even though the key allocation in BST is usually no problem for students, but it is useful for more complex analysis of trees to have absolutely clear intuition on allocation mechanism that can be formulated in two equivalent ways - each one is suitable in certain situations and it is convenient to be aware of both.

The rule that smaller keys are accessible through the left child and larger keys through the right child is directly prepared for searching, but it is also possible to say that inorder traversal of a tree gives a monotone key sequence. Since the standard tree drawing is based on inorder traversal, it is sufficient to put a column below each node having the height proportional to the key value and the rule is clearly *visible* (see [1] with **Show value** checked). Web visualizations of BST are not usually using this feature.

As an example, when deleting the key of a node with two children, only the key is removed and the key of another node is moved to a blind node to lower the complexity of the problem.

Now, where the new key comes from? We say "go to the left child and then, while possible, to the right" (a mirror formulation also possible). However, this is a *secondary* finding. The *primary* finding is that the new key must be a neighbor of the removed key in the column graph of keys, because otherwise the key allocation of keys would be violated. The idea is immediately clear when *and only when* the column graph is shown. Only when this is clear, we can look for a method of finding the replacement key (and demonstrate that its node is easier for deletion).

An expert in a field or an experienced teacher quite often forgets that knowledge that he or she sees as trivial is non-trivial for a beginner and it is important to show a natural and logical path to reach it.

[1] Data structures::Binary search tree::BST condition

2.2 A correspondence between red-black trees and special B-trees (2-3-4-trees)

An animation of a continuous and smooth transformation between different views of a given problem is a very strong tool for presenting their equivalence or relation. An example can be found in "Data structures: Red-Black Tree", a scene "Red-Black and 2-3-4".

Starting from the standard view of red-black trees, black nodes attract their red children in a continuous animation and the equivalent B-tree (2-3-4 tree) is essentially visible; it is just sufficient to use the second animated transformation that changes clusters of black and red circles to a standard view of B-tree nodes.

Historically the transformation went in the opposite direction - side data boxes of B-tree nodes went down to become red nodes.

Even though I had been teaching tree data structures for years, I was surprised to see how natural is the correspondence between red-black trees topics and properties and those of B-trees (e.g., the black depth of a red-black tree and the depth of a B-tree, or sending black node color down in a red-black tree that immediately transforms to B-tree node splitting with moving one data box upwards).

2.3 Standard heap node numbering and a representation by an array

Another example of animated transformation is a binary heap. Its usual drawing is based on an inorder traversal, but to explain an array representation we need breadth-first search indexing of nodes. It is therefore necessary to show both methods and an animation of their mutual transformation that also shows how the BFS view is naturally associated with a linear array.

2.4 Binomic heap

It is known that Jean Vuillemin discovered a binomial heap using an analogy with binary addition. It is therefore exceptionally easy to visualize the author's intuition - the addition process that corresponds to heap operations and is visually simpler can be shown in the background: 1 in the k-th column represents both the value 2^k in the addition and a tree T_k with 2^k nodes in the heap.

2.5 Bitonic sorting

The main problem of the notion of a bitonic sequence is that such a sequence is essentially a sequence that goes first up and then down (bi tonus = two directions), but the definition must be closed to rotations that are quite simple and natural when a cyclic representation of a sequence is used, but complicated when a linear representation is used. On the other hand, a linear sequence representation appears in a straightforward way in a bitonic sorter. It is therefore convenient to work simultaneously with both representations. Even if the correspondence of the drawing modes is easy, Algovision explicitly shows a continuous transformation linear $\rightarrow$ circular.

2.6 Static and dynamic Dijkstra's algorithm visualization

Algovision offers two visualizations of Dijkstra's algorithm[2].

The first visualization is called "Static", because the nodes of the graph do not move during the animation, just change colors and sometimes labels. There are many visualizations of this type, because the algorithm of Dijkstra is often the final highlight of Algorithm and Data Structure courses.

Such a visualization violates one of the basic rules of a good visualization: important information should not (only) be presented in the form of numbers that must be *read*, but using instant visual means.

The most important property of a node of a graph during Dijkstra computation is the estimate of the distance from the origin that is gradually decreasing. The dynamic Dijkstra's algorithm visualization lets nodes "slide" along horizontal lines so that their x-coordinate is linearly independent on the node estimate.

Not only an observe has a good idea about the estimate values, but it is very important that the figure clearly discloses relations that makes the algorithm work properly and that are lost or obscured in the static visualization:

- Closed nodes are left of a certain vertical line, while nodes that are still open are on the right side. It is convenient if the line is given as a boundary between a dark rectangle on the left and a light rectangle on the right. This visualizes one of important invariants of the correctness proof (making the formal proof less needed).

[2]Graph Algorithms :: Extremal Paths

- An open node that is selected in the algorithm loop is clearly visible, while finding this node in the static representation it is very difficult to find it.

- When an edge is relaxed, the edge terminus is attracted to the edge origin, but the attracted node could not finish left of the attractor, because edges are labeled by non-negative numbers.

A yet unpublished experiment at Czech Technical University proved that the dynamic visualization builds understanding much faster than the static one when presented to students without the prior knowledge.

2.7 Algorithm of Dijkstra and Bellman-Ford - how similar and how different

Codes of Dijkstra's algorithm and the algorithm of Bellman and Ford look almost the same, and consequently their usual animations that are visualizing just data changes look also almost the same.

However, if our goal is to explain why they give a correct solution and how much time they need, the visualization will be *very* different, because the behavior of algorithms is very different.

In both cases Algovision tries to visualize important features using changing x-coordinates of node locations that move along horizontal lines during the computation.

Dijkstra's algorithm changes node locations dynamically during the animation and it reflects the value of the distance estimates in nodes.

Bellman-Ford algorithm visualization shows the computation twice: first the original node locations are kept; when the computing is over and the tree of shortest distances is determined, the computation is repeated with node locations given by breadth-first search in this tree.

The aim of this subsection is to illustrate that a good visualization has, first of all, try to make visible the underlying ideas and their influence to the algorithm behavior.

2.8 Dinitz network flow algorithm

A drawback of Ford-Fulkerson algorithm is that some edges get saturated and out of sight, but in the same time some saturated edges get again unsaturated

and enter back into consideration, and therefore the algorithm can loop for a long time before it reaches the optimal result (or even to loop forever in some cases with irrational capacities).

Edmonds and Karp and independently Dinitz improved the algorithm by using always an augmenting path that has the smallest number of edges.

The key to a successful visualization is a *layered network*, introduced by Dinitz that makes it possible to visually and immediately distinguish between "fast" edges that can be used in a shortest augmenting path, and "slow" edges that are not usable in the actual phase of the computation.

And the main point of Dinitz-Edmonds-Karp algorithm that guarantees a good time complexity, is that in any give phase of the computation, "fast" edges get saturated, while edges that return back by loosing their saturation are slow, and hence not usable in the actual phase.

Thus, the key algorithm idea and the reason for a good time complexity can be made obvious and easily visible if - and only if - an appropriate visualization is chosen; in the present case a layered network of Dinitz.

2.9 Conjugate gradient method

Conjugate Gradients Method (CGM) is the basis of Numerical Linear Algebra (its implementation serves now as a secondary benchmark for supercomputer assessment). Computer science students come very close to CGM in the course of Linear Algebra and very small amount of knowledge is sufficient to explain it.

The idea is to transform a positive definite symmetric matrix to the unit matrix (so that the contour lines of the corresponding quadratic form become circles in 2D or spheres in higher dimensions), make one step of the intuitive steepest gradient method and return back. All this can easily be done in the case of a problem in 2D, using the 3D graphics.

Another nice example of a continuous transformation between two different views of the problem.

2.10 Convex hull - complexity

The algorithm itself is not too interesting and it is shown mainly because it illustrates well one successful complexity analysis.

During the computation, a certain triangular area is added to the domain that eventually becomes the convex hull of the original set of sites. It is not

immediately clear how many times the operation is performed.

The idea of the complexity analysis is that the central vertex of the added triangle is detached from the perimeter of the domain, and after two such detachments the vertex becomes an interior point of the domain and it is not used any more in the computation, which proves that at most $2N$ triangles are processed during the computation.

2.11 3D view of Voronoi diagram

Several animations of Voronoi diagram algorithm, discovered by Steve Fortune, can be found in the web. They are all similar to the visualization that appear in Algovision[3]. All of them are examples of animations that could be understood by an ordinary man or woman only with a previous knowledge of the algorithm.

But if 3D graphics is used, it is very simple and natural to show that Voronoi domains are projections of visible parts of cones, and a difficult to understand 2D visualization is sweeping of the cone mountains by an inclines plane[4]. This view is by far not new, but, at my best knowledge, Algovision is the first system to use it in an educational visualization.

The last two scenes of the Voronoi tour show how the way of presenting the algorithm changed over years, and how Fortune saw the cones to be able to sweep by a straight line.

2.12 Knuth-Morris-Pratt pattern matching algorithm

Two ways of drawing a pattern have been adopted, none of them is a static pattern of many pattern matching visualizations. The first one, called a *floating* pattern has been invented as the most straightforward way of the important notion of the "longest pattern prefix that is simultaneously a suffix of a read text".

The second way shows a multitude of pattern copies. The copies make it possible to show in the same time all pattern prefixes that match the read text and illustrate how the Knuth-Morris-Pratt operate with them to find the longest matching pattern prefix in the next step of the computation. Copies are also used to show how the failure function is computed.

[3]Geometric Algorithms::Voronoi Diagram::Animation

[4]Geometric Algorithms::Voronoi Diagram::Cones/Sweep Plane

The pattern matching tour shows how visualizations have to be "made to measure" to principal notions and objects that are used when the ideas behind the algorithm are explained.

2.13 Simplex algorithm

In 2D and 3D, Algovision shows that a collection of linear inequalities defines a polygon or a polyhedron and the goal is to find its extreme point in a given direction. The geometric point of view is well known, but not always sufficiently explained (see the way how LP is presented at many Schools of Management).

However, it is not sufficient to show a geometric view of LP without giving an explicit bridge between geometry and formal manipulations with matrices, vectors, and pivots. Algovision shows polygons and polyhedra always together with their defining inequalities and pays a lot of attention to explaining their relations. Algovision also brings scenes that explain that, when in the dimension N, being in a vertex means that N inequalities have equal sides, and moving to a neighbor vertex is relaxing one equality and, using some analytic geometry, finding the appropriate new N-th equality.

Only when this is sufficiently explained using visualizations, students are able to bring the geometric intuition to formal matrix manipulation.

3 Visual proofs

3.1 Correctness of binary search tree rotations

Keys are allocated in nodes of a binary search tree in a fixed way that is visualized by Algovision so that a column is drawn below every node and its height is proportional to the key value. The allocation condition says that the sequence of key columns is increasing.

Algovision animates rotations so that nodes move strictly *vertically* and therefore the column graph does not change at all. This is a visual proof of the fact that rotation keep the allocation condition valid.

3.2 Correctness of the 3D view of Voronoi diagram

When observed vertically, visible parts of Fortune algorithm cones[5] give obviously Voronoi domains of sites. Unfortunately, it happens too often that an "obvious" statement is wrong, the obviousness being just a result of our lack of imagination.

This is why Algovision offers a "visual proof" of the statement of the previous paragraph. A point on a cone surface forms a triangle with its projection to the site plane and the corresponding cone apex. A point in the valley formed by an intersection of two or more cone surfaces creates two or more *identical* triangle that prove the statement when observed from above.

Even though such visual proofs are not standard formal proofs, it can be viewed as a visual guide through a formal proof or (for better students) a hint how a formal proof can be written, and hence they are a big help for students.

4 Visual reminders

Quite often very important features of algorithms are overlooked by students simply because they are not sufficiently emphasized. Some examples are given how a good visualization reminds a student.

4.1 AVL tree nodes

The difference of the heights of the left and the right subtree of any node of an AVL tree must be equal to -1, 0, or +1. The value is often needed and it should be clearly visible in a visualization. This is why Algovision shows nodes as bars that can be horizontal (balanced) or inclined left or right. A direct inspiration comes from mobiles of Alexander Calder (1898-1976). One of his mobiles (Big Red, 1959) appears at the front page of the textbook T. H. Cormen, Ch. E. Leiserson, R.L. Rivest, C. Stein: Introduction to Algorithms, 2nd ed., MIT Press, 2001 (Calder also made "stabils", but Algovision does not know how to use them).

I have seen numerous animations and figures of AVL trees in the internet. Almost always nodes are painted as circles and hence no visual balance expression is possible and double node body described in the next subsection

[5]Geometric Algorithms::Voronoi Diagram::Cones

is completely impossible. (Some figures in Wikipedia AVL tree page use oval nodes, that, however, are not used to express imbalance and inclination).

4.2 Balance variables of AVL tree nodes

When a new node is inserted to an AVL tree and the AVL condition is fulfilled, a reader could think that the operation is over.

However, efficient execution of operations needs balance variables implemented in all nodes to have an information about node balance immediately, without lengthy computation. After a node addition, some or many node balance variable become outdated and an update is necessary.

Algovision indicates outdated balance variables by painting nodes with dual bodies[6] in the insertion and deletion scenes - the front body represents the actual node balance, the back body is related to the balance variable value. Outdated balance variable is reminded by having the back body partially visible.

4.3 Fast Fourier transform

For years, too many students, during an Algorithm and Data Structure exam, were telling me: the problem is subdivided to *two* subproblems, namely

$$A_\ell = \sum_{k=0}^{N-1} a_k \omega_N^{k\ell} = \sum_{k=0}^{N/2-1} a_{2k} \omega_{N/2}^{k\ell} + \omega_N^\ell \sum_{k=0}^{N/2-1} a_{2k+1} \omega_{N/2}^{k\ell},$$

the subproblems are solved recursively, and the solutions are combined to get the result.

The only minor detail that was overlooked was that the original problem is defined for $\ell = 0, \ldots, N-1$, while the subproblems give the solution for $\ell = 0, \ldots, N/2-1$ only, and they did not realized that they also need values for $\ell = N/2, \ldots, N-1$.

The Algovision visualization[7] clearly shows that the problem is subdivided into *four* subproblems of the dimension $N/2$ (two for $\ell = 0, \ldots, N/2-1$ and two for $\ell = N/2, \ldots, N-1$), see scenes "... Submatrices Compared/Overlayed", and the main contribution of James Cooley and John

[6] Data Structures :: AVL Tree

[7] Graph Algorithms :: Extremal Paths

Tukey, the authors of FFT, is that we have two pairs of *identical* matrices, and only then we get *two* subproblems of the dimension $N/2$.

When I started using the Algovision animation in my course, the percentage of students exhibiting the above misunderstanding dropped essentially to zero, one of my greatest teaching successes.

5 Games with visual objects

5.1 Binary tree rotations

Rotations are operations that are the basis for binary tree balancing, for example in AVL-trees and red-black trees. In a standard course, a definition of a rotation is given (formally and/or by an image) and it is shown that their use in insert and delete operations keep trees well balanced.

In my view, it is important to show the full power of rotations to students. This is why Algovision brings scenes, where a student can play with rotations[8].

One scene shows how to transform a random BST into a BST of the minimal possible depth and also to a BST of the maximal possible depth, while the other one requires that a student transform a random BST to another random BST drawn in the background.

5.2 Making AVL-trees

A scene, which is appreciate by students, brings a random tree; nodes that violate the AVL condition are emphasized. The goal is to change the tree to a correct AVL tree by rotations. Students learn much of the AVL tree theory by themselves, just by playing with different input trees.

5.3 Fibonacci trees

The second scene of the Fibonacci tour heap[9] lets students to build freely different trees that can occur in a Fibonacci heap, and the subsequent scene invites students to construct trees that Algovision draws in the background.

[8]Data Structures::Binary search tree::Rotation in Tree and Rotation Challenge

[9]Data Structures::Fibonacci Heap::Fibonacci Trees

Usual visualizations of Fibonacci heaps do not tell an observer how rich is the variety of such trees that, e.g., includes a path of an arbitrary length, a tree that can not occur in a binary and binomial heap and is the worst possible BST from the point of view of the time complexity, and the Algovision examples are trying to show it (the present collection of examples is going to be extended soon). Only having this knowledge, a student can appreciate certain important features of the Fibonacci heap, e.g., possible inefficiency of decreasing a key by "bubbling up".

6 Appendix: A few words about Algovision history

Algovision is essentially a one-man project. The first Algovision visualization were written as Java applets and appeared at the beginning of this century as a teacher's tools in a two-semester course of Algorithm and Data Structures at Charles University that covers most of what Algovision offers.

Java is an extremely powerful language - also for malware writing, and, gradually, different security measures became a big obstacle in web Java applet use. Taken together with a disease of my wife, a decade long gap appeared in the development of Algovision.

Fortunately, Algovision survived. It moved to Javascript, which is on the other end of the speed scale than Fortran and C++ and on the other end of the scale of suitability for writing large system than Java, but it is omnipresent - if you have a computer, you have a browser, and if you have a browser, you have Javascript. And even modern personal computers are fast enough to run even relatively complex 3D animation (e.g., Voronoi diagram) sufficiently well.

But the main reason why Algovision survived is a different approach to algorithm visualization that has been described above.

Algovision is now developed independently at the platform algovision.org. Its development was substantially influenced by Chinese virus. If had been originally conceived as a tool for face-to-face courses, but a switch to distant teaching for one academic year brought a strong need of a system that guides a student without a human instructor. This is why an integrated guide, based on subtitles and/or synthetic speech, was incorporated to the system. Just recently the system has been finished in the present phase, and even though

it is not yet fully tested, it is already for several years a useful tool for student of computer science.

7 Conclusion

The present text is not final, there are further examples of using the general Algovision principles in particular cases, the present collection is what I have already succeeded to put on the screen.