

# Algorithm Animation as a Narrative

Category: Research

## ABSTRACT

Algorithm animations have been shown to be useful aids for educational purposes. Nevertheless, in a sample survey we conducted it seems that the actual use of animation among students is not very high. Furthermore, numerous algorithm animation systems have been proposed, but none has gained acceptance in a large community. The foremost reason for this is inter-operability: special purpose systems would not be easily adopted by users. We believe the second reason is the fact that most algorithm animation systems are targeted at automating or parameterizing the animation creation, and specializing the systems for algorithm depictions. Our approach suggests using general animation tools which are much widely accessible. Furthermore, we claim that content management and design of the animation have more to do with creating effective educational aids than the tool used to create it. We present a novel framework for the creation of algorithm animations. The framework is based on viewing the animation as a narrative, forming a central story and defining participating characters. Using this framework, the basic elements defining an animation - visual metaphors and temporal transitions - are addressed. The characters which signify abstract notions in the algorithm define the depictions and the story defines the modifications over time. This focuses the creator on the mapping process, which is the basis of any visual depiction: from his or her mind to the animation and back to the viewer's mind.

**CR Categories:** K.5.1 [Information Systems ]: Multimedia Information Systems—Animations; I.3.0 [Computing Methodologies ]: Computer Graphics—General

**Keywords:** Algorithm-animation, Software-visualization, Narrative

## 1 INTRODUCTION

An animation of an algorithm is a visualization of the changes performed by the algorithm on data over time [19]. One can distinguish between two different purposes for algorithm animation and software visualization in general. The first is didactic and the second is analytic. Analysis tools for debugging or performance tuning can be seen as animations if they use visualization to depict the execution of a program or system. In this work we concentrate on the former type of animation which has educational and didactic purposes. There are numerous ways of creating such an animation, but at the base of all of them there are two defining principals: 1. The use of *visual metaphors* and 2. The use of *temporal transitions*.

In the last decade or so numerous works on algorithm animation have shown that the use of animation can promote and enhance the understanding of students who learn algorithms [20, 14]. Nevertheless, it was also found that one of the primary factors that can be accounted for better performance of students is *active learning* [20, 16, 15, 37, 2]. Teaching while giving the students a chance to participate in the learning process significantly promotes understanding with or without the use of animation. This suggests that animation per se may not always be beneficial and careful attention must be given to the design and production process of educational animations [18].

An algorithm animation is a manifestation of the algorithm as seen in the mind of the creator. Hence, the animation can be seen

as a link between two mapping steps: one from the algorithm in the creator's mind to the visual display, and the other from the visual display to the viewer's mind (Figure 1). This mapping process is in fact shared by any depiction, as an intermediate representation transferring notions and ideas from the creator to the viewer [32].

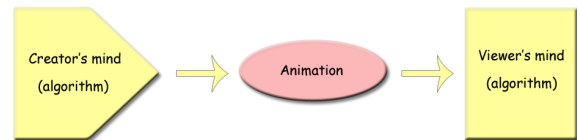


Figure 1: Two mental mappings at the base of any visual depiction are used to carry abstract notions defining an algorithm from creator to viewer.

For these reasons, the creation of an effective algorithm animation can become a complicated task. The creator must choose the underlying visual mapping carefully and must design an animation that engages the user over time to provide a clear learning benefit. In this work we propose a framework for algorithm animation creation which addresses these two challenges: 1. The visual mapping to the metaphors and 2. The temporal engagement of the transitions.

Our framework is based on viewing the animation as a *narrative*, similar to a production process usually used in the context of fiction-animation or movies. It is well known that people comprehend and remember information presented in a context of a story better than as a simple listing of facts [12]. Reason being that a story is more appealing to the viewer. We delimit the two primary components of algorithm animations, the *visual metaphors* and the *temporal transition* in context as a narrative. In our framework, the temporal transition is established by considering the algorithm as a story, and the visual metaphors are established by considering them as characters in the story with an appropriate mapping. We advocate that the creator of an algorithm animation must choose a good plot, find a cast of ‘characters’, define relationships between them etc. This is not to say that there is a need to wrap the animation inside a fiction story. Rather, that the events and happenings which are depicted during the animation should portray a plot leading to the understanding of the algorithm and creating a more engaging presentation.

## 2 THE SURVEY

To measure the level of use, origin and effectiveness of didactic algorithm animations, we have conducted a survey among the computer science students in our school. Over a hundred students voluntarily participated in the survey ranging from freshmen to seniors. Although not a thorough study, we believe it does reflect some of the trends in many computer science schools. We have used a web form where subjective questions were presented regarding the use of algorithm animation and regarding the evaluation of the importance of different attributes for animation.

In Figure 2(a) we can see that the general opinion is that animations are an effective learning aid. Nevertheless, Figure 2(b) shows that the actual use of animation is quite low. The results in Figure 2(c-d) show that most of the algorithm animations used in prac-

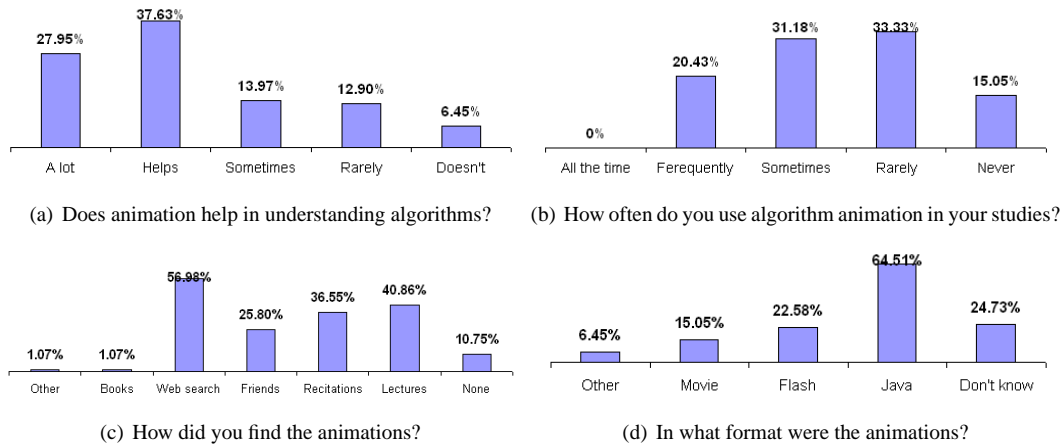


Figure 2: Subjective effectiveness of algorithm animation vs. the level of use, and the source and format of the animations used.

tice are web-based, and the web is by far the most popular place to search for such animations. The second part of the survey which includes the subjective evaluations of different animation parameters will be presented in Section 6.1.

Along with this survey, we have examined previous work on algorithm animation systems. Many attempts were made to design animation authoring systems specifically tailored for algorithm animation. Such systems allow a large degree of automation in the process of animation creation and can relieve the creator from some tedious tasks. Some can also provide powerful insights for analysis purposes and not only didactic presentations. Many other aspects were specifically address (see Section 3), but the fact is that none of these systems gained acceptance for educational purpose over a wide community. Among the many possible reasons for this, two are most significant:

**Inter-operability:** An animation creator would not easily adopt a system that ties the viewer to specific software or hardware, unless there is a clear and significant gain [16]. Numerous special purpose algorithm animation systems that were developed in a specific environment are not easily adapted to other environments. In fact, many of the systems today are moving towards Java to provide inter-operability (such as JSamba, JCAT [23], JAWAA [29], Leonardo Web [10] and [2]). On the other hand, animation technologies which are not specialized for algorithm animation such as Macromedia Flash or even Java Applets are available and are easily accessible today. These provide platform independence, and are already widely accepted. The incentive of developing and using specific production tools for algorithm animation is declining.

**Production:** Most algorithm animation systems emphasize automation and parametrization of the animation instead of focusing on the production process [40, 5]. We argue that it is more difficult to design an effective animation, with proper content management, rather than to create it using one tool or another. On the one hand, this difficulty does not depend on the tool being used, but more on defining an engaging story as a framework for the animation and the definition of effective visual mapping from the depiction to the viewer. On the other hand, general animation tools do provide the correct level of abstraction needed to manage the story, events, time-line and content of the animation.

For these reasons, unlike many previous works, and also to promote our production process approach, we do not advocate the use

of special purpose systems to create algorithm animation, but provide a framework that can be used in general-purpose animation tools.

In the following sections we first present some of the related work on algorithm animation. We then discuss the central idea of narrative creation including the definition of story and characters. Next we portray the two-level mapping of characters and attributes to visual display and give concluding remarks.

### 3 RELATED WORK

The early systems for algorithm animations (until the late 80's) were concerned with creating suitable graphics of algorithms. They dealt with fundamental problems in generating computer graphics, such as smoothness of the animation (Tango [35]), colored animations, basic support in sound (BALSE [8]) and multi-view editing (Zeus [6]). Following, came the systems that supported additional unique features. Mostly, these systems were designed for a special kind of algorithm or system that were built following a new architecture. Among them, we can list the client-server architecture that presented the animation-server idea (Mocha [1]), running animations of parallel algorithms (NESL [4]), geometric algorithms (GASP [38, 34]), animation of data structures (SKA [13], JAWAA [27]), or systems that animated code in specific programming language (BALSE [8], and an early version of Leonardo [11]).

Some systems offered an alternative way to standard coding or programming for the animation formation process. For example, making the system suitable for non-programmers by using graphical editing features in drag-and-drop style (ANIMAL [30], Leonardo Web [10]) or scripting (JAWAA [27], ANIMAL [30]).

More recently, the focus has shifted to accessibility and interactivity, mainly to meet the needs of the electronic classroom and e-learning (CAT [7] and JCAT [23, 24], and [2]). Concentrating on the pedagogical values of the animations by increasing interactivity [14], and increasing their portability by making platform-independent animations [10]. Many different ways were suggested to obtain the desired interactivity. The most common options include encouragement of the students to make predictions about the algorithm's operation [9], allowing entering their own input [36, 25, 18], asking questions regarding the operation of the algorithm [18, 14] and programming their own visualization [36, 20]. No matter what method is used, it is agreed that actively engaging students in the learning process is important, especially while viewing an animation.

The effectiveness of using animation as a teaching aid "had less

beneficial effects than hoped” in a number of studies [18]. Other studies conducted to evaluate the terms in which graphical representations or animations in general were effective also showed inconsistent results [22, 26, 31, 32, 21]. Many argue that the reason is found at the low interactivity level of the animations [25, 36, 20], some argue that animations represent an expert’s understating, not a novice’s [36, 18], and other argue that they use insufficient coding methods to depict the algorithm: animating the execution of an algorithm is simply not enough [18, 36, 39]. We have tried to address these by viewing the animation as a narrative.

Usually, an animation is made by an instructor, to help a novice understand an algorithm [36, 18, 17, 25]. The animation is a graphical representation of the external cognition [32] of the instructor. The viewer is required to understand this specific depiction of a specific execution of an algorithm, and generalize the algorithm simultaneously. Placing the algorithm in the context of a story can help create a more engaging and effective visualization which does not only show what happens during a specific execution, but also enables abstraction and generalization. Similarly, choosing correct ‘characters’ and their display can ease their comprehension as visual metaphors.

Other studies talk about the mental load of the users who watch animations [39, 14]. They claim that minimizing the load on cognitive system will increase visualizations’ effectiveness. They also suggest various techniques to reduce that load, such as keeping the animation as simple as possible, but including its most relevant details at the same time. Most important, they advocate considering the mapping of the algorithmic entities and their depiction in mind [39, 36]. This follows directly from our approach by considering the algorithmic entities as ‘characters’.

#### 4 NARRATIVE CONSTRUCTION

Any animation should tell a story, and didactic algorithm animations are no exception. Simply showing the execution of an algorithm is usually not enough to engage the viewer and promote understanding. It is important to build the animation around a plot that illustrates it, and displays its context. Such a plot will create an effective temporal transition which engages the viewer during the animation. We will demonstrate several of the issues arising in the production of an animation as a narrative using several examples.

##### 4.1 Objective

The first step in the creation of any educational aid is to determine its real objective. In our first example, we show four graph traversal algorithms: breadth first search (BFS), depth first search (DFS), Dijkstra’s algorithm for finding a shortest path from a single source, and Prim’s algorithm for creating a minimum spanning tree of a graph. Although the animation shows several graph algorithms, its goal is not only to provide an understanding of these methods, but to provide an insight that all these different algorithms can in fact be seen as variants of one general traversal method [33]. Hence, our basic story in a single sentence can be summarized as: *there are four algorithms which share a recurring structure that can be generalized (and possibly used to define new algorithms).*

One of the key differences between “old media” stories and new media is the non-linear nature of the storyline. Algorithm animation as a narrative does not have to follow a linear story-line. This can also promote the interactive nature of the animation by driving the user to progress the story. Consequently, in our graph traversal animation we have 4 parallel sub-plots, the four algorithms, and one main story: the generic method. The user has the option to view them in any order. Each algorithm has its own sub-plot illustrating its implementation of graph traversal. However, since our main objective is to show the similarity of these plots, we use the same

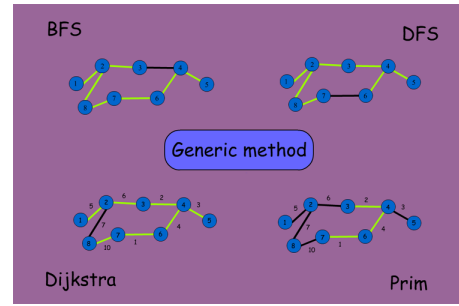


Figure 3: Graph traversal algorithm: the story of four algorithms that are similar and can be generalized. Each algorithm is depicted using the same sub-story of traversing from one node on the same graph.

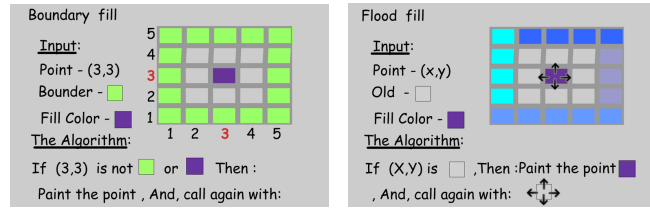


Figure 4: Boundary fill and flood fill algorithms have a similar story of filling a shape recursively pixel by pixel. The difference is depicted mostly by the use of ‘color’ as an entity and character in the story.

graph for all four, describe the algorithms in similar terms, and use similar visual metaphors (Figure 3 and 5).

In another example, we present two computer graphics algorithms to fill an area with a certain color: Boundary Fill and Flood Fill. Both algorithms rely on similar principle of recursion fill but are used in different cases depending on the boundary color. Hence, the basic story that is being told in this case is not a general method but one of the two algorithm stories (boundary fill): starting at a pixel within the shape, move to its neighbors and color them. Continue recursively, until the entire area is colored at the desired color. Again, both algorithms are visually similar. The difference between the two algorithms is emphasized by the question at the end of the first algorithm which distinguishes between situations when one or another should be used (Figure 4).

There are times when the real objective calls for specification rather than generalization. For example, an animation that aims at explaining how to balance an AVL tree after insertion or deletion can skip the insertion or deletion details themselves. It is not the essence of the balancing algorithm, and can distract the viewer (Figure 6). On the other hand, any algorithm should be shown within its context, e.g. by explaining that the balancing act in AVL is a result of insertion or deletion.

##### 4.2 Generalization and Abstraction

Most algorithms need to work on various inputs of a specific type (a number, a list of names etc.), and different inputs may result in different execution patterns for the same algorithm. However, there is no need (and it is practically impossible) to enumerate all possible inputs. Therefore, another reason to build a specific plot is to identify and distinguish between different possible algorithmic patterns, and to find a small set of examples that can represent them.

The story of the animation can be built around one or several of these examples. This will enable the viewer to classify different inputs to the proper algorithmic case, and generalize from a specific input to an execution pattern. For instance, in the AVL-

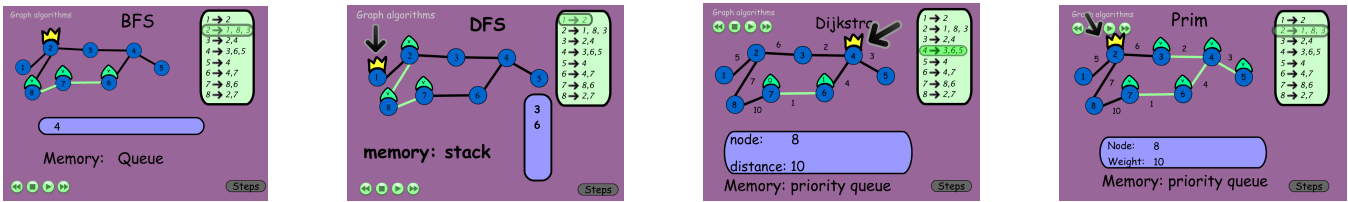


Figure 5: Four variations of the same generic story of graph traversal.

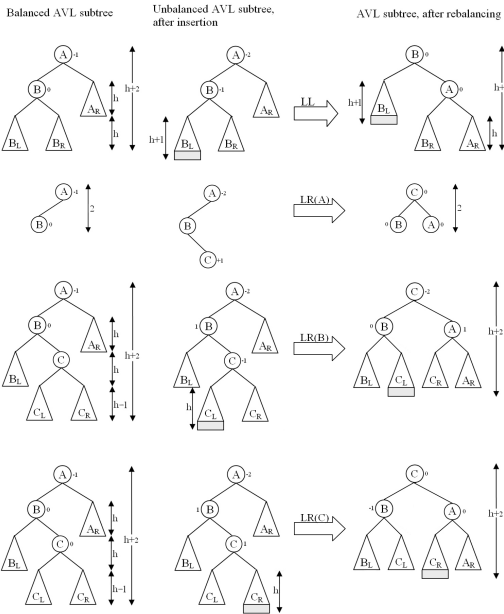


Figure 6: AVL tree balancing: the story concentrates on the balancing act after adding an entry to the tree. By choosing both nodes and sub-trees as characters and using symmetry, generalization can be achieved and only four different cases need to be enumerated.

tree balancing algorithm, there are an infinite number of possible AVL trees. However the balancing can be described in eight general cases by using an abstraction of a node and a sub-tree at once to describe a tree (see the discussion on characters below). Using symmetry, these can be reduced to four (Figure 6). Similarly, for an animation of a specific graph traversal algorithm, generalization can be achieved by using different graphs or different starting nodes. However, in our graph traversal animation we chose one graph and one starting node. We did not seek a generalization of the input, but a generalization of the four different algorithms to one generic method (Figure 5).

### 4.3 Level of Details

Any algorithm is formed from a series of simple and unambiguous instructions, but may also be decomposed into coarser stages. Therefore, the creator must identify these stages in the algorithm and decide at which granularity each of them should be shown. In our example, the general method for traversal is outlined in four simple steps which repeat again in all the different cases: BFS, DFS, Dijkstra and Prim (Figure 7). Moreover, we repeat the same rules verbally in the narration using the same words to describe the way each algorithm works. These are the same simple algorithmic steps we stated at the generic algorithm: “let’s follow the steps: take

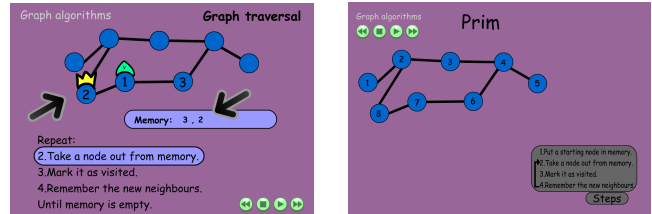


Figure 7: Simple algorithm steps show the stages of the algorithm and repeat in all four different traversals, emphasizing their generality. These steps become a participating character in the animation as they can be brought up to the front interactively by pressing the ‘steps’ button.

a node out from memory, mark it visited, check the new neighbors and remember them”. This emphasizes the similarity between the algorithms both visually and verbally using the level of abstraction for the algorithmic steps shared by all algorithms.

In the graphic shape-filling animation the algorithmic steps are becoming more than pseudo-code depicting the time-line of events. To stress the recursive nature of the algorithm, and to form the visual structure of a recursive call tree these steps are in fact becoming a character in the animation itself.

## 5 CHARACTERS

In an algorithm animation, similar to any story, there are leading and supporting characters. The entities and notions that participate in the animation while depicting the algorithm can be seen as ‘characters’ in the story. Choosing correct characters to depict the algorithm is not always trivial and has a large effect on the success of the whole depiction. For instance, for the AVL balancing animation, the AVL tree is obviously the lead character. Trees are composed of nodes and edges and usually depicted as circles and line segments. However, it would be difficult to generalize the rotation rules if we use specific AVL trees that are composed of nodes. Only by choosing whole sub-trees (visually depicted as triangles) for an additional ‘supporting role’, the schematic view of four principal rotation cases could be recognized easily (Figure 6).

In our graph traversal animation, the lead-role characters are the graphs consisting of nodes and edges. However because our main goal is generalizing the algorithm, a new character comes in – the memory structure. This character takes on different roles in each algorithm and these roles are crucial to distinguish between the different algorithms (Figure 5). Another character, the graph representation as an adjacency list is depicted at the top right of the frame. This character helps to clarify how the abstract structure of a graph can be stored inside the computer, and assists in understanding the algorithm operations such as visiting all neighbors of a node.

As can be seen from these examples, the main story can often be told without supporting roles (e.g. without showing the graph data structure in graph traversal). However, similar to other narratives, these roles often add depth to the story and enhance its understand-

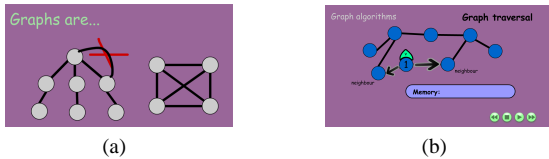


Figure 8: Example of graph definitions: (a) basic notions on graphs are given in a separate section (b) notions needed in the algorithm such as neighbors are described in context.

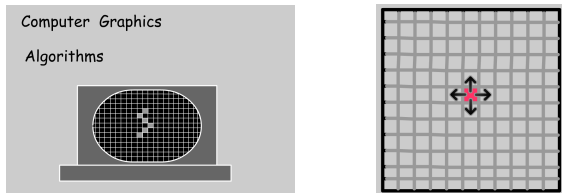


Figure 9: Computer graphics exposition: pixel as the main character and its depiction in the following algorithm animation.

ing. Thinking of these notions as characters provides the level of abstraction for the animation creator to recognize their specific role in the animation. Furthermore it enables composing a story where they interact with each other in a synchronized fashion.

To be able to bridge the gap from depiction to abstract notions in the algorithms, the viewer must first be familiar with the basic notions themselves, but also be able to easily recognize the visual metaphors used to represent them. This may sometimes be as simple as recognizing circles as graph nodes and segments as edges. However, it would be good to provide at least a short exposition or legend describing the notions and their visual depiction. In our graph traversal example there is an initial section that explains basic graph terminology like edges, nodes, connected components or paths (Figure 8a). This has two objectives: first, to present basic notions and second to familiarize the viewer with the visual metaphors being used. Similarly, in the graphics shape-filling algorithms we add an exposition which explains both the notion of a pixel and presents it as the main character in the story (Figure 9).

More complex terms or notions are used along the way in the graph animation. These include neighbors, graph representation, adjacency list and data structures like queue, priority queue or stack. These are included as part of the animation itself as they are more useful in context. For example, we explain what a neighbor is when it is first used (Figure 8b).

There are times that higher level abstractions become characters participating in the story. For example, since the boundary of a shape is defined by the pixels color while filling a shape, the color becomes a character in itself and not only a visual attribute of a pixel. Figure 4 illustrates that colors, as entities, are used in the legend and in the text describing the algorithm. Similarly, the text defining the steps of an algorithm can also become a character. In the graph traversal animation, the generic method steps can pop up at any time interactively, reminding the viewer of the similarity between all algorithms (Figure 7). This idea is taken one step further in the shape filling animation. To emphasize the recursive nature in both algorithms, the whole text outline of the algorithm is converted to a character and coded into a visual entity of a cloud. This cloud is duplicated to compose a tree symbolizing the recursive calls which repeat until the entire shape is colored (Figure 10).

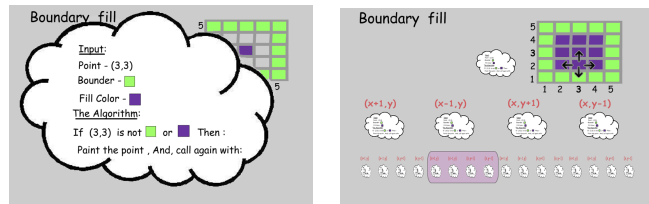


Figure 10: Using the textual algorithmic steps as a character to portray the recursive nature of the algorithm.

## 6 DEPICTION: TWO LEVELS OF MAPPING

Contrary to other forms of narrative, choosing a story and characters does not define the animation completely for algorithm animations. The ‘characters’ in the algorithms are usually abstract notions or entities such as numbers, nodes, or data-structures. One of the most important decisions to be made is how to depict them visually. Choosing a depiction is equivalent to mapping them from the creator’s mind to the visual display. We distinguish between two levels of this mapping: mapping the character to a *visual metaphor* (photograms), and mapping some of the character’s attributes to *visual attributes* of the photogram.

For example, in our graph traversal animation (Figure 5), a node is mapped to a circle, and edges linking neighbors are mapped to lines. The memory structure, be it a queue, a stack or a priority queue, is mapped to a box which contains the names of nodes and relevant attribute (distance or weight) in textual form: a number. In the second level of mapping the name attribute of a node is mapped to text inside its circle, and the weight of an edge to a text label on top of the line segment. The state of an edge is mapped to its color: non-visited edges are black, currently selected edges are red, and visited edges are light green. The state of a node is mapped to colored icons on top of the circles representing nodes: non-visited nodes exhibit no icon, the currently selected node has a yellow crown, and visited nodes have a green hat. These icons also hold other attributes such as the distance from the source node in Dijkstra or the distance to the tree in Prim.

The essence of these mappings lies in their invertibility. The viewer should perform the opposite process of converting the visual display back to the abstract notions in his or her mind (Figure 1). Hence, the mappings should be as simple and natural as possible to create a vessel which carries the abstract notion of the algorithm from the creator’s mind to the viewer’s. In fact, we have often found that ambiguities and inconsistencies in animations can be attributed to incomplete, inconsistent or even wrong use of these two levels of mappings.

In general, the choices to be made pertain the shapes, sizes, colors and textures, position and orientation of the depiction [3]. We call these the *free parameters* of the mapping. In addition, the temporal dimension of an animation adds parameters such as speed, entry and exit queues. As an example, Figure 11 shows snapshots from several students’ projects for visualizing sorting algorithms. All were created to depict the same entity, a number. In the first level of the mapping, the visual metaphor used to represent a number varies from bars to boxes to circles to stars. There is also a great variety in color, positioning and orientation. In the second level of mapping, the only relevant attribute of a number is its cardinality. Still, it is mapped either to length, height, size, text, or color, and sometimes to two of them together: color and height, or text and height.

The selection of the mapping’s free parameters are crucial for designing an effective animation. If these parameters are used too freely or inconsistently, then the interpretation of the mapping might become difficult for the viewer. Using too many visual

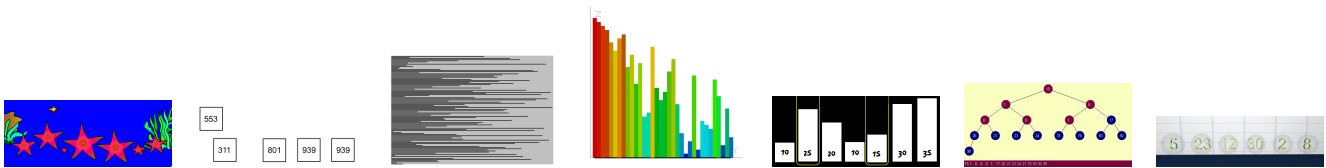


Figure 11: Different depictions of the same characters: numbers for sorting

queues such as too many colors or shapes can be cluttering. The use of visual effects such as 3D effects without a real incentive, may be distracting. This may direct the viewer’s attention to elements which are not important for understanding, or even worse, to elements which are not mapped (do not represent) any abstract notion or attribute in the algorithm. Using different arbitrary colors of nodes or bars may force the viewer to search for a reason or rule, and therefore harm his or her focus. On the other hand, neglecting to display and emphasize crucial parts in the algorithm using visual queues may also damage the viewer’s understanding of the algorithm.

### 6.1 Temporal Transitions

One of the most useful techniques to portray changes over time in an animation is to change the attributes of the visual metaphors. Meaning, usually the modification are applied to the second level of mapping and not to the first. For instance, many times a movement, i.e. change of position or orientation of visual objects define the temporal transition of an animation. However, other attribute modifications can signal change of state as well: the color of a pixel changes once it is visited in the filling algorithm, the icon at the top of the node is changed, or the color of an edge changes when they are visited. Such visual attribute changes signify the change of state of the depicted entity in the algorithm. For this reason, changing or even using visual queues which do not reflect a change of state in the algorithm may cause confusion and hurt the consistency of the animation.

Moreover, if several changes occur together, or if the change of state is depicted using several visual queues, these must be synchronized to produce a coherent animation. For instance, highlighting the line in a pseudo-code, displaying an arrow to the current node and changing the icon on top of its circle should occur simultaneously (Figure 7). If the animation is interactive, the director must also clearly specify the adjustable parts of the animation and synchronize them will.

As part of our survey for computer science students we also asked them to rate the importance of several parameters affecting animations of algorithms. Some of them are related to the mapping and the visual depiction including shape, size, color, sound, and the use of text. And some are more general such as the use of narration, the level of interactivity, and the overall availability of the animation (web-based technologies as opposed to other special purpose systems). Although these are subjective views of students, it seems in general, that the design intuition reflects what the viewers perceive as important: the use of shape and color is an effective means for any form of visualization, and also for conveying abstract notions such as in algorithms. Availability and interactivity are considered key factors for an animation. Nevertheless, sound and narration are rated very low, somewhat contradicting our view, which is discussed next.

## 7 TIEING IT UP: INTERACTION AND NARRATION

Many researches argues that interactivity is one of the most important feature that has pedagogical value in algorithm anima-

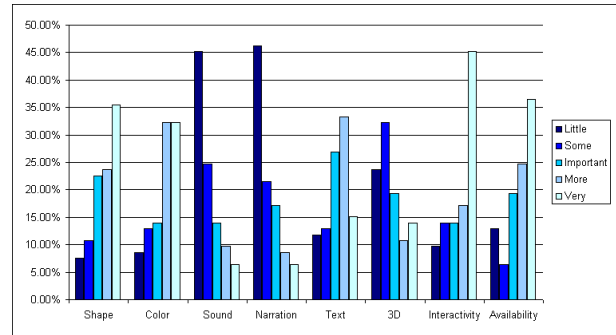


Figure 12: Importance of different animation parameters. Some of the free parameters that were inquired include: shape, size, color, sound, narration, text, dimensions, interactivity and availability.

tion [20, 25]. There have been several suggestions how to provide interactivity in animations of algorithms (see related works section). One of the options is to provide the viewer with the ability to control the pace of the animation, skip parts and return to other parts again. In our examples, this is achieved by providing basic animation controls that are always present at the bottom or top of the screen (see Figure 5). In addition, after each part of the animation is complete, it returns the viewer to the main screen (Figure 3). This allows the viewer to choose between different paths of the story, namely, which algorithm to view at any time.

While interactivity provides the viewer with some freedom and control over the display, narration is strongly linked to tutoring or instructing, which may be perceived as limiting. This may be the reason why students rated its importance as very low for algorithm animations. Still, several researches argue that an animation must be supplemented by some other explanation, either textual or by using voice narration, explaining the operation of the algorithm [18, 36, 39]. According to Paivio’s dual-coding theory, “cognition consists largely of the activity of two partly interconnected but functionally independent and distinct symbolic systems. One encodes verbal events (words) and other encodes nonverbal events (pictures)” [26, 31, 12]. Following this theory, visualizations that encode knowledge in both verbal and non-verbal modes assists the learning process, especially if more than one sensor is being used simultaneously.

Our approach treats the animation as a narrative. A narrative is naturally augmented with a story-teller. The use of voice based narration helps explain the algorithm and may supplement the visual story during temporal transitions. The storytelling technique, which is often used by film directors, can be more enriching than a set of moving images alone, and more compelling than using text. It can also be used to explain the visual metaphors representing the information before the story begins, and alleviate the difficulty in mapping the depiction to abstract notions.

Text can provide additional explanation about the algorithm and emphasizes the mapping from the depiction to the algorithm, by stressing key ideas. In our example, narration accompanies the en-

tire animation, while textual explanations appear only at important parts. For example, the basic steps of the generic method of the graph traversal are very important for generalization. Therefore, in addition to using the same words in the narration, the steps appear as text on the screen in all four traversal algorithms (Figure 7). Similarly, in the shape filling algorithms, the algorithm steps and algorithmic context appear on screen at all times (Figure 4).

It is clear that narration can provide a natural way to help absorb new material. Nevertheless, it should be used with a great care in algorithm animation, without sacrificing the sense of control of the viewer. The user should be allowed to go back and replay some of the explanations, skip some other parts etc. In addition, people tend to ignore unfamiliar or unclear parts of any explanation [28]. Hence, any key-idea that is being presented using narration should be coded in an additional way complementing it in visual form.

## 8 CONCLUSION

In this paper we presented a novel framework for the creation of effective didactic algorithm animations. The framework is based on viewing the animation as a narrative, forming a central story and defining participating characters. This view reflects the two principal elements defining an animation: characters are mapped to visual metaphors in the animation and the story composes the temporal transitions in the animation. The visual metaphors are defined by two levels of mapping: one specifying the photogram representing the character, and the other mapping attributes or states of the character to visual characteristics of the photogram. The animation itself is usually formed by modifying the attributes of the photogram over time, such as position and color.

Two examples that were created using this framework are discussed throughout the paper and attached as supplement material. One is about generalizing graph traversal algorithms to a generic method and the other presents two shape-filling algorithms in graphics.

An overall assessment of this framework is difficult to define using a user study, as there are too many aspects to cover. Nevertheless, there are several interesting parameters that we would like to examine in the future. For instance, the use of narration which we view as important was rated very low in the students questionnaire and should be examined further. Different depictions for a change of state in the algorithm using different parameters such as color or shape or position of the visual photograms should be compared for effectiveness. Other aspects of using general purpose animation tools vs. systems created specifically for algorithm animation should be studied.

In conclusion, we hope that such a narrative centered approach would promote the creation of more effective animations for computer science education, and increase their usability and exposure.

## REFERENCES

- [1] James E. Baker, Isabel F. Cruz, Giuseppe Liotta, and Roberto Tamassia. Algorithm animation over the World Wide Web. In *Proceedings of the workshop on Advanced visual interfaces*, pages 203–212. ACM Special Interest Group on Multimedia, 1996.
- [2] Jeff E. Beall, Adam M. Doppelt, and John F. Hughes. Developing an interactive illustration: Using java and the web to make it worthwhile. In *Proceedings of Computer Graphics Proceedings of 3D and Multimedia on the Internet, WWW and Networks*, April 1996.
- [3] Jacques Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.
- [4] Guy E. Blelloch. NESL: A Nested Data-Parallel Language (Version 3.1). Technical Report CMU-CS-95-170, Carnegie Mellon University, Pittsburgh, PA, 1995.
- [5] M. H. Brown. Perspectives on algorithm animation. In *CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–38, New York, NY, USA, 1988. ACM Press.
- [6] Marc H. Brown. Zeus: A system for algorithm animation and multi-view editing. In *Proceedings of the 1991 IEEE Workshop on Visual Languages*, pages 4–9, October 1991.
- [7] Marc H. Brown and Marc A. Najork. Collaborative active textbook: a web-based algorithm animation system for an electronic classroom. *IEEE Symposium on Visual Languages*, pages 266–275, 1996.
- [8] Marc H. Brown and Robert Sedgewick. A system for algorithm animation. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 177–186, New York, NY, USA, 1984. ACM Press.
- [9] M. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & Education*, 33:253–278, 1999.
- [10] Benedetto A. Colombo, Camil Demetrescu, Irene Finocchi, and Luigi Laura. A system for building animated presentations over the Web. In *the AICCSA'03 Workshop on Practice and Experience with Java Programming in Education*. IEEE, 2003.
- [11] C. Demetrescu and I. Finocchi. A general-purpose logic-based visualization framework. In *Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media*, pages 55–62, 1999.
- [12] Nahum Gershon and Ward Page. What storytelling can do for information visualization. *Communications of the ACM*, 44(8):31–37, 2001.
- [13] Ashley George Hamilton-Taylor and Eileen Kraemer. Ska: supporting algorithm and data structure discussion. In *SIGCSE: ACM Special Interest Group on Computer Science Education*, pages 58–62. ACM Press, New York, 2002.
- [14] Steven Hansen, Hari N. Narayanan, and Mary Hegarty. Designing educationally effective algorithm visualizations. *Journal of Visual Languages and Computing*, 13(3):291–317, 2002.
- [15] C. Hundhausen, S. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, June 2002.
- [16] Christopher D. Hundhausen. Toward effective algorithm visualization artifacts: designing for participation and negotiation in an undergraduate algorithms courses. In *Conference on Human Factors in Computing Systems*, pages 54–55. ACM Press, 1998.
- [17] Christopher D. Hundhausen and Sarah Douglas. Using visualizations to learn algorithms: should student construct their own, or view an expert's? In *IEEE Annual Symposium on Visual Languages, Los Alamitos, California*, pages 21–28. IEEE, 2000.
- [18] Colleen Kehoe, John T. Stasko, and Ashley Tayloe. Rethinking the evaluation of algorithm animations as learning aids: An observational study. *International Journal of Human Computer Studies*, 54(2):265–284, February 2001.
- [19] Andreas Kerren and John T. Stasko. Algorithm animation - introduction. In *Revised Lectures on Software Visualization, International Seminar*, pages 1–15. Lecture Notes In Computer Science 2269, Springer-Verlag, 2001.
- [20] Andrea Lawrence, Albert Badre, and John T. Stasko. Empirically evaluating the use of animations to teach algorithms. In *Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO*, pages 48–54, 1994.
- [21] Doris Lewalter. Cognitive strategies for learning from static and dynamic visuals. *Learning and Instruction*, 13:177–189, 2003.
- [22] Richard E. Mayer and Valerie K. Sims. For whom is a picture worth a thousand words? extensions of a dual-coding theory of multimedia learning. *Journal of Educational Psychology*, 86:389–401, 1994.
- [23] Marc A. Najork. Web-based algorithm animation. In *38th Design Automation Conference*, pages 506–511. ACM Press, 2001.
- [24] Marc A. Najork and Marc H. Brown. Three-dimensional web-based algorithm animations. Technical Report SRC Research Report 170, Compaq Systems Research Center, Palo Alto, 2001.
- [25] Thomas L. Naps, Guido Rossling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Christopher D. Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-

- Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, 2002.
- [26] A. Paivio. *Mental representations: a dual coding approach*. Oxford University Press, 1990.
- [27] Willard C. Pierson and Susan H. Rodger. Web-based animation of data structures using JAWAA. In *the 29th SIGCSE Technical Symposium on Computer Science Education*, pages 267–271. ACM Press, New York, 1998.
- [28] German Robledo. Words without meaning. *MCTM Bulletin*, XXVII(6):1, October 2002.
- [29] Susan H. Rodger. Introducing computer science through animation and virtual worlds. In *33rd SIGCSE Technical Symposium on Computer Science Education*, pages 186–190, 2002.
- [30] Guido Rössling, Markus Schürer, and Bernd Freisleben. The ANIMAL algorithm animation tool. In *The 5th Annual ACM SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, pages 37–40. ACM Press, New York, 2000.
- [31] Purvi Saraiya. Effective features of algorithm visualizations. Master's thesis, Virginia Polytechnic Institute and State University, July 2002.
- [32] Mike Scaife and Yvonne Rogers. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies*, 45(2):185–213, August 1996.
- [33] Robert Sedgewick. *Algorithms*. Addison-Wesley, Reading, MA, 1989.
- [34] Maria Shneerson and Ayellet Tal. Gasp-2: A geometric algorithm animation system for an electronic classroom. In *Gasp-2: A Geometric Algorithm Animation System for an Electronic Classroom*, pages 379–381. Annual Symposium on Computational Geometry, ACM Press, New York, 1997.
- [35] John T. Stasko. Tango: A framework and system for algorithm animation. *IEEE Computer Society Press*, 23(9):27–39, 1990.
- [36] John T. Stasko, Albert Badre, and Clayton Lewis. Do algorithm animations assist learning? an empirical study and analysis. In *Proceedings of ACM INTERCHI 1993 conference of human factors in computing systems*, pages 61–66, 1993.
- [37] John T. Stasko and Christopher D. Hundhausen. Algorithm visualization. In Sally Fincher and Marian Petre, editors, *Computer Science Education Research*, pages 199–228. Routledge Falmer, London, 2004.
- [38] Ayellet Tal and David Dobkin. Visualization of geometric algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):194–204, 1995.
- [39] M. Eduard Tudoreanu. Designing effective program visualization tools for reducing user's cognitive effort. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 105–114. ACM Press, New York, USA, 2003.
- [40] Melissa Wiggins. An overview of program visualization tools and systems. In *the 36th ACM annual Southeast regional conference*, pages 194–200. ACM Press, New York, 1998.