

Electronic Teaching using Shikav

Abhiram Ranade

ranade@cse.iitb.ac.in

Aditya Kelkar

adityakelkar@iitb.ac.in

Prasad Naik

prasadnaik@cse.iitb.ac.in

Department of Computer Science and Engg.,
Indian Institute of Technology Bombay, India.

Abstract: We have developed Shikav, a program for constructing and playing back electronic lessons. It has been used for developing mini-lessons in subjects such as Euclidean Geometry, Numerical Analysis, Optics, Mechanics, Computer Science and Economics (www.cse.iitb.ac.in/~ranade/shikav1.7). These lessons include animations and also support student interaction. The lessons are constructed mostly using a special language, though a rudimentary graphical editor has also been developed.

The space of computer based technologies for learning can be considered to be defined by the following extreme points: presentation technologies such as Microsoft Powerpoint, animation tools such as JAWAA[2], dynamic geometry programs such as Sketchpad[1], Intelligent tutoring systems such as Andes[3]. Shikav is closest in this space to technologies such as Powerpoint, and in addition it also supports many dynamic geometry capabilities.

In this paper, we describe the philosophy behind the Shikav project, and how this philosophy has led to novel features and generally influenced the design.

1 Introduction

What is the best way to study a subject— by reading a book, by browsing a website, by using an educational CD, or by attending a lecture? Books and electronic media can certainly deliver information, with electronic media enabling convenient navigation between topics as well. However, for basic education or for understanding complex subjects, it may be said that a lecture delivered by a good teacher gives a much better experience, even if the class size is large, even for a shy student who seldom asks questions. The same ideas that are in a book when presented by a teacher appear easier to understand. The rough diagrams drawn by the teacher on the blackboard appear easier to grasp than the beautifully drawn pictures in the book. Some of this can be attributed to the emotional pleasure in the interacting with a teacher (and other students), however, is that the full explanation?

One of the main theses behind Shikav is that the classroom model of teaching has many good features and these should be emulated in electronic teaching. Of course, it is very hard to simulate on a computer the personal warmth of the teacher, or the camaraderie between students. Another feature difficult to emulate is the personal attention that a teacher can give

to each individual; intelligent tutoring systems such as Andes[3] attempt to provide this, but it is clearly a daunting task. Our focus is on something more mundane. We feel that classroom style lecturing is a preferred model for learning also because of its *style of delivery*. There are a number of aspects.

Multiple streams of content: While in a book or a website or CD based program there is essentially a single stream of content, in a lecture there are at least two streams: the spoken word and the writing on the blackboard. Good teachers use the two streams quite effectively: the informal, intuitive content is usually spoken out, while the formal content is written out on the board. The formal and the informal get *interleaved* – the teacher might state the theorem and immediately do an example, then return to prove the theorem and then discuss how it applies to the example. This might also get mixed up with stories and anecdotes. What can keep these concurrent threads separate is their mode: the formal is written down on the board while informal is not written down or often written down on a separate part of the board. This is difficult to achieve in a book, and also on an HTML page.

Gradual development of material: Unlike a book where the entire chapter is visible at the outset, in a classroom the information appears gradually. This applies to textual matter as well as pictures. So for example while proving a geometry theorem, initially only just the portion of the diagram needed for the statement is drawn; the “construction” needed to prove the theorem comes in only later. Whereas in a book, or even on websites, only one final picture containing everything gets shown. This is potentially confusing as well as intimidating. While gradual development is supported in the Powerpoint slide model (through overlays) it is not well supported on webpages.

Fine-grained switching: If a teacher demonstrates an experiment or a gadget in class, she does not first describe all the theory and then mutely show the gadget. She discusses the theory along with the gadget. The reason for this is simple: it makes the lesson much easier to understand. This implies that if electronic teaching material includes animations, they must be integrated at a fine grain. In contrast, in most prevailing electronic teaching scenarios, the content/theory is separate from the animation/experiment. In such a scenario the student is first expected to read the theory and then work on the animation; the program delivering the theory cannot usually control the animation program.

Pointing and Highlighting: A good teacher spends a considerable amount of time highlighting previously written material (say by underlining it) or simply pointing to it. For example after writing down the statement of a theorem the teacher might highlight the relevant parts of the associated diagram, say by drawing over it. Pointing and highlighting can be done much nicer on a computer, since the highlights can be removed after the need has passed, whereas it is clumsy to erase the underlining on a blackboard. We feel that electronic teaching should contain very convenient mechanisms for highlighting.

1.1 Main Shikav Features

We developed Shikav so that the above mentioned features of class room teaching can be easily emulated in electronic lessons. Shikav consists of a single program which can be used to create lessons as well as view them. Lessons can be stored on disc using a notation which will be referred to as the Shikav language.

A Shikav lesson is essentially a transcript of the events that are to happen on the screen when a student views the lesson using the Shikav program. Each stream of content is assigned a separate panel on screen. In most lessons there will be at least two panels: one corresponding to what the teacher speaks, and one corresponding to the blackboard. A Shikav lesson just lists out all events (in all panels) in time order.

The second important feature of Shikav is the support for Dynamic Geometry, as pioneered in programs such as Sketchpad[1]. For readers not familiar with this notion: an on-screen object is said to have dynamic geometry if it responds to user commands while maintaining its defining constraints. For example if C is defined to be the midpoint of a line AB , then dragging A or B using a mouse will cause C to move such that it continues to remain the midpoint. More complex constraints may also be allowed, e.g. curve f on screen is the integral of curve g on screen. This allows us, for example, to show how a beam bends as force is applied on it at different points. Dynamic geometry thus allows student interaction as well as mathematically accurate animation. In addition to elementary support for Geometry (lines, arcs, functions, derivatives) the Shikav language also provides some support for graphs.

In addition, we provide support for navigation, glossaries etc. – this is similar to features found in most other programs.

1.2 Overview

We begin in Section 2 by showing how Shikav lessons are created and what to expect during viewing. In Section 3 we discuss advanced Shikav features. In Section 4 we give a tour of some of the Shikav lessons created so far. In Section 5 we review our work and present our plans for the future.

2 Creating and Viewing Lessons

We begin by describing how a teacher might go about creating a Shikav lesson. The easiest way is to first consider how the lesson might progress in the classroom and then express it in the Shikav model. In other words, the teacher first develops a detailed lesson plan (almost as usual), and then translates it into Shikav. As an example, consider a lesson on “Bipartite Graph Matching” which might go as follows:

1. The teacher enters the class and says as introduction “Suppose you have J jobs to be filled and C candidates. Not every candidate can do every job, but you are given for each

Maximum Matching

Java Applet Window

Topics File Next Enter Back

Bipartite Maximum Matching

Input: Bipartite graph $G = (U, V, E)$ where U and V are vertex sets with $|U| = n_1$, $|V| = n_2$, and E the set of edges.

Matching: $M \subseteq E$ is said to be a matching if at most one edge from M is incident on any vertex (in U or in V).

Output: Matching of maximum possible size.

Candidate \equiv vertex in U
 Job \equiv vertex in V
 Candidate u can do job $v \equiv$ presence of edge (u, v)

A matching is an assignment of candidates to jobs (or vice versa).

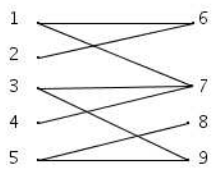


Figure 1: A screenshot of a lesson in progress

candidate the set of jobs that she is qualified for. Can you determine an assignment of candidates to jobs such that maximum number of jobs are filled?"

2. After giving the class a second to digest the preceding sentence, she clarifies, "Assume, that each candidate must be assigned at most one job, and at most one candidate can be assigned to any job. This, basically, is the bipartite maximum matching problem, which we will consider today."
3. She then begins to write on the board the definition of the maximum matching problem in bipartite graphs. This might be something like "Input: Bipartite graph $G = (U, V, E)$ where U and V are vertex sets with $|U| = n_1$, $|V| = n_2$, and E the set of edges."
4. Next, she defines bipartite graphs just in case some students don't know. So she speaks without writing down "Let me remind you what a bipartite graph is: This is a graph in which every edge has one endpoint in U and the other endpoint in V ."
5. Then she draws an example of a bipartite graph on the board.
6. She then says "Let us now define a matching."
7. She then writes down the definition of a matching on the board.
8. Then she highlights a matching from the picture drawn.
9. And so on..

The first task in translating this into Shikav is to identify the communication streams. Clearly, a stream is needed for the teacher's informal remarks which are made throughout the lesson. A stream is also needed for the main definitions and other formal matter. Detailed examples should not be mixed up with formal definition, and thus a stream would be useful for those as well. Finally, a fourth stream is allocated for the pictures drawn by the teacher. Each stream gets a panel on screen as shown in Figure 1: the top left panel is the formal panel (which will include theorem statements, definitions, summarizations), while the top right panel is more informal and will serve for running examples. The bottom left panel is the Speech panel, and this will show what the teacher says. The bottom right panel will be used for pictures.

The panels can be created by putting the following commands in the lesson file:

```
Formal = TextPanel("5","5","500","515","black","white","red")
Speech = TextPanel("5","525","500","160","black","white","red")
Informal = TextPanel ("510","5","475","375","black","white","blue")
Picture = DynamicGeometryPanel("510","385","475","295")
```

The arguments give coordinates of the north-east corner and the width and height respectively, and some information on what default colours to use. The above commands set the stage for the lesson.

Next comes the command to model the teacher saying "Suppose you have J jobs ...". This is done by using the following command:

Speech! Suppose you have J jobs ...

This when executed will cause the Speech panel to be first completely erased (because of the “!” operator) and then the given text to appear in the panel. In a similar manner text can be made to appear in other panels, with or without erasing their content first (by using operators other than “!”). Note that for blackboard panels, it is more appropriate to *append* text, while for speech it is more appropriate to erase the previous text and then put in the new one. This coincides with the observation that typically what you need to remember gets summarized and (semi) permanently recorded on the board, while what is only of incidental importance is spoken and not recorded. All this is controlled by using the appropriate operator (many are provided). Also HTML tags can be embedded into the text for changing font size, colour etc.

A blank line in the lesson file causes the Shikav interpreter to suspend execution and wait for the student to press the “Next” button.

Dynamic geometry objects can be defined using commands such as the following.

```
A = Point(250,100)
B = Point(75,300)
AB = Line(A,B)
C = MidPoint(A,B)
```

This would cause the line AB with midpoint C to appear in the current Geometry panel (named “Picture” as per the panel definitions above). It may appear cumbersome to have to give coordinates explicitly – in Section 3 we show a more convenient mechanism. In addition to lines, Shikav supports many dynamic objects including arcs, curves defined using equations, curves which are derivatives/integrals of curves defined earlier, and so on.

The Shikav language also includes commands for highlighting text in text panels and geometric objects. Highlights can be in any color and can be removed on command.

A Shikav lesson file can be created directly using a text editor, or using the *double view editor* feature of Shikav. We discuss this in Section 3.

2.1 Viewing

The Shikav program when invoked on a lesson file causes the lesson to be shown on the screen. For example, the lesson discussed above would begin by showing 4 panels on the screen. The user can move forward or backward in the lesson by pressing “Next” or “Back” buttons provided. Figure 1 incidentally is what you would get after the lesson described above has advanced a few more steps in the manner described.

3 Advanced Features

We have begun work towards a “what you see is what you get editor” which will ease the task of creating lessons. There are two aspects to this.

Creating objects by freehand drawing Text based input is inconvenient for drawing figures. Rather than define points A, B, C by specifying coordinates as above, it is much nicer if they can be defined graphically. For this we provide a “Draw” mode. In this the teacher can make freehand drawings which get recognized as appropriate objects such as points, lines, arcs, or circles. The corresponding Shikav commands (text) are also generated and added to the lesson file.

Double view editor: Shikav has a facility by which the current lesson file can be viewed and modified as the lesson executes. Also the commands inserted by the freehand drawing are also seen and can be modified if needed. With these two features, objects can be drawn and manipulated without the teacher having to worry about specifying coordinates for the most part.

We have also begun work towards extending the Shikav language to create richer animations. This consists of adding a *Group data structure* and function definitions.

Group Data Structure: This is analogous to lists in programming languages. A group can be formed out of any kind of objects. For example, if we have already defined P,Q,R to be points, then the following will define L to be a group of these.

```
L = [P,Q,R]
```

It is possible to *map* operations on groups, or *reduce* a group into a single objects by combining its constituent objects together, say by adding them up. Here is the code which will make T the centroid of points P, Q, R defined above.

```
cx = (reduce Sum (map X L)) / L.size()
cy = (reduce Sum (map Y L)) / L.size()
T = Point(cx,cy)
```

Note that the code would compute the centroid for an arbitrary collection of points, not necessarily just 3.

Shikav Functions It is possible to define functions which can be subsequently used. For example, centroid computation above can be written as a function:

```
centroid = func(L) {
  Point((reduce Sum (map X L)) / L.size(),
        (reduce Sum (map Y L)) / L.size())
}
```

Subsequently we could call the function as

```
T = centroid([P,Q,R])
```

4 Survey of Lessons created

We have created a library of mini-lessons which illustrate some interesting pedagogical idea and at the same time illustrate the use of a Shikav language feature (the latter often inspired by the former). These lessons can be found at www.cse.iitb.ac.in/~ranade/shikav1.7.

There are several lessons in Euclidean Geometry in which diagrams drawn with dynamic geometry objects allow the student to verify theorems conveniently. For example, in the lesson on the theorem asserting that opposite angles of a cyclic quadrilateral add up to 180 degrees, there is a diagram in which the student can drag the vertex of a cyclic quadrilateral and see how the sum of the angles stays 180. The lesson on Ceva's Theorem similarly allows the theorem to be verified as the triangle and the point inside the triangle (about which the theorem makes an interesting statement) is dragged around.

Ray optics is essentially Euclidean Geometry. Using dynamic geometry it is easy to specify conditions such as "Ray PQ must remain parallel to the principal axis", or "Ray PO must pass through the optic center" and so on. As a result an animation in which the student can drag an object whereupon its image as formed by the lens also moves, is easily constructed.

Reasonably simple dynamic geometry helps in illustrating an interesting phenomenon in Economics, called the "cobweb phenomenon". This is concerned with how next year's production varies based on a simplistic model of how producers respond to this year's price. This price evolution trajectory can be plotted in the space defined by the quantity of production and the price, given the demand and supply curves. The key benefit of dynamic geometry is to see how the trajectory shifts as the student moves the demand curve by dragging its endpoint. In fact, this general idea can be used to demonstrate the evolution trajectory of many systems and its dependence on some of the defining parameters of the evolution.

One more mini-lesson worth mentioning here deals with the mechanics of beams. How does a beam deflect under a point load? It turns out that the underlying mathematics is relatively simple – the curved shape that the beam assumes is the integral of the integral of the so called "bending moment diagram" which is itself a piece-wise linear curve. The bending moment diagram depends, obviously on the position of the point load and also the magnitude. It is possible to show the beam, the point load, the bending moment diagram, and the integrals (the final one giving the deflected shape) all in a single picture. Further, the student can move around the force and also change its magnitude using the mouse; and the effect of this on all the curves immediately shows up on the screen.

Shikav can be extended by writing Java code. The most common type of extension is to add a new panel type (by subclassing an appropriate panel already defined). The lesson on transforming Precedence Constrained Scheduling (PCS) problem to 0-1 Integer Linear Programming (ILP) is an example of this. It contains a specially programmed panel which takes a graph constructed by the user, and transforms it to linear inequalities representing the associated Precedence Constrained Scheduling problem. The Java programmer has full access to Shikav data structures – more so than available through the script language. This lesson also displays a feature by which a student can be asked to give a graph as input – by clicking in vertices and edges.

In all the lessons above, we feel that the use of multiple panels to clarify the different streams of content is very convenient and useful. Most of the above lessons use only 3 panels, the lesson on PCS to ILP transformation uses a fourth panel to show a running example in parallel with a proof.

5 Concluding Remarks

Shikav can be thought of as being defined mainly by the following 3 features: multiple streams of content, mathematically accurate animation through dynamic geometry, and seamless integration of the animations into the lesson (with fine grain control). Of these, we feel that the notion of multiple content streams is very novel. The use of dynamic geometry for constructing mathematically accurate animations is of course known. However, we believe that heretofore these animations were mostly used stand alone and not integrated closely into the content of a lesson. We feel that such integration helps understanding, but this claim needs to be evaluated through student trials, of course.

The electronic lessons available over the internet or on CD-ROMs use a far simpler delivery model than Shikav, in our opinion. In all lessons we have seen, there is essentially one stream of content (occasionally there is audio, which could be thought of as another stream)¹. The content is almost always delivered as a full page image. In some lessons there is a side panel giving “table of contents” – but this hardly qualifies as a “content stream”. Features such as “highlighting” were missing in many lessons which relied on HTML like technologies, though lessons using Powerpoint like technologies did contain highlighting. Many lessons included animations, however, these were not controlled by the lesson player program but controlled only by the student. The Shikav model which allows the lesson to directly control the animation (in addition to the student) can facilitate better “hand-holding” which can be helpful for weaker students.

We expect to evaluate Shikav through trials with students in the next few years. Shikav is currently being used to construct a course in Algorithm Design and Analysis. This will be available to engineering college students within India over the internet and through CDs (a partial set of lectures can be found at www.cse.iitb.ac.in/~ranade/nptel). This experience will be helpful to us in evaluating and improving Shikav.

Acknowledgements: The work on freehand input was done by Praveen Shinde. It is a pleasure to also acknowledge the work of Sanjay Sarode who wrote numerous utilities including a utility to render the Devanagari by transliterating from the Roman script. We would also like to thank Vivek Pandit who did the initial work on the parser. We would also like to thank many others whose contributions helped in the design and implementation, and also several who suggested interesting lessons to develop.

¹In Shikav we did consider playing out the content of the “Speech” stream over an audio output. One drawback with audio is that it is awkward to rewind it; also if put in a panel visually the text is visible to the student until he/she presses the “Next” button. But we may indeed add an audio stream in the future.

References

- [1] Key Curriculum Press. The geometer's sketchpad. <http://www.keypress.com/sketchpad/>.
- [2] S. Rodger. Using hands-on visualizations to teach computer science from beginning courses to advanced courses. In *Second Program Visualization Workshop, Hornstrup Centert, Denmark*, pages 103–112, June 2002.
- [3] K. VanLehn, C. Lynch, K. Schulze, J. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill. The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15:147–204, 2005.