



An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Science and Engineering
Indian Institute of Technology Bombay
Lecture No 26
A Graphical Editor and Solver for Circuits
Part - 4
Extensions and Concluding Remarks

(Refer Slide Time: 0:22)

What we discussed

- How circuits are represented as a system of equations
- Independent equations can be obtained by setting one unknown to 0.
- How the system can be solved

Next: Extensions and Concluding remarks




Welcome back, in the previous segment, we discussed the math representation of the circuits that were designed and we also discussed how we solve those circuits.

(Refer Slide Time: 0:42)

Extensions

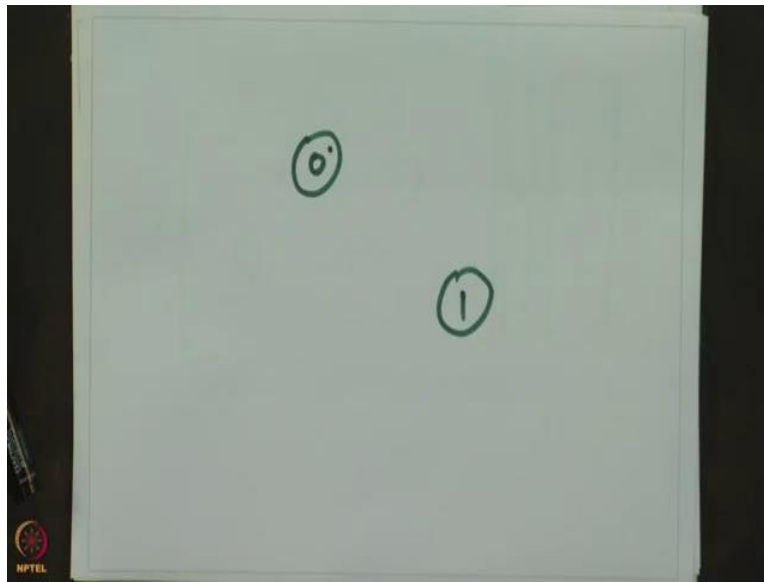
Improve the appearance:

- Put standard symbol of resistance and current source
- Show calculated voltages on canvas.
- Allow user to drag components to adjust visual appearance; see Chapter 20.
- Allow component values to be typed on screen; see Chapter 20.
- Change colour of Button/node once it is clicked, until the command completes.



Now we are going to talk about what kinds of extensions might be possible. So there are some obvious things that could be done and we could improve the appearance, so we should really be putting standard symbols of resistance and current sources. And maybe we should not really be using the shell window at all, everything should happen on the canvas. So the final value is should be shown on the canvas and in fact the user should be asked type directly to the canvas as well. So and if the user places a node but then later on does not like the position, may be it becomes too cluttered or something like that, then the user should allow to drag the components to adjust this visual appearance. So this can be done and it can and it is the primitives you need for it, I will discuss in chapter 20 in the book.


(Refer Slide Time: 1:43)



Extensions

Improve the appearance:

- Put standard symbol of resistance and current source
- Show calculated voltages on canvas.
- Allow user to drag components to adjust visual appearance; see Chapter 20.
- Allow component values to be typed on screen; see Chapter 20.
- Change colour of Button/node once it is clicked, until the command completes.



And here is something that you might have seen happens with nice CAD programs, so for example, if I have say vertex node 0 and node 1 here. And suppose I click on node 0. Then a good program will change the color of this node to indicate that, oh! you just clicked on it. So that way visually there is a queue given to you which, which tells you where you are and that, that makes it easy, easier to use the program. That is not hard, it is the matter of detail and you can do it even now and I certainly suggest that this is one of the extensions you should try out. So basically if I, if I am selecting this node in order to add a conductance, I am first going to click on it, then I am going to click on this and the color should change but the color should go

back to the original colors, once this entire operation has been, has been finished. So this is, this is a good exercise for you.

(Refer Slide Time: 2:48)

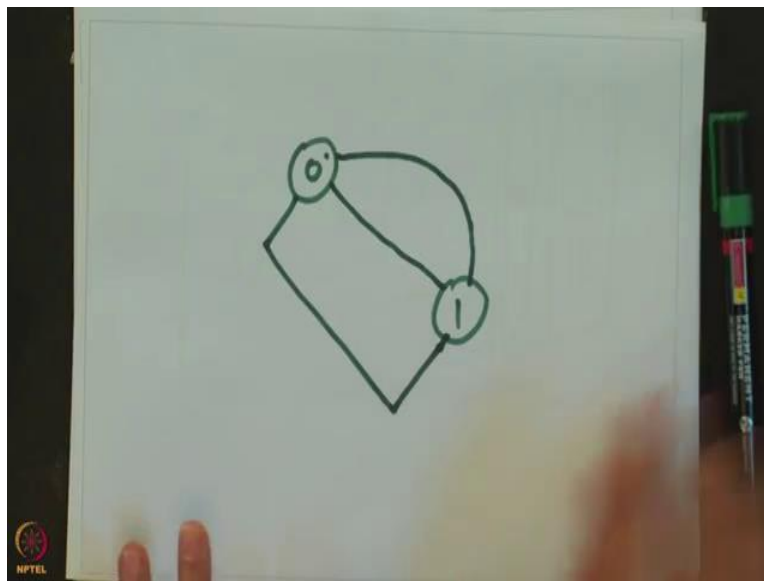

Extensions

Improve the appearance:

- Put standard symbol of resistance and current source
- Show calculated voltages on canvas.
- Allow user to drag components to adjust visual appearance; see Chapter 20.
- Allow component values to be typed on screen; see Chapter 20.
- Change colour of Button/node once it is clicked, until the command completes.

Other functionality:

- If several components are connected across same pair of nodes, draw them far apart.



Other functionality could also be added. So If several components are connected across same pair of nodes, then we need draw them far apart. Right now we are drawing them on straight line but may be if a second component gets put, we need put it in to curve or maybe we need put on line like that or maybe we need let the user decide what this ought to be. But whatever it is, that is something that needs some work and it needs some worrying about geometry thing like that.

(Refer Slide Time: 3:23)


Extensions

Improve the appearance:

- Put standard symbol of resistance and current source
- Show calculated voltages on canvas.
- Allow user to drag components to adjust visual appearance; see Chapter 20.
- Allow component values to be typed on screen; see Chapter 20.
- Change colour of Button/node once it is clicked, until the command completes.

Other functionality:

- If several components are connected across same pair of nodes, draw them far apart.
- Do not modify A, b in MatRep while solving, use temporary copy.
- Allow other components, e.g. capacitors. -- show waveforms.
- Tolerate errors in typing and clicking.




Then right now our code is modify A, b in MatRep directly while we are solving. So probably the right thing to do is make a copy it and then modify the copy of it and keep the values of A and B consistent with what we said when we defined A and B. In particular, $A[i][i]$ should equal the some of the conductance leaving i and we should maintain A and B to be that. So and we could allow some other components for examples capacitors, but in capacitors the solution is time dependent so a capacitor gets charged and as result the value is different at different times. So what we might have to do is we might have to show waveforms, so you could say for solve and then may be you could click on that component and that should pop up a window in which you can see the voltage waveform or may be not I do not really mean the window but just a rectangle should be there on the side, inside which that voltage waveform should be shown. And, a very important part of professional programs is that the gracefully tolerate errors.

So if I am expecting to click on two different nodes in order to insert a conductance, our program as it stands probably do something wrong, if I just happen to click on the same node twice. So what should happen is, somehow the program should indicate to me that oh you had made a mistake and please correct it. So similarly for typing also, if I say type non numerical value and numerical value is expected, the program should check that and tell me.

(Refer Slide Time: 5:44)

Concluding remarks

- Graphical user interfaces are very convenient and not too hard to develop.
- Our code manages to separate concerns fairly well:
 - MatRep: mathematical representation of circuit
 - CanvasContent: management of objects on canvas
 - Main program: overall control
- Hierarchy of objects makes organization clear:
 - Matrep contains matrix and vector
 - CanvasContent contains Nodes – put edge data structures here if they will be needed in extensions.
 - Avoid direct access deep inside the hierarchy from the outside, e.g. do not directly access member functions of sub-objects.



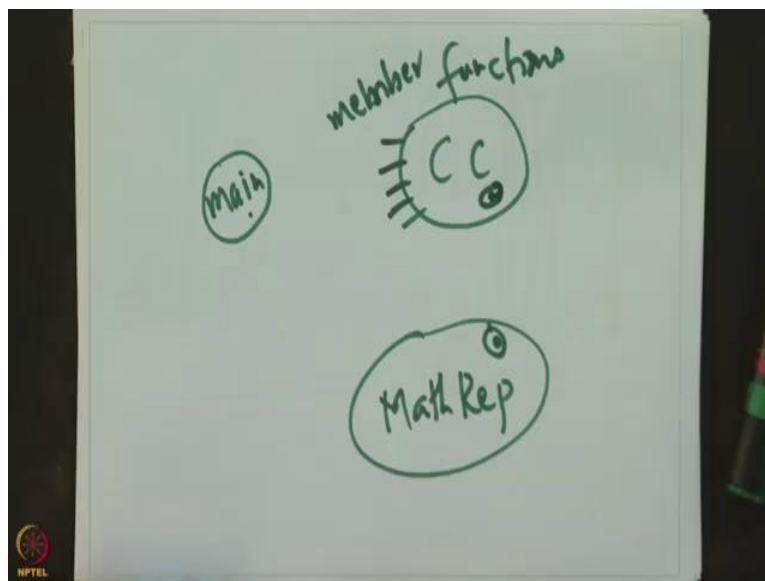
So we have finished the discussion of this graphics editor and solver and what are the conclusions? Graphical user interfaces are very convenient and they are actually not too hard to develop. So our code is interesting and some comments are worth making about it. So our code is separating the various concerns quite well, so what do I mean by this, well class MatRep keeps track mathematical representation, the class CanvasContent manages the visual appearance, what happens on screen and on canvas and the main program has sought of the overall control.

So if I want to think about complex system and you can get systems which much more complex than our editor, I do want this kind of separation of concerns. So I want overall control to be in one place. The canvas management to be in another place and the math representation to be in another place because if everything cluttered together, reader will get confused. So in fact in our program that seems have been managed quite nicely. The way it has been managed is because we have hierarchy of objects. So the MatRep should contain the matrix and the vector. Matrix and vector should not be the part of main program, so the, so those are low level details and the main program is meant to explain or is to meant to talk about how the overall control, how overall control flows inside our program and so this would be, the matrix and vector would be a fairly low level detail which you just confuse the issues if you put that matrix and the vector inside our main program. Canvas content contains Nodes and if you want, you can also put in the edge data structure. Right now we do not have any Edge data structures. Because visually we create an edge and it is imprinted on the screen subsequently there is no question of modifying it.

But as we said in extension you may need to, you may want to modify the edge value so say and might say look I entered a 5 for this conductance and but no no I really want change it 6, so in which case, we will need some visual representation of our edge as well and that, for example, should go into the canvas content, just as we had node array there, you should have an edge array over there. And one more principal is you have a hierarchy of objects and from one member of hierarchy you really should not reach out into the sub objects in another hierarchy because then that becomes confusing.

So main program knows about CanvasContent and MathRep, so main program should invoke member functions in CanvasContent and MathRep. The main program should not should not, directly excess the members in the data members in canvas content and further more say for example, the CanvasContent had this member called node PTR and that contained elements of the form node and it would be wrong for, for the main program to directly access the members of this node PTR. So what should happen is, that there should be a member function at the level of CanvasContent, which math, which which our main program is allowed to excess but not the details inside canvas content.

(Refer Slide Time: 10:06)



Concluding remarks

- Graphical user interfaces are very convenient and not too hard to develop.
- Our code manages to separate concerns fairly well:
 - MatRep: mathematical representation of circuit
 - CanvasContent: management of objects on canvas
 - Main program: overall control
- Hierarchy of objects makes organization clear:
 - Matrep contains matrix and vector
 - CanvasContent contains Nodes – put edge data structures here if they will be needed in extensions.
 - Avoid direct access deep inside the hierarchy from the outside, e.g. do not directly access member functions of sub-objects.
- Do try at least one extension!



So again, I should perhaps explain this pictorially. So you have main program, you have CanvasContent, you have MathRep. And inside it, there is something, inside it there is something and inside that there is something. So main has specific set of member functions that, that the designer of CC should expose so main should call this member function. It should not reach out and touch things inside CC and certainly not touch things which are deep inside CC. Because then that, then what your program is trying to do becomes very very unclear so keep things simple. Keep things at this level which is, that there are member functions, this program is allowing other to access and you should really only access things through those member functions. And I will strongly urge you to try out to extending this program in at least one way. So please do that. So for example, I said that if I click on a node, its color should change to indicate to me that I click on that. So that is relatively easy extension and I would definitely suggest that you try it out and it will be fun and I think you will see that it makes the program easier to use. So that is end of this lecture. Thank you.