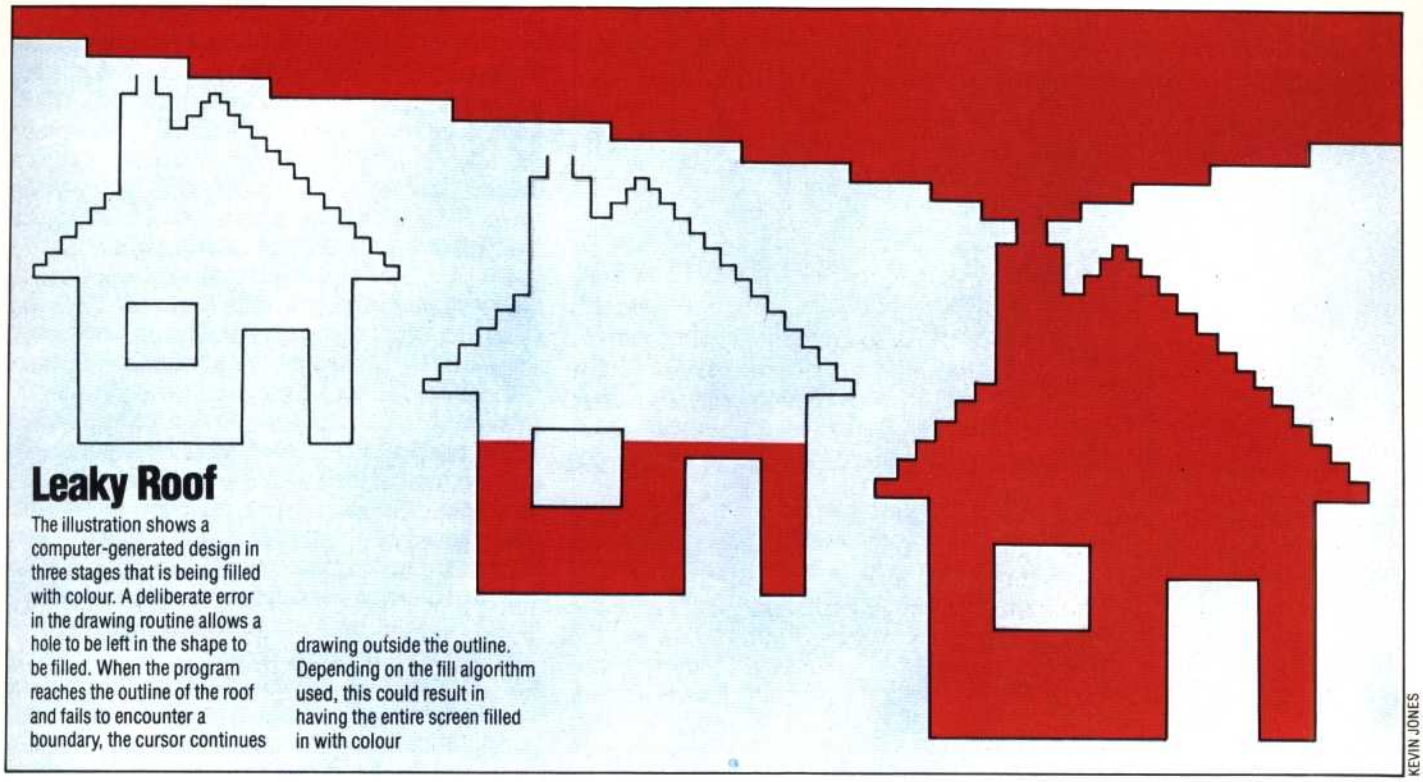

Leaky Roof

The illustration shows a computer-generated design in three stages that is being filled with colour. A deliberate error in the drawing routine allows a hole to be left in the shape to be filled. When the program reaches the outline of the roof and fails to encounter a boundary, the cursor continues drawing outside the outline. Depending on the fill algorithm used, this could result in having the entire screen filled in with colour

KEVIN JONES

a program to run normally, but invalidates the results. As an example, we have chosen to look at a fill pattern that draws a shape on the screen, then fills it with colour. Fill routines look for the boundaries of the shape. When a boundary is reached, the computer turns the cursor around and continues drawing until it reaches another boundary. For a fill routine to work, the boundaries must be well-defined and complete. In other words, there cannot be an open space in the shape's outline or the fill routine will spill the colour out beyond the boundaries.

The versions of the BASIC language used by most home micros make error-handling relatively easy, producing clear and concise error messages and allowing a crashed program to be continued after variable values have been altered at the keyboard — a useful facility when a program is being debugged. Most BASIC dialects will allow the use of a command such as ON ERROR GOTO to transfer the flow of control to a special error-handling routine and thus deal with otherwise 'fatal' bugs. This is done by including a program line such as:

```
30 ON ERROR GOTO 20000: REM error-handling
   routines
```

near the start of the program. Any error will then cause the program to act as though the GOTO 20000 command had been encountered. ON ERROR will usually also set two variables; the first of these stores an error number that indicates the type of error that has occurred, and the other simply holds the line number at which the mistake was encountered. The names given to these variables and the resulting error numbers will vary from machine to machine, so the manual must be consulted. Once an error has occurred, program flow is diverted to line 20000, the error is identified from the number held in the relevant variable and the appropriate action is taken.

A well-written program will not have more than one ON ERROR routine. Such a routine will not be able to deal with syntax errors, memory shortages, stack overflows, etc. The best that this facility can offer is an orderly shutdown of the system, ensuring that all files are CLOSEd and that the user knows exactly what has happened.

Some errors, such as a division by zero, which could be handled by such a routine, should in fact be dealt with in a different manner. There are several reasons for this:

- The ON ERROR GOTO command and the subsequent jump back to the main program constitute an extra entry to and exit from a routine. This violates the structured programming principle that routines should have only one entry and one exit point.
- The proper place to protect against a division by zero is in the routine that does the division. It is bad practice to design algorithms that may crash the system. If the extra error-checking involved slows the program to an unacceptable degree, the routine should be redesigned so that this hazard doesn't arise.
- Error-handling routines rapidly become complicated IF...THEN...ELSE chains with multiple exits. They are inevitably restricted by the line numbering of the rest of the program and so must be rewritten whenever any routine using them is redesigned. They are particularly difficult to design, test and debug, and any mistake in such a routine can introduce far-reaching problems by diverting the flow of control in unforeseen ways.