Short Notes

A Note on Computing the Square Root of an Integer

The method of successive subtraction for finding the square root of an integer or fixed-point number is particularly well-suited for implementation on a binary computer, and was in common use on first-generation machines. The method requires shifting but no division, and its execution time is dependent only on the precision required. A brief description of the method, a Pascal implementation, and some performance results are presented.

Introduction

The advent of pocket calculators has virtually abolished the need for mechanical calculation of square roots, with the result that the pencil-and-paper methods which formed the basis of the first computer algorithms may not be widely known to the present generation of computer scientists.

An early description of the subtraction method, one of the most widely used pencil-and-paper techniques, was given by Richards,[1] who commented on its suitability for computer implementation. The method is also referred to by Knuth,[2] Carberry *et al.*[3] and Gosling.[4] A report by the present authors[5] includes a full mathematical derivation. Horn[6] presents an alternative square root algorithm for integers, based on the conventional Newton-Raphson approach.

The subtraction method

The aim is to find the largest integer p say whose square does not exceed a given positive integer y. If b is the radix to be used for computation, y may be expressed in base b^2 as a sequence of digits $x_1x_2 \ldots x_n$, say where $x_1 > 0$ (unless $y = 0$). Suppose p_i is an approximation to the square root of y_i, where $y_i = x_1x_2 \ldots x_i$, i.e.

$$p_i^2 \leq y_i < (p_i + 1)^2$$

and the residue d_i given by

$$d_i = y_i - p_i^2$$

is known. A corresponding approximation p_{i+1} to the square root of y_{i+1} may be written as

$$p_{i+1} = p_ib + r$$

where $0 \leq r < b$, and it is not difficult to show that r is the largest integer such that the quantity

$$d_ib^2 + x_{i+1} - (2p_ibr + r^2)$$

is positive.

If the quantities $2p_ib + 1$, $2p_ib + 3$, etc, are subtracted from $d_ib^2 + x_{i+1}$ until the result is negative, the number of subtractions is $r + 1$, and the final subtrahend is $2p_{i+1} + 1$. Also, adding back the last subtrahend to render the residue positive again yields the value

$$d_{i+1} = y_{i+1} - p_{i+1}^2$$

This provides the basis of an iterative process for generating p in n steps. Suitable initial values are $p_0 = 0$ and $d_0 = 0$.

The computational cost of the method is clearly proportional to the number of digits of output generated, and only shift, add and subtract operations are involved. When $b = 2$ the algorithm takes a particularly simple form, because at most one subtraction is needed, the output digit r being either 0 or 1.

Implementation

In machine code the most convenient method of generating $d_ib^2 + x_{i+1}$ from d_i is usually to use a double-length register with d_i in the top half and the remaining digits $x_{i+1} \ldots x_n$ of y left justified in the bottom half. A double length left shift by 2 places then has desired effect.

For a high-level language representation it is simpler to work from the most significant end downwards, using a mask to extract the bit pairs. The following Pascal function illustrates the approach:

```
function isqurt (operand: integer): integer;
const ONEBIT = 1; TWOBITS = 2;
      TWOBITMASK = 3;
      WORDLENGTH = 32; (* typically *)
type bitaccess = 0 ... WORDLENGTH;
var residue, newresidue, twiceroot: integer;
  position: bitaccess;
  function nextbitpair
   (word: integer; location: bitaccess): integer;
   (* extracts bit pair whose left bit is location
      places from right of word *)
  begin (* next bit pair *)
  nextbitpair := logicaland (rightshift (word,
                      location), TWOBITMASK)
  end (* next bit pair *);
begin (* isqurt *)
position := WORDLENGTH;
residue := 0; twiceroot := 0;
while position > 0 do
  begin (* process next bit pair *)
  position := position - TWOBITS;
  residue := leftshift (residue, TWOBITS)
              + nextbitpair (operand, position);
  twiceroot := leftshift (twiceroot, ONEBIT)
                                         +1;
  newresidue := residue - twiceroot;
  if newresidue >= 0 then
    begin (* next result bit is 1 *)
    residue := newresidue;
    twiceroot := twiceroot + 1
    end (* next result digit is 1 *)
  else (* next result digit is 0 *)
    twiceroot := twiceroot - 1
  end (* process next bit pair *);
isqurt := rightshift (twiceroot, 1)
end (* isqurt *);
```

In Pascal the functions *leftshift*, *rightshift* and *logicaland* must be implemented using integer arithmetic with powers of 2, or provided as external machine code routines. Other languages have infix operators for shifting and masking.

A simple extension of the method permits the fractional part of the square root to be generated, to a precision limited only by the integer word length.

Performance

The running time will be a constant times half the word length, more or less independent of the operand. Provided masking and shifting are done efficiently, the constant factor will be a small multiple of the subtraction time.

The method has been implemented on a PDP 11/45 in the C programming language[7] as part of an integer-arithmetic conic-drawing package.[8,9] An average of 156 μs per square root was measured. This compares with a figure of 197 μs reported by Horn[6] for a PDP 11/70, using a rational-arithmetic version of the Newton–Raphson method. R. Hutchings of ICL has implemented a floating point version of the subtraction method in the microcode of the PERQ personal computer,[10] which offers improved performance compared to the Newton–Raphson version.[11]

Conclusion

The subtraction method for finding the square root of an integer offers an attractive alternative where hardware floating point operations are not available. It is particularly suitable for use on microcomputers, or for implementation in microcode or hardware.

C. J. PROSSER
Rutherford and Appleton Laboratory
Chilton
Didcot
Oxfordshire OX11 0QX
UK

A. C. KILGOUR
Department of Computing Science
University of Glasgow
Glasgow G12 8QQ
UK

References

1. R. K. Richards, *Arithmetic Operations on Digital Computers*. Van Nostrand, London (1958).
2. D. E. Knuth, *Seminumerical Algorithms*. Addison-Wesley, Massachusetts (1969).
3. M. S. Carberry, H. M. Khalil, L. S. Leathrum and L. S. Levy, *Foundations of Computer Science*. Pitman, London (1979).
4. J. B. Gosling, *Design of Arithmetic Units for Digital Computers*. Macmillan Press, London (1980).
5. C. J. Prosser and A. C. Kilgour, A fast integer square root algorithm, *Report CSC-82-R2*, University of Glasgow Computing Science Department (1982).
6. B. K. P. Horn, Rational arithmetic for minicomputers, *Software—Practice and Experience* **8**, 171–176 (1978).

Downloaded from https://academic.oup.com/comjnl/article/26/2/187/310075 by guest on 13 March 2024

© Wiley Heyden Ltd, 1983

7. B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Prentice-Hall, New Jersey (1978).
8. C. J. Prosser, Graphical output methods and their relation to display systems design, *M.Sc. thesis*, University of Glasgow (1981).
9. C. J. Prosser and A. C. Kilgour, Generating conic sections using integer arithmetic, *Report CSC-82-R1*, University of Glasgow Computing Science Department (1982).
10. F. R. A. Hopgood and R. W. Witty, PERQ and advanced raster graphics workstations, *Computer Graphics and Applications* **2** (No. 7), 9–15 (1982).
11. R. Hutchings, Private communication (1982).

Received July 1982

Comment on 'The Explicit Quad Tree as a Structure for Computer Graphics'

Woodwark[1] has proposed an indexing scheme for representing a complete quad tree without using pointers. The resulting data structure may be used to store pictorial information with pixel values stored in leaves. We propose the use of an alternative indexing scheme which is more suitable for machines having virtual memory. In addition, we propose that average intensity values should be stored in higher level nodes. Storage and processing efficiency are considered.

The quad tree or exponential pyramid data structure has been proposed as a mechanism which records various resolution versions (from fine to coarse) of a picture.[2] The structure may be viewed as a balanced 4-ary tree defined as follows: the root of the tree or 'father' node contains information about the entire picture. The father node is partitioned into 4 (sub)quadrants or 'son' nodes which contain information about the subquadrants. By viewing each son node as a father node and repeating the process, previously defined quadrants can be further partitioned into 4 subquadrants up to the level where the 'leaf' nodes or pixels contain the information obtained by 'raster-scanning' devices.

Woodwark[1] has proposed an indexing scheme for representing a complete quad tree without using pointers. We have found that an alternative indexing scheme, described on p. 401 of Knuth,[3] offers several advantages over the scheme proposed by Woodwark.

In describing this alternative indexing scheme, we will assume that the original picture contains $N \times N$ pixels, where N is a power of two. It is easy to show by induction that a complete quad tree contains $(4N^2 - 1)/3$ nodes. The values associated with these nodes may be stored in an array of length $(4N^2 - 1)/3$ as follows. The value associated with the root is stored in the first array element. For other nodes, the children of the node associated with location i are associated with locations $4i - 2$, $4i - 1$, $4i$ and $4i + 1$. The father of the node associated with location i is associated with location $\lfloor (i + 2)/4 \rfloor$. Hence the children of the root are associated with the second to fifth elements of the array, the grandchildren are associated with the next sixteen array elements, and so forth.

As with Woodwark's scheme, each level of the pyramid is stored as a block in consecutive storage locations. However, with the scheme proposed here, the entire quad tree is treated as a single array and indices are of fixed length. Index calculations can be performed easily in a high level language without bit manipulation facilities.

The top levels of the quad tree are stored together, which is useful in the case of a virtual memory machine. In addition, all descendants at a given level of a given node are located together. (For example, all great grandchildren of a given node are located together.) If there is at least one page available for each of the lower levels of a deep tree (recalling that the higher levels can share a single page) then page turns may be considerably reduced when traversing a quad tree. Page turns also are likely to be reduced in local operations, such as following a boundary between two regions.

The overheads of storing and using a quad tree rather than just the bottom level pixels are small when the above indexing scheme is used. Storage requirements are increased by just under $\frac{1}{3}$ (since there are $(4N^2 - 1)/3$ array elements rather than N^2 pixels).

We have found it useful to store in each non-leaf node the average intensity of the four children of the node. In this way, we can obtain different versions of a picture, having different resolutions, by going to different depths in the quad tree. We may optionally use an additional bit to indicate that a node is the root of a constant subtree. (Woodwark used a special value, called *transparent*, for all nodes which were not roots of constant subtrees.)

The proposed data structure has a clear advantage over a pixel array when many picture operations may be performed on low resolution approximations to the original picture. In these cases, only the upper levels of the quad tree need be considered.

In many cases it may be reasonable to attempt to solve a problem by first considering a low resolution approximation and then going on to higher resolution approximation if necessary. The number of nodes in the top k levels of a quad tree is $(4^k - 1)/3$. (If $k = 1 + \log_2 N$, as is the case with the full tree, then $(4^k - 1)/3 = (4N^2 - 1)/3$.) Hence, if we look at the top level, then the top two levels, and so forth until we traverse the entire quad tree, then we must examine $1 + 5 + \cdots + (4N^2 - 1)/3 = 16/9N^2 - 7/9 - (\log_2 N)/3 < 16/9N^2$ nodes. So if we examine the picture at every possible resolution down to and including the resolution at which we are able to solve the problem, and traverse the entire tree down to the particular resolution in each case, then the number of array element accesses is less than $16/9N^2$. Looking at all lower resolution approximations increases our work by only about $\frac{1}{3}$. This is because the vast majority of nodes in a quad tree are leaves. If there is a reasonable chance that we can solve a problem with a low resolution approximation to a picture, it is often worth trying since the cost of failure is small relative to the cost of inspecting the entire picture.

F. WARREN BURTON
School of Computing Studies
University of East Anglia
Norwich NR4 7TJ
UK

J. G. KOLLIAS
Department of Computer Science
National Technical University of Athens
9 Heroon Polytechniou Ave
Genikes Edres
Zografou
Athens (621)
Greece

References

1. J. R. Woodwark, The explicit quad tree as a structure for computer graphics. *The Computer Journal* **25** (2), 235 (1982).
2. A. Klinger, Regular decomposition and picture structure. *Proceedings 1974 IEEE Systems, Man and Cybernetics Conference*, pp. 307–310 (October 1974).
3. D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, Vol. 1. Addison-Wesley, Reading, Massachusetts (1968).

Received September 1982

Downloaded from https://academic.oup.com/comjnl/article/26/2/187/310075 by guest on 13 March 2024