Fixed Point Square Root

Ken Turkowski
Media Technologies: Computer Graphics
Advanced Technology Group
Apple Computer, Inc.

Abstract: The square root of a fixed-point number is computed using a simple method similar to longhand division.

3 October 1994

Apple Technical Report No. 96

Fixed Point Square Root

Ken Turkowski
3 October 1994

Introduction

Many graphics algorithms rely upon fixed-point arithmetic and its inherent speed advantage over floating-point. Often, a fixed-point algorithm requires the evaluation of a square root. This gem describes an algorithm which computes the square root directly in its fixed-point representation, saving the expense of (re)converting and evaluating in floating-point. A related gem [Musial91] computes an approximate integer square root through the use of integer divisions, but the following algorithm uses more elementary operations.

The Algorithm

The algorithm is based upon a fixed point format having two integer and thirty fractional bits, using conventional machine (integer) arithmetic instructions. This choice gives a domain of representation [-2.0, 2.0) suitable for representing normals, colors, and other graphic quantities whose magnitude is bounded by unity.

This algorithm is based upon a method, similar to long-hand decimal division, that was taught in schools before the advent of electronic calculators [Gellert75]. This implementation, called the “binary restoring square root extraction”, substitutes binary digits (bits), further streamlining the algorithm.

A radical r (the square root of the radicand x) is constructed a bit at a time such that $r^2 \le x$ is always preserved by application of the identity

$$(r+1)^2 = r^2 + 2r + 1$$

in which the $(2r+1)$ term is subtracted from the radicand x at each step. If the result is non-negative, a “1” is generated; otherwise a “0” is generated and the radicand is unaltered (i.e. restored).

Two radicand bits of are consumed and one radical bit in the radical is generated in each iteration of the loop. Although this algorithm has only O(n) (linear) convergence, the loop is so simple that it executes quickly, making it amenable to hardware implementation.

C Implementation

```
typedef long TFract;  /* 2 integer bits, 30 fractional bits */

TFract
FFracSqrt(TFract x)
{
    register unsigned long root, remHi, remLo, testDiv, count;

    root = 0;           /* Clear root */
    remHi = 0;          /* Clear high part of partial remainder */
    remLo = x;          /* Get argument into low part of partial remainder */
    count = 30;         /* Load loop counter */
    do {
        remHi = (remHi<<2) | (remLo>>30); remLo <<= 2;  /* get 2 bits of arg */
        root <<= 1; /* Get ready for the next bit in the root */
        testDiv = (root << 1) + 1;   /* Test radical */
        if (remHi >= testDiv) {
            remHi -= testDiv;
            root++;
        }
    } while (count-- != 0);

    return(root);
}
```

Discussion

A non-restoring version of the algorithm [Hwang79] may run faster at the expense of a slightly more complicated inner loop.

This algorithm may be modified to return the square root of a 32-bit *integer* by setting FRACBITS as zero, producing a variant with count=15. Other formats having an even numbers of fractional bits can be accommodated simply by adjusting these values. Note that the square root of a long int (thirty-two bits) is a short int (sixteen bits).

References

Gellert75	W. Gellert, H. Küstner, M. Hellwich, H. Kästner, *The VNR Concise Encyclopedia of Mathematics*, Van Nostrand Reinhold, 1975, pp. 52-53.
Hwang79	Kai Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, Wiley, 1979, pp. 360-379.
Musial91	Christopher Musial, An Integer Square Root Algorithm, In James Arvo, editor, *Graphics Gems II*, Academic Press, 1991, pp. 387-388.