快速平方根算法

本文转自： http://www.cfannet.com/bbs/dispbbs.asp?boardID=11&ID=151&page=2

原文如下：

快速平方根算法

作者：**Blackbird** 文章出处：友善之臂旅店

在3D图形编程中，经常要求平方根或平方根的倒数，例如：求向量的长度或将向量归一化。C数学函数库中的**sqrt**具有理想的精度，但对于3D游戏程式来说速度太慢。我们希望能够在保证足够的精度的同时，进一步提高速度。

Carmack在**QUAKE3**中使用了下面的算法，它第一次在公众场合出现的时候，几乎震住了所有的人。据说该算法其实并不是Carmack发明的，它真正的作者是**Nvidia**的**Gary Tarolli**（未经证实）。

```
//
// 计算参数x的平方根的倒数
//
float InvSqrt (float x)
{
  float xhalf = 0.5f*x;
  int i = *(int*)&x;
i = 0x5f3759df - (i >> 1); // 计算第一个近似根
  x = *(float*)&i;
x = x*(1.5f - xhalf*x*x); // 牛顿迭代法
  return x;
}
```

该算法的本质其实就是**牛顿迭代法**（Newton-Raphson Method，简称NR），而NR的基础则是**泰勒级数**（Taylor Series）。NR是一种求方程的近似根的方法。首先要估计一个与方程的根比较靠近的数值，然后根据公式推算下一个更加近似的数值，不断重复直到可以获得满意的精度。其公式如下：

函数：y=f(x)

其一阶导数为：y'=f'(x)

则方程：f(x)=0 的第n+1个近似根为

```
x[n+1] = x[n] - f(x[n]) / f'(x[n])
```

NR最关键的地方在于估计第一个近似根。如果该近似根与真根足够靠近的话，那么只需要少数几次迭代，就可以得到满意的解。

现在回过头来看看如何利用牛顿法来解决我们的问题。求平方根的倒数，实际就是求方程1/(x^2)-a=0的解。将该方程按牛顿迭代法的公式展开为：

```
x[n+1]=1/2*x[n]*(3-a*x[n]*x[n])
```

将1/2放到括号里面，就得到了上面那个函数的倒数第二行。

接着，我们要设法估计第一个近似根。这也是上面的函数最神奇的地方。它通过某种方法算出了一个与真根非常接近的近似根，因此它只需要使用一次迭代过程就获得了较满意的解。它是怎样做到的呢？所有的奥妙就在于这一行：

```
i = 0x5f3759df - (i >> 1); // 计算第一个近似根
```

超级莫名其妙的语句，不是吗？但仔细想一下的话，还是可以理解的。我们知道，IEEE标准下，**float**类型的数据在32位系统上是这样表示的（大体来说就是这样，但省略了很多细节，有兴趣可以GOOGLE）：

```
bits：31 30 ... 0
31：符号位
```

30-23：共8位，保存指数（E）
22-0：共23位，保存尾数（M）

所以，32位的浮点数用十进制实数表示就是：M*2^E。开根然后倒数就是：M^(-1/2)*2^(-E/2)。现在就十分清晰了。语句i>>1其工作就是将指数除以2，实现2^(E/2)的部分。而前面用一个常数减去它，目的就是得到M^(1/2)同时反转所有指数的符号。

至于那个**0x5f3759df**，呃，我只能说，的确是一个超级的**Magic Number**。

那个Magic Number是可以推导出来的，但我并不打算在这里讨论，因为实在太繁琐了。简单来说，其原理如下：因为IEEE的浮点数中，尾数M省略了最前面的1，所以实际的尾数是1+M。如果你在大学上数学课没有打瞌睡的话，那么当你看到(1+M)^(-1/2)这样的形式时，应该会马上联想的到它的泰勒级数展开，而该展开式的第一项就是常数。下面给出简单的推导过程：

对于实数R>0，假设其在IEEE的浮点表示中，
指数为E，尾数为M，则：

R^(-1/2)
= (1+M)^(-1/2) * 2^(-E/2)

将(1+M)^(-1/2)按泰勒级数展开，取第一项，得：

原式
= (1-M/2) * 2^(-E/2)
= 2^(-E/2) - (M/2) * 2^(-E/2)

如果不考虑指数的符号的话，
(M/2)*2^(E/2)正是(R>>1)，
而在IEEE表示中，指数的符号只需简单地加上一个偏移即可，
而式子的前半部分刚好是个常数，所以原式可以转化为：

原式 = C - (M/2)*2^(E/2) = C - (R>>1)，其中C为常数

所以只需要解方程：
R^(-1/2)
= (1+M)^(-1/2) * 2^(-E/2)
= C - (R>>1)
求出令到相对误差最小的C值就可以了

上面的推导过程只是我个人的理解，并未得到证实。而Chris Lomont则在他的论文中详细讨论了最后那个方程的解法，并尝试在实际的机器上寻找最佳的常数C。有兴趣的朋友可以在文末找到他的论文的链接。

所以，所谓的Magic Number，并不是从N元宇宙的某个星系由于时空扭曲而掉到地球上的，而是几百年前就有的数学理论。只要熟悉NR和泰勒级数，你我同样有能力作出类似的优化。

在GameDev.net上有人做过测试，该函数的相对误差约为0.177585%，速度比C标准库的sqrt提高超过20%。如果增加一次迭代过程，相对误差可以降低到e-004 的级数，但速度也会降到和sqrt差不多。据说在DOOM3中，Carmack通过查找表进一步优化了该算法，精度近乎完美，而且速度也比原版提高了一截（正在努力弄源码，谁有发我一份）。

值得注意的是，在**Chris Lomont**的演算中，理论上最优秀的常数（精度最高）是**0x5f37642f**，并且在实际测试中，如果只使用一次迭代的话，其效果也是最好的。但奇怪的是，经过两次NR后，在该常数下解的精度将降低得非常厉害（天知道是怎么回事！）。经过实际的测试，**Chris Lomont**认为，最优秀的常数是**0x5f375a86**。如果换成64位的**double**版本的话，算法还是一样的，而最优常数则为**0x5fe6ec85e7de30da**（又一个令人冒汗的Magic Number - -b）。

这个算法依赖于浮点数的内部表示和字节顺序，所以是不具移植性的。如果放到Mac上跑就会挂掉。如果想具备可移植性，还是乖乖用sqrt好了。但算法思想是通用的。大家可以尝试推算一下相应的平方根算法。

下面给出Carmack在QUAKE3中使用的平方根算法。Carmack已经将QUAKE3的所有源代码捐给开源了，所以大家可以放心使用，不用担心会收到律师信。

//

```
// Carmack在QUAKE3中使用的计算平方根的函数
//
float CarmSqrt(float x){
    union{
      int intPart;
    float floatPart;
     } convertor;
    union{
      int intPart;
    float floatPart;
     } convertor2;
  convertor.floatPart = x;
 convertor2.floatPart = x;
convertor.intPart = 0x1FBCF800 + (convertor.intPart >> 1);
convertor2.intPart = 0x5f3759df - (convertor2.intPart >> 1);
return 0.5f*(convertor.floatPart + (x * convertor2.floatPart));
}
```

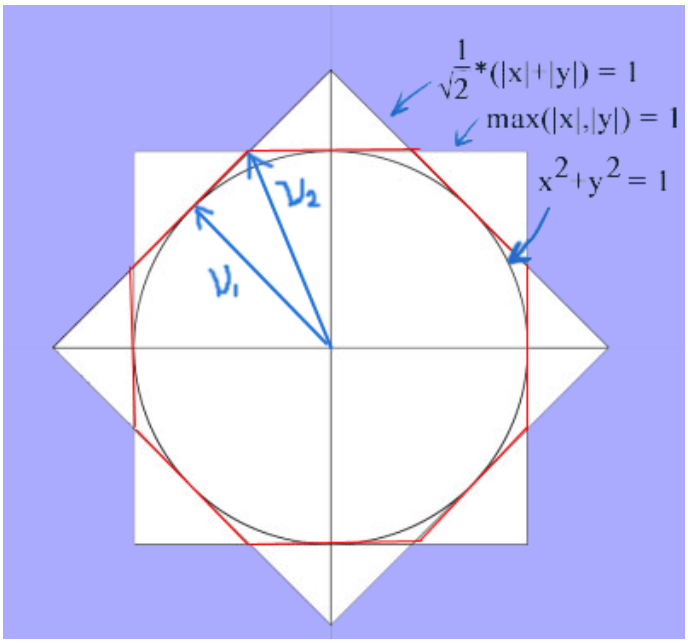
另一个基于同样算法的更高速度的sqrt实现如下。其只是简单地将指数除以2，并没有考虑尾数的方根。要看懂该代码的话必须知道，在IEEE浮点数的格式中，E是由实际的指数加127得到的。例如，如果实数是0.1234*2^10，在浮点表示中，E（第23-30位）的值其实为10+127=137。所以下面的代码中，要处理127偏移，这就是常数0x3f800000的作用。我没实际测试过该函数，所以对其优劣无从评论，但估计其精度应该会降低很多。

```
float Faster_Sqrtf(float f)
    {
   float result;
     _asm
      {
    mov eax, f
sub eax, 0x3f800000
    sar eax, 1
add eax, 0x3f800000
  mov result, eax
      }
  return result;
    }
```

除了基于NR的方法外，其他常见的快速算法还有**多项式逼近**。下面的函数取自《3D游戏编程大师技巧》，它使用一个多项式来近似替代原来的长度方程，但我搞不清楚作者使用的公式是怎么推导出来的（如果你知道的话请告诉我，谢谢）。

```
//
//  这个函数计算从(0,0)到(x,y)的距离，相对误差为3.5%
//
int FastDistance2D(int x, int y)
    {
  x = abs(x);
  y = abs(y);
int mn = MIN(x,y);
return(x+y-(mn>>1)-(mn>>2)+(mn>>4));
    }
//
// 该函数计算(0,0,0)到(x,y,z)的距离，相对误差为8%
//
float FastDistance3D(float fx, float fy, float fz)
    {
   int temp;
  int x,y,z;
 // 确保所有的值为正
x = int(fabs(fx) * 1024);
y = int(fabs(fy) * 1024);
z = int(fabs(fz) * 1024);
    // 排序
if (y < x) SWAP(x,y,temp)
if (z < y) SWAP(y,z,temp)
if (y < x) SWAP(x,y,temp)
int dist = (z + 11 * (y >> 5) + (x >> 2) );
  return((float)(dist >> 10));
    }
```

还有一种方法称为**Distance Estimates**（距离评估？），如下图所示：



红线所描绘的正八边形上的点为：

```
octagon(x,y) = min((1/√2) * (|x|+|y|), max(|x|,|y|))
```

求出向量v1和v2的长度，则：

```
√(x^2+y^2) = (|v1|+|v2|)/2 * octagon(x,y)
```

到目前为止我们都在讨论浮点数的方根算法，接下来轮到整数的方根算法。也许有人认为对整型数据求方根无任何意义，因为会得到类似99^(1/2)=9的结果。通常情况下确实是这样，但当我们使用定点数的时候（定点数仍然被应用在很多系统上面，例如任天堂的GBA之类的手持设备），整数的方根算法就显得非常重要。对整数开平方的算法如下。我并不打算在这讨论它（事实是我也没有仔细考究，因为在短期内都不会用到- -b），但你可以在文末**James Ulery**的论文中找到非常详细的推导过程。

```
//
// 为了阅读的需要，我在下面的宏定义中添加了换行符
//
#define step(shift)
if((0x400000001 >> shift) + sqrtVal <= val)
{
  val -= (0x400000001 >> shift) + sqrtVal;
sqrtVal = (sqrtVal >> 1) | (0x400000001 >> shift);
}
else
{
sqrtVal = sqrtVal >> 1;
}
//
// 计算32位整数的平方根
//
int32 xxgluSqrtFx(int32 val)
{
// Note: This fast square root function
  // only works with an even Q_FACTOR
        int32 sqrtVal = 0;
     step(0);
     step(2);
     step(4);
     step(6);
     step(8);
    step(10);
    step(12);
```

```
        step(14);
        step(16);
        step(18);
        step(20);
        step(22);
        step(24);
        step(26);
        step(28);
        step(30);
    if(sqrtVal < val)
            {
        ++sqrtVal;
            }
sqrtVal <<= (Q_FACTOR)/2;
    return(sqrtVal);
            }
```

关于sqrt的话题早在2003年便已在 GameDev.net上得到了广泛的讨论（可见我实在非常火星了，当然不排除还有其他尚在冥王星的人，嘿嘿）。而尝试探究该话题则完全是出于本人的兴趣和好奇心（换句话说就是无知）。其实现在随着FPU的提升和对向量运算的硬件支持，大部分系统上都提供了快速的sqrt实现。如果是处理大批量的向量的话，据说最快的方法是使用SIMD（据说而已，我压根不懂），可同步计算4个向量。

相关资源

这里是当年在GameDev.net的讨论，有趣的东西包括一些高手的评论和几个版本的sqrt的实测数值。

有关NR和泰勒级数的内容，请参见MathWorld。

还有两篇论文。一篇是关于Carmack算法的推导过程；另一篇是关于整数方根算法的推导过程：

- Fast Inverse Square Root by **Chris Lomont**（PDF）
- Computing Integer Square Root by **James Ulery**（PDF）

以上观点仅代表我个人看法，由于水平有限，如有错漏，还望指正。此外，如果你有任何看法或意见的话，欢迎到论坛留言或者给我发E-Mail：**wang_yong_cong@msn.com**。

相关文章

求平方根的代码