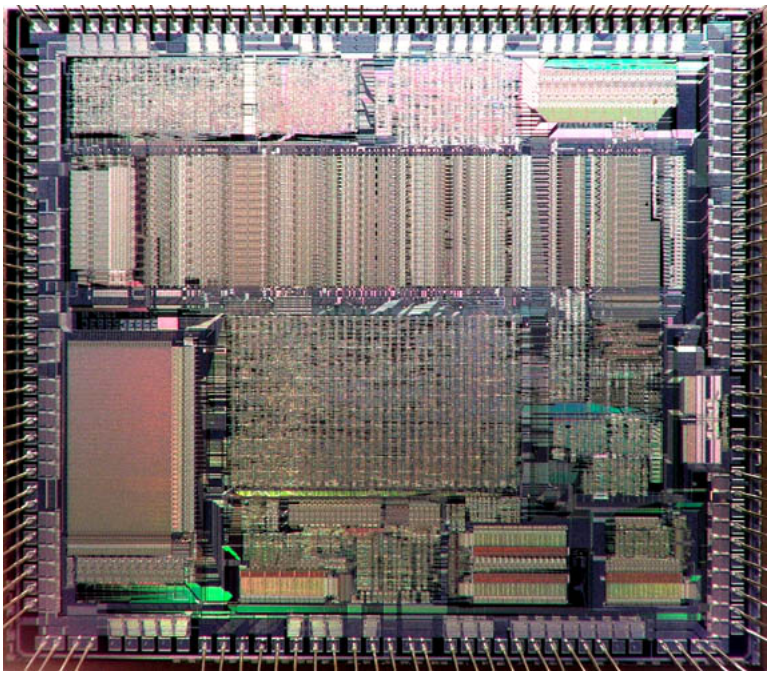
PatientZero вчера в 16:44 Разработка

Наглядное объяснение чисел с плавающей запятой

http://fabiensanglard.net/floating_point_visually_explained/index.php

Программирование, Математика, Алгоритмы

Перевод



В начале 90-х создание трёхмерного игрового движка означало, что вы заставите машину выполнять почти не свойственные ей за Персональные компьютеры того времени предназначались для запуска текстовых процессоров и электронных таблиц, а не для 3D вычислений со частотой 70 кадров в секунду. Серьёзным препятствием стало то, что, несмотря на свою мощь, ЦП не имел аппарат устройства для вычислений с плавающей запятой. У программистов было только АЛУ, перемалывающее целые числа.

При написании книги *Game Engine Black Book: Wolfenstein 3D* я хотел наглядно показать, насколько был велики были проблемы при работе без плавающей запятой. Мои попытки разобраться в числах с плавающей запятой при помощи каноничных статей мозг воспринимал в штыки. Я начал искать другой способ. Что-нибудь, далёкое от $(-1)^S * 1.M * 2^{(E-127)}$ и их загадочных экспонент с мантиссами. Может быть, в виде рисунка, потому что их мой мозг воспринимает проще.

В результате я написал эту статью и решил добавить её в книгу. Не буду утверждать, что это моё изобретение, но пока мне не приходилось видеть такого объяснения чисел с плавающей запятой. Надеюсь, статья поможет тем, у кого, как и меня, аллергия на математические обозначения.

Как обычно объясняют числа с плавающей запятой

Цитирую Дэвида Голдберта (David Goldbert):

> *Для многих людей арифметика с плавающей запятой кажется каким-то тайным знанием.*

Полностью с ним согласен. Однако важно понимать принципы её работы, чтобы полностью осознать её полезность при программировании 3D-движка. В языке C значения с плавающей запятой — это 32-битные контейнеры, соответствующие стандарт IEEE 754. Они предназначены для хранения и выполнения операций над аппроксимациями вещественных чисел. Пока я видел тольк такое их объяснение. 32 бита разделены на три части:

- S (1 бит) для хранения знака
- E (8 бит) для экспоненты

- M (23 бита) для мантиссы

Внутренности числа с плавающей запятой.

S	EXPONENT	MANTISSA

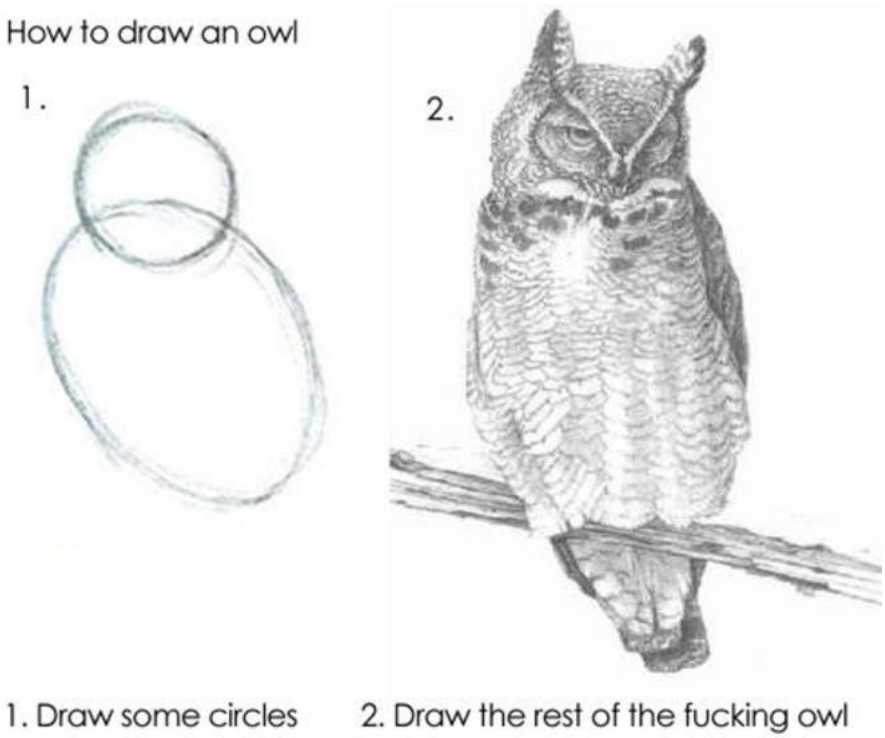
Три части числа с плавающей запятой.

Пока всё нормально. Пойдём дальше. Способ интерпретации чисел обычно объясняется с помощью такой формулы:

$$(-1)^S * 1, M * 2^{(E-127)}$$

Именно это объяснение чисел с плавающей запятой все ненавидят.

И здесь я обычно начинаю терять терпение. Возможно, у меня аллергия на математическую нотацию, но когда я это читаю, в моём мозгу ничего не «щёлкает». Такое объяснение похоже на способ рисования совы:



Другой способ объяснения

Хоть это изложение и верно, такой способ объяснения чисел с плавающей запятой обычно не даёт нам никакого понимания. Я вин ужасную запись в том, что она разочаровала тысячи программистов, испугала их до такой степени, что они больше никогда не пыт понять, как же на самом деле работают вычисления с плавающей запятой. К счастью, их можно объяснить иначе. Воспринимайте экспоненту как окно (Window) или интервал между двумя последовательными степенями двух целых чисел. Мантиссу воспринимай как смещение (Offset) в этом окне.

S	WINDOW	OFFSET

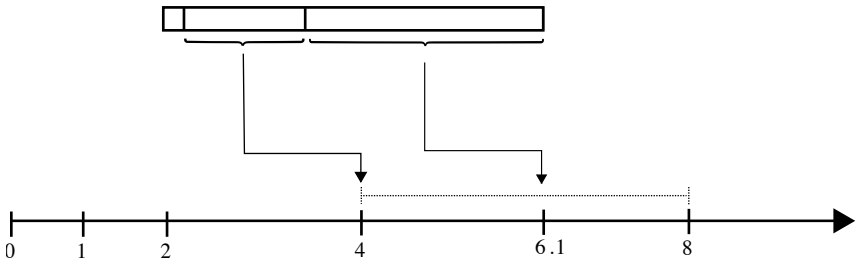
Три части числа с плавающей запятой.

Окно сообщает нам, между какими двумя последовательными степенями двойки будет число: [0,1], [1,2], [2,4], [4,8] и так далее (вп до $[2^{127}, 2^{128}]$. Смещение разделяет окно на $2^{23} = 8388608$ сегментов. С помощью окна и смещения можно аппроксимировать чис Окно — это отличный механизм защиты от выхода за границы. Достигнув максимума в окне (например, в [2,4]), можно «переплыть вправо и представить число в пределах следующего окна (например, [4,8]). Ценой этого будет только небольшое снижение точнос потому что окно становится в два раза больше.

Викторина: сколько точности теряется, когда окно закрывает больший интервал? Давайте возьмём пример с окном [0,1], в котором 8388608 смещений накладываются на интервал размером 1, что даёт нам точность $\frac{(1-0)}{8388608} = 0,00000011920929$. В окне [2048,40

8388608 смещений накладываются на интервал $(4096 - 2048) = 2048$, что даёт нам точность $\frac{(4096-2048)}{8388608} = 0,0002$.

На рисунке ниже показано, как кодируется число 6,1. Окно должно начинаться с 4 и заканчиваться следующей степенью двойки, т.
Смещение находится примерно посередине окна.



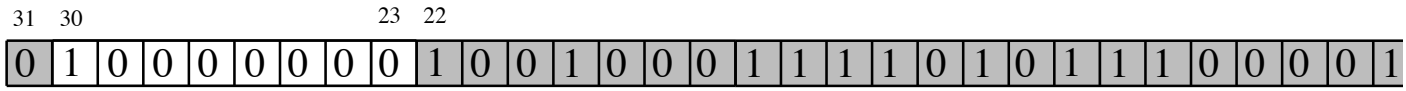
Значение 6,1 аппроксимированное с помощью числа с плавающей запятой.

Давайте возьмём ещё один пример с подробным вычислением представлением в виде числа с плавающей точкой хорошо известно
всем нам значения: 3,14.

- Число 3,14 положительно $\rightarrow S = 0$.
- Число 3,14 находится между степенями двойки 2 и 4, то есть окно числа с плавающей запятой должно начинаться с $2^1 \rightarrow E =$
(см. формулу, где окно — это $2^{(E-127)}$).
- Наконец, есть 2^{23} смещений, которыми можно выразить расположение 3,14 внутри интервала [2-4]. Оно находится в $\frac{3,14-2}{4-2} =$
внутри интервала, что даёт нам смещение $M = 2^{23} * 0,57 = 4781507$

В двоичном виде это преобразуется в следующее:

- S = 0 = 0b
- E = 128 = 10000000b
- M = 4781507 = 10010001111010111000011b



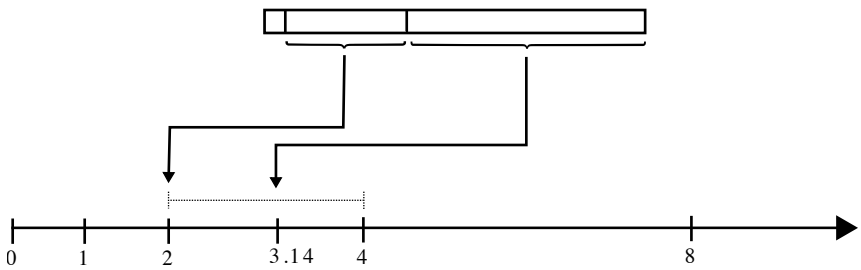
Двоичное представление с плавающей точной числа 3,14.

То есть значение 3,14 аппроксимируется как 3,14000010490417480468 75.

Соответствующее значение в непонятной формуле:

$$3,14 = (-1)^0 * 1,57 * 2^{(128-127)}$$

И, наконец, графическое представление с окном и смещением:

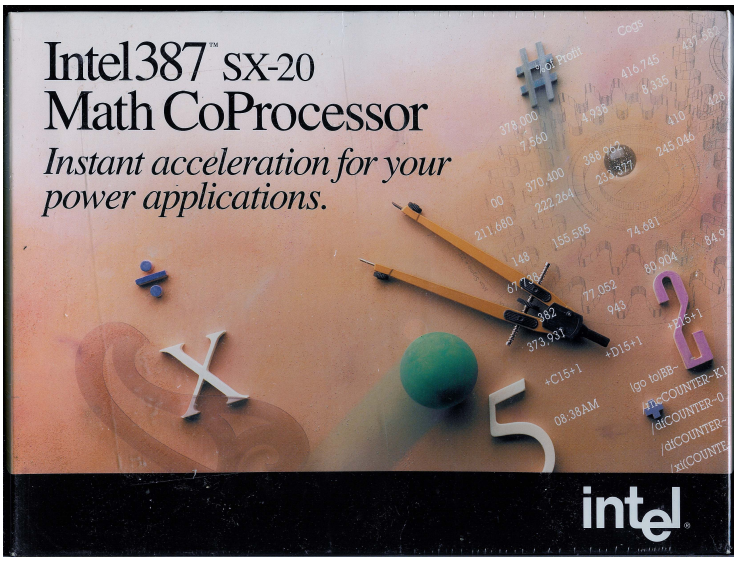


Окно и смещение числа 3,14.

Интересный факт: если модули операций с плавающей запятой были такими медленными, почему в языке C в результате использ
типы *float* и *double*? Ведь в машине, на которой изобретался язык (PDP-11), не было модуля операций с плавающей запятой! Дело в
что производитель (DEC) пообещал Деннису Ритчи и Кену Томпсону, что в следующей модели он будет. Они были любителями
астрономии и решили добавить в язык эти два типа.

Интересный факт: те, кому в 1991 году действительно нужен был аппаратный модуль операций с плавающей запятой, могли его к

Единственными, кому он мог понадобиться в то время, были учёные (по крайней мере, так Intel понимала потребности рынка). На р они позиционировались как «математические сопроцессоры». Их производительность была средней, а цена огромной (200 долла 1993 года — это 350 долларов в 2016 году.). В результате уровень продаж оказался посредственным.



Надеюсь, статья была вам полезна!

Метки: floating point, числа с плавающей точкой/запятой, экспонента, мантисса

+58 214 17,2k 25

350,5 Карма 519,2 Рейтинг 450 Подписчики

@PatientZero
Переводчик-фрилансер

Поделиться публикацией

ПОХОЖИЕ ПУБЛИКАЦИИ

22 мая 2015 в 10:02
Всё, точка, приплыли! Учимся работать с числами с плавающей точкой и разрабатываем альтернативу с фиксированной точностью десятичной дроби
+17 76,8k 256 13

16 апреля 2014 в 15:11
Пара слов о числах с плавающей точкой в Java
+16 65,2k 253 32

12 октября 2011 в 23:20
Арифметические операции над числами с плавающей точкой
+10 19,5k 97 19

Комментарии 25

rkfg 06.09.17 в 17:03

Более-менее понятно. Интересное следствие: из такой формы следует, что уже у сравнительно небольших чисел, порядка 2^24, падает точ представления даже целой части (потому что мантисса делит всегда на 2^23 элемента, и если элементов между двумя точками будет боль то каждому второму не достанется своего значения):

```
#include <iostream>

using namespace std;

int main() {
  cout << (16777215.f == 16777216.f) << endl;
  cout << (16777216.f == 16777217.f) << endl;
}
```

Выведет 0 и 1, т.е. во float-представлении 16777216 равняется 16777217. Мне кажется, этот пример даже интереснее известного многим 0.1+

Интересно было бы почитать в доступной форме, как производятся математические операции с float-числами в таком виде и как можно эффективно перевести число из обычной строки во float.

ToSHiC 07.09.17 в 00:07

В свете вашего комментария стоит отметить, что в javascript все числа — это double. Там, конечно, диапазон целых значений, вычисляем без погрешности, больше, но он заметно меньше, чем 2^64.

Ares_ekb 07.09.17 в 07:38

Есть ещё интересный пример:

1/3 + 1/3 + 1/3 = ?

Складываем 1/3 сначала 3 раза, затем 30 раз, 300 раз и т.д.:

```
float floatValue = 1F / 3F;
double doubleValue = 1D / 3D;
decimal decimalValue = 1M / 3M;
for (int i = 0; i <= 6; i++) {
    float floatResult = 0;
    double doubleResult = 0;
    decimal decimalResult = 0;
    int times = Convert.ToInt32(3*Math.Pow(10,i));
    for (int j = 1; j <= times; j++) {
        floatResult += floatValue;
        doubleResult += doubleValue;
        decimalResult += decimalValue;
    }
    Console.WriteLine("sum 1/3 times: {0}" , times);
    Console.WriteLine("flt = {0}", floatResult);
    Console.WriteLine("dbl = {0}", doubleResult);
    Console.WriteLine("dec = {0}", decimalResult);
    Console.WriteLine();
}
Console.WriteLine("flt = {0}", floatValue*3000000);
Console.WriteLine("dbl = {0}", doubleValue*3000000);
Console.WriteLine("dec = {0}", decimalValue*3000000);
```

Угадайте какой будет результат

```
sum 1/3 times: 3
flt = 1
dbl = 1
dec = 0,99999999999999999999999999999

sum 1/3 times: 30
flt = 9,999999
dbl = 10
dec = 9,99999999999999999999999999997

sum 1/3 times: 300
flt = 100,0002
dbl = 99,9999999999997
dec = 99,99999999999999999999999972

sum 1/3 times: 3000
flt = 999,9764
dbl = 1000,00000000004
dec = 999,99999999999999999999999720

sum 1/3 times: 30000
flt = 9999,832
dbl = 10000,0000000003
dec = 9999,99999999999999999997203

sum 1/3 times: 300000
flt = 100165,5
dbl = 99999,9999996892
dec = 99999,99999999999999999972026

sum 1/3 times: 3000000
flt = 976144,6
dbl = 1000000,00004317
dec = 999999,99999999999999999720256

flt = 1000000
dbl = 1000000
dec = 999999,99999999999999999999999
```

Видно, что при сложении 1/3 три раза float и double дают более точный результат, чем decimal. Правда при большом количестве сложен decimal точнее. С другой стороны, если не складывать 1/3 три миллиона раз, а сразу умножить, то decimal менее точный.

Всё это как-то связано с 60-ричной системой счисления. Я думаю, что при работе с долями (весами) или с географическими координатами арифметика с плавающей запятой может быть точнее. Если кто-то может рассказать подробней было бы интересно.

ustaspolansky 07.09.17 в 10:02

Разве во флоат не до 7 знаков включительно? Возможно это связано с большим кол. во знаков и некорректной обработкой именно на вы

Я не программист, не ругайте.

rkfg 07.09.17 в 11:23

Там не в знаках считается, а в битах. В статье сказано, что мантисса составляет 23 бита, так что максимальная достижимая точность н целых числах будет, когда между 2^E и 2^(E+1) (отбросим для простоты -127) будет 2^23 чисел, тогда каждому числу можно противопоставить одно значение мантиссы. Если же этих чисел больше, то можно найти такое число, которому уже не сопоставишь с выделенное значение мантиссы, вот я для примера выбрал участок, где 2^24 чисел, т.е. в два раза больше. Поэтому у чётных чисел ес своё значение мантиссы, а нечётным уже не хватает, и происходит «округление».

sbnur 06.09.17 в 17:07

Вообще то исходно архитектура процессоров разрабатывалась для выполнения арифметических операцуий с плавающей запятой — в частности, на таких системах я учился в свое время.
Но изобретение велосипедов сопутствует развитию — поэтому не удивляюсь

rfq 07.09.17 в 00:39

операции с плавающей запятой присутствовали в компьютерах с самого начала. Их не было только в специализированных компьютерах самых дешевых компьютерах общего назначения.

32bit_me 06.09.17 в 18:53

Мне как раз кажется понятной формула, и кажутся непонятными все эти объяснения с картинками. Формула сразу даёт понимание того, ка
этими числами работать, то есть как складывать, умножать и делить, пользуясь их бинарным представлением.
К тому же тема не раскрыта до конца, потому что, кроме описанной здесь нормализованной формы, есть также денормализованная форма
разные *нечисла*, типа NaN и Inf.

Wano987 06.09.17 в 22:36

Я, конечно, могу ошибаться, но каждому — своё.
Лично я односимвольные регистрочувствительные имена переменных/классов без обильных комментариев воспринимаю достаточно посредственно. Так что мне эта статья была небесполезна.

WebConn 07.09.17 в 00:12

С формулой понятно, как оперировать этими значениями.
С картинкой получилось как-то более явно увидеть грабли, спрятанные в арифметике с плавающей точкой.

rfq 07.09.17 в 00:45

Это вы увидели только первый слой граблей. А там внизу еще целая куча. И основная — поддержка точности. Точность теряется на вс
этапах вычислений, так что конечный результат может не иметь ничего общего с действительностью, несмотря на то, что все используемые в расчетах формулы были правильными.

slavap 07.09.17 в 10:02

Формула ужасна, объяснение, а особенно картинка, отличные. Наверное, нужно быть математиком, чтобы сходу понять, как эти числа де
глядя только на формулу.

Varim 07.09.17 в 02:55

экспонента — расстояние между шагами
мантиса — номер шага

в статье не увидел, почему записывают
3,14 = 1,57 *2 (2 в степени 1 равно 2, то есть экспонента 128-127 = 1)

вместо
3,14 = 3,14*1 (2 в степени 0 равно 1, то есть экспонента 127-127 = 0)

Varim 07.09.17 в 03:02

в двоичном виде мантиса 1.M, то есть выигрывают 1н бит, а в десятичном может быть 1.M или 9.M

Varim 07.09.17 в 03:09

2в23 * 0,57 = 4781506.56

igormu 07.09.17 в 06:57

Нет окна [0, 1]. Есть только [1/2, 1), [1/4, 1/2) и так далее. 0 в нормализованном представлении отобразить нельзя, поэтому он искусственно принят как (E=0, M=0).

GlebSemenov 07.09.17 в 10:02

Не могу согласиться с утверждением, что у процессора PDP-11 не было модулей плавающей точки. Был FIS (Floating Instruction Set) и кое-гд
FPP (Floating Point Processor)

DrAndyHunter 07.09.17 в 10:02

Статья доходчиво объясняет сложную для понимая вещь. Спасибо за перевод автору!
Но остался для меня один непонятный момент:

> То есть значение 3,14 аппроксимируется как 3,140000104904174804687 5.

Я не понял, как вычисляется вот эта часть 0,000000104904174804687 5?

GarryC 07.09.17 в 10:35

(1)10010001111....011b = 13170115d /4/2/2.../2 = 3,140000104904174804687 5.

Den3D 07.09.17 в 10:49

M = 2^23*0.57 = 8388608 *0.57 = 4781506,56
4781507 — 4781506,56 = 0.44
0.44*2/8388608 = 0,0000001049041748046875

GalayZloy 07.09.17 в 10:15

А есть процессоры или теории с другим представлением чисел с запятой? Вообще описанное представление оптимально или исрользуется традиции?

MacIn 07.09.17 в 11:56

del

Busla 07.09.17 в 10:29

> В языке С значения с плавающей запятой — это 32-битные контейнеры, соответствующие стандарту IEEE 754

неправда:
типов с плавающей запятой несколько
они платформозависимы
IEEE 754 в C99 носит рекомендательный характер, а C11 ссылается на более поздние стандарты

MacIn 07.09.17 в 11:50

Как же вы будете читать статьи типа такой:
sites.math.washington.edu/~morrow/336_12/papers/ben.pdf

Если простая матзапись числа с плавающей запятой непонятна?

PaulZi 07.09.17 в 15:20

Может нубский вопрос, а чем интересно такое представление отличается в плане скорости/точности?
(-1)^S * 0.S * 10 ^ (E-127)

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.