Floating Point

Tarcísio Fischer

tarcisio.fischer.cco@gmail.com

github.com/tarcisiofischer

Tópicos

- Introdução
- Resultados curiosos
- Representação
- Comparação aproximada
- Exemplos

Introdução

Problema: Representação de números Reais

Exemplo: 123.456

Introdução

Problema: Representação de números Reais
Exemplo: 123.456

Solução - Escolher um número fixo de valores antes e depois da vírgula
valor = 123456

(Ou seja, 123.456 = 123456 / 1000)

Introdução

Problema: Representação de números Reais
Exemplo: 123.456

Outra solução - Utilizar notação científica ($m \cdot B^e$), com B fixo
(Exemplo para B=10):
m = 1.23456
e = 2

(Ou seja, 123.456 = 1.23456 * 10^2)

floats & doubles utilizam essa notação, com B=2

Introdução

Conversão decimal para binário

1.625			= **1** + 0.625
			.
0.625	*2=	1.25	= **1** + 0.25
0.25	*2=	0.5	= **0** + 0.5
0.5	*2=	1.0	= **1** + 0.0

Resultado

1.625 = **1.101**

Introdução

Conversão binário para decimal

1.101b = 1 * 2^(0) + 1 * 2^(-1) + 0 * 2^(-2) + 1 * 2^(-3)

= 1.0 + 0.5 + 0.0 + 0.125

= **1.625**

Introdução

```
#include <stdio.h>
int main() {
    float x = 123.456;
    printf("%3.3f\n", x); // 123.456
}
```

Resultados Curiosos

Resultados Curiosos

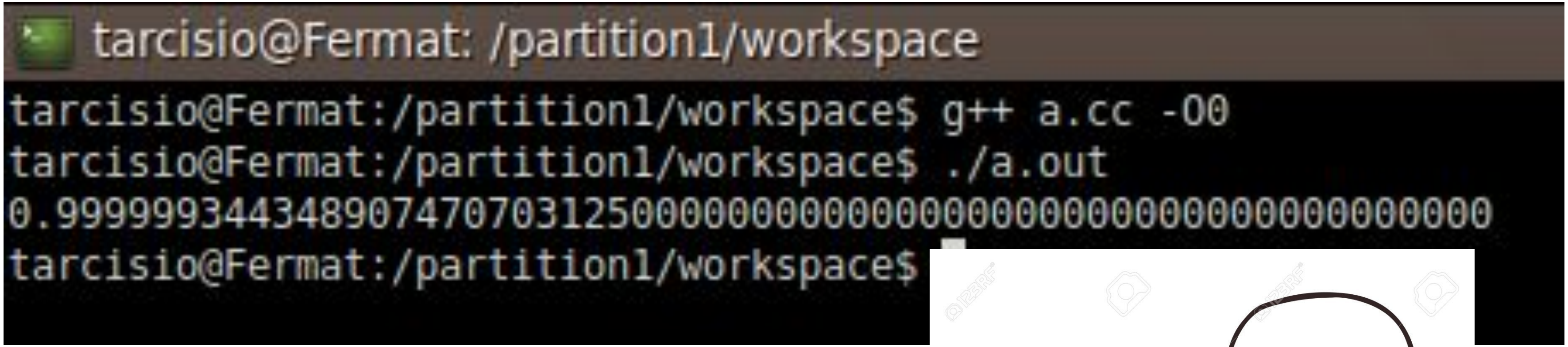
```
tarcisio@Fermat: /partition1/workspace
tarcisio@Fermat:/partition1/workspace$ cat example1.cc
#include <stdio.h>
int main() {
    printf("%1.40f\n", 0.1 + 0.2);
}
tarcisio@Fermat:/partition1/workspace$ g++ example1.cc && ./a.out
0.3000000000000000444089209850062616169453
tarcisio@Fermat:/partition1/workspace$
tarcisio@Fermat:/partition1/workspace$
tarcisio@Fermat:/partition1/workspace$ python -c "print(0.1 + 0.2)"
0.30000000000000004
tarcisio@Fermat:/partition1/workspace$
```

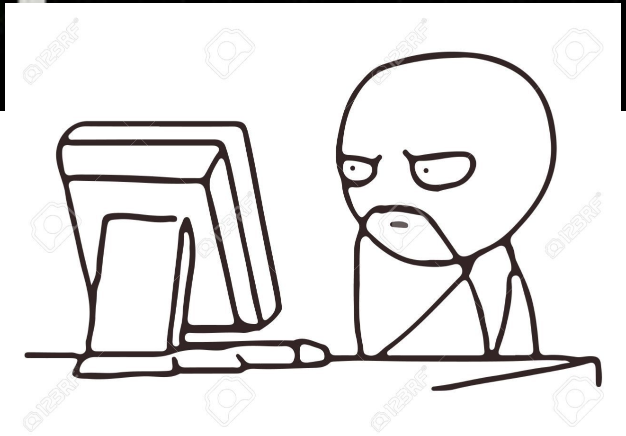
Resultados Curiosos

Resultados Curiosos

```
int main() {
  float sum = 0.0;
  for (int i = 0; i < 100; ++i)
    sum += 0.01;
  printf("%.60f\n", sum);
}
```

Resultados Curiosos



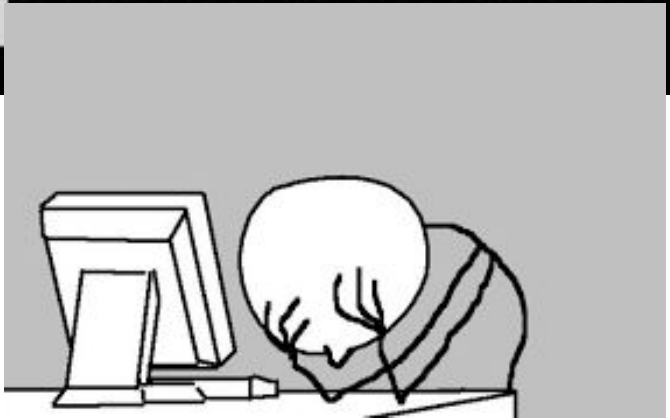


Resultados Curiosos

```
int main() {
  float a = 10e+5;
  float b = -10e+5;
  float c = 0.01;
  float result1 = (a + b) + c;
  float result2 = a + (b + c);
  printf("result1 = %.60f\n", result1);
  printf("result2 = %.60f\n", result2);
}
```

Resultados Curiosos

```
tarcisio@Fermat: /partition1/workspace
tarcisio@Fermat:/partition1/workspace$ g++ b.cc -O0
tarcisio@Fermat:/partition1/workspace$ ./a.out
result1 = 0.009999999776482582092285156250000000000000000000000000000000
result2 = 0.000000000000000000000000000000000000000000000000000000000000
tarcisio@Fermat:/partition1/workspace$
```



Resultados Curiosos

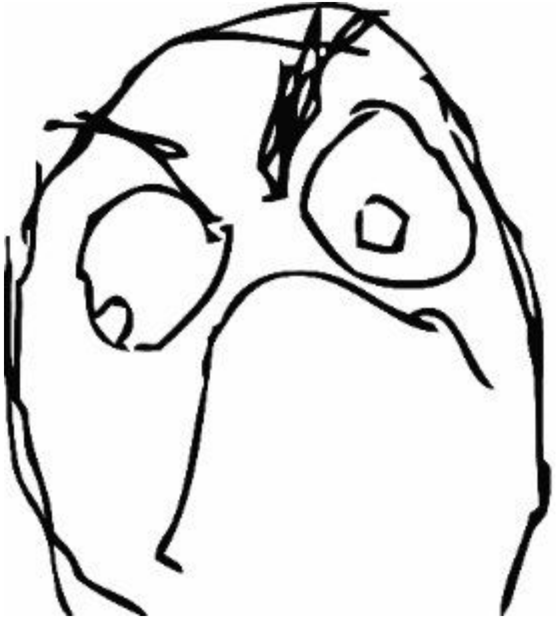
```
int main() {
  float a = 1.5f;
  for (int i = 0; i < 10000000; ++i) {
    a = a * 1.5f;
    a = a / 2.5f;
  }
}
```

Performance (média de 10 rodadas): **1.036s**

Resultados Curiosos

```
int main() {
  float a = 1.5f;
  for (int i = 0; i < 10000000; ++i) {
    a = a * 1.5f;
    a = a / 2.5f;
    a = a + 0.1f;
    a = a - 0.1f;
  }
}
```

Performance (média de 10 rodadas): **0.151s**



Representação

Representação (float & double)

Notação científica

$$m \cdot B^e$$

m = mantissa

e = expoente ou ordem de grandeza

B = base

Exemplos (B = 10)

0.1 = 1.0 * 10^(-1)

0.052 = 5.2 * 10^(-2)

891.7 = 8.917 * 10^(2)

Em binário (B = 2):

[bin] * 2^(e)

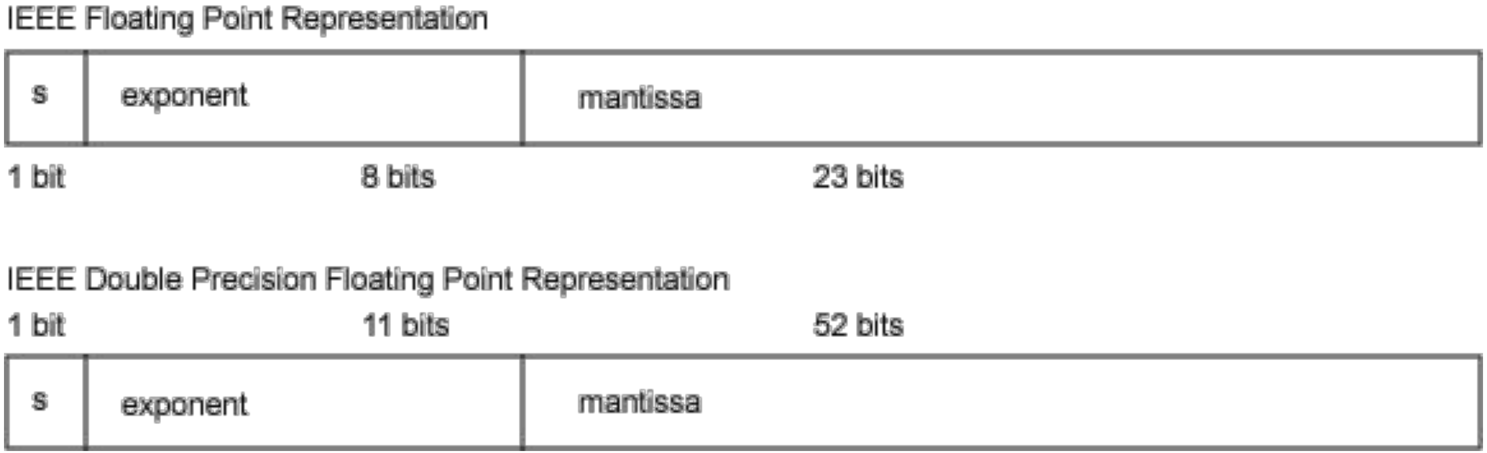
Representação (float & double)

"Squeezing infinitely many real numbers into a finite number of bits requires an approximate representation."

Referência: What Every Computer Scientist Should Know About Floating-Point Arithmetic

Representação (float & double)

IEEE 754 standard



Referência: http://www.ibm.com/developerworks/library/j-jtp0114/float.gif

Representação (float & double)

"Floating-point representations are not necessarily unique.
For example, both 0.01 × 101 and 1.00 × 10-1 represent 0.1."

Referência: What Every Computer Scientist Should Know About Floating-Point Arithmetic

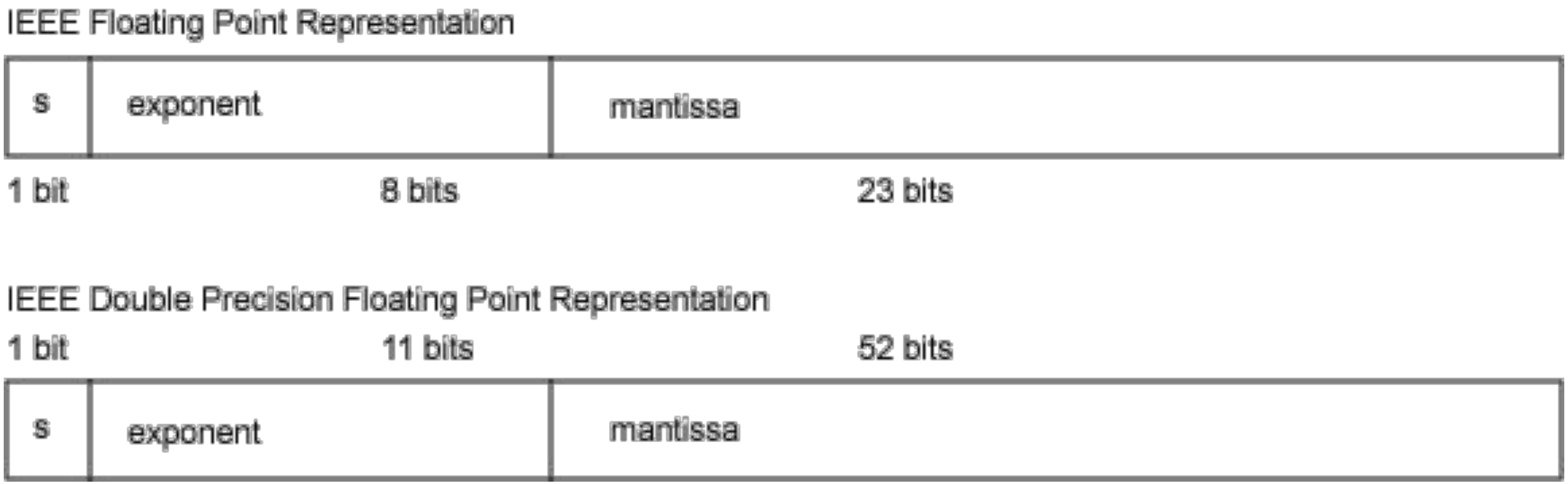
Representação (float & double)

`Normal form:   `**`1.[m]`**` * B**e`

Representação (float & double)

float `-1^s * 2^(e - 127 ) * (1.0 + m),   0.0 <= m < 1.0`

double `-1^s * 2^(e - 1023) * (1.0 + m),   0.0 <= m < 1.0`



Representação (float & double)

Tem exemplo?

Representação (float & double)

Exemplo: Representação de **0.5** em float & double

```
int main() {
  float f = 0.5f;
  printf("%.40f\n", f);
  int f_bits = *(int*)&f;
  printf("%8x\n", f_bits);

  double d = 0.5;
  printf("%.60f\n", d);
  long d_bits = *(long*)&d;
  printf("%16lx\n", d_bits);
}
```

0.50000000000000000000000000000...

3f000000

0.50000000000000000000000000000...

3fe0000000000000

Representação (float & double)

Exemplo: Representação de **0.5** em **float**

```
3f000000

0 01111110 00000000000000000000000

s = 0           =        => s = +
e = 01111110    = 126    => e - 127 = -1
m = 000...000   = 0      => m = 0

(-1)**0 1.0 * 2**(-1) = 0.5
```

Representação (float & double)

Exemplo: Representação de **0.5** em **double**

3fe0000000000000

```
0 01111111110 0000000000000000000000000000...
```

```
s = 0           =        => s = +
e = 01111110    = 1022   => e - 1023 = -1
m = 000...000   = 0      => m = 0
```

```
(-1)**0 1.0 * 2**(-1) = 0.5
```

Representação (float & double)

Exemplo: Representação de **0.01** em **float**

```
#include <stdio.h>
int main()
{
  float f = 0.01f;
  printf("%.40f\n", f);
  int f_bits = *(int*)&f;
  printf("%8x\n", f_bits);
}
```

0.00999999977648258209228515625...

3c23d70a

Representação (float & double)

Exemplo: Representação de **0.01** em **float**

(Os exemplos 1 e 2 tem relação com essa característica!)

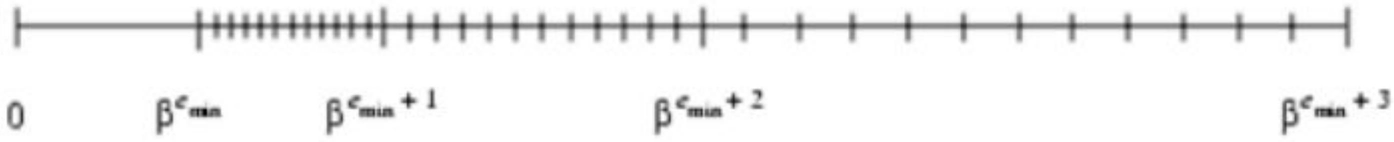
3c23d70a

0 01111000 01000111101011100001010

```
s = 0                  =        => s = +
e = 01111110           = 120    => e - 127 = -7
m = 010...010          = ...    => m = 1.2799999...
```

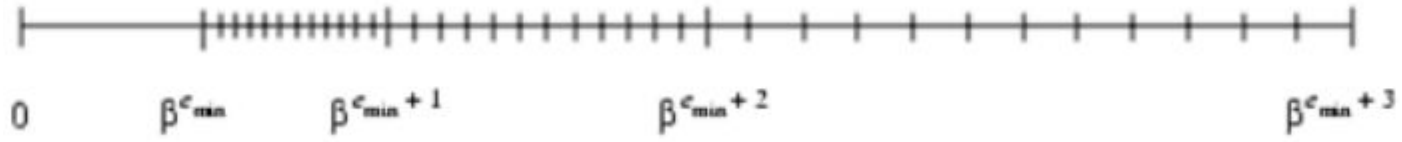
```
(-1)**0 1.m * 2**(-7) =
0.00999999977648258209228515625
```

Representação (float & double)



Nota: Quanto maior o expoente, maior o “passo” entre dois valores representáveis utilizando **float**

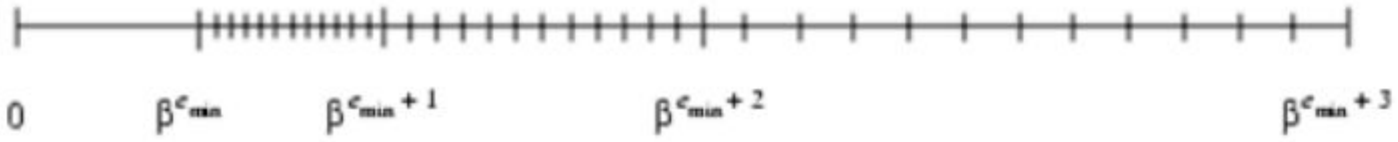
Representação (float & double)



Exemplo:
(Números pequenos possuem “passo” pequeno)
0.00003051759267691522836685180664062500
0.00003051759631489403545856475830078125 >> **Passo de ~10^-12**

(Números grandes possuem “passo” grande)
32768.01562500
32768.01953125 >> **Passo de ~10^-3**

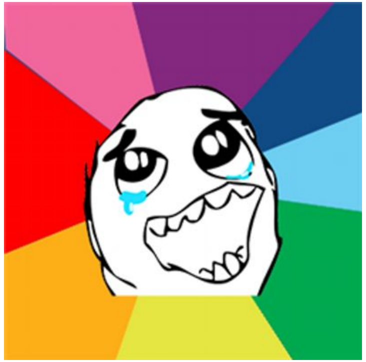
Representação (float & double)



Isso significa que operações envolvendo números com grandezas diferentes podem resultar em valores diferentes, ou seja,

não é verdade que
(a + b) + c == a + (b + c)
para todo a, b, c

O exemplo 3 tem relação com essa característica



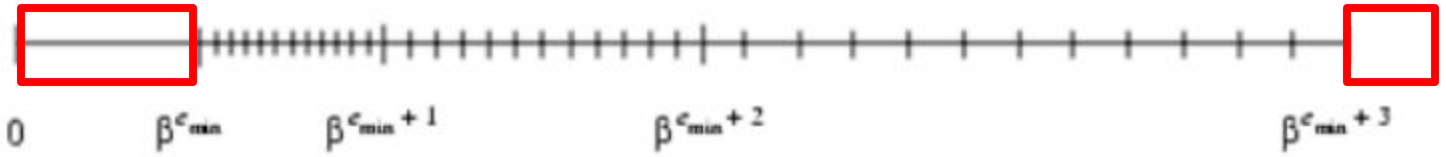
Overflow & Underflow

Overflow

Uma operação pode produzir um valor tão alto que não pode ser representado em um float/double.

Underflow

Uma operação pode produzir um valor tão próximo de 0.0 que não pode ser representado por um float/double.



Representação - Casos especiais

+0	0 00000000 00000000000000000000000
-0	1 00000000 00000000000000000000000
+inf	0 11111111 00000000000000000000000
-inf	1 11111111 00000000000000000000000
NaN	**<any signal>** 11111111 **<At least one 1 in m>**

Representação - Casos especiais

“Just as **NaNs** provide a way to continue a computation when expressions like **0/0** or **sqrt(-1)** are encountered, **infinities** provide a way to continue when an **overflow** occurs.”

Referência: What Every Computer Scientist Should Know About Floating-Point Arithmetic

```
int main() {
  printf("%.40f\n", sqrt(-1));          → NaN

  printf("%.40f\n", 0.0f/0.0f);         → NaN
  printf("%.40f\n", 1.0f/0.0f);         → Inf
}
```

Representação - Números Denormais/Subnormais

```
int main() {
  // 1.175494e-38
  printf("%e\n", std::numeric_limits<float>::min());
  // 1.175494e-38
  printf("%e\n", FLT_MIN);

  float really_small = std::numeric_limits<float>::min()/2f;
  // 5.877472e-39 -> Como é possível?
  printf("%e\n", really_small);
  // 2.938736e-39 -> Como é possível?
  printf("%e\n", really_small / 2.0f);
}
```

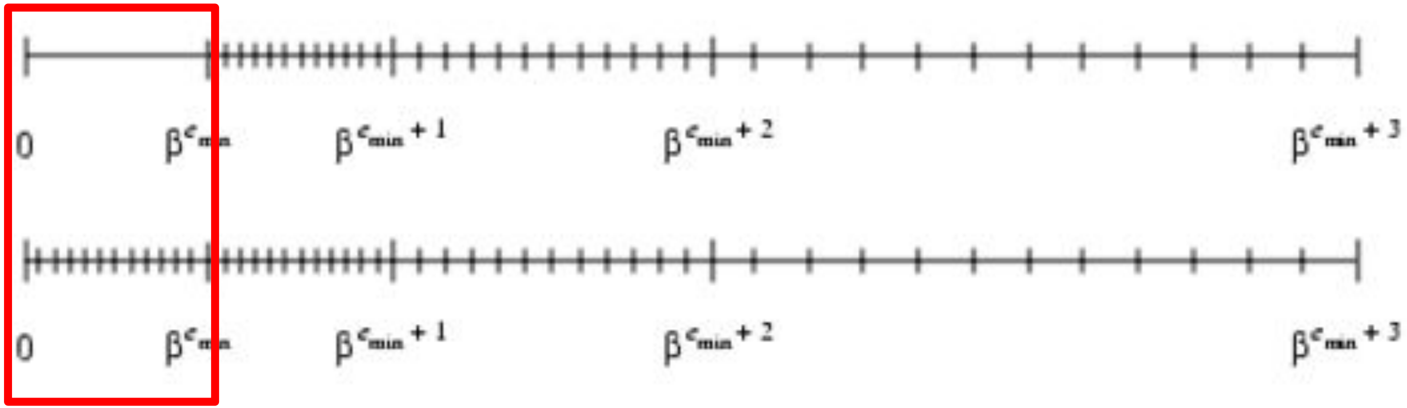
Representação - Números Denormais/Subnormais

+0	0 00000000 00000000000000000000000
-0	1 00000000 00000000000000000000000
+inf	0 11111111 00000000000000000000000
-inf	1 11111111 00000000000000000000000
NaN	<any signal> 11111111 <At least one 1 in m>
Denormal	~~0~~ 00000000 **<Any except all zeroes>**

0 ou 1

`0.[m] * 2 ** -126`

Representação - Números Denormais/Subnormais



Representação - Números Denormais/Subnormais

Mas.....

"(...) the speed of computation is significantly reduced on many modern processors"

Referência: https://en.wikipedia.org/wiki/Denormal_number

(O exemplo 4 tem relação com essa característica!)

Resultados Curiosos | Números Denormais

```
int main() {
  float a = 1.5f;
  for (int i = 0; i < 10000000; ++i) {
    a = a * 1.5f;
    a = a / 2.5f;


    a = a + 0.1f;  // Evita que números denormais sejam gerados
    a = a - 0.1f;  // (Mas note que o resultado será DIFERENTE)
  }
}
```

Performance (média de 10 rodadas): **0.151s**

Comparações com floats/doubles

Comparando floating points

```c
int main() {
  float sum = 0.0f;
  for (int i = 0; i < 100; ++i){
    sum += 0.01f;
  }
  if (sum == 0.01f * 100.0f) { printf("Equal!\n"); }
  else { printf("Different!\n"); }
}
```

Different!

0.99999934434890747070312500000000000000000000000000

1.00000000000000000000000000000000000000000000000000

Comparando floating points - Diferença absoluta

```
int main() {
  float sum = 0.0f;
  for (int i = 0; i < 100; ++i){
    sum += 0.01f;
  }

  float eps = 1e-8;
  float b = 0.01f * 100.0f;
  if (fabs(sum - b) < eps) { printf("Equal!\n"); }
  else { printf("Different!\n"); }
}
```

$$|a - b| < \epsilon$$

Se a e b forem muito grandes, sua diferença pode ser sempre um valor maior que epsilon

Comparando floating points - Diferença relativa

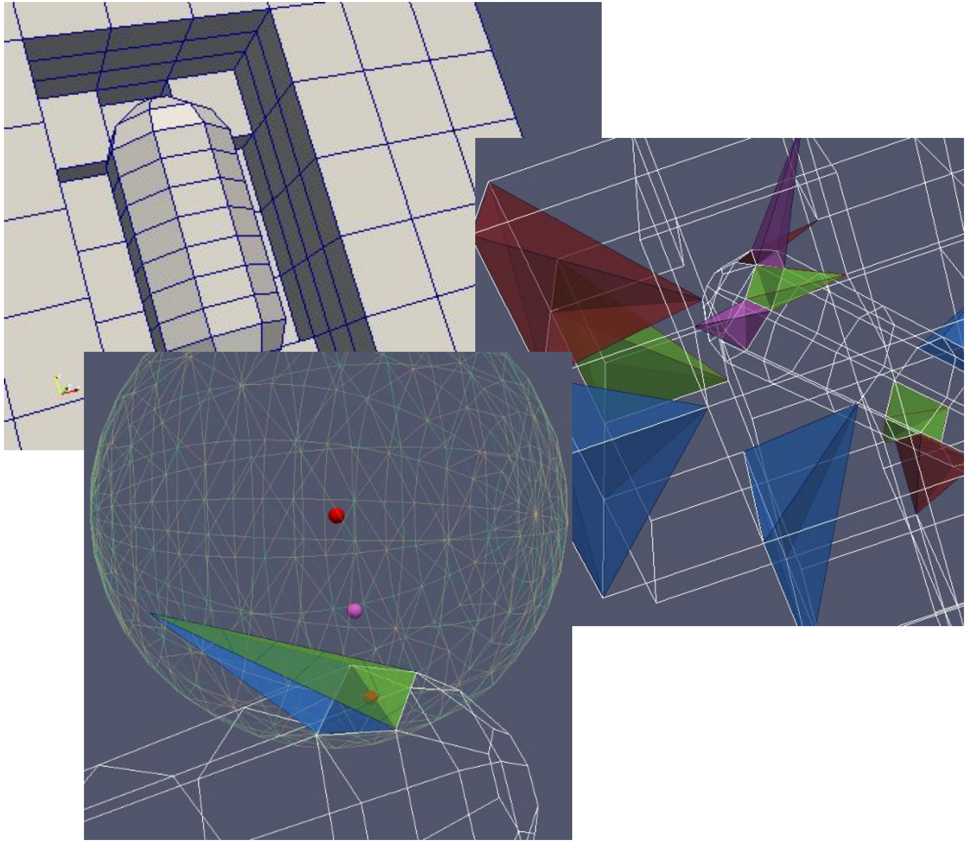
```
int main() {
  float sum = 0.0f;
  for (int i = 0; i < 100; ++i){
    sum += 0.01f;
  }

  float eps = 1e-8;
  float b = 0.01f * 100.0f;
  if (fabs((sum - b) / b) < eps) { printf("Equal!\n"); }
  else { printf("Different!\n"); }
}
```

$$\frac{|a-b|}{b} < \epsilon$$

E se b == 0?
E se a == b == 0?
Comparar esses casos separadamente?

Exemplos reais

Geração automatizada de malhas tridimensionais



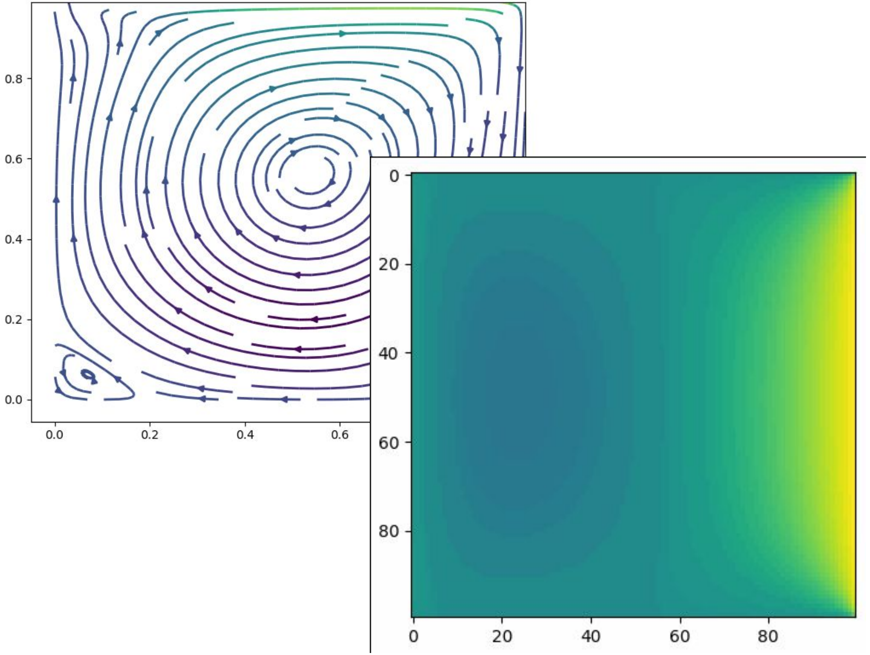
- Utiliza ponto flutuante para a representação dos nós no espaço 3D (x, y, z)

- Algoritmos de limpeza de nós muito próximos

- Algoritmos de detecção de elementos com volume “muito pequeno”

- Algoritmos de detecção de elementos muito “finos”

Simulação (Difusão de calor / Escoamento fluido)



- Utilização de ponto flutuante para a representação das grandezas físicas (Temperatura, Pressão, Velocidade…)

- Algoritmos numéricos iterativos para **aproximar** soluções de equações diferenciais

- Cuidados ao testar resultados (Utilização de métodos de comparação que levam em conta a grandeza esperada das soluções)

Perguntas?

Referências

“What Every Computer Scientist Should Know About Floating-Point Arithmetic” - DAVID GOLDBERG

https://www.floating-point-gui.de/

https://randomascii.wordpress.com/ ← **RECOMENDADO!**

Representação - Casos especiais

“The rule for determining the result of an operation that has **infinity** as an operand is simple: Replace **infinity** with a finite number x and take the **limit as x** → ∞”

“When the limit doesn't exist, the result is a NaN”

Referência: What Every Computer Scientist Should Know About Floating-Point Arithmetic

```c
#include <stdio.h>
#include <math.h>
int main()
{
  float inf = 1.0f / 0.0f;
  printf("%.40f\n", 1.0f / inf);
  printf("%.40f\n", 4.0f - inf);
  printf("%.40f\n", sqrt(inf));
}
```

lim(x→∞){1/x} = 0.00000000

lim(x→∞){4 - x} = -inf

lim(x→∞){sqrt(x)} = +inf

Representação - Casos especiais

“Since the sign bit can take on two different values, there are **two zeros, +0 and -0**.”

“If zero did not have a sign, then the relation 1/(1/x) = x would fail to hold when x = +/-∞”

Referência: What Every Computer Scientist Should Know About Floating-Point Arithmetic

```c
#include <stdio.h>
int main()
{
  float inf = 1.0f / 0.0f;
  printf("%.40f\n", 1.0f/+inf);          // +0.0
  printf("%.40f\n", 1.0f/(1.0f/+inf));   // +inf
  printf("%.40f\n", 1.0f/-inf);          // -0.0
  printf("%.40f\n", 1.0f/(1.0f/-inf));   // -inf
}
```