Magic	Two Worlds		

# Using Floating Point Without Losing Your Sanity

### Don Clugston

Sociomantic Labs GmbH

May 2016

< <p>I I

A.

э.

	Two Worlds		
Sanity Cheo	cks		

• Sanity Checks

Magic	Two Worlds		
Sanity (	Checks		

- Sanity Checks
- A Crisis of Confidence

Magic	Two Worlds	Unique to D	Rounding	Bisection
Sanity Chec	cks			

- Sanity Checks
- A Crisis of Confidence
- The specialists, "Numerical Analysts", are rare -- yet ordinary programmers need to use floating point

< D >

Magic	Two Worlds		
Sanity Chec	cks		

- Sanity Checks
- A Crisis of Confidence
- The specialists, "Numerical Analysts", are rare -- yet ordinary programmers need to use floating point
- It's more fun if you view it as magic

< 🗆 🕨

Magic	Two Worlds		
A Child's Ma	agic Trick		

Think of a number ...

Double it

Add 8

Halve it

Take away the number you first thought of

And your answer is ...

Sac

э

• • • • • • • •

∃ > < ∃ >

Magic	Two Worlds		
An Adul	t's Magic Trick		

```
• Think of a floating point number...
```

```
float magic ( float x ) {
return x + 35 - x;
}
```

Sac

э

∍ →

< • • •

< A

-

Magic	Two Worlds		
An Adult's M	lagic Trick		
• Think of a	floating point numb	er	

```
float magic ( float x )
{
return x + 35 - x;
}
```

```
• magic( 1000 ) == 35
```

Sac

3 🖌 3

∃ ▶ ∢

< • • •

< A

Magic	Two Worlds		
An Adult's N	lagic Trick		
<ul> <li>Think of a</li> </ul>	floating point numb	per	
float magic ( flo	pat x )		

```
return x + 35 - x;
```

ł

}

```
• magic( 1000 ) == 35
```

```
• magic( 1_000_000_000 ) == 64
```

Sac

3 🖌 3

∃ ▶ ∢

< • • •

< A

Magic	Two Worlds		
An Adul	t's Magic Trick		
<ul> <li>Think</li> </ul>	of a floating point	number	

```
float magic ( float x )
{
return x + 35 - x;
```

}

```
magic( 1000 ) == 35
```

- magic( 1\_000\_000\_000 ) == 64
- magic( 5\_000\_000\_000 ) == 0

3) 3

 $\bullet \Box \rightarrow$ 

Magic	Two Worlds	Unique to D	Rounding	Bisection
An Adult	's Magic Trick			

I hink of a floating point number...

```
float magic ( float x )
{
return x + 35 - x;
```

}

- magic( 1000 ) == 35
- magic( 1\_000\_000\_000 ) == 64
- magic( 5\_000\_000\_000 ) == 0
- "Catastrophic Cancellation"

< D >

Magic	Two Worlds		
Why does th	nis happen?		

• A float is 32 bits wide. It can only store 4 billion different numbers. 1000000035 is not one of them.

 Sac

< 🗆 🕨

A @ ► A ≡ ►

Magic	Two Worlds		
Why does	this happen?		

- A float is 32 bits wide. It can only store 4 billion different numbers. 1000000035 is not one of them.
- 100000000 and 100000064 are the closest available numbers

4 🗆 🕨

Magic	Two Worlds	Unique to D	Rounding	Bisection
Why c	loes this happen?			

- A float is 32 bits wide. It can only store 4 billion different numbers. 1000000035 is not one of them.
- 100000000 and 100000064 are the closest available numbers
- In float land, 1000000035 == 100000064

Magi	Two Worlds	Unique to D	Rounding	Bisection
W	hy does this happe	en?		
	• A float is 32 bits wide	It can only store 4 billi	on different numb	ore

- A float is 32 bits wide. It can only store 4 billion different numbers. 1000000035 is not one of them.
- 100000000 and 100000064 are the closest available numbers
- In float land, 1000000035 == 100000064
- Putting the uncountably infinite real number line...

4 🗆 🕨

Magic	Two Worlds	Unique to D	Rounding	Bisection
Wh	y does this happen?			
G	A float is 32 bits wide. It ca	in only store 4 b	llion different numbe	ers.

- 1000000035 is not one of them.1000000000 and 100000064 are the closest available numbers
- In float land, 1000000035 == 100000064
- Putting the uncountably infinite real number line...
- ... into a 32 bit float

Magic	Two Worlds	Unique to D	Rounding	Bisection
Why o	does this happen?			
• A 1(	float is 32 bits wide. It ca 000000035 is not one of	an only store 4 bi them.	llion different numbe	ers.

- 100000000 and 100000064 are the closest available numbers
- In float land, 1000000035 == 100000064
- Putting the uncountably infinite real number line...
- ... into a 32 bit float

We're pulling a trillion rabbits out of a 32-bit hat

Magic	Two Worlds		
		a tabula	

## Floating point is a conjuring trick

• Cannot exactly represent

Magic	Two Worlds		
Floatin	g point is a conj	uring trick	

Cannot exactly represent

• PI

Using Floating Point Without Losing Your Sanity

2

글 🖌 🖌 글 🕨

•

Magic	Two Worlds		
Floating po	pint is a con	juring trick	
<ul><li>Cannot</li><li>PI</li></ul>	exactly represe	nt	

• sqrt(2)

(ロ)、(型)、(E)、(E)、(E)、(C)、(C)

Magic	Two Worlds		
Floa	ating point is a conju	ring trick	
•	Cannot exactly represent PI		
٩	sqrt(2)		

• 0.1

990

æ

< □ > < □ > < □ > < Ξ > < Ξ > .

Magic	Two Worlds	Unique to D	Rounding	Bisection
Floating	point is a conj	uring trick		
<ul> <li>Can</li> <li>PI</li> <li>sqrt(</li> <li>0.1</li> </ul>	not exactly represer 2)	nt		
Addi	tion isn't even asso	ciative		

Magic	Two Worlds	Unique to D	Rounding	Bisection
Floating	g point is a conj	uring trick		
<ul><li>Can</li><li>Pl</li><li>sqrt</li></ul>	not exactly represen	t		
• 0.1				
<ul><li>Add</li><li>(35)</li></ul>	ition isn't even assoc + 1000000000) - 100	ciative 00000000 == 64		

Magic	Two Worlds		
Float	ting point is a conju	ring trick	
• ( • F • s • 0	Cannot exactly represent Pl sqrt(2) ).1		
• A • ( • 3	Addition isn't even associa 35 + 1000000000) - 1000 35 + (1000000000 - 10000	ative 000000 == 64 000000) == 35	

Magic	Two Worlds	Unique to D	Rounding	Bisection
Floa	ting point is a conju	ring trick		
• • •	Cannot exactly represent PI sqrt(2) 0.1			
•	Addition isn't even associ (35 + 1000000000) - 1000 35 + (1000000000 - 1000	ative 0000000 == 64 000000) == 35	nt turo 2	
٩	Why do we use such a gr	otesque, fraudule	nt type?	

## Floating point is a success story

### • All modern engineering is based on floating point calculations

< D >

æ

э

э

Magic	Two Worlds		
Floating	point is a succ	cess story	

- All modern engineering is based on floating point calculations
- Floating-point hardware is ubiquitous

- ·

< D >

Magic	Two Worlds			
Floating	point is a succ	ess story		
● All m ● Floa	nodern engineering i ting-point hardware	s based on floating	point calculations	

• Total GPU power exceeds CPU power

DAG

э

4 □ ト < □ ト < 三 ト < 三 ト</p>

Magic	Two Worlds			
Floating	g point is a succ	ess story		
<ul><li>All n</li><li>Floa</li></ul>	nodern engineering is ting-point hardware i	s based on floating	point calculations	

- Total GPU power exceeds CPU power
- Despite being a horrendous approximation, 64 bit floating point is "good enough"

< D >

	Two Worlds		
Two worlds			

• The Mathematician's World

Magic	Two Worlds		
Two worlds			

- The Mathematician's World
- The uncountably infinite real number line

Sac

3. 3

. . . . . .

Magic	Two Worlds		
Two worlds			

- The Mathematician's World
- The uncountably infinite real number line
- The world where algebra works

< <p>I I

-

-

Magic	Two Worlds	Unique to D	Rounding	Bisection
Two worlds				

- The Mathematician's World
- The uncountably infinite real number line
- The world where algebra works
- The Magician's World

< D >

Magic	Two Worlds	Unique to D	Rounding	Bisection
Two worlds				

- The Mathematician's World
- The uncountably infinite real number line
- The world where algebra works
- The Magician's World
- In reality we only have 4-10 bytes

 $\bullet \Box \rightarrow$ 

Magic	Two Worlds	Unique to D	Rounding	Bisection
Two worlds				

- The Mathematician's World
- The uncountably infinite real number line
- The world where algebra works
- The Magician's World
- In reality we only have 4-10 bytes
- Sometimes we try too hard to stay in the Mathematician's World

Two Worlds		
tiono		

### 3 Misconceptions

#### • BELIEF: Floating point arithmetic is "fuzzy", not deterministic

Sac

э
	Two Worlds		
3 Misconce	ptions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra

< D >

	Two Worlds		
3 Misconce	ptions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.

Magic	Two Worlds		
3 Misconce	eptions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.
- In fact a double is a 54 bit int with a bonus

Magic	Two Worlds		
3 Misconce	ptions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.
- In fact a double is a 54 bit int with a bonus
- BELIEF: "100000064" means "every number between 1000000033 and 1000000095"

nan

Magic	Two Worlds		
3 Misconce	otions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.
- In fact a double is a 54 bit int with a bonus
- BELIEF: "100000064" means "every number between 1000000033 and 1000000095"
- REALITY: 100000064 means 100000064. 1000000033 is an alias for 1000000064

Magic	Two Worlds		
3 Misconce	otions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.
- In fact a double is a 54 bit int with a bonus
- BELIEF: "100000064" means "every number between 1000000033 and 1000000095"
- REALITY: 100000064 means 100000064. 1000000033 is an alias for 1000000064
- BELIEF: Floating point is weird

Magic	Two Worlds		
3 Misconce	otions		

- BELIEF: Floating point arithmetic is "fuzzy", not deterministic
- REALITY: Floats don't obey normal algebra BUT they obey floating-point algebra
- Every exact int calculation is exact in double.
- In fact a double is a 54 bit int with a bonus
- BELIEF: "100000064" means "every number between 1000000033 and 1000000095"
- REALITY: 100000064 means 100000064. 1000000033 is an alias for 1000000064
- BELIEF: Floating point is weird
- REALITY: Most real-world measurements are similar

Magic	Two Worlds		
Floa	ts are just ints with	a scale	
struc	t float {		
	bool sign;		
	int mantissa;		
	int exponent;		

• mantissa \* 2 ^^ exponent

2

▲□▶ ▲@▶ ▲ ≧▶ ▲ ≧▶

Magic	Two Worlds		
Floats a	re just ints with	a scale	
struct float	t {		
bool	l sign;		
int n	nantissa;		
int e	exponent;		

- mantissa \* 2 ^^ exponent
- If exponent is 0, it really is an integer

3 🖌 3

< D >

A ∰ ▶ A ≣ ▶ A

Magic	Two Worlds		
Floats	are just ints with	a scale	
struct fl	oat {		
b	ool sign;		
ir	nt mantissa:		

int exponent;

}

- mantissa \* 2 ^^ exponent
- If exponent is 0, it really is an integer
- Most important property is the precision: the number of bits in the mantissa.

< D >

Magic	Two Worlds		
Floats a	are just ints with	a scale	
struct floa	at { ol sign:		

int mantissa; int exponent;

mantissa \* 2 ^^ exponent

}

- If exponent is 0, it really is an integer
- Most important property is the precision: the number of bits in the mantissa.
- In D, float.mant\_dig gives the precision

Two Worlds		

## The Precision Budget

The larger the precision, the more extravagent you can be

float	22 bits
double	54 bits
real	64 bits
quadruple	112 bits

Operation	Cost
Multiplication	1 bit
Division	1 bit
Addition	Many
Take away the number you first thought	Bankrupt
of	

< D >

< 🗇

э

Magic	Two Worlds		
The Fu	nny Values		

• -0.0 exists, though it almost always means +0.0

Magic	Two Worlds		
The Fu	inny Values		

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero

< D >

The Funny Values	

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)

Magic	Two Worlds		
The Funny \	/alues		

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:

Magic	Two Worlds		
The Funny \	/alues		

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:
- Overflow: double.max \* 2 == double.infinity

Magic	Two Worlds	Unique to D	Rounding	Bisection
The Funny V	/alues			

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:
- Overflow: double.max \* 2 == double.infinity
- Division by zero: 1.0 / 0.0 == double.infinity

Magic	Two Worlds	Unique to D	Rounding	Bisection
The Funny V	/alues			

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:
- Overflow: double.max \* 2 == double.infinity
- Division by zero: 1.0 / 0.0 == double.infinity
- NaN happens when something evil happens

Magic	Two Worlds	Unique to D	Rounding	Bisection
The Funny \	/alues			

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:
- Overflow: double.max \* 2 == double.infinity
- Division by zero: 1.0 / 0.0 == double.infinity
- NaN happens when something evil happens
- double.infinity double.infinity is double.nan

Magic	Two Worlds	Unique to D	Rounding	Bisection
The Funny V	/alues			

- -0.0 exists, though it almost always means +0.0
- It exists because tiny numbers get rounded to zero
- If exponent is int.max, value is infinity or "Not a Number" (nan)
- Infinity happens when:
- Overflow: double.max \* 2 == double.infinity
- Division by zero: 1.0 / 0.0 == double.infinity
- NaN happens when something evil happens
- double.infinity double.infinity is double.nan
- 0.0 / 0.0 is double.nan

Magic	Two Worlds		
Floatin	ng Point Exceptio	ns	

- The hardware can generate hardware traps when funny values are produced. Most programs should enable the severe traps inside main()
- FloatingPointControl fpctrl;
- // Enable hardware exceptions for division by zero,
- // overflow to infinity, and invalid operations
- fpctrl.enableExceptions(FloatingPointControl.severeExceptions);

	Two Worlds			
Flo	ating Point Exceptions			
C	The hardware can generate produced. Most programs sh	hardware traps hould enable the	when funny values e severe traps inside	are e main()

FloatingPointControl fpctrl;

- // Enable hardware exceptions for division by zero,
- // overflow to infinity, and invalid operations
- fpctrl.enableExceptions(FloatingPointControl.severeExceptions);

## • Unfortunately there is no way to detect Catastrophic Cancellation

	Two Worlds		
"Don't use =	=="		

## • Why not? Because it destroys the illusion

э

3 1 4 3 1

•

DAG

	Two Worlds		
"Don't use =	:="		

- Why not? Because it destroys the illusion
- "x == y" really means:

x and y are equal to as many significant figures as the CPU supports

< D >

Sar

	Two Worlds		
"Don't use =	:="		

- Why not? Because it destroys the illusion
- "x == y" really means:

x and y are equal to as many significant figures as the CPU supports

Exposes the implementation

< D >

	Two Worlds		
"Don't use =	-="		

- Why not? Because it destroys the illusion
- "x == y" really means:
  x and y are equal to as many significant figures as the CPU supports
- Exposes the implementation
- But +0.0 == -0.0

	Two Worlds		
"Don't use	=="		

- Why not? Because it destroys the illusion
- "x == y" really means:
  x and y are equal to as many significant figures as the CPU supports
- Exposes the implementation
- But +0.0 == -0.0
- The horror: NaN != NaN

	Two Worlds		
"Don't use =	=="		

- Why not? Because it destroys the illusion
- "x == y" really means:
  x and y are equal to as many significant figures as the CPU supports
- Exposes the implementation
- But +0.0 == -0.0
- The horror: NaN != NaN
- Some implementation details are hidden

	Two Worlds		
"Don't use =	=="		

- Why not? Because it destroys the illusion
- "x == y" really means:
  x and y are equal to as many significant figures as the CPU supports
- Exposes the implementation
- But +0.0 == -0.0
- The horror: NaN != NaN
- Some implementation details are hidden
- == is still useful for low-level code and unittests.

nan

	Two Worlds		
Alternative	s to ==		

## • In D, "x is y" compares implementation, no tricks

< D >

- A 🗇

 DAG

Magic	Two Worlds		
Alternati	ves to ==		

- In D, "x is y" compares implementation, no tricks
- Can we create a 'better ==' ?

Magic	Two Worlds		
Alterna	tives to ==		

- In D, "x is y" compares implementation, no tricks
- Can we create a 'better ==' ?

• No :(

4 日 > 4 日 > 4 日 > 4 日 > 4 日 > 4 日 > 9 4 0 4

Magic	Two Worlds		
Alternatives	to ==		

- In D, "x is y" compares implementation, no tricks
- Can we create a 'better ==' ?
- No :(
- Reduce the number of bits that must be equal

< 🗆 🕨

	Two Worlds		
Alternatives	to ==		

- In D, "x is y" compares implementation, no tricks
- Can we create a 'better ==' ?
- No :(
- Reduce the number of bits that must be equal
- std.math.feqrel gives number of equal bits

		Two Worlds			
AI	Iternatives	to ==			

- In D, "x is y" compares implementation, no tricks
- Can we create a 'better ==' ?
- No :(
- Reduce the number of bits that must be equal
- std.math.feqrel gives number of equal bits
- How many must be equal? Arbitrary!
| Magic              | Two Worlds         |                      |  |  |  |  |
|--------------------|--------------------|----------------------|--|--|--|--|
| Alternatives to == |                    |                      |  |  |  |  |
| ● In D, "x is      | v" compares implen | nentation, no tricks |  |  |  |  |

- Can we create a 'better ==' ?
- No :(
- Reduce the number of bits that must be equal
- std.math.feqrel gives number of equal bits
- How many must be equal? Arbitrary!
- In Physics, there is no "exact equality" either

500

Magic	Two Worlds		
Alternatives	to ==		
<ul> <li>In D, "x is</li> </ul>	y" compares implen	nentation, no tricks	

- Can we create a 'better ==' ?
- No :(
- Reduce the number of bits that must be equal
- std.math.feqrel gives number of equal bits
- How many must be equal? Arbitrary!
- In Physics, there is no "exact equality" either
- Always need to specify the precision

500

	Two Worlds	Unique to D	
How D	Makes It Better		

## • Standard IEEE arithmetic, bizarro implemenations are forbidden

э

Sac

• • • • • • • •

3 > < 3 >

Magic	Two Worlds	Unique to D	
How D Mal	kes It Better		

- Standard IEEE arithmetic, bizarro implemenations are forbidden
- Built-in floating point properties

Sac

< • • •

.∋. ▶

Magic	Two Worlds	Unique to D	
How D N	Aakes It Better		

- Standard IEEE arithmetic, bizarro implemenations are forbidden
- Built-in floating point properties
- max, epsilon, mant\_dig, infinity, nan ...

< D >

Magic	Two Worlds	Unique to D				
How D Makes It Better						
• Sta	andard IEEE arithmetic	c, bizarro implemen	ations are forbidde	en		

- Built-in floating point properties
- max, epsilon, mant\_dig, infinity, nan ...
- Unit tests

⊒

Sac

< • • •

- A - A - B - A - B - A

Magic	Two Worlds	Unique to D		
How [	O Makes It Better			
• Si	tandard IEEE arithmetic	, bizarro impleme	nations are forbidde	en

- Built-in floating point properties
- max, epsilon, mant\_dig, infinity, nan ...
- Unit tests
- static if

Sac

< • • •

æ

.∋. ▶

3 🖌 3

Magic	Two Worlds	Unique to D				
How D Makes It Better						
<ul> <li>St</li> </ul>	andard IEEE arithmetic,	bizarro impleme	nations are forbidde	n		

- Built-in floating point properties
- max, epsilon, mant\_dig, infinity, nan ...
- Unit tests
- static if
- But sometimes we have Orwellian experiences...

	Two Worlds	Unique to D		
Some Num	erals Are More	Equal Tha	n Others	

• float x = 1.30;

Magic	Two Worlds	Unique to D			
Some Numerals Are More Equal Than Others					
floa	t x = 1.30;				
ass	ert( x == 1.30 ); // FA	ILS!!			

## Using Floating Point Without Losing Your Sanity

## Don Clugston

DAG

2

(日)

	Two Worlds	Unique to D		
Some Nu	imerals Are M	lore Equal Thar	n Others	
<ul><li>float x</li><li>assert</li><li>assert</li></ul>	= 1.30; ( x == 1.30 ); // FA ( x == 1.30f ); // O	AILS!! I <mark>K</mark>		

Magic	Two Worlds	Unique to D					
Son	Some Numerals Are More Equal Than Others						
0 0 0	float x = 1.30; assert( x == 1.30 ); // FAIL assert( x == 1.30f ); // OK double y = 1.30;	S!!					

Magic	Two Worlds	Unique to D		
Son	ne Numerals Are Mo	re Equal Tha	an Others	
•	float x = 1.30; assert( x == 1.30 ); // FAIL assert( x == 1.30f ); // OK	SII		
•	double $y = 1.30;$			
٩	assert( y == 1.30 ); // OK			

Magic	Two Worlds	Unique to D		
Som	e Numerals Are Mo	re Equal Tha	an Others	
0	float x = 1.30; assert( x == 1.30 ); // FAIL assert( x == 1.30f ); // OK	S!!		
•	double $y = 1.30;$			
•	assert( y == 1.30 ); // OK			

• assert( y == 1.30f ); // OK?!!!!

DAG

э

• • • • • • • • •

'글 ▶ ◀ 글 ▶

Magic	Two Worlds	Unique to D		
Some N	Numerals Are M	lore Equal Thar	n Others	
<ul><li>float</li><li>asse</li><li>asse</li></ul>	: x = 1.30; ert( x == 1.30 ); // FA ert( x == 1.30f ); // O	ILS!! K		
<ul> <li>doul</li> </ul>	ble y = 1.30;			
asse	ert( y == 1.30 ); // Oł	<		

- assert( y == 1.30f ); // OK?!!!!
- assert( y == x ); // FAILS

DAG

3 🖌 3

< • • •

< 🗇 🕨

-∢≣≯

	Two Worlds	Unique to D		
Sor	ne Numerals Are Mo	re Equal T	han Others	
0 0 0	float x = 1.30; assert( x == 1.30 ); // FAIL assert( x == 1.30f ); // OK	S!!		
٩	double $y = 1.30;$			
•	assert( y == 1.30 ); // OK			

- assert( y == 1.30f ); // OK?!!!!
- assert( y == x ); // FAILS
- assert( 1.30 == 1.30f ); // OK!!

3) 3

Sac

< D >

Magic	Two Worlds	Unique to D	
Sociomantio	c's Nine Trillion	Dollar Bug	
<ul> <li>Losing yo</li> </ul>	ur sanity, #1		
if ( price < 0 ) { if ( price ) { bid( Irou	error(); } nd( price ) );		
}	(1 / //		

Magic	Two Worlds	Unique to D	Rounding	Bisection
Sociomant	ic's Nine Tri	llion Dollar Bug		
<ul> <li>Losing y</li> </ul>	our sanity, #1			
if ( price < 0 ) if ( price ) {	{ error(); }			
bid( iroi }	una(price));			
price wa	s NaN			

æ

< □ > <@ > < 돌 > < 돌 > ...

996

Magic	Two Worlds	Unique to D	Rounding	Bisection
Sociomantio	c's Nine Trillion	Dollar Bug		
<ul> <li>Losing yo</li> </ul>	ur sanity, #1			
if ( price < 0 ) { if ( price ) { bid( lroun }	error();        } nd( price )       );			
price was	NaN			

• In an auction, we made a bid of \$9223372036855

Sac

3. 3

< ∃ > <

Magic	Two Worlds	Unique to D	Rounding	Bisection
Sociomar	ntic's Nine Tri	llion Dollar Bug		
<ul> <li>Losing</li> </ul>	your sanity, #1			
if ( price < 0 if ( price ) { bid( In }	) { error(); } ound( price ) );			
o price w	as NaN			

- In an auction, we made a bid of \$9223372036855
- DMD Issue #13489 never do an implicit cast from float to bool unless you can guarantee it is not NaN.

< D >

Sar

Magic	Two Worlds	Unique to D	
Generi	c Programming		

• Mathematically, reals are an extension of integers

Magic	Two Worlds	Unique to D	
Generi	ic Programming		

- Mathematically, reals are an extension of integers
- int and float both have hardware support

- ·

Sac

< D >

Magic	Two Worlds	Unique to D	
Generio	c Programming		

- Mathematically, reals are an extension of integers
- int and float both have hardware support
- Replace 'int' with 'double' and everything will compile

	Two Worlds	Unique to D	
Generic	Programming		

- Mathematically, reals are an extension of integers
- int and float both have hardware support
- Replace 'int' with 'double' and everything will compile
- Test cases will still work

Magic	Two Worlds	Unique to D	
Generic Pro	ogramming		
<ul> <li>Mathema</li> </ul>	tically, reals are an e	extension of integers	

- int and float both have hardware support
- Replace 'int' with 'double' and everything will compile
- Test cases will still work
- So let's make our code work with any numeric type!

## • The code will compile, but it will be wrong for floats

⊒

э

▲ □ ▶ ▲ ∰ ▶ ▲ Ξ

Magic	Two Worlds	Unique to D	Rounding	Bisection
"Any Num	eric Type" is	s a Bad Idea		
• The cod	le will compile,	but it will be wrong for	floats	

Magic	Two Worlds	Unique to D		
"Any N	umeric Type" is	a Bad Idea		
• The	e code will compile, bi	ut it will be wrong for	floats	

- The problem: Floats are a conjuring trick
- Floats are not a subset of mathematical reals. Floats are not a superset of int.

< D >

Sar

Magic	Two Worlds	Unique to D	Rounding	Bisection
"Any Nur	neric Type" is	a Bad Idea		
• The co	ode will compile, k	out it will be wrong for	floats	
The pr	oblem: Floats are	e a conjuring trick		

• Floats are not a subset of mathematical reals. Floats are not a superset of int.

< 🗆 🕨

• The VALUES are a superset of int

Magic	Two Worlds	Unique to D	Rounding	Bisection
"Any Nu	meric Type" is	a Bad Idea		
The c	ode will compile, b	ut it will be wrong for	floats	
The p	roblem: Floats are	a conjuring trick		

- Floats are not a subset of mathematical reals. Floats are not a superset of int.
- The VALUES are a superset of int
- The SEMANTICS are not

Magic	Two Worlds	Unique to D	Rounding	Bisection
"Any Nur	neric Type" is	a Bad Idea		
The contract	ode will compile, b	out it will be wrong for	floats	
The pr	oblem: Floats are	e a conjuring trick		

- Floats are not a subset of mathematical reals. Floats are not a superset of int.
- The VALUES are a superset of int
- The SEMANTICS are not
- For generic code we need common semantics

nan

Magic	Two Worlds	Unique to D	Rounding	Bisection
Losing Your	Sanity, #2			
<ul> <li>A simple f</li> </ul>	oreach range			
int doTen(T)( {	T from )			

```
int howmany = 0;
foreach (x; from .. from + 10)
++howmany;
return howmany;
```

3 🖌 3

< 🗆 🕨

A ∰ ▶ A ≣ ▶ A

nac

Magic	Two Worlds	Unique to D	Rounding	Bisection
Losing Your	Sanity, #2			
• A simple f	oreach range			
int doTen (T)(	T from )			
{				
int howm	any = 0;			
foreach (	x; from from + 10)	)		
++	howmany;			
return ho	wmany;			
}				
odoTen!floa	t( 500 ) == 10			

Magic	Two Worlds	Unique to D	Rounding	Bisection
Losing Your	Sanity, #2			
• A simple f	oreach range			
int doTen ( T )(	T from )			
{				
int howm	any = 0;			
foreach (	x; from from + 10	)		
++	howmany;			
return ho	wmany;			
}				
<ul> <li>doTen!floa</li> </ul>	at( 500 ) == 10			

odoTen!float( 16777242 ) == 9

2

▲□▶ ▲@▶ ▲ ≧▶ ▲ ≧▶

590

Magic	Two Worlds	Unique to D	Rounding	Bisection
Losing Your	Sanity, #2			
<ul> <li>A simple for</li> </ul>	oreach range			
int doTen (T)(	T from )			
{				
int howm	any = 0;			
foreach (	x; from from + 10)	)		
++	howmany;			
return ho	wmany;			
}				
doTen!floa	at( 500 ) == 10			

< • • •

**∢/型 ) ∢ ⊒ ) ∢** 

- doTen!float( 16777242 ) == 9
- doTen!float( 18000000 ) does not terminate

DAG

3. 3

	Two Worlds	Unique to D	
Increment	(or not)		

For integers, ++x; --x; is a no-op For floats it's more fun

Х	After ++x;x;
31837	31837
1.25e-6	1.20e-6
-1e-20	0
16777250	16777252

< 🗆 🕨

• If you use ++ on a float, someone will go insane.

nan
Magic	Two Worlds	Unique to D	
isNumerio	c() in Phobos		

• All uses of isNumeric() are trivial, except two

DAG

3 🖌 3

< 🗆 🕨

A ∰ ▶ A ≣ ▶ A

Magic	Two Worlds	Unique to D	
isNumeric()	in Phobos		

- All uses of isNumeric() are trivial, except two
- std.complex just casts integers to floating point

< D >

N	lagic	Two Worlds	Unique to D	Rounding	Bisection
i	sNumeric()	in Phobos			
	<ul> <li>All uses of</li> <li>std.completion</li> </ul>	f isNumeric() are tri ex just casts integer	vial, except two s to floating point		
	std.randor	m.dice() is incorrect	for pathological case	es	

∃ 990

▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶

Magic	Two Worlds	Unique to D		
isNumeri	c() in Phobos			
<ul> <li>All use</li> <li>std.con</li> <li>std.rar</li> <li>There integer</li> </ul>	es of isNumeric() a mplex just casts in idom.dice() is inco are probably no m rs and floating poi	are trivial, except two negers to floating po prrect for pathologics nathematical algorith nt	o oint al cases hms that work for b	oth

< □ > < □ > < □ > < □ > < □ > < □ >

Magic	Two Worlds	Unique to D	
"More Pree	cision Is Alv	vays Better"	
More presented in the second secon	ecision improve	es the illusion.	
double magic	( double x )		
{			
return	x + 35 - x;		
}			

▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶

Magic	Two Worlds	Unique to D	
"More Preci	sion Is Always	Better"	
<ul> <li>More prec</li> </ul>	sision improves the	illusion.	
double magic (	double x )		
{			
return x -	+ 35 - x;		
}			
magic(100)	)0000000) == 35		

< □ > < □ > < □ > < □ > < □ > < □ >

	vo Worlds	Unique to D	
"More Precisi	on Is Always I	Better"	
<ul> <li>More precisi</li> </ul>	on improves the il	lusion.	
double magic ( do	ouble x )		
{			
return x + 3	35 - x;		
}			
magic(10000)	000000) == 35		
magic(5e17)	== 64		

< □ > < □ > < □ > < □ > < □ > < □ >

	Two Worlds	Unique to D	
"More Preci	sion Is Always	Better"	
<ul> <li>More prec</li> </ul>	cision improves the	illusion.	
double magic( {	double x )		
return x	+ 35 - x;		
<ul><li>magic(10)</li><li>magic(5e)</li></ul>	0000000) == 35 17) == 64		

• Corner cases move but don't disappear

DAG

э

- ∢ ⊒ →

-∢≣≯

Two Worlds	Rounding	
a 1		

Round	ina	Mod	
liouna	ing i		103

Rounding Mode	2.5	-5.5
Round to Near- est	2	-6
Round Up	3	-5
Round Down	2	-6
Round To Zero	2	-5

▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶
 ▲□▶

Algorithms should be written to work based on the minimum precision of the calculation. They should not degrade or fail if the actual precision is greater. -- The D Spec

Unfortunately this is not generally possible

Sar

Algorithms should be written to work based on the minimum precision of the calculation. They should not degrade or fail if the actual precision is greater. -- The D Spec

- Unfortunately this is not generally possible
- Double rounding is a problem.

Sar

 Magic
 Two Worlds
 Unique to D
 Rounding
 Bisection

 "More Precision Is Always Better"
 Image: Compare the second se

Algorithms should be written to work based on the minimum precision of the calculation. They should not degrade or fail if the actual precision is greater. -- The D Spec

- Unfortunately this is not generally possible
- Double rounding is a problem.
- 3.49 rounds down to 3

 Magic
 Two Worlds
 Unique to D
 Rounding
 Bisection

 "More Precision Is Always Better"
 Image: Compare the sector of the sector of

Algorithms should be written to work based on the minimum precision of the calculation. They should not degrade or fail if the actual precision is greater. -- The D Spec

- Unfortunately this is not generally possible
- Double rounding is a problem.
- 3.49 rounds down to 3
- 3.49 rounds up to 3.5, which rounds up to 4

	Two Worlds	Rounding	
Secret F	Precision		

• Extra hidden precision can happen when:

	Two Worlds	Rounding	
Secret Pr	ecision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines

Sac

< • • •

< A

-

	Two Worlds	Rounding	
Secret F	Precision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)

< D >

Sar

	Two Worlds	Rounding	
Secret Prec	ision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision

Sar

	Two Worlds	Rounding	
Secret Pre	ecision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision

Magic	Two Worlds	Rounding	
Secret Pr	ecision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision
- double has 54 bits. So no rounding happens! We're OK.

Magic	Two Worlds	Rounding	
Secret Pre	cision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision
- double has 54 bits. So no rounding happens! We're OK.
- If we do double calculations at real precision:

Secret Precision	

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision
- double has 54 bits. So no rounding happens! We're OK.
- If we do double calculations at real precision:
- double (54 bits) \* double == 118 bits precision

Magic	Two Worlds	Rounding	
Secret Pr	ecision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision
- double has 54 bits. So no rounding happens! We're OK.
- If we do double calculations at real precision:
- double (54 bits) \* double == 118 bits precision
- real only has 64 bits. We'll round twice.

Magic	Two Worlds	Rounding	
Secret Pr	recision		

- Extra hidden precision can happen when:
- The x87 FPU is used on x86 machines
- Processors support FMA (PPC, recent x86\_64, Itanium...)
- If we do float calculations at double precision
- float (22 bits) \* float == 44 bits precision
- double has 54 bits. So no rounding happens! We're OK.
- If we do double calculations at real precision:
- double (54 bits) \* double == 118 bits precision
- real only has 64 bits. We'll round twice.
- One in 1024 calculations has an out-by-1 error

	Two Worlds	Rounding	
In Practice			

Most library code splits the possible input values into smaller ranges, and then performs a different calculation for each range

< D >

	Two Worlds		Bisection
Root finding	J		

double f( double x ), f( x0 ) > 0, f( x1 ) < 0 find the point where f(x) == 0

• State Of The Art: TOMS 748. Inverse cubic polynomial fitting.

< D >

Sar

	Two Worlds		Bisection
Root finding			

double f( double x ), f( x0 ) > 0, f( x1 ) < 0

find the point where f(x) == 0

- State Of The Art: TOMS 748. Inverse cubic polynomial fitting.
- Every iteration triples number of known bits. Best case 5 calls to f(x)

	Two Worlds		Bisection
Root finding	J		

double f( double x ), f( x0 ) > 0, f( x1 ) < 0

find the point where f(x) == 0

- State Of The Art: TOMS 748. Inverse cubic polynomial fitting.
- Every iteration triples number of known bits. Best case 5 calls to f(x)
- If this fails, use binary chop. Gives one bit per iteration in the worst case.

Magic	Two Worlds		Bisection
Root finding	l i i i i i i i i i i i i i i i i i i i		

double f( double x ), f( x0 ) > 0, f( x1 ) < 0

find the point where f(x) == 0

- State Of The Art: TOMS 748. Inverse cubic polynomial fitting.
- Every iteration triples number of known bits. Best case 5 calls to f(x)
- If this fails, use binary chop. Gives one bit per iteration in the worst case.
- But x => x\*x\*x; takes 1830 calls to converge!

Magic	Two Worlds		Bisection
Root finding	J		

double f( double x ), f( x0 ) > 0, f( x1 ) < 0

find the point where f(x) == 0

- State Of The Art: TOMS 748. Inverse cubic polynomial fitting.
- Every iteration triples number of known bits. Best case 5 calls to f(x)
- If this fails, use binary chop. Gives one bit per iteration in the worst case.
- But x => x\*x\*x; takes 1830 calls to converge!
- With 80-bit reals, worst case is > 16000 calls

Magic	Two Worlds		Bisection
The Bin	ary Chop That	lsn't	

auto midpoint = (x0 + x1)/2;

• Let x0 == 1e100, x1 = 1e-100, and ultimate solution is 2e-100

Sac

< D >

	Two Worlds		Bisection
The Binary	Chop That Isn'	t	

*auto midpoint* = (x0 + x1)/2;

- Let x0 == 1e100, x1 = 1e-100, and ultimate solution is 2e-100
- Midpoints are 5e99, 2.5e99, 1.2e99, 6e98, ...

Image: 0

	Two Worlds		Bisection
The Binary	Chop That Isn'i	t	

*auto midpoint* = (x0 + x1)/2;

- Let x0 == 1e100, x1 = 1e-100, and ultimate solution is 2e-100
- Midpoints are 5e99, 2.5e99, 1.2e99, 6e98, ...
- We get to 2e-100 after 600 iterations

Sar

Magic	Two Worlds		Bisection
Binary C	hop For Real		

Midpoint in implementation space

```
ulong x0_raw = reinterpret!ulong(x0);
```

```
ulong x1_raw = reinterpret!ulong(x1);
```

```
auto midpoint = reinterpret!double( x0_raw + x1_raw ) / 2;
```

< D >

Magic	Two Worlds	Unique to D	Rounding	Bisection
Binary Cl	nop For Real			
<ul> <li>Midpoi</li> </ul>	nt in implementat	ion space		

```
ulong x0_raw = reinterpret!ulong(x0);
ulong x1_raw = reinterpret!ulong(x1);
auto midpoint = reinterpret!double( x0 raw + x1 raw ) / 2;
```

• Again let x0 == 1e100, x1 = 1e-100, and solution is 2e-100

Magi		Two Worlds	Unique to D	Rounding	Bisection
Bi	nary Chor	o For Real			
	<ul> <li>Midpoint i</li> </ul>	n implementation sp	Dace		

```
ulong x0_raw = reinterpret!ulong(x0);
ulong x1_raw = reinterpret!ulong(x1);
auto midpoint = reinterpret!double( x0_raw + x1_raw ) / 2;
```

- Again let x0 == 1e100, x1 = 1e-100, and solution is 2e-100
- Midpoints are 5e0, 2.5e-50, 1.2e-75, 6e-88, 3e-94 ...

Magic	Two Worlds	Unique to D	Rounding	Bisection
Binary Chop	o For Real			
<ul> <li>Midpoint in</li> </ul>	n implementation sp	Dace		

```
ulong x0_raw = reinterpret!ulong(x0);
ulong x1_raw = reinterpret!ulong(x1);
auto midpoint = reinterpret!double( x0_raw + x1_raw ) / 2;
```

- Again let x0 == 1e100, x1 = 1e-100, and solution is 2e-100
- Midpoints are 5e0, 2.5e-50, 1.2e-75, 6e-88, 3e-94 ...
- We reach 2e-100 after 9 iterations
|            | Two Worlds |  | Bisection |
|------------|------------|--|-----------|
| Performanc | e Impact   |  |           |

• For 80 bit reals, worst case improves from 16000 calls, to about 150.

 Sac

< 🗆 🕨

A @ ► A ≡ ►

Magic	Two Worlds		Bisection
Perforr	nance Impact		

- For 80 bit reals, worst case improves from 16000 calls, to about 150.
- TOMS 748 has a similar problem with linear interpolation

< D >

Magic	Two Worlds		Bisection
Perform	ance Impact		

- For 80 bit reals, worst case improves from 16000 calls, to about 150.
- TOMS 748 has a similar problem with linear interpolation
- Fixing that improves the average case as well.

	Two Worlds		Bisection
Performa	ance Impact		

- For 80 bit reals, worst case improves from 16000 calls, to about 150.
- TOMS 748 has a similar problem with linear interpolation
- Fixing that improves the average case as well.
- Available in std.numeric.findRoot

	Two Worlds		Bisection
Moral			

Even when floating point code compiles, and gives the mathematically correct answer, it can still be algorithmically wrong

< <p>I I

	Two Worlds		Bisection
Summary			

## • Floating point is a trick created for engineers, not mathematicians.

⊒

Sac

. . . . . . . .

Magic	Two Worlds		Bisection
Summary			

- Floating point is a trick created for engineers, not mathematicians.
- "Take away the number you first thought of" destroys the illusion

< D >

Magic	Two Worlds	Unique to D	Rounding	Bisection
Summary				

- Floating point is a trick created for engineers, not mathematicians.
- "Take away the number you first thought of" destroys the illusion
- More precision improves the illusion, but corner cases remain

Magic	Iwo Worlds	Unique to D	Bisection
Summary			

- Floating point is a trick created for engineers, not mathematicians.
- "Take away the number you first thought of" destroys the illusion
- More precision improves the illusion, but corner cases remain
- float requires great care. Prefer double or real.

мадіс	Iwo Worlds	Unique to D	Rounding	Bisection
~				
Summary				

- Floating point is a trick created for engineers, not mathematicians.
- "Take away the number you first thought of" destroys the illusion
- More precision improves the illusion, but corner cases remain
- float requires great care. Prefer double or real.
- Use == only when you want to expose implementation details

	Iwo worlds	Unique to D		Bisection
Summary				
Floating	point is a trick	created for engineers,	not mathematicia	ans.

- "Take away the number you first thought of" destroys the illusion
- More precision improves the illusion, but corner cases remain
- float requires great care. Prefer double or real.
- Use == only when you want to expose implementation details
- Generic numeric code is almost certainly wrong in horrible, subtle ways

	Iwo worlds	Unique to D		Bisection
Summary				
Floating	point is a trick	created for engineers,	not mathematicia	ans.

- "Take away the number you first thought of" destroys the illusion
- More precision improves the illusion, but corner cases remain
- float requires great care. Prefer double or real.
- Use == only when you want to expose implementation details
- Generic numeric code is almost certainly wrong in horrible, subtle ways
- D is (mostly) a pleasant language for floating point.

	Two Worlds		Bisection
Questions?			



## www.sociomantic.com

2

▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶
▲□▶