Notes on the replacement of the engine

Although the replacement was made with keeping as much of the version 0.8 behaviour as possible in mind, there are a few points where the new engine behaves different.

Accuracy

The old engine computed higher mathematical functions to 75-150 places precision, depending on the argument and function. This wide range was not really of use, because you got no indication of the size of the error whatsoever. So, to be on the save side, you always had to assume the worst case (75 digits precision), and exceeding this limit was simply a waste of computation effort most of the time.

The new engine has been adjusted to compute all functions independent of the scale of the argument to 78 digits (+3 guard digits). No effort is made to be more precise. The 78 digits limit has been chosen, because (1) it is near the guaranteed precision of the old engine, and (2) 256 bit unsigned integers can be stored without loss of digits. Three extra guard digits usually keep rounding errors from being visible, so (1/3)*3 is shown to 78 places as '1.0' instead of '0.99...'.

The avoidance of overly precise computation alone saves about 50% of computation time. On the other hand, the reduced accuracy is sometimes visible, when handling large arguments (> 1e50). So the trigonometric functions sin, cos and tan finally do not recognize their periodicity any more when an argument lies beyond 1e80, because adding one to its least significant digit covers already more than one cycle (2*pi). Instead of returning a questionable (almost random) result, they yield NaN now. The old engine kept an extra 80 digits with large arguments, so if you somehow could guarantee the validity of these digits, trigonometric functions could be evaluated for arguments even beyond 1e80. This particular extended precision has been sacrificed for the sake of a uniform and time-efficient computation.

Several bug reports made clear, that the old engine's implementation of higher functions was not particularly appropriate for large or tiny arguments. The new engine's implementations have been completely reworked to be stable (where possible*) across all ranges of numbers. In addition, they internally use 5 extra digits (78+5) to prevent rounding errors to effect the first 78 digits. The careful implementation sped up computation again, so in total, on the average, the new engine evaluates higher functions about 9 times faster than the old engine.

-- *Some computations like subtractions with almost full cancellation, or evaluation of periodic functions with arguments much larger than a full cycle, are inherently numerically unstable. No engine can get around that, unless it uses algebra or excessively many digits (and input data is precise to that degree as well). Hopefully, 78 digits will cover most requirements of real world problems, though.

Number range

A change in the internal representation of numbers made it possible to vastly extend the number range. The new engine now supports numbers from 1e-268435456 to 9.99..e+268435455. While this, on the whole, can be seen as a beneficial extension, some pitfalls require attention.

1. While basic operations like + or * easily extend to the enlarged range, this is not true for all operations. Periodical functions like sin cannot be computed to full precision, if arguments are large, and if arguments grow huge, the function simply cannot be computed any more. The same holds for the 'mod' operation, to give another example. Its computation time grows linearly with the difference of the logarithm of the operands. That means that about half a billion computation steps are necessary in the worst case scenario.

All these problems could magically go away if you limit the possible number range to, say, 1e-100 to 1e+100. In this range, all numerically instabilities could be bypassed using sheer computation power.
Thus, to the unwary, a small range calculator can appear more predictable and explainable.
(The new engine, currently, returns 'NaN', if it is requested to perform an unstable operation, or if the computation time exceeds reasonable limits.)

2. The old engine could convert the integral part of numbers to other bases. While it is feasible to display a 150 decimal digit integer in hex in full length, this is not possible with huge numbers any more. So you have to conceive a new way of how to display huge numbers in other bases. Again, this problem does not exist with a small range calculator.
(The new engine displays huge numbers in a scientific hex (oct, bin) format. But this is not something an average user is familiar with.)
3. The old engine's underflows to zero, so you can often continue a computation even after an underflow. The new engine behaves differently, because underflow yields a NaN. This is not really a problem, because the new engine can deal with numbers as tiny as 1e-268435456, so underflow should be a really rare occurrence.
4. While small range numbers can easily be displayed in fix-point format, this is not true for huge or tiny numbers. Numbers near the overflow or underflow limit of the new engine would require millions of digits in fix-point format. This clearly is out of discussion. So the new engine falls back to scientific format, if a fix-point display is not appropriate.

Display formats and rounding

Scientific format reflects the internal representation of numbers in the new engine. The old engine worked more like a fix point format calculator.

It should be noted that some operations like 'round' or 'trunc' have different interpretations in both formats. When a bookkeeper rounds a value like 1234.567 to two places, he or she most likely writes 1234.57 as a result, because he/she operates on financial data. An engineer on the other hand likes to have 1200 instead, because the value most probably represents physical data, and he or she wants to express that this value is valid to 2 places (only).

A special sub mode of scientific format is the engineering format, where the significant may assume values between 1 and 1000, and the exponent is always a multiple of 3. Again, it is not clear, what exactly precision means in this format. The old engine used to count the digits after the decimal point, so the number of displayed digits varied with the magnitude of the value. The new engine, in contrast, always displays the same number of digits as in scientific format, assuming its meaning of precision. So engineering format, compared to scientific format, merely shifts the decimal point and adjusts the exponent appropriately, but does not vary the number of displayed digits. To me, this looks more natural than the old engine's interpretation, so I changed SpeedCrunch's behaviour in that respect.

The native round or trunc functions of the new engine use the 'engineers' point of view. But to comply to the old engine, hmath 'translates' these function calls, so the old behaviour is mostly retained. There are two exceptions: a negative precision parameter rounds to a digit on the left of the decimal point. For example, a call like 'round(1234567.89, -3)' yields 1235000. So if you are interested in the thousands only, this is a good way to accomplish this. Second, 0.5, rounded to an integer, yields 0, not 1. This is conforming to international rounding rules, that say, if the cut off digit is a single 5 only, rounding occurs such that the last digit of the rounded result is even. This eliminates a slight bias against rounding down, that otherwise would be present. (Note: according to this rule, 1.5 is rounded to 2).

It is open to discussion, whether and how the 'engineers view' of rounding can be implemented efficiently.

Zero

Zero is a special number, that neither has a sign nor an exponent. Hence, throughout the engine, zero receives special treatment. Displaying it, is no exception to this rule: Regardless of the activated format and precision, it is always displayed as “0”. The old engine reflected the current settings by allowing “0.00” or “0e0”, so the new engine's in fact presents (in the case of zero) less information. If this turns out to be a problem, a post-processor has to filter output strings, and replace “0” by something else.

Factorial

This function, formerly defined on integers only, has been extended to real numbers. It is possible to evaluate (-2.3)! now. For those familiar with this branch of mathematics, internally the function Γ(x) (read: gamma of x) is used to compute factorials for large or non-integer values. The same holds for the two valued factorial, which is a rephrasing of the falling Pochhammer symbol. Again, the Γ-function (or its residues if poles are involved) is used to compute this function.

For the time being, it is just the (one- or two-valued) factorial that is extended in that way. Other functions based on factorials, like nCr, could be interpolated as well, but applications of that are really rare, so this idea has been shelved.

Singularities

An expression like 0^0 has been defined to be 1 in the old engine. While there are many situations, in which 1 is a good result, there are others, where 1 is definitely wrong. As a matter of fact, the two valued power-function power(x, y) cannot be continued analytically to (0,0), and therefore has a singularity there. The new engine does not try to fill in singularities with (ultimately) arbitrarily chosen function results. It always returns “NaN” in doubtful cases.

testfloatnum

The engine comes with a huge test suite called 'testfloatnum'. After each modification of the engine, you should run this suite to ensure proper function of its basic features. In order to run this suite you have to modify the file 'floatconfig.h' first. Set the define _FLOATNUMTEST and compile the whole suite. Run the compiled testfloatnum on a command line console afterwards. If the suite detects an error, it stops and displays the faulting function, sometimes giving additional information. If not, the last message reads 'all tests PASSED'.

Don't forget to disable the define _FLOATNUMTEST in 'floatconfig.h' after testing. Any SpeedCrunch executable compiled with this setting, will show strange behaviour!

Extensions

The new engine has much more capabilities, none of them are activated in this first version. This is to provide a continuous development, and to let people focus on bug fixing and extending current features first.