Assignment 1: Robustness of the orientation predicate [8 points]

Computational Geometry course at DCGI FEE CTU, winter 2023

This is the first exercise for the Computational Geometry class. Its goal is to get some experience with a non-robust and robust version of geometric predicates. You can also find links to additional interesting material about floating-point arithmetic here.

IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic

Read the third chapter about the numerical precision of the "IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic" [1]. It can be found in the IEEE digital library [2], and it is accessible from within the CTU network. There is also a "light" version, which you can find on Wikipedia [3].

For a deeper understanding of the representation of floating-point numbers, you may find it interesting to play with the Float Converter applet [4] or Online Binary-Decimal Converter [5], or to read the paper by David Goldberg [6]. For details about floating-point calculation support on various platforms, see [8].

Shewchuk predicates

Jonathan Richard Shewchuk implemented fast, robust orientation predicates [7] and released his code for public use. The predicates assume the input float/double parameters are exact numbers and use adaptive precision floating-point arithmetic to compute precise results. The adaptive approach for the sign of determinant computation ("do only as much work as necessary to guarantee a correct result") and an unusual approach to exact arithmetic (splitting the operand into non-overlapping chunks of bits with increasing precision, pioneered by Douglas Priest) are the key reasons why are his algorithms faster than traditional libraries of arbitrary precision numbers.

Compilation & Testing

Download the robustness.zip package for the first exercise and unpack it. Then, open the project robustness.vcxproj and compile it. The program creates a set of *.tga images generated for different data types utilizing different predicates and different exponent values of the delta. View them with your favorite image viewer (IrfanView, ACDSee, Picasa, etc.).

The true computation precision depends on the compiler setting. To force the compiler to use 24-bit precision for floats and 53-bit precision for double, we added `setFPURoundingTo24Bits()` and `setFPURoundingTo53Bits()`.

The tasks

1. [0 pt] **Write two versions of code** for ***naïve*** **floating-point orientation predicate** computation:
   a) using complete evaluation of the **3x3 matrix determinant**
      $\text{orientation}\,(a,b,c) = \text{sign}\big(\,a_x b_y\, +\, b_x c_y\, +\, c_x a_y - a_x c_y -\, b_x a_y - c_x b_y\big)$
   b) using a **2x2 matrix determinant** after subtraction of a pivot – point c as Shewchuk
      $\text{orientation}\,(a,b,c) = \text{sign}\left((a_x - c_x)(b_y - c_y) - (a_y - c_y)(b_x - c_x)\right)$
2. Shewchuk adaptive and exact predicates compute the correct result from given **input** numbers. Therefore, the only source of errors can be the limited precision of the *IEEE 754-2008 float* and *double* data types on the predicate input. Error appears:
   A) When converting the decimal representation of each input point coordinates to binary form,
   B) during computation of the shifted point p from p_0 and $\Delta = k * 2^{exp}$ along the image area.

[3 pts] In datasets 0 to 3, some individual input point coordinates x, y are rounded when stored into an IEEE float or double. For three of these six rounded coordinates, write the absolute rounding error, the value of the *ulp* (Unit in the Last Place) of their binary significand, and the absolute rounding error in *ulps.* For float, then for double. Use the converters [4,5].

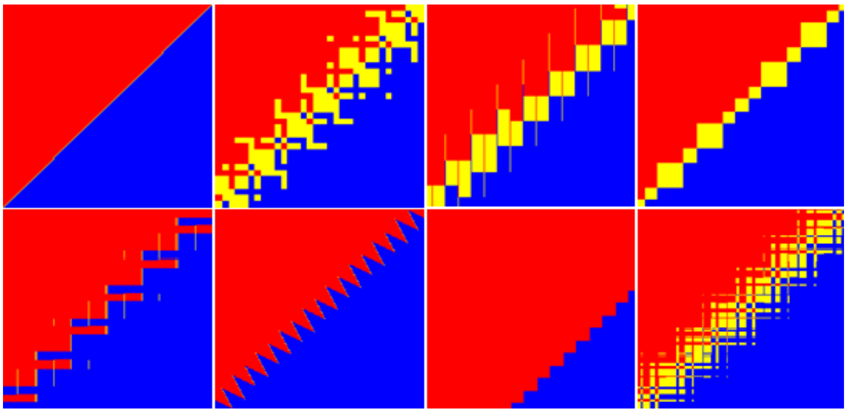
3. In naïve implementation of floating-point orientation predicate 1a) and 1b), 24-bit and 53-bit precision arithmetic introduce additional errors. These errors can be caused by:

   C) The relative size of the operands (input coordinates and computed terms, choice of pivot),
   D) type of arithmetic operations ($+, -, \times$),
   E) order of operations during computation etc.

   - [1 pt] Create a table of small images of all visually different "shapes" (with similar image size and error pattern size to the image below) for all four datasets and three pivots.
   - [3 pts] Choose a single erroneous pixel in one of the cases (omit the cases with simple yellow squares caused by the input quantization) and explore the computation in detail down to the level of individual term computations and their combination.
   - [1pt] Measure the execution times of predicates (do not include other parts) and order the naïve, exact, and adaptive predicates according to the measured execution time.

SUBMIT robustness.cpp + pdf ONLY!!! Write briefly and to the point!

Some examples of generated images



Links

[1] 754-2008 - IEEE Standard for Floating-Point Arithmetic, DOI: 10.1109/IEEESTD.2008.4610935

[2] http://ieeexplore.ieee.org/

[3] IEEE floating point. (2014, October 1). In *Wikipedia, The Free Encyclopedia*. Retrieved 11:17, October 2, 2014, http://en.wikipedia.org/wiki/IEEE_754-2008, or in Czech http://cs.wikipedia.org/wiki/IEEE_754

[4] Harald Schmidt: Float Converter applet, http://www.h-schmidt.net/FloatConverter/IEEE754.html

[5] François Grondin: Online Binary-Decimal Converter. https://www.binaryconvert.com/

[6] David Goldberg: What Every Computer Scientist Should Know About Floating-Point Arithmetic, Computing Surveys, March, 1991, https://www.fer.unizg.hr/_download/repository/paper%5B1%5D.pdf

[7] Jonathan Richard Shewchuk: Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry, 1967, http://www.cs.cmu.edu/~quake/robust.html

[8] Deterministic cross-platform floating point arithmetic. http://christian-seiler.de/projekte/fpmath/