ACM *Sigplan Notices 28*,11 (November 1993), 22-27.

Complex Gaussian Integers for 'Gaussian Graphics'

Henry G. Baker
Nimble Computer Corporation, 16231 Meadow Ridge Way, Encino, CA 91436
(818) 986-1436 (818) 986-1360 (FAX)

Some recent computer languages incorporate rational numbers, complex numbers, and rational complex numbers. We extend these numeric facilities to deal properly with *Gaussian integers*—i.e., complex numbers whose real and imaginary parts are both ordinary (rational) integers. In addition to their intrinsic mathematical interest, such extensions also raise interesting questions regarding polymorphism and multiple inheritance.

Since Gaussian integers are the coordinates of discrete square pixels in the complex plane, complex operations can be used to implement 2-D graphics operations. Many 2-D algorithms are more elegant in complex number form—e.g., one can envision a 2-D spreadsheet for scientific applications whose coordinates are Gaussian integers.

A. INTRODUCTION

The level of abstraction in abstract algebra and abstract data types may have been taken too far—we learn about abstract rings and fields instead of enjoying the beauty of particular rings and fields—e.g., number theory. Once the basic concepts are in hand, we should revel in the polymorphism of Gaussian integers as pairs of integers, as well as atomic elements of the Gaussian ring, much as we revel in the dual role of C "ints" as both integers and bit strings.

Many computer languages—e.g., Fortran-77, APL—incorporate complex numbers as a primitive datatype,[1] and some recent languages—e.g., Common Lisp, Scheme—incorporate rational numbers, as well. Unfortunately, these two datatypes have not been well integrated with each other in these computer languages, although mathematicians have known since the time of Gauss how such an integration may be accomplished by means of *Gaussian integers*—i.e., complex numbers whose real and imaginary parts are both ordinary (rational) integers.

Gaussian integers, in addition to their intrinsic mathematical interest, also serve as an important "test case" for theories of polymorphism in object-oriented programming, since Gaussian integers "inherit" from both the integers and the complex numbers, and since complex rational numbers can be seen either as Q[i] (the rationals extended with the complex number i, i^2=-1, i^4=1), or as the field of quotients of Gaussian integers (ratios of Gaussian integers analogous to the ratios of rational integers found in the field Q of the rational numbers).

We give code for the appropriate functions in Common Lisp, because it already has arbitrary-precision integers, arbitrary-precision rational numbers, and complex rational numbers, and therefore the incremental code required for incorporating Gaussian integers in Common Lisp is quite small relative to most other languages.

We suggest that facilities for Gaussian integers be included in generic complex number standards for arbitrary computer languages [Hodgson91]. Such facilities are in accord with the basic philosophy that if a primitive numeric datatype is provided at all, then *all* of the relevant primitive functions should be extended to deal with this datatype, so long as there is general agreement among mathematicians regarding the meaning of such extensions.

B. PROPERTIES OF THE GAUSSIAN INTEGERS

The Gaussian integers Z[i] are complex numbers in which both the real and imaginary parts are ordinary (or "rational") integers. They form a countable subset of the traditional complex numbers which is closed under addition and multiplication—i.e., the Gaussian integers are a *subring* of the complex numbers. The *units* of this subring are the four Gaussian integers which have multiplicative inverses, namely ±1 and ±i.

The Gaussian integers have a Euclidean greatest common divisor algorithm, and hence they form a Euclidean *principle ideal domain*.[2] They thus have a unique[3] factorization into *Gaussian primes*, which are the numbers: a) ±1±i, b) ordinary (rational) integer primes of the form |p|=4k+3, and c) ±m±ni, ±n±mi, where $m^2+n^2=4k+1=|p|$, p an ordinary (rational) integer prime. Thus, in going from the ordinary (rational) integers to the Gaussian integers, each ordinary (rational) prime becomes either *four* (|p|=2, |p|=4k+3) or *eight* (|p|=4k+1) Gaussian primes, and these are the only Gaussian primes.

[1]As attractive as they are, Mandelbrot pictures are not the only use for complex numbers—e.g., trigonometry without complex numbers gives me a severe case of *sinus*itis (*sinus* is German for *sine*).

[2]An *ideal* is a subset I of a ring R which is closed under subtraction (I-I=I) and multiplication by arbitrary elements of the parent ring (R•I=I). In a principal ideal domain D, every ideal I can be represented as the set of multiples of a single generator element g—i.e., I=D•g. Every ideal of Z[i] is a magnified and tilted version of the square lattice Z[i].

[3]Clearly, any number of units can be introduced into a factorization.

We can now count the number of Gaussian integers which lie on a circle of radius $\sqrt{n}$ around the origin. If any prime of the form $|p|=4k+3$ appears with an *odd* exponent in the (rational) factorization of n, then the circle includes *no* Gaussian integers. Otherwise, let $n=n_0 n_1 n_3$, where n_0 contains all of the powers of 2, n_1 contains all of the primes of the form $|p|=4k+1$, and n_3 contains all of the primes of the form $|p|=4k+3$; the circle of radius $\sqrt{n}$ then contains $4 \bullet d(n_1)$ points, where $d(n_1)$ is the number of (rational) divisors of n_1. For example, there are *no* Gaussian integers on the circle of radius $\sqrt{7}$, but 4(3+1)(2+1)=48 Gaussian integers on the circle of radius $\sqrt{169000}=\sqrt{2^3\ 5^3\ 13^2}$.

Since ordinary (rational) primes of the form $|p|=4k+3$ are also Gaussian primes, the ring of Gaussian integers Z[*i*] modulo p is isomorphic to the finite field $Z_p[x]/(x^2+1)$ of norm(p)=p^2 elements, since the polynomial x^2+1 is irreducible (non-factorable) over Z_p [Lange70]. On the other hand, if $|p|=4k+1$, then the ring of Gaussian integers Z[*i*] modulo m+n*i*, where $m^2+n^2=|p|$, is a finite field of norm(m+n*i*)=|p| elements, which is therefore isomorphic to the finite field Z_p [Lange70], but which folds up its structure in a more interesting 2-dimensional way.

C. GAUSSIAN-INTEGERP

We first need a function which will tell whether we have a Gaussian integer.

```
(defun gaussian-integerp (z)
  (assert (numberp z))
  (and (integerp (realpart z)) (integerp (imagpart z))))
```

D. NORM

The next new function is the `norm` function, which is equal to the square of the usual absolute value function (`abs`), but is more conveniently defined as the product of a number z by its complex conjugate z^*. For some obscure reason, Common Lisp and most other implementations of complex numbers do *not* define the `norm` function, even though it is almost as useful (e.g., it is also *multiplicative*—norm(a•b)=norm(a)•norm(b)) and it is usually less expensive to compute than the `abs` function.[4]

```
(defun norm (z)
  (realpart (* z (conjugate z)))) ; "realpart" needed only for non-Lisp langs.
```

E. FLOOR, CEILING, TRUNCATE, ROUND

We now consider extending to the Gaussian integers (actually, to all complex numbers) the four division functions of Common Lisp which produce integer quotients. While all four functions produce a quotient q and a remainder r from a dividend z and a divisor d, such that $z = q \bullet d + r$ and $0 \le |r| \le |d|$, the `floor` function produces a remainder r of the *same sign as the divisor*, the `ceiling` function produces a remainder of the *opposite sign as the divisor*, the `truncate` function produces a remainder of the *same sign as the dividend*, and the `round` function produces a remainder of the *least absolute value*.

When we divide *complex* numbers, however, we must now worry about the *phase* of the divisor rather than just its *sign*. We know that for a Gaussian integer divisor m+n*i*, we need m^2+n^2 distinct remainders, so the most elegant choice of representative remainders is *a square of area* m^2+n^2 *which is tilted at an angle of atan(n/m)*;[5] equivalently, we consider the remainder fraction r/d to reside in an *upright square* of area 1. The correct quotient for this choice is computed by q=divide(realpart(zd*),dd*)+*i*•divide(imagpart(zd*),dd*),[6] where "divide" is one of `floor`, `ceiling`, `truncate` or `round`, and the correct remainder is computed by r=z-q•d. This q produces the correct remainder even when z,d are both real, so long as divide(z,d)=divide(-z,-d). We give definitions for complex `floor` and `round`; the other two are analogous. The remainders r produced by the `round` function with a complex divisor d *do* satisfy the requirement $0 \le |r| \le |d|$, but the remainders from the other three functions *do not* satisfy this requirement.

```
(defun complex-floor (z d)
  ;;; Should use defmethod on "floor", instead.
  (let* ((dc (conjugate d)) (dn (* d dc)) (zdc (* z dc)))
    (complex (floor (realpart zdc) dn) (floor (imagpart zdc) dn))))
```

---

[4]We *strongly* recommend against extending the relational operations—e.g., <,>,≤,≥—to the complex numbers by means of comparing absolute values or norms, as has been proposed by some in the Lisp and C++ communities. This is because these relational operators would then become *discontinuous* functions across the negative real axis.

[5]This choice of representatives is different from that in [McDonnell73] and [Forkes81], for reasons given below. The edges of these squares would make Bresenham [Newman79] proud.

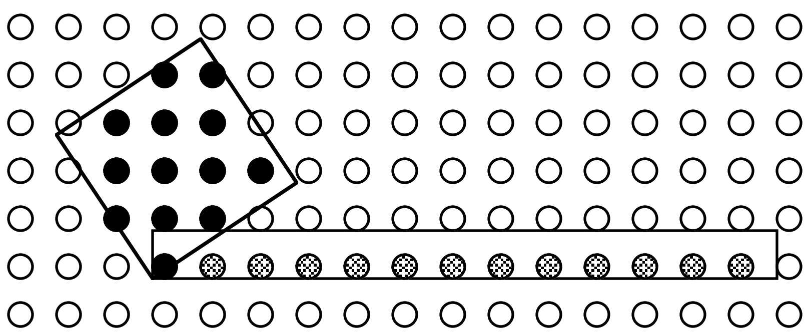
[6]The multiplication of both the dividend and the divisor by a normalization factor (in this case d^*) is reminiscent of Knuth's Algorithm D [Knuth81,4.3.1].


```
(defun complex-round (z d)
  ;;; Should use defmethod on "round", instead.
  (let* ((dc (conjugate d)) (dn (* d dc)) (zdc (* z dc)))
    (complex (round (realpart zdc) dn) (round (imagpart zdc) dn))))
```

These functions do not exhaust the possibilities for Gaussian integer division. Criteria for evaluating the possibilities are the following (see also [McDonnell73] [Forkes81]):

- the set of remainders forms a simple shape[7]
- the set of remainders "tiles" the complex plane in a doubly periodic fashion[8]
- each remainder should have its norm smaller than the norm of the divisor[9]
- the set of remainders is a *convex* set
- division treats real and imaginary parts similarly, but not necessarily independently[10]
- division "works" for expressing numbers with complex bases [Knuth81,4.1][11]
- division "works" for doing continued fractions with Gaussian integer elements [Wall48][12]

For example, consider the Gaussian integer divisor $d=m+ni$, where $\gcd(m,n)=1$. Then $\gcd(n,m^2+n^2)=1=an+b(m^2+n^2)$, so n has a multiplicative inverse $n^{-1}\equiv a \pmod{m^2+n^2}$. Since $m^2+n^2=(m+ni)(m-ni)$, $m+ni\equiv 0 \pmod{m^2+n^2}$, and therefore, $i\equiv -n^{-1}m \pmod{m^2+n^2}$. In other words, i is congruent (mod m^2+n^2) to the rational integer $-n^{-1}m$. Hence, for such a $d=m+ni$, *every* Gaussian integer $z=x+yi$ is congruent to some rational integer j, which can be found by considering $z=x+yi=x+y(-n^{-1}m)\equiv j \pmod{m^2+n^2}$. But choosing the set of representatives j is equivalent to tiling the plane with horizontal rectangles which are 1 unit high and m^2+n^2 units wide [Holladay80].



`floor` remainders (mod $3+2i$) (black) and (mod $(3+2i)(3-2i)=13$) (grey); $i \equiv 5$ (mod $3+2i$).

F. `GCD` AND `LCM`—PUTTING SQUARE PEGS INTO ROUND HOLES

Using the `round` function, we can compute the `gaussian-gcd` function with the Euclidean algorithm. We define `(gaussian-gcd z1 z2)` to be the greatest common divisor computed by this algorithm, up to units. In order to make the greatest common divisor unique, we multiply by a unit so that it always ends up on the non-negative real axis or in the first quadrant.

---

[7]The set of remainders of a complex division by a divisor d need not be convex or even connected—e.g., *any* set S of complex numbers which represents every domain point (D=d•D+S), will work as a set of remainders [Goffinet90].

[8]Each (mod d) tile has area norm(d). However, the individual tiles may have non-convex *fractal* shapes such as the "twindragon" curves produced by bit strings interpreted as numbers of base i-1 [Knuth81,4.1]. Note that Common Lisp's `round` and `truncate` do not produce unique remainders, and therefore do not tile the plane. To tile the plane with *hexagons* instead of squares, utilize the ring $Z[\omega]=Z+Z\omega$ instead of $Z[i]$, where $\omega^3=-1$ [Hardy79]. Square tiles can include only 2 edges and 1 corner; hexagonal tiles can include only 3 edges and 2 corners [Grünbaum87].

[9]*Voronoi* regions around the lattice points give the tiles with the least remainders; in $Z[i]$, these regions are squares.

[10][McDonnell73]'s complex floor function fails this criterion because its remainders form rectangles, not squares.

[11]`ceiling` can be used to encode base i-1 numbers whose digits are the bits 0,1. However, *quater-imaginary* numbers (base $2i$) can't be handled by the usual APL-style *encode* function, since it produces *imaginary* digits—e.g., $1+i$. *Quadtrees* [Samet88] are equivalent to base-2 numbers with the digits 0, 1, i, $1+i$, which are gotten by interlacing the binary digits of the real and imaginary parts of a Gaussian integer.

[12][Hurwitz88] suggests a complex round-like function based on the real function divide(x,d)=floor(x+d/2,d), which minimizes the size of remainders, but—unlike Common Lisp's `round`—produces unique representative remainders.


```
(defun gaussian-gcd1 (z1 z2)
  ;;; Compute gcd using Euclidean algorithm and least abs remainders.
  (let* ((nz1 (norm z1)) (nz2 (norm z2)))
    (if (< nz1 nz2) (gaussian-gcd1 z2 z1)
      (if (= nz2 0) z1
        (gaussian-gcd1 z2 (- z1 (* z2 (complex-round z1 z2))))))))

(defun gaussian-gcd (z1 z2)
  ;;; Should use defmethod on "gcd", instead.
  ;;; Fix up result of gaussian-gcd1 using units.
  (let* ((g (gaussian-gcd1 z1 z2))
         (gr (realpart g)) (gi (imagpart g)))
    (cond ((plusp gr) (if (minusp gi) (* (complex 0 1) g) g))
          ((minusp gr) (if (plusp gi) (* (complex 0 -1) g) (- g)))
          (t (abs gi)))))

(defun gaussian-lcm (z1 z2)
  ;;; Should use defmethod on "lcm", instead.
  (/ (* z1 z2) (gaussian-gcd z1 z2)))
```

The convergence of this gcd algorithm requires that the complex round function fit a *square peg*—the set of remainders $\{x+yi=z-d\bullet\mathrm{round}(z,d)\}$—into a *round hole* $\{x^2+y^2<\mathrm{norm}(d)\}$. Slowly converging Gaussian gcd's can be constructed from a Gaussian Fibonacci sequence: i, $1+i$, $1+2i$, $2+3i$, $3+5i$, ..., F_k+iF_{k+1}, where F_k is the k'th Fibonacci number [Harmon81]. Gaussian Fibonacci numbers have a Fibonacci norm: $\mathrm{norm}(F_k+iF_{k+1})=F_k{}^2+F_{k+1}{}^2=F_{2k+1}$. Furthermore, $F_{2k}{}^2\equiv F_{2k-1}{}^2\equiv -1 \pmod{F_{2k}+1}$, so we have an explicit formula for i: $i\equiv F_{2k} \pmod{F_{2k+1}}$.

G. EVENP AND ODDP

The four numbers $\pm1\pm i$ can all be gotten from $1+i$ by multiplying by the units ±1, $\pm i$; e.g., $1-i = -i\,(1+i)$. Since $2 = (1-i)(1+i)$, the presence of a factor of $1+i$ in an ordinary (rational) integer indicates evenness. Therefore, to extend `evenp` to Gaussian integers, we test for divisibility by $1+i$. Since the "even" and "odd" lattice points tile the Gaussian plane in a chessboard fashion, the predicates `evenp` and `oddp` are equivalent to "whitep" and "blackp".[13]

Divisibility of a number z by $1+i$ is equivalent to divisibility of norm(z) by 2, so we can more efficiently check for the evenness of norm(z). But we can do even better [Knuth81,4.1#28ans]. If $z=m+ni$, then $\mathrm{norm}(m+ni)$ is even if and only if m^2 and n^2 are both odd or both even. But m^2 is even if and only if m is even so $\mathrm{norm}(m+ni)$ is even if and only if m and n are both odd or both even. Furthermore, m+n is even if and only if m and n are both odd or both even. Finally, $z=m+ni$ is even if and only if m+n is even.

```
(defun gaussian-evenp (z)
  ;;; Should use defmethod on "evenp", instead.
  (evenp (+ (realpart z) (imagpart z))))

(defun gaussian-oddp (z)
  ;;; Should use defmethod on "oddp", instead.
  (not (gaussian-evenp z)))
```

H. NUMERATOR AND DENOMINATOR

These two functions return the integral numerator and integral denominator of a fraction which has the same value as the argument, i.e., numerator(m/n)/denominator(m/n) = m/n. Furthermore, the numerator and denominator are in "lowest terms"—i.e., gcd(numerator(m/n),denominator(m/n)) = 1. Finally, the denominator should be standardized (canonicalized), which is achieved in most representations of the rationals by forcing the denominator to be positive.

The numerator and denominator functions can be extended to any complex number whose real and imaginary parts are rational numbers by using the `gaussian-gcd` algorithm given above to reduce the numerator and denominator to lowest terms. We must also canonicalize the denominator to the region of the positive real axis and the first quadrant by multiplying by i, except when the gcd is real. We note that `gaussian-numerator` and `gaussian-denominator` are not trivial functions—e.g.,

`(gaussian-numerator (complex 3/25 -4/25)) = 1` and
`(gaussian-denominator (complex 3/25 -4/25)) = (complex 3 4).`

---

[13] Adding $\pm1\pm i$ preserves parity, proving that chess bishops retain parity. Similarly, the fact that $2+i$ and $2-i$ are distinct primes, and hence $\gcd(2+i,2-i)=1$, proves that chess knights can get to every square on an infinite chessboard.


```
(defun gaussian-numerator (z)
  ;;; Should used defmethod on "numerator", instead.
  (let* ((x (realpart z)) (y (imagpart z))
         (xd (denominator x)) (yd (denominator y))
         (zn (complex (* (numerator x) yd) (* (numerator y) xd)))
         (zd (* xd yd)) (g (gaussian-gcd zn zd)) (r (/ zn g)))
    (if (zerop (imagpart g)) r (* (complex 0 1) r))))

(defun gaussian-denominator (z)
  ;;; Should use defmethod on "denominator", instead.
  (let* ((x (realpart z)) (y (imagpart z))
         (xd (denominator x)) (yd (denominator y))
         (zn (complex (* (numerator x) yd) (* (numerator y) xd)))
         (zd (* xd yd)) (g (gaussian-gcd zn zd)) (r (/ zd g)))
    (if (zerop (imagpart g)) r (* (complex 0 1) r))))
```

I. APPLICATION: GAUSSIAN PRIMES

In the section above on the properties of the Gaussian integers, we indicated that ordinary primes of the form $|p|=4k+1$ can be factored in the Gaussian integers as $p=(m+ni)(m-ni)=(n+mi)(n-mi)$, but we did not provide a method for finding the integers m,n. We now show one method for finding m,n.

Wilson's Theorem tells us that $(p-1)! \equiv -1 \pmod p$,[14] and if $p=4k+1$ then $((2k)!)^2 \equiv -1 \pmod p$,[15] so we have an explicit solution $h = \mathrm{mod}((2k)!,p)$[16] to the equation $h^2 \equiv -1 \pmod p$. Since $h^2+1=(h+i)(h-i)=tp$, for some integer $t<p$, there must be a non-trivial factor $m+ni$ in common between $h+i$ and p, and yet this factor cannot be p itself, because $tp<p^2$. Therefore, $\gcd(h+i,p)=m+ni$ is one of the prime factors of p that we seek [Knuth81,3.3.4#11].[17] The other factor is $m-ni$, and any other factorization of p is produced by including unit factors of the forms ±1 and $\pm i$.

For example, take the prime $53=4\cdot13+1$. $23^2 \equiv -1 \pmod{53}$, or equivalently, $23^2+1 = 10\cdot53$, so $t=10=2\cdot5$ (any odd prime factor q of t with exponent 1 must also be of the form 4k+1, otherwise we have produced a square root of $-1 \pmod q$, which is impossible for primes not of the form 4k+1). Computing the gcd of $23+i$ and 53, we get $2+7i$, so $m=\pm2$, $n=\pm7$ for p=53.

J. APPLICATION: "GAUSSIAN GRAPHICS"

The usual examples of abstract data types place great significance on the *differences* between treating pairs of real numbers as Cartesian coordinates and treating pairs of real numbers as complex numbers. We believe that this approach obscures the real beauty and power of the complex numbers interpreted as coordinates of the complex plane.

Most graphics texts—e.g., [Newman79]—treat the 2-D pixel plane in terms of real vectors rather than complex numbers. While this vector algebra approach—first popularized by Gibbs—extends easily to 3 (and higher) dimensions, it ignores the profoundly different characters of 2-D and 3-D spaces. For example, polynomial equations of even degree with real coefficients may not have any real roots, while such equations of odd degree always have at least one real root; this simple difference means that nontrivial rotations of a "sphere" in 2-D have no fixed points at all, while nontrivial rotations of a sphere in 3-D always have a fixed pole. Physicists and electrical engineers know

---

[14]Proof: take the numbers 1,2,...,p-1 in pairs x,y, $x\neq y$, such that $xy \equiv 1 \pmod p$. The only numbers left over will be 1 and -1, so the entire product will be $\equiv -1 \pmod p$.

[15]Proof: take the numbers 1,2,...,p-1 in pairs x,p-x, such that $x(p-x) \equiv -x^2 \pmod p$. Since p=4k+1, there are an even number (2k) of such pairs, so we can rearrange the product of all of them as $((2k)!)^2$. But this is the same product as in Wilson's theorem, which we have already shown to be $\equiv -1 \pmod p$.

Since squaring (mod p) is easier than computing factorials (mod p), we can more quickly find a quadratic non-residue c, such that $c^{2k}=(c^k)^2\equiv-1 \pmod p$, and thus $h\equiv c^k \pmod p$ [Wagon90].

[16]*Don't* compute the factorial before reducing mod p; *do* reduce mod p for every product [Baker92]; *do* use x-p if $x\geq p/2$.

[17]Gauss gave explicit formulae for m,n: $m \equiv \frac{(2k)!}{2(k!)^2} \pmod p$ and $n \equiv (2k)!\, m \pmod p$, where the m,n have the smallest absolute values [Davenport92]. This method for finding m,n is not at all obvious or as easy to remember as the gcd method.

Legendre used the continued fraction expansion (with a possibly long cycle) of $\sqrt{p}$ to produce m,n [Davenport92].

[Wagon90] shows that the gcd algorithm applied to the rational integers p,h produce m,n as the first two remainders less than $\sqrt{p}$. His method produces exactly the same intermediate results as the Gaussian gcd method, however.

that the wave equation preserves impulses (Huyghen's principle) only for *odd*-dimensional spaces—e.g., the wave equation in 2-D has an infinite impulse response, whereas in 3-D it obeys Huyghen's principle [Courant62]. Therefore, when working in 2-D, one should take advantage of the special characteristics of 2-D and utilize the most powerful representation possible—i.e., the complex number—without worrying about whether it extends to higher dimensions.[18] For example, the additional algebraic structure of the complex multiplicative inverse gives us the Möbius transforms $\frac{Az+B}{Cz+d}$, where A,B,C,D are complex constants, which provide us with elegant uniform representations for lines and circles in the complex plane [Schwerdtfeger62]. This algebraic structure also gives us a complex "interval arithmetic" in which the "intervals" are small circular disks [Gargantini72].

2-D Image Representation

The Gaussian integers are the *coordinates* of individual graphical pixels, and the `round` function extended to the complex numbers computes the center of the square[19] pixel in which a point falls. This property is the major reason for preferring *square* sets of representative remainders from a complex Euclidean division operation.

An interesting coordinate transformation for such a 2-D image is the *log-polar* transform, which takes the pixel at $x+yi$ and maps it to the position $\log(x+yi) = \log|x^2+y^2|+i\ \mathrm{atan}(y/x)$. After such a "texture-mapping" operation, the image can be size- and rotation-normalized by simple translation operations, making image recognition easier.

Contour Representation

The simplest description of a set of pixels which forms a connected contour in the plane is a *chain code*, which gives a sequence of steps from one pixel to the next. The typical chain code utilizes the 8 directions ±1, $\pm i$, $\pm1\pm i$, which can therefore be represented as a sequence of 3-bit codes.

Another elegant way to represent a closed contour on the complex plane (whether it interseects itself or not) is by means of *Fourier descriptors* [Pavlidis77]. If the contour is approximated by means of a list of points $x+yi$ which are (approximately) equally spaced in terms of path length, then a *one*-dimensional Fourier transform of that list provides an equivalent representation, which can be less sensitive to rotation and scaling. Contour shapes can be approximated by keeping only the coefficients for the lower frequencies—e.g., 15 harmonics are sufficient to preserve recognizable shapes for the digits 0-5 [Brill68]. For simple shapes in which the entire boundary is "visible" from the origin—i.e., the boundary is a single-valued function in polar coordinates—the boundary points for the Fourier transform can be chosen at equal angles around the origin.

As an example of a very simple shape, a *circle* can be represented by the parameterization $C_0+R\ e^{i2\pi t}$, where C_0 is the center of the circle, and R is a real number radius—i.e., circles can be represented by only 2 harmonics.

Raster Scanning

The Gaussian integers modulo a particular Gaussian integer form a tilted square array of pixels centered approximately at the origin when the `round` function is used to compute the remainders.[20] (This square can also be thought of as a torus, with the top edge connected to the bottom, and the left edge connected to the right.) If the modulus is a Gaussian prime of the form $m+ni$, $m\neq0$, $n\neq0$, then the rational integers Z, taken modulo $m+ni$, will explore this tilted square array of $|m+ni|^2=\mathrm{norm}(m+ni)$ pixels centered at the origin. This exploration is complete, because every rational or Gaussian integer whose norm is less than norm($m+ni$) is relatively prime to $m+ni$, as $m+ni$ is a Gaussian prime.[21] If n=1, then Z (mod $m+i$) will explore a slightly tilted square in a non-interlaced, horizontal, *raster* pattern!

Consider, for example, the Gaussian prime $10+i$ whose norm is 101. In order to demonstrate a raster scan, we will define the `modulo` function to be used for computing (mod $10+i$) as follows:

---

[18]Hamilton's *quaternions* are elegant extensions of the complex numbers useful in 3-D and 4-D, but multiplication isn't commutative [Altmann86]. Quaternions are most conveniently manipulated as pairs of complex numbers.

[19]Gaussian graphical algorithms work best with *square* pixels, but then so do most graphics algorithms.

[20]If another division function is used, the shape of this array of pixels may be changed.

[21]That this scan of the square "fills the space" of pixels should not be surprising. König's corollary to Kronecker's Theorem [Hardy79] tells us that a reflected ray which makes an angle—whose tangent is irrational—to the side of a square with reflecting sides will densely and uniformly explore the interior of the square. In the present case of a rational tangent n/m, and a square of area m^2+n^2, an initially horizontal ray will be uniformly periodic with only integer-valued horizontal lines. There will also be a ruling of non-horizontal lines at angle $2\bullet\mathrm{atan}(n/m)$, which we conjecture will never intersect the lattice points inside the square. Thus, our fundamental square can be considered to be a "folded-up" version of the horizontal linear strip of height 1 and length m^2+n^2.


```
(defun modulo (z d)
  ;;; Modulo function for doing raster scans.
  ;;; This modulo function won't work for computing gcd's.
  (- z (* d (complex-floor z d))))
```

The above `modulo` function does *not* produce a result whose norm is smaller than the divisor, but it does "tile the plane". Each of these tiles, when considered as a pattern of discrete pixels, forms an L-shaped tile consisting of a 10x10 square above a 1x1 square, as might be expected from the fact that $101=10^2+1^2$. This familiar tiling pattern repeats on an angle of atan(1/10).[22]

If we now compute 0 (mod 10+*i*), 1 (mod 10+*i*), 2 (mod 10+*i*), 3 (mod 10+*i*), ..., 100 (mod 10+*i*), then we get the following "inverse table", where the number at position x+y*i* indicates the integer 0-100 that maps to x+y*i*.

10*i*	1	2	3	4	5	6	7	8	9	10
9*i*	11	12	13	14	15	16	17	18	19	20
8*i*	21	22	23	24	25	26	27	28	29	30
7*i*	31	32	33	34	35	36	37	38	39	40
6*i*	41	42	43	44	45	46	47	48	49	50
5*i*	51	52	53	54	55	56	57	58	59	60
4*i*	61	62	63	64	65	66	67	68	69	70
3*i*	71	72	73	74	75	76	77	78	79	80
2*i*	81	82	83	84	85	86	87	88	89	90
1*i*	91	92	93	94	95	96	97	98	99	100
0*i*	0									
	0	1	2	3	4	5	6	7	8	9

Table of n (mod 10+*i*), for n = 0 (1) 100, showing raster scan pattern.

If we start counting with 1, then our raster scan will start at the top left corner (0+10*i*) and sweep out successive rows of pixels on our Gaussian screen whose upper left hand corner is 0+10*i* and whose lower right hand corner is 9+*i*. The number 101 maps to the same pixel as the number 0—i.e., the pixel 0+0*i*, which acts as a kind of "vertical retrace" pixel. Even more interestingly, we can do a *vertical* raster scan by using the series 0*i*, 1*i*, 2*i*, 3*i*, ..., etc. (mod 10+*i*)—i.e., the series Z*i*. In fact, we can produce almost any conceivable scan of this "square" by considering the series Zk, for any Gaussian integer k, and scans without a "retrace pixel" can be made by considering the *powers* of primitive elements (see [Brenner73] and [Baker74]).

Gaussian primes of the form m+*i* appear to be moderately common—e.g., there are about 50 whose norms are less than 100,000. Particularly interesting are the Gaussian factors of *Fermat primes* of the form $2^{2^n}+1$. For example, the Fermat prime 65,537 splits into Gaussian factors 256±*i*, which gives us a 256x256 Gaussian graphics screen. Unfortunately, there may be only 5 Fermat primes, of which 65,537 is the largest known, and it is conjectured that these 5 are all that exist [Hardy79]. It is not known if there are more than a finite number of primes of the form m^2+1 [Hardy79], although—unlike Fermat primes—these primes are empirically rather easy to find.

For example, each of the following integers m produces a prime $p=m^2+1$.

2,4,6,10,14,16,20,24,26,36,40,54,56,66,74,84,90,94
110,116,120,124,126,130,134,146,150,156,160,170,176,180,184
204,206,210,224,230,236,240,250,256,260,264,270,280,284,300,306,314,326,340,350,384,386,396
400,406,420,430,436,440,444,464,466,470,474,490,496,536,544,556,570,576,584,594
634,636,644,646,654,674,680,686,690,696,700,704,714,716,740,750,760,764,780,784
816,826,860,864,890,906,910,920,930,936,946,950,960,966,986
1004,1010,1036,1054,1060,1066,1070,1080,1094,1096.

Affine Transformations

The standard graphics textbooks utilize homogeneous coordinates to implement affine transformations in the plane [Newman79] [Pavlidis82]. Homogeneous coordinates come from projective geometry which shows that translations

---

[22] Rational primes like $p=m^2+n^2=\mathrm{norm}(m+ni)$, m>n>1, give rise to tilings with a greater tilt.

can be seen in projective space as rotations around a point at infinity, and therefore both rotations and translations can be handled by the same (cumbersome and inefficient) mechanism.

$$\begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax+by+e \\ cx+dy+f \\ 1 \end{bmatrix}$$

Affine transformations in the complex plane can also be accomplished by means of complex arithmetic:

affine(z) = A•z + B•z* + C, where A, B, C are complex constants and z* = conjugate(z).

This general form is preserved when composing affine transformations, as can be readily checked with simple algebra. That this general form can accomplish any affine transformation can be seen by converting the homogeneous matrix form of affine transformation into the complex form. If a,b,c,d,e,f are the parameters of the transformation matrix, as above, then we can express the complex constants A,B,C in terms of a,b,c,d,e,f:

$$A = \frac{(a+d) - (b-c)\, i}{2}, \quad B = \frac{(a-d) + (b+c)\, i}{2}, \quad C = e + f\, i$$

Unlike the case with homogeneous matrices, the complex form of an affine transformation is particularly perspicuous when it comes to *rotations*. If we have a transformation which is just a *proper* rotation (i.e., without a flip), then $A=e^{i\theta}$, where θ is the angle of rotation, and B=C=0. If we have an *improper* transformation which involves a flip, then $B=e^{i\theta}$, where θ is the angle of rotation, and A=C=0. Thus, for proper Euclidean transformations, |A|=1 and B=0.

Complex CORDIC Rotations

The Gaussian integers $2^k \pm i$ offer an elegant and efficient way to rotate on a binary computer. We can represent any rotation angle θ in the range [-90°,90°] in a mixed-radix positional number system [Knuth81] in which the digits d_k are ±1 and the weights α_k are $\mathrm{atan}(2^{-k})$ [Volder59]—i.e.,

$$\theta = \sum_k^\infty d_k \bullet \alpha_k = \sum_k^\infty d_k \bullet \mathrm{atan}(2^{-k}).$$

Equivalently, we can express a rotation $e^{i\theta}$ as an infinite product of complex factors —i.e.,

$$e^{i\theta} = \prod_k^\infty \frac{(2^k+i)^{d_k}}{|(2^k+i)^{d_k}|} = \prod_k^\infty \frac{2^k+d_k \bullet i}{\sqrt{2^{2k}+1}} = \frac{1}{K} \bullet \prod_k^\infty (1+d_k \bullet 2^{-k} \bullet i),$$

$$\text{where the factor } K = \sqrt{\prod_k^\infty (1+2^{-2k})} \approx 1.64676$$

is a normalization constant[23] independent of θ. Therefore, any rotation can be factored into a series of additions and shifts (followed by a single normalization) in a manner reminiscent of the decomposition of normal multiplication into additions and shifts. The binary digits $d_k = \pm 1$ are found by a binary search of a virtual table of arctangents of powers of 2. Only 1 bit needs to be stored for each factor in the product, thus providing a compact code for rotations.

Halftoning

Halftoning is a standard technique for trading spatial resolution of a 2-D display for grey-scale resolution. Traditional halftoning algorithms coalesce a fixed number of binary pixels together to display one grey-scale pixel. Gaussian integers provide an elegant way to do halftoning with square groups (!) of binary pixels. If l+1 levels of grey must be represented, then l binary pixels are required within each group; j of them will be turned on to achieve a brightness level of j/l.[24] If l is representable as a sum of 2 squares—i.e., $l=m^2+n^2$—and gcd(m,n)=1, then the Gaussian integers (mod m+ni) will form a tilted square which tiles the plane and can be used as the "unit cell" for halftoning. Furthermore, the set of integers [0,l) has a 1-1 mapping onto the tilted square (as we have shown above for 3+2i), and can therefore also act as a basic repeat pattern which lies within a single scan line [Holladay80]. Interestingly, Gaussian halftones[25] can be done only if l has no prime factors |p|=4k+3 which have odd exponents; this makes it difficult to provide a number of grey levels which is exactly a power of two.

---

[23]The product inside the radical is a generating function evaluated at 1/4 for distinct partitions of integers [Hardy79]. W. Gosper can accelerate convergence but is not aware of any simple "closed-form" representation for this constant.

[24]We will not discuss a mechanism for choosing the particular pixels to be turned on in each group.

[25]Gaussian bells that ring accidentally!

Raster Frame Buffer Memory Design[26]

The design of memory systems for high performance computers has already benefitted from number theory. The Burroughts Scientific Processor utilized a Fermat prime ($2^4+1=17$) number of memory banks to minimize memory contention in vector operations [Kuck78].

It should be no surprise, then, that the design of raster graphics 2-D frame buffers can benefit from the properties of the Gaussian integers. Modern 2-D frame buffers must be designed in such a way that groups of adjacent pixels in a single scan line can be accessed from the frame buffer in a single memory cycle to refresh the display, which implies that each pixel in the group must be stored in a separate memory chip. However, most graphics algorithms access the frame buffer in patterns which have locality in *two* dimensions, which implies that each pixel in a small 2-D neighborhood should be stored in a separate memory chip. Both of these properties are elegantly satisfied by using a Gaussian modulo function for the mapping of 2-D coordinates to memory chips—e.g., chip#(x,y) = mod(x+yi,m+ni), where m,n are rational integer constants, and gcd(m,n)=1. For example, if m+ni=3+2i, then $3^2+2^2=13$ memory chips are required, and the two access patterns are shown in the figure of section E, above. Furthermore, access to *any* square of the fundamental size and orientation—regardless of its translation—can be performed in a single memory cycle, and these pixels can then be permuted to standard positions using a cyclic *barrel shifter*. In the above example, consider accessing the tilted square whose bottom corner is located at 1+2i. Then the fundamental slice is cyclically shifted left by $1+2i\equiv1+2\bullet5=11$, since $i\equiv5$ (mod $3+2i$).

Gaussian integer constants of the form 2^k+i are particularly appropriate for such memory systems. These moduli involve computing mod(x,2^{2k}+1), which can be done by the binary equivalent of "casting out elevens"—i.e., trivially express x in base-2^{2k} notation, and then alternately add and subtract each big digit. This procedure works because $(2^{2k})^j \equiv (-1)^j \equiv \pm1 \pmod{2^{2k}+1}$.

[Chor86] effectively suggests the use of Fibonacci Gaussian integer constants F_k+iF_{k+1} for raster graphics memory systems, because Fibonacci systems provide interference-free access to rectangles of the largest area.

Memory *cache designs* can be optimized for 2-D access by means of these mapping techniques. Even with no special hardware, software mappings of this kind can optimize existing caches. Cache "lines" then become cache "squares"!

Pythagorean Chip Dip

Consider integrated circuit chips with two different layers, both of which utilize square meshes of lines. If the ratio of the areas of these squares is m^2+n^2, gcd(m,n)=1, and the layers are deposited with an angle of atan(n/m), then the larger squares will look like the picture in section E.[27] If the lines of the larger squares are "reflective", then one can conceive of utilizing the "space-filling" nature of the internal reflections to address the internal intersection points. These internal intersection points can be consider on a kind of time-multiplexed "ring-bus". The internal intersections could either be amplifiers to create a kind of folded-up laser, or could be taps in a folded-up filter. The addressibility feature could also be used as a kind of switching network, and if m^2+n^2 is quite large, then a Pythagorean chip could be used as the basis for a flat-panel display of pixels. More than two layers with distinct pitches and angles can also be envisioned.

K. CONCLUSIONS

We have shown how the numeric functions of modern computer languages can be extended to deal correctly with Gaussian integers. Since these extensions are so simple, once the basic datatypes have been implemented, it makes sense to include these capabilities in any standard for complex numbers in computer languages.

We prefer the use of tilted *squares* for representing the remainders of complex division, rather than the rectangles of APL. Square residue sets seem to offer the most elegant utilization of the two real parameters (real and imaginary parts) implicit in a complex divisor—as defining the size and orientation of the square. Different division functions are useful in different contexts, however, so we recommend that a multiplicity of division functions be provided.[28]

If one prefers *hexagonal* pixels to square pixels, this paper can be rewritten to uniformly utilize the ring Z[ω]=Z+Zω instead of the Gaussian ring Z[i], where ω is a primitive root of unity—i.e., $\omega^6=1$ ($\omega=(1\pm i\sqrt{3})/2$) instead of $i^4=1$. The Euclidean gcd algorithm works in this ring with norm(m+nω)=m^2+mn+n^2. The basic lattice is equilateral triangular, and the remainders can be made to tile the plane with *hexagons* [Hardy79]. Such a tiling is ideal for the RGB triads of a color television, because the lattice points (pixels) then fall into exactly *three* classes— $\equiv0,\pm1$ (mod $2\omega-1=i\sqrt{3}$).

---

[26]These ideas are from [Holladay80] and [Chor86], which could both benefit handsomely from Gaussian integers.

[27]We assume that pitches and angles can be tightly controlled during fabrication, but not necessarily registration. However, registration in a Pythagorean chip should not matter very much!

[28]To paraphrase Emerson, "foolish consistency is the hobgoblin of little languages".

L. REFERENCES

Altmann, S.L. *Rotations, Quaternions, and Double Groups*. Clarendon Press, Oxford, 1986.

Baker, H.G. "On the Permutations of a Vector Obtainable through the Restructure and Transpose Operators of APL". 1974, published recently in *APL Quote Quad 23*,2 (Dec. 1992), 27-31.

Baker, H.G. "Computing A*B (mod N) Efficiently in ANSI C". *Sigplan Not. 27*,1 (Jan. 1992), 95-98.

Beeler, M., *et al*. "HAKMEM". MIT AI Memo 239, Feb. 29, 1972. See items #107,136-138.

Bracewell, R.N. *The Fourier Transform and Its Applications, 2nd. Ed*. McGraw-Hill, NY 1986.

Brenner, N. "Algorithm 467: Matrix transpose in place". *CACM 16*,11 (1973), 692-694.

Brill, E.L. "Character Recognition via Fourier Descriptors". *Proc. WESCON* Paper 25/3, Los Angeles, 1968.

Chor, B., *et al*. "An Application of Number Theory to the Organization of Raster-Graphics Memory". *JACM 33*,1 (Jan. 1986), 86-104.

Courant, R., and Hilbert, D. *Methods of Mathematical Physics, Vol. II*. Wiley & Sons, New York, 1962.

Davenport, H. *The Higher Arithmetic: An Introduction to the Theory of Numbers, 6th Ed*. Camb. U. Press, 1992.

Diaz, B.M., and Bell, S.B.M., *eds*. *Spatial Data Processing using Tesseral Methods*. Nat. Envir. Res. Council, Swindon, 1986.

Duprat, J., *et al*. "New Redundant Representations of Complex Numbers and Vectors". *IEEE Trans. Computers 42*,7 (July 1993), 817-824.

Eggleton, R.B., *et al*. "Euclidean Quadratic Fields". *AMM 99*,9 (Nov. 1992), 829-837.

Forkes, D. "Complex Floor Revisited". *APL'81*, ACM *APL Quote Quad 12*,1 (Sept. 1981), 107-111.

Gargantini, I., & Henrici, P. "Circular Arithmetic and the Determination of Polynomial Zeros". *Numer. Math 18* (1972), 305-320.

Goffinet, D. "Number Systems with a Complex Base: a Fractal Tool for Teaching Topology". *AMM 98*,3 (Mar. 1991), 249-255.

Grünbaum, B., and Shepard, G.C. *Tilings and Patterns*. W.H.Freeman & Co., New York, 1987.

Hardy, G.H., and Wright, E.M. *An Introduction to the Theory of Numbers, 5th Ed*. Clarendon Press, Oxford, 1979.

Harmon, C.J. "Complex Fibonacci Numbers". *Fibonacci Quart. 19*,1 (Feb. 1981), 82-86.

Hodgson, G.S. "Rationale for the Proposed Standard for Packages of Real and Complex Type Declarations and Basic Operations for Ada". ACM *Ada Letters XI*, 7 (Fall 1991), 131-139.

Holladay, T.M. "An Optimum Algorithm for Halftone Generation for Displays and Hard Copies". *Proc. Soc. Info. Disp. 21*,2 (1980), 185-192.

Hurwitz, A. "Uber die Entwicklung Complexer Grossen in Kettenbruche". *Acta Math. 11* (1888).

Knuth, D.E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 2nd Ed*. Addison-Wesley 1981.

Kota, K., and Cavallaro, J.R. "Numerical Accuracy and Hardware Tradeoffs for CORDIC Arithmetic for Special-Purpose Processors". *IEEE Trans. Computers 42*,7 (July 1993), 769-779.

Kuck, D.J. *The Structure of Computers and Computations, Vol. I*. John Wiley & Sons, New York, 1978.

Lang, Serge. *Algebra*. Addison-Wesley, Reading, MA 1970.

McDonnell, E.E. "Complex Floor". *Proc. APL Congress 73*, North-Holland, Amsterdam, 1973.

McIlroy, M.D. "Best Approximate Circles on Integer Grids". ACM *Trans. Graphics 2*,4 (Oct. 1983), 237-263.

Middleditch, A.E. "Intersection Algorithms for Lines and Circles". ACM *Trans. Graphics 8*,1 (Jan. 1989), 25-40.

Newman, W.M., and Sproull, R.F. *Principles of Interactive Computer Graphics*. McGraw-Hill, New York, 1979.

Ore, Oystein. *Number Theory and its History*. McGraw-Hill, New York, 1948.

Pavlidis, T. *Structural Pattern Recognition*. Springer-Verlag, Berlin 1977.

Pavlidis, Theo. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.

Pavlidis, T. "Curve Fitting with Conic Splines". ACM *Trans. Graphics 2*,1 (Jan. 1983), 1-31.

Penfield, P. "Principle Values and Branch Cuts in APL". *Proc. APL81, APL Quote Quad 12*,1 (Sept. 1981).

Penney, W. "A 'Binary' System for Complex Numbers". *JACM 12*,2 (Apr. 1965), 247-248.

Rokne, J.G., *et al*. "Fast Line Scan-Conversion". ACM *Trans. Graphics 9*,4 (Oct. 1990), 376-388.

Samet, H. "Hierarchical Representations of Collections of Small Rectangles". ACM *Comput. Surv. 20*,4 (Dec. 1988),271-309.

Schwerdtfeger, Hans. *Geometry of Complex Numbers: Circle Geometry, Moebius Transformation, Non-Euclidean Geometry*. Dover Publs., New York, 1962.

Shallit, J.O. *Integer Functions and Continued Fractions*. Princeton Univ., 1979.

Steele, Guy L. *Common Lisp, the Language, 2nd Ed*. Digital Press, Bedford, MA, 1990, 1029p.

Stein, S.K. "Algebraic Tiling". *Amer. Math. Monthly 81* (1974), 445-462.

Turkowski, K. "Anti-Aliasing through the Use of Coordinate Transformations". ACM *Trans Graph 1*,3 (1982), 215.

Volder, J.E. "The CORDIC Trigonometric Computing Technique". *IRE Trans. Elect. Comps. EC-8* (Sept. 1959).

Wagon, S. "The Euclidean Algorithm Strikes Again". *AMM 97*,2 (Feb. 1990), 125-129.

Wall, H.S. *Analytic Theory of Continued Fractions*. Chelsea Publ. Co., NY, 1948, 1973.

