

WG14/N311

X3J11/93-058

Numerical Extensions to ANSI C in the C^H Programming Language

X3J11.1/93

November 30, 1993

Harry H. Cheng

**Department of Mechanical and Aeronautical Engineering
University of California
Davis, CA 95616**

Program Real Numbers in the Entire Real Domain with NaN, ± 0.0 , and $\pm \text{Inf}$

Deliver a Consistent Correct Numerical Value or NaN

For example, because $\lim_{x \rightarrow 0} x^x = 0^0 = 1$ and $\lim_{x \rightarrow 0} x^{\frac{1}{\log(x)}} = 0^0 = e$, `pow(0.0,0.0)` delivers NaN

Polymorphic Functions

`pow(2,3) == 8`; `pow(2.0F, 3.0F) == 8.0F`; `pow(2.0D, 3.0F) == 8.0D`; `pow(complex(2,0), 3) == complex(8.0, 0)`, `pow(dual(2,0), 3) == dual(8.0, 0)`

Add Binary Integral Constants

Binary constants: `0b11110`, `0B11110`

Octal constants: `036`

Decimal constants: `30`

Hexadecimal constants: `0x1e`, `0X1e`, `0X1E`

Add Binary Format Specifier

`printf("3 = (%b)2, 3 = (%8b)2\n", 3, 3);`

output: `3 = (11)2, 3 = (00000011)2`

Add Double Constants

double constants: `3.4D`, `3.4e9D`, `3.4E9d`

Default constants such as `2.3` are float, which can be switched to double by the compiler option or by the function `floatconst(on_off)` for interpretive implementation.

Add Exclusive-or Operator

exclusive-or `^^`

Increment and Decrement Operation

```
++++i      i = i+2
i++++      i = i+2
i++++++i   i = i+4
----i      i = i-2
i-----    i = i-2
```

Bitwise Shifting Operation

For lvalue << rvalue, if rvalue is negative, reverse the shifting direction.

Functional Type Conversion Operation

```
i = int(9.5);
f = float(9);
z = complex(i, f);
z = complex(3.5F, 5.3D);
```

Relational Operations

```
NaN == NaN gives 1
NaN != NaN gives 0
Inf == Inf gives 1
Inf != Inf gives 0
ComplexNaN == ComplexNaN gives 1
ComplexNaN != ComplexNaN gives 0
ComplexInf == ComplexInf gives 1
ComplexInf != ComplexInf gives 0
```

If $x == y$, $(x - y) == -(y - x)$ is TRUE even for NaN, ± 0.0 , $\pm \text{Inf}$.

Single Delimiter Comment

```
i = complex(1,2);           /* single delimiter comment
```

Alternative Solutions to Restricted Pointers for Anti-Aliasing

1. Add do-loop.
2. Add assumed-shaped arrays, only arrays can be passed to assumed-shape arrays.
3. Add nested-functions that modularize the program and ease the detection of aliasing.
4. Add array syntax without aliasing.

Declaration of Complex Variables

```
complex z1, zz1[10], *zp1, **zpp1, *zpa1[10];
double complex z2, zz2[10], *zp2, **zpp2, *zpa2[10];
long double complex z3, z3[10], *zp3, **zpp3, *zpa3[10];
```

Complex Constructor

```
complex z1;
double complex z2;
z1 = complex(1, 2);    \# complex = complex
z1 = complex(1F, 2D);  \# complex = double complex
z2 = complex(1F, 2);    \# double complex = complex
z2 = complex(1D, 2D);  \# double complex = double complex
```

I/O for Complex Numbers

```
complex z1, z2, *zptr;
zptr = &z2;           \# zptr points to z2's memory
printf("Please type in four numbers for two complexes \n");
scanf(&z1, zptr);
printf("The first complex number is ", z1, "\n");
printf("The second complex number is %f \n", z2);
```

The result of the interactive execution of the above program is as follows

Please type in four numbers for two complex

1 2.0 3.0 4

The first complex number is complex(1.000,2.000)

The second complex number is complex(3.000000,4.000000)

Table 1: Complex operations

Definition	C ^H Syntax	C ^H Semantics
negation	$-z$	$-x - iy$
addition	$z1 + z2$	$(x_1 + x_2) + i(y_1 + y_2)$
subtraction	$z1 - z2$	$(x_1 - x_2) + i(y_1 - y_2)$
multiplication	$z1 * z2$	$(x_1 * x_2 - y_1 * y_2) + i(y_1 * x_2 + x_1 * y_2)$
division	$z1 / z2$	$\frac{x_1 * x_2 + y_1 * y_2}{x_2^2 + y_2^2} + i \frac{y_1 * x_2 - x_1 * y_2}{x_2^2 + y_2^2}$
equal	$z1 == z2$	$x_1 == x_2 \text{ and } y_1 == y_2$
not equal	$z1 != z2$	$x_1 != x_2 \text{ or } y_1 != y_2$

Table 2: The syntax and semantics of built-in complex functions.

C ^H Syntax	C ^H Semantics
sizeof(<i>z</i>)	8
abs(<i>z</i>)	$\sqrt{x^2 + y^2}$
real(<i>z</i>)	<i>x</i>
imaginary(<i>z</i>)	<i>y</i>
complex(<i>x</i> , <i>y</i>)	$x + iy$
conjugate(<i>z</i>)	$x - iy$
polar(<i>z</i>)	$\sqrt{x^2 + y^2} + i\Theta$; $\Theta = \text{atan2}(y, x)$
polar(<i>r</i> , <i>theta</i>)	$r \cos(\text{theta}) + ir \sin(\text{theta})$
sqrt(<i>z</i>)	$\sqrt{\sqrt{x^2 + y^2}}(\cos \frac{\Theta}{2} + i \sin \frac{\Theta}{2})$; $\Theta = \text{atan2}(y, x)$
sqrt(<i>z</i> , <i>k</i>)	$\sqrt{\sqrt{x^2 + y^2}}(\cos \frac{\Theta + 2k\pi}{2} + i \sin \frac{\Theta + 2k\pi}{2})$; $\Theta = \text{atan2}(y, x)$
exp(<i>z</i>)	$e^x(\cos y + i \sin y)$
log(<i>z</i>)	$\log(\sqrt{x^2 + y^2}) + i\Theta$; $\Theta = \text{atan2}(y, x)$
log(<i>z</i> , <i>k</i>)	$\log(\sqrt{x^2 + y^2}) + i(\Theta + 2k\pi)$; $\Theta = \text{atan2}(y, x)$
log10(<i>z</i>)	$\frac{\log(z)}{\log(10)}$
log10(<i>z</i> , <i>k</i>)	$\frac{\log(z, k)}{\log(10)}$
pow(<i>z</i> ₁ , <i>z</i> ₂)	$z_1^{z_2} = e^{z_2 \ln z_1} = \exp(z_2 * \log(z_1))$
pow(<i>z</i> ₁ , <i>z</i> ₂ , <i>k</i>)	$z_1^{z_2} = e^{z_2 \ln z_1} = \exp(z_2 * \log(z_1, k))$
ceil(<i>z</i>)	$\text{ceil}(x) + i \text{ceil}(y)$
floor(<i>z</i>)	$\text{floor}(x) + i \text{floor}(y)$
fmod(<i>z</i> ₁ , <i>z</i> ₂)	z ; $\frac{z_1}{z_2} = k + \frac{z}{z_2}$, $k \geq 0$
modf(<i>z</i> ₁ , & <i>z</i> ₂)	$\text{modf}(x_1, \&x_2) + i \text{modf}(y_1, \&y_2)$
frexp(<i>z</i> ₁ , & <i>z</i> ₂)	$\text{frexp}(x_1, \&x_2) + i \text{frexp}(y_1, \&y_2)$
ldexp(<i>z</i> ₁ , <i>z</i> ₂)	$\text{ldexp}(x_1, x_2) + i \text{ldexp}(y_1, y_2)$
sin(<i>z</i>)	$\sin x \cosh y + i \cos x \sinh y$
cos(<i>z</i>)	$\cos x \cosh y - i \sin x \sinh y$
tan(<i>z</i>)	$\frac{\sin z}{\cos z}$
asin(<i>z</i>)	$-i \log(iz + \sqrt{1 - z^2})$
asin(<i>z</i> , <i>k</i>)	$-i \log(iz + \sqrt{1 - z^2}, k)$
asin(<i>z</i> , <i>k</i> ₁ , <i>k</i> ₂)	$-i \log(iz + \sqrt{1 - z^2}, k_1), k_2)$
acos(<i>z</i>)	$-i \log(z + i \sqrt{1 - z^2})$
acos(<i>z</i> , <i>k</i>)	$-i \log(z + i \sqrt{1 - z^2}, k)$
acos(<i>z</i> , <i>k</i> ₁ , <i>k</i> ₂)	$-i \log(z + i \sqrt{1 - z^2}, k_1), k_2)$
atan(<i>z</i>)	$\frac{1}{2i} \log\left(\frac{1 + iz}{1 - iz}\right)$
atan(<i>z</i> , <i>k</i>)	$\frac{1}{2i} \log\left(\frac{1 + iz}{1 - iz}, k\right)$
atan2(<i>z</i> ₁ , <i>z</i> ₂)	$\frac{1}{2i} \log\left(\frac{1 + iz_1/z_2}{1 - iz_1/z_2}\right)$
atan2(<i>z</i> ₁ , <i>z</i> ₂ , <i>k</i>)	$\frac{1}{2i} \log\left(\frac{1 + iz_1/z_2}{1 - iz_1/z_2}, k\right)$
sinh(<i>z</i>)	$\sinh x \cos y + i \cosh x \sin y$
cosh(<i>z</i>)	$\cosh x \cos y + i \sinh x \sin y$
tanh(<i>z</i>)	$\frac{\sinh x \cos y + i \cosh x \sin y}{\cosh x \cos y + i \sinh x \sin y}$

continued on next page

Table 3: Valid lvalues related to complex numbers.

Meaning of lvalue	Example
simple variable	<code>z = complex(1.0, 2);</code>
an element of a complex array	<code>zarray[i] = complex(1.0, 2) + ComplexInf;</code>
complex pointer variable	<code>zptr = malloc(sizeof(complex) * 3;</code> <code>zptr = &z;</code>
address pointed to by a complex variable	<code>*zptr = complex(1.0, 2) + z;</code>
an element of a complex pointer array	<code>zarrayptr[i] = malloc(sizeof(complex) * 3;</code> <code>zarrayptr[i] = &z;</code> <code>*zarrayptr[i] = complex(1.0, 2);</code>
address pointed to by an element of a complex pointer array	
real part of a complex variable	<code>real(z) = 3.4;</code>
real part of a complex variable	<code>real(*zptr) = 3.4;</code>
real part of a complex variable	<code>real(*(zptr+1)) = 3.4;</code>
real part of a complex variable	<code>real(*zarrayptr[i]) = 3.4;</code>
imaginary part of a complex variable	<code>imaginary(z) = complex(1.0, 2);</code>
imaginary part of a complex variable	<code>imaginary(*zptr) = 3.4;</code>
imaginary part of a complex variable	<code>imaginary(*(zptr+1)) = 3.4;</code>
imaginary part of a complex variable	<code>imaginary(*zarrayptr[i]) = 3.4;</code>
float pointer variable	<code>fptr = &z;</code> <code>fptr = zptr;</code> <code>*fptr = 1.0;</code> <code>*(fptr+1) = 2.0;</code>
pointer to real part of a complex variable	
pointer to imaginary part of a complex variable	

Basic Complex Analysis

W.H. Freeman and Company San Francisco

Jerrold E. Marsden

University of California, Berkeley

270] Conformal Mappings

Riemann sphere

Note:
H.C.

For some purposes it is convenient to introduce a point " ∞ " in addition to the points $z \in \mathbb{C}$. This has not been done systematically in this text because such a device often leads to confusion and abuse of the symbol ∞ , which, in fact, is merely a convenience.

In contrast to the real line to which $+\infty$ and $-\infty$ are added, we have only one ∞ for \mathbb{C} : the reason is that \mathbb{C} has no natural ordering as \mathbb{R} does.

Formally we add a symbol " ∞ " to \mathbb{C} to obtain the *extended complex plane* $\bar{\mathbb{C}}$ and define operations with ∞ by the "rules"

$$z + \infty = \infty$$

$$z \cdot \infty = \infty$$

$$\infty + \infty = \infty$$

$$\infty \cdot \infty = \infty.$$

We also define, for example, $\lim_{z \rightarrow \infty} f(z) = z_0$ to mean that for any $\epsilon > 0$ there is an R such that $|z| \geq R$ implies that $|f(z) - z_0| < \epsilon$.

Thus a point $z \in \mathbb{C}$ is "close to ∞ " when it lies outside a large circle. This type of closeness can be pictured geometrically by means of the Riemann sphere. The Riemann sphere S is shown in Figure 5.3. By "stereographic projection," illustrated in this figure, a point z' on the sphere is associated with each point z in \mathbb{C} . Exactly one point on the sphere S has been omitted—the "north" pole. We assign ∞ in $\bar{\mathbb{C}}$ to the north pole. Then we see geometrically that z is close to ∞ iff, on the Riemann sphere, these points are close in the usual sense of closeness in \mathbb{R}^3 . Proof of this assertion is requested in exercise 11.

In summary, then, the Riemann sphere represents a convenient geometric picture of the extended plane $\bar{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$. This picture can also be used to make precise the ideas of analytic at ∞ and residue at ∞ (see exercise 10, Section 4.2). It can also be used conveniently to observe points that correspond to ∞ in conformal mappings. We shall have occasion to use it for this purpose in the next section.

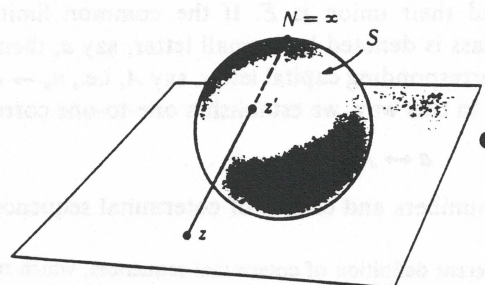


Figure 5.3 Riemann sphere.

THEORY OF FUNCTIONS
OF A COMPLEX VARIABLE

A. I. MARKUSHEVICH

Professor of Mathematics
Moscow State UniversityRevised English Edition
Translated and Edited by

Richard A. Silverman

CHELSEA PUBLISHING COMPANY

NEW YORK N. Y.

5

INFINITY AND STEREOGRAPHIC
PROJECTION

20. Proper and Improper Complex Numbers

In the theory of functions of a complex variable, an important role is played by an "improper" complex number called *infinity* and denoted by the symbol ∞ . In order to define this concept suitably, we first introduce a new interpretation of complex numbers, which comes up quite naturally in problems involving limits.

Two convergent sequences of complex numbers $\{u_n\}$ and $\{v_n\}$ are said to be *coterminal* if they have the same limit.¹ Let E be the set of all convergent sequences of complex numbers, and divide E into all classes (subsets) of coterminal sequences. In other words, $\{u_n\}$ and $\{v_n\}$ belong to the same class if and only if $\{u_n\}$ and $\{v_n\}$ converge to the same limit. Obviously, these classes are disjoint and their union is E . If the common limit of all the sequences in a given class is denoted by a small letter, say a , then we denote the class itself by the corresponding capital letter, say A , i.e., $u_n \rightarrow a$ as $n \rightarrow \infty$ if and only if $\{u_n\} \in A$. In this way, we establish a one-to-one correspondence

$$a \leftrightarrow A, \quad b \leftrightarrow B, \dots \quad (5.1)$$

between the complex numbers and classes of coterminal sequences.

¹ For a somewhat different definition of coterminal sequences, which reduces to the present definition in the case where the sequences are convergent, see G. E. Shilov, *An Introduction to the Theory of Linear Spaces* (translated by R. A. Silverman), Prentice-Hall, Inc., Englewood Cliffs, N. J. (1961), p. 253.

If A and B are two classes of coterminal sequences, we define $A \pm B$ as the class of all sequences of the form $\{u_n \pm v_n\}$, AB as the class of all sequences of the form $\{u_n v_n\}$, and A/B as the class of all sequences of the form $\{u_n/v_n\}$, where $\{u_n\} \in A$, $\{v_n\} \in B$ but are otherwise arbitrary.² Since $\{u_n\} \in A$, $\{v_n\} \in B$, we have

$$\lim_{n \rightarrow \infty} u_n = a, \quad \lim_{n \rightarrow \infty} v_n = b$$

and hence

$$\lim_{n \rightarrow \infty} (u_n \pm v_n) = a \pm b, \quad \lim_{n \rightarrow \infty} u_n v_n = ab, \quad \lim_{n \rightarrow \infty} \frac{u_n}{v_n} = \frac{a}{b}. \quad (5.2)$$

It follows that $A \pm B$, AB and A/B are again classes of coterminal sequences. Moreover (5.2) implies that

$$a \pm b \leftrightarrow A \pm B, \quad ab \leftrightarrow AB, \quad \frac{a}{b} \leftrightarrow \frac{A}{B},$$

i.e., the one-to-one correspondence (5.1) is an *isomorphism*³ between the field of complex numbers and the field of coterminal sequences. In this sense, we are justified in identifying a and A , b and B , etc., and then every class of coterminal sequences is called a *proper complex number*. In particular, the class A can be represented geometrically by the same point in the complex plane as that representing the complex number a . With this interpretation, a convergent sequence $\{u_n\}$ belongs to the class $A = a$ if and only if the unique limit point of the set of points representing the complex numbers $u_1, u_2, \dots, u_n, \dots$ coincides with a .

We now adjoin a single *improper complex number* to the set of all proper complex numbers (i.e., the set of all classes of coterminal sequences). This improper complex number, which we denote by ∞ , is the class of all sequences $\{u_n\}$ with the property that given any $\rho > 0$, there exists an integer $n_0 > 0$ (depending on ρ and $\{u_n\}$) such that $|u_n| > \rho$ whenever $n > n_0$. If a sequence $\{u_n\}$ belongs to the class ∞ , we say that $\{u_n\}$ *converges to infinity*, and we write $u_n \rightarrow \infty$ as $n \rightarrow \infty$ or

$$\lim_{n \rightarrow \infty} u_n = \infty.$$

The union of the set of all proper complex numbers and the improper complex number ∞ is called the extended complex number system. Algebraic operations are defined for the extended complex number system in exactly the same way as for the set of proper complex numbers. However, as the following examples show, in some cases an algebraic expression involving the class ∞ does not lead to a class of coterminal sequences, and hence is meaningless:

² Whenever we write u_n/v_n or a/b , it is assumed that $v_n \neq 0$ or $b \neq 0$.

³ See e.g., G. Birkhoff and S. MacLane, *op. cit.*, p. 35.

1. If a is a proper complex number, then $a + \infty = \infty + a = \infty$, but $\infty + \infty$ is meaningless, as shown by the two sequences

$$1, 3, 3, 5, 5, 7, 7, 9, \dots \quad (5.3)$$

and

$$-1, -2, -3, -4, -5, -6, -7, -8, \dots \quad (5.4)$$

belonging to the class ∞ . In fact, adding (5.3) and (5.4) term by term, we obtain the sequence

$$0, 1, 0, 1, 0, 1, 0, 1, \dots,$$

which obviously does not belong to any class of coterminal sequences (since it has no limit).

2. Similarly, $a - \infty = \infty - a = \infty$ if a is a proper complex number, but $\infty - \infty$ is meaningless.
3. If $a \neq 0$ is a proper complex number, then $a \cdot \infty = \infty \cdot a = \infty$ and moreover $\infty \cdot \infty = \infty$, but $0 \cdot \infty$ is meaningless.
4. If a is any proper complex number, then $a/\infty = 0$ and $\infty/a = \infty$, but ∞/∞ is meaningless.

Remark. This approach also allows us to divide by zero. In fact, $a/0 = \infty$ if $a \neq 0$ is a proper complex number, but $0/0$ is meaningless.

21. Stereographic Projection. Sets of Points on the Riemann Sphere

In order to represent the extended complex number system geometrically, it is convenient to use the following construction, due to Riemann (1826-1866). Consider a sphere Σ of unit radius and center O , and let Π be a plane passing through O (see Figure 5.1). Introducing a rectangular coordinate system in the plane Π , with origin at O , we can represent any proper complex number $z = x + iy$ by a point (x, y) in the plane Π . To associate a point on the sphere Σ with a given point $P \in \Pi$, we first draw the diameter NS of the sphere which is perpendicular to Π and intersects Π at O . Then we draw the line segment joining one end of this diameter, say N , to the point P . The line segment NP (or its prolongation) intersects the sphere Σ in some point P^* different from N . It is clear that this construction establishes a one-to-one correspondence between the points of the sphere Σ (except for the point N itself) and the points of the plane Π . This mapping of the sphere into the plane (or of the plane into the sphere) is called *stereographic projection*, and the sphere Σ is called the *Riemann sphere*. If the point $P \in \Pi$ represents the complex number z , we also regard the point $P^* \in \Sigma$ as representing z .

Note:
H.C.

To be as descriptive as possible, we use geographic terminology. Thus, the circle in which the sphere Σ intersects the plane is called the *equator*, the points N and S are called the *north pole* and *south pole*, respectively, the great circles going through N and S are called *meridians*, and in particular, the meridian lying in the plane NOx is called the *prime* (or *initial*) *meridian*. Then it is easy to see that the points of the plane Π lying inside the unit circle $|z| = 1$ (which coincides with the equator) are mapped into points of the *southern hemisphere* (containing S), while the points of Π lying outside the unit circle are mapped into points of the *northern hemisphere* (containing N). Similarly, the upper half-plane $y > 0$ is mapped into the *eastern hemisphere* (which is intersected by the positive y -axis), while the lower half-plane $y < 0$ is mapped into the *western hemisphere*, and so on.

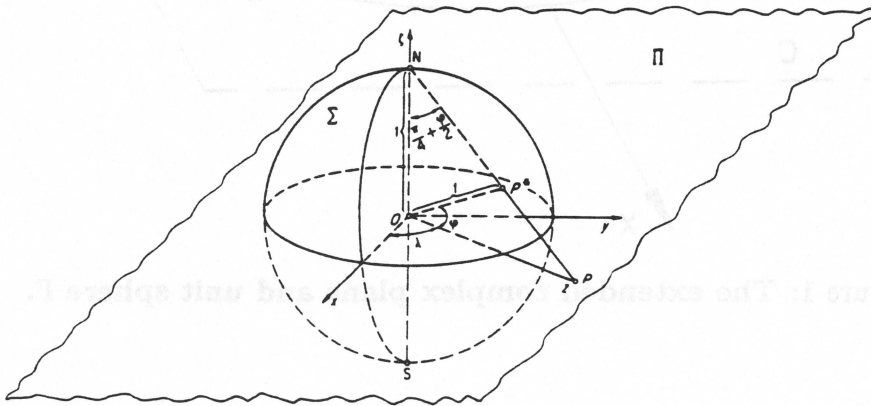


FIGURE 5.1

We now introduce spherical (or geographic) coordinates on Σ , i.e., the *latitude* φ , measured from the equator and ranging from 0 to $\pi/2$ in the northern hemisphere and from 0 to $-\pi/2$ in the southern hemisphere, and the *longitude* λ , measured from the prime meridian (more exactly, from the point of intersection of the prime meridian with the positive x -axis) and ranging from 0 to π (including π) in the eastern hemisphere and from 0 to $-\pi$ (excluding $-\pi$) in the western hemisphere. As shown by Figure 5.1, under stereographic projection we have

$$\arg z = \lambda, \quad |z| = \tan \left(\frac{\pi}{4} + \frac{\varphi}{2} \right).$$

Therefore, the point of the sphere with coordinates λ and φ is the image of the complex number

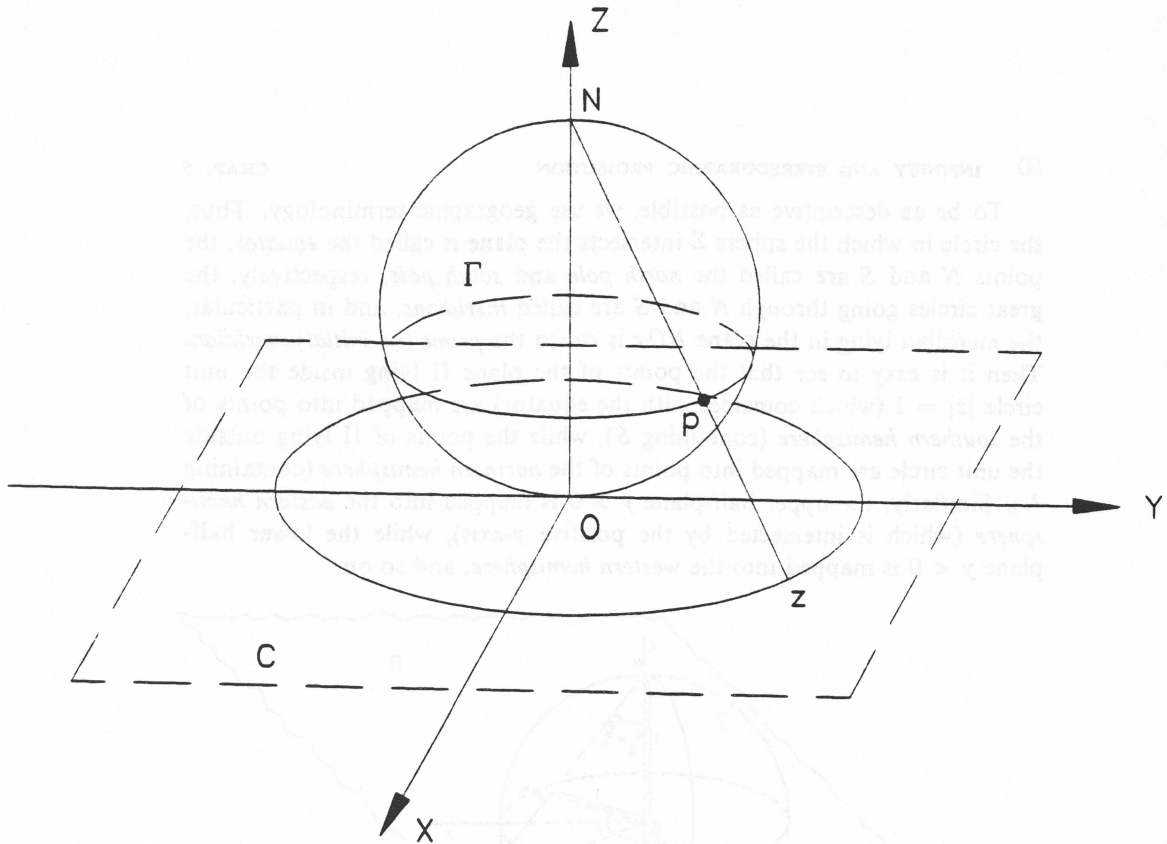


Figure 1: The extended complex plane and unit sphere Γ .

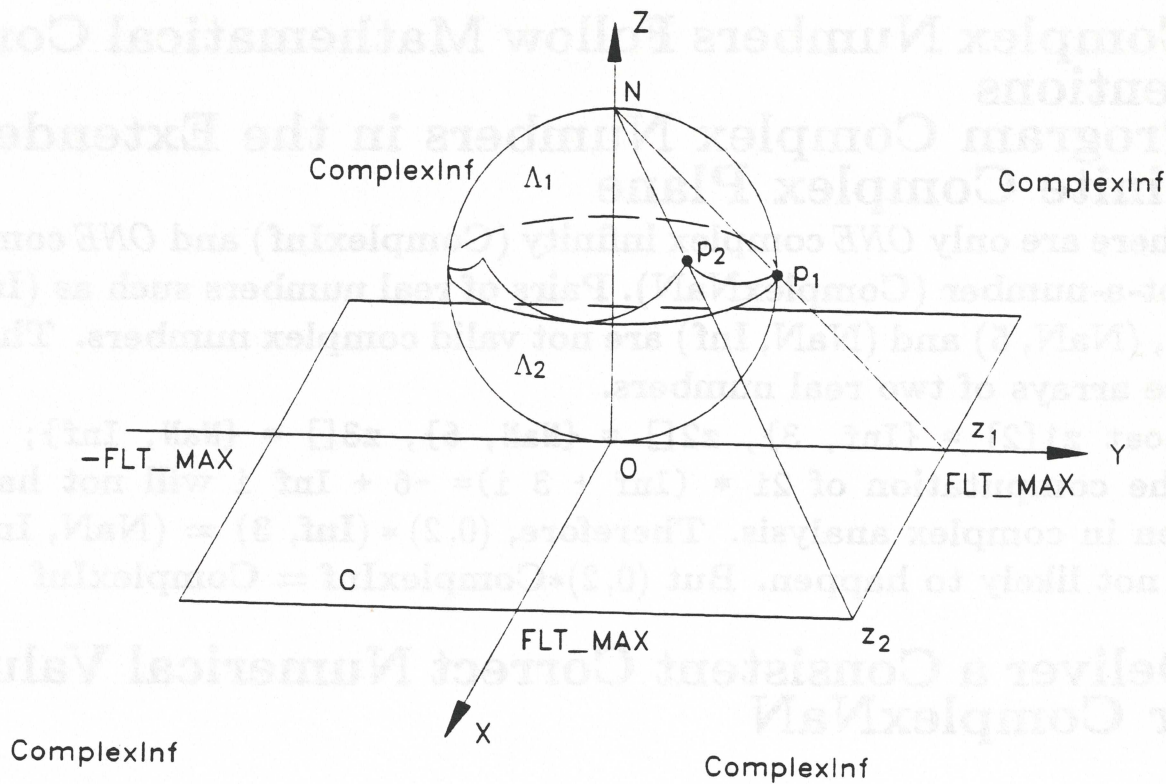


Figure 2: The extended finite complex plane and unit sphere Λ .

Complex Numbers Follow Mathematical Conventions

Program Complex Numbers in the Extended Finite Complex Plane

There are only *ONE* complex infinity (ComplexInf) and *ONE* complex-not-a-number (ComplexNaN). Pairs of real numbers such as (Inf, 3), (NaN, 5) and (NaN, Inf) are not valid complex numbers. They are arrays of two real numbers.

```
float z1[2] = {Inf, 3}, z2[] = {NaN, 5}, z3[] = {NaN, Inf};
```

The computation of $2i * (\text{Inf} + 3i) = -6 + \text{Inf} i$ will not happen in complex analysis. Therefore, $(0, 2) * (\text{Inf}, 3) = (\text{NaN}, \text{Inf})$ is not likely to happen. But $(0, 2) * \text{ComplexInf} = \text{ComplexInf}$

Deliver a Consistent Correct Numerical Value or ComplexNaN

Distinguish -0.0 from 0.0 in Real Numbers, not in Complex Numbers

The origin of the complex plane can be approached from any direction by $\lim_{r \rightarrow 0} r e^{i\theta}$ using `polar(r, theta)`. If the branch cut is not along the coordinate axes, the sign of zeros will be overpowered by round-off errors.

The Principal Value Θ lies in the Range of $-\pi < \Theta \leq \pi$

Unique value for each branch of a multiple-valued complex function.

$F(x + i0) = F(x) + i0$, if x is Within the Valid Domain of $F(x)$

Real numbers form a subset of complex numbers. When the imaginary part of a complex operand or a complex argument is identically zero, the system will deliver a complex result with an identically zero imaginary part. `sqrt(complex(3,0)) == sqrt(3)`
`sqrt(complex(-3,0)) != sqrt(-3)`

Table 4: Addition and subtraction results.

Addition and Subtraction \pm				
left operand	right operand			
complex(0.0, 0.0)	complex(0.0, 0.0)	z2	ComplexInf	ComplexNaN
	complex(0.0, 0.0)	$\pm z2$	ComplexInf	ComplexNaN
z1	z1	$z1 \pm z2$	ComplexInf	ComplexNaN
ComplexInf	ComplexInf	ComplexInf	ComplexNaN	ComplexNaN
ComplexNaN	ComplexNaN	ComplexNaN	ComplexNaN	ComplexNaN

Table 5: Results of complex functions for $\text{complex}(0.0, 0.0)$, ComplexInf , and ComplexNaN .

function	z value and results		
	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{sizeof}(z)$	8	8	8
$\text{abs}(z)$	0.0	Inf	NaN
$\text{real}(z)$	0.0	NaN	NaN
$\text{imaginary}(z)$	0.0	NaN	NaN
$\text{conjugate}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{polar}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{sqrt}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{exp}(z)$	$\text{complex}(1.0, 0.0)$	ComplexNaN	ComplexNaN
$\text{log}(z)$	ComplexInf	ComplexInf	ComplexNaN
$\text{log10}(z)$	ComplexInf	ComplexInf	ComplexNaN
$\text{ceil}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{floor}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{modf}(z, \&z2)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$z2$	$\text{complex}(0.0, 0.0)$	$\text{complex}(0.0, 0.0)$	ComplexNaN
$\text{frexp}(z, \&z2)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$z2$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{ldexp}(z, z2)$	$\text{complex}(0.0, 0.0)$	$\text{complex}(0.0, 0.0)$	ComplexNaN
$\text{sin}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{cos}(z)$	$\text{complex}(0.0, 0.0)$	ComplexNaN	ComplexNaN
$\text{tan}(z)$	$\text{complex}(1.0, 0.0)$	ComplexNaN	ComplexNaN
	$\text{complex}(0.0, 0.0)$	ComplexNaN	ComplexNaN
$\text{asin}(z)$	Note: $\text{tan}(\text{complex}(\pi/2 + k * \pi, 0.0)) = \text{ComplexInf}$		
$\text{acos}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{atan}(z)$	$\text{complex}(\pi/2, 0.0)$	ComplexInf	ComplexNaN
	$\text{complex}(0.0, 0.0)$	$\text{complex}(\pi/2, 0.0)$	ComplexNaN
	Note: $\text{atan}(\text{complex}(0.0, \pm 1.0)) = \text{ComplexInf}$;		
	$\text{atan}(\text{ComplexInf}, k) = \text{complex}(\pi/2 + k * \pi, 0.0)$		
$\text{sinh}(z)$	$\text{complex}(0.0, 0.0)$	ComplexNaN	ComplexNaN
$\text{cosh}(z)$	$\text{complex}(1.0, 0.0)$	ComplexNaN	ComplexNaN
$\text{tanh}(z)$	$\text{complex}(0.0, 0.0)$	ComplexNaN	ComplexNaN
	Note: $\text{tanh}(\text{complex}(0.0, \pi/2 + k * \pi)) = \text{ComplexInf}$		
$\text{asinh}(z)$	$\text{complex}(0.0, 0.0)$	ComplexInf	ComplexNaN
$\text{acosh}(z)$	$\text{complex}(0.0, \pi/2)$	ComplexInf	ComplexNaN
$\text{atanh}(z)$	$\text{complex}(0.0, 0.0)$	$\text{complex}(0.0, \pi/2)$	ComplexNaN
	Note: $\text{atanh}(\text{complex}(\pm 1.0, 0.0)) = \text{ComplexInf}$;		
	$\text{atanh}(\text{ComplexInf}, k) = \text{complex}(0.0, \pi/2 + k * \pi)$		

Add Assumed-shape Arrays

```
int funct(complex a[:], int b[:][:])
int funct(char *c[:]; float **f[:][:], double ***d[:][:][:]);
```

Rules and restrictions

1. An assumed-shape specification is to replace the constant expression for extent of each dimension or the empty argument of the first dimension of an array declaration in a function definition or function prototypes in C by a colon ':'.
2. The rank of an assumed-shape array is equal to the number of colons in the assumed-shape specification.
3. If an array index of the assumed-shape array is less than zero, the lower bound of zero will be used as the index. If an index of the assumed-shape array is greater than the upper bound of the actual array, the upper bound of the actual array will be used as the index. In either case, a warning message will be generated by the system.
4. Only arrays, including assumed-shape and deferred-shape arrays, can be passed to an assumed-shape array. Pointers or pointers to array, which do not have the complete shape information, can not be passed to an assumed-shape array. Otherwise, an error message will be generated by the system.
5. Polymorphic operations and intrinsic functions will be applied to expressions involving an assumed-shape array based upon the data type of its formal definition. If the data types of the actual and assumed-shape arrays are different, a warning message will be generated by the system at compilation time; the data type of an actual array will be converted to the data type of the assumed-shape array implicitly at execution time.
6. **SHOULD THIS BE A RULE?** *The same array can not be passed to the assumed-shape arrays of a function more than once. Otherwise, a warning message will be generated by the system and the behavior of the function is undefined.*

Add the Keyword *array* or *matrix* for Array Syntax

Array Syntax Follows the Mathematical Conventions

```
/****** CH *****/
```

```
matrix int X[100], Y[100], Z[100];
```

```
X = Y + Z;
```

```
X += 17;
```

```
/****** DPCE: C* *****/
```

```
shape [100]Shape;
```

```
int:Shape X, Y, Z;
```

```
X = Y + Z;
```

```
X += 17;
```

```
/****** Iterators: Cray *****/
```

```
int X[100], Y[100], Z[100];
```

```
iter I;
```

```
X[I] = Y[I] + Z[I];
```

```
X[I] += 17;
```

```
/****** ANSI C *****/
```

```
int X[100], Y[100], Z[100];
```

```
int i;
```

```
for(i=0; i<100; i++)
```

```
    X[i] = Y[i] + Z[i];
```

```
for(i=0; i<100; i++)
```

```
    X[i] += 17;
```

Dual Numbers and Dual Constructor

Dual numbers $d \in \mathcal{D} = \{(x, y) | x, y \in \mathcal{R}\}$ can be defined as ordered pairs


$$d = (x, y)$$

with specific addition and multiplication rules (Clifford, 1873).

The real number \mathcal{R} is a subset of \mathcal{D} , i.e., $\mathcal{R} = \{(x, y) | x \in \mathcal{R}, y = 0\}$ and $\mathcal{R} \subset \mathcal{D}$. If a real number is considered either as x or $(x, 0)$ and let ε denote the *pure dual number* $(0, 1)$ with the property of $\varepsilon^2 = 0$, dual numbers can be written as

$$d = x + \varepsilon y$$

a dual number can be created in \mathcal{C}^H by the *dual constructor* `dual(x, y)` with $x, y \in \mathcal{R}$. For example, `dual(1.2, 3.4)`.

data type	order
dual	
complex	
double	
float	
int	
char	
	high
	low

The order of basic data types in C^H .

Data types of char, int, float, double, complex, and dual can be converted implicitly and explicitly in C^H .

Dual Variables

```
dual d1;           /* declare d1 as dual variable */
dual *dptr1;        /* dptr1 is pointer to dual variable
dual d2[2];         /* d2 is an array of dual
dual *dptr2[2][4];  /* dptr2 is array of pointer to dual
dptr1 = &d1;        /* dptr1 points at the address of d1
*dptr1 = dual(1,2); /* d1 with real 1.0 and dual 2.0
```


I/O for Dual Numbers

The following C^H program will illustrate how dual numbers are handled by the I/O functions printf() and scanf().

```
dual d1, d2, *dptr;  
dptr = &d2;           /* dptr points to d2's memory  
printf("Please type in four numbers for two duals \n");  
scanf(&d1, dptr);  
printf("The first dual number is ", d1, "\n");  
printf("The second dual number is  %f \n", d2);
```

The result of the interactive execution of the above program is shown as follows

Please type in four numbers for two duals

1 2.0 3.0 4

The first dual number is dual(1.000,2.000)
The second dual number is dual(3.000000,4.000000)

where the second line in italic is the input and the rest are the output of the program.

Table 6: Dual operations.

Definition	C^H Syntax	C^H Semantics
negation	$-d$	$-x - \varepsilon y$
addition	$d1 + d2$	$(x_1 + x_2) + \varepsilon(y_1 + y_2)$
subtraction	$d1 - d2$	$(x_1 - x_2) + \varepsilon(y_1 - y_2)$
multiplication	$d1 * d2$	$x_1 * x_2 + \varepsilon(y_1 * x_2 + x_1 * y_2)$
division	$d1 / d2$	$\frac{x_1}{x_2} + \varepsilon \frac{y_1 * x_2 - x_1 * y_2}{x_2^2}$
equal	$d1 == d2$	$x_1 == x_2$ and $y_1 == y_2$
not equal	$d1 != d2$	$x_1 != x_2$ or $y_1 != y_2$

Dual Functions

A dual function $f(d) = f(x + \varepsilon y)$ with a dual variable can be obtained by expanding it formally in a Taylor series. Because $\varepsilon^n = 0$ for $n > 1$, then,

$$f(d) = f(x + \varepsilon y) = f(x) + \varepsilon y f'(x)$$

where $f'(x)$ in the dual part is the derivative of function $f(x)$. Similarly, a dual function with two dual variables can be obtained as

$$\begin{aligned} f(d_1, d_2) &= f(x_1 + \varepsilon y_1, x_2 + \varepsilon y_2) \\ &= f(x_1, x_2) + \varepsilon(y_1 f'_{d_1}(x_1, x_2) + y_2 f'_{d_2}(x_1, x_2)) \end{aligned}$$

where $f'_{d_1}(x_1, x_2)$ is the partial derivative with respect to the first variable d_1 and $f'_{d_2}(x_1, x_2)$ with respect to the second.

Table 7: Syntax and semantics of built-in dual functions.

C^H Syntax	C^H Semantics
<code>sizeof(d)</code>	8
<code>abs(d)</code>	$\text{sqrt}(x^2 + y^2)$
<code>real(d)</code>	x
<code>imaginary(d)</code>	y
<code>dual(x, y)</code>	$x + \varepsilon y$
<code>conjugate(d)</code>	$x - \varepsilon y$
<code>polar(d)</code>	$\text{sqrt}(x^2 + y^2) + i \Theta$
<code>sqrt(d)</code>	$\text{sqrt}(x) + \varepsilon \frac{y}{2\text{sqrt}(x)}$
<code>exp(d)</code>	$\exp(x)(1 + \varepsilon y)$
<code>log(d)</code>	$\log(x) + \varepsilon \frac{y}{x}$
<code>log10(d)</code>	$\frac{\log(d)}{\log(10)}$
<code>pow(d_1, d_2)</code>	$d_1^{d_2} = x_1^{x_2} + \varepsilon(y_1 x_2 x_1^{x_2-1} + y_2 x_1^{x_2} \log(x_1))$
<code>sin(d)</code>	$\sin(x) + \varepsilon y \cos(x)$
<code>cos(d)</code>	$\cos(x) - \varepsilon y \sin(x)$
<code>tan(d)</code>	$\tan(x) + \varepsilon \frac{y}{\cos(x) \cos(x)}$
<code>asin(d)</code>	$\text{asin}(x) + \varepsilon \frac{y}{\text{sqrt}(1-x^2)}$
<code>acos(d)</code>	$\text{acos}(x) - \varepsilon \frac{y}{\text{sqrt}(1-x^2)}$
<code>atan(d)</code>	$\text{atan}(x) + \varepsilon \frac{y}{1+x^2}$
<code>sinh(d)</code>	$\sinh(x) + \varepsilon y \cosh(x)$
<code>cosh(d)</code>	$\cosh(x) + \varepsilon y \sinh(x)$
<code>tanh(d)</code>	$\tanh(x) + \varepsilon \frac{y}{\cosh(x) \cosh(x)}$
<code>asinh(d)</code>	$\text{asinh}(x) + \varepsilon \frac{y}{\text{sqrt}(x^2+1)}$
<code>acosh(d)</code>	$\text{acosh}(x) - \varepsilon \frac{y}{\text{sqrt}(x^2-1)}$
<code>atanh(d)</code>	$\text{atanh}(x) + \varepsilon \frac{y}{1-x^2}$
<code>ceil(d)</code>	$\text{ceil}(x) + \varepsilon \text{ceil}(y)$
<code>floor(d)</code>	$\text{floor}(x) + \varepsilon \text{floor}(y)$
<code>ldexp($d1, d2$)</code>	$\text{ldexp}(x_1, x_2) + \varepsilon \text{ldexp}(y_1, y_2)$
<code>fmod($d1, d2$)</code>	$d; \quad \frac{d_1}{d_2} = k + \frac{d}{d_2}, k \geq 0$
<code>modf($d1, \&d2$)</code>	$\text{modf}(x_1, \&x_2) + \varepsilon \text{modf}(y_1, \&y_2)$
<code>frexp($d1, \&d2$)</code>	$\text{frexp}(x_1, \&x_2) + \varepsilon \text{frexp}(y_1, \&y_2)$

Table 8: Valid lvalues related to dual numbers.

Meaning of lvalue	Example
simple variable	<code>d = dual(1.0, 2);</code>
an element of a dual array	<code>darray[i] = dual(1.0, 2) + DualInf;</code>
dual pointer variable	<code>dptr = malloc(sizeof(dual) * 3);</code> <code>dptr = &d;</code>
address pointed to by a dual variable	<code>*dptr = dual(1.0, 2) + z;</code>
an element of a dual pointer array	<code>darrayptr[i] = malloc(sizeof(dual) * 3);</code> <code>darrayptr[i] = &d;</code> <code>*darrayptr[i] = dual(1.0, 2);</code>
address pointed to by an element of a dual pointer array	
real part of a dual variable	<code>real(d) = 3.4;</code>
real part of a dual variable	<code>real(*dptr) = 3.4;</code>
real part of a dual variable	<code>real(*(dptr+1)) = 3.4;</code>
real part of a dual variable	<code>real(*darrayptr[i]) = 3.4;</code>
imaginary part of a dual variable	<code>imaginary(d) = dual(1.0, 2);</code>
imaginary part of a dual variable	<code>imaginary(*dptr) = 3.4;</code>
imaginary part of a dual variable	<code>imaginary(*(dptr+1)) = 3.4;</code>
imaginary part of a dual variable	<code>imaginary(*darrayptr[i]) = 3.4;</code>
float pointer variable	<code>fptr = &d;</code> <code>fptr = dptr;</code> <code>*fptr = 1.0;</code> <code>*(fptr+1) = 2.0;</code>
pointer to real part of a dual variable	
pointer to imaginary part of a dual variable	

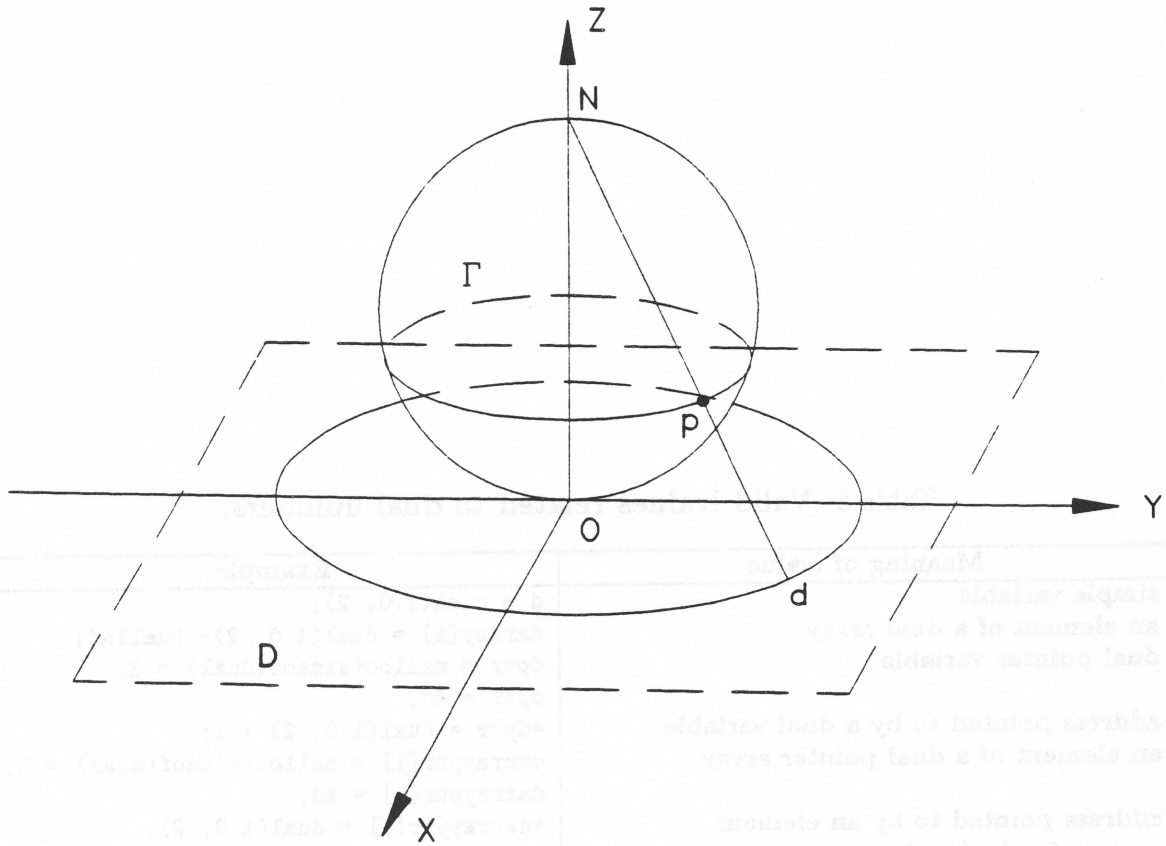


Figure 3: The extended dual plane and unit sphere Γ .

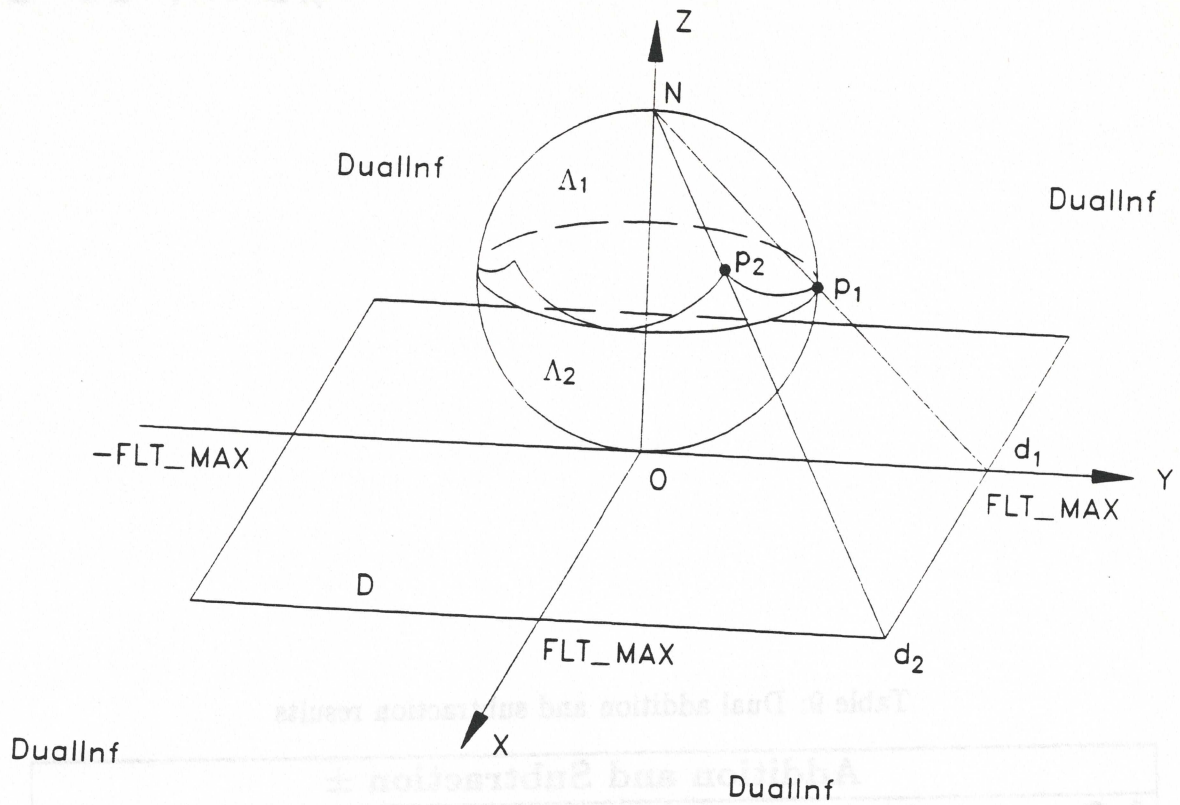


Figure 4: The extended finite dual plane and unit sphere Λ .

Table 9: Dual addition and subtraction results

Addition and Subtraction \pm				
left operand	right operand			
	DualZero	d2	DualInf	DualNaN
DualZero	DualZero	$\pm d2$	DualInf	DualNaN
d1	d1	$d1 \pm d2$	DualInf	DualNaN
DualInf	DualInf	DualInf	DualNaN	DualNaN
DualNaN	DualNaN	DualNaN	DualNaN	DualNaN

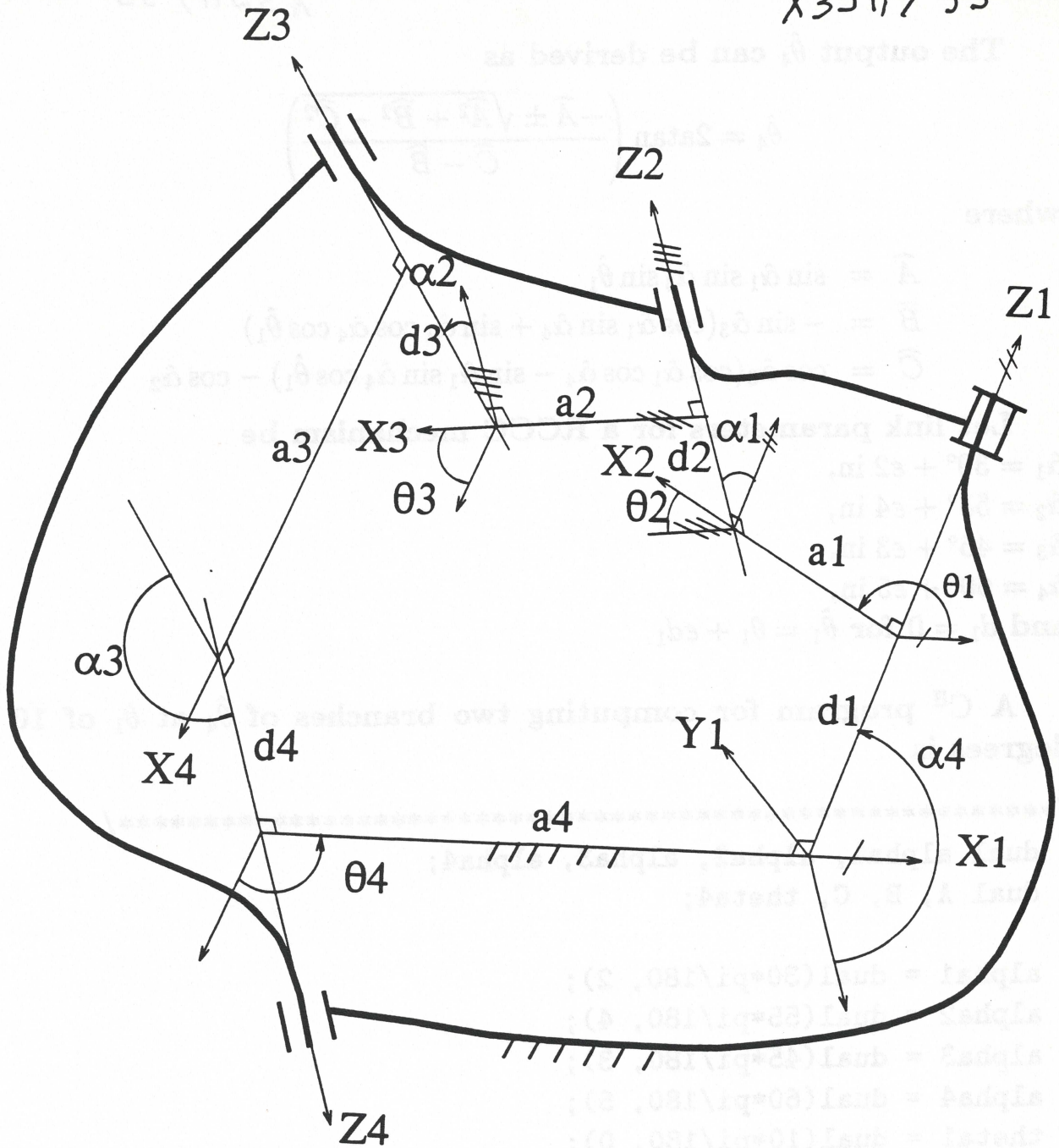


Figure 5: The RCCC mechanism.

The output $\hat{\theta}_4$ can be derived as

$$\hat{\theta}_4 = 2 \operatorname{atan} \left(\frac{-\hat{A} \pm \sqrt{\hat{A}^2 + \hat{B}^2 - \hat{C}^2}}{\hat{C} - \hat{B}} \right)$$

where

$$\hat{A} = \sin \hat{\alpha}_1 \sin \hat{\alpha}_3 \sin \hat{\theta}_1$$

$$\hat{B} = -\sin \hat{\alpha}_3 (\cos \hat{\alpha}_1 \sin \hat{\alpha}_4 + \sin \hat{\alpha}_1 \cos \hat{\alpha}_4 \cos \hat{\theta}_1)$$

$$\hat{C} = \cos \hat{\alpha}_3 (\cos \hat{\alpha}_1 \cos \hat{\alpha}_4 - \sin \hat{\alpha}_1 \sin \hat{\alpha}_4 \cos \hat{\theta}_1) - \cos \hat{\alpha}_2$$

Let link parameters for a RCCC mechanism be

$$\hat{\alpha}_1 = 30^\circ + \varepsilon_2 \text{ in,}$$

$$\hat{\alpha}_2 = 55^\circ + \varepsilon_4 \text{ in,}$$

$$\hat{\alpha}_3 = 45^\circ + \varepsilon_3 \text{ in,}$$

$$\hat{\alpha}_4 = 60^\circ + \varepsilon_5 \text{ in,}$$

$$\text{and } d_1 = 0 \text{ for } \hat{\theta}_1 = \theta_1 + \varepsilon d_1$$

A C^H program for computing two branches of $\hat{\theta}_4$ at θ_1 of 10 degrees is

```

/*****
dual alpha1, alpha2, alpha3, alpha4;
dual A, B, C, theta4;

alpha1 = dual(30*pi/180, 2);
alpha2 = dual(55*pi/180, 4);
alpha3 = dual(45*pi/180, 3);
alpha4 = dual(60*pi/180, 5);
theta1 = dual(10*pi/180, 0);
A = sin(alpha1)*sin(alpha3)*sin(theta1);
B = -sin(alpha3) * (cos(alpha1)*sin(alpha4) +
    sin(alpha1)*cos(alpha4)*cos(theta1));
C = cos(alpha3) * (cos(alpha1)*cos(alpha4) -
    sin(alpha1)*sin(alpha4)*cos(theta1)) - cos(alpha2);
theta4 = 2*atan((-A + sqrt(A*A + B*B - C*C))/(C-B));
theta4 = 2*atan((-A - sqrt(A*A + B*B - C*C))/(C-B));
*****/

```