Sistemas–L, Tortugas y Poliedros Recursivos
Manual del Usuario

por Eduardo Virueña Silva

Ocubre 2005

1. Introducción

En los años ochenta, el biólogo suizo Aristid Lindenmayer, propuso un modelo para la descripción del crecimiento de las plantas usando gramáticas formales al que llamó *Sistemas–L* [1].

En la descripción, usaba *geometría de la tortuga*, que fue definida en los años setenta, en el lenguaje de programación LOGO.

Con este par de conceptos es posible trazar *curvas recursivas*[2] en el plano, pero yendo más allá, Don Aristid extendió el lenguaje de la tortuga para que ésta pudiera moverse en un espacio tridimensional y trazar curvas ahí.

En este trabajo se presenta un programa que expande sistemas–L, un par de tortugas, una plana y otra tridimensional, y se extiende el lenguaje de las tortugas para que éstas puedan construir polígonos en el espacio. Al reunir estos polígonos pueden crearse poliedros y también crearse *poliedros recursivos*.

Se ha tratado, en lo posible, de hacer todos los programas con funcionalidad de *filtro*, es decir, son programas que toman su entrada estándar, la procesan y producen un resultado que escriben en su salida estándar. De esta manera es más fácil hacer que funcionen bajo distintos sistemas

[1] Por supuesto, la "L" es por la inicial de su apellido

[2] Inductivas

operativos y en distintas máquinas. Su desarrollo se ha logrado usando solamente herramientas de licencia libre[3], inclusive la documentación se escribió con estas herramientas.

Para mantener los programas relativamente simples y capaces de producir imágenes en formatos conocidos, se decidió usar un lenguaje de descripción de páginas que se llama POSTSCRIPT. La conversión a formatos gráficos puede hacerse con programas como GhostScript, que es de distribución gratuita.

Los programas fueron probados en sistemas relativamente viejos (equipos Intel 486, Intel Pentium) solamente bajo el sistema operativo UNIX, en particular con Linux Fedora Core 3, FreeBSD 4.10 y 5.4. También fueron probados en sistemas basados en Windows 98 o posterior. Es posible también compilarlos usando herramientas de desarrollo viejas que funcionen bajo MS-DOS pero, bajo estas circunstancias, se ven muy restringidos por limitaciones de memoria.

2. La tortuga bidimensional

2.1. Descripción

El lenguaje de programación LOGO fue pensado para enseñar a los niños a programar. Se les contaba la historia de que había una tortuga que se encontraba en un piso plano de color blanco. La tortuga traía consigo una pluma de un color dado y que la tortuga ponía sobre el piso cuando se le ordenaba. Se le podía ordenar también dar un paso, dar vuelta a la derecha, dar vuelta a la izquierda, cambiar el color de la pintura y otras cosas. Con estas instrucciones, la tortuga dibujaba en el piso y los niños podían crear dibujos fácilmente. Es increíble cómo este cuento puede dar una idea tan clara de cómo quiere instruirse a una computadora para trazar un dibujo.

2.2. Parámetros de la tortuga

Tradicionalmente, la tortuga usa como medida angular los grados, pero no define las unidades de longitud que define, en este trabajo la unidad de

[3]Herramientas que pueden conseguirse en Internet sin que sea necesario pagar por una licencia para su uso legal.

longitud será el centímetro. Los parámetros que describen el estado de la tortuga son:

Posición en el plano: Es una pareja de números reales que describe en qué coordenadas se encuentra la tortuga.

La posición de la tortuga se establece escribiendo: $G(expr_x, expr_y)$, donde $(expr_x, expr_y)$ son expresiones que al evaluarse dan números reales que indican las coordenadas de la tortuga en el plano. Inicialmente, la posición de la tortuga es $(0, 0)$. Dentro de una expresión, la abscisa de la tortuga se llama x y la ordenada se llama y.

Ángulo de rotación: Es el ángulo con el que la tortuga gira a la derecha o a la izquierda cada vez que se le ordena hacerlo. También es la medida angular de los arcos que traza.

El valor ángulo de rotación se establece escribiendo: $A(expr)$, donde $expr$ es una expresión que al evaluarse da un número real. Inicialmente su valor es 0. Dentro de una expresión, el ángulo de rotación se llama A.

Ángulo de dirección: Al dar un paso, la tortuga lo hace en una dirección que forma un cierto ángulo, en grados, con respecto al eje X. Ese ángulo se llama *ángulo de dirección*.

El valor ángulo de dirección se establece escribiendo $D(expr)$, donde $expr$ es una expresión que al evaluarse da un número real. Su valor inicial es 0. Dentro de una expresión el ángulo de dirección se llama D.

Tamaño del paso: Es la medida en centímetros que le dice a la tortuga de qué tamaño serán los pasos que va a dar. Su valor se establece escribiendo $S(expr)$, donde $expr$ es una expresión que, al evaluarse, da un número real. Su valor inicial es 0. Dentro de una expresión, el tamaño del paso se llama S.

Radio de curvatura: Es la medida del radio con el que la tortuga traza arcos. Su valor se establece escribiendo $R(expr)$, donde $expr$ es una expresión que, al evaluarse, da un número real. Su valor inicial es 0. Dentro de una expresión, el radio de curvatura se llama R.

Si el valor del radio de curvatura es positivo, la tortuga traza un arco hacia la izquierda de su orientación actual; si es negatvo, a la derecha. La medida angular del arco depende del ángulo de giro.

Color de la pintura: Son tres número reales entre 0 y 1, que describen las componentes rojo, verde y azul del color con el que la tortuga pinta.

El color de la pintura se establece escribiendo $P(expr_r, expr_g, expr_b)$, donde $expr_r$ $expr_g$ y $expr_b$ dan un número en real en el intervalo $[0, 1]$. Si, al evaluar las expresiones, su valor fuera mayor que uno, el valor se establece como uno; si fuera menor que cero, su valor se establece como 0. Si no se establece, su valor es cero (negro).

Ancho de la pluma: El ancho de la pluma se establece escribiendo $W(expr)$ donde, $expr$ establece cuantos centímetros mide la pluma de ancho. El ancho de la pluma se llama W. Su valor inicial es de 0.05cm, aproximadamente

2.3. Instrucciones para mover a la tortuga

Las instrucciones para una tortuga en el plano son pocas y simples. Las que usamos en estos programas se escriben en un archivo de texto y todas ellas constan de un solo carácter.

`f` Dar un paso hacia adelante sin pintar.

`F` Dar un paso hacia adelante pintando.

`+` Girar a la izquierda.

`-` Girar a la derecha.

`T` Trazar un arco.

`[` Guardar en la pila el estado de la tortuga.

`]` Restablecer el estado de la tortuga, sacándolo de la pila.

Por *estado de la tortuga* queremos decir: posición, ángulo de inclinación, ángulo de giro, tamaño del paso, radio de curvatura, ancho de la pluma y

color de la pintura.

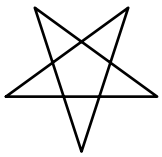
2.3.1. Ejemplos

Considérese el texto:

```
A(144) S(2) F + F + F + F + F +.
```

- Los ángulos de giro serán de 144 grados.
- El tamaño del paso es de 2 cm.
- Dar un paso pintando, y dar vuelta a la derecha (que se repite 5 veces).

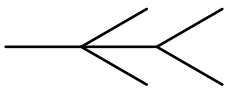
El resultado de estas instrucciones será:



Considérese ahora:

```
A(30) S(1) F [+F] [-F] F [+F] [-F]
```

Aquí debe notarse que la tortuga está guardando su estado cada vez que encuentra al corchete izquierdo, y lo recupera cada vez que encuentra al corchete derecho, de manera que las instrucciones de arriba dibujan:



signo	nombre
$-$	cambio de signo
$\wedge$	exponenciación
$/$	división
$*$	multiplicación
$-$	diferencia
$+$	suma

Cuadro 1: Operadores

2.4. Expresiones

Los parámetros de la tortuga se establecen mediante expresiones que determinen sus valores. El cuadro siguiente enlista los operadores de las expresiones y la precedencia de su asociatividad.

Los operadores tienen el mismo orden de asociación que el que tienen en C, pero la tortuga puede calcular potencias mediante el operador de exponenciación $\wedge$, que se asocia antes que los productos, pero después que el cambio de signo lo cual es un poco extraño.

Se dispone además de las funciones más comunes:

`sin, cos, tan, asin, acos, atan, ln, sqrt, exp, sinh, cosh, tanh, abs, deg, rad`.

Las funciones trigonométricas operan en grados. `deg` convierte radianes a grados y `rad` convierte grados en radianes.

Pueden usarse referencias a los nombres de los parámetros X, Y, D, A, S, W. Los números reales que siguen la convención de escritura de los lenguajes de programación pero si aparece en ellos el punto decimal, debe estar precedido por un dígito

Hay dos constantes simbólicas, $pi = \pi$ y tau, la proporción áurea, cuyo valor es:

$$\tau = \frac{1 + \sqrt{5}}{2}$$

2.4.1. Ejemplo

Las instrucciones: `A(A/2) S(sqrt(2)) D(acos(tau/(tau+2))` significan:

- Cambiar el ángulo de giro por su valor entre dos.
- Establecer el tamaño de paso como $\sqrt{2}$.
- Dar el valor $\frac{\tau}{\tau+2}$ al ángulo de dirección.

3. La tortuga tridimensional

3.1. Descripción

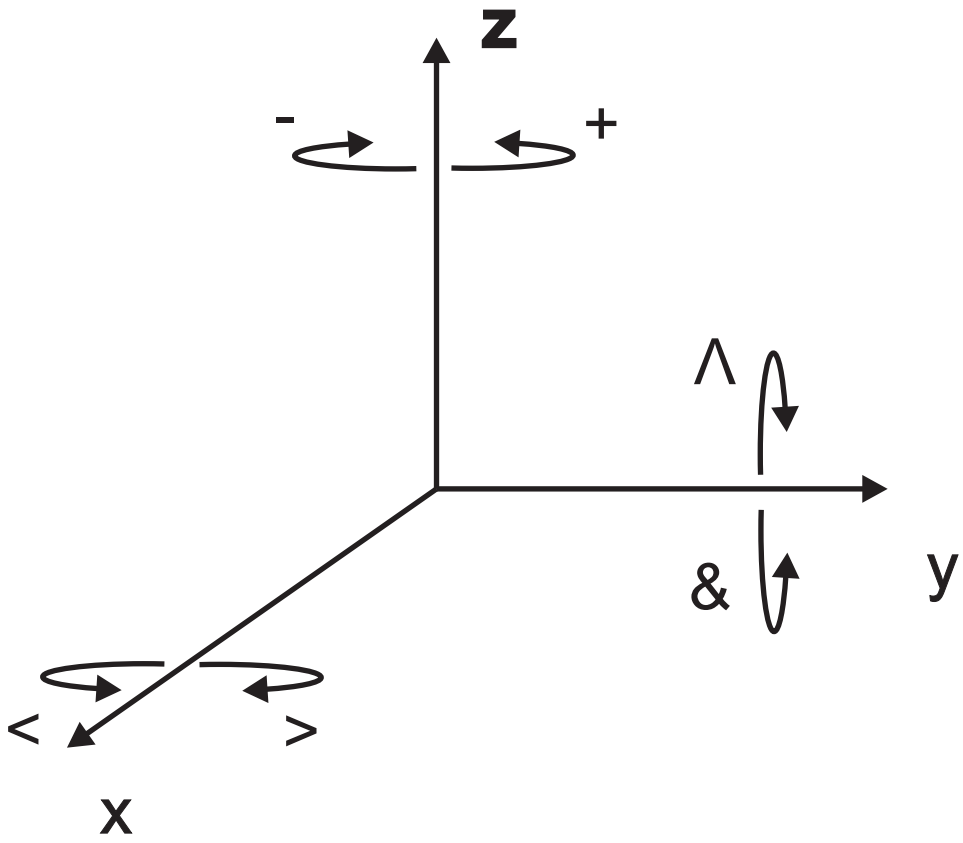
Ahora dotaremos a la tortuga de la capacidad de abandonar el plano, para seguir con el cuento, pensaremos en que se le añaden alas r'ıgidas, como las de un avión.

3.2. Instrucciones

La tortuga tiene tres grados de libertad en el espacio, que están asociadas a nuevas instrucciones:

+	Girar a la derecha
-	Girar a la derecha
$\wedge$	Girar hacia arriba
&	Girar hacia abajo
$<$	Bajar el ala izquierda (y subir la derecha)
$>$	Bajar el ala derecha (y subir la izquierda)
(	Empezar polígono
)	Terminar polígono

Formalmente, supondremos que la tortuga lleva consigo un sistema de referencia respecto al cual hace los giros. El giro a la izquierda o a la derecha es equivalente a rotar alrededor del eje Z; girar para subir o para bajar equivale a rotar alrededor del eje Y; por último, bajar el ala izquierda o bajar el ala derecha equivale a rotar alrededor del eje X.



La tortuga tridimensional necesita ahora tres expresiones para establecer su posición, así que ahora escribiremos: $G(expr_x, expr_y, expr_z)$ y los nombres de sus coordenadas dentro de las expresiones son: X, Y, Z, respectivamente.

Los ángulos de giro ahora se establecen con los nombres de los ejes respecto a los cuales giran: $X(expr)$ establece el valor del ángulo de giro alrededor del eje X; $Y(expr)$ establece el valor del ángulo de giro alrededor del eje Y; $Z(expr)$ establece el valor del ángulo de giro alrededor del

eje Z. Los nombres de los ángulos de giro dentro de las expresiones son Ax, Ay, Az, respectivamente.

La tortuga tridimensional no puede establecer fácilmente el equivalente al ángulo de dirección[4]. Inicialmente, la tortuga está orientada hacia la parte positiva del eje de X, es decir, en dirección $(1, 0, 0)$ y el cielo está en la dirección $(0, 0, 1)$.

La tortuga tiene la posiblidad de dibujar también polígonos en el espacio, para ello se usan los paréntesis "(" y ")" que indicarán cuando empieza un polígono y cuando acaba, respectivamente.

Los vértices del polígono son los puntos a los que las instrucciones `F` y `f` *llegan*; sus aristas pueden trazarse o no, dependiendo si se usa `F` (trazar) o bien `f` (no trazar). Si ninguna de las aristas del polígono se trazan, el polígono solo representará un área opaca en el dibujo.

Así, tenemos que si decimos: `S(1) Z(90) (F+F+F+F+)`, estamos instruyendo a la tortuga para que dé pasos de un centímetro de largo y que dé giros de 90 grados cuando gire alrededor del eje Z. La tortuga empieza en el origen $(0, 0, 0)$, traza un polígono (un cuadrado): $(1, 0, 0) \rightarrow (1, 1, 0) \rightarrow (0, 1, 0) \rightarrow (0, 0, 0)$. Traza, además, cada arista del cuadrado: $(0, 0, 0) \rightarrow (1, 0, 0) \rightarrow (1, 1, 0) \rightarrow (0, 1, 0)$

La tortuga tridimensional no tiene posibilidades de dibujar a color ni de trazar arcos[5].

3.3. Ejemplo: El tetraedro

Consideremos el problema de dibujar un tetraedro regular. El tetraedro, como su nombre lo indica es un poliedro con cuatro caras, que sea regular significa que cada una sus caras es un triángulo equilátero.

Vamos a estudiar las instrucciones siguientes:

```
00      Z(120)
01      X(acos(1/3))
02       (F+F+F+)
03      >
04       (F+F+F+)
05     <f+>
```

[4]Requeriría de dos vectores para orientarse correctamente, así que mejor se ha omitido esta posibilidad

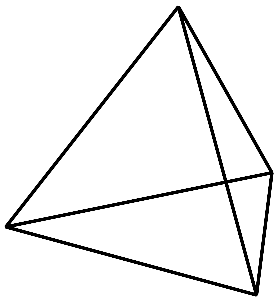
[5]Pero en un futuro muy cercano, tendrá

```
06      (F+F+F+)
07    <f+>
08       (F+F+F+)
09    <f+
```

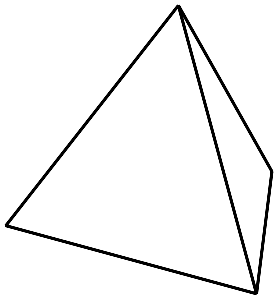
Se han incluido unos número sucesivos del lado derecha que van a servir como referencia.

La descripción del tetraedro es más o menos sencilla:

- Se establece un ángulo de giro de 120^o, que es igual al ángulo exterior de los triángulos equiáteros, respecto al eje Z (es decir, el que ocupa la tortuga para dar vuelta a la derecha o a la izquierda), mismo que se establece en la línea 00.
- El ángulo de giro respecto al eje X (el que se ocupa para bajar el ala izquierda o el ala derecha) será de $\arccos \frac{1}{3} \approx 70.53^o$, que es el ángulo diedro del tetraedro, mismo que establecemos en la línea 01.
- La línea 02 describe la base del tetraedro.
- Luego, la línea 03, le ordena a la tortuga girar para que ahora su *panza* esté sobre una de las caras.
- Con la misma secuencia de instrucciones con que se hace la base, la tortuga traza esa cara (línea 04).
- La línea 05 ordena a la tortuga volver a poner la panza en la base del tetraedro, avanzar sin pintar sobre una de las aristas de la base y girar a la izquierda para volver alinearse con la tercera cara.
- La línea 06 define otra de las caras.
- La línea 07 avanza sobre la base, da vuelta y alinea a la tortuga con la última de las caras.
- La línea 08, traza la última cara.
- Para que la tortuga no quede abandonada a su suerte apuntando a quién sabe donde, la línea 09 vuelve a colocarla en el lugar donde todo empezó.



La figura anterior muestra un tetraedro de alambre, realmente a la tortuga se le ha ordenado crear polígonos (triángulos), que son opacos, como éste:



3.4. Observaciones

La unidad de longitud para dar pasos nuevamente es el centímetro, sin embargo, la tortuga ahora ya no dibuja directamente en el papel, más bien genera un archivo de texto con la descripción de la escena que está dibujando. La interpretación de esta escena depende de un programa externo que pueda interpretarlo, por lo que las unidades de longitud dejan de tener significado, ahora dependen de cómo las interprete el programa de despliegue gráfico.

El despliegue tridimensional podría hacerse de muchas maneras: Dibujando un par estereoscópico, un estereograma de puntos aleatorios, un anaglifo[6], o traduciendo la escena a un lenguaje que pueda leer un pro-

[6]anaglifo. (Del gr. $\alpha\nu\alpha\gamma\lambda\upsilon\phi o\zeta$, tallado en relieve). Superposición de dos imgenes, una en color rojo y otra en verde, que producen, al ser miradas con lentes especiales, una impresión de relieve.

grama de visualización de VRML[7], etc. Inclusive puede traducirse a un lenguaje para la contrucción de rápida de prototipos, para que una impresora tridimensional la esculpa. Las posibilidades son muchas.

3.5. Otro ejemplo: El dodecaedro

Tal vez no haya figura más bella en la Geometría como la figura del dodecaedro. Una de las primeras ilustraciones de las que se tiene memoria se debe a Leonardo da Vinci, que ilustró el libro de Luca Pacioli, *De Divina Proporcione*, publicado en el año 1509[8].

El dodecaedro tiene ángulo diedro[9] igual a:

$$\begin{aligned} \text{diedro} &= \arccos\left(-\frac{\tau}{\tau+2}\right) \\ \tau &= \frac{1+\sqrt{5}}{2} \end{aligned}$$

τ es el famoso *número áureo, proporción áurea o divina* de la que hablaba Luca Pacioli en su libro. Los valores de estos números son, aproximadamente:

$$\begin{aligned} \text{diedro} &\approx 116.565051177077989351572193720 45^o \\ \tau &\approx 1.6180339887498948482045868343656 \end{aligned}$$

El ángulo externo que forman las aristas de los pentágonos es de 108^o, para formarlo, necesitaríamos dar giros de 72^o pero no usaremos ese ángulo sino su mitad (36^o), porque vamos a necesitar dar giros de 36^o, 72^o y 108^o. Usaremos giros sobre el eje X para cambiar de cara pero no usaremos el ángulo diedro sino su suplemento.

Entonces, los ángulos de giro van a ser:

```
X(acos(tau/(tau+2)))
Z(36)
```

[7]Lenguaje de marcas para realidad virtual, por sus siglas en inglés: *Virtual Reality Mark-up Language.*

[8]Debería considerarse la celebración de los 500 años de esta publicación

[9]Ángulo interior que forman dos de sus caras

Para dibujar un pentágono se necesita una regla como esta:

```
(F++F++F++F++F++);
```

A la cual, para hacer más legible nuestra explicación, la abreviaremos llaméndole `p`.

Los movimientos para crear al dodecaedro son relativamente simples, el problema está en que la interpretación mental que logramos de los movimientos en el espacio puede no resultar fácil.

Primero vamos a trazar la cara de arriba y las cinco caras laterales a las que es adyacente:

```
[ p
  [>---p]f++
  [>---p]f++
  [>---p]f++
  [>---p]f++
  [>---p] ]
```

Aquí también hemos guardado el estado inicial de la tortuga, trazamos la cara de arriba y luego, para cada cara se coloca a la tortuga con la panza sobre la cara y se traza. El corchete final nos regresa al estado original.

Luego hay que dirigirse al vértice opuesto (respecto al centro del dodecaedro) y trazar desde ahí la otra mitad.

```
+++f--f<++f++f>--f--f<++
```

y, finalmente, trazamos la otra mitad:

```
[ p
  [>---p]f++
  [>---p]f++
  [>---p]f++
  [>---p]f++
  [>---p] ]
```

Tenemos entonces, un dodecaedro:

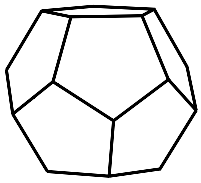


Figura 1: Dodecaedro

o, si sus caras fueran transparentes:

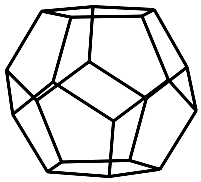


Figura 2: Dodecaedro

En este punto tenemos dos posibilidades: O substituimos manualmente a `p` por su equivalente `[F++F++F++F++F++]` o bien habrá que idear un mecanismo de substitución que permita hacerlo automáticamente. Este mecanismo nos lo ofrecen los *Sistemas–L*

4. Sistemas–L

4.1. Descripción

Vamos a suponer que el lector tiene cierto conocimiento de gramáticas formales.

Un sistema-L es entonces una gramática, independiente del contexto y cuyas reglas de substitución se obtienen *recursivamente*[10]. El sistema-L parte de una *palabra* a la que llamaremos *axioma*, que es similar al símbolo inicial de la gramática.

[10]La Real Academia Española, no reconoce la palabra *recursivo*, nos referimos a que las reglas se obtienen el Principio de Inducción Matemática

Vamos a desarrollar un ejemplo y con base en él daremos la descripción de cómo se escriben los sistemas-L.

4.2. Las curvas de Koch

La curva de Koch de orden cero es sencillamente un segmento de recta.

Figura 3: Curva K_0

La curva de Koch de orden uno, se construye a partir de la anterior, partiéndola en tres segmentos iguales, elminando el de enmedio y añadiendo dos segmentos de recta más que, junto con el segmento eliminado, formaría un triángulo equilátero.

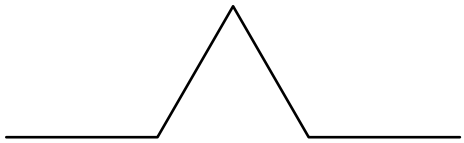


Figura 4: Curva K_1

La siguiente curva de Koch, se construye a partir de la anterior, partiendo cada uno de los segmentos que la componen en tres segmentos iguales, eliminando el segmento de enmedio y añadiendo otros dos, de manera que los nuevos segmentos y el eliminado forman un triángulo rectángulo:

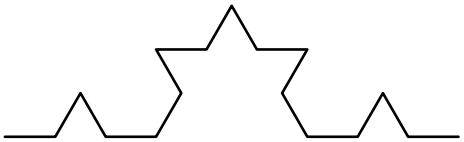


Figura 5: Curva K_2

Puede notarse el empleo del Principio de Inducción Matemática en la definición de la curva: Hemos dado la definición de la primera curva (*base*

de inducción). Luego, de suponer que se puede construir una curva dada (*hiótesis de inducción*), se define la siguiente en términos de la anterior (*paso de inducción)*. De esta manera, podremos construir cualquiera de las curvas. Dicho con el lenguaje de la tortuga, tenemos:

$$\begin{aligned} K_{n+1} &::= K_n + K_n - -K_n + K_n \\ K_0 &::= F \end{aligned}$$

De esta forma, y partiendo de la regla K_2, obtendríamos:

$$\begin{aligned} K_2 &\rightarrow K_1 + K_1 - -K_1 + K_1 \\ &\rightarrow K_0 + K_0 - -K_0 + K_0 + \\ &\quad K_0 + K_0 - -K_0 + K_0 - - \\ &\quad K_0 + K_0 - -K_0 + K_0 + \\ &\quad K_0 + K_0 - -K_0 + K_0 \\ &\rightarrow F + F - -F + F + \\ &\quad F + F - -F + F - - \\ &\quad F + F - -F + F + \\ &\quad F + F - -F + F \end{aligned}$$

Lo cual coincide con la curva que dibujamos arriba. Sin embargo, no sería muy fácil para un humano encontrar la expansión de la reglas anteriores para trazar la curva K_5, sin la ayuda de una computadora.

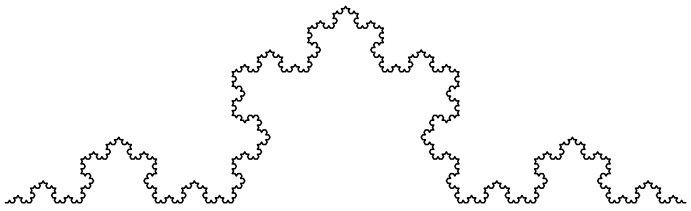


Figura 6: Curva K_5

4.3. Definición de los sistemas–L

Como puede verse en la descripción de la construcción de las curvas, los subíndices realmente no se están usando. Los índices de la derecha son todos iguales y su valor es el entero anterior al que da el índice de la izquierda. La otra posibilidad es que el índice de la izquierda sea cero. Con esta observación en la mente, podemos convenir en escribir el sistema así:

```
K  =  K + K - - K + K
K  =  F
```

En nuestro ejemplo anterior, el axioma es:

```
A(60) S(0.04) K
```

Lo que si es importante es saber cuántas veces va a aplicarse la primera regla. Adelantándonos un poquito, escribiremos el sistema de reglas así:

```
axiom  A(60)  S(0.04)   K;
rules 5
       K= K+K--K+K;
end
rules 1
      K= F;
end
```

Aquí dice que `K= K+K--K+K` representa a cinco reglas de producción que se aplican sobre el axioma. Se debe recordar que en nuestra notación se han omitido los subídices. Después de aplicar las reglas anteriores, se aplica `K= F`, que representa una regla solamente.

Suponiendo que las llaves { y } indiquen que el texto que contienen puede repetirse cero o más veces, entonces la sintaxis para escribir un sistema-L es:

```
{  axiom  texto; { rules n { letra= texto; } end }  }
```

Donde n es un número entero, *letra* es una letra del alfabeto inglés, *texto* es un texto cualquiera que va a ser el axioma o va subtituir a *letra*.

En caso de necesitar incluir un punto y coma (;) dentro del texto del axioma o de las reglas, se puede usar una secuencia de escape: \;.

A veces es necesario que cierta parte del texto del sistema-L no se expanda. Por ejemplo, si tuviéramos:

```
axiom
   A(acos(1/3))
   a;
rules 2
   a=(a);
end
```

La expansión nos da:

```
A(((a))cos(1/3))
((a))
```

Aunque tal vez nuestra intención era obtener:

```
A(acos(1/3))
((a))
```

Efectivamente, la expansión no respetó el nombre `acos`, encontró una a y se aplicaron las reglas. La expansión no tiene por qué respetar los nombres de las funciones porque ni siquiera los conoce, ésos los conoce la tortuga.

4.3.1. Protectores

Para proteger una cierta región del texto —es decir, para que la expansión la deje intacta— la delimitamos con el signo de número (#). En el ejemplo anterior, deberíamos escribir:

```
axiom
   #A(acos(1/3))#
   a;
rules 2
   a= (a);
end
```

para obtener lo que se desea.

Dos signos de número (##) juntos, indican que desde ese punto y hasta el final de la línea, hay un comentario. Los comentarios no se anulan, se copian al texto expandido, pero se les escribe con un sólo signo de número.

Por ejemplo:

```
axiom
   #A(acos(1/3))#  ## diedro del tetraedro.
   a;
rules 2
   a= (a);
end
```

se expande en:

```
A(acos(1/3) # diedro del tetraedro.
((a))
```

Si hubiera necesidad de incluir el carácter # dentro del texto de una regla o del axioma, puede usarse laa secuencia de escape \#.

4.4. Ejemplo: La curva del dragón

Hay una curva recursiva que se llama *la curva del dragón*, quizás se llame así porque parece el dibujo de las llamas que arroja un dragón (¿?). A continuación las mostramos sus reglas de construcción:

```
axiom
   #G(-0.65,-0.25) A(45) S(0.5)#
   d;
rules 16
   d= +d--i+;
   i= -d++i-;
end
rules 1
   d=F;
   i=F;
end
```

El axioma establece la posición para ver la curva centrada en una hoja de papel tamaño carta. Dice también que el ángulo de giro va a ser de 45^o. Luego dice que debe expandirse el símbolo d_{16} (en un momento explicaremos por qué).

Después, hay un conjunto de 32 reglas:

$$\left.\begin{array}{rcl} d_{k+1} & ::= & +d_k - -i_k+ \\ i_{k+1} & ::= & -d_k + +i_k- \end{array}\right\} \quad k = 16, 15, \ldots, 1.$$

Por último, hay otras dos reglas:

$$\begin{aligned} d_0 &::= F \\ i_0 &::= F \end{aligned}$$

que, por ser las últimas que se aplican, deben de tener los índice más pequeños. Esto explica por qué el axioma tiene al símbolo d_16. La gráfica del sistema-L anterior es ésta:

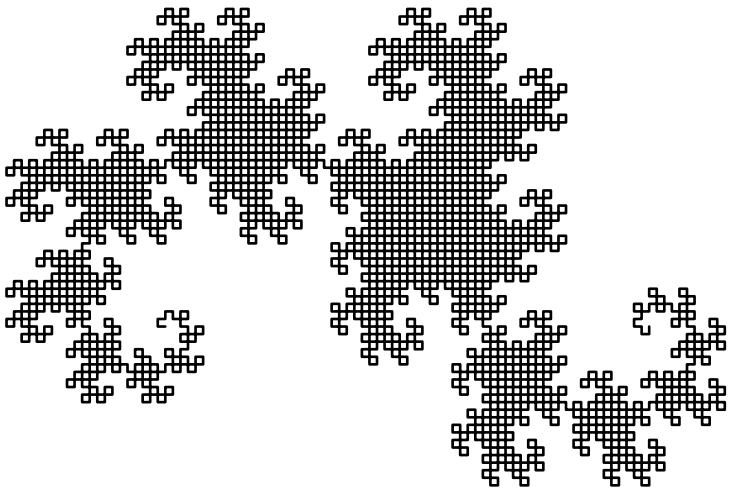


Figura 7: Curva d_{12}

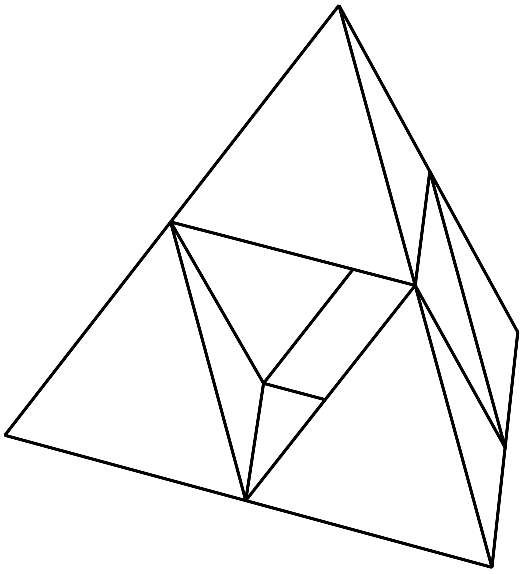
5. Poliedros recursivos

Ahora vamos a tratar un tema por demás obscuro y difícil: Así como la tortuga puede dibujar curvas *recursivas*, ¿podría la tortuga tridimensional trazar poliedros que se definan inductivamente? Una respuesta natural sería: Sí, usando un sistema-L.

5.1. El tetraedro de Sierpinski

Vamos a crear un poliedro compuesto de muchos tetraedros, al que llamaremos el Tetraedro de Sierpinski. Aunque Sierpinski no conoció este objeto tridimensional, lleva su nombre por un famoso triángulo que él diseñó. Procederemos de acuerdo a las reglas siguientes:

- El tetraedro de Sierpinski de orden cero, es el tetraedro platónico que todos conocemos.
- El tetraedro de Sierpinski de orden $k+1$ se forma con cuatro tetraedros de orden k. Los cuatro tetraedros de orden k se acomodan de manera tal que cada vértice coincida con un único vértice del tetraedro de orden $k+1$ como lo muestra la figura que sigue.



La idea para generar este tipo de figuras *debe* apegarse a la definición para que sea más fácil hacerlo.

Primero que nada, recordemos nuestro tetraedro:

```
t= (F+F+F+) > (F+F+F+) <f+>
   (F+F+F+) <f+> (F+F+F+) <f+ ;
```

que ahora hemos dispuesto en una sola línea y con la que definiremos una regla que hemos llamado t. Es decir, la regla anterior dice: *t se cambia por un tetraedro*. Algunas personas verían a t como una *macroinstrucción*, a las que suelen llamar *macros*.

Con esta idea, es relativamente sencillo construir un tetraedro de Sierpinski. Consideremos las instrucciones siguientes:

```
00        [ #S(S/2)#
01          t ff +
02          t ff +
03          t ff +
04          > f + f - <
05         t ]
```

Que deben entenderse así:

- Vamos a trazar un tetraedro de orden $k+1$.
- La línea 00 nos ayuda a regresar fácilmente al lugar a donde empieza el tetraedro de orden $k+1$, empilando el estado inicial de la tortuga. Además establece que el tamaño del paso para construir los tetraedros de orden k debe ser la mitad de lo que es para el de orden $k+1$.
- La línea 01 traza un tetraedro de orden k, ahora se requieren dos pasos para llegar al siguiente vértice del tetraedro de orden $k+1$, así que la tortuga los da y gira a la izquierda.
- La línea 02 traza otro tetraedro de orden k y procede al siguiente vértice.
- Lo mismo pasa en la línea 03. Debe observarse que la tortuga volvió al punto en donde empezó.

  Ahora va a dar una vuelta un poco enredada para llegar a la base del cuarto tetraedro.
- La línea 04 ordena a la tortuga colocar su panza sobre una de las caras laterales del tetraedro, avanzar sobre el plano de la base hasta la mitad de la arista, dar una vuelta de 120^o, subir, cancelar la vuelta anterior y poner la panza sobre el plano que define la base del cuarto tetraedro, que es paralela a la base de los otros tetraedros de orden k.
- La línea 05 traza al último tetraedro y, para hacer las cosas más rápido, volvemos al punto donde empezamos desempilando el estado inicial.

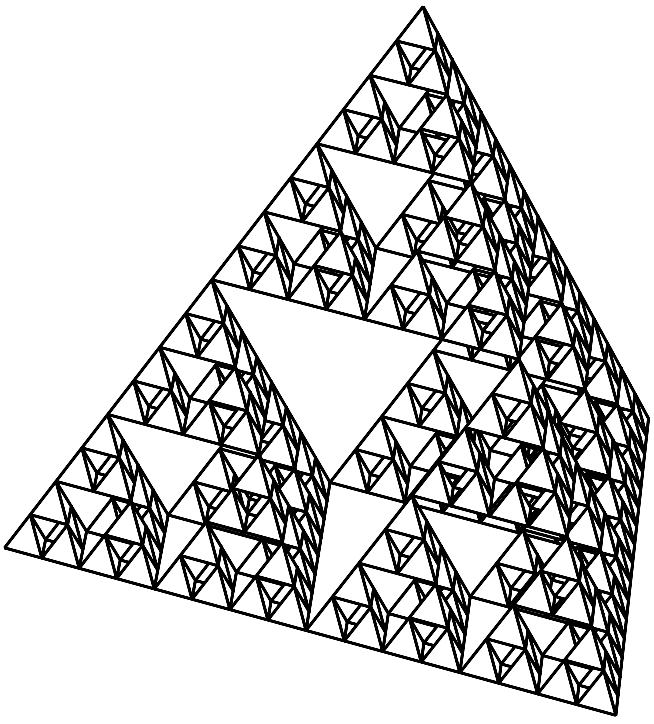
Con estas consideraciones podemos escribir entonces el sistema-L que traza al tetraedro de Sierpinski.

```
axiom
  #
   X(acos(1/3))
   Z(120)
   S(40)
  #
   t ;

rules 4
  t= [ #S(S/2)# t ff+ t ff
        + t ff + > f + f - < t ];
end

rules 1
  t=  (F+F+F+) > (F+F+F+) <f+>
      (F+F+F+) <f+> (F+F+F+) <f+ ;
end
```

La descripción es muy sencilla. Ahora solo falta verlo:



5.2. Advertencia

Debemos hacer notar un hecho que es muy significativo: La cantidad de información que tenemos al crear un poliedro recursivo.

Por ejemplo, el tetraedro manda trazar sus aristas una vez por cada triángulo que tiene, por lo tanto, cada arista se traza dos veces; lo mismo pasa con los vértices, que se dan una vez por cada cara, así que cada vértice se da tres veces.

Si uno hace cuentas, el número de vértices, de aristas o de caras del tetraedro de Sierpinski crece exponencialmente. Hay forma de simplificar las componentes gráficas de los poliedros, pero, por la naturaleza del problema, pueden tardar mucho tiempo en ser ejecutados.

6. Los programas

Los programas son de uso muy simple. Todos ellos toman un archivo y sacan sus resultados por su salida estándar (excepto `geom` que no tiene nada que escribir).

Cuando el archivo de entrada se omite, suponen que la información viene por la entrada estándar, de esta manera pueden usarse como filtros.

6.1. LSystems

Este programa toma un texto en donde se describe un sistema–L y lo expande, eso es todo. Los archivos de texto que toma deberían escribirse con un editor de texto ASCII (notepad, emacs, vi, etc.)

Este programa se usa para expandir los sistemas-L en donde hay instrucciones para las tortugas (la bidimensional y la tridimensional). El programa revisa sintaxis y manda mensajes por la salida estándar de errores en caso de que encuentre alguno. Los mensajes ubican al error dando el número de línea donde se encuentra.

6.2. 2DTurtle

Este programa toma un texto con instrucciones para tortuga bidimensional y escribe en su salida estándar un la descripción del dibujo en POSTSCRIPT que puede usarse para imprimir el dibujo que la tortuga traza. Tam-

bién es posible mandar esta salida hacia un archivo y de ahí convertirlo a un cierto formato gráfico.

6.3. 3DTurtle

Este programa toma un texto con instrucciones de para tortuga tridimensional y escribe en su salida estndar la descripción de la escena que trazó la tortuga en un formato al que llamaremos `.geo`. La salida puede enviarse a un archivo para que un programa de despliegue gráfico la tome y la procese.

6.4. Programas de despliegue gráfico tridimensional

6.4.1. vrml

Este programa toma un texto en formato `.geo` y lo convierte en formato `vrml`. El archivo generado puede desplegarse con un navegador de Internet y con él uno puede viajar y explorar el dibujo de una manera muy impresionante.

6.4.2. hidebw

Este programa toma un texto en formato `.geo` y lo convierte en formato POSTSCRIPT, eliminando las líneas ocultas en la escena. El archivo generado puede imprimirse o convertirse en PDF con relativa facilidad.

6.4.3. anaglyph

Este programa toma un texto en formato `.geo` y lo convierte en formato POSTSCRIPT, generando un anaglifo que puede verse con lentes azules/rojos, eliminando las líneas ocultas en la escena. El archivo generado puede imprimirse o convertirse en PDF.

6.4.4. stereo

Este programa toma un texto en formato `.geo` y lo convierte en formato POSTSCRIPT, convirtiéndolo en estereograma de puntos aleatorios. El archivo generado puede imprimirse o convertirse en PDF.

7. Uso de los programas

El problema más grave que debe enfrentarse ahora es el de describir cómo usar los programas bajo un sistema operativo dado. De ahora en adelante, todas las instrucciones que se den corresponden al sistema operativo Windows, para las versión 98, o superior (ME, NT, 2000, 2003, XP, etc.), pero los programas pueden usarse bajo el sistema operativo UNIX y sus distintas versiones, pues, como se dijo en la introducción, los programas se desarrollaron usando herramientas de uso universal.

Se ha tratado de simplificar el procesamiento de los sistemas–L, por eso se ha confinado al sistema a un directorio que se llama `3dt`. Dentro de este directorio, el subdirectorio `bin` contiene todos los programas ejecutables del sistema y el directorio `work` se usa como directorio de trabajo. Hay un otro directorio que se llama `doc` con los manuales en formato PDF.

Para empezar, el usuario deberá abrir una ventana de *símbolo de sistema* (modo MS-DOS) y dirigirse al directorio `3dt\work`, una vez ahí, tendrá que moverse a uno de los directorios `2DTurtle` o `3DTurtle`, dependiendo si quiere desarrollar un proyecto bidimensional o tridimensional.

7.1. 2DTurtle, para sistemas bidimensionales

Una vez en `2DTurtle` debe crear un directorio para hacer un proyecto nuevo. Dentro de ese directorio, se crea un archivo de texto con el sistema–L que se desee expandir en dos dimensiones. Supongamos que el directorio se llama `direc` y el archivo `arch.txt`.

Volviendo al directorio `3dt\2DTurtle`, el usuario encontrará un archivo *batch* que se llama `L2D` y que deberá de invocar así:

```
L2D direc\arch
```

Obsérvese que no se escribió la extensión `.txt`, sólo el nombre del archivo. Una vez que se ejecuta esa instrucción, dentro del directorio `direc` aparece el archivo `arch.pdf` y es es todo.

En caso de haber algún error, el sistema lo informará con un mensaje adecuado, ubicando el error en el archivo `arch.txt`.

7.2. 3DTurtle, para sistemas tridimensionales

De maneera análoga, si el usuario se dirige al directorio `3DTurtle`, encontrará ahí algunos archivos *batch* que se describirán más adelante. El usuario debe crear un directorio para hacer un proyecto nuevo, supongamos de nuevo que ese directorio se llama `direc`.

Dentro de ese directorio, se crea un archivo de texto con el sistem–L que se desee expandir en tres dimensiones. Supongamos que el directorio el archivo se llama `arch.txt`.

Volviendo al directorio `3dt\2DTurtle`, hay varios *scripts* que pueden ejecutarse:

L `L direc\arch`

Este archivo *batch* expande al sistema–L que contiene `arch.txt`. En el directorio `direc` aparece el archivo `arch.geo` que contiene la descripción tridimensional del objeto que obtuvo el sistema. En caso de haber errores en la descripción del sistema–L o en las instrucciones de la tortuga, el sistema lo informará.

V `V direc\arch`

Este archivo *batch* convierte a `arch.geo`, en el archivo `arch.wrl`, en formato VRML, mismo que puede verse con tan solo hacer clic sobre su icono, una vez que el visualizador de realidad virtual (*Cortona*, por ejemplo) esté instalado en su computadora. Si se usa *Cortona*, podría sugerírse presionar el botón `fit` y el botón `study` y luego, con el botón izquierdo del ratón presionado, mover la figura que aparece en la pantalla.

G `G direc\arch`

Este archivo *batch* puede usarse para ver la imagen tridimensional de la escena creada por el sistema–L. La ventana de despliegue es pequeña, pero puede ajustarse al gusto del usuario. No es un visualizador tan profesional como *Cortona*, pero es bastante bueno para exhibir los resultados del sistema–L, de una manera rápida.

El programa se maneja con el teclado y se dispone de las siguientes funciones:

`ESC` La tecla `ESC` permite abandonar el programa.

$+$ La tecla $+$, ya sea del teclado numérico o del teclado normal, aleja a la cámra de la escena.

$-$ La tecla $-$, ya sea del teclado numérico o del teclado normal, acerca a la cámara a la escena.

$\rightarrow$ La flecha derecha, mueve la cámara del observador a la derecha, alrededor de la escena.

$\leftarrow$ La flecha izquierda, mueve la cámara del observador a la izquierda, alrededor de la escena.

Los que siguen son programas que toman una *fotografía* de la escena, por lo tanto, se necesita ubicar a la cámara que la está enfocando. Necesitaremos para ello un archivo que dé los parámetros de la cámara. Si el archivo del sitema–L se llama `arch.txt`, el archivo de su cámara debe llamarse `arch.cam` y debe ser un archivo de texto.

Dentro del archivo de texto, pueden darse los parámetros siguientes, debe notarse que el nombre del parámetro debe separarse de su valor mediante un espacio en blanco (por lo menos).

- `position`␣(x, y, z)
  Donde (x,y,z) son las coordenadas en las que está ubicada la cḿara
- `look_at`␣(x, y, z)
  Donde (x, y, z), son las coordenadas del punto al que la cámara tiene dirigida la mirada (si se omite, se supone que será el origen.
- `sky`␣(x, y, z)
  Donde (x, y, z), es el vector que apunta hacia el cielo. Si se omite, se supone que es $(0, 0, 1)$.
- `distance`␣d
  Donde d, es la distancia que hay entre el observador y la pantalla de proyección en donde la cámara va a proyectar la imagen. Si se omite se supone que vale 40cm.
- `sizex`␣x
  Donde x, es el ancho, en centímetros, del papel en donde va a imprimirse la foto. Si se omite, se supone que será de 21.59cm (tamaño carta, 8.5″).

- `sizey␣`y
  Donde y, es el largo, en centímetros, del papel en donde va a imprimirse la foto. Si se omite, se supone que será de 27.94cm (tamaño carta, 11″).

Ahora continuamos con los programas que utilizan cámara.

S `S direc\arch`

Este programa calcula, a partir de `arch.geo`, un esteregrama de puntos aleatorios desde el punto de vista de la cámara. El archivo en formato PDF de dicha imagen queda en `direc\arch-st.pdf`.

H `H direc\arch`

Este programa calcula, a partir de `arch.geo`, una proyección de la escena vista desde la cámara, donde se eliminan las íneas ocultas de la misma. El archivo en formato PDF de dicha imagen queda en `direc\arch-hi.pdf`.

A `A direc\arch`

Este programa calcula, a partir de `arch.geo`, dos proyecciones de la escena en un anaglifo desde el punto de vista de la cámara. En el archivo `direc\arch-an.pdf` queda la imagen en formato PDF. Es posible también dar un archivo de descripción de cámara especial para los anaglifos, si se crea y se le llama: `direc\arch-an.cam`.

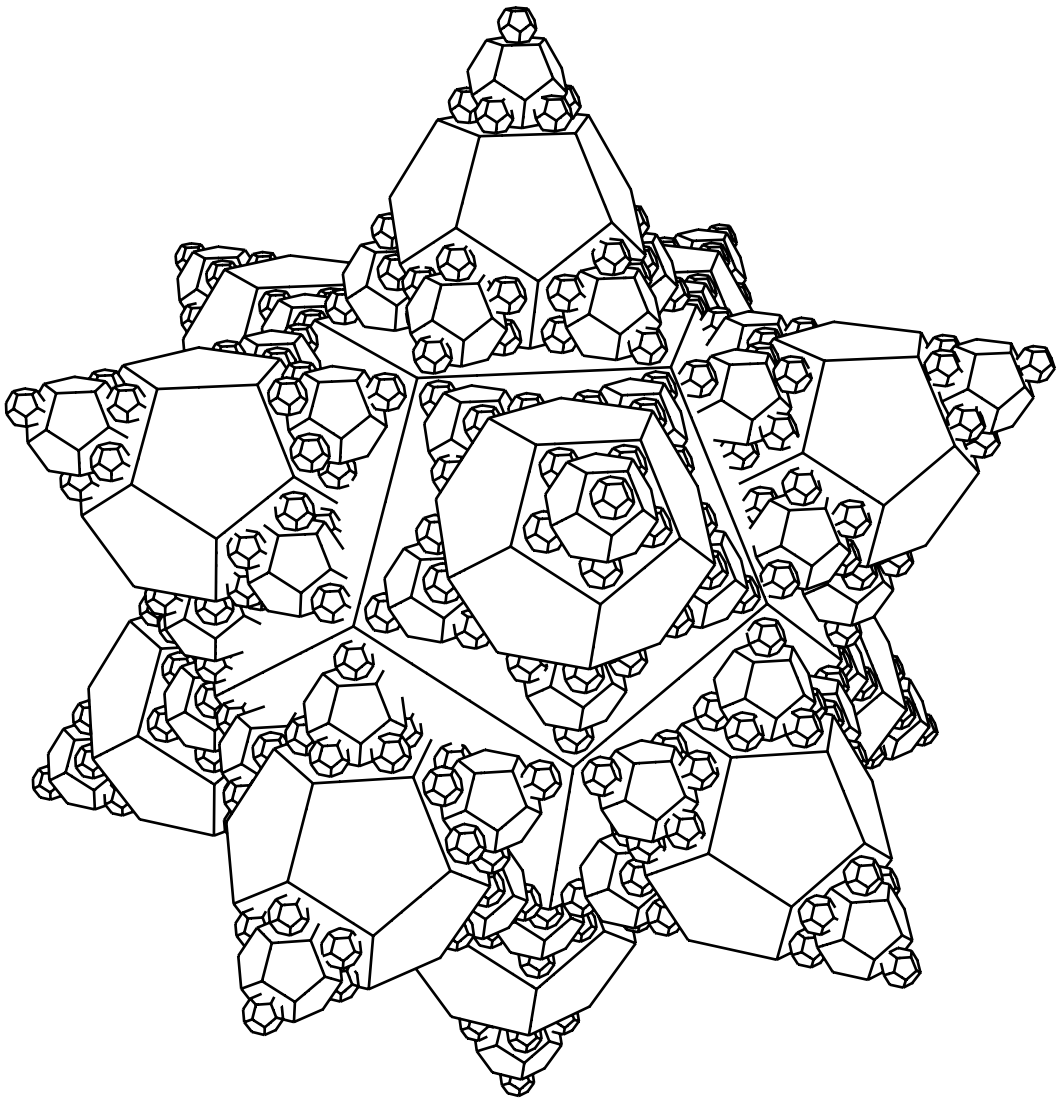
Para ver un anaglifo es necesario usar lentes azul/rojo.

8. Ejemplos

Hay varios ejemplos en el directorio examples del disco de distribución. Todas las imágenes que generan los ejemplos están en un directorio que se llama `Gallery`.

9. Observaciones y mejoras futuras

- La tortuga tridimensional debe dibujar a color.
- ¿Por qué nos detenemos en un espacio tridimensional? Puede pensarse en una tortuga en espacios de dimensión superior. Por ejemplo, en $\mathbb{R}^4$, (que es un espacio de dimensión 4) existe un poliedro cuyas 120 *facetas* son dodecaedros regulares. No podemos verlo, pero sí podemos proyectarlo a un espacio de dimensión 3 y ver esa proyección en una proyección en el plano.
- El proceso de expansión de los sistemas-L no está ligado al lenguaje de la tortuga, se ha usado el programa de expasión para generar programas en C.
- Los algoritmos para eliminar líneas ocultas deben mejorarse, de manera que puedan calcular la intersección de dos polígonos y los corten adecuadamente. Ésta es, básicamente, la razón por la que la tortuga tridimensional no puede dibujar a color.



10. Agradecimientos

- El autor agradece al IPN la facilitación del equipo de cómputo en donde se crearon estos programas.
- A todos los autores de Software Libre, por su amable disposición

de entregar gratuitamente el fruto de su trabajo para el bien de la Humanidad.

- A Paul McCartney, que le puso pista de sonido a las interminables horas de programación y depuración.
- A la Sra. N, esposa del autor, quien brindó su apoyo y comprensión, sirvió de inspiración y le hizo la vida sencilla cuando se desarrollaron los programas.
- A Cinemas Lumiere, por facilitar gratuitamente los lentes para ver anaglifos.