1 画像処理プログラミング

1.1 タートルグラフィックス

ペンをもったタートル (亀)

コンピュータのディスプレイ上に図形を描画する手法の一つに**タートルグラフィックス** (turtle graphics) がある．これは，キャンバスにみたてた描画領域 (ディスプレ上の 1 つのウィンドウ) を，ペンをもったタートル (亀) を歩かせることで図形を描く方法である．タートルには，「前へ進め」，「右を向け」，「ペンをあげろ」などと命令することができ，タートルが上げ下げしながら歩いた軌跡がさまざまな図形となる．

IPU-Scheme では，(open) を評価するとキャンパスに相当する図 1.1 のウィンドウが生成され，タートルグラフィックスをはじめられるようになる．

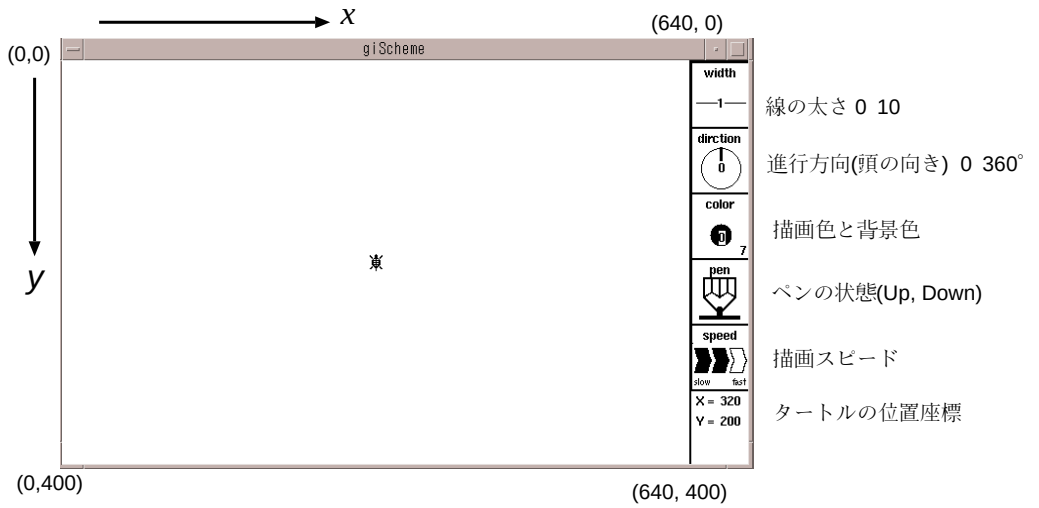


図 1.1 IPU-Scheme のキャンバス

画面中央にあるのがタートルである．右側には上から順に次の項目が並んでおり，項目ごとに属性 (太さ，角度，色など) を変更するための命令 (手続き) が用意されている．

ペンの太さ (描画線の太さ)	ペンの太さは (width w) によって変更可能である．ただし，$w = 0, 1, 2, \cdots, 10$.
タートルの進行方向 (頭の向き)	表示される数値は 0　360° であり，北 (真上) が 0° である．方向を変えるための命令は後述する．
描画色 (円)，背景色 (四角)	描画色 fg，背景色 bg に変更するためには (color fg bg) を用いる．なお，色はつぎの数で指定される 8 色のいずれかとする． 0:黒，1:青，2:緑，3:水色，4:赤，5:紫，6:黄，7:白
ペンの状態	ペンを上げ・下げするために (penup) と (pendown) を用いる．
移動スピード	タートルが移動するスピードは，(slow) と (fast) によって，それぞれ 1 段階ずつ低速・高速になる．
タートルの位置 (座標)	タートルの現在位置は, キャンバスの左上を原点とする xy-座標系で表される．ただし，$0 \leq x \leq 640$, $0 \leq y \leq 400$．タートルを移動するための命令は後述する．なお，キャンバスの中央にあたる座標

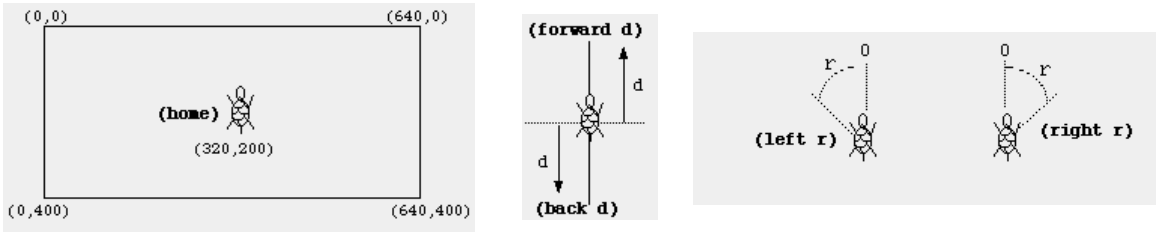
$(320, 200)$ をホーム (位置) とよぶ.

キャンバス (ウィンドウ) を操作するための命令はつぎのとおりである.

`(open)`	キャンバスの生成
`(close)`	キャンバスの消滅
`(cls)`	キャンバス上の図形の消去とタートルのホーム $(320, 200)$ への移動

キャンバスの歩き方

タートルは図 1.2(a) のような座標系のキャンバス上を歩く．左上のコーナーが原点であり，右下のコーナーが x 軸と y 軸，それぞれの最大値にあたる．IPU-Scheme の場合, キャンバスのサイズは変更できない．なお，初期画面において，タートルはキャンバスの中央，すなわち，ホーム $(320, 400)$ にいる．



(a) 座標系 (b) 前進・後進手続き (c) 方向転換手続き

図 1.2 タートルグラフィックスの手続き (命令)

タートルは，現在の位置から，図 1.2(b) の手続きによって前進や後進をし，同図 (c) の手続きによって方向を変えることができる．これらに加えてつぎの手続きが用意されている.

- タートルの移動

`(home)`	ホーム $(320, 200)$ へ移動．ペンが下がった状態でも移動中に描画はしない.
`(forward` d`)`	向いている方向へ d だけ前進．ペンが下がった状態では移動中に描画する. `(go` d`)` でも同じ.
`(back` d`)`	向いている方向から d だけ後退．ペンが下がった状態では移動中に描画する.
`(jump` x y`)`	座標 (x, y) へ移動．ペンが下がった状態でも移動中に描画はしない.

- タートルの方向転換

`(right` r`)`	向きを右回りに r° 変更.
`(left` r`)`	向きを左回りに r° 変更.
`(rotate` r`)`	$r \geq 0$ のとき右回りに r° 変更．$r < 0$ のとき左回りに r° 変更.
`(north)`	北 (0°) を向く.
`(south)`	南 (180°) を向く.
`(east)`	東 (90°) を向く.
`(west)`	西 (270°) を向く.

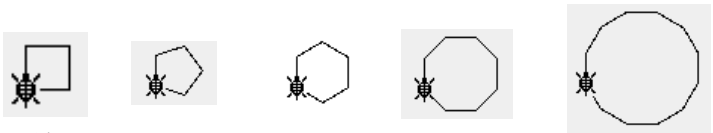
【例 1.1】 正方形の描画

次の項目をすべて満たしながら，図 1.3(a) の正方形 (正四角形) を描く手続き `(suqare x)` はつぎのように定義される．そのときに,

- 描き始める最初にペンを下ろし，描き終えたあとにはペンを上げる.
- 正方形の一辺の大きさは x.

- 正方形を描き終えたタートルの向きは，北 (０度).
- 再帰呼び出しは使わない.

```
(define (square x)
    (begin (pendown)
           (forward x) (right 90)
           (forward x) (right 90)
           (forward x) (right 90)
           (forward x) (right 90)
           (penup)                ))
```



(a) 正方形 (b) 正五角形 (c) 正六角形 (d) 正八角形 (e) 正十二角形

図 1.3 多角形

[問 1.1]
例 1.1 を再帰呼び出しを使った手続き `(square-rec x)` として定義しなおしなさい.

【例 1.2】　図形の繰り返し描画

例 1.1 の正方形の描画では，“`(forward x) (right 90)`” が4回繰り返されている．もし，これら 2 つの手続き `forward` と `right` を n 回続けて評価する手続き `(repeat-f-r` x d n`)` があれば，`square*` はつぎのように簡潔に定義される．ここで，x と d は，それぞれ，“前進する距離” と “右回りの変更角度” である.

```
(define (f-r x d)
   (begin (forward x) (right d)))

(define (square* x)
   (begin (pendown) (repeat-f-r x 90 4) (penup)))
```

このとき，`repeat` はつぎのように定義される．なお，この定義の場合，繰り返される手続きは *proc* 2 つの引数を必要とするものに限られる[1)].

```
(define (repeat-f-r x d  n)
   (cond ((<= n 1) (f-r x d))
         (else (f-r x d) (repeat-f-r x d (- n 1)))))
```

[問 1.2]
図 1.3 (b)　(e) について，以下の問に答えなさい.

i) 図 1.3 (b) の正五角形を描画する手続き `(pentagon` x`)` を定義しなさい．ここで，x は一辺の長さとする.
ii) 図 1.3 (c) の正六角形を描画する手続き `(hexagon` x`)` を定義しなさい．ここで，x は一辺の長さとする.
iii) 図 1.3 (d) の正八角形 (octagon) や同図 (e) の正十二角形 (dodecagon) も正五角形や正六角形と同様にして描くことができよう．そこで，一辺が x の正 n 角形を描く `(polygon n x)` を定義せよ.
ヒント：`repeat-f-r` を活用するとよい.

1) この定義の仕方では，必要とされる引数の個数毎に repeat を定義しなければならず，好ましくない.

1.2 関数のグラフ描画

プログラミング言語の中には，ウィンドウ上に点，直線，長方形，円など基本的な図形を描くための関数 (手続き) として，図形の属性値 (端点や中心の座標と線分の長さなど) を引数として与える形式のものが用意されている．代表的なものは BASIC の line や box である．IPU-Scheme にもつぎのような手続きが用意されている．

`(pset` x y`)`	(x, y) に点を描画
`(preset` x y`)`	(x, y) の点を消す
`(line` $x0$ $y0$ $x1$ $y1$`)`	$(x0, y0)$ から $(x1, y1)$ への線分を描画
`(box` $x0$ $y0$ $x1$ $y1$`)`	$(x0, y0)$ から $(x1, y1)$ を囲む長方形を描画
`(boxf` $x0$ $y0$ $x1$ $y1$`)`	$(x0, y0)$ から $(x1, y1)$ を囲む長方形を描画し，内部を塗りつぶす．
`(circle` x y r`)`	中心を (x, y) 半径を r とする円を描画
`(circlef` x y r`)`	中心を (x, y) 半径を r とする円を描画し， 内部を塗りつぶす．

【例 1.3】 xy-平面の座標軸表示

キャンバス上に図 1.4 のような xy-平面の座標系のための x 軸と y 軸を表示しよう．

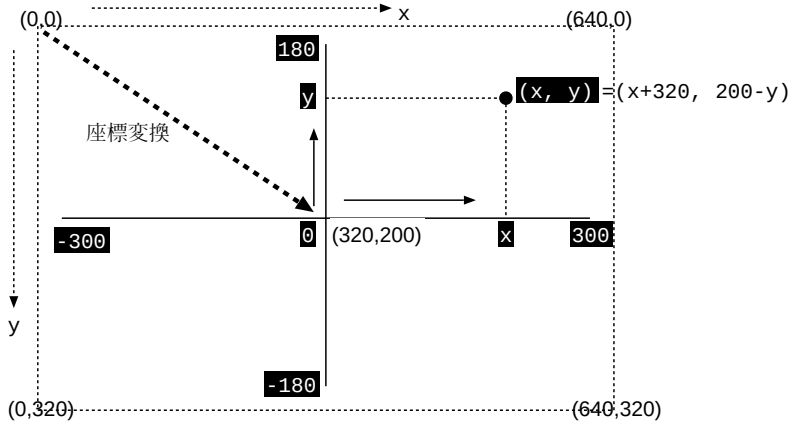


図 1.4 キャンバス座標系と xy-平面座標系

ここで，注意すべきことは，つぎの 2 つである．

- キャンバス座標系の原点 (左上のコーナー) が xy-座標系では中央 (ホーム位置) に対応する
- キャンバス座標系の y 軸が下に向かって増加するのに対し，xy-座標系では上に向かって増加する．

よって，xy-座標系の点 $\boxed{(x, y)}$ は，キャンバス座標系の $(x + 320, 200 - y)$ に対応させるとよい．そこで，x 軸の範囲を -300　300，y 軸の範囲を -180　180 として，軸を描く手続きはつぎの `axies` となる．また，xy-座標系の点 $\boxed{(x, y)}$ をプロットする手続き `(plot x y)` はつぎのようになる．

```
(define (axies)
   (begin (cls)
          (line 20 200 620 200)
          (line 320 20 320 380)))

(define (plot x y)
   (circlef (+ x 320) (- 200 y) 1) 0)
```

```
(define *black* 0)      (define *blue* 1)
(define *green* 2)      (define *water* 3)
(define *red* 4)        (define *purple* 7)
(define *yellow* 6)     (define *white* 7)
```

ここで，点は半径 1 の円としている[2)]また，`*black*`, `*blue*` は点の色の設定を，数字ではなく名前で行うための定義である．これらを用いれば，たとえば，`(color *red* *white*)` を評価したのち，`(plot x y)` で赤い点が表示される．

【例 1.4】　関数のグラフの描画

第**??**章の問**??**では，2 次関数 $f(x) = x^2+x+1$ における $f(1), f(2), f(3), \ldots$ を印字する手続き `f-print` を作成した．ここでは，xy-平面上にグラフとして描く `f-graph` を定義してみよう．`graph-f` は，`f-print` における `(display ...)` の箇所を xy-平面上への点 (小さな円) を描画する手続き `(circle ...)` に変更することで実現される．描画にあたっては，x を min から max までを間隔 (刻み幅) dx で変化させ，その時々の $f(x)$，すなわち，$f(min), f(min+dx), f(min+2dx), \ldots, f(max)$ の値とする．そのため，dx, min, max は `f-graph` の引数とする．

```
(define (f-graph dx min max)
   (cond ((>= min max) (plot max (f max)));
         (else    (plot min (f min))
                  (f-graph dx (+ min dx)  max))))
```

[問 1.3]

$f(x) = 3x + 1$ を赤い点で $-100 \leq x \leq 100$ の範囲で，$f(x) = \frac{1}{200}x^2 - 10$ を赤い点で $-200 \leq x \leq 200$ の範囲で，それぞれ間隔 (刻み幅) を 1 として描け．

[問 1.4]

正方形から始めて，正五角形，正六角形，正八角形，正十二角形と描いてみると，だんだんと円に近づいた図形ができあがっていることがわかる．さらに，正 24 角形，正 36 角形，…，というようにしていけば，より円に近づいた図形が得られる．そこで，`(repeat-f-r` x d p`)` を利用して円を描くことにする．このとき，以下の問に答えなさい．

i) 正 p 角形を描くとき，d はいくつにすればよいだろうか．p を使った式を答えなさい．

ii) x はタートルが 1 回に移動する距離で，それを p 回繰り返す．このとき描画される多角形が円の近似だとするとき，その半径を求める式を x, p を使って答えなさい．

iii) 半径 75 の円を近似する多角形を描いてみるために，p を $12, 36, 72, 90, 180, 360$ と変化させるときの移動距離 x と方向変換の角度 d をそれぞれ求め，次表を完成させなさい．なお，数値は小数第 1 位まで求めなさい．

p	12	36	72	90	180	360
x						
d						

iv) 前問の表にしたがって，`(repeat-f-r` x d p`)` を評価し，6 つの多角形を描きなさい．なお，正整数のみを実引数とすること．

v) 中心 (x, y)，半径 r の円を描く手続き `(circle` x y r`)` を使って，前問で描いた多角形の中心を求め，それを中心とした半径 50 の円を描きなさい．ただし，円は赤色で描画すること．

vi) これまで描かれた図形をみながら，多角形と円の違いについてわかることを答えなさい．

[2)] 手続き `pset` を用いると点が小さく見づらいため．

1.3 自己相似図形

1.3.1 相似比と次元

紙のサイズには A4, A3, A2 などという規格がある．これらは相似な図形で，たとえば，A4 は A3 を二つ折りにした大きさ (面積) であって，相似比は $\sqrt{2}$ である[3]．つまり 2 次元図形の場合，大きさ (面積) の比は相似比の 2 乗にあたる．

一方，1 次元図形では相似な 2 つの線分の大きさ (長さ) の比は相似比の 1 乗にあたる．そして，3 次元図形では相似な 2 つの立体の大きさ (体積) の比は相似比の 3 乗にあたる．

このように，n 次元[4] の相似な図形の大きさの比は相似比の n 乗にあたる．すなわち，

$$(\text{相似比})^n = \text{大きさの比}$$

いいかえると，2 つの図形の大きさが相似比の n 乗であることがわかれば，それらの図形の次元 n が求められる．

1.3.2 フラクタル

ある図形において，部分が全体と似ている形（相似）であるとき，その図形を**自己相似図形**といい，とくに，次元が自然数でないときに**フラクタル** (fractal) とよぶ．

たとえば，図 1.5 (a) は，「長さ d を三等分したのち，中央の線分の部分を線分の長さを一辺とする山片[5]に描く」ことで得られる．この図形の長さは $\frac{d}{3} \times 4$ になる．そして，同図 (b) は，同図 (a) の各線分に対して，「長さを三等分したのち，中央の線分の部分を線分の長さを一辺とする山片に描く」ことで得られる．この図形の長さは $\frac{d}{3^2} \times 4$ になる．このような図形は，**コッホ曲線** (koch curve) とよばれる．

このとき，たとえば，同図 (b) の部分は同図 (a) と相似であり，同図 (b) の 1 つの線分は同図 (a) の 1 つの線分の $\frac{1}{3}$ なので，相似比は 3 である．また，同図 (b) の大きさは同図 (a) の大きさ (長さ) の 4 倍であり，つぎのようにして次元 n が求められる．

$$3^n = 4 \quad \text{の両辺の対数をとり，} n\log 3 = \log 4 \quad \text{より，} n = \frac{\log 3}{\log 4} \fallingdotseq 1.26$$

このようにコッホ曲線の次元は 1.26 であり，フラクタルであることがわかる．

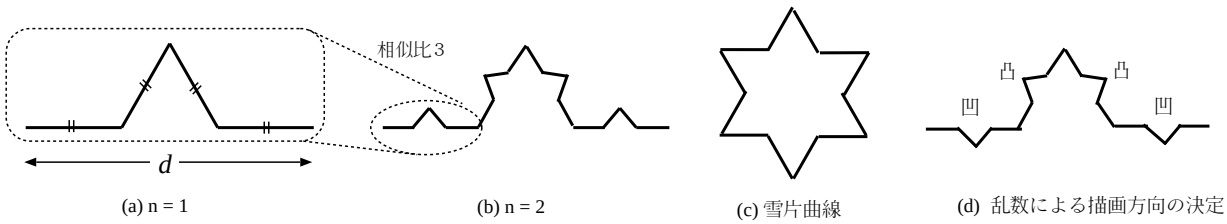


図 1.5 フラクタルの例：コッホ曲線

【例 1.5】 コッホ曲線描画の手続き

---

[3] A4 の縦と横をそれぞれ $\sqrt{2}$ 倍すれば，A3 が得られる．

[4] 次元は自由度に相当する．たとえば，直線 (曲線も同様) の場合，1 つの変数により (大きさ) を表すことができる．一方，平面 (曲面も同様) の場合は 2 つの変数により表され，立体 (空間) は 3 つの変数により表される．

[5] 線分を一辺とする底辺なしの三角形に相当

長さ d で，相似比 x のコッホ曲線をタートルグラフィックスで描く場合，相似比が 1 のときにはタートルが距離 $\frac{d}{3}$ を線分とするように前進しながら図 1.5(a) を描き，$d > 1$ のときには各線分に対して再帰的 (自己相似的) に図 1.5(a) を描いていけばよい．これにしたがう手続き `koch` はつぎのように定義される．

```
(define (koch x d)
   (cond ((<= x 0) (forward d))
         (else  (koch (- x 1) (/ d 3)) (right -60)
                (koch (- x 1) (/ d 3)) (right 120)
                (koch (- x 1) (/ d 3)) (right -60)
                (koch (- x 1) (/ d 3))  )))
```

さらに，描画のための前処理 (画面のクリア，始点の設定等) をするための手続き `koch-start` をつぎのように定義しておけば，さまざまな相似比のコッホ曲線を描くことができる．

```
(define (koch-start n d)
   (begin (cls) (jump 50 250) (east) (pendown)
          (koch n d) ))
```

[問 1.5]
長さ $d = 550$ として，相似比 $x = 1, 2, 3, 4, 5$ について，コッホ曲線を描いてみよ．

[問 1.6]
図 1.5(a) の図形を 3 つ使って，同図 (c) のように結合すると，雪の結晶を近似したような閉曲線がえられる．この曲線は**雪片曲線**あるいは**コッホ島**とよばれる．
`(koch n d)` を利用しながら，雪片曲線を描く手続き `(snowflake n d)` を定義せよ．
ヒント：図 1.5(a) の図形を一辺とする正三角形を構成するとよい．

【例 1.6】　乱数を用いたコッホ曲線描画

手続き `(koch` n d`)` における n を大きくすればするほど，得られるコッホ曲線は海岸線に似てくる．ただし，手続き `(koch` n d`)` における三角 (底辺なし) は常に一方 (上方向) に飛び出した 凸 形状である．より自然の海岸線 (たとえば，リアス式海岸) に近づけるためには，三角 (底辺なし) を両方 (上下方向) に飛び出した凸凹形状にするとよい．そこで，手続き `koch` を呼び出したとき，凸あるいは凹のいずれに描くのか乱数を使って決めることにする．IPU-Scheme の場合，つぎの手続き `random` を使えば，0 $(n-1)$ の (1 つの) 自然数が得られる．その値は `random` を評価するたびに異なり (同じ数が再びでることもありえる)，得られる自然数の順番に規則性がないことから乱数とよばれている[6]．

$$\texttt{(random } n\texttt{)} \Longrightarrow (0 \quad n-1) \text{ の自然数}$$

ここでは，0　9 を範囲とする乱数を発生し，5 未満であれば凹，5 以上であれば凸のコッホ曲線を描くことにする．そのために，つぎのような手続きを定義する．

```
(define (koch-rand-start n d)   ;  画面の準備
    (begin (cls) (jump 50 200) (east) (pendown)
           (koch-rand n d) ))

(define (koch-rand n d)         ;  乱数を使ったコッホ曲線
    (if (< (random 10) 5) (koch-down n d) ; 乱数が 5 未満なら凹を描く
         (koch-up n d)))                  ;        5 以上なら凸を描く

(define (koch-up n d)           ;  凹 のコッホ曲線
```

[6] プログラムによって生成される乱数は，ある数式にしたがっており，厳密な意味で不規則性を保証しておらず，**疑似乱数**とよばれている

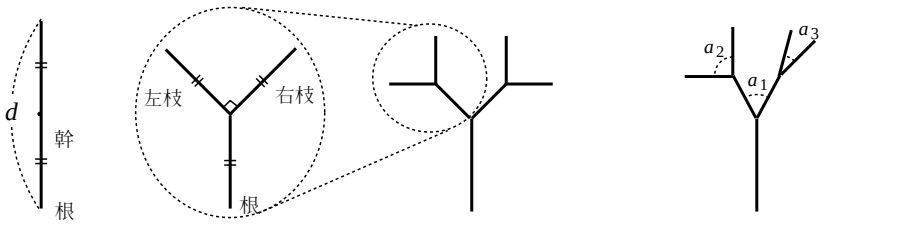
```
    (cond ((<= n 0) (forward d))
          (else  (koch-rand (- n 1) (/ d 3)) (right -60)
                 (koch-rand (- n 1) (/ d 3)) (right 120)
                 (koch-rand (- n 1) (/ d 3)) (right -60)
                 (koch-rand (- n 1) (/ d 3)) )))

(define (koch-down n d)          ;  凸 のコッホ曲線
    (cond ((<= n 0) (forward d))
          (else  (koch-rand (- n 1) (/ d 3)) (right 60)
                 (koch-rand (- n 1) (/ d 3)) (right -120)
                 (koch-rand (- n 1) (/ d 3)) (right 60)
                 (koch-rand (- n 1) (/ d 3)) )))
```

たとえば，(koch-rand-start 5 550) を評価すると，乱数により凸凹のあるコッホ曲線が描かれる．

【例 1.7】　樹木曲線

図 1.6(a) の 1 本の幹がのび，幹の中心から同図 (b) のように 90° 開くように枝分かれをしたら Y 字型の木が描かれる．枝分かれした各枝が幹になり，それぞれで再帰的に枝分かれすると同図 (c) のような樹木が得られる．



(a) 幹と根　(b) 枝分かれ n=1　(c) 枝分かれ n=2　(d) 乱数による枝分かれ n=2

図 1.6　フラクタルの例：樹木曲線

樹木曲線を描くには，タートルをつぎのように動かせばよい．n は繰り返しの回数 (木の成長の回数に相当)，d は最初の幹の長さであり，a は枝分かれの角度とする．

1) 根から真上 (現在向いている方向) へ $\frac{d}{2}$ 進む (幹が成長する)．
2) 左枝がのびる方向を向く ($\frac{a}{2}$ 度左を向く)．
3) 現時点を根として，幹の長さを $\frac{d}{2}$ の左枝 (の樹木) を再帰的に描く (n は 1 回分減じる)．
   ただし，$n = 1$ なら真上 (現在向いている方向) へ $\frac{d}{2}$ 進む．
4) 右枝がのびる方向を向く (a 度 右を向く)．
5) 現時点を根として，幹の長さを $\frac{d}{2}$ の右枝 (の樹木) を再帰的に描く (n は 1 回分減じる)．
   ただし，$n = 1$ なら真上 (現在向いている方向) へ $\frac{d}{2}$ 進む．
6) 真上を向いてから，根の位置まで戻る (降りる)．

タートルの動かし方のポイントは，手続き (tree n d) 呼び終わったときに，呼び始めたときと同じ位置と方向に戻っていることである．これら一連の操作に対応する手続き (tree n d a) はつぎのように定義される．

```
(define (tree-start n d)
   (begin (cls)  (jump 320 350)
   (tree n d 90)))

(define (tree n d a)
   (cond ((< n 1) (forward d)(back d)) ; 枝を描き，後戻り
         (else (forward (/ d 2))       ; 幹を描く
```

```
(left (/ a 2))          ; 左枝の成長方向を向く
(tree (- n 1) (/ d 2) a); 左枝を再帰的に描く
(right a)               ; 右枝の成長方向を向く
(tree (- n 1) (/ d 2) a); 右枝を再帰的に描く
(left (/ a 2))          ; 真上（北）を向く
(back (/ d 2)))))       ; 幹の長さ分降りて根に戻る.
```

［問 1.7］
手続き `tree` の実引数にさまざまな数値をあてはめて，樹木曲線を描け.

［問 1.8］
手続き `tree` では，枝分かれが常に 90 度である．樹木曲線をより自然の木の形状に近づけるためには，枝分かれする際の角度を固定せずに，図 1.6(d) のように，a_1, a_2, a_3 と角度を変化させた方がよい．そこで，乱数を使って，枝分かれの角度を 30　90 度の範囲にした手続き `tree-rand` を定義せよ.
ヒント：`(tree` n d a`)` における a を，30　90 の整数とすればよい．そのために，`(random 60)` の評価値を利用する.