アルゴリズムとデータ構造入門
2.2.4 図形言語 (Picture Language)

奥 乃 博
大学院情報学研究科
知能情報学専攻音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/12/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

平山 直樹 if mod(学籍番号の下3桁, 3) ≡0
黄 楊暘 if mod(学籍番号の下3桁, 3) ≡1
山口 雄紀 if mod(学籍番号の下3桁, 3) ≡2

NoStudent Left Behind

12月4日・本日のメニュー

NoStudent Left Behind

2.2.4 Picture Language(図形言語)
1. Square-limit
2. Square limit variation, 和田の解説
3. Space Padding Functions
4. Fractal (Self-Similarity)
   1. Hilbert curve
   2. Koch snowflake
   3. Sierpinski's Gasket
   4. Peano curve
5. Circle limit

今日は必修課題の説明です.

2

補足: tustk2 の読込み時エラー

NoStudent Left Behind

- JAKLD では関係ありません.
- エラーメッセージ

```
> (load 'zorro.scm)
Loading zorro.scm...
Error:  Compile error.
| is not a pair.
To print debugger commands, type :H.
Debug[1]>
```

- cygwin上のTUS: 改行が nl (¥n) でないといけない
  - od で改行が "cr nl" か"nl"だけかをチェック.

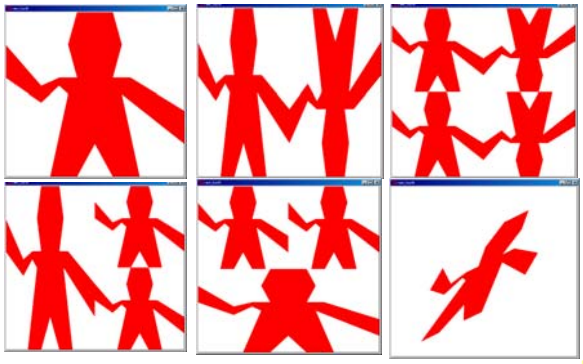
    od -a ファイル | more
  - "cr nl" ならば, "nl" に変換

    tr '¥r' ' '< ファイル > 新しいファイル名

    空白

4

図形言語 Picture language とは



5

図形言語 Picture languageとは

図形言語での学習目標

1．図形操作の抽象化の修得

2．実際に図形言語で，フラクタル，空間充填曲線が作成できる技術の修得

6

図形言語の使い(JAKLKD-README[jp.txt)

```
% java –Xss1m –jar d:/java/scheme/jakld.tar        あるいは
% rlwrap java –Xss1m –jar d:/java/scheme/jakld.tar  (コマンド行編集可)
> (load "init.lsp")
    ここで，(start-picture) と show の定義，frm1, frm2, frm3 等の設定.
> (define orida (image->painter "orida-sensei.gif"))
> (orida frm1)      ; 標準フレーム用キャンバス frm1 で描画
> (clear-picture)   ; 描いた絵を白紙に戻す
> (orida frm2)      ; 歪んだキャンバス frm2 で描画
> (clear-picture)
> (load "wave.lsp")  ; wave の定義を読み込む
> (wave frm1)        ; 人形を描画
> ((square-limit wave 4) frm1)   ; square-limit を描画
> (save-picture "wave.jpg") ; 絵をjpegファイルへ出力
```

7

init.lsp for JAKLD

NoStudent Left Behind

```
;;; sample init file for JAKLD Picture Language
(start-picture)
;; Standard frame
(define frm1 (make-frame (make-vect 0.0 0.0)
                         (make-vect 1.0 0.0)
                         (make-vect 0.0 1.0)))
;; Shearing frame
(define frm2 (make-frame (make-vect 0.0 0.0)
                         (make-vect 0.66 0.33)
                         (make-vect 0.33 0.66)))
;; Compress to left
(define frm3 (make-frame (make-vect 0.0 0.0)
                         (make-vect 0.5 0.0)
                         (make-vect 0.0 1.0)))
;; Compress to bottom
(define frm4 (make-frame (make-vect 0.0 0.0)
                         (make-vect 1.0 0.0)
                         (make-vect 0.0 0.5)))
(define (show-painter . frame)
  (clear-picture)
  (painter (if (null? frame) frm1 (car frame))))
```

showは使わない

8

図形言語の使い2（README.jp.txt）

NoStudent Left Behind

> (set-color *<color>*)
> (set-bg-color *<color>*)
<color>::= black | blue | cyan | dark-gray | gray | green |
light-gray | magenta | orange | pink | red | white | yellow |
#xrrggbb
> (clear-picture) ；ウィンドウの白紙化
> (show-picture) ；ウィンドウの再描画
> (hide-picture) ；ウィンドウを隠す
> (save-picture *<file>*) ；描画された絵をファイルに出力
<file>::= *.bmp | *.jpeg | *.jpg | *.png
> (point->painter *<x>* *<y>*) ；ピクセル(x,y)に点を描画
> (procedure->painter *<function>* [*<arg>*]) ；ピクセルを描画

picture.lsp や sample.lsp に多数のヒントあり.

9

図形言語の使い3（README.jp.txt）

NoStudent Left Behind

> (point->painter *<x>* *<y>*) ；ピクセル(x,y)に点を描画
> (procedure->painter *<function>* [*<arg>*]) ；ピクセルを描画
> (vertexes->painter *<vertex-list>* *<fill?>*)
> (segments->painter *<segments-list>*)
> (image->painter *<image-file>*)

> (draw-line *<absolute-start-point>* *<absolute-end-vertex>*)
> (draw-polygon *<absolute-vertex-list>* *<fill?>*)

picture.lsp や sample.lsp に多数のヒントあり.

プログラムングでは人のコードを読むこと.

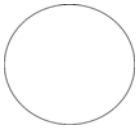
1. 絵を模写，文章を写す，歌を真似る，…
2. Copy&paste ではないことに注意（剽窃 plagiarismは不可）

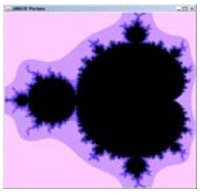
10

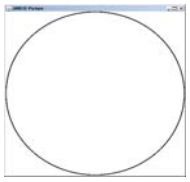
Procedure->painter in JAKLD

NoStudent Left Behind

```
(define (circle z)
  (let ((r (+ (expt (- (car z) 0.5) 2)
              (expt (- (cdr z) 0.5) 2) )))
    (if (and (>= r 0.245) (< r 0.250))
        #x000000      ; black
      #xffffff ) ))   ; white

((procedure->painter circle) frm1)
```







1.1

図形言語の使い(jakld)

NoStudent Left Behind

```
% java -Xss1m -jar jakld.jar
> (load "picture.lsp")
> (load "sample.lsp")
> (start-picture)          ; picture windowが現れない場合実行
> (wave frm1)              ; 標準フレーム用キャンバスの作成
> (wave  frm2)
> (wave  frm3)
> (clear-picture)          ; windowを白紙に戻す
> (show-picture)           ; windowを表示
> (hide-picture)           ; windowを隠す
> (save-picture "filename.png")   ; 絵をpngファイルへ出力
> ((right-split wave 4) frm1)
> (letterlambda frm1)
> ((right-split letterlambda 3))
> ((corner-split letterlambda)  frm2)
```

1.2

図形言語の使い(README.tustk)

NoStudent Left Behind

```
% tustk
> (load '/usr/local/lib/tustk/demos/picl)
> (load '/usr/local/lib/tustk/demos/picl-test)
> (make-canvas frm1)        ; 標準フレーム用キャンバスの作成
> ((square-limit wave 2)  frm1)
> (tk:update)                    ; 上記実行で描画されなかった場合や
                                 ; 再描画が必要なときに実行
> (clear)                        ; 描いた絵を消す
> ((square-limit wave 4) frm1)
> (clear)
> ((squash-inwards wave) frm1)
> (forget-canvas)                ; 現在のキャンバスを削除
> (make-canvas frm2)        ; 傾いたフレーム用キャンバスの作成
> ((square-limit wave 4) frm2)
> (output-canvas 'wave.ps)  ; 絵をPostScriptファイルへ出力
```

1.3

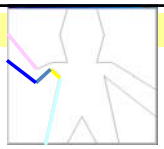
図形言語の使い方(README.tustk)

- *bg-color* ; 背景色の定義
- *line-color* ; 描画線の定義
- (set! *bg-color* <色>) ; 色の設定を変更
- <色> ; 色名(blue, "red", ...), RGB指定 ("#RRGGBB")
- 図形は painter で定義
  - 点 (make-vect <*x-coordinate*> <*y-coordinate*>)
  - 線(セグメント) (make-segment <*from-point*> <*to-point*>)
  - 線画 (segments->painter <*list-of-seguments*>)
  - 多角形(ポリゴン) (vects->painter <*list-of-points*>)
  - (vects->painter <*list-of-points*> <*smooth-or-not*> <*degree-of-smoothing*> <*filling-color*>)
  - (pgm-file->painter <*file-name*>) ; GIF/PPM/PGM

```
(define sicp (pgm-file->painter
              "/usr/local/lib/tustk/demos/sicp.ppm"))
((square-limit sicp 4) frm1)
```

1.4

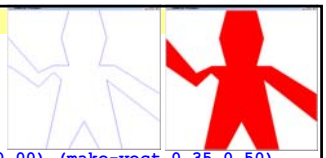
wave の定義



線を次々描いていく.

```
(define wave
  (segments->painter
    (list (make-segment (make-vect 0.25 0.00) (make-vect 0.35 0.50))
          (make-segment (make-vect 0.35 0.50) (make-vect 0.30 0.55))
          (make-segment (make-vect 0.30 0.55) (make-vect 0.20 0.45))
          (make-segment (make-vect 0.20 0.45) (make-vect 0.00 0.60))
          (make-segment (make-vect 0.00 0.80) (make-vect 0.20 0.55))
          (make-segment (make-vect 0.20 0.55) (make-vect 0.30 0.60))
          (make-segment (make-vect 0.30 0.60) (make-vect 0.40 0.60))
          (make-segment (make-vect 0.40 0.60) (make-vect 0.35 0.80))
          (make-segment (make-vect 0.35 0.80) (make-vect 0.40 1.00))
          (make-segment (make-vect 0.60 1.00) (make-vect 0.65 0.80))
          (make-segment (make-vect 0.65 0.80) (make-vect 0.60 0.60))
          (make-segment (make-vect 0.60 0.60) (make-vect 0.70 0.60))
          (make-segment (make-vect 0.70 0.60) (make-vect 1.00 0.40))
          (make-segment (make-vect 1.00 0.20) (make-vect 0.65 0.50))
          (make-segment (make-vect 0.65 0.50) (make-vect 0.75 0.00))
          (make-segment (make-vect 0.60 0.00) (make-vect 0.50 0.20))
          (make-segment (make-vect 0.50 0.20) (make-vect 0.40 0.00))
    )))
```

filled-wave の変形



JAKLD

```
(define filled-wave
  (vertexes->painter
    (list (make-vect 0.25 0.00) (make-vect 0.35 0.50)
          (make-vect 0.30 0.55) (make-vect 0.20 0.45)
          (make-vect 0.00 0.60) (make-vect 0.00 0.80)
          (make-vect 0.20 0.55) (make-vect 0.30 0.60)
          (make-vect 0.40 0.60) (make-vect 0.35 0.80)
          (make-vect 0.40 1.00) (make-vect 0.60 1.00)
          (make-vect 0.65 0.80) (make-vect 0.60 0.60)
          (make-vect 0.70 0.60) (make-vect 1.00 0.40)
          (make-vect 1.00 0.20) (make-vect 0.65 0.50)
          (make-vect 0.75 0.00) (make-vect 0.60 0.00)
          (make-vect 0.50 0.20) (make-vect 0.40 0.00) )
    #t ))
```

点をつないで, ポリゴンを作成し, 色を塗る.

1.6

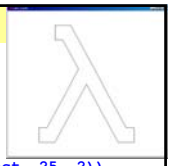
filled-wave の変形

Tustk

```
(define red-wave
  (vects->painter
    (list (make-vect 0.25 0.00) (make-vect 0.35 0.50)
          (make-vect 0.30 0.55) (make-vect 0.20 0.45)
          (make-vect 0.00 0.60) (make-vect 0.00 0.80)
          (make-vect 0.20 0.55) (make-vect 0.30 0.60)
          (make-vect 0.40 0.60) (make-vect 0.35 0.80)
          (make-vect 0.40 1.00) (make-vect 0.60 1.00)
          (make-vect 0.65 0.80) (make-vect 0.60 0.60)
          (make-vect 0.70 0.60) (make-vect 1.00 0.40)
          (make-vect 1.00 0.20) (make-vect 0.65 0.50)
          (make-vect 0.75 0.00) (make-vect 0.60 0.00)
          (make-vect 0.50 0.20) (make-vect 0.40 0.00) )   #f 0
  'red ))
```


点をつないで，ポリゴンを作成し，色を塗る.

letterlambda の定義



```
(define letterlambda
  (segments->painter
   (list
     (make-segment (make-vect .45 .6) (make-vect .25 .2))
     (make-segment (make-vect .25 .2) (make-vect .2 .2))
     (make-segment (make-vect .2 .2) (make-vect .2 .1))
     (make-segment (make-vect .2 .1) (make-vect .3 .1))
     (make-segment (make-vect .3 .1) (make-vect .5 .5))
     (make-segment (make-vect .5 .5) (make-vect .7 .1))
     (make-segment (make-vect .7 .1) (make-vect .8 .1))
     (make-segment (make-vect .8 .1) (make-vect .8 .2))
     (make-segment (make-vect .8 .2) (make-vect .75 .2))
     (make-segment (make-vect .75 .2) (make-vect .4 .9))
     (make-segment (make-vect .4 .9) (make-vect .3 .9))
     (make-segment (make-vect .3 .9) (make-vect .3 .8))
     (make-segment (make-vect .3 .8) (make-vect .35 .8))
     (make-segment (make-vect .35 .8) (make-vect .45 .6))
   )))
```


filled-letterlambda の定義

JAKLD

```
(define filled-letterlambda
  (vertexes->painter
    (list
      (make-vect .45 .60) (make-vect .25 .20)
      (make-vect .25 .20) (make-vect .20 .20)
      (make-vect .20 .20) (make-vect .20 .10)
      (make-vect .20 .10) (make-vect .30 .10)
      (make-vect .30 .10) (make-vect .50 .50)
      (make-vect .50 .50) (make-vect .70 .10)
      (make-vect .70 .10) (make-vect .80 .10)
      (make-vect .80 .10) (make-vect .80 .20)
      (make-vect .80 .20) (make-vect .75 .20)
      (make-vect .75 .20) (make-vect .40 .90)
      (make-vect .40 .90) (make-vect .30 .90)
      (make-vect .30 .90) (make-vect .30 .80)
      (make-vect .30 .80) (make-vect .35 .80)
      (make-vect .35 .80) (make-vect .45 .60) )
    #t ))
```

filled-letterlambda の定義

Tustk

```
(define red-letterlambda
  (vects->painter
    (list
      (make-vect .45 .60) (make-vect .25 .20)
      (make-vect .25 .20) (make-vect .20 .20)
      (make-vect .20 .20) (make-vect .20 .10)
      (make-vect .20 .10) (make-vect .30 .10)
      (make-vect .30 .10) (make-vect .50 .50)
      (make-vect .50 .50) (make-vect .70 .10)
      (make-vect .70 .10) (make-vect .80 .10)
      (make-vect .80 .10) (make-vect .80 .20)
      (make-vect .80 .20) (make-vect .75 .20)
      (make-vect .75 .20) (make-vect .40 .90)
      (make-vect .40 .90) (make-vect .30 .90)
      (make-vect .30 .90) (make-vect .30 .80)
      (make-vect .30 .80) (make-vect .35 .80)
      (make-vect .35 .80) (make-vect .45 .60) )
    #f 0 'red ))
```

NoStudent Left Behind

flipped-pair(拡張版)

NoStudent Left Behind

```
(define wave2
    (beside wave (flip-vert wave)) )
(define wave4 (below wave2 wave2))
```

wave2

```
(define (flipped-pairs painter)
    (let ((painter2
            (beside painter
              (flip-vert painter) )))
      (below painter2 painter2)))
```

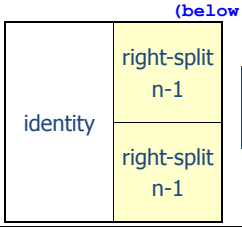
こうすると

```
(define wave4
      (flipped-pairs wave))
```

wave4

right-split n

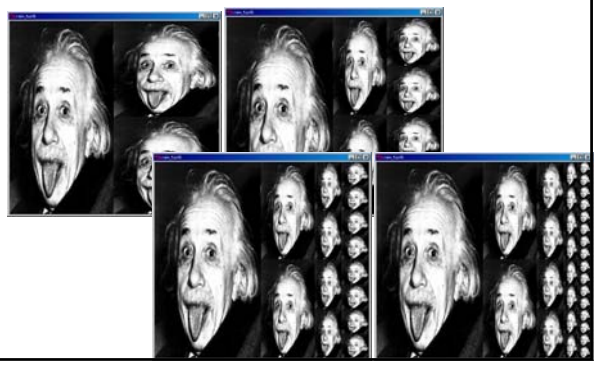
```
(define (right-split painter n)
   (if (= n 0)
       painter
       (let ((smaller
               (right-split painter (- n 1))) )
         (beside painter
                 (below smaller smaller) ))))
```



未定義の手続き
- (below bottom top)
- (beside left right)

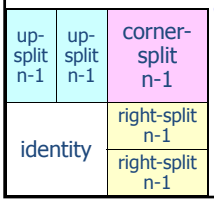
NoStudent Left Behind

right-split n　の動き



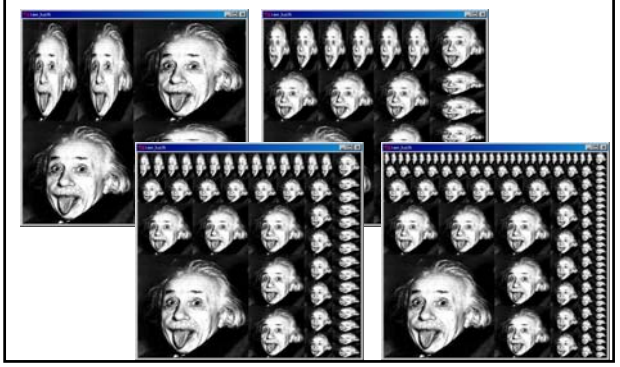
corner-split n

```
(define (corner-split painter n)
  (if (= n 0)
      painter
      (let ((up (up-split painter (- n 1)))
            (right (right-split painter (- n 1))) )
        (let ((top-left (beside up up))
              (bottom-right (below right right))
              (corner (corner-split painter (- n 1))) )
          (beside (below painter top-left)
                  (below bottom-right corner) )))))
```



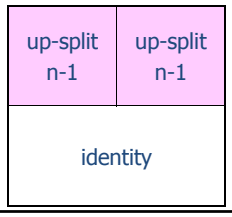
未定義の手続き
- (below bottom top)
- (beside left right)

corner-split n　の動き



Ex.2.44 up-split

```
(define (up-split painter n)
   (if (= n 0)
       painter
       (let ((smaller
              (up-split painter (- n 1))) )
         (below painter
                (beside smaller smaller) ))))
```



- 未定義の手続き
- (below bottom top)
- (beside left right)

NoStudent Left Behind

square-limit n

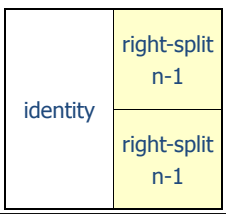
NoStudent Left Behind

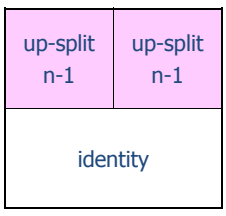
```
(define (square-limit painter n)
   (let ((quarter
          (corner-split painter n)) )
      (let ((half (beside
                   (flip-horiz quarter)
                   quarter )))
        (below (flip-vert half)
               half))))
```

split群を抽象化

NoStudent Left Behind

```
(define right-split (split beside below))
(define up-split (split below beside))
(define (split op1 op2)
   (op1 half (op2 quarter quarter)) ))
```





square-limit n の動き

38

Square Limit をより抽象化

```
(define (square-of-four tl tr bl br)
   (lambda (painter)
      (let ((top (beside (tl painter)
                         (tr painter)) )
            (bottom (beside (bl painter)
                            (br painter) )))
         (below bottom top) )))
```

tl	tr
bl	br

未定義の手続き
- (below bottom top)
- (beside left right)

40

flipped-pairsの別の定義

```
(define (flipped-pairs painter)
   (let ((combine4
          (square-of-four
               identity flip-vert
               identity flip-vert )))
      (combine4 painter) ))
```

identity	flip-vert
identity	flip-vert

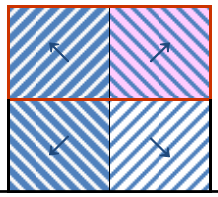
未定義の手続き
- (below bottom top)
- (beside left right)

41

square-limit の別の定義

NoStudent Left Behind

```
(define (square-limit painter n)
  (let ((combine4
         (square-of-four
             flip-horiz identity
             rotate180 flip-vert )))
    (combine4
       (corner-split painter n)) ))
```



flip-horiz	identity
rotate180	flip-vert

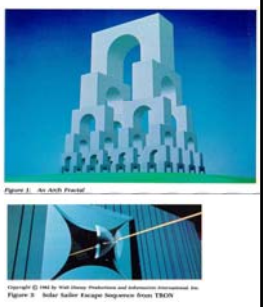
42

TRON のベース ASAS on Lisp

NoStudent Left Behind

Craig W. Reynolds (III): Computer animation with scripts and actors, *Computer Graphics*, Vo.16, No.3, pp.289-296.

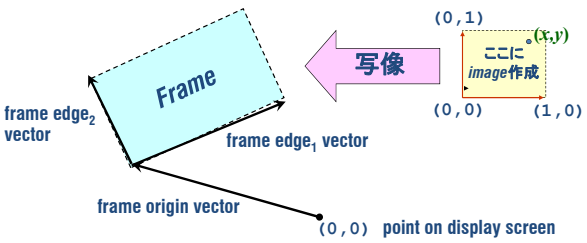
```
(defop arch-fractalizer
   (param: arch-element top-color bot-color levels
           fractal-ratio height width leg-width)
   (local: (total-levels  levels)
           (offset-dist  (half (dif width leg-width)))
           (sub-tower-offset-1 (vector offset-dist 0 0))
           (sub-tower-offset-2 (mirror x-axis
                                 sub-tower-offset-1)))
   (arch-tower levels))

(defop arch-tower
   (param: levels)
   (if (zerop levels)
       (then nothing)
       (else (add-arch-level (arch-tower
                               (dif levels 1))))))
(defop add-arch-level
    (param: sub-tower)
    (grasp sub-tower
           (scale factal-ratio)
           (move (vector 0 height 0))
           (rotate 0.25 y-axis))
    (grasp arch-element
           (recolor (interp (quo levels total-levels)
                            bot-color top-color)))
    (subworld (group arch-element
              (move subtower-offset-1 sub-tower)
              (move subtower-offset-2 sub-tower)))))
```



43

Frame coordinate map

NoStudent Left Behind



1. coordinate: unit squareに作成
2. frameに写像

$$\overrightarrow{\text{Origin}}(\text{Frame}) + x \cdot \overrightarrow{\text{Edge}}_1(\text{Frame}) + y \cdot \overrightarrow{\text{Edge}}_2(\text{Frame})$$

48

Frames

```
(define (frame-coord-map frame)
  (lambda (v)
    (add-vect
      (origin-frame frame)
      (add-vect (scale-vect (xcor-vect v)
                    (edge1-frame frame) )
                (scale-vect (ycor-vect v)
                    (edge2-frame frame) )
      ))))

    ((frame-coord-map a-frame)
     (make-vect 0 0) )
```

の返す値: `(origin-frame a-frame)`

49

Frames

```
(define (make-frame origin edge1 edge2)
  (list origin edge1 edge2) )

(define (make-frame origin edge1 edge2)
  (cons origin (cons edge1 edge2)) )
```

それぞれに対する選択子を書け.

50

Painters

```
(define (segments->painter segment-list)
  (lambda (frame)
    (for-each
     (lambda (segment)
       (draw-line
        ((frame-coord-map frame)
         (start-segment segment) )
        ((frame-coord-map frame)
         (end-segment segment) )))
     segment-list )))
```

`(foreach` *<procedure>* *<list-of-items>*`)`

51

Transforming and combining painters

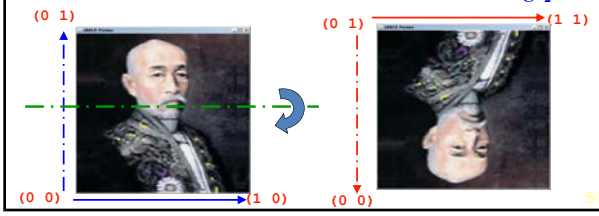
```
(define (transform-painter painter
                origin corner1 corner2)
  (lambda (frame)
    (let ((m (frame-coord-map frame)))
      (let ((new-origin (m origin)))
        (painter
         (make-frame new-origin
            (sub-vect (m corner1)
                      new-origin)
            (sub-vect (m corner2)
                      new-origin)))))))
```

NoStudent Left Behind

Transforming and combining painters

NoStudent Left Behind

```
(define (flip-vert painter)
  (transform-painter
     painter
      (make-vect 0.0 1.0)  ; new origin
      (make-vect 1.0 1.0)  ; new end of edge1
      (make-vect 0.0 0.0)))  ; new end of edge2
```



53

Transforming and combining painters

NoStudent Left Behind

```
(define (shrink-to-upper-right painter)
  (transform-painter
     painter
     (make-vect 0.5 0.5)
     (make-vect 1.0 0.5)
     (make-vect 0.5 1.0) ))
```

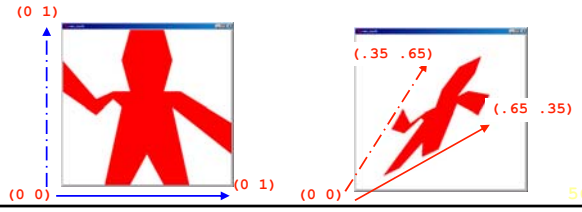
54

Transforming and combining painters

```
(define (rotate90 painter)
  (transform-painter
    painter
    (make-vect 1.0 0.0)
    (make-vect 1.0 1.0)
    (make-vect 0.0 0.0)
  ))
```

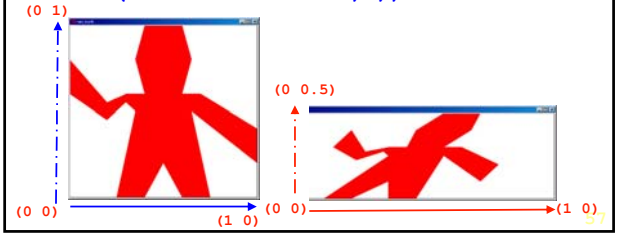
Transforming and combining painters

```
(define (squash-inwards painter)
  (transform-painter
      painter
      (make-vect 0.0 0.0)
      (make-vect 0.65 0.35)
      (make-vect 0.35 0.65)))
```



傾いたフレーム

```
(define frm2
    (make-frame
      (make-vect 0 0)
      (make-vect 1.0 0)
      (make-vect 0.5 0.5) ))
```



transform-painter（まとめ）

```
(define (half-vert painter)
  (transform-painter
     painter
     (make-vect 0.0 0.0)
     (make-vect 1.0 0.0)
     (make-vect 0.0 0.5)) )
```

```
(define (flip-vert painter)
  (transform-painter
     painter
     (make-vect 0.0 1.0
     (make-vect 1.0 1.0)
     (make-vect 0.0 0.0)) )
```

```
(define (half-horiz painter)
  (transform-painter
     painter
     (make-vect 0.0 0.0)
     (make-vect 0.5 0.0)
     (make-vect 0.0 1.0)) )
```

```
(define (shrink-to-upper-right painter)
    (transform-painter
       painter
       (make-vect 0.5 0.5)
       (make-vect 1.0 0.5)
       (make-vect 0.5 1.0)) )
```

```
(define (rotate90 painter)
  (transform-painter painter (make-vect 1.0 0.0)
                             (make-vect 1.0 1.0)
                             (make-vect 0.0 0.0) ))
```

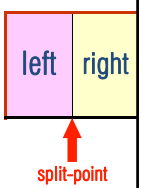
58

Gifに出力，animated GIF作成

59

beside

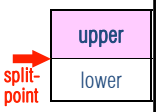
```
(define (beside painter1 painter2)
  (let ((split-point (make-vect 0.5 0.0)))
    (let ((paint-left
           (transform-painter
                painter1
                (make-vect 0.0 0.0)
                split-point
                (make-vect 0.0 1.0) ))
          (paint-right
           (transform-painter
                painter2
                split-point
                (make-vect 1.0 0.0)
                (make-vect 0.5 1.0) )))
      (lambda (frame)
        (paint-left frame)
        (paint-right frame) ))))
```



60

Ex.2.51 below

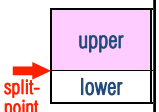
```
(define (below painter1 painter2)
  (let ((split-point (make-vect 0.0 0.5)))
    (let ((paint-lower
           (transform-painter
                painter1
                                          ))
          (paint-upper
           (transform-painter
                painter2
                                          )))
      (lambda (frame)
        (paint-lower frame)
        (paint-upper frame) ))))
```



61

above m:n で分割

```
(define (above painter1 painter2 . a)
  (let* ((m (if (null? a) 1 (car a)))
         (n (if (or (null? a) (null? (cdr a)))
                1 (cadr a)))
         (r (/ n (+ m n)))
         (split-point (make-vect 0.0 r)) )
    (let ((paint-lower
           (transform-painter painter2
                              (make-vect 0.0 0.0)
                              (make-vect 1.0 0.0)
                              split-point) )
          (paint-upper
           (transform-painter painter1
                              split-point
                              (make-vect 1.0 r)
                              (make-vect 0.0 1.0) )))
      (lambda (frame)
        (paint-lower frame)
        (paint-upper frame) ))))
```



63

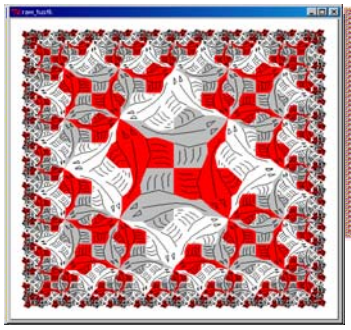
Stratified Design (成層設計)

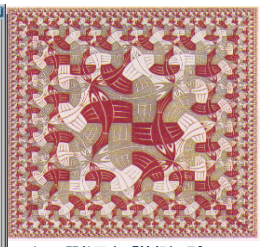
図形言語のような設計法

- 各レベルでの部品化
  - Square-limit
  - below, beside
  - transform-painter
  - draw-line

68

Escher's square-limit



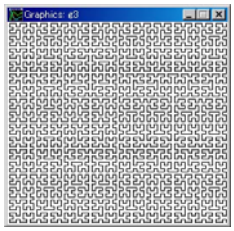


■和田：関数画家、『情報処理』、Vol.46, No.10 (2005) 1163-1171

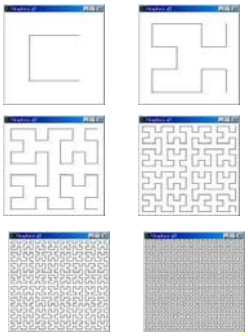
■http://www.ecs.soton.ac.uk/~ph/papers/funcgeo2.pdf

70

Hilbert curve の作成方法



(hilbert 5)



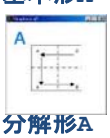
73

Hilbert curve の作成方法

4つの基本形

1. 基本形A： D⇒A⇒A⇒B
2. 基本形B： C⇒B⇒B⇒A
3. 基本形C： B⇒C⇒C⇒D
4. 基本形D： A⇒D⇒D⇒C

基本形A

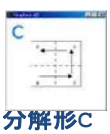


分解形A

基本形B

分解形B

基本形C



分解形C

基本形D

分解形D

75

Hilbert curve の手続き

NoStudent Left Behind

1. 各基本形に対して，レベル0ならコ型を書くための頂点のリストを求める．
2. さもなければ，分解形を再帰的に呼び出し，頂点を求める．
3. 求まった頂点リストから segment を求め

   `vectors->segment` と

   `segments->painter`を使って作成する．

```
(vectors->segment <list of vectors>)
(segments->painter <list of segments>)
```

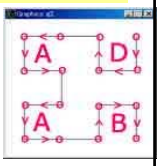
`vertexes->painter` を使用してはいけない．

76

Hilbert curve の手続き

NoStudent Left Behind

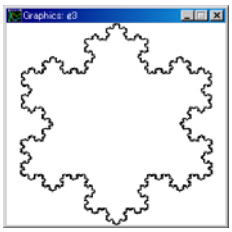
```
(define (hilbert-a p0 q0 p1 q1 i)
  (let ((xs (/ (+ (* 3.0 p0) p1) 4.0))
        (ys (/ (+ (* 3.0 q0) q1) 4.0))
        (xm (/ (+ p0 p1) 2.0))
        (ym (/ (+ q0 q1) 2.0))
        (xl (/ (+ p0 (* 3.0 p1)) 4.0))
        (yl (/ (+ q0 (* 3.0 q1)) 4.0)) )
    (if (= i 0)
        (list (make-vect p1 q1) (make-vect xs ql)
              (make-vect xs ys) (make-vect xl ys) )
        (append (hilbert-d xm ym p1 q1 (- i 1))
                (hilbert-a p0 ym xm q1 (- i 1))
                (hilbert-a p0 q0 xm ym (- i 1))
                (hilbert-b xm q0 p1 ym (- i 1)) ))))

(define (hilbert n)
  (segments->painter
    (vectors->segments (hilbert-a 0.0 0.0 1.0 1.0 n))) )
```

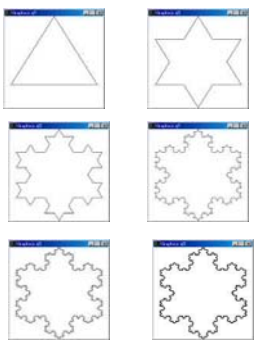


77

Koch snowflake の作成方法

NoStudent Left Behind





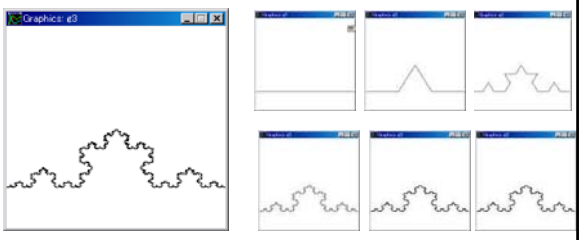
```
(koch 5)
```

78

Koch snowflake の作成方法

NoStudent Left Behind

線分の分解



80

Koch snowflake の手続き

NoStudent Left Behind

1. 各線分に対して，レベル0なら，三角形の頂点リストを求める．
2. さもなければ，分解形を再帰的に呼び出し，頂点を求める．
3. 【JAKLD】求まった頂点リストから 折れ線を vertexes->painter を使用して作成．【Tustk】求まった頂点リストからsegment を求め vectors->segment と segments->painterを使って作成する．

```
(vectors->segment <list of vectors>)
(segments->painter <list of segments>)
```

81

Koch snowflake（続）

NoStudent Left Behind

```
(define (koch-line p0 q0 p1 q1 r i)
  (if (= i 0)
      (list (make-vect p0 q0) (make-vect p1 q1))
      (let* ((r1 (/ r 3.0))
             (x3 (/ (- p1 p0) 3.0))
             (y3 (/ (- q1 q0) 3.0))
             (xs (/ (+ (* 2.0 p0) p1) 3.0))
             (ys (/ (+ (* 2.0 q0) q1) 3.0))
             (xl (/ (+ p0 (* 2.0 p1)) 3.0))
             (yl (/ (+ q0 (* 2.0 q1)) 3.0))
             (xm (+ (* 0.5 x3) (* 0.866 y3) xs))
             (ym (+ (* 0.5 y3) (* -0.866 x3) ys)) )
        (append (koch-line p0 q0 xs ys r1 (- i 1))
                (koch-line xs ys xm ym r1 (- i 1))
                (koch-line xm ym xl yl r1 (- i 1))
                (koch-line xl yl p1 q1 r1 (- i 1))
        ))))
```

82

Koch snowflake の手続き(続)

NoStudent Left Behind

```
(define (koch n)
  (let* ((h (/ 0.75 0.86))
         (p0 (/ (- 1.0 h) 2))
         (p1 (- 1.0 p0)) )
    (segments->painter
      (vectors->segments
        (append
          (koch-line p0 0.25 p1 0.25 1 n)
          (koch-line p1 0.25 0.5 1.0 1 n)
          (koch-line 0.5 1.0 p0 0.25 1 n)
        )))))
```

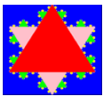
let* は let と違い、変数値対を順番に評価

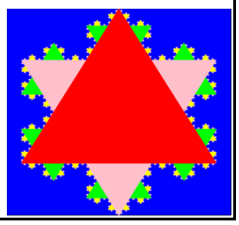
83

色つきKoch curve の手続き

NoStudent Left Behind

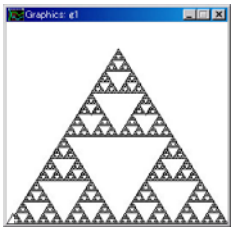
```
(define (koch-fill n . args)
  (let* ((h (/ 0.75 0.86))
         (p0 (/ (- 1.0 h) 2))
         (p1 (- 1.0 p0))
         (color (if (null? args) 'red (car args))) )
    (vects->painter
      (append (koch-line p0 0.25 p1 0.25 1 n)
        (koch-line p1 0.25 0.5 1.0 1 n)
        (koch-line 0.5 1.0 p0 0.25 1 n))
      #f 0 color )))

(define (fun-koch x)
   ((koch-fill 5 'pink) x)
   ((koch-fill 4 'while) x)
   ((koch-fill 3 'yellow) x)
   ((koch-fill 2 'green) x)
   ((koch-fill 1 'pink) x)
   ((koch-fill 0 'red) x)
  )
```



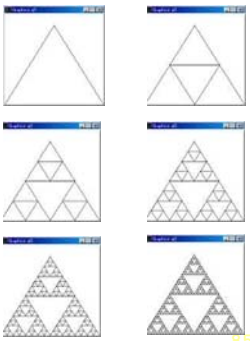


Sierpinski's Gasket の作成法

NoStudent Left Behind



(sierpiski 6)



85

Sierpinski's Gasket の手続き

NoStudent Left Behind

```
(define (gasket p0 q0 p1 q1 i)
  (let* ((xm (/ (+ p0 p1) 2.0))
         (ym (+ (* (- p1 p0) 0.866) q0))
         (xs (/ (+ (* 3.0 p0) p1) 4.0))
         (xl (/ (+ (* 3.0 p1) p0) 4.0))
         (ys (+ (* (- p1 p0) 0.433) q0)) )
    (if (= i 0)
        (list (make-vect p0 q0)  (make-vect p1 q1)
              (make-vect (/ (+ p0 p1) 2.0) ym)
              (make-vect p0 q0) )
        (append (gasket p0 q0 xm q0 (- i 1))
                (gasket xm q0 p1 q0 (- i 1))
                (list (make-vect p0 q0))
                (gasket xs ys xl ys (- i 1))
                (list (make-vect p0 q0) )))))

(define (sierpenski n)
  (segments->painter
    (vectors->segments (gasket 0.0 0.0 1.0 0.0 n)) ))
```

86

色つきSierpinski's Gasket

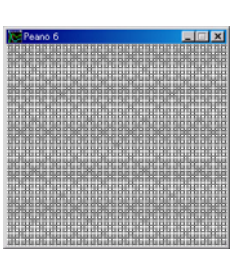
NoStudent Left Behind

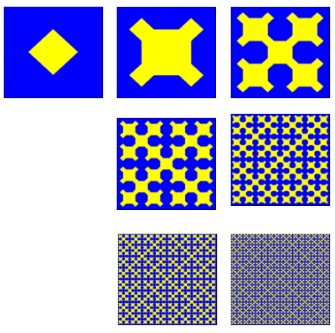
```
(define (gasket-fill p0 q0 p1 q1 i color)
  (let* ((xm (/ (+ p0 p1) 2.0))
         (ym (+ (* (- p1 p0) 0.866) q0))
         (xs (/ (+ (* 3.0 p0) p1) 4.0))
         (xl (/ (+ (* 3.0 p1) p0) 4.0))
         (ys (+ (* (- p1 p0) 0.433) q0)) )
    (if (= i 0)
        (list (vects->painter
                (list (make-vect p0 q0)
                      (make-vect p1 q1)
                      (make-vect xm ym) )
                #f 0 color))
        (append (gasket-fill p0 q0 xm q0 (- i 1) color)
                (gasket-fill xm q0 p1 q0 (- i 1) color)
                (gasket-fill xs ys xl ys (- i 1) color) ))))

(define (sierpenski-fill n . args)
  (let ((color (if (null? args) 'red (car args))))
    (do ((i (gasket-fill 0.0 0.0 1.0 0.0 n color) (cdr i)))
        ((null? i))
      (i frm1) )))
```

87

Peano Curve の作成方法

NoStudent Left Behind





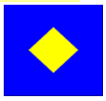
```
(peano 6)
```

88

Peano Curve の手続き

NoStudent Left Behind

```
(define (peano-a p0 q0 p1 q1 i)
  (append (peano-a-1 p0 q0 p1 q1 i)
    (peano-a-2 p0 q0 p1 q1 i) ))

(define (peano-a-1 p0 q0 p1 q1 i)
  (let ((xs (/ (+ (* 3.0 p0) p1) 4.0)) (ys (/ (+ (* 3.0 q0) q1) 4.0))
        (xm (/ (+ p0 p1) 2.0)) (ym (/ (+ q0 q1) 2.0))
        (xl (/ (+ p0 (* 3.0 p1)) 4.0)) (yl (/ (+ q0 (* 3.0 q1)) 4.0)) )
    (if (= i 0)
        (list (make-vect xm yl) (make-vect xs ym))
        (append (peano-a-1 xm ym p1 q1 (- i 1))
                (peano-d-1 p0 ym xm q1 (- i 1))
                (peano-d-2 p0 ym xm q1 (- i 1))
                (peano-a-1 p0 q0 xm ym (- i 1)) ))))

(define (peano-a-2 p0 q0 p1 q1 i)
  (let ((xs (/ (+ (* 3.0 p0) p1) 4.0)) (ys (/ (+ (* 3.0 q0) q1) 4.0))
        (xm (/ (+ p0 p1) 2.0)) (ym (/ (+ q0 q1) 2.0))
        (xl (/ (+ p0 (* 3.0 p1)) 4.0)) (yl (/ (+ q0 (* 3.0 q1)) 4.0)) )
    (if (= i 0)
        (list (make-vect xm ys) (make-vect xl ym))
        (append (peano-a-2 p0 q0 xm ym (- i 1))
                (peano-b-1 xm q0 p1 ym (- i 1))
                (peano-b-2 xm q0 p1 ym (- i 1))
                (peano-a-2 xm ym p1 q1 (- i 1)) ))))
```



89

図形言語に複素数を導入

NoStudent Left Behind

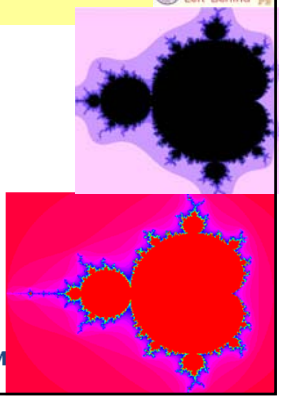
$$z_{n+1} = z_n^{\,2} + C$$
$$z_0 = C$$

が収束する点$C = (x, y)$

Mandelbrot Set

`procedure->painer` 使用

http://mathworld.wolfram.com/MandelbrotSet.html



宿題：12月25日17時締切

NoStudent Left Behind

DON'T PANIC!

1. square-limit を完成させる.
2. letterlambda色付を作成.
3. square-limit に適用.
4. 【随意】square-limit-n (m:nに分割)

Program（そのまま動くもの）と出力の絵を

sicp-10@zeus.kuis.kyoto-u.ac.jp

- TA3名と奥乃がすべてチェックします.

動かない，不完全な場合には，**できるまで再提出**していただきます.

必修課題2：2月13日正午締切

NoStudent Left Behind

1. painterを1種類作成.
2. それをsquare-limit等に適用.
3. 空間充填曲線を1種類作成.
   (Hilbert curve, Peano curve, …)
4. フラクタルを一種類作成.
   (Koch Snowflake, Sierpinsky's Gasket, …)

- レポートはメイル，あるいは，紙で提出.
- プログラムはメイルで sicp-15@zeus.kuis.kyoto-u.ac.jp
- 例は：http://winnie.kuis.kyoto-u.ac.jp/ から「図形言語作品集」
- 教えてもらった場合には，明記すること.
- 1は人と同じプログラム・作品（年度不問）は再提出.

JAKLDの音（MIDI)インタフェース

NoStudent Left Behind

```
% java –Xss1m –jar d:/java/scheme/jakld-sound.tar        あるいは
% rlwrap java –Xss1m –jar d:/java/scheme/jakld-sound.tar （編集機能）
> (load "summer.lsp")
> (load "theEntertainer.lsp")
```

マニュアルは，http://winnie.kuis.kyoto-u.ac.jp/
～okuno/Lecture/09/IntroAlgDs/index.html

- 随意課題S-1：図形言語のアナロジーとして，楽曲生成システムを作成.
- 随意課題S-2：無限音階（Sheperd Tone）を作成
  例1
  例2

93

Scot Joplin The Entertainer by 大塚君

NoStudent Left Behind

```
; ============================
; initialization
; ============================
(define synth (get-synthesizer))
(define seqr (get-sequencer))
(set-receiver seqr synth)

(define seq (sequence 384))
(define trk0 (create-track seq)) ; track 0 (high part)
(define trk1 (create-track seq)) ; track 1 (right hand)
(define trk2 (create-track seq)) ; track 2 (left hand (bass))

; ============================
; write the score here
; ============================
(set-mml trk0
 "C00116k60t75"
;;Intro
 "o6 dec<a^16bg8"
 "r2r2r4"
 "o5 q110d0g8 r8^8^16 r8"
;;Melody
 "t70o6 r2 r4^16 cdd+ecde^16<b>d8c4^8"
)

(set-mml trk1
 "C10116k60t75"
;;Intro
 "o5 dec<a^16bg8 >dec<a^16bg8 dec<a^16baa-g8r8"
 ">q110g0d8 r8^8^16"
;;Melody
 "t70o4 dd+e>c8<e>c8<e>c^4^16"
 "o5 c0e d0f d+0f+ e0g c0e d0f e0g^16 <b0>d d0f8 c0e4^8"
 )

(set-mml trk2
 "C20116k60t75"
;;Intro
 "o4 r2 dec<a^16bg8 dec<a^16baa-g8r8"
 "q110 g0<g8 r8^8^16"
;;Melody
 "t70o318 g0b <c >e0g0>c <<g0>g g0b-0>c <<f0>f a0>c"
 "<<e0>e g0>c <<g >e0g0>c <<g >f0g0b c e0g0>c <<c0>e0g0c8.."
)
```

95

図形言語の作品

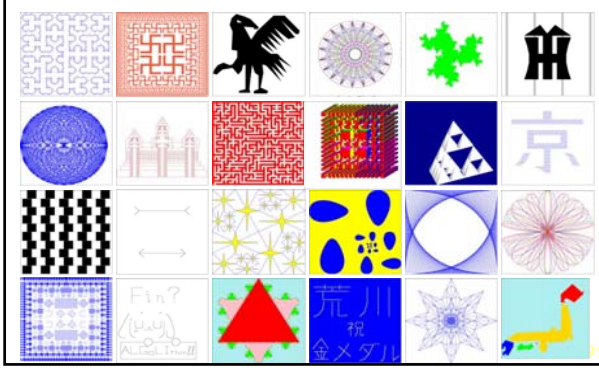
NoStudent Left Behind

- Gallery に掲載
- http://winnie.kuis.kyoto-u.ac.jp/
  から「図形言語作品集」をクリック
- 面白いものは講義等で使わせてもらいます

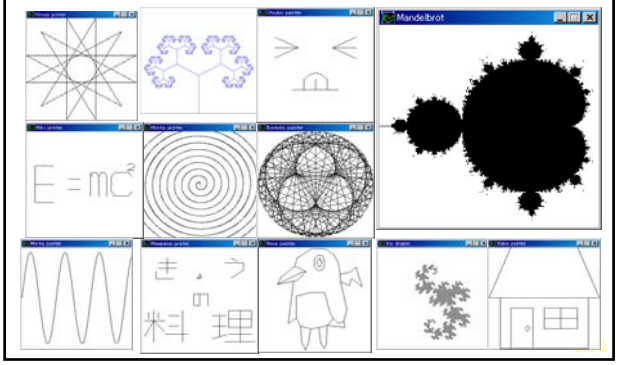
96

気の利いたって？（H18受講生の作品集）

気の利いたって？（H17受講生の作品集）



気の利いたって？(H16受講生の作品集)



随意課題3: 3月15日午後5時締切

NoStudent Left Behind

circle-limit を作成せよ。

プログラムはメイルで

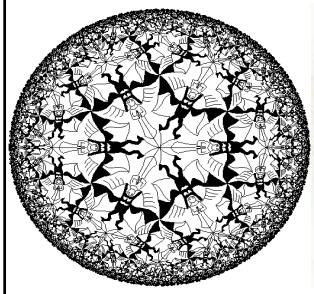
okuno@i.kyoto-u.ac.jp

Escher's Square-Limit(1)

NoStudent Left Behind

H.G. Okuno **M.C. Escher**

EscherのCircle-limit IV (1960)

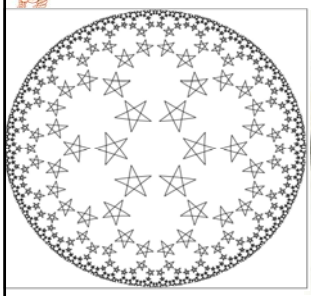


Y.Y. Huang
（2008年度入学）

M.C. Escher

NoStudent Left Behind

EscherのCircle-limit IV (1960)

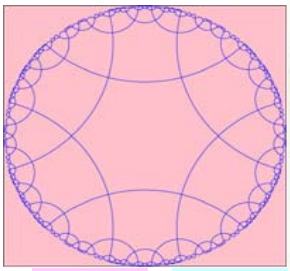


前田一貴
（2005年度入学，数理　学振（DC1））

M.C. Escher

NoStudent Left Behind

双曲三角形 $(\frac{\pi}{3},\frac{\pi}{6},\frac{\pi}{6})$

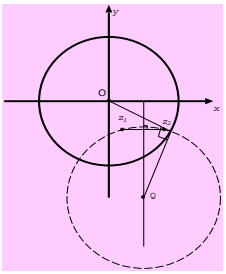


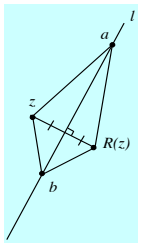
$(\frac{\pi}{3},\frac{\pi}{4},\frac{\pi}{4})$　$(\frac{\pi}{3},\frac{\pi}{6},\frac{\pi}{6})$

120

前田君のCircle-limit:双曲幾何・鏡映変換

■ポアンカレ円盤上での鏡映　直線での鏡映





122

前田君の双曲幾何システム

reflection 2point-circle

双曲幾何パッケージ Hyperbolic Geometry

add sub mul div

汎用算術演算パッケージ

Genetic arithmetic

add-rat sub-rat mul-rat div-rat	add-complex sub-complex mul-complex div-complex	+ - * /
有理数算術演算 Rational arithmetic	複素数算術演算 Complex arithmetic	通常算術演算 Ordinary arithmetic
	直交座標表現 Rectangular / 極座標表現 Polar	

リスト構造と基本マシン算術演算

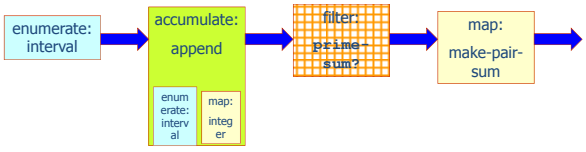
123

写像の入れ子(nesting of mapping)

1≤j<i≤n なる異なる正の整数 i,j に対して、i+jが素数となるものをすべて求める

n=6のとき

i	2	3	4	4	5	6	6
j	1	2	1	3	2	1	5
i+j	3	5	5	7	7	7	11



132

list of pairs of integers の作り方

```
(accumulate
   append
   nil
   (map
     (lambda (i)
       (map
         (lambda (j) (list i j))
         (enumerate-interval 1 (- i 1) )))
     (enumerate-interval 1 n) ))
```

この呼び出しパターンを手続きとして定義

```
(define (flatmap proc seq)
  (accumulate append nil (map proc seq)) )
```

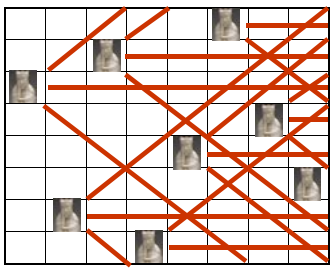
133

list of pairs of integers の作り方

```
(define (prime-sum? pair)
   (prime? (+ (car pair) (cadr pair)) )

(define (make-pair-sum pair)
   (list (car pair) (cadr pair)
         (+ (car pair) (cadr pair)) ))

(define (prime-sum-pairs n)
   (map make-pair-sum
        (filter prime-sum?
          (flatmap
             (lambda (i)
                (map (lambda (j) (list i j))
                     (enumerate-interval
                               1 (- i 1) )))
             (enumerate-interval 1 n) ))))
```

134

n-queens n人の女王の問題



8-queens puzzle

変種：すべての盤面をカバーする最小の女王の数は

- 女王は将棋の飛車角行
- お互いに取り合わないように配置

135

n-queens の作り方:数え上げ

```
(define (permutation s)
   (if (null? s)
       (list nil)
       (flatmap (lambda (x)
                   (map (lambda (p) (cons x p))
                        (permutation (remove x s)) ))
                s )))

(define (remove item sequence)
   (filter (lambda (x) (not (= x item))) sequence) )

(define (safe? k positions)
   (null?
     (filter
       (lambda (x)
          (not (or (= (cadr k) (cadr x))
                   (= (+ (car k) (cadr k))
                      (+ (car x) (cadr x)) )
                   (= (- (car k) (cadr k))
                      (- (car x) (cadr x)) ))))
       positions )))

(define (adjoin-position new k rest-of-q)
   (filter (lambda (x) (not (= x item))) sequence) )
```

136

n-queens の本体

```
(define (queens n)
  (define (queen-cols k)
    (if (= k 0)
        (list empty-board)
        (filter
          (lambda (positions)
            (safe? k positions) )
          (flatmap
            (lambda (rest-of-q)
              (map (lambda (new-row)
                     (adjoin-position new-row
                                      k rest-of-q ))
                   (enumerate-interval 1 n) ))
            (queens-cols (- k 1)) ))))
  (queens-cols board-size) )
```

137